[*3. apríla 2002*]

Programujem s COFAXom

1994 – 1998

zostavil: Michal Winczer

Katedra vyučovania informatiky, Matematicko-fyzikálna fakulta Univerzity Komenského, 1998

Úvod

Súťaž Programujem s COFAXom oslávila v roku 1998 piate výročie. Prvý raz sa konala počas výstavy COFAX v Bratislave v júni 1994, odkedy tvorí pravidelnú súčasť sprievodného programu výstavy. Podnet na usporiadanie stredoškolskej súťaže vyšiel od organizátorov výstavy, firmy D&D Studio, ktorí sa obrátili na MFF UK, aby im túto súťaž organizačne zabezpečila. Kolektív organizátorov mal v jednotlivých ročníkoch takéto zloženie:

1994 Peter Borovanský (vedúci organizačného teamu), Štefan Dobrev, Silvia Kravčíková (administratíva), Andrej Lúčny (hlavný porotca), Monika Obrancová, Zuzka Repaská, Daniel Štefankovič;

1995 Bronislava Brejová, Rastislav Kráľovič, Ľubica Lenhartová (administratíva), Anrej Lúčny (hlavný porotca), Daniel Štefankovič, Tomáš Vinař, Michal Winczer (vedúci organizačného teamu);

1996–1997 Bronislava Brejová, Vierka Heglasová (administratíva), Rastislav Kráľovič, Anrej Lúčny (hlavný porotca), Daniel Štefankovič, Tomáš Vinař, Miroslav Wágner (výpočtová technika), Michal Winczer (vedúci organizačného teamu);

1998 Ivona Bezáková (administratíva), Bronislava Brejová, Rastislav Kráľovič, Martin Pál (administratíva), Daniel Štefankovič, Tomáš Vinař (hlavný porotca), Miroslav Wágner (výpočtová technika), Michal Winczer (vedúci organizačného teamu);

Počas súťaže a jej bezprostrednej prípravy pomáhalo vždy ešte približne desať ďalších pomocníkov.

V prvom roku súťaž iba hľadala svoju budúcu tvár. Súťaž mala iba jedno kolo, ktoré prebehlo na MFF UK. Prihlásiť sa mohli dvojčlené družstvá, ktoré musela vyslať ich škola. Súťažiaci dostali šesť programátorských problémov, ktoré mohli riešiť v programovacích jazykoch C alebo Pascal. Každé družstvo malo k dispozícii jeden počítač a počas troch hodín malo vyriešiť čím viac z predložených úloh.

Počas ďalších štyroch ročníkov sa mierne upravili pravidlá súťaže. Pribudlo domáce kolo, ktorým sa vyberalo tridsať postupujúcich družstiev, zastupujúcich rôzne školy. Súťaž sa predĺžila na štyry hodiny a v štvrtom ročníku sa zaviedlo obmedzenie, že súťaže sa môžu zúčastniť iba tí študenti, ktorí sa neumiestnili v druhom kole dva razy do piateho miesta.

Väčšina príkladov, ktoré sa v tejto súťaži objavili, je originálnych. Veríme, že ich uverejnenie v úplnej forme spolu s komentármi k ich riešeniu a výpisom programu môže byť poučné pre študentov pripravujúcich sa na podobné súťaže, pre učiteľov, ktorí hľadajú inšpiráciu, prípadne aj hotové príklady, a v neposlednom rade aj pre každého kto rád rieši, dúfame, že zaujímavé, úlohy.

Zadania a riešenia príkladov v tejto zbierke sú oproti pôvodným obsahovo nezmenené. Opravili sme iba drobné preklepy a zjednotili sme grafickú úpravu materiálu. Autormi textov príkladov a ich riešení sú (v abecednom poradí) Peter Borovanský, Bronislava Brejová, Štefan Dobrev, Rastislav Kráľovič, Monika Obrancová, Zuzka Repaská, Daniel Štefankovič, Tomáš Vinař a Michal Winczer. Dlhoročný hlavný porotca Andrej Lúčny poctivo preriešil všetky príklady a objavil v ich zadaniach množstvo nepresností a chýb ešte pred začiatkom súťaže, čím výrazne prispel k jej hladkému priebehu.

Materiál je rozdelený na tri časti: 1. zadania príkladov, 2. riešenia príkladov, 3. pravidlá súťaže a výsledkové listiny. Príklady sú rozdelené podľa toho, ako sa vyskytli v jednotlivých ročníkoch a kolách súťaže.

Každý príklad má v zadaní uvedené svoje meno, napríklad `hra`. Držíme sa pravidla, že všetky súbory súvisiace s týmto príkladom budú mať rovnaké meno a budú sa odlišovať iba príponami. `.PAS` a `.CPP` sú vyhradené pre súbory obsahujúce text programu v jazyku Pascal, resp. C a prípony `.IN` a `.OUT` sa používaju pre súbor obsahujúci vstupné údaje, resp. výstupné údaje. Súčasťou materiálu sú aj vstupné súbory obsahujúce príklady vstupných údajov.

V prípade, že v zadaní príkladu nie je uvedené inak, držíme sa konvencií:

- vstupný aj výstupný súbor sú textové súbory,
- riadky vo vstupnom súbore sa nezačínajú ani nekončia medzerami,
- na konci súboru nie sú prázdne riadky,
- jednotlivé údaje vo vstupnom súbore sú oddelené jednou medzerou,
- celé čísla sa myslia rozsahu `integer` (čo môže byť závislé od použitého prekladača programovacieho jazyka),
- vstupný aj výstupný súbor končia znakmi `CR LF`.

Súbory ako aj prípadné opravy textu sú k dispozícii na `www.edi.fmph.uniba.sk/cofax/`. V textoch úloh, programoch a vstupných dátach sme sa snažili opraviť chyby. Je temer isté, že sa nám nepodarilo nájsť všetky. V prípade, že nejakú objavíte, oznámte to, prosím na adresu `winczer@fmph.uniba.sk`, alebo KVI MFF UK, Mlynská Dolina, 842 15 Bratislava.

Na záver ešte poďakovanie Ivone Bezákovej za starostlivé prečítanie rukopisu a všetkým vyššie menovaným organizátorom zabezpečujúcim príklady, ich riešenia a opravovanie, techniku a administratívu a aj množstvu ďalších nemenovaných pomocníkov, ktorí sa podielali na príprave a priebehu súťaže a tiež firme D&D Studio, ktorá súťaž založila a finačne zabezpečila.

V Bratislave 3. apríla 2002

Michal Winczer

Zadania

1.2 Finále

1.2.1. Lov(A)

Program: `LOV.PAS` alebo `LOV.CPP`

Príklad:
```
LOV.IN
4 1 1 4 1 1 3
4 1 1 4 2 1 3

LOV.OUT
nemoze
moze
```

Vlk Rasťo po dlhšej prestávke dostal chuť na zajačinu. Po dlhej naháňačke sa mu podarilo zahnať zajačicu Zuzu do uzavretej miestnosti s rozmermi $N \times N$. Keďže obaja sú dobrí matematici, rozhodli sa, že sa nebudú nahánať ako vlk a zajac v lese, ale budú hrať o Zuzin život hru s nasledovnými pravidlami:

Na začiatku hry je vlk na políčku so súradnicami $[V_x, V_y]$ a zajac na políčku so súradnicami $[Z_x, Z_y]$. Počas hry sa striedavo pohybujú v ľubovoľnom smere hore, dole, doprava, doľava (v rámci miestnosti). Zajačica sa pohne v každom ťahu presne o Z_r políčok a vlk v každom ťahu presne o V_r políčok. Keďže zajačica má prednosť, prvá ide Zuza. Vlk chytí zajačicu, keď sa na konci svojho ťahu nachádza na tom istom políčku ako ona. Napíšte program, ktorý zistí, či môže vlk zajačicu chytiť za predpokladu, že zajačica hrá tak, aby ju nechytil.

Daná je hra s nasledovnými pravidlami:

1. Hrací plán má rozmer $N \times N$.
2. Hráč 1 (zajačica) sa na začiatku hry nachádza na políčku $[Z_x, Z_y]$.
3. Hráč 2 (vlk) sa na začiatku hry nachádza na políčku $[V_x, V_y]$.
4. Hráči striedavo idú (hráč 1 začína)
5. Hráč 1 sa v každom ťahu pohne o Z_r políčok.
6. Hráč 2 sa v každom ťahu pohne o V_r políčok.
7. Hráč 2 chytí hráča 1, keď sa na konci svojho ťahu nachádza na tom istom políčku ako hráč 1.
8. Jeden ťah o X políčok je zloženie ľubovolných X elementárnych ťahov, pričom elementárny ťah je pohyb o jedno políčko v jednom z daných smerov: hore, dole, doprava, doľava.

Napíšte program, ktorý pre miestnosť rozmeru N, súradnice zajačice $[Z_x, Z_y]$, súradnice vlka $[V_x, V_y]$, počet krokov zajačice Z_r a počet krokov vlka v jednom ťahu V_r, zistí, či môže hráč 2 chytiť hráča 1. Predpokladáme, že hráč 1 hrá tak, aby ho hráč 2 nechytil.

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko zadaní. Každé zadanie je v samostanom riadku. Zadanie tvoria nezáporné celé čísla N, Z_x, Z_y, V_x, V_y, Z_r, V_r oddelené medzerou.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každé zadanie v samostatnom riadku správu `moze`, keď vlk chytí zajačicu, a `nemoze`, ak ju nechytí.

1.2.2. Kontrolóri kondenzátorov(E)

Program: `KAPACITY.PAS` alebo `KAPACITY.CPP`

Príklad:
```
KAPACITY.IN
5 2 4 -2 3 4 -3 2 -2 -5 1 0 -3
6 4 -5 3 4 -1 0 4 1 -6 -2 4 5 5

KAPACITY.OUT
12 11 10 6 3 1 0
13 12 10 9 8 4 1 0
```

V podniku na výrobu kondenzátorov pracuje výstupná kontrola. Každé ráno sa kontrolór dozvie N – maximálnu povolenú odchýlku toho dňa vyrábaných kondenzátorov. Daný deň kontrolór pre každý výrobok zistí rozdiel predpísanej kapacity a skutočnej kapacity výrobku. Pokiaľ daná hodnota nie je z množiny $\{-N, ..., N\}$, kondenzátor vyradí. V opačnom prípade toto číslo zapíše do súboru (každý deň sa píše do nového riadku). Po určitom období sa vyhodnotí úspešnosť výroby pre každý pracovný deň v takzvanej dlhodobej správe o úspešnosti výroby. Kontrolór musí uviesť do správy pre každý pracovný deň údaje o tom, koľko bolo výrobkov s absolútnou hodnotou rozdielu väčšou ako X, pre každé X z množiny $\{-1, .., N\}$. Napíšte program, ktorý bude kontrolórovi pomáhať pri písaní polročnej správy.

Špecifikácia vstupu:

Každý riadok vstupného súboru obsahuje údaje výstupnej kontroly za jeden deň. Riadky sa začínajú číslom N a pokračujú rozdielmi zistenými výstupnou kontrolou, oddelenými medzerou. Všetky rozdiely sú celočíselné. Môžete predpokladať, že $N < 4000$.

Špecifikácia výstupu:

Program pre každý zo vstupných riadkov vypíše riadok obsahujúci počty rozdielov, ktorých absolútne hodnoty sú väčšie ako X, pre $-1, 0, 1, .., N$, oddelené medzerou.

1.2.3. Ostrov(C)

Program: `OSTROV.PAS` alebo `OSTROV.CPP`

Príklad:

```
OSTROV.IN
7
0 0 0 1 0 0 0
0 0 0 1 0 0 0
0 0 1 1 1 0 0
0 1 1 1 1 1 1
0 1 1 1 1 0 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
6 5
3 5
7
0 0 0 1 0 0 0
0 0 1 1 1 0 0
0 1 1 1 1 1 0
0 1 1 1 1 0 0
0 1 1 1 1 0 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
4 5
3 3

OSTROV.OUT
Tresk 7
naraz 9
```

Bol raz jeden pirát, volal sa Bumbác, a mal dvoch synov Treska a Pleska. Bumbác bol veľmi šikovný a podarilo sa mu nahromadiť veľký majetok. Svoje bohatstvo ukryl na ostrove Krach v jednej jaskyni na pobreží. Keď cítil, že sa blíži jeho posledná hodina, zavolal si synov a vysvetlil im, kde poklad nájdu. Po otcovej smrti sa Tresk a Plesk vybrali dedičstvo hľadať. Kedže Tresk a Plesk boli vychovaní podľa pirátskych zásad, nechceli sa o poklad deliť. Preto sa dohodli, že sa vylodia niekde na ostrove Krach a Plesk pôjde doľava - v smere hodinových ručičiek, Tresk doprava - proti smeru hodinových ručičiek. Obidvaja pôjdu pozdĺž pobrežia rovnako rýchlo, nebudú si cestu skracovať a nemôžu ísť šikmo. Poklad bude patriť tomu, kto prvý objaví jaskyňu.

Štvorcová mapa ostrova pozostáva iba z 0 a 1 (1 - pevnina, 0 - more). Na mape je len jeden ostrov, t.j. len jeden súvislý útvar pozostávajúci z 1. Štvorček na mape je na pobreží, ak niektorý z jeho 8 susedov je more. Miesto vylodenia a miesto uloženia pokladu sú dva body na pobreží ostrova zadané súradnicami. Ísť po pobreží doľava (doprava) znamená ísť po pobrežných políčkach čo najviac vľavo (vpravo). Nie je dovolené ísť šikmo, teda je možné prejsť len na políčko susediace stranou. Napríklad pre mapu

```
0 1 0 0
0 1 1 1
0 1 0 0
0 0 0 0
```

a súradnice vylodenia $[3, 2]$ ísť popri pravom pobreží znamená, ísť po políčkach v poradí $[2, 2]$, $[2, 3]$, $[2, 4]$, $[2, 3]$, $[2, 2]$, $[1, 2]$, $[2, 2]$, $[3, 2]$.

Pre každý zadaný ostrov a polohu vylodenia a pokladu, je vašou úlohou vypísať, ktorý z bratov nájde poklad prvý a koľko políčok musí prejsť.

Špecifikácia vstupu:

Vstupný súbor obsahuje aspoň jeden blok zo zadaním. V prvom riadku každého bloku je rozmer štvorcovej mapy n, $n \leq 100$, v nasledujúcich n riadkoch je mapa ostrova, v každom riadku je n čísiel oddelených medzerami. Posledné dva riadky v bloku obsahujú súradnice vylodenia (riadok a stĺpec) a súradnice pokladu (riadok a stĺpec).

Špecifikácia výstupu:

Vo výstupnom súbore bude pre každý blok jeden riadok obsahujúci meno toho, čo našiel prvý poklad a počet jeho krokov po pobreží oddelený jednou medzerou. V prípade, že poklad nájdu obaja bratia naraz, bude riadok obsahovať slovo `naraz` a počet ich krokov.

1.2.4. Matica(D)

Program: `MATICA.PAS` alebo `MATICA.CPP`

U-matica je matica rozmerov $m \times n$, ktorej všetky riadky a stĺpce sú vzostupne usporiadané, teda pre každé r, j, k také, že $1 \leq r \leq m$ a $1 \leq j < k \leq n$, je $a[r, j] \leq a[r, k]$ a tiež pre každé j, k, s také, že $1 \leq j < k \leq m$ a $1 \leq s \leq n$, je $a[j, s] \leq a[k, s]$ Pre takúto maticu sa ľahšie zisťuje, či obsahuje nejaké číslo d.

Napíšte program, ktorý zistí, či sa dané čísla nachádzajú v U-matici.

Vašou úlohou je pre danú postupnosť celých čísel $d_1, d_2, \ldots, d_q$ čo najskôr zistiť, či sa čísla nachádzajú v zadanej matici alebo nie.

Príklad:

```
MATICA.IN
2 3
1 3 5
2 4 6
4
3 10 1 12
2 2
5 7
8 9
5
1 5 6 8 10
0 0
```

```
MATICA.OUT
ano nie ano nie
nie ano nie ano nie
```

Špecifikácia vstupu:

Vstupný súbor pozostáva z niekoľkých blokov. Každý blok tvorí jedno zadanie. V prvom riadku bloku sú rozmery matice m (počet riadkov), $1 \leq m \leq 100$ a n (počet stĺpcov), $1 \leq n \leq 100$, oddelené medzerou. V nasledujúcich m riadkoch sú prvky matice $a[i,j]$, oddelené medzerou, $-1000000 \leq a[i,j] \leq 1000000$. V prvom z nich sú prvky $a[1,1]$, $a[1,2], \ldots, a[1,n]$, v druhom sú $a[2,1]$, $a[2,2], \ldots, a[2,n]$ a v m-tom $a[m,1]$, $a[m,2], \ldots, a[m,n]$. V ďalšom riadku je počet čísel k v postupnosti, ktoré sa budú hľadať v matici. V nasledujúcom riadku sú čísla $d_1, d_2, \ldots, d_k$, oddelené medzerou. Posledný blok obsahuje iba $m = n = 0$.

Špecifikácia výstupu:

Výstupný súbor obsahuje jeden riadok pre každé zadanie okrem posledného, v ktorom $m = n = 0$. V riadku je `ano`, ak sa zodpovedajúce d_i v matici nachádza a `nie` keď sa nenachádza. Za každým `ano` alebo `nie` je jedna medzera.

1.2.5. Kocka(E)

Program: `KOCKA.PAS` alebo `KOCKA.CPP`

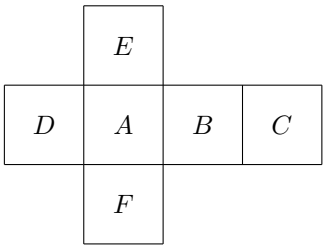
Príklad:

```
KOCKA.IN
RGBYWM
a1b1c1d1e1f1f3e3d3c3b3a3
YWMRGB
a3b3c3

KOCKA.OUT
    MMM
    MMM
    MMM
YYY RRR GGG BBB
YYY RRR GGG BBB
YYY RRR GGG BBB
    WWW
    WWW
    WWW

    BWW
    BBY
    RRY
YRG YYW BBM BBR
BRG YYG WWG MMM
BRG YYG WWG MMM
    WWM
    GGM
    RRR
```

Máme Rubikovu kocku. Každá jej stena je označená písmenami A, B, C, D, E, F nasledujúcim spôsobom:



Na začiatku je Rubikova kocka poskladaná, teda každá stena pozostáva z 9-tich malých štvorcov rovnakej farby. Každá stena má na začiatku inú farbu. Máme tri druhy otáčania stien, prvý typ je otočenie o 90°, druhý o 180° a tretí o 270°. Všetky otáčania sa prevádzajú ľavotočivým smerom, keď sa na zvolenú stranu pozeráme spredu.

Vašou úlohou je simulovať otáčanie stien tejto kocky, teda pre zadanú postupnosť otáčania stien vypísať výsledný stav kocky (vo forme plášťa).

Špecifikácia vstupu:

Vstupný súbor sa skladá z blokov. Každý blok má dva riadky. Prvý riadok obsahuje reťazec farieb jednotlivých stien. Prvý znak reťazca je farba steny A, druhý farba steny B, atď. Napríklad: `RGBYWM` - stena A má farbu `R`, stena B farbu `G`, atď. V druhom riadku je reťazec otáčaní stien. V ňom je vždy uvedená stena, reprezentovaná znakmi `a, b, c, d, e, f` a typ otáčania 1, 2 alebo 3. Teda ak je reťazec otáčaní `a1c3d2`, na zloženej kocke sa vykonajú nasledujúce otočenia: Najprv otočíme stenu A o 90° doľava, potom stenu C o 270° doľava a nakoniec stenu D o 180° doľava.

Špecifikácia výstupu:

Výstupny súbor obsahuje plášte kociek po otáčaní oddelené prázdnym riadkom. Za posledným plášťom už nie je prázdny riadok. Plášte majú taký tvar, ako na obrázku vyššie.

1.2.6. Preteky(F)

Program: `PRETEKY.PAS` alebo `PRETEKY.CPP`

Ako iste viete, Preteky Mieru sa majú s veľkou pompou znovu organizovať. Záujem o sponzorovanie prejavili mnohé spoločnosti. Každá z nich však chce vystaviť svoje reklamné tabule len na istom intervale

$\langle x_i, y_i \rangle$ z trasy pretekov. Organizátori chcú, aby nevyužitých zostalo čo najmenej kilometrov, lebo vtedy zarobia najviac peňazí. Žiadna spoločnosť ale nechce, aby sa s inou delila o nejaký úsek. Keďže každá spoločnosť trvá bezpodmienečne na svojom intervale $\langle x_i, y_i \rangle$, organizátori musia niektoré sponzorské zmluvy zrušiť. Za takýchto okolností by radi vedeli, koľko zarobia peňazí, čo záleží od toho, koľko kilometrov dokážu pokryť reklamami a koľko zostane nevyužitých.

Napíšte program, ktorý pre danú dĺžku pretekov a záujem spoločností o úseky vypíše, koľko kilometrov z celkovej trasy bude bez reklamných tabúľ. Keďže preteky sa plánujú na niekoľko rokov vopred, program musí umožniť zistiť žiadaný počet kilometrov pre niekoľko ročníkov.

Príklad:

```
PRETEKY.IN
20 4
0 4 2 8 5 19 10 15
10 6
1 9 0 2 2 4 4 6 6 8 8 10
100 4
0 20 19 49 25 35 10 15

PRETEKY.OUT
2
0
65
```

Formálne si úlohu pre jeden ročník môžeme predstaviť nasledovne: Je daný interval $\langle 0, N \rangle$ a K jeho podintervalov $\langle x_i, y_i \rangle$. Hľadáme také pokrytie intervalu $\langle 0, N \rangle$ niekoľkými zadanými intervalmi, pri ktorom je nepokrytá plocha minimálna, a pritom žiadne dva pokrývajúce intervaly nemajú prienik (až na okrajové body). Je potrebné vypočítať veľkosť tejto nepokrytej plochy, t.j. súčet dĺžok úsekov, ktoré nie sú pokryté. Pod dĺžkou úseku $\langle x_i, y_i \rangle$ rozumieme $y_i - x_i$.

Špecifikácia vstupu:

Vstupný súbor pozostáva z blokov, kde každý blok predstavuje zadanie pre jeden ročník Pretekov Mieru. V prvom riadku bloku je zadaná dĺžka trasy N a počet spoločností K. Môžete predpokladať, že $0 < N \leq 1000$, $0 \leq K \leq 10000$. V druhom riadku je K dvojíc x_i, y_i udávajúcich záujem i-tej spoločnosti o interval trasy $\langle x_i, y_i \rangle$.

Špecifikácia výstupu:

Výstupný súbor má niekoľko riadkov, pričom i-ty riadok obsahuje odpoveď na zadanie dané i-tým blokom vstupu. Odpoveďou je celé číslo.

2.1 Domáce kolo

2.1.1 Archeológ

Program: `ARCHEOLO.PAS` alebo `ARCHEOLO.CPP`

Príklad:

```
ARCHEOLO.IN
1
6 4
.XX...
.X....
.X.XXX
....X.
X...XX
X.....
XXXX
OOOX
XOXX
XOOX

ARCHEOLO.OUT
3 4
```

Archeológ Tadeáš mal plný stôl rôznych hlinených črepov a zostavoval z nich nádoby. Práve chcel prilepiť posledný kúsok k jednému tanieru, ale vtom zazvonil telefón. Tadeáš všetko rýchlo položil na stôl a keď sa vrátil, nevedel posledný chýbajúci úlomok medzi ostatnými nájsť. Napíšte program, ktorý nájde chýbajúci úlomok.

Špecifikácia vstupu:

Vstupný súbor obsahuje v prvom riadku počet zadaní. Zadanie príkladu sa začína číslami N a M v samostatnom riadku ($N < 100$, $M < 100$). Nasledujúcich N riadkov po N znakov reprezentuje rozloženie úlomkov na stole. Pomocou písmena `X` sú vyznačené hlinené úlomky a `.` (bodka) označuje medzery medzi nimi. Žiadne dva úlomky sa nedotýkajú. Ďalších M riadkov po M stĺpcov obsahuje tvar diery v tanieri, do ktorej chýbajúci úlomok presne zapadá (priložený úlomok nemôže vytŕčať z obdĺžnika $M \times M$). Diera je vyznačená znakmi `O`, nádoba znakmi `X`. Máte veľké šťastie, lebo Tadeáš úlomok aj tanier rovnako otočil.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každé zadanie jeden riadok s číslom riadku a číslom stĺpca prvého štvorčeka hľadaného úlomku. Čísla sú oddelené medzerou. Prvý štvorček je ten, na ktorý najskôr natrafíme, ak postupujeme po riadkoch zhora dole, v riadku zľava doprava.

2.1.2 Hra

Program: `HRA.PAS` alebo `HRA.CPP`

CUFT je stará burundijská loptová hra. Hrajú ju dvaja hráči, ktorí sa postavia oproti sebe na obdĺžnikovom ihrisku. Ihrisko má rozmery $M \times N$ ($M < N$) a hráči stoja v stredoch kratších strán. Každý z

Príklad:

```
HRA.IN
2
10 10
30 100
230 10
10 100
0 19.8
180 101.8

HRA.OUT
nezrazia sa
zrazia sa
```

nich má tvrdú loptu z kože nosorožca naplnenú pieskom. Steny ihriska sú z pružného dreva kaučukovníkov. Na znamenie šamana každý z hráčov hodí svoju loptu tak, aby sa kotúľala po zemi. Lopta sa kotúľa rovnakou rýchlosťou a dokonale sa odráža od dlhších strán ihriska. Keď sa lopty na ihrisku zrazia, obaja hráči prehrajú, inak obaja vyhrajú a nasleduje veľká oslava (hráči teda vyhrajú, keď sa lopty po niekoľkých odrazoch od dlhších strán ihriska vykotúľajú von z ihriska kratšími stranami). V kmeni Kumbov hráči natoľko zleniveli, že sa im nechce hádzať ťažká lopta. Oslavovať by však napriek tomu chceli. Preto potrebujú program, ktorý by im pomohol určiť a vypísať pre zadané M, N, uhol hodu prvého hráča, rýchlosť hodu prvého hráča, uhol hodu druhého hráča a rýchlosť hodu druhého hráča, či majú hráči oslavovať alebo nie.

Uhol sa zadáva v stupňoch proti smeru hodinových ručičiek. Uhol 0° je rovnobežný s ihriskom. Uhol hodu prvého hráča je medzi 0° a 90° alebo medzi 270° a 360°, uhol hodu druhého hráča je medzi 90° a 270°.

Špecifikácia vstupu:

Vo vstupnom súbore je v prvom riadku počet hraných hier. Pre každú hranú hru nasledujú tri riadky obsahujúce popísané údaje.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každú hranú hru v samostatnom riadku `zrazia sa` alebo `nezrazia sa`, podľa toho, či nastane zrážka.

2.1.3 Karty

Program: `KARTY.PAS` alebo `KARTY.CPP`

Baz, Daz, Gaz a Maz sú štyria kamaráti, ktorí sa každý večer stretávajú, aby si zahrali svoju obľúbenú kartovú hru, ktorá sa volá "Srdcia". Hra sa hrá s 52 kartami. Sada kariet obsahuje štyri farby - List, Káro, Srdce a Pika. Z každej farby je trinásť kariet - 2,3,4,5,6,7,8,9,D,J,Q,K,A. Na začiatku hry sa karty zamiešajú a každý hráč dostane 13 kariet, ktoré si na ruke usporiada podľa farby a v rámci jednej farby podľa hodnoty (v poradiach, aké sme uviedli). Rozdáva sa striedavo každému hráčovi po jednej karte. Táto činnosť im z celej hry zaberá veľa času a preto by potrebovali robota, ktorý zoberie zamiešaný balíček kariet rozdá ich a každému hráčovi podá jeho usporiadané karty. Vašou úlohou je napísať program pre takéhoto robota.

Špecifikácia vstupu:

Vo vstupnom súbore je v prvom riadku počet hraných hier. Pre každú hru je v samostatnom riadku 104 znakov - postupnosť kariet v balíčku.

Špecifikácia výstupu:

Do výstupného súboru vypíšte usporiadané karty pre jednotlivých hráčov, pre každú hru. Karty každého hráča do samostaného riadku. Za štvrtým hráčom dajte jeden prázdny riadok.

Príklad:

```
KARTY.IN
1
2P3P4P5P6P7P8P9PDPJPQPKPAP2L3L4L5L6L7L8L9LDL     (pokračovanie)
       JLQLKLAL2S3S4S5S6S7S8S9SDSJSQSKSAS2K3K4K5K6K7K8K9KDKJKQKKKAK

KARTY.OUT
5L9LKL3K7KJK4S8SQS2P6PDPAP
2L6LDLAL4K8KQK5S9SKS3P7PJP
3L7LJL5K9KKK2S6SDSAS4P8PQP
4L8LQL2K6KDKAK3S7SJS5P9PKP
```

2.1.4 Malá násobilka

Program: `MALA-NAS.PAS` alebo `MALA-NAS.CPP`

V krajine XYZ je 36 miest očíslovaných podľa veľkosti od 2 do 37. V meste N používajú N-kovú číselnú sústavu (t.j. v hlavnom a najväčšom meste dvojkovú, v meste 10 desiatkovú, v meste 37 tridsaťsedmičkovú).

Príklad:
```
MALA-NAS.IN:
2
4
2

MALA-NAS.OUT:
   1   2   3  10
   2  10  12  20
   3  12  21  30
  10  20  30 100

   1  10
  10 100
```

V mestách s číslom väčším než 10 používajú ako cifry *10*, *11*, *12*, ..., *36* veľké písmená *A*, *B*, *C*, ..., *Z* a znak *?* (teda v meste 14 zapíšu číslo *29777* ako *ABCD*). Každé dieťa v tejto krajine musí naspamäť ovládať malú násobilku, prirodzene v číselnej sústave svojho mesta. Zuzka bývala v meste 2 a žilo sa jej výborne, lebo jej násobilka obsahovala len 4 čísla. Ale potom sa jej rodičia rozhodli presťahovať do mesta N a úbohá Zuzka sa musí učiť oveľa väčšiu násobilku. Potrebovala by program, ktorý jej pre zadané N vypíše tabuľku malej násobilky pre toto mesto. Tabuľka obsahuje súčiny všetkých dvojíc čísel od 1 po N. Napíšte program, ktorý Zuzke pomôže vytvoriť potrebné tabuľky.

Špecifikácia vstupu:

Vstupný súbor má v prvom riadku počet tabuliek, ktoré treba vypísať. V nasledujúcich riadkoch je po jednom čísle N.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každé N zo vstupného súboru príslušnú tabuľku. Tabuľky sú oddelené prázdnym riadkom. Čísla v tabuľke sú vypísané na 4 miesta a zľava doplnené medzerami.

2.1.5 Žabka

Program: `ZABKA.PAS` alebo `ZABKA.CPP`

Príklad:
```
ZABKA.IN
2
4
0001
0010
0100
1000
3
101
000
100

ZABKA.OUT
nemoze prejst
moze prejst
```

Žabka Monika sa rozhodla dostať z jedného rohu močiara porasteného leknami do jeho druhého rohu. Pretože je veľmi unavená, nechce sa jej plávať. Močiar má tvar štvorca $N \times N$, pričom na začiatku je žabka v ľavom dolnom rohu a chce sa dostať do pravého horného rohu. Žabka sa môže hýbať iba v smeroch rovnobežných so stranami močiara a môže prejsť z lekna na susedné lekno, alebo preskočiť cez jedno políčko na ďalšie lekno. Močiar je reprezentovaný N riadkami po N stĺpcoch jednotiek a núl, pričom jednotka znamená lekno a nula znamená vodu. Pre vstup N a maticu jednotiek a núl rozmerov $N \times N$ zistite, či môže žabka prejsť z jedného rohu močiara na druhý.

Špecifikácia vstupu:

Vstupný súbor má v prvom riadku počet zadaní uvedených v súbore. Každé zadanie začína riadkom s číslom N, za ktorým nasleduje N riadkov, s ktorých každý obsahuje N núl alebo jednotiek.

Špecifikácia výstupu:

Do výstupného súboru, pre každé zadanie na vypíšte do samostatného riadku text `nemoze prejst` alebo `moze prejst`, podľa toho, či sa jej podarí preskákať.

2.2 Finále

2.2.1. Hra(A)

Program: `HRA.PAS` alebo `HRA.CPP`

Baz a Gaz sa zapojili do súťaže v riešení hlavolamov. Hlavolam sa skladá zo štvorcovej krabičky rozmerov $N \times N$ a $N^2 - 1$ štvorcových dielikov očíslovaných číslami od 1 po $N^2 - 1$. Na začiatku sú dieliky poukladané podľa čísiel tak, že v hornom riadku sa nachádzajú dieliky 1 až N (v tomto poradí), v ďalšom riadku dieliky $N+1$ až $2N$ atď. V pravom dolnom rohu sa nenachádza žiaden dielik. Riešiteľ môže v každom ťahu presunúť dielik susediaci s prázdnym políčkom na toto políčko. Úlohou je nájsť postupnosť ťahov, ktorá vedie k danému rozloženiu dielikov. Bazovi sa podarilo úlohu vyriešiť a preto sa rozhodol, že zavolá Gazovi a povie mu výsledok. Postupne mu oznamuje čísla dielikov, ktorými má ťahať. Gaz si čísla zapísal na papier a teraz rozmýšľa, či sa Baz nepomýlil. Zdá sa mu totiž, že ťahanie dielikmi v uvedenom poradí vôbec nie je možné.

Napíšte program, ktorý Gazovi pomôže. Váš program načíta zo vstupu čísla dielikov, ktoré si Gaz zapísal a zistí, či sa dá ťahať dielikmi v uvedenom poradí.

Špecifikácia vstupu:

V prvom riadku vstupného súboru je počet riešení úloh, nachádzajúcich sa v súbore a v ďalších riadkoch sa postupne nachádzajú riešenia úloh. Na začiatku každého riešenia je v samostatnom riadku číslo $N \leq 100$ - rozmer krabičky. V nasledujúcom riadku sa nachádza postupnosť dielikov, ktorými treba ťahať, postupnosť je ukončená nulou.

Špecifikácia výstupu:

Pre každé riešenie úlohy váš program zapíše do samostatného riadku výstupného súboru `DA SA`, ak sa dielikmi dá ťahať v uvedenom poradí, alebo `NEDA SA`, ak sa to nedá.

Príklad:

```
HRA.IN                                                    HRA.OUT
3                                                         DA SA
4                                                         NEDA SA
15 11 7 8 12 7 10 14 13 9 14 6 5 1 2 3 4 12 7 15 11 13 0  DA SA
4
12 8 7 11 15 14 15 11 7 8 12 0
3
8 7 4 5 6 3 2 1 5 6 7 8 3 0
```

2.2.2. Kalendár(B)

Program: `KALENDAR.PAS` alebo `KALENDAR.CPP`

Príklad:

```
KALENDAR.IN
10051985
25051995
01011995
-1
```

```
KALENDAR.OUT
10.5.1987 BOLA NEDELA V 19. TYZDNI
25.5.1995 BOL STVRTOK V 22. TYZDNI
1.1.1995 BOLA NEDELA V 1. TYZDNI
```

Napíšte program, ktorý pre daný dátum určí, ktorý deň v týždni a ktorý týždeň v roku to bol. Môžete predpokladať, že zadaný dátum je medzi 1.1.1901 a 31.12.1999.

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko riadkov a v každom z nich je celé číslo predstavujúce dátum v tvare DDMMRRRR (napríklad 01061995 predstavuje dátum 1. 6. 1995). Vstupný súbor je ukončený samostatným riadkom, v ktorom je číslo -1.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každý riadok vstupného súboru v samostatnom riadku text podľa nasledujúceho vzoru:

```
25.5.1995 BOL STVRTOK V 22. TYZDNI
```

Týždeň končí v nedeľu, prvý týždeň v roku je týždeň, v ktorom sa nachádza 1. január.

2.2.3. Koláč(C)

Program: `KOLAC.PAS` alebo `KOLAC.CPP`

Príklad:

```
KOLAC.IN
1
2 3
2 2
3 2
3 3
```

```
KOLAC.OUT
2 2
```

V rodine Hilltonovcov sa každú nedeľu pečie veľký koláč tvaru štvorca, ktorého rozmery sú $2N \times 2N$. Na jeho povrchu sa nachádza K hrozienok, ktoré mama Hilltonová vždy po jednom hádže na koláč a zapisuje si súradnice políčka, do ktorého hrozienko padlo (ľavé horné políčko má súradnice $[1, 1]$). Malý Dežko má z celého koláča rád iba hrozienka a preto sa snaží, aby ich mal čo najviac. Môže si z koláča vyrezať ľubovoľný štvorec rozmerov $N \times N$ (aj zo stredu), ale najradšej taký, ktorý obsahuje najviac hrozienok. Preto vždy pozorne študuje mamine zápisy a potom sa rozhodne, ktorý štvorec si vyreže.

Urobte pre Dežka program, ktorý načíta zo vstupného súboru mamine zápisky a vypíše súradnice ľavého horného rohu štvorca, ktorý má Dežko vyrezať tak, aby mal najviac hrozienok.

Špecifikácia vstupu:

V prvom riadku vstupného súboru je počet zadaní, ktoré sa v ňom nachádzajú. V ďalších riadkoch sa postupne nachádzajú zadania. V prvom riadku každého zadania sa nachádzajú čísla N (polovica rozmeru koláča), K (počet hrozienok na jeho povrchu). Potom nasleduje K riadkov so súradnicami hrozienok zapísané

v poradí stĺpec, riadok. Môžete predpokladať, že $N \leq 50$ a že žiadne dve hrozienká v jednom koláči nemajú rovnaké obe súradnice.

Špecifikácia výstupu:

Výstupom sú súradnice (stĺpec, riadok) ľavého horného rohu štvorca $N \times N$, ktorý obsahuje najviac hrozienok. Pre každé zadanie dajte výsledok do samostaného riadku.

2.2.4. Mince(D)

Program: `MINCE.PAS` alebo `MINCE.CPP`

Pri výplatách vo veľkých podnikoch sa často vyskytne problém, že chýbajú drobné na vyplatenie výplat všetkým zamestnancom. Vašou úlohou je napísať program, ktorý pre zadané výplaty zamestnancov určí, koľko ktorých bankoviek a mincí bude treba vyžiadať, aby bolo možné vyplatiť všetky výplaty. Samozrejme, počet jednotlivých platidiel musí byť minimálny.

Príklad:

```
MINCE.IN
2
753.60
428.73
-1
0.75
-1

MINCE.OUT
11 1 1 0 1 6 2 0 3 0 3
0 0 0 0 0 0 1 1 0 0 0
```

Špecialitou nášho podniku je, že zamestnanci dostávajú svoje výplaty v dolároch. Prípustné platidlá sú teda: \$100, \$50, \$20, \$10, \$5, \$1, 50 centov, Quarters (25 centov), Dimes (10 centov), Nickels (5 centov) a Pennies (1 cent); platí 1 cent=\$0.01. Keď teda zamestnanec má dostať na výplatu \$750.75 tak dostane 7 stodolároviek, jednu päťdesiatdolárovku, 50 centov a jeden Quarter (a nie 750 jednodolároviek a 75 pennies).

Napíšte program, ktorý načíta sumy, ktoré treba vyplatiť jednotlivým pracovníkom a vypíše rozpis, koľko platidiel si má pokladníčka vyžiadať z jednotlivých druhov. Počet vyžiadaných platidiel musí byť minimálny.

Špecifikácia vstupu:

V prvom riadku vstupného súboru sa nachádza celé číslo N, ktoré udáva počet zadaní. Nasledujúce riadky potom obsahujú jednotlivé zadania. V každom riadku zadania sa nachádza jedno kladné reálne číslo, ktoré znamená výšku výplaty jedného pracovníka podniku. Každé zadanie je ukončené samostatným riadkom s číslom -1. Výšky výplat sú celočíselným násobkom 0.01.

Špecifikácia výstupu:

Výstupný súbor musí pre každé zadanie obsahovať v samostatnom riadku 11 čísel oddelených medzerou. Každé číslo predstavuje počet potrebných platidiel jedného druhu a to v poradí, ako sú vymenované vyššie. Môžete predpokladať, že počet potrebných platidiel z každého druhu je maximálne 32000.

2.2.5. Obchod(E)

Program: `OBCHOD.PAS` alebo `OBCHOD.CPP`

Príklad:

```
OBCHOD.IN
4
1
0
5
0 0 1 2 1
4
0 1 2 3
4
0 0 0 0

OBCHOD.OUT
1
1 5 3 2 4
4 3 2 1
1 2 3 4
```

V dnešných dňoch ruské obchody zívajú prázdnotou. Ale v Rusku si ľudia zvykli na disciplínu a tak stále stoja v radoch pred obchodmi. Ba čo viac, ich zvyky sú tak silné, že každý deň prichádzajú k obchodu v rovnakom poradí. Preto sa každý deň stojac v rade rozprávajú s tými istými ľuďmi.

Istá predavačka sa rozhodla využiť generátor náhodných čísiel na svojej kalkulačke, ktorý normálne používa na generovanie kurzov západných mien. Každému zákazníkovi pri príchode vygeneruje číslo od nula po počet ľudí, ktorí prišli pred ním. Toto číslo vyjadruje, o koľko sa môže zákazník posunúť v rade smerom dopredu (tzn. koľkých zákazníkov predbehne). Takto sa môže zákazník časom stretnúť s hociktorým iným človekom v rade.

Vašou úlohou je napísať program, ktorý určí konečné poradie, v ktorom zákazníci strávia zvyšok dňa.

Špecifikácia vstupu:

V prvom riadku vstupného súboru sa nachádza počet príkladov v súbore. Dáta pre každý príklad sú v súbore v dvoch riadkoch. V prvom z nich je číslo N ($N < 1000$) udávajúce počet zákazníkov v rade. V druhom riadku je N čísel oddelených medzerami, i-te z nich je číslo vygenerované pre i-teho zákazníka (a teda je z rozsahu od 0 do $i - 1$).

Špecifikácia výstupu:

Pre každý príklad zo vstupného súboru vypíšte do jedného riadku N čísel, kde i-te z nich určuje, na ktorom mieste v rade stál i-ty zákazník po príchode všetkých zákazníkov.

2.2.6. Stroj(F)

Program: `STROJ.PAS` alebo `STROJ.CPP`

Starý Jim si kúpil nový stroj na vyrezávanie plechových mnohouholníkov. Základom stroja je rameno, ktoré sa môže posunúť do ľubovoľného zadaného bodu (posunúť sa do bodu (x, y) znamená posunúť rameno po priamke z bodu (mx, my) do bodu (x, y), kde súradnice (mx, my) označujú momentálnu polohu ramena). Pre každú súčiastku dostane stroj postupnosť príkazov, podľa ktorých sa má rameno hýbať, aby vyrezal súčiastku. Stroj pozná dva druhy príkazov, `M x y` - ktorý posunie rameno do bodu (x, y) a príkaz `E`, ktorý spôsobí, že sa stroj presunie z momentálneho bodu po priamke do bodu, v ktorom začal a ukončí vyrezávanie.

Príklad:

`STROJ.IN`
```
2
2.00
M 10.00 10.00
M 20.00 10.00
M 10.00 20.00
E
1.00
M 2.00 2.00
M 3.50 2.00
M 4.15 2.00
M 4.15 5.22
M 2.23 5.22
M 2.00 5.22
E
```

`STROJ.OUT`
```
M 9.00 9.00
M 22.41 9.00
M 9.00 22.41
E
M 1.50 1.50
M 3.50 1.50
M 4.65 1.50
M 4.65 5.72
M 2.23 5.72
M 1.50 5.72
E
```

Starý Jim napísal postupnosti príkazov pre všetky konvexné súčiastky od výmyslu sveta, ale vtom si uvedomil, že zabudol zobrať do úvahy šírku plameňa, ktorým sa súčiastka vyrezáva a stroj by mu teda vyrezal menšie súčiastky. Teraz je veľmi nešťastný, lebo musí všetky príkazy prepočítať, tak aby mu stroj vyrezal súčiastky správnych rozmerov.

Napíšte Jimovi program, ktorý načíta zo vstupu postupnosť príkazov pre stroj a na výstup vypíše novú postupnosť príkazov, ktorá uvažuje šírku plameňa. Plameň má kruhový prierez a jeho šírka je priemer tohoto kruhu. Výsledná súčiastka má mať pôvodne uvažovaný rozmer.

Špecifikácia vstupu:

V prvom riadku vstupného súboru je počet súčiastok popísaných v súbore (celé číslo), v ďalších riadkoch sú popisy jednotlivých súčiastok. V prvom riadku popisu súčiastky je uvedená šírka plameňa (reálne číslo) a v ďalších riadkoch sa postupne nachádzajú postupnosti príkazov. Každý príkaz je zapísaný v samostatnom riadku. Každá vyrezávaná súčiastka je konvexná, má najviac 1000 vrcholov a môžete predpokladať, že je na samostatnom dostatočne veľkom plechu.

Špecifikácia výstupu:

Výstupom je nová postupnosť príkazov, ktorá zohľadňuje šírku plameňa. Každý príkaz je zapísaný v samostatnom riadku. Výsledná postupnosť príkazov musí mať rovnakú dĺžku ako vstupná postupnosť a nesmie meniť poradie vyrezávania jednotlivých hrán. Súradnice bodov vypisujte na dve desatinné miesta.

2.2.7. Váhy(G)

Program: `VAHY.PAS` alebo `VAHY.CPP`

Príklad:

`VAHY.IN`
```
2
3 1
320
640
480
300
2 2
300
100
900
487
```

`VAHY.OUT`
```
NIE
ANO
```

Čarodejník Ignác si jedného letného dňa kúpil červenú a modrú farbu, aby pekne nafarbil všetky závažia, ktoré ležali doma na polici. Keď ich nafarbil, napadla ho ťažká úloha: Môžem položiť na dvojramennú váhu závažia tak, aby na ľavej strane boli iba červené a na pravej strane iba modré závažia a váha bola v rovnováhe, pričom na každej strane je aspoň jedno závažie? Pretože bol Ignác čarodejník, mohol si pomáhať kúzelnou paličkou, ktorá ak s ňou ukáže na nejaké závažie, mu vyrobí jeho kópiu, teda bude mať nové závažie rovnakej farby a hmotnosti (paličku môže Ignác použiť na to isté závažie aj viac krát). Nosnosť váhy, s ktorou Ignác pracuje je obmedzená, teda na jednu misku nemožno položiť ľubovoľnú hmotnosť.

Urobte pre Ignáca program, ktorý načíta počet modrých závaží, počet červených závaží, ich hmotnosti a zistí, či je možné položiť závažia na váhu tak, aby boli splnené požadované podmienky, pričom Ignác môže používať čarovnú paličku.

Špecifikácia vstupu:

V prvom riadku vstupného súboru je počet zadaní N, ktoré obsahuje. V ďalších riadkoch sa nachádza N zadaní. V prvom riadku každého zadania sa nachádzajú čísla M, C (počty modrých a červených závaží). Potom nasleduje M riadkov - hmotnosti modrých a ďalších C riadkov - hmotnosti červených závaží.

Hmotnosť je vždy celé číslo v rozmedzí od 1kg do 1000kg. Môžete predpokladať, že na začiatku mal Ignác najviac 40 závaží. Na jednu misku váhy možno položiť najviac 1000kg.

Špecifikácia výstupu:
Pre každé zadanie váš program zapíše do samostatného riadku výstupného súboru odpoveď `ANO`, ak sa dajú splniť požadované podmienky, alebo `NIE`, ak sa nedajú.

3.1 Domáce kolo

3.1.1 Interval

Program: `INTERVAL.PAS` alebo `INTERVAL.CPP`

Príklad:
`INTERVAL.IN`
```
3
0830 1045
0900 1400
1400 1611
2
1000 1200
1700 2200
0
```

`INTERVAL.OUT`
```
PODARI
NEPODARI
```

Organizátori programátorskej súťaže si chceli rozdeliť služby počas prípravy súťažných úloh. Pretože každý z nich mal aj iné povinnosti dohodli sa, že každý napíše časový interval, kedy môže (intervalov môže byť i viac). Interval zapíše vo formáte `HHMM`, kde `HH` je hodina od `00` po `23` a `MM` sú minúty od `00` po `59`. Treba zistiť, či sa im podarí zabezpečiť súvislú službu od príchodu prvého až po odchod posledného z organizátorov. Môžete predpokladať, že všetky intervaly sú v rámci jedného dňa (t.j. od `0000` po `2359`).

Špecifikácia vstupu:
Vstupný súbor obsahuje niekoľko vstupných sád. Každá sada začína riadkom obsahujúcim počet intervalov N v nej, $N < 10000$. V nasledujúcich riadkoch budú zapísané korektné intervaly, každý v samostatnom riadku vo formáte `HHMM HHMM`. Za poslednou sadou bude riadok obsahujúci `0`.

Špecifikácia výstupu:
Výstupný súbor bude obsahovať pre každú vstupnú sadu, v rovnakom poradí, jeden riadok `PODARI` alebo `NEPODARI` podľa toho, či sa organizátorom podarí alebo nepodarí súvislú službu zabezpečiť.

3.1.2 Šifra

Program: `ŠIFRA.PAS` alebo `ŠIFRA.CPP`

Príklad:
`SIFRA.IN`
```
2
.xxx
xxxx
xxxx
xxxx
2
x.x.
xxxx
x.x.
xxxx
0
```

`SIFRA.OUT`
```
NIE
ANO
```

Iste poznáte metódu šifrovania pomocou štvorcovej mriežky rozmeru $2n \times 2n$. Mriežka má niektoré štvorčeky vystrihnuté. V každej zo štyroch polôh, ktoré vieme získať otočením mriežky o 90°, do políčok pod vystrihnutými štvorčekmi zapíšeme znaky správy. Mriežka je korektná, keď pomocou nej do toho istého políčka pod mriežkou môžeme zapísať viackrát iba keď je mriežka rovnako otočená. Mriežka je úplná, keď vieme zapísať do každého políčka pod ňou. Vašou úlohou bude napísať program, ktorý zistí, či zadaná mriežka je korektná a súčasne úplná (prakticky použiteľná).

Špecifikácia vstupu:
Vstupný súbor obsahuje niekoľko mriežok. Zadanie každej mriežky má v prvom riadku celé číslo n (polovica dĺžky strany), v nasledujúcich $2n$ riadkoch je popísaná mriežka. Každý riadok obsahuje $2n$ znakov '`.`' a '`x`'. Znak '`.`' predstavuje vystrihnuté a '`x`' nevystrihnuté miesto. Vstupný súbor je ukončený riadkom obsahujúcim `0`.

Špecifikácia výstupu:
Výstupný súbor obsahuje pre každú mriežku zo vstupného súboru (v rovnakom poradí) v samostanom riadku `ANO` ak je zodpovedajúca mriežka korektná a úplná a `NIE` v opačnom prípade.

3.1.3 Lístky

Program: `LISTKY.PAS` alebo `LISTKY.CPP`

Janko a Marienka sa vybrali MHD do prírody, aby si zabežkovali. Pretože nemali električenky, cvikli si po nastúpení do električky lístky. Janko pohľadom skontroloval predierkovaný lístok a spýtal sa Marienky:

"Počúvaj, keby Ti povedali, ktoré diery robí strojček, vedela by si rozhodnúť, či sa dá nejaká kombinácia dier vydierovať? Pritom by si mohla lístok vkladať do strojčeka viackrát, otáčať ho a dokonca aj neúplne do strojčeka vsunúť". Marienka rozmýšľala, až kým neprišli na konečnú. Aby sa vyhla odpovedi, ktorú nevedela, navrhla Jankovi súťaž, kto bude prvý v cieli ich bežkárskej trasy.

Príklad:
`LISTKY.IN`
```
XOX
XOX
XXX
XOO
XXX
XXX
```
`LISTKY.OUT`
```
NEDA SA
```

Pomôžte Marienke a napíšte program, ktorý úlohu vyrieši. Pripomíname, že lístky majú 3×3 políčok očíslovaných od 1 po 9. Pri správnom vložení do strojčeka je číslo 1 v ľavom hornom políčku a 9 v pravom dolnom políčku.

Špecifikácia vstupu:

Vstupný súbor môže obsahovať niekoľko zadaní. Každé zadanie obsahuje 6 riadkov. V prvých troch riadkoch je zapísaný kód, ktorý dieruje strojček pri správnom zasunutí lístka. V ďalších troch riadkoch je zadaná nejaká kombinácia dier, o ktorej treba zistiť, či ju je možné horeuvedeným spôsobom na danom strojčeku vytvoriť. Každý riadok obsahuje tri znaky. `X` predstavuje miesto bez diery a `O` predstavuje dieru. Za posledným zadaním je jeden prázdny riadok.

Špecifikácia výstupu:

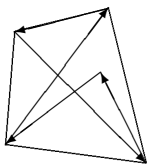
Výstupný súbor obsahuje pre každé zadanie jeden riadok, v rovnakom poradí ako vo vstupnom súbore, v ktorom je `DA SA` ak úloha má riešenie a `NEDA SA` inak.

3.1.4 Robot

Program: `ROBOT.PAS` alebo `ROBOT.CPP`

Príklad:
`ROBOT.IN`
```
1 1
2 1
2 2
1 1
0 0
1 1
2 2
3 1
2 5
1 1
0 0
0 0
```
`ROBOT.OUT`
```
ANO
NIE
```

Pracovníci Výskumného ústavu dráh robotov (VÚDR) riešia úlohu o dráhe najnovšieho robota. Robot sa pohybuje v rovine a len priamo. Po zastavení sa vie otočiť a znovu pohnúť. Jeho dráha je zaznamenená súradnicami bodov, kde sa zastavil a otáčal. Body sú zadané v takom poradí ako cez ne robot prechádzal. O dráhe vieme, že končí a začína v rovnakom bode a neprechádza bodom $[0, 0]$.



Pracovníci VÚDR potrebujú zistiť, či dráha robota tvorí konvexnú čiaru (obrazne povedané: keby sme do bodov kde robot stál a otáčal zatĺkli kolíky a omotali okolo nich niť, tak by sa niť dotýkala všetkých kolíkov; na obrázku je niť vyznačená hrubo). Pomôžte pracovníkom VÚDR a napíšte im program, ktorý bude riešiť ich problém.

Špecifikácia vstupu:

Vo vstupnom súbore môže byť viac vstupných sád. Každá sada je ukončená riadkom obsahujúcim `0 0`. Zvyšné riadky sady obsahujú súradnice po sebe idúcich bodov dráhy robota vo formáte `x y`. Súradnice sú celočíselné, žiadne tri body neležia na jednej priamke. Vstupný súbor ukončuje sada obsahujúca len ukončujúci riadok.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každú vstupnú sadu v rovnakom poradí jeden riadok `ANO` alebo `NIE` podľa toho, či daná sada popisuje konvexnú dráhu robota alebo nie.

3.1.5 Cesty

Program: `CESTY.PAS` alebo `CESTY.CPP`

Príklad:
`CESTY.IN`
```
3 2
1 3 3
2 3
3 1 1 2
4 2
1 4 2 3
2 1
3 1 4
4 3 1
0 0
```

V Softwarelande majú niekoľko dedín pospajáných navzájom cestami. Všetky cesty v každej dedine sa zbiehajú na jedinej križovatke a sú dvojsmerné. V rámci racionalizačných opatrení sa Softwarelandský parlament rozhodol, že ponechá len tie dediny, v ktorých sa zbieha aspoň k ciest. Samozrejme, aby sa čo najmenej narobili, treba zrušiť len nevyhnutný počet dedín. Ked sa zruší dedina, zrušia sa aj všetky cesty, ktoré do nej viedli, čím sa zväčší orná pôda Softwarelandu. Pomôžte Softwarlandskému parlamentu vyriešiť túto neľahkú úlohu. Dediny sú očíslované číslami od 1 po $n < 100$.

Špecifikácia vstupu:

Vo vstupnom súbore môže byť niekoľko sád vstupných údajov. Každá sada začína riadkom obsahujúcim dve celé čísla n a k. V nasledujúcich n riadkoch je popis ciest

medzi dedinami. Na začiatku riadku je číslo mesta a ostané čísla v riadku sú čísla miest, do ktorých vedie z tohoto mesta priama cesta. Vstupný súbor je ukončený riadkom `0 0`.

```
CESTY.OUT
1 3
1 3 4
```

Špecifikácia výstupu:
Výstupný súbor obsahuje pre každé zadanie riadok s číslami miest, ktoré treba nechať. Čísla miest v riadkoch výstupu sú utriedené vzostupne. Ak žiadne mesto neostane, vo výstupnom súbore bude prázdny riadok.

3.2 Finále

3.2.1. Fotograf(B)

Program: `FOTOGRAF.PAS` alebo `FOTOGRAF.CPP`

Príklad:
```
FOTOGRAF.IN
2
20 1
20.5 0.5
23 -1.5
20 -1
20.5 0.5
23 1.5

FOTOGRAF.OUT
1.20
0.00
```

Malý Jožko je vášnivý zberateľ fotografií lokomotív. Aby si rozšíril svoju zbierku, kúpil si nový fotoaparát. Na zberateľskom trhu sú najviac cenené fotografie zhora a to najmä také, na ktorých sa naraz nachádza viac lokomotív. Vybral sa preto na výšok nad trojkoľajku prechádzajúcu ich dedinou, aby si takúto fotografiu spravil. Ale beda! Už aspoň dvadsaťkrát prechádzali okolo naraz tri lokomotívy, ale Jožko nevedel, kedy stlačiť spúšť, aby sa mu všetky tri lokomotívy zmestili na fotografiu. Už je mu naozaj do plaču a potrebuje vašu pomoc.

Napíšte pre Jožka program, ktorý pre dané pozície troch lokomotív a ich rýchlosti vypočíta, za aký čas budú všetky tri na najkratšom úseku trojkoľajky. Každá z lokomotív sa nachádza na inej kolaji. Pozícia je zadaná ako vzdialenosť v metroch od železničného mostu v Nových Zámkoch (všetky lokomotívy sú na tej istej strane mostu) a rýchlosť je udaná v metroch za sekundu, pričom má kladné znamienko, keď sa lokomotíva vzďaluje od mostu a záporné znamienko, keď sa k mostu približuje. Rozmery lokomotív zanedbávame, teda ich považujeme za body.

Špecifikácia vstupu:
V prvom riadku je vo vstupnom súbore počet vstupov, ktoré sa v súbore nachádzajú. Potom pre každý vstup nasledujú tri riadky, každý pre jednu lokomotívu. Riadok popisujúci lokomotívu obsahuje dve reálne čísla oddelené medzerou, pozíciu a rýchlosť lokomotívy (v tomto poradí).

Špecifikácia výstupu:
Vo výstupnom súbore bude pre každý vstup v samostatnom riadku čas v sekundách, s presnosťou na dve desatinné miesta, za ktorý sa budú všetky tri lokomotívy nachádzať na najkratšom úseku trojkoľajky. Keď existuje viacej riešení, vypíšte minimálne. Riadok nesmie obsahovať žiadne nadbytočné medzery.

3.2.2. Kontrola(A)

Program: `KONTROLA.PAS` alebo `KONTROLA.CPP`

Príklad:
```
KONTROLA.IN
10 2 2 1 3 1 2 2 3 2 2
10 1 5 1 3 1 2 2 3 2 2
0

KONTROLA.OUT
2
NEEXISTUJE
```

Výstupná kontrola v továrni na kolíky má za úlohu zistiť dĺžku kolíkov vychádzajúcich z výrobnej linky. Každý kolík odmerajú a zapíšu si jeho dĺžku. Na konci smeny majster vymyslel, že potrebuje vedieť, či nadpolovičná väčšina kolíkov, ktoré sa počas smeny vyrobili má rovnakú dĺžku a aj chce vedieť, aká táto dĺžka je.

Napíšte program, ktorý zistí, či sa v danej postupnosti n celých čísiel nachádza niektoré viac než $n/2$ ráz, ak áno, treba ho aj určiť.

Špecifikácia vstupu:
Vstupný súbor obsahuje niekoľko postupností celých čísiel, každú v samostatnom riadku. Prvé číslo v riadku je N, $N \leq 10000$ a za ním nasleduje N celých čísiel, `int` resp. `integer`, oddelených medzerami. Vstupný súbor je ukončený riadkom, v ktorom $N = 0$.

Špecifikácia výstupu:
Výstupný súbor obsahuje pre každý riadok vstupného súboru v samostatnom riadku hľadané číslo, alebo text `NEEXISTUJE`, ak sa v príslušnom riadku hľadané číslo nenachádza.

3.2.3. Zlatokop(C)

Program: ZLATOKOP.PAS alebo ZLATOKOP.CPP

Príklad:
```
ZLATOKOP.IN
2
1 2 3 2 1 5
2 1 4 1 4 3
2 1 2 3 4 3
3 3 5 3 3 6

ZLATOKOP.OUT
ano
nie
```

Zlatokopovia na Klondike sú už naozaj riadne nahnevaní. Včera sa totiž v krčme znovu strhla hádka, pretože dva pozemky sa prekrývali. Príčinou je prílev veľkého množstva nových zlatokopov, ktorých požiadavky na pozemky pozemkový úrad nezvláda vybavovať. Najťažšia vec, ktorú na úrade musia robiť, je zisťovanie, či sa dva pozemky prekrývajú, alebo nie. Každý pozemok má tvar obdĺžnika so stranami rovnobežnými s rovníkom a poludníkmi a je zadaný súradnicami troch jeho vrcholov, pričom súradnice sú celočíselné.

Vašou úlohou je napísať program, ktorý bude vždy pre dvojicu obdĺžnikov zisťovať, či sa prekrývajú, pričom dva obdĺžniky považujeme za prekrývajúce sa práve vtedy, keď obsah ich prieniku je nenulový.

Špecifikácia vstupu:

V prvom riadku je vo vstupnom súbore počet vstupov, ktoré sa v súbore nachádzajú. Potom pre každý vstup nasledujú dva riadky, každý pre jeden obdĺžnik. Riadky popisujúce obdĺžnik obsahujú šesť celých čísiel X_1, Y_1, X_2, Y_2, X_3, Y_3 - súradnice troch jeho vrcholov.

Špecifikácia výstupu:

Vo výstupnom súbore bude pre každý vstup v samostatnom riadku výstup `ano`, alebo `nie` podľa toho, či sa obdĺžniky prekrývajú, alebo nie.

3.2.4. Lampy(D)

Program: LAMPY.PAS alebo LAMPY.CPP

Príklad:
```
LAMPY.IN
2
5
1A
1C
2B
3A
3C

1
1A

LAMPY.OUT
Da sa
Neda sa
```

Vrátnik Jakub má naozaj veľa povinností. Každý večer musí prejsť všetky miestnosti, pozatvárať okná, pozamykať dvere a hlavne zhasnúť všetky svetlá. A to je niekedy naozaj problém. V každej miestnosti je lampa pozostávajúca z deviatich žiaroviek usporiadaných do štvorca 3×3. Stĺpce sú označené písmenami A,B,C (v tomto poradí) a riadky číslami 1,2,3. Každý stĺpec a každý riadok má zvláštny vypínač. Po stlačení niektorého vypínača sa všetky svietiace žiarovky z príslušného riadku alebo stĺpca zhasnú a zhasnuté sa zažnú. V niektorých miestnostiach však nie sú zažaté všetky žiarovky a nech Jakub prepína ako prepína, vždy mu nejaké zostanú svietiť. Napíšte program, ktorý vypíše pre danú miestnosť, či sa v nej dajú zhasnúť všetky žiarovky, alebo nie.

Špecifikácia vstupu:

Vstupný súbor obsahuje v prvom riadku počet miestností. V ďalších riadkoch sa nachádzajú popisy jednotlivých miestností. Popis miestnosti začína riadkom s počtom zažatých žiaroviek v miestnosti, v každom ďalšom riadku sú súradnice jednej zažatej žiarovky. Údaje o jednotlivých miestnostiach sú oddelené prázdnym riadkom.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každú miestnosť zo vstupného súboru v zvláštnom riadku správu `Da sa`, ak sa dajú vypnúť všetky žiarovky v miestnosti alebo správu `Neda sa`, ak to nie je možné.

3.2.5. Mesto(E)

Program: MESTO.PAS alebo MESTO.CPP

Mesto Rekešvár má veľmi rozvinutý systém mestskej hromadnej dopravy. V Rekešvári je N križovatiek ($N \leq 100$) a na každej križovatke je zastávka autobusu. Každá autobusová linka má dve konečné stanice a premáva pravidelne z jednej konečnej stanice do druhej po pevne stanovenej trase, pričom po ceste stojí na každej zastávke. Žiadna linka nemá na svojej trase viackrát tú istú zastávku. Je známe, že v Rekešvári sa možno dostať na konečný počet prestupov z ľubovoľnej križovatky na ľubovoľnú inú.

V rámci nového programu ľavých rúk poslanci mestského zastupiteľstva schválili takéto rozhodnutie:

§1: Počnúc pondelkom sa zriaďuje okružná vyhliadková linka, t.j. linka, ktorá začína a končí na tej istej zastávke (a neprechádza počas jednej jazdy okrem konečnej žiadnou zastávkou dvakrát) a obsahuje aspoň tri zastávky.

§2: Táto linka zastavuje na každej zastávke, okolo ktorej prechádza.

§3: Keďže všetky párne čísla sú nešťastné, musí mať linka nepárny počet zastávok.

Páni z radnice boli so svojim rozhodnutím veľmi spokojní a preto išli oslavovať do radničného bufetu. Neuvedomili si však, že v Rekešvári sa nemusí dať takáto linka zriadiť. To by prirodzene dopravný podnik priviedlo ku krachu a poslancom by hrozili predčasné voľby.

Príklad:

```
MESTO.IN
4
1 2
1 3
1 4
2 3
2 4
3 4
-1
6
1 3
3 5
5 2
2 4
4 6
6 1
-1
-2

MESTO.OUT
Dobre rozhodnutie
Poslancom hrozia predcasne volby
```

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko blokov vstupných dát. Každý blok obsahuje v prvom riadku celé číslo n, ktoré udáva počet križovatiek v Rekešvári. Každá križovatka má jednoznačne priradené číslo z rozsahu $1, \ldots, n$. Ďalšie riadky obsahujú dvojice celých čísel i a j, $1 \leq i, j \leq n$, pričom platí, že križovatky s číslami i a j sú spojené ulicou. Každý blok je ukončený samostatným riadkom s číslom `-1`. Vstupný súbor je ukončený samostatným riadkom s číslom `-2`.

Špecifikácia výstupu:

Výstupný súbor musí obsahovať pre každý blok dát samostatný riadok so správou `Dobre rozhodnutie`, ak je možné vyhliadkovú linku zriadiť, alebo správou `Poslancom hrozia predcasne volby`, ak nemožno zaviesť novú vyhliadkovú linku podľa rozhodnutia.

3.2.6. O Kocúrkovskom Čase(G)

Program: `PREVODY.PAS` alebo `PREVODY.CPP`

Keď sa Kocúrkovčanom zasekol strojček v slnečných hodinách, rozhodli sa nikdy viac nemerať čas podľa slnka, ale urobiť si vlastný – Čas. Do prostriedku námestia privliekli stovky obrovských ozubených kolies a zoradili ich tak, aby každé susedné dve do seba zapadali. Na každom z nich potom vyznačili červenou farbou jeden zub. K prvému kolesu zapriahli obecného vola, ktorý ním otáčal a tým sa otáčali aj všetky ostatné kolesá. Keď niekto chcel vedieť presný čas, stačilo zájsť na námestie a pozrieť sa, v akej polohe sú označené zuby na jednotlivých kolesách. Po určitej dobe ale Kocúrkovčania s hrôzou zistili, že Čas sa zacyklil (t.j. že natočenie označených zubov je rovnaké ako na začiatku) a tak rýchlo uviedli všetko v mestečku do takého stavu, ako to bolo pred začiatkom času a začali všetko robiť odznova tak, ako im to diktoval Čas. Ako dlho trvá Kocúrkovský Čas (od naštartovania vola po prvé zacyklenie), keď vieme, že obecný vôl otáča prvým kolesom rýchosťou jeden zub za sekundu?

Príklad:

```
PREVODY.IN
4
2 21 12
6 1 2 3 4 5 6
9 2 6 24 120 720 8 16 32 64
3 35 21 15

PREVODY.OUT
Zadanie 1:
Cas v Kocurkove trva 84 sekund.

Zadanie 2:
Cas v Kocurkove trva 60 sekund.

Zadanie 3:
Cas v Kocurkove trva 2880 sekund.

Zadanie 4:
Cas v Kocurkove trva 105 sekund.
```

Špecifikácia vstupu:

V prvom riadku je vo vstupnom súbore počet vstupov, ktoré sa v súbore nachádzajú. Potom pre každý vstup nasleduje jeden riadok popisujúci kolesá. Prvé číslo je z rozsahu 1 až 60000 a udáva počet kolies. Potom nasleduje pre každé koleso počet zubov.

Špecifikácia výstupu:

Vo výstupnom súbore budú pre každý vstup tri riadky. V prvom riadku bude číslo zadania (dodržujte formát z príkladu), v druhom dĺžka Kocúrkovského Času v sekundách a tretí bude prázdny ako oddeľovač.

Môžete predpokladať, že Čas v sekundách bude kratší ako 60000.

3.2.7. O slimákovi(H)

Program: `SLIMAK.PAS` alebo `SLIMAK.CPP`

Vladko dostal raz na narodeniny malé slímä. Kúpil preňho akvárium a vzorne sa oňho staral, až z neho po čase vyrástol poriadny slimák. Vladko pravidelne sledoval, ako slimák rastie a robil si fotografie. Pretože slimák sa najradšej prechádzal po dne, na fotografiách bol vždy zachytený, ako uháňa doprava alebo doľava. Vladko si všimol, že každý mesiac slimákovi dorastie jeden závit ulity a pretože fotografií už bolo veľmi veľa, rozhodol sa odložiť iba údaje o veku a smere pohybu slimáka. Teraz by sa chcel kamarátom pochváliť, ako vyzeral jeho slimák a na to potrebuje zrekonštruovať fotografie.

Špecifikácia vstupu:

Každý riadok súboru obsahuje celé číslo a znak, oddelené jednou medzerou. Celé číslo udáva vek slimáka a je z intervalu 1 až 500. Znak je buď `P` alebo `L`, podľa toho, či slimák ide doprava alebo doľava. Posledný riadok súboru obsahuje dvojicu `-1 Q`.

Špecifikácia výstupu:

Pre každé zadanie vypíšte vek a smer slimáka (dodržte formát ako v príklade), jeden prázdny riadok, potom vykreslite slimáka tak, aby prvý stĺpec obrázku neobsahoval medzery. Každý slimák je ukončený dvomi prázdnymi riadkami.

Príklad:

```
SLIMAK.IN            SLIMAK.OUT
1 L                  Slimak ma 1 mesiacov a ide smerom vlavo  :
2 P
3 L                  H *
-1 Q                 H**


                     Slimak ma 2 mesiacov a ide smerom vpravo  :

                     ****
                     *  *
                     * **
                     *     H
                     ******H


                     Slimak ma 3 mesiacov a ide smerom vlavo  :

                        ********
                        *      *
                        * **** *
                        * *  * *
                        * ** * *
                        *    * *
                        ****** *
                     H         *
                     H**********
```

3.2.8. Papier(F)

Program: `PAPIER.PAS` alebo `PAPIER.CPP`

Bonifác práve zmaturoval (mimochodom na čisté jedničky) a teraz sedí doma za písacím stolom a strašne sa nudí. Na písacom stole, kde sa celý minulý týždeň učil matematiku, zostali ležať nožnice a čistý list štvorčekového papiera.

Bonifác teda zobral štvorčekový papier a začal ho postupne strihať, ale nie hocijako. Vždy si vybral nejaké dva vrcholy štvorčekov, ktoré boli spojené rovnou čiarou predtlače a po tejto čiare papier prestrihol. Ako tak strihal, rozpadal sa mu papier na viac a viac malých častí . . ..

Teraz chudák Bonifác sedí nad hŕbou malých kúskov štvorčekového papiera a ani za svet nemôže spočítať, koľko tých kúskov vlastne je.

Príklad:

```
PAPIER.IN
2 2
0 1 2 1
1 2 1 0

4 5
0 1 2 1
2 4 4 4
2 1 2 2
2 2 0 2

-1

PAPIER.OUT
Pripad 1. Papier sa rozpadne na 4 casti
Pripad 2. Papier sa rozpadne na 2 casti
```

Špecifikácia vstupu:

Vstupný súbor sa skladá z niekoľkých blokov dát. Každý blok dát v prvom riadku obsahuje dve celé čísla m a n ($1 \leq m, n < 80$), kde m je počet štvorčekov nášho papiera v zvislom smere a n vo vodorovnom smere. Ďalších niekoľko riadkov obsahuje dvojice súradníc vrcholov v tvare `a b c d`, kde `a` predstavuje vzdialenosť prvého vrcholu od horného okraja papiera, `b` vzdialenosť prvého vrchola od ľavého okraja, `c` a `d` podobne pre druhý vrchol. Tieto dvojice predstavujú tie dvojice vrcholov, medzi ktorými Bonifác papier prestrihol. Každý blok dát je ukončený prázdnym riadkom. Za posledným blokom dát nasleduje riadok s jediným číslom `-1`.

Špecifikácia výstupu:

Pre každý blok dát má výstupný súbor obsahovať riadok, v ktorom sa nachádza text:

`Pripad` I`. Papier sa rozpadne na` L `casti`

I označuje číslo bloku dát a L je počet častí, na ktoré sa štvorčekový papier rozpadne.

4.1 Domáce kolo

4.1.1 Mašinky

Program: `MASINKA.PAS` alebo `MASINKA.CPP`

Kedysi dávno sa vyrábali veľké mechanické kódovacie mašinky. Kódovacia mašinka vždy fungovala pre abecedu o n znakoch – označme si ich pre jednoduchosť $a_1, a_2, \ldots, a_n$. (Vo všeobecnosti totiž nemusíme používať len našu klasickú abecedu, niekto potrebuje kódovať len čísla, niekto ruštinu, niekto japončinu.)

Mašinka mala v sebe pevne zadefinovaný kód. Kód mašinky je možné zapísať ako n-ticu celých čísel od 1 po n, pričom žiadne číslo sa v kóde nevyskytuje dvakrát. Pri kódovaní potom mašinka postupovala takto: zobrala si písmenko pôvodného textu. Ak bolo toto písmenko a_i, potom namiesto neho napísala znak zodpovedajúci i-temu číslu v kóde. Napríklad ak $n = 5$ a kód mašinky je $(1, 3, 2, 5, 4)$, potom sa text $a_1a_3a_5a_4a_4a_1$ zakódoval ako $a_1a_2a_4a_5a_5a_1$. Keď bolo potrebné zvýšenie bezpečnosti kódovania, zapájali sa dve mašinky za sebou. To znamená, že výsledok práce prvej mašinky sa dal ešte zakódovať druhej mašinke.

Príklad:

```
MASINKA.IN
5
3 1 4 2 5

MASINKA.OUT
2 4 1 3 5
```

Špecifikácia vstupu:

Vo vstupnom súbore je v prvom riadku zadané číslo $n < 10000$ – počet znakov abecedy mašinky a v druhom riadku, medzerami oddelených, n čísel – kód mašinky.

Špecifikácia výstupu:

Výstupný súbor obsahuje kód takej mašinky, že keď zapojíme obe mašinky za sebou, dostaneme výstupe vždy pôvodný text.

4.1.2 Myš

Program: `MYS.PAS` alebo `MYS.CPP`

Myška Píšťalka objavila v mape brlôžtekov tri komôrky s dobrotami. Rada by ich navštívila, ale najradšej

by to urobila tak, aby prešla čo najmenej. Brlôžteky boli očíslované od 1 po n. Z mapy sa dalo zistiť, ktoré dvojice sú spojené chodbou a akou dlhou. Teraz by potrebovala pomôcť nájsť najkratšiu trasu.

Špecifikácia vstupu:

Vstupný súbor `MYS.IN` má v prvom riadku päť čísel n, z, b_1, b_2 a b_3, z je číslo brlôžteka, kde je Píšťalka a b_i sú čísla brlôžtekov, ktoré chce navštíviť (na poradí nezáleží). Každý z nasledujúcich n riadkov obsahuje v_j a niekoľko dvojíc (v_{jk}, d_{jk}), znamenajúcich, že brlôžky v_j a v_{jk} sú spojené chodbou dĺžky d_{jk}.

Špecifikácia výstupu:

Výstupný súbor `MYS.OUT` obsahuje čísla b_1, b_2, b_3 v takom poradí, ako ich má navštíviť, aby prešla čo najmenej.

4.1.3 Počtár

Program: `BOLESLAV.PAS` alebo `BOLESLAV.CPP`

Príklad:
`BOLESLAV.IN`
```
3
9
16
25
```
`BOLESLAV.OUT`
```
2
16
3
```

Boleslav je vášnivý matematik. Jeho najobľúbenejším počtárskym výkonom je sčítavanie niekoľkých za sebou nasledujúcich prirodzených čísiel počnúc nejakým číslom a. Keď takto dostane nejakú druhú mocninu b^2 je veľmi rád a hovorí, že b^2 je vysčítateľné z a. Jeho matematickým snom je, aby mal program, ktorý mu pre dané b^2 nájde najmenšie také a, z ktorého je b^2 vysčítateľné. Splňte Boleslavovi jeho sen.

Špecifikácia vstupu:

Vo vstupnom súbore je v prvom riadku nezáporné číslo $N < 1000$ a v nasledujúcich N riadkoch sa nachádzajú druhé mocniny menšie ako 32767.

Špecifikácia výstupu:

Pre každú z N druhých mocnín vo vstupnom súbore je v samostatnom riadku výstupného súboru najmenšie číslo, z ktorého je daná druhá mocnina vysčítateľná.

4.1.4 Vlaky

Program: `VLAKY.PAS` alebo `VLAKY.CPP`

Príklad:
`VLAKY.IN`
```
2
2
6 2
4 3
3
1 0
3 2
5 1
```
`VLAKY.OUT`
```
Vlaky sa nestretnu.
11
```

V Zmrzlinove sa často stávalo, že sa dva vlaky zrazili. Preto sa Zmzlinovská vláda rozhodla vybudovať úplne novú sieť železničných tratí. Každému vlaku postavili jeho vlastnú uzavretú trať. Jediné miesto, kde sa tieto trate stretávali, bola hlavná stanica Zmrzlinova. Na hlavnej stanici vybudovali dôkladnú sieť semaforov, výhybiek a nadjazdov tak, aby nedošlo k zrážke ani v prípade, že všetky vlaky dorazia na hlavnú stanicu naraz. V deň uvedenia trate do prevádzky umiestnili každý vlak niekam na jeho trať a naraz sa to celé spustilo. Odvtedy každý vlak premáva dookola po svojej trati, pričom jeden okruh mu vždy trvá rovnako dlho. Vláda Zmrzlinova bola nadšená svojim múdrym rozhodnutím a nariadila usporiadať veľkolepú oslavu prvého stretnutia všetkých vlakov na hlavnej stanici. Teraz potrebujú vedieť, kedy tento okamih nastane.

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko blokov. Prvý riadok súboru obsahuje číslo udávajúce počet blokov. Každý blok popisuje jednu železničnú sieť. Prvý riadok bloku obsahuje počet železničných tratí N, $0 < N < 100$. Každý z nasledujúcich N riadkov popisuje jednu trať siete. Obsahuje dve čísla M a A oddelené medzerou, $0 \leq A < M < 1000$, kde M je doba, za ktorú vlak obíde túto trať a A čas, ktorý uplynul medzi uvedením siete do prevádzky a prvým príchodom vlaku na hlavnú stanicu.

Špecifikácia výstupu:

Výstupný súbor má pre každý blok vstupného súboru obsahovať v zvláštnom riadku čas T, ktorý uplynie medzi uvedením siete do prevádzky a prvým stretnutím všetkých vlakov na hlavnej stanici. V prípade, že sa všetky vlaky nikdy nestretnú na hlavnej stanici, vypíšte do tohoto riadku vetu `Vlaky sa nestretnu.`

4.1.5 Stavbári

Program: `STAVBARI.PAS` alebo `STAVBARI.CPP`

V Nalomenej Trieske sa rozhodli postaviť nové námestie, na ktorom budú stáť domy rôznych tvarov. Keďže Nalomená Trieska stojí v úzkej doline, do ktorej by sa stavebné stroje nedostali, časti domov sa najprv

poskladajú inde, potom sa vrtuľníkmi dopravia nad Nalomenú Triesku a tam sa spustia na zem. Potom sa už iba pospájajú a pripevnia k zemi a zavedie sa do nich vodovod. Tu je ale problém, lebo na to, aby sa voda dostala do vysokých poschodí, je nevyhnutný veľký tlak v potrubí. Obyvatelia Nalomenej Triesky teda potrebujú vedieť, aké vysoké budú domy, aby mohli urobiť dostatočne pevné potrubie.

Príklad:

`STAVBARI.IN`
```
11 10
1110111111
1000001001
1011101000
1000101010
1110101010
1000101010
1011101010
1000001001
1000011001
1000001111
0000000000
```

`STAVBARI.OUT`
```
Vyska mesta je 10.
```

Špecifikácia vstupu:

Vstupný súbor predstavuje pohľad zboku na časti domov zavesené na vrtuľníkoch. Obsahuje v prvom riadku dve celé čísla M, N, $1 \leq M, N \leq 100$. Každý z nasledujúcich M riadkov obsahuje N núl a jednotiek. Súvislé plochy jednotiek (uvažujeme 4 susedov) predstavujú časti domov a žiadne dve plochy sa nedotýkajú. Posledný riadok obsahuje samé nuly. Pri spúšťaní sa každá časť pohybuje nezávisle smerom nadol, pokiaľ je to možné (t.j. je pod ňou voľné políčko).

Špecifikácia výstupu:

Napíšte program, ktorý zistí výšku v najvyššieho bodu mesta nad povrchom po zmontovaní domov a do výstupného súboru vypíše `Vyska mesta je` v`.`

4.2 Finále

4.2.1. O zaujímavom trojuholníku(A)

Program: `PASCAL.PAS` alebo `PASCAL.CPP`

Boleslav má veľmi rád trojuholníky a úplne najradšej má Pascalov trojuholník. Ako iste viete, Pascalov trojuholník pozostáva z riadkov očíslovaných číslami $0, 1, 2 \ldots$ V n-tom riadku Pascalovho trojuholníka sa nachádzajú čísla $\binom{n}{0}$, $\binom{n}{1}$, $\ldots$, $\binom{n}{n}$ (v tomto poradí).

Boleslav veľmi rád rôzne experimentuje s Pascalovým trojuholníkom. Už ho však nebaví stále prerátavať kombinačné čísla, ktoré sú často veľmi veľké. Teraz by rád vedel, ktoré kombinačné čísla sú párne a ktoré nepárne. Preto by potreboval program, ktorý by to vyrátal namiesto neho. Poznámka: kombinačné číslo $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ a pre $n, k > 0$ platí $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$; $0! = 1$ a pre $n > 0$ je $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$.

Príklad:

`PASCAL.IN`
```
6
1
0
```

`PASCAL.OUT`
```
*
**
*.*
****
*...*
**..**
*
```

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko riadkov. V každom riadku sa nachádza celé číslo n, $0 < n < 100$. Toto číslo určuje počet riadkov Pascalovho trojuholníka, ktoré je treba zobraziť. Posledný riadok obsahuje číslo 0.

Špecifikácia výstupu:

Do výstupného súboru zapíšte pre každý riadok vstupného súboru (okrem posledného) prvých n riadkov Pascalovho trojuholníka (vrátane nultého riadku), pričom namiesto každého kombinačného čísla vypíšte znak `*`, ak je toto kombinačné číslo nepárne a znak `.` (bodka), ak je toto číslo párne. Každý riadok Pascalovho trojuholníka vypisujte do nového riadku výstupného súboru. Do výstupného súboru nevypisujte žiadne medzery a prázdne riadky.

4.2.2. Kôň(B)

Program: `KON.PAS` alebo `KON.CPP`

Šachista Kubo sa trápi s riešením známeho hlavolamu: ako preskákať so šachovým koňom všetky políčka šachovnice, pričom na každé políčko môžeme prísť iba jeden raz. Zaujímalo ho riešenie pre šachovnicu rozmerov $m \times n$. Úloha sa ukázala nad jeho sily, ale pomohol mu kamarát, ktorý mu napísal program riešiaci túto

úlohu. Kubo nedával na hodinách programovania veľmi pozor, a preto teraz nevie posúdiť, či program dáva dobré výsledky. Pomôžte mu a napíšte mu program, ktorý bude kontrolovať, či zadaná postupnosť skokov koňa popisuje trasu, ktorá vedie cez všetky políčka danej šachovnice.

Špecifikácia vstupu:

Vstup sa skladá z niekoľkých blokov. Každý blok má v prvom riadku dve kladné čísla m a n, $0 < m, n < 100$, kde m je počet riadkov a n počet stĺpcov šachovnice. Druhý riadok bloku obsahuje dve kladné čísla p a r, $0 < p \leq m$, $0 < r \leq n$, kde p je riadok a r je stĺpec, odkiaľ začína kôň skákať. Nasledujúcich niekoľko riadkov obsahuje niekoľko dvojíc kladných čísiel x_i a y_i, $0 < x_i \leq m$, $0 < y_i \leq n$, predstavujúcich po sebe idúce políčka, ktoré kôň navštívil, x_i je riadok a y_i je stĺpec políčka. Posledná dvojica každej postupnosti skokov koňa je vždy $0, 0$. Posledný blok má jediný riadok `0 0`. Čísla v každom riadku sú oddelené jednou medzerou.

Špecifikácia výstupu:

Výstup obsahuje pre každý blok vo vstupnom súbore jeden riadok. Ak blok popisuje:

- nekorektnú postupnosť, t.j. postupnosť skokov koňa nezodpovedá šachovým pravidlám, výstup bude obsahovať riadok `Nekorektna` a b, kde a, b sú riadok a stĺpec políčka, na ktoré sa nedá pokračovať bez porušenia pravidiel, vždy prvé také vo vstupnej postupnosti,
- korektnú postupnosť skokov koňa podľa šachových pravidiel, ale takú, že dvakrát skočí na to isté políčko, bude vo výstupnom súbore riadok `Dvakrat` a b, kde a, b sú riadok a stĺpec prvého políčka, ktoré sa v postupnosti zopakuje (pozrite si príklad),
- korektnú postupnosť podľa šachových pravidiel, ale kôň nenavštívi všetky políčka zadanej šachovnice. Tie ktoré navštívi, navštívi iba raz. V tomto prípade bude výstupný riadok `Neuplna`,
- korektnú postupnosť skokov podľa šachových pravidiel a na každé políčko danej šachovnice skočí práve raz, riadok vo výstupnom súbore bude `OK`.

Poslednému bloku zodpovedá prázdny riadok. Kôň na šachovnici skáče do "L" rozmeru 2×3 políčok, ktoré môže byť otočené alebo aj zrkadlovo obrátené.

Príklad:

```
KON.IN
3 3
1 1
3 2 1 3 2 1 3 3 1 2
3 1 2 3 0 0
4 3
3 2
1 3 2 1 4 2
2 3 1 1 2 3 4 2
0 0
3 3
1 1
2 3 3 1 1 2 3 1 3 3 2 1 0 0
6 6
6 1
5 3 6 5 4 6 2 5 1 3 2 1 3 3 4 1 6 2 5 4 6 6 4 5 2 6 1 4 2 2 4 3 3 5
1 6 2 4 1 2 3 1 5 2 6 4 5 6 4 4 3 6 1 5 2 3 1 1 3 2 5 1 6 3 5 5 3 4 4 2 0 0
0 0

KON.OUT
Neuplna
Dvakrat 2 3
Nekorektna 3 3
OK
```

4.2.3. Krížové odkazy(G)

Program: `ODKAZY.PAS` alebo `ODKAZY.CPP`

Sem tam sa vyskytne potreba urobiť si tabuľku krížových odkazov napríklad identifikátorov mien premenných, funkcií alebo procedúr v programe. V tejto úlohe bude cieľom napísať generátor krížových odkazov pre veľmi zjednodušené `*.RC` súbory.

Príklad:

```
ODKAZY.IN                ODKAZY.OUT
okno_3 106               100
okno_1 100
tlacidloOK 300           105
tlacidloZrus 301
nadpis 200               106
edit1 201
okno_2 105               200
tlacidloAno 303          100 106
tlacidloNie 302          201
skupinaTlacidiel 250     100
sprava 202               202
                         105
okno_3                   250
TlacidloOk               105 106
SkupinaTlacidiel         300
Nadpis                   106
                         301
okno_1                   100 105
tlacidloAno              302
tlacidloNie              100 105
tlacidloZrus             303
edit1                    100 105
nadpis

okno_2
tlacidloAno
tlacidloNie
tlacidloZrus
skupinatlacidiel
sprava
```

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko častí (aspoň dve) navzájom oddelených jedným prázdnym riadkom.

Prvá časť obsahuje v každom riadku dvojicu *identifikátor číslo*. *Identifikátor* je reťazec zložený z veľkých a malých písmen, číslic a podčiarovníka, dĺžky najviac 100 znakov. *Číslo* je kladné číslo menšie než 32768. Dvojica určuje, že daný *identifikátor* má hodnotu *číslo*. Veľké a malé písmená sa v identifikátoroch nerozlišujú (`ab` je to isté čo `aB`). Žiadny identifikátor nemá dve hodnoty. Identifikátorov je najviac 1000.

Zvyšné časti obsahujú vždy niekoľko riadkov (aspoň dva). Každý riadok obsahuje jeden identifikátor. Ich interpretácia je nasledovná. Identifikátor v prvom riadku je meno okna a identifikátory vo zvyšných riadkoch sú mená komponentov použitých v okne. Meno okna sa nevyskytuje ako komponent v žiadnom inom okne a žiadne okno sa nevyskytuje viac než raz. Zato komponenty môžu byť použité vo viacerých oknách. Vstup je zadaný korektne.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každý identifikátor prehľad okien, v ktorých sa vyskytuje. Namiesto všetkých mien sú vo výstupnom súbore uvedené len hodnoty identifikátorov. Pre každý identifikátor, v poradí podľa rastúcej hodnoty identifikátora, obsahuje výstupný súbor dva riadky. V prvom je hodnota identifikátora. V druhom je zoznam čísiel okien, v ktorých sa vyskytuje. Čísla sú oddelené jednou medzerou a usporiadané neklesajúco. Identifikátor okna bude mať druhý riadok prázdny.

4.2.4. O Bonifácových číslach(D)

Program: `CISLA.PAS` alebo `CISLA.CPP`

Príklad:

```
CISLA.IN
3
1009
1010
1938

CISLA.OUT
ANO
NIE
ANO
```

Bonifác včera prišiel do školy, napísal na tabuľu dlhočizné číslo a povedal: "Toto číslo som dostal, keď som k číslu A neobsahujúcemu číslice 0 pričítal číslo vzniknuvšie z A otočením!" Anička to nevydržala a spýtala sa ho "Čo je to otočenie čísla?". I Bonifác jej odpovedal: "Otočením čísla $a_1a_2...a_m$ je číslo $a_ma_{m-1}...a_1$". Anička by teraz chcela overiť, či Bonifác naozaj mohol číslo napísané na tabuli získať tak ako povedal.

Špecifikácia vstupu:

Vo vstupnom súbore je v prvom riadku počet zadaní N a v každom z nasledujúcich N riadkov sa nachádza jedno číslo, ktoré má najviac 100 cifier.

Špecifikácia výstupu:

Pre každé číslo zo vstupného súboru vypíšte do výstupného súboru `ANO`, keď ho mohol Bonifác získať sčítaním nejakého čísla A neobsahujúceho nuly a čísla vzniknuvšieho z A

otočením. Keď Bonifác nemohol číslo získať popísaným spôsobom vypíšte do výstupného súboru `NIE`.

4.2.5. O námestí(E)

Program: `NAMESTIE.PAS` alebo `NAMESTIE.CPP`

V Kocúrkove postavili nové obdĺžnikové námestie vydláždené kachličkami $1m \times 1m$. Aby bolo zaujímavejšie, jednu kachličku vybrali a na jej miesto postavili sochu kocúra. Starosta sa postavil na jednu kachličku a teraz rozmýšľa: "Dá sa prejsť po námestí tak, aby som na každú kachličku stúpil práve raz a aby som skončil na kachličke, z ktorej som vyšiel?" A keďže je to úloha ťažká, zadal ju svojim programátorom.

Príklad:

```
NAMESTIE.IN
2
5 3
.....
.K.S.
.....
5 3
.....
K..S.
.....

NAMESTIE.OUT
ANO
NIE
```

Špecifikácia vstupu:

Vo vstupnom súbore je v prvom riadku počet zadaní N. Potom nasleduje N zadaní. Zadanie začína riadkom obsahujúcim šírku a dĺžku námestia. Plocha námestia nie je väčšia ako $30000m^2$. V ďalších riadkoch nasleduje nákres námestia, kde bodky predstavujú kachličky, písmeno K kocúra a písmeno S miesto, kde stojí starosta. V jednom kroku môže starosta prejsť iba na kachličku, ktorá hranou susedí s kachličkou, na ktorej sa práve nachádza. Na kocúra sa nedá stúpiť.

Špecifikácia výstupu:

Vašou úlohou je napísať program, ktorý zistí, či starosta môže prejsť po námestí tak, aby na každú kachličku stúpil iba raz a vrátil sa na kachličku, z ktorej vyšiel. Pre každý vstup vypíšte do výstupného súboru `ANO`, ak je takáto prechádzka možná, v opačnom prípade vypíšte do výstupného súboru `NIE`.

4.2.6. O optickom prístroji(F)

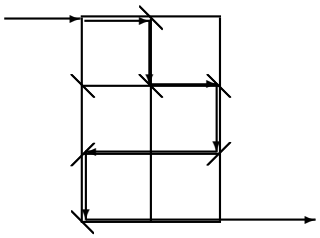
Program: `ZRKADLA.PAS` alebo `ZRKADLA.CPP`

Príklad:

```
ZRKADLA.IN
4 3
0 1 0
1 1 1
1 0 1
1 0 0
-1 -1

ZRKADLA.OUT

Vstup cislo 1: DA SA.
```



Optický prístroj sa skladá z obdĺžnikovej krabičky, v ktorej je $M \times N$ dierok. V niektorých dierkach sú umiestnené malé zrkadlá, ktoré môžu zvierať s osami krabičky uhol 45° resp. 135°. Ľavý horný roh krabičky má súradnice $(1, 1)$, pravý dolný (M, N). V ľavom hornom rohu vstupuje do krabičky vodorovný lúč, ktorý sa postupne odráža od zrkadiel. Cieľom je nastaviť zrkadlá tak, aby v pravom dolnom rohu vychádzal vodorovný lúč z krabičky. Zrkadlá sú obojstranné (lúč sa môže odrážať z oboch strán) a nepriepustné.

Špecifikácia vstupu:

Vo vstupnom súbore je niekoľko zadaní. Každé zadanie obsahuje v prvom riadku rozmery krabičky $1 \leq M \leq 6$, $1 \leq N \leq 6$ nasledované mapou krabičky: 0 znamená prázdnu dierku, 1 zrkadlo. Súbor je ukončený dvojicou $-1, -1$.

Špecifikácia výstupu:

Výstup obsahuje pre každý vstup riadok:

`Vstup cislo` i`: DA SA.`

ak sa dajú zrkadlá otočiť podľa zadania, prípadne

`Vstup cislo` i`: NEDA SA.`

inak.

4.2.7. Tucty(C)

Program: `TUCTY.PAS` alebo `TUCTY.CPP`

Naši predkovia kedysi nepočítali takým spôsobom ako my, ale na rôzne, dnes pre nás podivné, jednotky. Takýmito jednotkami boli napríklad tucet (to znamená dvanásť), kopa (šesťdesiat) alebo veletucet (to znamená tucet tuctov).

Dnes je niekedy problém porozumieť starším ľuďom, ktorí ešte stále tieto jednotky používajú. Preto vašou úlohou bude napísať program, ktorý prevedie takéto číslo do obyčajného číselného desiatkového zápisu.

- Každé ⟨*číslo*⟩ sa môže skladať z niekoľkých ⟨*malých čísel*⟩, ktoré sú od seba oddelené spojkou `A`; hodnota čísla sa potom získa tak, že sa tieto čísla sčítajú.
- ⟨*Malé číslo*⟩ môže byť `JEDEN`, `PAR` (to znamená dva), ďalej to môže byť nejaká ⟨*skupinová číslovka*⟩, alebo môže byť tvaru ⟨*zlomok*⟩ ⟨*skupinová číslovka*⟩ (v tomto prípade je hodnotou príslušná časť skupinovej číslovky, pričom možno predpokladať, že to je celé číslo) alebo tvaru ⟨*malé číslo*⟩ ⟨*skupinová číslovka*⟩ (potom je hodnotou hodnota malého čísla násobená hodnotou skupinovej číslovky).
- ⟨*Skupinová číslovka*⟩ je `TUCET`, `KOPA` alebo `VELETUCET`.
- ⟨*Zlomok*⟩ môže byť `STVRT`, `POL`, `TRETINA` alebo `DESIATOK` (ten možno použiť iba v súvislosti s kopou).

Všetky časti slov sa používajú so slovenským skloňovaním (napríklad `TRETINA TUCTA` alebo `TUCET VELETUCTOV`). Tucet sa skloňuje podľa vzoru dub a kopa podľa vzoru žena.

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko riadkov, v každom riadku sa nachádza jedno číslo (viď. vyššie), pričom jednotlivé slová čísla sú oddelené práve jednou medzerou a za posledným slovom čísla nasleduje bodka. Posledný riadok súboru obsahuje iba bodku. Môžete predpokladať, že všetky čísla sú zapísané správne.

Špecifikácia výstupu:

Výstupný súbor má pre každý riadok vstupného súboru (okrem posledného s bodkou) obsahovať jeden riadok, v ktorom je číslo v desiatkovej sústave predstavujúce hodnotu slovne zapísaného čísla zo vstupného súboru. Môžete predpokladať, že hodnota žiadneho čísla zo vstupu neprekročí 32767.

Príklad:

`TUCTY.IN`	`TUCTY.OUT`
`PAR KOP.`	`120`
`STVRT TUCTA TUCTOV A VELETUCET.`	`180`
`JEDEN A JEDEN.`	`2`
`TRETINA VELETUCTA TUCTOV A DESIATOK KOPY KOP A JEDNA KOPA.`	`996`

`.`

4.2.8. Mestá(H)

Program: `MESTA.PAS` alebo `MESTA.CPP`

Príklad:

`MESTA.IN`	`MESTA.OUT`
`3 2`	`2`
`1 3`	`4`
`2 3`	
`7 18`	
`1 5`	
`2 7`	
`3 6`	
`2 5`	
`2 6`	
`4 7`	
`5 6`	
`3 7`	
`4 5`	
`1 3`	
`1 4`	
`4 6`	
`1 6`	
`1 7`	
`2 3`	
`2 4`	
`5 7`	
`6 7`	
`-1`	

V krajine Bororo je n miest pospájaných navzájom m cestami. Cesty vedú priamo medzi jednotlivými mestami a mimo miest sa križujú iba mimoúrovňovo.

František sedel nad mapou krajiny Bororo a z dlhej chvíle na ňu pozeral a delil mestá do rôznych zaujímavých skupín. František vedel, že dve skupiny miest A a B sú úplne poprepájané, ak pre každé mesto x zo skupiny A a y zo skupiny B platí, že x a y sú spojené priamou cestou. Zrazu sa František veľmi potešil, pretože sa mu mestá podarilo rozdeliť do troch skupín, pričom každé dve z týchto skupín boli navzájom úplne poprepájané. Veľmi by ho teraz zaujímalo, na koľko najviac takých skupín môže mestá rozdeliť.

Špecifikácia vstupu:

Vstupný súbor je rozdelený na niekoľko blokov. Každý blok obsahuje popis jednej mapy. V prvom riadku každého bloku sú dve čísla n a m oddelené medzerou ($0 < n \leq 100$, $0 \leq m \leq 10000$). Mestá sú očíslované od 1 po n. Potom nasleduje m riadkov, pričom v každom riadku sú dve čísla a, b oddelené medzerou, ktoré určujú, že mestá a a b sú prepojené priamou cestou. Za posledným blokom súboru je riadok s číslom -1.

Špecifikácia výstupu:

Výstupný súbor pre každý blok vstupného súboru bude obsahovať riadok s jediným číslom, ktoré bude znamenať najväčší počet skupín, na

ktoré možno mestá rozdeliť (každé mesto bude v práve jednej skupine) tak, aby každé dve skupiny boli navzájom úplne poprepájané.

5.1 Domáce kolo

5.1.1 O horolezcovi Pištovi

Program: `HOROLEZ.PAS` alebo `HOROLEZ.CPP`

Príklad:
`HOROLEZ.IN`
```
4 4
4 2 3 6
1 3 4 5
5 5 3 3
3 4 4 6
1 2
```
`HOROLEZ.OUT`
```
ANO
```

Horolezec Pišta sa vybral na výlet do Himalájí. Keď už tam bol, rozhodol sa, že si vylezie na nejaký kopec. A aby ukázal, že sa dokáže pochlapiť, povedal si, že vylezie rovno na ten najvyšší. Problém bol v tom, že nemal so sebou svoj horolezecký výstroj, takže nemohol loziť po príliš strmých stenách. Teraz smutne sedí v tábore, pozerá sa do mapy a hľadá vrch, na ktorý by mohol vyliezť. Pomôžte Pištovi zistiť, či môže vyliezť na najvyšší vrch.

Špecifikácia vstupu:

Vo vstupnom súbore máte čísla $M \leq 200$, $N \leq 200$ (počet riadkov, počet stĺpcov) a maticu $A_{M \times N}$ kladných celých čísel menších než 10000. Číslo v matici A udáva nadmorskú výšku. Ďalej máte dve čísla a, b, ktoré udávajú polohu (riadok a stĺpec) základného tábora. Pišta môže prejsť z jedného políčka na druhé, iba ak susedia hranou a ich výšky sa líšia najviac o 1.

Špecifikácia výstupu:

Do výstupného súboru Zapíšte `ANO`, ak Pišta môže vyjsť na (niektorú) najvyššiu horu a `NIE` inak.

5.1.2 Digitálne korytnačky

Program: `KORYT.PAS` alebo `KORYT.CPP`

Digitálny Gejza sa potrebuje dostať cez širokú digitálnu rieku. Našťastie mu na pomoc prišlo N digitálnych korytnačiek. Rozostavili sa v rieke do radu tak, aby sa dalo preskočiť z brehu na prvú korytnačku, z prvej korytnačky na druhú, atď. až napokon z poslednej korytnačky na druhý breh. Digitálne korytnačky sa v pravidelných intervaloch ponárajú a vynárajú a na ponorenú korytnačku sa nedá doskočiť. Každá korytnačka má iný rytmus, presnejšie i-ta korytnačka je $A[i]$ digitálnych sekúnd nad hladinou, potom $A[i]$ digitálnych sekúnd pod hladinou, atď. Na začiatku sa všetky korytnačky naraz vynoria. Gejza chce vedieť, či sa dá rieka preskočiť tak, že v určitom okamihu od začiatku skočí na prvú korytnačku, hneď na druhú, hneď na tretiu, atď. až napokon sa ocitne na druhom brehu. Vieme, že každý skok trvá jednu sekundu.

Príklad:
`KORYT.IN`
```
5 3 5 2 7 11
4 3 3 3 3
0
```
`KORYT.OUT`
```
ANO
NIE
```

Špecifikácia vstupu:

Vo vstupnom súbore sa v každom riadku nachádza zadanie pre jednu rieku. Na začiatku riadku je počet korytnačiek $N \leq 500$ a nasledujú rytmy korytnačiek $A[1]$, ..., $A[N]$ ($1 \leq A[i] \leq 300$). V poslednom riadku sa nachádza 0.

Špecifikácia výstupu:

Napíšte program, ktorý pre každý riadok vo vstupnom súbore vypíše `ANO` do výstupného súboru ak sa daná rieka dá preskočiť a `NIE` ak sa nedá.

5.1.3 Nové kúzlo

Program: `KUZLO.PAS` alebo `KUZLO.CPP`

Príklad:
`KUZLO.IN`
```
2
5
1 2 3 4 5
4
100 100 7 100
```

Kúzelník Oto vymyslel nové kúzlo. Používa pri ňom kopu kariet, pričom na každej karte je napísané jedno celé číslo. Z obecenstva vyberie náhodného diváka a vyzve ho, aby z kopy kariet odobral niekoľko kariet tak, aby sa poradie ostatných kariet v kope nezmenilo. Potom Oto s nenápadnou pomocou svojho pomocníka Bimbáca vymenuje v

správnom poradí čísla kariet, ktoré zostali na kope, bez toho, aby sa ne pozrel. Bimbác je však veľmi nešikovný a občas sa mu stane, že kúzlo zbabre. Oto vtedy musí hádať a chce vedieť, akú má šancu, že uhádne správne. Potrebuje teda vedieť, koľko navzájom rôznych kôp môže zostať po tom, ako divák odoberie karty.

Špecifikácia vstupu:

Vstupný súbor obsahuje v prvom riadku číslo k určujúce počet kôp kariet, ktoré má Oto pripravené. Potom nasleduje k dvojíc riadkov, pričom každá dvojica opisuje jednu kopu. Prvý riadok z dvojice obsahuje číslo n ($1 \leq n \leq 100$) určujúce počet kariet na kope. V druhom riadku sa nachádzajú jednotlivé čísla kariet v tom poradí, v akom sa nachádzajú na kope.

Špecifikácia výstupu:

Do výstupného súboru pre každú prichystanú kopu vypíšte do jedného riadku počet rôznych kôp, ktoré môžu zostať po tom, ako divák odoberie karty. Zarátajte aj možnosť, že divák odoberie všetky karty alebo že neodoberie žiadnu. Môžete predpokladať, že počet možností neprekročí 2^{30}.

5.1.4 Rozbal

Program: `ROZBAL.PAS` alebo `ROZBAL.CPP`

Príklad:
(hexadecimálne)

```
ROZBAL.IN
00 61 01 FF 62
08 FD 0A 00 0D

ROZBAL.OUT
61 61 62 61 61 62
61 61 62 61 61 0A 0D
```

Iste všetci používate nejaký program na kompresiu údajov. Jeden zo spôsobov, ktorý sa používa, vytvorí kompresovaný súbor obsahujúci len dvojice (d, z), prípadne trojice (d, s, z) celých čísel, kde $0 \leq d, s < n$ a $0 \leq z \leq 255$ je kód znaku. V tomto príklade uvažujeme $n = 256$, t.j. zo zakompresovaného súboru sa čítajú dva alebo tri byty. Pri rekonštruovaní pôvodného súboru zo zakompresovaného si pamätáme n naposledy "rozbalených" znakov, napríklad v poli $R[0..2n-1]$ od $R[0]$ po $R[n-1]$. Na začiatku obsahuje pole R nuly. Ak zo zakompresovaného súboru prečítame $d = 0$, znamená to, že sa jedná o dvojicu a môžeme prečítať z. Ak bolo $d > 0$ musíme prečítať trojicu, t.j. ešte s, z. V oboch prípadoch do poľa R od indexu n pripíšeme $d + 1$ znakov, ktoré tiež vypíšeme aj do výstupného súboru: i-ty z týchto znakov bude $R[s + i - 1]$, pre $i = 1, \ldots, d$ a $d + 1$-vý znak bude mať kód z. Na koniec pole R posunieme o $d + 1$ miest doľava, čím aktualizujeme posledných n "rozbalených" znakov. Takto pokračujeme kým nevyčerpáme celý zakompresovaný súbor.

Napíšte program, ktorý bude čítať korektne zakompresovaný vstupný súbor a rozbalí ho do výstupného súboru. Môžete pritom použiť aj efektívnejší spôsob rozbaľovania ako hore popísaný.

5.1.5 List

Program: `LIST.PAS` alebo `LIST.CPP`

Rubikov list je veľmi zaujímavý hlavolam, ktorý vyzerá ako list papiera $N \times N$ cm ($N \leq 20$) rozdelený na štvorčeky 1×1 cm. Rubikov list je v *základnej polohe* ofarbený na vrchnej strane červenou farbou a na spodnej strane zelenou farbou.

Príklad:

```
LIST.IN       LIST.OUT
3             da sa
ZZC
ZZC
ZCC

LIST.IN       LIST.OUT
4             neda sa
ZCZC
CZCZ
ZCZC
CZCZ
```

Riadková os rubikovho listu je priamka, ktorá prechádza stredom rubikovho listu a rozpoľuje každý jeho riadok. Podobne *stĺpcová os* je priamka, ktorá prechádza stredom rubikovho listu a rozpoľuje každý jeho stĺpec. Na rubikovom liste možno otočiť ľubovoľný riadok o 180° okolo riadkovej osi a ľubovoľný stĺpec o 180°. Napríklad ak v liste uvedenom v prvom príklade otočíme prvý riadok, dostávame prvý riadok `ZCC`, ostatné riadky sa nemenia.

Napíšte program, ktorý pre daný rubikov list zistí, či je ho možné nejakou postupnosťou otočení riadkov a stĺpcov poskladať do základnej polohy.

Špecifikácia vstupu:

Vo vstupnom súbore sa v prvom riadku nachádza číslo N určujúce veľkosť rubikovho listu. Nasledujúcich N riadkov obsahuje popis jedného rubikovho listu. Pre štvorček v riadku i a stĺpci j sa v príslušnom riadku a stĺpci vstupného súboru nachádza buď písmenko `Z` – potom je tento štvorček na vrchnej strane zelený a na spodnej strane červený, alebo písmenko `C` – potom je tento štvorček na vrchnej strane červený a na spodnej strane zelený.

Špecifikácia výstupu:
Do výstupného súboru váš program zapíše riadok `da sa`, ak je možné poskladať rubikov list do základnej polohy a riadok `neda sa`, ak to nie je možné.

5.2 Finále

5.2.1. Olympiáda(A)

Program: `OLYMP.PAS` alebo `OLYMP.CPP`

Príklad:
```
OLYMP.IN
15
Lupsa ROMANIA 200
Mironov RUSSIA 192
Guo CHINA 190
Soviani ROMANIA 188
Elizarov RUSSIA 175
Bargachev RUSSIA 175
Huang CHINA 161
Praun ROMANIA 161
Zhang CHINA 157
Ghenea ROMANIA 142
Lapunov RUSSIA 141
Chai CHINA 136
Calderon COLOMBIA 93
Gil COLOMBIA 52
Hoyos COLOMBIA 41

OLYMP.OUT
ROMANIA 691
RUSSIA 683
CHINA 644
COLOMBIA 186
```

Medzi najvýznamnejšie súťaže v programovaní patrí Medzinárodná olympiáda v informatike. Na tejto súťaži sa z každej krajiny môžu zúčastniť najviac štyria súťažiaci. Je to však súťaž jednotlivcov, preto jediná oficiálna výsledková listina obsahuje iba poradie a počty bodov jednotlivých súťažiacich. Vedúci jednotlivých výprav vždy strávia noc po slávnostnom vyhodnotení tým, že každý z nich podľa výsledkovej listiny sčítava body jednotlivých štátov a zostavuje poradie krajín, aby vedel doma povedať, ako sa jeho krajina celkovo umiestnila. Ušetrite im túto prácu a napíšte program, ktorý poradie krajín zostaví.

Špecifikácia vstupu:
Vstupný súbor obsahuje údaje z oficiálnej výsledkovej listiny. V prvom riadku je počet účastníkov olympiády N ($1 \leq N \leq 400$). V každom z ďalších N riadkov sú údaje o jednom súťažiacom. Každý takýto riadok začína menom súťažiaceho (reťazec obsahujúci iba malé a veľké písmená anglickej abecedy dlhý najviac 20 znakov). Potom nasleduje jedna medzera a názov krajiny (reťazec obsahujúci iba veľké písmená anglickej abecedy dlhý najviac 20 znakov), opäť jedna medzera a počet bodov, ktoré daný súťažiaci získal (celé číslo z rozsahu 0 až 2000). Súťažiaci sú vo vstupnom súbore utriedení podľa počtu získaných bodov.

Špecifikácia výstupu:
Výstupný súbor bude pre každú krajinu, ktorá mala na olympiáde aspoň jedného účastníka, obsahovať jeden riadok obsahujúci názov krajiny a súčet bodov účastníkov z tejto krajiny. Medzi položkami vynechajte práve jednu medzeru. Krajiny vypisujte v poradí podľa súčtu bodov od najlepšej po najhoršiu, krajiny s rovnakým počtom bodov utrieďte podľa abecedy.

5.2.2. Kód(B)

Program: `KOD.PAS` alebo `KOD.CPP`

Príklad:
```
KOD.IN
2 1 3 3
3 3 3 3 3 2 4 4
-1

KOD.OUT
0 2 2
1 1 0
2 3 6
3 3 7

0 3 2
1 3 3
2 3 4
3 3 5
4 3 6
5 2 0
6 4 14
7 4 15
```

Iste ste sa všetci stretli s programami, ktoré kompresujú údaje. Často používanou metódou je nahradenie pevných (zvyčajne 8 bitových) dĺžok kódov znakov premenlivými. Dĺžku kódu v ďalšom texte uvažujeme v bitoch. Znaky, ktoré sa vyskytujú viac ráz, budú mať kratšie kódy, než znaky, ktoré sa vyskytujú zriedkavejšie. Aby sa dal takto zakódovaný súbor jednoznačne dekódovať (napríklad do ASCII kódovania) nesmie byť kód žiadneho znaku zhodný s počiatočnými bitmi kódu iného znaku. Takýto kód nazývame prefixový. Napríklad kód $1, 01, 00$ v dvojkovej sústave je prefixový, ale kód $1, 01, 11$ nie je, lebo 1 je začiatkom kódu 11. Všimnite si, že pri čítaní bitov prefixového kódu, napriek tomu, že kódy majú rôzne dĺžky, vždy vieme, kedy sme dočítali kód jedného znaku.

Pri kompresii sa zvykne vytvárať kód vhodný špeciálne pre daný súbor (závisí od počtu výskytov jednotlivých znakov). Aby sme ho vedeli dekódovať, k zakódovanému súboru musíme ešte pridať aj kódovaciu tabuľku. Ak sa však dohodneme, že prefixový kód bude spĺňať, že

1. ak u a v sú kódy dvoch slov, pričom u je kratší ako v a i je prvá pozícia zľava, na ktorej sa u a v líšia, tak u má na tejto pozícii nulu a v jednotku a
2. všetky kódy s rovnakou dĺžkou majú lexikograficky nasledujúce hodnoty v rovnakom poradí ako znaky, ktoré reprezentujú,

bude nám stačiť namiesto kódovacej tabuľky pridať iba dĺžky kódov pre všetky písmená abecedy a kódovaciu tabuľku si na základe dĺžok vieme skonštruovať.

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko riadkov. V jednom riadku je najviac 256 nezáporných celých čísel predstavujúcich dĺžky kódov v prefixovom kóde. Dĺžky kódov budú aspoň 1 a najviac 30. Kód s dĺžkou na i-tom mieste ($0 \leq i \leq 255$) zodpovedá znaku s kódom i v ASCII kódovaní. Čísla v riadku sú oddelené medzerou. Posledný riadok súboru obsahuje iba číslo -1. Môžete predpokladať, že pre každé zadanie zo vstupného súboru existuje jediný prefixový kód spĺňajúci uvedené podmienky a že vstupný súbor obsahuje aspoň jedno zadanie.

Špecifikácia výstupu:

Riadku vstupného súboru s i číslami, bude vo výstupnom súbore zodpovedať i riadkov s tromi číslami oddelenými jednou medzerou. Za i-tym riadkom bude jeden prázdny riadok. Prvé číslo v riadku bude ASCII kód znaku druhé bude dĺžka prefixového kódu (zo vstupného súboru) a tretie bude desiatkovo zapísaná hodnota prefixového kódu tohoto znaku. Riadky budú usporiadané podľa rastúcich ASCII kódov.

5.2.3. Hra(C)

Program: `HRA.PAS` alebo `HRA.CPP`

Janko a Marienka nemajú čo cez dlhé nudné vyučovacie hodiny robiť a tak si vymysleli novú hru. Najprv si na milimetrovom papieri vyznačia hraciu plochu tvaru obdĺžnika a dohodnú sa na čísle P a na počte ťahov N. Potom na hracej ploche vyfarbujú obdĺžniky, každý hráč svojou farbou, podľa týchto pravidiel:

Hráč, ktorý je na ťahu, si zvolí jeden štvorček hracej plochy. Potom hráči nájdu na hracej ploche obdĺžnik s najväčšou plochou, ktorý má vo zvolenom štvorčeku ľavý dolný roh a neobsahuje žiadne vyfarbené políčko. Ak je obdĺžnikov s najväčšou plochou viac, ťah je neplatný a hra sa končí prehrou hráča na ťahu. Ťah je tiež neplatný, ak plocha najväčšieho obdĺžnika (v mm^2) presahuje dohodnuté číslo P alebo ak hráč zvolil už vyfarbený štvorček. Ak ťah nebol neplatný, najväčší obdĺžnik sa vyfarbí farbou hráča, ktorý je na ťahu a pokračuje druhý hráč. Hra sa tiež končí, ak každý hráč bol na ťahu N krát. Vyhráva hráč, ktorý pokryl svojimi obdĺžnikmi väčšiu plochu.

Odohrali niekoľko hier, pričom počas nich neodhalili žiaden neplatný ťah. Väčšinu hier však vyhral Janko. Marienku to veľmi hnevá a myslí si, že Janko určite niekde nedodržal pravidlá. Hľadať však chybu na tak veľkom papieri je ťažké, tak to nechá na vás a váš program.

Príklad:

`HRA.IN`

```
3
7 6 4 2
6 5
5 3
4 2
3 2
7 6 4 3
6 5
5 3
4 2
3 2
3 3
6 1
4 4 1 2
3 3
2 2
0 1
1 0
```

`HRA.OUT`

```
Janko
Vyfarbene policko
Viacero obdlznikov
```

Špecifikácia vstupu:

Vstupný súbor v prvom riadku obsahuje číslo K určujúce počet hier, ktoré Janko a Marienka odohrali. Ďalej je pre každú z hier vo vstupnom súbore blok dát. Tento blok obsahuje v prvom riadku celé čísla S, D, P a N, pričom S a D sú šírka a dĺžka hracej plochy (v milimetroch), kde $2 \leq S, D \leq 1000$. Číslo P je dohodnutá plocha ($1 \leq P \leq 10^6$) a N je počet ťahov, ktoré obaja hráči odohrali ($1 \leq N \leq 50$). V každom z ďalších $2N$ riadkov je dvojica celých čísel x a y popisujúca jeden ťah, pričom x je vzdialenosť zvoleného štvorčeka od ľavého okraja hracej plochy (v mm) a y je vzdialenosť od spodného okraja. Políčko v ľavom dolnom rohu má teda súradnice $(0, 0)$ a v pravom hornom $(S-1, D-1)$. Prvá na ťahu je vždy Marienka.

Špecifikácia výstupu:

Výstupný súbor obsahuje pre každú hru zo vstupného súboru jeden riadok. V prípade, že v celej hre nebol žiaden neplatný ťah, program vypíše text `Marienka`, ak vyhrala Marienka, alebo `Janko`, ak vyhral Janko alebo `Remiza`, ak obaja vyfarbili rovnaký počet políčok. V opačnom prípade nájde prvý neplatný ťah a vypíše text

- `Vyfarbene policko`, ak zvolené políčko už bolo v predchádzajúcich ťahoch vyfarbené
- `Viacero obdlznikov`, ak v tomto ťahu existuje viacero rôznych obdĺžnikov s maximálnou plochou
- `Privelka plocha`, ak v tomto ťahu existuje iba jeden najväčší obdĺžnik, ale má plochu väčšiu ako P.

Môžete predpokladať, že ak ťah nie je neplatný, Janko a Marienka správne nájdu a vyznačia obdĺžnik s najväčšou plochou.

5.2.4. Škatule(D)

Program: `SKATULE.PAS` alebo `SKATULE.CPP`

Príklad:
`SKATULE.IN`
```
100
7
6
0
```

`SKATULE.OUT`
```
2
0
1
```

V Kocúrkove majú továreň na výrobu škatúľ. Všetky škatule, ktoré táto továreň vyrába, majú tvar kvádra s hranami celočíselnej dĺžky. Navyše chceli zjednodušiť účtovníctvo, a preto vyrábajú iba také škatule, ktorých povrch je P a teda spotreba kartónu na každú škatuľu je rovnaká. Aby však ich sortiment nebol príliš chudobný, vyrábajú škatule všetkých rozmerov, ktoré vyhovujú týmto podmienkam. No jedného dňa prišli na to, že vlastne nevedia, koľko druhov škatúľ vyrábajú. A tak začali počítať...

Špecifikácia vstupu:

V každom riadku vstupného súboru okrem posledného riadku sa nachádza celé číslo P, $1 \leq P \leq 10^6$. Posledný riadok vstupného súboru obsahuje číslo 0.

Špecifikácia výstupu:

Výstupný súbor bude pre každý riadok zo vstupného súboru (okrem posledného) obsahovať počet navzájom rôznych škatúľ s povrchom P a s celočíselnými hranami. Škatule, ktoré vzniknú len výmenou poradia rozmerov (napr. $1 \times 2 \times 3$ a $3 \times 1 \times 2$), považujeme za rovnaké.

5.2.5. Záhon(F)

Program: `ZAHON.PAS` alebo `ZAHON.CPP`

Príklad:
`ZAHON.IN`
```
6 7
...#...
.####..
#####..
.#####.
.####..
..#....
5 5
.....
####.
.###.
.###.
.....
0 0
```

`ZAHON.OUT`
```
ANO
NIE
```

Bonifác sa stal novým kráľovským záhradníkom v Kocúrkove. Toto mestečko je preslávené novým umeleckým štýlom zvaným štvorizmus. Materiálnou podstatou štvorizmu je štvorčekový papier, ktorý sa používa dokonca aj na maľovanie obrazov. Duchovnou podstatou sú dva tvary ŠIN a ŠON:

```
 #          ###
###         ###
 #          ###

ŠIN         ŠON
```

Obraz namaľovaný štvorizmom sa skladá iba z oblastí tvarov ŠIN a ŠON, pričom každá oblasť je vymaľovaná inou farbou. Panovníkovi sa obrazy veľmi páčia a preto sa rozhodol, že aj jeho záhrada bude pokrytá rôznofarebnými záhonmi, pričom každý záhon bude tvaru ŠIN, alebo ŠON. Predložil Bonifácovi mapu mesta, kde bodky (`.`) znázorňujú cesty a krížiky (`#`) znázorňujú záhradu. Bonifác teraz musí zistiť, či je možné záhradu pokryť záhonmi v štýle štvorizmu.

Špecifikácia vstupu:

Vstupný súbor obsahuje niekoľko zadaní. Každé zadanie obsahuje na prvom riadku dve čísla V a S vyjadrujúce výšku a šírku mapy. Potom nasleduje V riadkov po S znakoch, $3 \leq V, S \leq 200$. Vstupný súbor je ukončený riadkom obsahujúcim `0 0`.

Špecifikácia výstupu:

Urobte program, ktorý pre každé zadanie zo vstupného súboru vypíše do výstupného súboru odpoveď `ANO` ak možno záhradu pokryť záhonmi v štýle štvorizmu a `NIE`, ak to možné nie je.

5.2.6. Slová(E)

Program: `SLOVA.PAS` alebo `SLOVA.CPP`

V redakcii novín Some Times zriadili sklad slov. Slová sú naukladané pekne v policiach a každé je zreteľne označené evidenčným číslom. Keď sa niektorému redaktorovi minie slovná zásoba, požiada skladníka, aby mu vydal nejaké nové slová. Slová v sklade pozostávajú z písmen $a, \ldots, z$ a každé z nich má dĺžku aspoň jedna a

najviac 20. Všetky slová takejto dĺžky, pozostávajúce z písmen $a, \ldots, z$, zoradíme najprv podľa dĺžky a slová s rovnakou dĺžkou zoradíme podľa abecedy. Potom slovo, ktoré je v takomto usporiadaní na i-tom mieste, dostane evidenčné číslo i. Takto priradené čísla sú však pomerne veľké. Skladník musí často vypočítavať pre dané slovo, aké má evidenčné číslo a naopak pre dané číslo, aké slovo mu prináleží. Je to však veľmi únavné a preto by chcel používať počítač.

Špecifikácia vstupu:

Vstupný súbor pozostáva z viacerých riadkov, pričom každý riadok obsahuje buď evidenčné číslo alebo slovo. Vstupný súbor je ukončený riadkom obsahujúcim znak `*`.

Špecifikácia výstupu:

Do výstupného súboru pre každý riadok vstupu obsahujúci evidenčné číslo vypíšte zodpovedajúce slovo a naopak pre každý riadok obsahujúci slovo vypíšte jeho evidenčné číslo. Pre posledný riadok obsahujúci `*` nevypisujte nič.

Príklad:

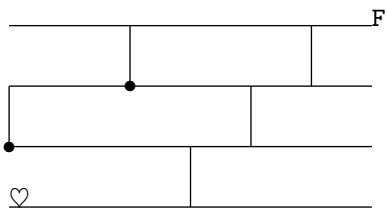
SLOVA.IN	SLOVA.OUT
704	aab
c	3
53	ba
zzzzzzzzzzzzzzzzzzzz	207252748510177851843380527O
*	

5.2.7. Ferdo(H)

Program: `FERDO.PAS` alebo `FERDO.CPP`

Príklad:

```
FERDO.IN
1
4 6 3 1
1 3
2 0* 4
2 2* 5
```




```
KOD.OUT
Zadanie 1: stihne
```

Mravec Ferdo sa rozhodol, že postaví vo svojom mravenisku mrakodrap. Zavolal svojich kamarátov chrobáčikov a za pár dní sa nad mraveniskom týčila vysokánska konštrukcia zo stebiel rákosia, medzi ktorými viedli rebríky z ihličia. Mravcom zo susedného mraveniska sa to však vôbec nepáčilo, a preto sa rozhodli, že Ferdov mrakodrap podpália. To však nevedeli, že Ferdo na začiatok aj koniec každého rebríka namontoval požiarny snímač (bol to priviazaný nejaký malý chrobáčik, ktorý keď zacítil dym, začal strašne kvíliť). Keď sa špióni zo susedného mraveniska dostali až k mrakodrapu, Ferdo sa práve kochal výhľadom na celú lúku, keď zrazu začali niektoré snímače kvíliť (požiar vznikol na viacerých miestach, ale vždy pri dolnom konci rebríka, lebo tam bolo ihličie najsuchšie). Ferdo vedel, ako rýchlo dokáže utekať aj to, ako rýchlo sa šíri požiar a tak sa okamžite rozbehol dolu. Tam už jeho kamaráti uzatvárali stávky, či to stihne. Koľko by ste vsadili vy?

Napíšte program, ktorý načíta tvar mrakodrapu, miesta vzniku požiaru, rýchlosť šírenia požiaru a Ferdovu rýchlosť a vypíše `stihne` keď sa Ferdo môže vyhnúť požiaru a `zhori` keď nie. Mrakodrap má N podlaží očíslovaných 0,...,$N-1$ (podlažie 0 je zem). Podlažia sú vzdialené na výšku jeden Fm (Ferdometer, 1Fm=0.0278...m). Každé podlažie je široké M Fm. Ferdo stojí na najpravejšom okraji najvyššieho podlažia a rád by dobehol na najľavejší okraj prízemia. Požiar sa šíri len po steblách a rebríkoch všetkými smermi rovnako rýchlo. V prípade, že Ferdo dobehne na nejaké miesto v rovnakom okamihu ako oheň, zhorí mu zadoček a umrie.

Špecifikácia vstupu:

Prvý riadok obsahuje počet zadaní. Každé zadanie má v prvom riadku štyri čísla N, M, F, O označujúce počet podlaží ($N \geq 3$), veľkosť podlaží, rýchlosť Ferda a rýchlosť ohňa. Dĺžky sú vo Fm, rýchlosti vo Fm/s. Potom nasleduje $N-1$ riadkov, z ktorých i-ty riadok opisuje rebríky medzi $(i-1)$-vým a i-tym podlažím. Prvé číslo je počet rebríkov (aspoň jeden) a potom nasleduje pre každý rebrík jeho vzdialenosť od ľavého okraja nasledovaná znakom `*`, ak je na dolnom konci tohoto rebríka založený požiar. Rebríky sú usporiadané podľa vzdialenosti. Môžete predpokladať, že počet všetkých rebríkov je najviac 499 a kvôli bezpečnosti nie sú dva rebríky tesne pod sebou.

Špecifikácia výstupu:
Pre každé zadanie vypíšte na zvláštny riadok číslo zadania a slovo `stihne` alebo `zhori`. Dodržte formát z príkladu.

5.2.8. Čísla(G)

Program: `CISLA.PAS` alebo `CISLA.CPP`

Príklad:
`CISLA.IN`
```
2
8 2
17 2 3
```

`CISLA.OUT`
```
ANO
NIE
```

Malý Janko vymyslel novú matematickú hru pre jedného hráča. Hra sa hrá s papierom a perom. Na začiatku je na papieri napísaných niekoľko kladných celých čísel a je určené cieľové kladné celé číslo C. Hra má jediné pravidlo: ak sú na papieri napísané čísla A a B, potom možno na papier pripísať aj číslo $A + B + AB$. Cieľom hráča je dosiahnuť, aby na papieri bolo napísané cieľové číslo C. Janko sa už druhý týždeň pokúša vyriešiť jedno zadanie tejto hry, ale bezúspešne. Napíšte program, ktorý Jankovi pomôže.

Špecifikácia vstupu:
Vo vstupnom súbore je v prvom riadku nezáporné číslo N a v každom z nasledujúcich N riadkov sa nachádza jedno zadanie hry. Každé zadanie začína cieľovým číslom C, za ktorým nasleduje zoznam čísel napísaných na papieri. Váš program by mal pre každé zadanie vypísať do výstupného súboru odpoveď `ANO`, alebo `NIE`, podľa toho, či je možné hru s daným zadaním vyhrať. Čísla nachádzajúce sa v súbore sú menšie ako 10^9.

Špecifikácia výstupu:
Váš program by mal pre každé zadanie vypísať do výstupného súboru odpoveď `ANO`, alebo `NIE`, podľa toho, či je možné hru s daným zadaním vyhrať.

Riešenia

1.2 Finále

1.2.1 Lov

Políčka miestnosti si nafarbíme ako šachovnicu. Potom ak sa nejaká figúrka pohne o párny počet políčok, bude na konci svojho ťahu na políčku rovnakej farby ako na začiatku ťahu. Ak sa pohne o nepárny počet políčok, bude na políčku opačnej farby.

1. Ak stoja vlk a zajac na začiatku na políčkach rôznej farby a parita Z_r a V_r je rovnaká, potom bude na konci vlkovho ťahu farba políčka na ktorom stojí rôzna od farby políčka, na ktorom stojí zajac a preto vlk nemôže zajaca chytiť.
2. Ak nenastal prípad 1, potom ak beží vlk rýchlejšie ako zajac, vždy ho chytí, lebo ich vzdialenosť (definovaná ako minimálny počet ťahov potrebných ne presunutie z jedného políčka na druhé) sa v každej dvojici ťahov zmenšuje.
3. Ak nenastal prípad 1 a vlk beží rovnako rýchlo ako zajac, potom ho tiež môže vždy chytiť. Jeho stratégia bude nasledovná: Vyberie si jeden roh miestnosti a postaví sa tak, aby zajac bol medzi ním a rohom R. Potom zajac bude ťahať buď tak, že ostane medzi rohom R a vlkom - vtedy môže unikať iba po rohové políčko, alebo môže ťahať tak, že nebude medzi vlkom a rohom R - vtedy sa vzdialenosť vlka a zajaca zmenší.
4. Ak nenastali prípady 1,2,3, potom ak existuje také políčko, že sa z neho môže vlk jedným ťahom dostať na ľubovoľné políčko, potom zrejme môže zajaca vždy chytiť. Podobne, ak je N párne a vlk sa z jedného zo stredových políčok môže dostať jedným ťahom na všetky políčka okrem jedného, môže vlk zajaca tiež vždy chytiť.
5. Ak nenastali prípady 1,2,3,4, potom ak N je nepárne a vlk sa zo stredového políčka môže dostať na ľubovoľné políčko okrem štyroch rohových, potom ak Z_r je nepárne, vlk zajaca chytí, ak Z_r je párne, vlk zajaca nechytí.
6. Ak nenastal ani jeden z predchádzajúcich prípadov môže zajac hrať tak, aby ho vlk nechytil.

Podľa týchto pravidiel potom môžme napísať vzorový program:

```
program Lov;
var
  vx, vy, vr, zx, zy, zr,
  n, d, r : integer;
  f, g : text;

begin
  assign(f, 'lov.in'); reset(f);
  assign(g, 'lov.out'); rewrite(g);
  while not eof(f) do begin
    readln(f, n, zx, zy, vx, vy, zr, vr);
    {vypocitame vzdialenost vlka a zajaca}
    d := abs(zx - vx) + abs(zy - vy);
    {je problem s farbou policka?}
    if (vr mod 2 = zr mod 2) and
       (d mod 2 = 1) then
      writeln(g, 'nemoze')
    else
      {ak vlk bezi rychlejsie,
       alebo rovnako rychlo ako zajac,
       tak ho urcite chyti}
      if vr >= zr then
        writeln(g, 'moze')
      else
        {moze vlk dojst na lubovolne
         policko zo stredneho}
        if ((n mod 2 = 0) and
            (vr >= n - 1)) or
           ((n mod 2 = 1) and
            (vr >= n - 1)) then
          writeln(g, 'moze')
        else
          {ESTE TIETO PRIPADY}
          if ((n mod 2 = 1) and
              (vr = n - 2)) then
            if zr mod 2 = 0 then
              writeln(g, 'nemoze')
            else
              writeln(g, 'moze')
          else
            writeln(g, 'nemoze');
  end;
  close(g); close(f);
end.
```

1.2.2 Kontrolóri kondenzátorov

Počet čísel z riadku, ktorých absolútna hodnota je väčšia ako -1 je rovný počtu týchto čísel. Rozdiel počtu čísel, ktorých absolútna hodnota je väčšia ako X a počtu čísel, ktorých absolútna hodnota je väčšia ako $X+1$ je rovný počtu čísel, ktorých absolútna hodnota sa rovná $X+1$. Keď si budeme pre každé X z

množiny $\{0, .., N\}$ pamätať počet čísel, ktorých absolútna hodnota je rovná X a budeme vedieť počet čísel v riadku, môžme postupne získavať počty čísel, ktorých absolútna hodnota je väčšia ako -1,0,1,2,3 atď.

```
program Kapacity;
const
  max = 4000;
var
  a : array[0..max] of integer;
  i, n, s, x : integer;
  f, g : text;

begin
  assign(f, 'kapacity.in'); reset(f);
  assign(g, 'kapacity.out'); rewrite(g);
  while not eof(f) do begin
    read(f,n);
    s := 0;
    for i := 0 to n do
      a[i] := 0;
    while not eoln(f) do begin
      read(f,x);
      a[abs(x)] := a[abs(x)] + 1;
      s := s + 1;
    end;
    readln(f);
    write(g,s);
    for i := 0 to n do begin
      s := s - a[i];
      write(g, ' ', s);
    end;
    writeln(g);
  end;
  close(g); close(f);
end.
```

1.2.3 Príklad o ostrove

Tento príklad je pomerne jednoduchý. Stačí prechádzať políčkami na pobreží - súčasne vľavo aj vpravo - a testovať, či už náhodou nie sme na políčku s pokladom. Treba však vymyslieť, čo najefektívnejší spôsob ako prejsť z políčka so súradnicami $[i, j]$ na nasledujúce políčko tak, aby sme išli po pobreží. Toto sa dá vyriešiť jednoducho, ak poznáme smer (prípadne políčko), z ktorého sme prišli. Toto je vo vzorovom programe uložené v premenných *odkial1* - pre Treska, a v premennej *odkial2* pre Pleska.

Pri prechádzaní ľavého pobrežia pozeráme políčka susediace hranou v smere hodinových ručičiek a hľadáme prvé políčko s jednotkou (pevnina). Na toto políčko sa posunieme v nasledujúcom kroku. Pri prechádzaní pravého pobrežia ideme proti smeru hodinových ručičiek.

Napríklad, ak sme na políčko so súradnicami $[i, j]$ prišli zdola (teda z políčka $[i + 1, j]$) a chceme ísť popri ľavom pobreží, tak začneme pozerať susedné políčka v poradí: $[i, j - 1]$, $[i - 1, j]$, $[i, j + 1]$ a $[i + 1, j]$ (teda ľavé, horné, pravé, dolné).

Je výhodné pamätať si poradie, v akom sa susedné políčka budú prezerať v pomocných poliach (vo vzorovom programe sú to polia v a p). Ďalej je výhodné urobiť okolo matice A zarážky - pridať stĺpce a riadky s indexami 0 a $N+1$ a naplniť ich nulami, aby sme pri prezeraní susedných políčok nemuseli testovať, či sme už náhodou neprekročili hranice matice A.

```
program Ostrov;
var
  p, v : array[0..3, 1..2] of integer;
  a : array[0..95, 0..95] of integer;
  riadok1, riadok2, stlpec1, stlpec2,
  odkial1, odkial2,
  dlzka, n, zacr, zacs, konr, kons,
  i, j : integer;
  nenasli : boolean;
  vstup, vystup : text;

begin
  p[0, 1] := -1; p[0, 2] := 0;
  p[1, 1] := 0; p[1, 2] := -1;
  p[2, 1] := 1; p[2, 2] := 0;
  p[3, 1] := 0; p[3, 2] := 1;
  v[0, 1] := -1; v[0, 2] := 0;
  v[1, 1] := 0; v[1, 2] := 1;
  v[2, 1] := 1; v[2, 2] := 0;
  v[3, 1] := 0; v[3, 2] := -1;
  assign(vstup, 'ostrov.in');
  reset(vstup);
  assign(vystup, 'ostrov.out');
  rewrite(vystup);
  while not eof(vstup) do begin
    readln(vstup, n);
    { nacitanie vstupnych hodnot }
    for i := 1 to n do begin
      a[0, i] := 0;
      a[i, 0] := 0;
      a[n + 1, i] := 0;
      a[i, n + 1] := 0;
      for j := 1 to n do
        read(vstup, a[i, j]);
      readln(vstup);
    end;
    readln(vstup, zacr, zacs);
    readln(vstup, konr, kons);
    { nastavenie pociatocnych hodnot
      pre hladanie }
    dlzka := 0;
    odkial1 := 0;
    riadok1 := zacr; stlpec1 := zacs;
    riadok2 := zacr; stlpec2 := zacs;
    repeat
      odkial1 := (odkial1 + 1) mod 4;
    until a[zacr + p[odkial1, 1],
            zacs + p[odkial1, 2]] = 0;
    if odkial1 in [0, 2] then
      odkial2 := odkial1
```

```
  else
    odkial2 := (odkial1 + 2) mod 4;
  nenasli := true;
  { samotne hladanie }
  while nenasli do begin
    repeat
      odkial1 := (odkial1 + 1) mod 4;
    until a[riadok1 + p[odkial1, 1],
            stlpec1 + p[odkial1, 2]] = 1;
    riadok1 := riadok1 + p[odkial1, 1];
    stlpec1 := stlpec1 + p[odkial1, 2];
    odkial1 := (odkial1 + 2) mod 4;
    repeat
      odkial2 := (odkial2 + 1) mod 4;
    until a[riadok2 + v[odkial2, 1],
            stlpec2 + v[odkial2, 2]] = 1;
    riadok2 := riadok2 + v[odkial2, 1];
    stlpec2 := stlpec2 + v[odkial2, 2];
    odkial2 := (odkial2 + 2) mod 4;
    inc(dlzka);
    nenasli := false;
    { zistovanie vysledku }
    if (riadok1 = konr) and
       (stlpec1 = kons) then
      if (riadok2 = konr) and
         (stlpec2 = kons) then
        writeln(vystup, 'naraz ', dlzka)
      else
        writeln(vystup, 'Tresk ', dlzka)
    else if (riadok2 = konr) and
            (stlpec2 = kons) then
      writeln(vystup, 'Plesk ', dlzka)
    else
      nenasli := true
  end
end;
close(vstup); close(vystup);
end.
```

1.2.4 Matica

V riešení sa používa postupná eliminácia riadkov alebo stĺpcov matice. Máme dve pozície, z ktorých môžeme začať pri prehľadávaní matice. Ľavý dolný roh matice alebo pravý horný roh matice. My budeme začínať od ľavého dolného rohu matice. V ľavom dolnom rohu určite vieme, že prvky napravo v riadku sú väčšie a prvky nahor v stĺpci menšie ako políčko, na ktorom stojíme. Teda keď je číslo D väčšie ako číslo, na ktorom v matici stojíme, posunieme sa o jedno políčko doprava (v opustenom stĺpci sa D nemôže nachádzať). A keď je číslo D menšie ako číslo, na ktorom v matici stojíme, posunieme sa o jedno políčko nahor (v opustenom riadku sa D nemôže nachádzať). Takýmto spôsobom buď nájdeme v matici číslo D, alebo posúvaním sa prekročíme hranice matice, teda D sa v matici nebude nachádzať. Pri hľadaní D sa po matici budeme posúvať najviac $m + n$ - krát, teda algoritmus je lineárny vzhľadom na rozmery matice.

```
program matica;
var
  vstup, vystup : text ;
  m, n, i, j : integer;
  a : array [1..150, 1..150] of integer;

procedure ries_dotazy;
{ hladanie prvkov v matici }
var
  kolko, k, hladany : integer;
begin
  readln(vstup, kolko);
  for k := 1 to kolko do begin
    read(vstup, hladany);
    i := m;
    j := 1;
    while (i >= 1) and (j <= n) and
          (a[i, j] <> hladany)  do
      if a[i, j] < hladany then
        inc(j)
      else
        dec(i);
    if (i < 1) or (j > n) then
      write(vystup, 'nie ')
    else
      write(vystup, 'ano ')
  end;
  if kolko > 0 then
    writeln(vystup)
end;

begin
  Assign(vstup,'matica.in');
  Reset(vstup);
  Assign(vystup,'matica.out');
  Rewrite(vystup);
  while true do begin
    read(vstup, m);
    readln(vstup, n);
    if (m = 0) and (n = 0) then break;
    for i := 1 to m do begin
      for j := 1 to n do
        read(vstup, a[i, j]);
      readln(vstup);
    end;
    ries_dotazy;
  end;
  close(vstup);
  close(vystup);
end.
```

1.2.5 Kocka

Samotné otáčanie steny kocky sa skladá z dvoch činností. Jednak je to otočenie čelnej steny, ktoré urobíme jednoduchým prepočítaním súradníc riadkov a stĺpcov.

A potom sa musíme ešte postarať o otočenie častí stien, ktoré sú bezprostrednými susedmi otáčanej steny. Využijeme na to tabuľku, kde máme zaznamenané ku každej stene údaje, ktoré časti jej susedov sa

budú pri otáčaní steny otáčať tiež. Keďže máme každú stenu kocky reprezentovanú dvojrozmernou maticou, tak situácia bude vždy iná, lebo raz otáčame riadok matice, inokedy stĺpec. V rôznych situáciách otáčame iné riadky alebo iné stĺpce, a tiež sa môže stať, že otáčanie začína od konca riadku, inokedy od začiatku.

Pre jednotlivé steny máme susedov zaznamenaných postupne v pravotočivej orientácii. Pomôže nám to pri implementácii samotného otáčania. Napr. pre stenu A budú nasledovať údaje najprv o stene F, potom D, E, B. Je jedno, ktorou stenou začíname, len treba dodržať ľavotočivosť.

Samotný údaj o stene je štruktúra pozostávajúca z položiek:

- s, sused - o ktorom susedovi steny sa teraz podáva informácia
- r, riadok - tu je naznačené vhodným spôsobom, či sa budú otáčať riadky alebo stĺpce
- c, číslo - číslo riadku alebo stĺpca, ktorý sa bude otáčať
- t, status - či sa začne otáčať od začiatku alebo konca popísaného riadku alebo stĺpca

Operáciu otáčania prislúchajúcich častí bočných stien urobíme tak, že do pomocnej premennej nazbierame tieto časti v ľavotočivej orientácii, a potom podľa typu otočenia uložíme tieto časti na posunuté miesta.

```
program Kocka_;

const
  dl_strany = 3;
type
  co_toc = record
    s : byte;    { sused }
    r : boolean; { true  -  riadok tocime,
                   false  -  stlpec tocime}
    c : byte;    { cislo toceneho riadku
                   alebo stlpca }
    t : byte;    { na ktorom stlpci zacina
       posuvany riadok resp. na ktorom
       riadku zacina posuvany stlpec }
  end;
  stvorec = array[1..dl_strany,
                  1..dl_strany] of char;
const
   sused :
     array[1..6, 1..4] of co_toc = (
       ( (s : 6; r : true; c : 3; t : 1),
         (s : 4; r : false; c : 3; t : 0),
         (s : 5; r : true; c : 1; t : 0),
         (s : 2; r : false; c : 1; t : 1) ),
       ( (s : 1; r : false; c : 3; t : 0),
         (s : 5; r : false; c : 3; t : 0),
         (s : 3; r : false; c : 1; t : 1),
         (s : 6; r : false; c : 3; t : 0) ),
       ( (s : 6; r : true; c : 1; t : 0),
         (s : 2; r : false; c : 3; t : 0),
         (s : 5; r : true; c : 3; t : 1),
         (s : 4; r : false; c : 1; t : 1) ),
       ( (s : 1; r : false; c : 1; t : 1),
         (s : 6; r : false; c : 1; t : 1),
         (s : 3; r : false; c : 3; t : 0),
         (s : 5; r : false; c : 1; t : 1) ),
       ( (s : 1; r : true; c : 3; t : 1),
         (s : 4; r : true; c : 3; t : 1),
         (s : 3; r : true; c : 3; t : 1),
         (s : 2; r : true; c : 3; t : 1) ),
       ( (s : 1; r : true; c : 1; t : 0),
         (s : 2; r : true; c : 1; t : 0),
         (s : 3; r : true; c : 1; t : 0),
         (s : 4; r : true; c : 1; t : 0) ) );

var
  kocka : array[1..6] of stvorec;
  pom : array[1..4 * dl_strany] of char;
  a : stvorec;
  farby_stien : string[6];
  tocenie : string;
  i, j, k, l, dlzka : integer;
  strana, toc : byte;
  f, g : text;

procedure inicializacia;
{ nainicializovanie farieb stien }
begin
  for i := 1 to 6 do
   for j := 1 to dl_strany do
    for k := 1 to dl_strany do
     kocka[i, j, k] := farby_stien[i];
end;

procedure vypis;
begin
  writeln(g,  '      ', kocka[6, 1, 1],
    kocka[6, 1, 2], kocka[6, 1, 3]);
  writeln(g,  '      ', kocka[6, 2, 1],
    kocka[6, 2, 2], kocka[6, 2, 3]);
  writeln(g,  '      ', kocka[6, 3, 1],
    kocka[6, 3, 2], kocka[6, 3, 3]);
  writeln(g,  '  ', kocka[4, 1, 1],
    kocka[4, 1, 2], kocka[4, 1, 3], ' ',
    kocka[1, 1, 1], kocka[1, 1, 2],
    kocka[1, 1, 3], ' ', kocka[2, 1, 1],
    kocka[2, 1, 2], kocka[2, 1, 3], ' ',
    kocka[3, 1, 1], kocka[3, 1, 2],
    kocka[3, 1, 3]);
  writeln(g,  '  ', kocka[4, 2, 1],
    kocka[4, 2, 2], kocka[4, 2, 3], ' ',
    kocka[1, 2, 1], kocka[1, 2, 2],
    kocka[1, 2, 3], ' ', kocka[2, 2, 1],
    kocka[2, 2, 2], kocka[2, 2, 3], ' ',
    kocka[3, 2, 1], kocka[3, 2, 2],
    kocka[3, 2, 3]);
  writeln(g,  '  ', kocka[4, 3, 1],
    kocka[4, 3, 2], kocka[4, 3, 3], ' ',
    kocka[1, 3, 1], kocka[1, 3, 2],
    kocka[1, 3, 3], ' ', kocka[2, 3, 1],
    kocka[2, 3, 2], kocka[2, 3, 3], ' ',
    kocka[3, 3, 1], kocka[3, 3, 2],
    kocka[3, 3, 3]);
  writeln(g,  '      ', kocka[5, 1, 1],
    kocka[5, 1, 2], kocka[5, 1, 3]);
  writeln(g,  '      ', kocka[5, 2, 1],
    kocka[5, 2, 2], kocka[5, 2, 3]);
  writeln(g,  '      ', kocka[5, 3, 1],
    kocka[5, 3, 2], kocka[5, 3, 3]);
end;

procedure uloz(strana, toc : byte);
```

```
{ otocenie prisluchajucich bocnych
  casti pri otacani strany }
var
  i : integer;
begin
  for i := 1 to 4 do
  begin
    if sused[strana, toc].r then
      { ukladame riadky }
      if sused[strana, toc].t = 1 then
        { od zaciatku riadka }
        for l := 1 to dl_strany do
          kocka[sused[strana, toc].s,
                sused[strana, toc].c,l] :=
            pom[3 * (i - 1) + l]
      else
        { od konca riadka }
        for l := 1 to dl_strany do
          kocka[sused[strana, toc].s,
                sused[strana, toc].c, 4 - l] :=
            pom[3 * (i - 1) + l]
    else
      { ukladame stlpce }
      if sused[strana, toc].t = 1 then
        { od zaciatku stlpca }
        for l := 1 to dl_strany do
          kocka[sused[strana, toc].s,
                l, sused[strana, toc].c] :=
            pom[3 * (i - 1) + l]
      else
        { od konca stlpca }
        for l := 1 to dl_strany do
          kocka[sused[strana, toc].s,
                4 - l, sused[strana, toc].c] :=
            pom[3 * (i - 1) + l];

    inc(toc);
    if toc > 4 then
      toc := toc - 4;
  end;
end;

procedure nacitaj(strana : byte);
{ nacitanie prislusnych bocnych casti
  pri otacani strany do pomocneho pola }
var
  i : integer;
begin
  for i := 1 to 4 do
    if sused[strana, i].r then
      { nacitanie riadku }
      if sused[strana, i].t = 1 then
        { od zaciatku riadku }
        for l := 1 to dl_strany do
          pom[3 * (i - 1) + l] :=
            kocka[sused[strana, i].s,
                  sused[strana, i].c, l]
      else
        { od konca riadku }
        for l := 1 to dl_strany do
          pom[3 * (i - 1) + l] :=
            kocka[sused[strana, i].s,
                  sused[strana, i].c, 4 - l]
    else
      {nacitanie stlpca}
      if sused[strana, i].t = 1 then
        { od zaciatku stlpca }
        for l := 1 to dl_strany do
          pom[3 * (i - 1) + l] :=
            kocka[sused[strana, i].s,
                  l, sused[strana, i].c]
      else
        { od konca stlpca }
        for l := 1 to dl_strany do
          pom[3 * (i - 1) + l] :=
            kocka[sused[strana, i].s,
                  4 - l, sused[strana, i].c];
end;

begin   { hlavny program }

  Assign(f, 'kocka.in'); Reset(f);
  Assign(g, 'kocka.out'); Rewrite(g);

  while not eof(f) do
  begin
    readln(f, farby_stien);
    inicializacia;
    readln(f, tocenie);
    dlzka := length(tocenie) div 2;

    for i := 1 to dlzka do
    begin
      strana :=
        ord(tocenie[2 * i - 1]) - 96;
      toc := ord(tocenie[2 * i]) - 48;
      {otocenie strany}
      for j := 1 to dl_strany do
        for k := 1 to dl_strany do
          case toc of
            1 : a[k + 2 * ord(k = 1) -
                      2 * ord(k = 3), j] :=
                  kocka[strana, j, k];
            2 : a[j + 2 * ord(j = 1) -
                      2 * ord(j = 3),
                  k + 2 * ord(k = 1) -
                      2 * ord(k = 3)] :=
                  kocka[strana, j, k];
            3 : a[k,
                  j + 2 * ord(j = 1) -
                      2 * ord(j = 3)] :=
                  kocka[strana, j, k];
        end;
      kocka[strana] := a;
      { otocenie prislusnych
        bocnych casti }
      nacitaj(strana);
      uloz(strana, toc + 1);
    end;
    vypis;
    writeln(g);
  end;
  close(f); close(g);
end.
```

1.2.6 Preteky

Označme si $a[i]$ riešenie pre prvých i kilometrov. To znamená koľko kilometrov z $\langle 0, i\rangle$ ostane nepokrytých, ak pokrývame neprekrývajúcimi sa intervalmi interval $\langle 0, i\rangle$ (čiže vyberáme len tie podintervaly zo vstupných K intervalov, ktoré celé ležia v $\langle 0, i\rangle$). Vzorové riešenie vychádza z nasledujúcej úvahy:

Keby sme poznali riešenia $a[0], \ldots, a[i]$, riešenie $a[i+1]$ dostaneme nasledovne:

1. Ak neexistuje vstupný podinterval $\langle j, i+1\rangle$, tak $a[i+1] = a[i]+1$. Optimálnym pokrytím bude v tomto prípade optimálne pokrytie pre prvých i kilometrov. Toto triviálne vyplýva z toho, že $a[i]$ je riešenie pre prvých i kilometrov.
2. V opačnom prípade je $a[i+1]$ minimum z tých $a[j]$, pre ktoré existuje vstupný podinterval $\langle j, i+1\rangle$. Optimálnym pokrytím bude optimálne pokrytie pre prvých j kilometrov, ku ktorému pridáme interval $\langle j, i+1\rangle$.

Programová realizácia:

Pre každé i si budeme pamätať zoznam intervalov, ktoré v ňom končia. Toto vytvárame počas čítania vstupu. Prípad 1. rozpoznáme tak, že tento zoznam je prázdny. Prípad 2. sa realizuje jednoduchým cyklom, v ktorom prejdeme týmto zoznamom.

```
program preteky;
const
  MAX_N = 1000;
type
  PInterval = ^Interval;
  Interval = record
               x : integer;
               next : PInterval
             end;
var
  N, K : integer;
  a : array[0..MAX_N] of integer;
  z : array[0..MAX_N] of PInterval;
  i, x, y : integer;
  zz : PInterval;
  p : pointer;
  f, g : text;

function rob : integer;
var
  i : integer;
  zz : PInterval;
begin
  a[0] := 0;
  for i := 1 to N do begin
    a[i] := a[i - 1] + 1;
    zz := z[i];
    while (zz <> nil) and (a[i] > 0) do begin
      if a[zz^.x] < a[i] then
        a[i] := a[zz^.x];
      zz := zz^.next
    end
  end;
  rob := a[N]
end;

begin
  assign(f, 'PRETEKY.IN'); reset(f);
  assign(g, 'PRETEKY.OUT'); rewrite(g);
  while not eof(f) do begin
    readln(f, N, K);
    mark(p);
    for i := 1 to N do z[i] := nil;
    for i := 1 to K do begin
      read(f, x, y);
      new(zz);
      zz^.next := z[y];
      zz^.x := x;
      z[y] := zz
    end;
    readln(f); writeln(g, rob);
    release(p);
  end;
  close(f); close(g);
end.
```

2.1 Domáce kolo

2.1.1 Archeológ

Hoci tento príklad nebol veľmi ťažký, väčšina vašich riešení obsahovala nejakú chybu. Ale ako ste si mnohí všimli, ani zadanie nebolo celkom dokonalé, lebo nebolo z neho úplne jasné, či dva štvorčeky dotýkajúce sa rohom tvoria jeden úlomok. Vo vzorovom riešení sa takéto štvorčeky považujú za dva úlomky.

Celkovo sa vyskytli dva druhy riešení:

1. "Prikladacie" algoritmy. Tu ste sa snažili vziať mapu diery, postupne jej začiatok prikladať na všetky políčka mapy stola a kontrolovať, či je všetko v poriadku. Je zrejmé, že všade tam, kde je na mape diery `O`, musí byť na mape stola `X`. Ale nie všetci si uvedomili, že tam, kde je na mape diery `X`, môže byť na stole tiež `X`, lebo do štvorca na stole, na ktorý sme priložili tanier, môžu zasahovať aj iné úlomky, ktoré sa s tým správnym nedotýkajú. Aby však úlomok neprečnieval z diery von, pre všetky `X` na mape diery, ktoré susedia s nejakým `O` musí platiť, že na príslušnom políčku stola je bodka.

Vo vašich riešeniach sa tiež často vyskytli problémy s okrajmi polí. Mnohí ste si neuvedomili, že N môže byť menšie ako M. Alebo ak sa okraj diery nachádzal na kraji mapy, nekontrolovali ste, či úlomok náhodou neprečnieva z priloženého štvorca a podobne. Niektoré z týchto problémov sa dali vyriešiť tým, že ste z mapy diery odrezali tie stĺpce a riadky, v ktorých sa nevyskytovali žiadne časti diery a potom ste okolo obidvoch máp vytvorili takzvanú zarážku (na mape stola pozostávala zo znakov `.` (bodka), na mape diery zo znakov `X`). Algoritmus sa dal mierne zlepšiť tým, že ste našli na mape diery prvé `O` a potom ste tento štvorček prikladali len na tie štvorčeky mapy stola, kde sa nachádzalo písmeno `X`.

Tento typ riešení bol však všeobecne menej efektívny a preto ste zaňho mohli získať maximálne 8 bodov.

2. "Vyfarbovacie" algoritmy. Tieto sa zakladali na postupe ako prejsť všetky políčka jedného úlomku. Buď ste použili rekurzívnu procedúru, ktorá previedla potrebné operácie a potom sa zavolala pre všetky susedné políčka, alebo ste si do nejakého pomocného poľa ukladali políčka cez ktoré ste prešli, prípadne akým smerom ste išli a podobne. V oboch prípadoch je dôležité nezabudnúť už spracované políčka stola nejako označiť, aby sme sa do nich už nevracali pri spracúvaní aktuálneho úlomku a tiež aby sme ich nepovažovali za nový úlomok, keby sa aktuálny nezhodoval s dierou. Aj pri tomto type riešení treba dať pozor na vybehnutie z poľa.

Vyfarbovanie je efektívnejšie ako prikladanie, lebo každé políčko mapy stola porovnávame iba konštantný počet krát. Za takéto riešenie ste mohli získať až 10 bodov.

Vzorové riešenie využíva rekurziu. Do polí *Stol* a *Diera* sa načíta vstup, okolo poľa *Stol* sa vytvorí zarážka z bodiek. Potom procedúra *Porovnaj* nájde prvý štvorček diery a postupne hľadá v poli *Stol* ešte nespracované úlomky. Pre prvé políčko úlomku sa zavolá funkcia *Porovnaj*. Tá prejde postupne celý úlomok a s ním susediace políčka, úlomok prepíše z `X` na `x`. Okrem toho skontroluje, či každé políčko úlomku zodpovedá diere v mape diery a každé políčko s úlomkom susediace tanieru v mape diery. Ak to naozaj platí, zapíšu sa súradnice prvého políčka úlomku do výstupného súboru. Keďže sme použili zarážku, nemusíme sa obávať, že by sme vyšli z mapy stola. Ale kvôli kontrole okraja mapy diery je v programe použitá funkcia *JeDiera*, ktorá pre zadané súradnice určí, či je na mape diery `O`, alebo vráti *false*, ak aspoň jedna zo súradníc nie je z rozsahu 1 až M.

```
program archeolog;
const
  max = 99;
var
  Stol : array[0..max+1, 0..max+1] of char;
  {mapa stola}
  Diera : array[1..max, 1..max] of char;
  {mapa diery}
  i, prikladov : integer; {pocet prikladov}
  n, m : integer;   {rozmery stola a diery}
  fi, fo : text; {vstupny a vystupny subor}

procedure Nacitaj;
var
  x, y : integer;
begin
  readln(fi, n, m);        {nacitame rozmery}
  for y := 1 to n do begin  {nac. mapu stola}
    for x := 1 to n do {'X'-ulomok, '.'-stol}
      read(fi, Stol[x, y]);
    readln(fi);
  end;
  for y := 1 to m do begin  {nac. mapu diery}
    for x := 1 to m do {'X'-tanier,'O'-diera}
      read(fi, Diera[x, y]);
    readln(fi);
  end;
  {zarazka okolo mapy stola}
  for y := 0 to n + 1 do begin
    Stol[0, y] := '.';
    Stol[y, 0] := '.';
    Stol[n + 1, y] := '.';
    Stol[y, n + 1] := '.';7
  end;
end;

procedure Hladaj;
var
  xc, yc, {surad. prave skumaneho crepu}
  xd, yd,              {suradnice diery}
  xposun,   {rozdiel sur. crepu a diery}
  yposun : integer;
  ok : boolean;       {ci sme uz nasli}

  procedure Dalsi(var x, y : integer;
                  max:integer);
  begin
    inc(x);       {posunieme sa doprava}
    if x > max then begin {ak sme za koncom}
      x := 1; inc(y);   {riadku, skocime na}
    end;               {zaciatok dalsieho}
  end;

  function JeDiera(x,y:integer):boolean;
  begin
    if (x >= 1) and (y >= 1) and
       (x <= m) and (y <= m) then
      JeDiera := (Diera[x, y] = 'O')
    else
      JeDiera := false;  {x,y su mimo pola}
  end;

  function Rovnake(x, y : integer) : boolean;
  var
    ok : boolean;
  begin
    if Stol[x, y] = 'X' then begin
      {este nespracovany crep}
      Stol[x, y] := 'x'; {oznacime ho}
      {skontrolujeme, ci je diera}
      ok := JeDiera(x + xposun, y + yposun);
      {rekurzivne zavolame susedov}
      if not Rovnake(x - 1, y) then ok := false;
      if not Rovnake(x, y - 1) then ok := false;
      if not Rovnake(x + 1, y) then ok := false;
      if not Rovnake(x, y + 1) then ok := false;
      Rovnake := ok;
    end else if Stol[x, y] = 'x' then
      {uz spracovany crep}
      Rovnake := JeDiera(x + xposun, y + yposun)
    else                {stol}
      {ci diera nepretrca}
      Rovnake :=
        not JeDiera(x + xposun, y + yposun);
  end;

begin  {procedure hladaj}
  {najdeme dieru v nadobe ... prve pismeno O}
  xd := 1; yd := 1;
  while Diera[xd, yd] <> 'O' do
```

```
    Dalsi(xd, yd, m);
  xc := 1; yc := 1;
  repeat
    {najdeme na stole crep...pismeno X}
    while Stol[xc, yc] <> 'X' do
      Dalsi(xc, yc, n);
    xposun := xd - xc;
    yposun := yd - yc;
    {zistime, ci sa zhoduje
        a prepiseme ho malymi x}
    ok := Rovnake(xc, yc);
  until ok;
  writeln(fo, yc, ' ', xc); {vypiseme vysledok}
end;
```

```
begin
  assign(fi, 'archeolo.in');
  assign(fo, 'archeolo.out');
  reset(fi);
  rewrite(fo);
  {nacitame pocet prikladov}
  readln(fi, prikladov);
  {a pre kazdy z nich}
  for i := 1 to prikladov do begin
    Nacitaj;  {nacit. n,m a polia Stol,Diera}
    Hladaj;   {a hladame spravny ulomok}
  end;
  close(fi); close(fo);
end.
```

2.1.2 Hra

Vaše riešenia možno rozdeliť v zásade na dva typy:

Simulačné riešenia – tj. také, ktoré s nejakým časovým krokom sledovali pozíciu oboch lôpt a tak zisťovali, či sa zrazia, alebo nie - keďže ide o veľmi nepresnú metódu, maximálne bolo možné za takéto riešenie získať 6 bodov

Výpočtové riešenia – tj. riešenia, ktoré použili na riešenie úlohy jednoduchý výpočet (viď. ďalej) - maximálne 10 bodov.

Oproti maximálnemu počtu bodov za určitý typ riešenia bolo možné stratiť body za nešikovne naprogramované detaily riešenia, za neošetrenie špeciálnych prípadov, alebo za chýbajúce ukážky behu programu.

Vzorové riešenie najprv rozloží rýchlosti lôpt na rýchlosť v x-ovej a y-ovej súradnice (známe vzťahy z goniometrie). Ďalej vieme vypočítať čas, kedy sa lopty (ak sa zrazia) zrazia, keďže v tom čase musia mať rovnaké x-ové súradnice a potom už len stačilo spočítať, či sa y-ové súradnice v danom čase zhodujú.

Pri písaní programu si bolo treba dať pozor na špeciálny prípad, keď sa obe lopty nepohybujú v x-ovom smere (delenie nulou pri výpočte času), na efektívnu realizáciu výpočtu y-ovej súradnice (veľa z vás y-ovú súradnicu počítalo pomocou odčítania pre každý odraz - tých ale môže byť veľmi veľa), ako aj na porovnávanie reálnych čísel (pri výpočtoch sa naráža na hranicu presnosti výpočtov s reálnymi číslami, preto je potrebné pri porovnaní povoliť nejakú zanedbateľnú chybu - viď. konštanta *eps*).

```
program hra;
const
  pi = 3.1415926535;
  eps = 0.001;
var
  m, n, u1, v1, u2, v2 : real;
  v1x, v1y, v2x, v2y : real;
  y1, y2, t : real;
  pochier, i : integer;
  inp, out : text;

function nahr(a : real) : real;
{vstupom do procedury je vypocitana
 y-ova suradnica bez zapocitania
 odrazov,  vystupom je suradnica
 so zapocitanim odrazov}
var
  n : real;
begin
  {odcitame odrazy od obidvoch
  okrajov ihriska}
  if abs(a) > m then
    n := -((a / m) - trunc(a / m)) * m
  else
    n := a;
  if n > m / 2 then n := m - n;
  if n < -m / 2 then n := m + n;
  nahr := n;
end;
```

```
begin
  assign(inp, 'hra.in'); reset(inp);
  assign(out, 'hra.out'); rewrite(out);
  readln(inp, pochier);
  for i := 1 to pochier do begin
    readln(inp, m, n);
    readln(inp, u1, v1);
    readln(inp, u2, v2);
    {rozlozime rychlosti do kolmych smerov}
    v1x := v1 * cos(u1 * pi / 180);
    v1y := v1 * sin(u1 * pi / 180);
    v2x := -v2 * cos(u2 * pi / 180);
    v2y := v2 * sin(u2 * pi / 180);
    {osetrime specialny pripad}
    if v1x + v2x = 0 then
      writeln(out, 'nezrazia sa')
    else begin
     {vypocitame cas rovnakych
      x-ovych suradnic}
      t := n / (v1 + v2);
     {vypocitame y-ovu suradnicu}
      y1 := nahr(t * v1y);
      y2 := nahr(t * v2y);
     {porovnanie rieseni}
      if abs(y1 - y2) > eps then
        write(out, 'ne');
      writeln(out, 'zrazia sa')
    end;
```

```
  end;
  close(inp); close(out);
end.
```

2.1.3 Karty

Tento príklad sa pokúsila riešiť väčšina účastníkov. Napriek tomu, že vyzerá ľahký len málo ľudí získalo plný počet bodov. Vaše riešenia pracovali väčšinou tak, že načítali balíček, rozdali ho štyrom hráčom a potom karty usporiadali na ruke každého hráča. Takéto riešenie nie je zlé, ale príklad sa dá vyriešiť aj bez použitia sortovania. Vzorové riešenie pracuje tak, že postupne načítava karty z balíčka a pre každú kartu si v poli *Vlastnik* značí, ktorému hráčovi patrí. Potom prejde všetkými kartami od najmenšej po najväčšiu a pridá ju na ruku hráčovi, ktorému patrí. Tým, že karty prechádzame od najmenšej po najväčšiu dosiahneme, že hráči ich budú mať usporiadané v dobrom poradí.

```
program Karty;
const
  Farby : array[1..4] of char = 'LKSP';
  Cisla : array[1..13] of char=
    '23456789DJQKA';
var
  Vlastnik : array[1..4,1..13] of byte;
  Hraci : array[1..4] of string[26];
  Fin, Fout : text;
  i, j, Hier:integer;
  C, F:char;

begin
  assign(Fin, 'karty.in');
  reset(Fin);
  assign(Fout, 'karty.out');
  rewrite(Fout);
  readln(Fin, Hier);
  while (Hier > 0) do begin
    for i := 0 to 51 do begin
      read(Fin, C, F);
      Vlastnik[pos(F, Farby), pos(C, Cisla)]
        := 1 + (i mod 4);
    end;
    for i := 1 to 4 do Hraci[i] := '';
    for i := 1 to 4 do
      for j := 1 to 13 do
        Hraci[Vlastnik[i, j]] :=
          Hraci[Vlastnik[i, j]] +
          Cisla[j] + Farby[i];
    for i := 1 to 4 do
      writeln(Fout, Hraci[i]);
    Hier := Hier - 1;
  end;
  close(Fin); close(Fout);
end.
```

2.1.4 Malá násobilka

Tento príklad bol najľahší, jeho riešenie nám poslalo 90 družstiev. Všetky riešenia boli správne, líšili sa navzájom iba programátorskými detailami. Najjednoduchšie riešenie bolo použitím dvojrozmerného poľa rozmerov 37×37. Do ktorého ste ukladali výsledky násobenia. Ak ste nevyužili, že platí $ab = ba$, tak ste za takéto riešenie mohli získať 7 bodov. Pri využití symetrie v poli 9 bodov. Riešenia bez poľa, v ktorom ste každú dvojicu čísiel vynásobili bolo za 9 bodov. A ak ste využili, že susedné čísla v riadoch tabuľky sa líšia vždy o konštantu ($a(b+1) = ab + a$), tak ste mohli získať 10 bodov.

Za prevod čísiel sme body nestrhávali ani nepridávali. Bod sa strhával za programátorské chyby, ktoré máte označené v riešení a za chýbajúci ľubovoľný popis (komentár, prípadne vysvetľujúca veta). Riešenie kolektívu Marko, Vozárová:

```
program nasobilka;
const
  pismeno : array[0..36] of char =
    ('0', '1', '2', '3', '4', '5', '6',
     '7', '8', '9', 'A', 'B', 'C', 'D',
     'E', 'F', 'G', 'H', 'I', 'J', 'K',
     'L', 'M', 'N', 'O', 'P', 'Q', 'R',
     'S', 'T', 'U', 'V', 'W', 'X', 'Y',
     'Z', '?');
var
  pocet, i, j, k, n, h1, h2 : integer;

begin
  assign(input, 'mala-nas.in');
  reset(input);
  assign(output, 'mala-nas.out');
  rewrite(output);
  readln(pocet);
  for i := 1 to pocet do begin
    readln(n);
    {nasobime j * k}
    for j := 1 to n - 1 do begin
      {h1 je prva cifra, h2 druha}
      h1 := 0; h2 := 0;
      for k := 1 to n - 1 do begin
        inc(h1, j);
        if h1 >= n then begin
          {ak sme prevysili zaklad}
          inc(h2);
          dec(h1, n)
        end;
        write(' ');
        {nevypisujeme veduce 0}
        if h2 = 0 then write(' ')
        else write(pismeno[h2]);
        write(pismeno[h1])
```

```
      end;
      writeln(' ', pismeno[j], '0')
    end;
    {nasobenie zakladom je lahke}
    for j := 1 to n - 1 do
      write(' ', pismeno[j], '0');
    writeln(' 100')
  end;
  close(input); close(output)
end.
```

2.1.5 Žabka

Väčšina z Vás riešila tento príklad správne, boli však rozdiely v časovej i pamäťovej efektívnosti. Riešenia sa dajú rozdeliť do dvoch skupín: rekurzívne a nerekurzívne. V rekurzívnych riešeniach treba mať na pamäti, že parametre funkcií sa ukladajú do zásobníka, čím narastá pamäťová zložitosť. Často ste tiež po dosiahnutí správneho riešenia síce volali *exit*, ale týmto ste opustili iba najspodnejšiu funkciu a tak ste prehľadali aj všetky zostávajúce cesty. Nerekurzívne riešenia často simulovali rekurziu zásobníkom, používali ste aj front (cyklický alebo acyklický). Tu ste niekedy vyhradili primálo pamäti (cesta môže byť dlhá rádovo N^2). Iné riešenia spočívali vo vypĺňaní - väčšinou rádovo zhoršili časovú zložitosť. Cyklicky totiž prechádzali celé pole a označili okolia už označených políčok. Ďalší nápad je využiť existujúcu štruktúru a zapísať si "návratové adresy" priamo na políčka močiara.

```
program Zabka;
const
  MaxN = 200;
  smery_i : array[1..8] of integer =
    (-1, -2, 0, 0, 0, 0, 2, 1);
  smery_j : array[1..8] of integer =
    ( 0, 0, 1, 2, -2, -1, 0, 0);
var
  Mociar : array[1..MaxN, 1..MaxN] of integer;
  PocetUloh : integer;
  N : integer;

function DaSaSkocit(i, j,
          smer : integer) : boolean;
begin
  if (i + smery_i[smer] > N) or
     (i + smery_i[smer] < 1) or
     (j + smery_j[smer] > N) or
     (j + smery_j[smer] < 1) then
    DaSaSkocit := false
  else
    DaSaSkocit :=
      Mociar[i + smery_i[smer],
             j + smery_j[smer]] = 1
end;

function Skac : boolean;
var
  i, j, smer : integer;
  koniec : boolean;
begin
  i := N; j := 1;
  koniec := false;
  while not koniec and
        ((i <> 1) or (j <> N)) do begin
    smer := 1;
    while (smer < 9) and not
          DaSaSkocit(i, j, smer) do
      inc(smer);
    if smer < 9 then begin
      i := i + smery_i[smer];
      j := j + smery_j[smer];
      Mociar[i,j] := smer - 9
    end else if (i = N) and (j = 1) then
      koniec := true
    else begin
      smer := -Mociar[i, j];
      Mociar[i, j] := 0;
      i := i + smery_i[smer];
      j := j + smery_j[smer]
    end
  end;
  if koniec then Skac := false
  else
    Skac := true
end;

var
  i, j, k : integer;
  s : string[maxN];
  f, g : text;
begin
  assign(f, 'zabka.in');
  reset(f);
  assign(g, 'zabka.out');
  rewrite(g);
  read(f, PocetUloh);
  for k := 1 to PocetUloh do begin
    readln(f, N);
    for i := 1 to N do begin
      readln(f, s);
      for j := 1 to N do
        Mociar[i,j] := ord(s[j]) - ord('0')
    end;
    if not Skac then
      write(g, 'ne');
    writeln(g, 'moze prejst')
  end;
  close(f); close(g)
end.
```

2.2 Finále

2.2.1 Hra

V tomto príklade stačilo mať zapamätaný momentálny stav hracej plochy a simulovať každý ťah vstupnej postupnosti. (T.j. nájdem, ktorým štvorčekom mám potiahnuť, zistím, či susedí s dierou, ak áno, potiahnem,

ak nie, vyhlásim chybu). Jediný problém môže byť s reprezentáciou hracej plochy. Je totiž výhodnejšie pamätať si pre každý štvorček jeho pozíciu (číslované po riadkoch zhora dolu). Vyhľadanie je potom v možné v konštantnom čase a výmena za dieru tiež.

```
program Posunovacky;
{ maximalny rozmer plochy }
const
  MaxN = 100;
var
  fin, fout : text;
    { Pre kazde cislo mam zapamatane,
      na ktorej pozicii plochy
      sa prave nachadza. Cislo N * N
      oznacuje dieru. }
  a : array[1..MaxN * MaxN] of integer;
  N, PocetDavok, NN : integer;
  i, j, davka, pom : integer;

begin
  assign(fin, 'hra.in');
  reset(fin);
  assign(fout, 'hra.out');
  rewrite(fout);
  readln(fin, PocetDavok);
  for davka := 1 to PocetDavok do begin
    readln(fin, N);
    NN := N * N;
    for i := 1 to NN do a[i] := i;
    read(fin, i);
    while i > 0 do begin
      { Cislo sa da posunut,  ak susedi
        s dierou vodorovne al. zvislo }
      if abs(a[i] - a[NN]) in [1, N] then begin
        { Cislo a diera sa vymenia }
        pom := a[i];
        a[i] := a[NN];
        a[NN] := pom;
        read(fin, i)
      end
      else begin
        { Neda sa posunut  koncime }
        readln(fin);
        i := -1
      end
    end;
    if i < 0 then
      write(fout, 'NE');
    writeln(fout, 'DA SA')
  end;
  close(fin);
  close(fout)
end.
```

2.2.2 Kalendár

Pri riešení tohoto príkladu si bolo treba uvedomiť, že každý štvrtý rok v uvažovanom období je priestupný. Očíslujme si jednotlivé dni od pondelku do nedele číslami od 0 po 6.

V prvom rade bolo treba spočítať, ktorý bol posledný deň predchádzajúceho roku. Vieme, že každý rok má 365 dní s výnimkou priestupných, ktoré majú 366 dní. Keďže 1.1.1901 bol utorok, vypočítame číslo tohoto dňa vzťahom $((\text{rok}-1)\times 365 + ((\text{rok}-1) \text{ div } 4)) \bmod 7$. Potom už len stačí pripočítať počet dní predchádzajúcich mesiacov a počet dní, ktoré ubehli z mesiaca, ktorý počítame. Výsledkom bude poradové číslo dňa v roku plus číslo dňa 1. januára. Poradové číslo dňa teraz spočítame mod 7 a týždeň div 7 a pripočítaním jednotky.

```
program kalendar;
const
  dni : array [0..6] of string [10]=
         (' PONDELOK', ' UTOROK',
          'A STREDA', ' STVRTOK', ' PIATOK',
          'A SOBOTA', 'A NEDELA');
  mesiace : array [1..12, 1..2] of integer =
              ((0, 0), (31, 0), (59, 1),
               (90, 1), (120, 1), (151, 1),
               (181, 1), (212, 1), (243, 1),
               (273, 1), (304, 1), (334, 1));
var
  ch1, ch2 : char;
  den, mesiac, rok, q : longint;
  inp, out : text;

begin
  assign(inp, 'KALENDAR.IN'); reset(inp);
  assign(out, 'KALENDAR.OUT'); rewrite(out);
  repeat
    {nacitanie}
    read(inp, ch1, ch2);
    if ch1 <> '-' then begin
      den := (ord(ch1) - ord('0')) * 10 +
             ord(ch2) - ord('0');
    read(inp, ch1, ch2);
    mesiac := (ord(ch1) - ord('0')) * 10 +
              ord(ch2) - ord('0');
    read(inp, ch1, ch2); read(inp, ch1, ch2);
    rok := (ord(ch1) - ord('0')) * 10 +
           ord(ch2) - ord('0');
    readln(inp);
    {samotny vypocet}
    q := ((rok - 1) * 365 +
          ((rok - 1) div 4)) mod 7;
    {posledny den predch. roku
     1.1.1901 bol utorok}
    q := q + mesiace[mesiac, 1] + den;
    if (rok mod 4) = 0 then
      {osetrenie priestupnych rokov}
      q := q + mesiace[mesiac, 2];
    if rok < 10 then
      writeln(out, den, '.', mesiac, '.190',
                   rok, ' BOL', dni[q mod 7],
                   ' V ', (q div 7)+1,
                   '. TYZDNI')
    else
      writeln(out, den, '.', mesiac, '.19',
                   rok, ' BOL', dni[q mod 7],
                   ' V ', (q div 7)+1,
```

```
                    '. TYZDNI');
    end;
  until ch1 = '-';
  close(inp);
  close(out);
end.
```

2.2.3 Koláč

Program riešiaci danú úlohu si pre každé políčko spočíta počet hrozienok, ktoré sa nachádzajú v obdĺžniku od ľavého horného rohu koláča po toto políčko. Toto urobíme v dvoch krokoch, najprv napočítame pre každé políčko počet hrozienok v riadku, v ktorom sa nachádza od ľavého okraja koláča po dané políčko. Po druhom kroku sa v $A[i, j]$ nachádza počet hrozienok nachádzajúcich sa v obdĺžniku od ľavého horného rohu po políčko $[i, j]$ a pomocou tejto informácie môžme rýchlo vypočítať obsah hociktorého štvorca $N \times N$ s ľavým horným políčkom so súradnicami x, y ako $A[x + N - 1, y + N - 1] + A[x - 1, y - 1] - A[X + N - 1, y - 1] - A[x - 1, y + N - 1]$. Program vypočíta počet hrozienok pre všetky možné štvorce $N \times N$ a vypíše tú najlepšiu možnosť.

```
program Kolac;
const
  Maxn=50;
var
  A : Array[0..2 * Maxn,
            0..2 * Maxn] of Integer;
  i, j, k, n, x, y, h : Integer;
  Fin, Fout : Text;
  Vstupov : integer;
  {Najlepsie Riesenie}
  Best, BestX, BestY : Integer;

begin
  assign(Fin, 'kolac.in');
  reset(Fin);
  assign(Fout, 'kolac.out');
  rewrite(Fout);
  readln(Fin, Vstupov);
  for i := 1 to Vstupov do begin
    readln(Fin, n, k);
    for x := 0 to 2 * n do
      for y := 0 to 2 * n do
        A[x, y]:=0;
    for j := 1 to k do begin
      readln(Fin, x, y);
      A[x, y] := A[x, y] + 1;
    end;
  {Napocitame pocet hrozienok v stvorci
   od laveho horneho rohu po dane policko}
    for y := 1 to 2 * n do
      for x := 1 to 2 * n do
        A[x, y] := A[x, y] + A[x - 1, y];
    for x := 1 to 2 * n do
      for y := 1 to 2 * n do
        A[x, y] := A[x, y] + A[x, y - 1];
    Best := -1;
    for y := 1 to n + 1 do
      for x := 1 to n + 1 do begin
        h := A[x + n - 1, y + n - 1] +
             A[x - 1, y - 1] -
             A[x + n - 1, y - 1] -
             A[x - 1, y + n - 1];
        If h > Best then begin
          Best := h;
          BestX := x;
          BestY := y;
        end;
      end;
    writeln(Fout, BestX, ' ', BestY);
  end;
  close(Fin); close(Fout);
end.
```

2.2.4 Mince

Uvedomme si, že v americkej peňažnej sústave platí, že každá nižšia minca je aspoň dvakrát menšia ako každá vyššia minca. Z toho ale vyplýva, že keď chceme nejaký plat vyplatiť najmenším počtom platidiel, stačí postupovať od najväčších bankoviek a vyplatiť vždy ich najväčšie možné množstvo (to by neplatilo, ak by sme v našej peňažnej sústave mali trebárs ešte dvadsaťpäťdolárovku, to by totiž potom 40 vyplatil náš algoritmus spôsobom 25+10+5, čo je viac platidiel ako 20+20).

```
program mince;
const
  pocminci = 11;
  hodnoty : array [1..pocminci] of longint =
    (10000, 5000, 2000, 1000, 500, 100,
      50, 25, 10, 5, 1);
var
  pocet : array [1..pocminci] of integer;
  suma : longint;
  vstup : real;
  z, i, poczad : integer;
  inp, out : text;
  ch : string[1];

begin
  assign(inp, 'MINCE.IN'); reset(inp);
  assign(out, 'MINCE.OUT'); rewrite(out);
  readln(inp, poczad);
  for z := 1 to poczad do begin
    {inicializacia pola}
    for i := 1 to pocminci do pocet[i] := 0;
    {vypocet}
    readln(inp, vstup);
    suma := trunc(vstup * 100);
    while suma >= 0 do begin
      for i := 1 to pocminci do begin
```

```
      inc(pocet[i], suma div hodnoty[i]);
      suma := suma mod hodnoty[i];
    end;
    readln(inp, vstup);
    suma := trunc(vstup * 100);
  end;
  {vypis}
  ch := '';
  for i := 1 to pocminci do begin
    write(out, ch, pocet[i]);
    ch := ' ';
  end;
  writeln(out);
end;
close(inp); close(out);
end.
```

2.2.5 Obchod

Algoritmus pracuje podobným spôsobom ako quicksort. Vezme prvého zákazníka a ostatných rozdelí na dve skupiny: tých, čo na konci dňa budú pred ním a tých, čo budú za ním. Pre obidve tieto skupiny sa potom rekurzívne vykoná tá istá činnosť. Na začiatku je každý človek v rade charakterizovaný dvomi číslami: poradovým číslom (podľa príchodu) a na ktoré miesto sa zaradí pri svojom príchode. Tieto údaje sú uložené v poli A v tom poradí, ako ľudia prichádzali k obchodu. Pole A sa potom postupne preusporiada tak, že na konci budú ľudia v tom poradí, ako zostanú stáť zvyšok dňa. Pri rozdeľovaní poľa na vyššie popísané skupiny postupne prechádzame poľom a počítame umiestnenie prvého človeka. Každého zákazníka, ktorý sa zaradí pred prvého zaradíme do prvej skupiny a prvého "posunieme" (zvýšime jeho aktuálne umiestnenie). Ak sa zákazník zaradí za prvého, zaradí sa do druhej skupiny. Namiesto miesta v celom rade, na ktoré sa zaradí pri svojom príchode mu priradíme číslo miesta v druhej časti radu, t.j. o koľko miest sa zaradí za prvého. V obidvoch častiach radu treba zachovať relatívne poradie ľudí. Preto nemôžeme prvky jednoducho vymieňať, ako sa to deje pri quicksorte. Ľudí z prvej skupiny ukladáme do začiatku poľa A, za nich sa uloží prvý človek v rade, ľudí z druhej skupiny ukladáme pri prechode poľom do pomocného poľa a potom ich presunieme späť do druhej časti poľa A.

```
program RuskyObchod;
const
  max = 1000;
type
  clovek = record
    kam, c : integer;
  end;
var
  fi, fo : text;
  prikladov, n, i, j : longint;
  a, pom : array[1..max] of clovek;
  por, vacsich : integer;
  c : clovek;

procedure rad(z, k : integer);
var
  i : integer;
begin
  if z < k then begin
    {este je co preusporiadat}
    {odlozime prveho z useku}
    c := a[z];
    por := 1;
    vacsich := 0;
    for i := z + 1 to k do
      {pre vsetkych ludi v useku}
      if a[i].kam <= por then begin
        {je pred prvym}
        a[z + por - 1] := a[i];
        inc(por);
        end else begin
        {je za prvym}
        inc(vacsich);
        a[i].kam := a[i].kam - por;
        {kolko je za prvym}
        pom[vacsich] := a[i];
      end;
    vacsich := z + por - 1;
    a[vacsich] := c;
    {prveho vloz na spravne miesto}
    for i := z + por to k do
      a[i] := pom[i - vacsich];
    i := vacsich;
    rad(z, i - 1);
    {uprac obidva useky}
    rad(i + 1, k);
  end;
end;

begin
  assign(fi, 'obchod.in');
  assign(fo, 'obchod.out');
  reset(fi);
  rewrite(fo);
  reset(fi);
  readln(fi, prikladov);
  for i := 1 to prikladov do begin
    readln(fi, n);
    for j := 1 to n do
      with a[j] do begin
        read(fi, kam);
        kam := j - kam;
        c := j;
      end;
    rad(1, n);
    for j := 1 to n do
      pom[a[j].c].c := j;
    write(fo, pom[1].c);
    for j := 2 to n do
      write(fo, ' ', pom[j].c);
    writeln(fo);
  end;
  close(fo);
end.
```

2.2.6 Stroj

Program potrebuje dva prechody na každý vstup. V poli `a` sú zapamätané súradnice pôvodných a posunutých bodov súčiastky.

V prvom prechode sa určia posunuté body v rohoch. Uvažujme hocaký roh (t.j. keď dve susedné strany neležia na jednej priamke). Označme jeho vrcholy ABC. Vo výstupe bude namiesto bodu B bod X. Zrejme bude ležať na priamke rovnobežnej s AB vo vzdialenosti polovičnej šírky plameňa "zvonka" súčiastky (označme ju p) a na rovnakej priamke pre stranu BC (označme ju q). Keď teda nájdeme tieto priamky, bod X sa ľahko zráta ako ich priesečník. Ako nájsť priamku p? Veďme z bodu A kolmicu na vektor $\mathbf{B} - \mathbf{A}$ (to je jednoduché, stačí si uvedomiť, že ak $\mathbf{u} = (u_1, u_2)$, $\mathbf{n} = (-u_2, u_1)$, potom $\mathbf{u}$ je kolmé na $\mathbf{n}$). Na nej vo vzdialenosti polovičky šírky plameňa bude ležať bod P (to sa opäť zrealizuje ľahko, ak do vzorca pre vzdialenosť AP dosadíme súradnice $P = A + k\mathbf{n}$, kde $\mathbf{n}$ je vektor kolmý na $\mathbf{B} - \mathbf{A}$, dostaneme kvadratickú rovnicu z ktorej vyjadríme k). Najväčší problém je zistiť, ktorá strana je "zvonka" (t.j., ktoré z dvoch riešení kvadratickej rovnice treba zobrať). Tu sa dá využiť fakt, že trojuholníky ABC a ABP musia mať opačnú orientáciu (orientácia trojuholníka ABC je znamienko výrazu $a_1b_2 - c_1b_2 + a_2c_1 - c_2a_1 + b_1c_2 - b_1a_2$). Takže máme bod P. Priamka p má potom tvar $X = P + k(\mathbf{B} - \mathbf{A})$. Rovnakým spôsobom nájdeme priamku q. Zrátať priesečník p a q znamená vlastne riešiť sústavu dvoch rovníc o dvoch neznámych - v programe je použitá metóda determinantov.

V druhom prechode sa pre zvyšné body dopočítajú posunuté body tak, že sa zráta priesečník priamky kolmej na príslušnú stranu v danom bode a priamky rovnobežnej s príslušnou stranou a prechádzajúcou posunutým bodom ľavého suseda.

```
program NC_stroj;
const
  MaxN = 1000;
  nic = 1e-10;
type
  bod = array[1..2] of real;
var
  fin, fout : text;
  PocetDavok : integer;
  davka, i, Z_abc : integer;
  { A prvom stlpci su povodne body,
    v druhom posunute.
    a[0]=a[N],  a[N+1]=a[0] }
  a : array[0..MaxN + 1, 1..2] of bod;
  N : integer;
  { hrubka plamena  :  }
  d : real;
  c : char;
  p, q, u, v, nv : bod;
  r : real;

procedure Urob_vektory(var u, v : bod;
                       i : integer);
{pre trojicu bodov i, i+1, i+2 zrata
 vektory stran suciastky}
begin
  u[1] := a[i+1, 1, 1] - a[i, 1, 1];
  u[2] := a[i+1, 1, 2] - a[i, 1, 2];
  v[1] := a[i+2, 1, 1] - a[i+1, 1, 1];
  v[2] := a[i+2, 1, 2] - a[i+1, 1, 2]
end;

function Nie_na_priamke(u, v : bod) : boolean;
{vrati false, ak jeden vektor
je nasobkom druheho}
begin
  Nie_na_priamke :=
    not((u[1] = 0) and (v[1] = 0) or
        (u[1] <> 0) and
        (abs(v[2] - v[1] * u[2] / u[1]) < nic))
end;

function Smer(a, b, c : bod) : integer;
{vrati znamienko determinantu
 trojuholnika ABC,
 ktore udava jeho orientaciu.}
begin
 if (a[1] * b[2] - c[1] * b[2] +
     a[2] * c[1] - c[2] * a[1] +
     b[1] * c[2] - b[1] * a[2]) > 0
   then Smer := 1
   else Smer := -1
end;

procedure Zrataj_bod(var p : bod;
                     a, b, u : bod;
                     sm : integer);
{zrata bod p,  ktory lezi zvonka
 suciastky pri bode a }
var
  n : bod;
  k : real;
begin
  {kolmy vektor}
  n[1] := -u[2];
  n[2] := u[1];
  k := d / sqrt(sqr(n[1]) + sqr(n[2]));
  p[1] := a[1] + k * n[1];
  p[2] := a[2] + k * n[2];
  if Smer(a, b, p) = sm then begin
    p[1] := a[1] - k * n[1];
    p[2] := a[2] - k * n[2]
  end
end;

begin
  assign(fin, 'stroj.in');
  reset(fin);
  assign(fout, 'stroj.out');
  rewrite(fout);
  readln(fin, PocetDavok);
  for davka := 1 to PocetDavok do begin
    readln(fin, d);
    d := d / 2;
    N := 0;
    read(fin, c);
```

```
  while c = 'M' do begin
    inc(N);
    readln(fin, a[N, 1, 1], a[N, 1, 2]);
    read(fin, c)
  end;
  readln(fin);
  {prvy prechod }
  a[0] := a[N];
  a[N+1] := a[1];
  for i := 0 to N-1 do begin
    Urob_vektory(u, v, i);
    if Nie_na_priamke(u, v) then begin
      Z_abc := Smer(a[i, 1], a[i+1, 1],
                    a[i+2, 1]);
      Zrataj_bod(p, a[i, 1], a[i+1, 1],
                 u, Z_abc);
      Zrataj_bod(q, a[i+1, 1], a[i+2, 1],
                 v, Z_abc);
      r := ((q[2] - p[2]) * v[1] -
            (q[1] - p[1]) * v[2]) /
           (u[2] * v[1] - u[1] * v[2]);
      a[i + 1, 2, 1] := p[1] + r * u[1];
      a[i + 1, 2, 2] := p[2] + r * u[2]
    end
  end;
  {druhy prechod }
  a[0] := a[N];
  a[N + 1] := a[1];
  for i := 0 to N - 1 do begin
    Urob_vektory(u, v, i);
    if Nie_na_priamke(u, v) then
      writeln(fout, 'M ',
        a[i + 1, 2, 1] : 0 : 2,
        ' ', a[i+1, 2, 2] : 0 : 2)
    else begin
      nv[1] := -u[2];
      nv[2] := u[1];
      r := ((a[i, 2, 2] -
             a[i + 1, 1, 2]) * u[1] -
            (a[i+1, 2, 1] - a[i, 1, 1]) *
            u[2]) /
           (nv[2] * u[1] - nv[1] * u[2]);
      a[i+1, 2, 1] :=
        a[i+1, 1, 1] + r * nv[1];
      a[i+1, 2, 2] :=
        a[i+1, 1, 2] + r * nv[2];
      writeln(fout, 'M ',
        a[i+1, 2, 1] : 0 : 2, ' ',
        a[i+1, 2, 2] : 0 : 2)
    end
  end;
  writeln(fout, 'E')
end;
close(fin);
close(fout)
end.
```

2.2.7 Váhy

Pre modré aj červené závažia si vypočítame, aké hmotnosti do 1000 sa dajú dosiahnuť použitím závaží jednej farby. Toto budeme robiť napríklad pre modré závažia nasledovne: Hmotnosť 0 sa dá dosiahnuť. Ak vieme, či sa dajú dosiahnuť hmotnosti 0 až k, potom hmotnosť $k+1$ sa dá dosiahnuť práve vtedy, ak existuje modré závažie $M[i]$ také, že hmotnosť $k+1-M[i]$ je dosiahnuteľná. To, či takéto závažie existuje, overíme jednoducho tak, že ich všetky vyskúšame. Keď máme dosiahnuteľné hmotnosti pre modré a červené závažia do 1000, potom jednoducho zistíme, či existuje dosiahnuteľná hmotnosť pomocou modrých a červených závaží. Ak áno, potom je Ignácova úloha splniteľná, inak nie.

```
program Vahy;
const
  MaxHmot = 1000;
  MaxZavazi = 40;
var
  Fin, Fout : Text;
  Vstupov : integer;
  {Hmotnosti Zavazi}
  M, C : array[1..MaxZavazi] of integer;
  {Dosiahnutelne Sucty}
  SM, SC : array[0..MaxHmot] of boolean;
  {Pocet Modrych/Cervenych zavazi}
  PM, PC : integer;
  i, j, k : integer;

begin
  Assign(Fin, 'vahy.in');
  Reset(Fin);
  Assign(Fout, 'vahy.out');
  Rewrite(Fout);
  Readln(Fin, Vstupov);
  for i := 1 to Vstupov do begin
    Readln(Fin, PM, PC);
    {Nacitame hmotnosti modrych zavazi}
    for j := 1 to PM do
      Readln(Fin, M[j]);
    {Nacitame hmotnosti cervnych zavazi}
    for j := 1 to PC do
      Readln(Fin, C[j]);
    {Vypocitame Sucty dosiahnutelne
     pomocou modrych zavazi}
    for j := 1 to MaxHmot do
      SM[j] := False;
    SM[0] := True;
    for j := 1 to MaxHmot do
      for k := 1 to PM do
        if (j - M[k] >= 0) then
          if SM[j - M[k]] then
            SM[j] := True;
    {Vypocitame Sucty dosiahnutelne
     pomocou cervenych zavazi}
    for j := 1 to MaxHmot do
      SC[j] := False;
    SC[0] := True;
    for j := 1 to MaxHmot do
      for k := 1 to PC do
        if (j - C[k] >= 0) then
          if SC[j - C[k]] then
            SC[j] := True;
    {Zistime, ci maju spolocnu hmotnost}
    while (k <= MaxHmot) and
          not (SM[k] and SC[k]) do
      k := k + 1;
    if (k <= MaxHmot) then
      Writeln(fout, 'ANO')
    else
```

```
      Writeln(fout, 'NIE');
  end;
```

```
  Close(Fin); Close(Fout);
end.
```

3.1 Domáce kolo

3.1.1 Interval

Tento príklad ste riešili štyrmi spôsobmi. V stručnosti uvedieme hlavnú myšlienku a maximálne bodové ohodnotenie príslušného riešenia. Asi najmenej šikovné bolo riešenie založené na postupnom zjednocovaní intervalov. Na začiatku sa zvolí jeden interval a zistí sa, či niektorý zo zvyšných intervalov nemá s daným intervalom prienik. Ak má, zjednotia sa do jedného a pôvodné intervaly sa ďalej neuvažujú. Ak žiadny zo zvyšných intervalov s daným intervalom nemá prienik, súvislú službu sa nepodarí zabezpečiť. Keď nám ostane iba jeden interval, súvislú službu sa podarí zabezpečiť. Takéto riešenie vyžaduje v najhoršom prípade až $O(n^2)$ operácií, lebo v každom kroku sa nám môže podariť zjednotiť iba dva intervaly. Krokov zjednotenia môže byť až $n-1$ (6 bodov).

Šikovnejšie bolo utriediť intervaly podľa počiatočného času. Potom vieme, ktorý interval je prvý a treba iba preveriť, či intervaly na seba nadväzujú. Tu ste sa dali mnohí nachytať a kontrolovali ste iba či koniec i-teho intervalu je neskôr ako začiatok $i+1$-vého. To nestačí lebo napr. 2. interval v poradí môže končiť až za n-tým a 3. so 4. sa nemusia vôbec prekrývať (2 body). V správnom riešení si treba pamätať doterajší maximálny čas pokiaľ trvá súvislá služba a tento sa snažiť predlžovať. Zložitosť tohoto riešenia je podľa typu triedenia (6–8 bodov).

Mnohí ste si všimli, že deň má 1440 minút a použili pole zodpovedajúce minútam (vzhľadom na maximálny počet intervalov 9999, pole veľkosti 1440 je úplne akceptovateľné). Pri čítaní intervalov jednoducho do prvkov poľa, ktoré zodpovedajú minútam intervalu napíšeme napr. *true*. Potom stačí zistiť, či je oblasť s hodnotami *true* súvislá. Takýto algoritmus je veľmi jednoduchý a jeho zložitosť závisí od celkového súčtu dĺžok všetkých intervalov (8 bodov). Zopár z Vás si všimlo, že sa dá ušetriť aj zapisovanie *true* do celého intervalu. Uvedomte si, že stačí keď si počítame v koľkých intervaloch sa práve nachádzame. To sa dá jednoducho, na začiatok intervalu dáme číslo 1 a na koniec -1. Teda k prvku zodpovedajúcemu minúte, v ktorej začína interval pripočítame 1 a od prvku, zodpovedajúcemu koncu zase odčítame 1. Na začiatku sú všade 0. Teraz stačí postupne rátať súčet a kontrolovať, či neklesne na 0 ešte pred koncom posledného intervalu. V najhoršom prípade bude počet operácií lineárne závisieť od veľkosti vstupu (10 bodov).

Na záver ešte spomenieme, že bodové ohodnotenie neovplyvnilo považovanie intervalov typu `1000 1005` a `1006 1010` za súvislý interval alebo nie.

```
program interval;
var
  n, x, i, Prichod, KolkoSluziebSucasne,
  PrvyPrichod, PoslednyPrichod : integer;
  f, g : text;
  Cas : array[0..1439] of integer;

function Minuty(x : integer) : integer;
begin
  Minuty := (x div 100) * 60 + x mod 100
end;

begin
   assign(f, 'interval.in'); reset(f);
   assign(g, 'interval.out'); rewrite(g);
   for i := 0 to 1439 do Cas[i] := 0;
   PrvyPrichod := 1440;
   PoslednyPrichod := 0;
   read(f, n);
   while n > 0 do begin
     for i := 1 to n do begin
       Read(f, x);
       Prichod := Minuty(x);
       if Prichod < PrvyPrichod then
         PrvyPrichod := Prichod
       else if Prichod > PoslednyPrichod then
         PoslednyPrichod := Prichod;
       inc(Cas[Prichod]);
       Read(f, x);
       dec(Cas[Minuty(x)]);
     end;
     KolkoSluziebSucasne := Cas[PrvyPrichod];
     Prichod := PrvyPrichod;
     while KolkoSluziebSucasne > 0 do begin
       inc(Prichod);
       inc(KolkoSluziebSucasne, Cas[Prichod])
     end;
     if Prichod < PoslednyPrichod then
       write(g, 'NE');
     writeln(g, 'PODARI');
     read(f, n)
   end;
   close(f); close(g)
end.
```

3.1.2 Šifra

Na políčko papiera so súradnicami $[x, y]$ sa pri rôznom natočení mriežky môžu dostať políčka mriežky

so súradnicami $[x, y], [2n - y + 1, x], [2n - x + 1, 2n - y + 1], [y, 2n - x + 1]$. Ak je mriežka nekorektná, musia existovať súradnice $[x, y]$ také, že viac ako jedno z týchto štyroch políčok mriežky bude vystrihnuté. Ak je neúplná, musia existovať súradnice $[x, y]$ také, že žiadne z týchto štyroch políčok mriežky nie je vystrihnuté. Aby bola mriežka korektná a úplná, musí pre všetky súradnice $[x, y]$ platiť, že práve jedno z políčok $[x, y], [2n - y + 1, x], [2n - x + 1, 2n - y + 1], [y, 2n - x + 1]$ je vystrihnuté. Nad políčko papiera so súradnicami $[x, y]$ sa však môžu dostať tie isté políčka mriežky ako nad políčka so súradnicami $[2n - y + 1, x], [2n - x + 1, 2n - y + 1], [y, 2n - x + 1]$. Preto, ak sa nad ľubovoľné z týchto štyroch políčok papiera môže dostať práve jeden vystrihnutý štvorček mriežky, aj pre ostatné tri to platí (a nemusíme to teda už kontrolovať). Keď si rozdelíme papier na štyri štvorce rozmerov $n \times n$, každé z týchto štyroch políčok bude v inom štvorci. Teda nám stačí skontrolovať iba jeden takýto štvorec (napr. ľavý horný).

V programe je štvrtina papiera reprezentovaná maticou a. Pred načítaním údajov sa celá vymaže (nastaví na *false*). Pri načítaní každej diery sa zistí, aká bude jej poloha, keď sa mriežka natočí tak, aby táto diera bola v ľavom hornom štvorci. Na tieto súradnice sa do poľa a zapíše *true*. Ak tam už predtým bolo *true*, mriežka je nekorektná, lebo sa dá dvakrát zapisovať na to isté miesto. Nakoniec treba ešte skontrolovať, či sa dalo písať na každé miesto. To sa v programe realizuje počítadlom dier, ktoré sa zvýši s každou dierou. Ak je mriežka korektná a dier je n^2, je aj úplná. Tento algoritmus má tú výhodu, že netreba ukladať do pamäti celú mriežku, ale iba jej štvrtinu. Za takéto riešenie ste mohli získať plný počet, t.j. 10 bodov. Za riešenia s časovou zložitosťou horšou ako $O(n^2)$ sa strhávali 4 body, za použitie jedného alebo viacerých polí veľkosti $2n \times 2n$ sa strhávali 1 až 2 body. Za chýbajúce príklady vstupu a výstupu ste mohli stratiť 1 bod, za chýbajúci alebo nedostatočný popis ďalšie 2 body.

```
program sifra;
const
  maxn = 120;
var
  a : array[1..maxn, 1..maxn] of boolean;
        {lava horna stvrtina papiera}
  n, dier : integer;
  i, j : integer;
  fin, fout : text;
  korektna : boolean;
  z : char;

procedure pis(i, j : integer);
begin
  {skontroluje, ci na suradniciach
   i,j nebola diera a zapise ju tam}
  if a[i,j] then korektna := false
  else begin
    a[i,j] := true;
  end;
end;

begin
  {otvorenie suborov}
  assign(fin, 'sifra.in');
  reset(fin);
  assign(fout, 'sifra.out');
  rewrite(fout);
  readln(fin, n);
  while n > 0 do begin
    {inicializacia premennych}
    for i := 1 to n do
      for j := 1 to n do
        a[i, j] := false;
    dier := 0;
    korektna := true;
    {spracovanie mriezky}
    for i := 1 to 2 * n do begin
      for j := 1 to 2 * n do begin
        read(fin, z);
        {ak je to diera...}
        if z = '.' then begin
          inc(dier);
          {treba otocit dieru
          do lavej hornej casti papiera
          a poznacit si to do pola}
          if i <= n then
            if j <= n then
              {lava horna cast mriezky}
              pis(i, j)
            else
              {prava horna cast mriezky}
              pis(2 * n + 1 - j, i)
          else if j <= n then
            {lava dolna cast mriezky}
            pis(j, 2 * n + 1 - i)
          else
            {prava dolna cast mriezky}
            pis(2 * n + 1 - i, 2 * n + 1 - j)
        end
      end;
      readln(fin)
    end;
    {vypis vysledku}
    if korektna and (dier = n * n) then
      writeln(fout, 'ANO')
    else
      writeln(fout, 'NIE');
    readln(fin, n)
  end;
  close(fin); close(fout)
end.
```

3.1.3 Lístky

Riešenie tohoto príkladu bolo založené na jedinej myšlienke, a to že lístku nevadí, keď mu strojček predieruje už predierované políčko. Je totiž iba šesť možností ako zasunúť lístok do strojčeka – tri povytiahnutia

a dve otočenia (mnohí ste ho otáčali aj všakovako do strán, ale skúste to niekedy urobiť s normálnym lístkom – nejde to). Niektorí z vás sa ale uspokojili so skúšaním všetkých 2^6 postupností najviac šesť cviknutí (na to, že nemá zmysel cviknúť dvakrát rovnako ste prišli všetci). Pritom stačí prejsť tých šesť možností a o každej si zistiť, či neurobí dierku tam, kde nemá (t.j. porovnáme ju políčko po políčku so vzorovým lístkom). Ak nepokazí lístok, cvikneme si ju na pokusný papierik tvaru lístka (ten máme stále rovnaký a na začiatku tam nie je žiadna dierka). Keď takýmto spôsobom prejdeme všetky možnosti, máme na pokusnom papieriku najviac dierok, koľko sa len dá. A teraz stačí porovnať papierik so vzorom (zrejme na papieriku nebudú dierky naviac, lebo to sme kontrolovali priebežne). Ak je na papieriku vydierovaný vzor – super. Ak nie, tak tam nejaké dierky chýbajú. Tie tam ale nemôžme nijako pridať, lebo každá možnosť, ktorú sme necvikli do papierika by pridala dierku tam, kam nepatrí. Preto – nedá sa to.

Realizácia ponúkala viacero možností. Tu je riešenie bez rôznych zefektívnení, tie sú totiž väčšinou na úkor čitateľnosti. Tu v cykle skúšam všetky možnosti a kontrolujem, či sa môže cviknúť. Ak áno, cviknem si papierik a na konci porovnám:

```
program Listky;
type
  Listok = array[1..3, 1..3] of boolean;
     {true, ak je na danom mieste diera}
var
  Strojcek,
  Vysl_listok,
  Moj_listok : Listok;
  vysun, otoc : integer;
  f, g : text;

procedure Citaj(var L : Listok);
var
  i, j : integer;
  c : char;
begin
  for i := 1 to 3 do begin
    for j := 1 to 3 do begin
      read(f, c);
      L[i,j] := (c = 'X')
    end;
    readln(f);
  end
end;

function Mozem_cviknut(S, L : Listok;
            vysun, otoc : integer) : boolean;
var
  i, j : integer;
begin
  for i := vysun to 3 do
    for j := 1 to 3 do
     if S[i - vysun + 1,
          otoc * (4 - 2 * j) + j] and
        not L[i, otoc * (4 - 2 * j) + j]
      then begin
        Mozem_cviknut := false;
        exit
      end;
  Mozem_cviknut := true
end;

procedure Cvikni(var S, L : Listok;
                 vysun, otoc : integer);
var
  i, j : integer;
begin
  for i := vysun to 3 do
    for j := 1 to 3 do
     if S[i - vysun + 1,
          otoc * (4 - 2 * j) + j] then
      L[i, otoc * (4 - 2 * j) + j] := true
end;

function Su_rovnake(S, T : Listok) : boolean;
var
  i , j : integer;
begin
  for i := 1 to 3 do
    for j := 1 to 3 do
      if S[i, j] <> T[i, j] then begin
        Su_rovnake := false;
        exit
      end;
  Su_rovnake := true
end;

begin
  assign(f, 'pokus.in');
  reset(f);
  assign(g, 'pokus.out');
  rewrite(g);
  while not(eof(f)) do begin
    Citaj(Strojcek);
    Citaj(Vysl_listok);
    for vysun := 1 to 3 do
      for otoc := 0 to 1 do
       if Mozem_cviknut(Strojcek, Vysl_listok,
                        vysun, otoc)
       then
         Cvikni(Strojcek, Moj_listok,
                vysun, otoc);
    if not Su_rovnake(Vysl_listok, Moj_listok)
    then write(g, 'NE');
    writeln(g, 'DA SA')
  end;
  close(f); close(g)
end.
```

Vylepšenia môžu byť v tom, že najvyšší riadok lístka sa musí cviknúť pri plnom zasunutí a druhý riadok najneskôr pri jednom povytiahnutí. Teda keď vyskúšam plné zasunutie, skontrolujem najvyšší riadok a keď nesedí, ani ďalej nepokračujem.

Ďalšia možnosť je pamätať si lístky nie v poli ale v jedinom čísle ako hodnoty jednotlivých bitov deväťbitového čísla. Potom môžeme napríklad kontrolu urobiť jedinou inštrukciou. Ale to už sú technické triky a trochu zahmlievajú zrozumiteľnosť.

3.1.4 Robot

Úvodom je potrebné povedať, že tento príklad nebol veľmi šťastne zadaný. Mnohí z Vás sa dali popliesť intuitívnou predstavou pojmu konvexná čiara a neprečítali si poriadne jeho definíciu. Títo potom riešili ľahšiu úlohu a my sme im museli udeliť 0 bodov (aj napriek nešťastne zvolenému názvu, zadanie bolo napísané jednoznačne). Pre jednoznačnosť: našou úlohou je vlastne zistiť, či body, v ktorých sa robot otáčal, sú vrcholmi nejakého konvexného N-uholníka.

Body sme rozdeľovali medzi funkčnosť programu (2 body), časovú efektívnosť algoritmu (maximálne 5 bodov za algoritmus $O(n \log_2 n)$), popis (maximálne 2 body) a príklady vstupu a výstupu (1 bod). Niektoré vaše riešenia boli založené na tvrdeniach o konvexnosti mnohouholníkov, ktoré boli buď nutnou, alebo postačujúcou podmienkou pre konvexnosť mnohouholníka, ale nie nutnou a postačujúcou súčasne. Takéto algoritmy potom o niektorých konvexných N-uholníkoch povedia, že konvexné nie sú, alebo o niektorých nekonvexných, že sú konvexné. Za takéto algoritmy sme nemohli priznať body.

Vzorové riešenie sa zakladá na nasledujúcej myšlienke: vezmime si taký bod, ktorý má najmenšiu y-ovú súradnicu zo všetkých bodov, označme ho A. Zotrieďme teraz ostatné body podľa uhlov, ktoré zvierajú s x-ovou osou vzhľadom k bodu A v protismere hodinových ručičiek. Ak body môžu byť vrcholmi nejakého konvexného N-uholníka, potom týmto usporiadaním dostaneme ich poradie po obvode takéhoto N-uholníka v protismere hodinových ručičiek. V opačnom prípade dostaneme nekonvexný N-uholník. Teraz už teda treba iba zistiť, či vytvorený N-uholník je konvexný. Na to je ľahká pomoc, stačí zistiť, či pre každú trojicu za sebou idúcich vrcholov i, $i+1$, $i+2$ leží bod $i+1$ vpravo od polpriamky určenej i. a $i+2$. vrcholom.

Ešte nám zostáva vyriešiť problém, ako porovnávať uhly, ktoré jednotlivé body zvierajú s osou x vzhľadom k bodu A. Mohli by sme to urobiť tak, že pre každý bod spočítame jemu zodpovedajúci uhol a tieto uhly číselne porovnávame. To má ale nepríjemný dôsledok – musíme použiť reálne čísla. Je to však možné urobiť bez reálnych čísel – pozri funkciu *porovnaj* v programe. To, či bod neleží náhodou vľavo od nejakej polpriamky zistíme pomocou elementárnych vzťahov analytickej geometrie (pozri funkciu *vlavo*).

```
program ROBOT;
const
  max = 1000;  {max.pocet bodov}
var
  x, y : array[1..max] of integer;
  n : integer;
  inp, out : text;

procedure swap(i, j : integer);
var
  c : integer;
begin
  c := x[i]; x[i] := x[j]; x[j] := c;
  c := y[i]; y[i] := y[j]; y[j] := c
end;

function kvadrant(i : integer) : integer;
begin
  if (x[i] > 0) and (y[i] >= 0) then
    kvadrant := 1
  else if (x[i] <= 0) and (y[i] > 0) then
    kvadrant := 2
  else if (x[i] < 0) and (y[i] <= 0) then
    kvadrant := 3
  else
    kvadrant := 4
end;

function porovnaj(i, j : integer) : integer;
var
  p, q, x1, y1, x2, y2, a, b : integer;
begin
  {najprv zistime, v ktorych kvadrantoch
   sa nachadzaju body}
  a := kvadrant(i); b := kvadrant(j);
  if a < b then
    porovnaj := 1
  else if a > b then
    porovnaj := -1
  else begin
    {body su v rovnakom kvadrante}
    if (a = 1) or (a = 3) then begin
      x1 := abs(x[i]); x2 := abs(x[j]);
      y1 := abs(y[i]); y2 := abs(y[j])
    end else begin
      x1 := abs(y[i]); x2 := abs(y[j]);
      y1 := abs(x[i]); y2 := abs(x[j])
    end;
    {test na tangensy uhlov}
    p := y2 * x1; q := y1 * x2;
    if p > q then
      porovnaj := 1
    else if p < q then
      porovnaj := -1
    else
      porovnaj := 0
   end
end;

function vlavo(i, j, k : integer) : boolean;
var
  p : integer;
begin
 vlavo :=
    (x[j] - x[i]) * (y[k] - y[i]) -
    (x[k] - x[i]) * (y[j] - y[i]) > 0
end;

procedure QuickSort(l, r : integer);
{quicksort - triedenie
 vyuziva nasledujuce funkcie
   porovnaj(i,j) - porovna i-ty a j-ty
                   prvok; vysl.
                     +1 - ak i-ty < j-ty
```

```
                     0 - ak i-ty = j-ty
                    -1 - ak i-yu > j-ty
   swap(i,j) - vymeni i-ty a j-ty prvok}

var
  m, n, x : integer;
begin
  m := l; n := r;
  x := (m + n) div 2;
  repeat
    while porovnaj(m, x) = 1 do inc(m);
    while porovnaj(n, x) = -1 do dec(n);
    if m <= n then begin
      swap(m, n);
      inc(m); dec(n)
    end
  until m > n;
  if l < n then QuickSort(l, n);
  if m < r then QuickSort(m, r)
end;

function Nacitanie:boolean;
{nacita jednu sadu vstupnych dat}
begin
  n := 1;
  readln(inp, x[n], y[n]);
  while (x[n] <> 0) or (y[n] <> 0) do begin
    inc(n);
    readln(inp, x[n], y[n])
  end;
  n := n - 2; {posledne nacitane je 0, 0
          a predp. bod je ten isty co prvy}
  Nacitanie := (n > 0)
end;

procedure UrobNuh;
var
  i, min : integer;
begin
  {vyberie bod s min. y-ovou suradnicou;
  podla neho budeme triedit podla polarnych
  suradnic}
  min := 1;
  for i := 2 to n do
    if y[min] > y[i] then min := i;
  {ten ulozi na zaciatok}
  swap(min, 1);
  {a spusti triedenie ostatnych bodov}
  QuickSort(2, n)
end;

function Konvexny : boolean;
var
  i : integer;
begin
  {skopiruje prvy a druhy bod na koniec
   pre jednoduchsie zaobchadzanie}
  x[n + 1] := x[1]; y[n + 1] := y[1];
  x[n + 2] := x[2]; y[n + 2] := y[2];
  {obieha okolo a zistuje pre kazdu trojicu
   bodov i,i + 1,i + 2, ci i + 1 lezi
   napravo od polpriamky i - i + 2}
  Konvexny := true;
  for i := 1 to n do
    if vlavo(i, i + 2, i + 1) then
      Konvexny := false
end;

begin {main}
  assign(inp, 'robot.in'); reset(inp);
  assign(out, 'robot.out'); rewrite(out);
  while Nacitanie do begin
    if n < 4 then writeln(out, 'ANO')
    else begin
      UrobNuh;
      if Konvexny then writeln(out, 'ANO')
                  else writeln(out, 'NIE')
    end;
  end;
  close(inp); close(out)
end.
```

3.1.5 Cesty

Základná myšlienka fungujúceho algoritmu je jednoduchá: Dediny, v ktorých sa zbieha menej ako k ciest zrušíme a tiež zrušíme cesty, ktoré do nich viedli. Mnohí ste si, ale neuvedomili, že týmto nám môžu vzniknúť ďalšie dediny, v ktorých sa zbieha menej ako k ciest a teda rušenie dedín treba opakovať, kým sa v každej dedine zbieha aspoň k ciest, alebo už nemáme žiadne dediny. Takýto algoritmus sa vždy zastaví, lebo po každom kroku nám buď klesne počet nezrušených dedín, alebo je splnená podmienka, že v každej dedine sa zbieha aspoň k ciest. Algoritmus zruší iba nevyhnutný počet dedín. Za takýto algoritmus ste mohli získať najviac 9b. Miernym vylepšením je použitie rekurzívnej procedúry *zruš*, ktorá zruší dedinu a pre každú dedinu, ktorá s ňou mala spoločnú cestu vypočíta nový počet ciest. Ak je tento menší ako k, zavolá procedúru *zruš* pre túto dedinu. Túto procedúru zavoláme pre každú dedinu, v ktorej sa zbieha menej ako k ciest. Za mierne vylepšený algoritmus ste mohli získať 10b.

Teraz k realizácii: Vyskytli sa dva spôsoby reprezentácie ciest. Prvým spôsobom je mať pre každú dedinu zoznam ciest, ktoré z nej vychádzajú. Druhým spôsobom je mať pre každú dvojicu dedín počet ciest, ktoré medzi nimi existujú. Druhý spôsob je v tomto prípade lepší, už len preto, že počet ciest nebol ohraničený. Za použitie horšej reprezentácie ste mohli stratiť 2b. V poli $Cesty[i, j]$ si teda budeme pamätať počet ciest medzi dedinami i a j, v poli $Ciest[i]$ si budeme pamätať počet ciest zbiehajúcich sa v dedine i. Počet ciest zbiehajúcich sa v dedine i sa síce dá zakaždým vypočítať z prvkov poľa $Cesty[i, j]$, ale zhorší to efektívnosť programu a tým zníži Váš bodový zisk o 2b. Dedine, ktorú zrušíme nastavíme $Ciest[i] = 0$. Dediny, ktoré ostanú nezrušené majú potom $Ciest[i] \geq k$. Utriedený výpis dedín, ktoré ostali urobíme jednoducho tak, že postupne pre čísla dedín $1, \ldots, n$ vypíšeme tie, pre ktoré $Ciest[i] \geq k$.

```
program Cofax_Cesty;
const
  Dedin = 100;
var
  Cesty : Array[1..Dedin, 1..Dedin] Of Integer;
  Ciest : Array[1..Dedin] Of Integer;
  Fin, Fout : Text;
  n, k, i, j, l : Integer;

procedure Zrus(x : Integer);
  var
    i : Integer;
  begin
    Ciest[x] := 0;
    for i := 1 to n do
      if (Cesty[x,i] > 0) and
         (Ciest[i] > 0) then begin
        Ciest[i] := Ciest[i] - Cesty[x,i];
        if Ciest[i] < k then Zrus(i);
      end;
  end;

begin
  Assign(Fin, 'cesty.in');
  Reset(Fin);
  Assign(Fout, 'cesty.out');
  Rewrite(Fout);
  Readln(Fin,n,k);
  while n <> 0 do begin
    for i := 1 to n do begin
      Ciest[i] := 0;
      for j := 1 to n Do
        Cesty[i,j] := 0;
    end;
    for i := 1 to n do begin
      Read(Fin, j);
      while not Eoln(Fin) do begin
        Read(Fin, l);
        Cesty[j, l] := Cesty[j, l] + 1;
        Ciest[j] := Ciest[j] + 1;
      end;
    end;
    for i := 1 to n do
      if (Ciest[i] > 0) and
         (Ciest[i] < k) then
        Zrus(i);
    for i := 1 to n do
      if Ciest[i] > 0 then
        Write(Fout, i, ' ');
    Writeln(Fout);
    Readln(Fin, n, k);
  end;
  Close(Fout); Close(Fin);
end.
```

3.2 Finále

3.2.1 Fotograf

V tomto príklade si bolo dôležité uvedomiť, že lokomotívy sú na najkratšom úseku trojkoľajky buď na začiatku (teda v čase 0.00), alebo vtedy, keď sú nejaké lokomotívy na rovnakej súradnici. Toto môžme odôvodniť tak, že keď sa žiadne dve lokomotívy nenachádzajú na rovnakej súradnici, čas nie je 0.00 a pozrieme sa na krajné lokomotívy, potom môžu nastať tri možnosti. Ak sa k sebe približujú potom za maličkú chvíľku (takú malú, že žiadne dve lokomotívy sa nedostanú na rovnakú súradnicu) budú lokomotívy na kratšom úseku. Ak sa vzďaľujú, potom pred maličkou chvíľkou boli lokomotívy na kratšom úseku. A napokon ak idú rovnakou rýchlosťou, potom pred maličkou chvíľkou boli rovnako vzdialené. Ukázali sme teda, že prípady, keď čas nie je 0.00 a žiadne dve lokomotívy nie sú na rovnakej súradnici nemôžu byť riešeniami. Teraz nám stačí zistiť všetky časy, v ktorých sa nejaké dve lokomotívy nachádzajú na rovnakej súradnici (funkcia *Cas*), v týchto a na začiatku vypočítať dĺžku úseku trojkoľajky, na ktorom sa všetky tri lokomotívy nachádzajú (funkcia *Sirka*) a vybrať, kedy bude dĺžka tohto úseku minimálna.

```
program Fotograf;
const
  N = 3;
var
  p, r : Array[1..N] Of Real;
  t, s, mint, mins : Real;
  i, j, k, Vstupov : Integer;
  f, g : Text;

function Sirka(t : Real) : Real;
var
  i, j : Integer;
  max, h : Real;
begin
  max := Abs((p[N - 1] + t * r[N - 1]) -
             (p[N] + t * r[N]));
  for i := 1 to N - 2 do
    for j := i + 1 to N do begin
      h := Abs((p[i] + t * r[i]) -
               (p[j] + t * r[j]));
      if h > max then max := h
    end;
  Sirka := max
end;

function Cas(i, j : Integer) : Real;
begin
  if r[i] <> r[j] then
    Cas := (p[i] - p[j]) / (r[j] - r[i])
  else
    Cas := 0;
end;

begin
  Assign(f, 'fotograf.in');
  Assign(g, 'fotograf.out');
  Reset(f);
  Rewrite(g);
  Read(f, Vstupov);
  for k := 1 to Vstupov do begin
```

```
  for i := 1 to N do
    Read(f, p[i], r[i]);
  mint := 0;
  mins := Sirka(0);
  for i := 1 to N - 1 do
    for j := i + 1 to N do begin
      t := Cas(i, j);
      if t > 0 then begin
        s := Sirka(t);
        if s < mins then begin
          mins := s;
          mint := t;
        end;
      end;
    end;
    Writeln(g, mint : 0 : 2);
  end;
  Close(g); Close(f);
end.
```

3.2.2 Kontrola

Asi najjednoduchšie bolo postupnosť čísiel utriediť. V utriedenej postupnosti už ľahko skontrolujeme dĺžky úsekov s tou istou hodnotou. Triediť treba čo najšikovnejšie. Výhodu v tomto smere majú C-čkari (funkcia *qsort*). Takéto riešenie vyžaduje približne toľko operácií, koľko treba na utriedenie postupnosti, lebo na kontrolu vzniknutých úsekov potrebujeme približne n operácií. Ak použijeme Quicksort tak v priemere potrebujeme približne $n \log n$ operácií.

Lepšie riešenie je nájsť medián (prvok, ktorý je v poradí $n/2$-tý, rátajú sa aj rovnaké prvky). Ak hľadaný prvok existuje musí byť rovný mediánu. Takže stačí spočítať koľkokrát sa v postupnosti nachádza medián.

A na koniec riešenie, ktoré vyžaduje iba približne n operácií (teda rovnako ako riešenie založené na výpočte mediánu), ale od predchádzajúceho je podstatne programátorsky jednoduchšie. Algoritmus je založený na nasledovnej myšlienke: Ak $x_i \neq x_j$ a oba prvky z postupnosti vylúčime, hľadaný prvok z pôvodnej postupnosti bude rovnaký ako v novej postupnosti. To nám umožní redukovať veľkosť problému. Ak nájdeme nerovnaké hodnoty, obe vylúčime a nájdeme hľadaný prvok v novej postupnosti. Nakoniec skontrolujeme, či je to skutočne on (kontrolu musíme robiť, lebo naša myšlienka je iba implikácia a nie ekvivalencia). Čo v prípade, že nájdeme dva rovnaké prvky? Ak sú všetky prvky rovnaké, máme iba jedného kandidáta. V prípade, že nie sú, aplikujeme prvý prípad. Namiesto skutočného vylučovania prvkov z postupnosti si v programe udržujeme dve premenné: kandidáta K a jeho početnosť P. Keď spracúvame x_i, postupnosť $x_1, x_2, \ldots, x_{i-1}$ môžeme rozdeliť na dve časti veľkosti $2l$ a P. Skupina veľkosti $2l$ predstavuje nerovnaké dvojice prvkov a druhá zase P rovnakých prvkov. Prvok x_i porovnáme s K a v prípade rovnosti zvýšime P o jedna inak P zmenšíme o jedna. Ak P dosiahne 0, za K zvolíme momentálne x_i.

```
program kontrola;
const
  MaxN = 10000;
var
  i, N, K, P : integer;
  X : array[1..MaxN] of integer;
  f, g : text;
begin
  assign(f, 'kontrola.in'); reset(f);
  assign(g, 'kontrola.out'); rewrite(g);
  read(f, N);
  while N <> 0 do begin
    for i := 1 to N do
      read(f, X[i]);
    K := X[1];
    P := 1;
    for i := 2 to N do
      if P = 0 then begin
        K := X[i];
        P := 1
      end else if X[i] = K then
        inc(P)
      else
        dec(P);
    if P = 0 then
      K := -1
    else begin
      P := 0;
      for i := 1 to N do
        if X[i] = K then
          inc(P);
      if P <= N div 2 then
        K := -1
    end;
    if K > 0 then
      writeln(g, K)
    else
      writeln(g, 'NEEXISTUJE');
    read(f, N)
  end;
  close(f); close(g)
end.
```

3.2.3 Zlatokop

Tento príklad bol pomerne jednoduchý. Zo súradníc troch vrcholov obdĺžnika, ktorého strany sú rovnobežné s osami ľahko vypočítame X-ovú súradnicu ľavej a pravej strany a Y-ovú súradnicu hornej a dolnej strany. Potom, keď sa obdĺžniky neprekrývajú, musí byť druhý obdĺžnik buď nad, pod, naľavo, alebo napravo od prvého obdĺžnika, čo ľahko overíme porovnaním súradníc strán.

```
program Zlatokop;
var
  f, g : Text;
  k, l1, p1, h1, d1, l2,
  p2, h2, d2, Vstupov : Integer;

function Max(a, b : Integer) : Integer;
begin
  if a > b then
    Max := a
  else
    Max := b
end;

function Min(a, b : Integer) : Integer;
begin
  if a < b then
    Min := a
  else
    Min := b
end;

procedure Nacitaj(var l, p, h, d : Integer);
var
  x1, y1, x2, y2, x3, y3 : Integer;
begin
  readln(f, x1, y1, x2, y2, x3, y3);
  l := Min(Min(x1, x2), x3);
  p := Max(Max(x1, x2), x3);
  d := Min(Min(y1, y2), y3);
  h := Max(Max(y1, y2), y3)
end;

begin
  Assign(f, 'zlatokop.in');
  Assign(g, 'zlatokop.out');
  Reset(f);
  Rewrite(g);
  Read(f, Vstupov);
  for k := 1 to Vstupov do begin
    Nacitaj(l1, p1, h1, d1);
    Nacitaj(l2, p2, h2, d2);
    if (p2 <= l1) or (l2 >= p1) or
       (h2 <= d1) or (d2 >= h1) then
      writeln(g, 'nie')
    else
      writeln(g, 'ano');
  end;
  Close(g); Close(f);
end.
```

3.2.4 Lampy

Pri riešení tejto úlohy je dôležité si uvedomiť, že pri stláčaní vypínačov nezáleží na poradí, v akom ich stláčame. Taktiež sa neoplatí skúšať možnosti, pri ktorých sa niektorý z vypínačov stlačí viac ako raz, lebo keď ho stlačíme párny počet krát, je to to isté, ako keby sme ho nestlačili vôbec, ak ho stlačíme nepárny počet krát, je to to isté, ako keby sme ho stlačili iba raz.

V najjednoduchšom riešení program načíta údaje o miestnosti do poľa a. Žiarovky očísluje číslami 1 až 9 (postupuje po riadkoch) a platí, že $a[i] = true$ práve vtedy, keď žiarovka s číslom i svieti. V premennej *svieti* máme počet svietiacich žiaroviek. Potom postupne skúšame všetky možnosti, ako stláčať vypínače. Na to slúži rekurzívna procedúra $ries(i)$, ktorá najprv skúsi stlačiť i-ty vypínač a pre tento prípad otestuje všetky možnosti stláčania ďalších vypínačov (t.j. zavolá $ries(i+1)$) a ak žiadna z nich nebola správna, skúsi i-ty vypínač znovu stlačiť (čo je to isté ako keby sa vôbec nepoužil) a znovu testuje všetky možnosti pre ďalšie vypínače. Ak niekedy v priebehu výpočtu bude platiť, že $svieti = 0$, znamená to, že sa dajú zhasnúť všetky žiarovky.

Trocha lepšie riešenie je založené na vylepšení, že stačí preveriť všetkých $2^6 = 64$ podmnožín vypínačov.

Zoberme si teraz niektorý z vypínačov, napríklad ten, ktorý prepína pravý stĺpec. Ten istý účinok, aký dosiahneme jeho stlačením, sa dá dosiahnuť aj stlačením všetkých ostatných vypínačov. Preto ak pri zhasínaní niektorej miestnosti použijeme tento vypínač, dá sa zhasnúť aj bez jeho stlačenia – len musíme stlačiť práve tie vypínače, ktoré sme predtým nepoužili. Takže jeden vypínač môžeme z našich úvah úplne vynechať, zostáva teda iba $2^5 = 32$ podmnožín vypínačov.

Program najprv pre každú takúto podmnožinu vypočíta, ktoré žiarovky budú svietiť, keď začneme so zhasnutými žiarovkami a stlačíme vypínače tejto podmnožiny. Pre každú podmnožinu tento výsledok uloží do poľa *Ok*. Platí, že v danej miestnosti sa dajú všetky žiarovky vypnúť práve vtedy, keď sa v poli *Ok* nachádza popis tejto miestnosti. Pre každú miestnosť zo vstupného súboru sa údaje uložia do poľa a. Žiarovky očísluje číslami 1 až 9 (postupuje po riadkoch) a platí, že $a[i] = true$ práve vtedy, keď žiarovka s číslom i svieti. Potom už stačí prejsť polom *Ok* a zisťujeme, či sa v ňom niekde nenachádza pole a.

```
program Lampy;
  {ktory vypinac co prepina}
const
  vyp : array[1..5, 1..3] of integer =
    ((1, 2, 3), (4, 5, 6),
     (7, 8, 9),  {riadky}
     (1, 4, 7), (2, 5, 8)); {stlpce}
type
  lampa = array[1..9] of boolean;
var
  ok : array[1..32] of lampa;
  a : lampa;
  pocok : integer;
  miestnosti, m : integer;
  fi, fo : text;
```

```
procedure nacitaj;
var
  i, svieti : integer;
  cislo, riadok, stlpec : integer;
  r, s : char;
begin
  {vynulovanie pola a}
  for i := 1 to 9 do a[i] := false;
  {pocet lamp v mniestnosti}
  readln(fi, svieti);
  for i := 1 to svieti do begin
    readln(fi, r, s);
    {poradove cislo lampy}
    riadok := ord(r) - ord('0');
    stlpec := ord(s) - ord('A') + 1;
    cislo := (riadok - 1) * 3 + stlpec;
    a[cislo] := true;
  end;
end;

procedure priprav;
{pripravi pole ok}
var
  i, j, k : integer;
begin
  {zhasnuta miestnost}
  for i := 1 to 9 do
    ok[1, i] := false;
  pocok := 1;
  for i := 1 to 5 do begin
    {do pola ok pridame podmoziny
     obsahujuce i-ty vypinac }
    for j := pocok + 1 to pocok * 2 do begin
      {do ok[j] dame to iste, ako bolo
       v ok[j - pocok], len s prepnutym
       i-tym vypinacom}
      ok[j] := ok[j - pocok];
      for k := 1 to 3 do
        ok[j, vyp[i, k]] :=
          not ok[j, vyp[i, k]];
    end;
    pocok := pocok * 2;
  end;
end;

function nasiel : boolean;
{hlada a v poli ok}
var
  uz : boolean;
  i, j : integer;
begin
  uz := false;
  i := 1;
  while not uz and (i <= pocok) do begin
    uz := true;
    {porovna polia ok[i] a a}
    for j := 1 to 9 do
      uz := uz and (ok[i, j] = a[j]);
    inc(i);
  end;
  nasiel := uz;
end;

begin
  assign(fi, 'lampy.in');
  assign(fo, 'lampy.out');
  reset(fi);
  rewrite(fo);
  readln(fi, miestnosti);
  priprav;
  for m := 1 to miestnosti do begin
    nacitaj;
    if nasiel then writeln(fo, 'Da sa')
    else writeln(fo, 'Neda sa');
  end;
  close(fi);
  close(fo);
end.
```

3.2.5 Mesto

Uvedomme si, že úlohou je vlastne zistiť, či zadaný graf obsahuje cyklus nepárnej dĺžky. Takýto cyklus v našom grafe neexistuje práve vtedy, ak možno rozdeliť množinu vrcholov na dve podmnožiny A, B, pričom vo vnútri množiny A, ani vo vnútri množiny B, nie sú žiadne dva vrcholy spojené hranou. Z tejto úvahy vyplýva jednoduchý algoritmus:

Načítame si hrany a vytvoríme pre každý vrchol maticu jeho susedov. Potom začneme z ľubovoľného vrcholu ofarbovanie dvoma farbami a to tak, že žiadne dva susedné vrcholy nemajú rovnakú farbu — to môžeme robiť jednoduchým prehľadávaním grafu. Ak sa nám stane, že niektorý vrchol nevieme ofarbiť (lebo susedí s vrcholmi oboch farieb), graf obsahuje nepárny cyklus.

```
program mesto;
const
  maxn = 100;
var
  z : array [1..maxn, 0..maxn] of integer;
  o : array [1..maxn] of integer;
  zas : array [1..maxn, 1..2] of integer;
  uzas : integer;
  n : integer;
  koniec, nep : boolean;
  fin, fout : text;

procedure nacitaj(var koniec : boolean);
var
  i, a, b : integer;

begin
  koniec := false;
  readln(fin, n);
  if n = -2 then begin
    koniec := true;
    exit;
  end;
  {inicializacia}
  for i := 1 to n do begin
    z[i, 0] := 0;
    o[i] := 0;
  end;
  {citanie hran}
  read(fin, a);
  while a > -1 do begin
```

```
    readln(fin, b);
    inc(z[a, 0]);
    z[a, z[a, 0]] := b;
    inc(z[b, 0]);
    z[b, z[b, 0]] := a;
    read(fin, a);
  end;
end;

procedure work(var neparnyc : boolean);
var
  i, u, v, f : integer;
begin
  neparnyc := false;
  uzas := 1;
  zas[1, 1] := 1;zas[1, 2] := 1;
  o[1] := 1;
  while uzas > 0 do begin
    v := zas[uzas, 1];
    f := zas[uzas, 2];
    dec(uzas);
    for i := 1 to z[v, 0] do begin
      u := z[v, i];
      if o[u] = f then begin
        neparnyc := true;
        exit;
      end;
      if o[u] = 0 then begin
        o[u] := -f;
        inc(uzas);
        zas[uzas, 1] := u;
        zas[uzas, 2] := -f;
      end;
    end;
  end;
end;

begin
  assign(fin, 'mesto.in');
  reset(fin);
  assign(fout, 'mesto.out');
  rewrite(fout);
  nacitaj(koniec);
  while not koniec do begin
    work(nep);
    if not nep then
      writeln(fout,
        'Poslancom hrozia predcasne volby')
    else
      writeln(fout, 'Dobre rozhodnutie');
    nacitaj(koniec);
  end;
  close(fin); close(fout);
end.
```

3.2.6 Papier

Tento príklad bol riešiteľný priamočiaro. Vytvoríme si maticu, v ktorej si budeme uchovávať, ktoré hrany sú "rozstrihnuté" a ktoré nie. V programe sme použili maticu `r` pre hrany vpravo od jednotlivých políčok a maticu d pre hrany pod jednotlivými políčkami. Vhodné je vytvoriť matice tak, aby obsahovali zarážky (tj. na okrajoch sú všetky hrany rozstrihnuté).

Pri načítavaní vstupu teraz simulujeme strihanie a označujeme v poliach rozstrihnuté hrany. Po dočítaní pre každé ešte neprehľadané políčko prehľadáme všetky, ktoré sú s ním spojené. Tak zistíme (podľa toho, koľkokrát budeme musieť začať prehľadávať), na koľko častí sa papier rozpadol.

Pri písaní programu si ešte treba uvedomiť, že na prehľadávanie nemôžeme použiť jednoduchú rekurziu – pri papieri, ktorý má maximálny rozsah a ostane na konci súvislý, nám totiž nebude stačiť stack na rekurzívne volania.

```
program papier;
const
  max=80;
var
  r, d : array [0..max, 0..max] of boolean;
  a : array [1..max, 1..max] of integer;
  zas : array [1..max * max, 1..2] of integer;
  uzas : integer;
  m, n : integer;
  koniec : boolean;
  fin, fout : text;
  kt, poc : integer;

procedure swap(var a, b : integer);
var
  c : integer;
begin
  c := a; a := b; b := c;
end;

procedure rozrez(a, b, c, e : integer);
var
  i : integer;
begin
  if a = c then begin
    if b > e then swap(b, e);
    for i := b + 1 to e do
      d[a, i] := false;
  end else begin
    if a > c then swap(a, c);
    for i := a + 1 to c do
      r[i, b] := false;
  end;
end;

procedure try(i, j, f : integer);
begin
  if a[i, j] = 0 then begin
    a[i, j] := f;
    inc(uzas);
    zas[uzas, 1] := i;
    zas[uzas, 2] := j;
  end;
end;

procedure oznac(i, j, f : integer);
begin
  uzas := 0;
  try(i, j, f);
  while uzas > 0 do begin
```

```
    i := zas[uzas, 1];
    j := zas[uzas, 2];
    dec(uzas);
    if r[i, j] then try(i, j + 1, f);
    if r[i, j - 1] then try(i, j - 1, f);
    if d[i, j] then try(i + 1, j, f);
    if d[i - 1, j] then try(i - 1, j, f);
  end;
end;

procedure init;
var
  i, j : integer;
begin
  for i := 1 to m do
    for j := 1 to n do begin
      a[i, j] := 0;
      r[i, j] := true;
      d[i, j] := true;
    end;
  for i := 1 to m do begin
    r[i, 0] := false;
    r[i, n] := false;
  end;
  for i := 1 to n do begin
    d[0, i] := false;
    d[m, i] := false;
  end;
end;

procedure work(var koniec : boolean;
               var poc : integer);
var
  aa, b, c, d, i, j : integer;
begin
  koniec := false;
  read(fin, m);
  if m = -1 then begin
    koniec := true;
    exit
  end;
  readln(fin, n);
  init;
  while not eoln(fin) do begin
    readln(fin, aa, b, c, d);
    rozrez(aa, b, c, d);
  end;
  readln(fin);
  poc := 0;
  for i := 1 to m do
    for j := 1 to n do
      if a[i, j] = 0 then begin
        inc(poc);
        oznac(i, j, poc);
      end;
end;

begin
  assign(fin, 'papier.in');
  reset(fin);
  assign(fout, 'papier.out');
  rewrite(fout);
  kt := 1;
  work(koniec, poc);
  while not(koniec) do begin
    writeln(fout, 'Pripad ', kt,
      '. Papier sa rozpadne na ',
      poc, ' casti.');
    inc(kt);
    work(koniec, poc);
  end;
  close(fin); close(fout);
end.
```

3.2.7 O Kocúrkovskom čase

Zrejme dĺžka kocúrkovského Času bude násobok doby otočenia každého kolesa. Keďže nás zaujíma prvé zacyklenie, treba nájsť najmenší spoločný násobok doby otáčania všetkých kolies. Prvé koleso sa ale otáča rýchlosťou 1 zub $\cdot\, \mathrm{s}^{-1}$ a keďže kolesá do seba zapadajú, všetky sa otáčajú touto rýchlosťou; doba otočenia v sekundách sa potom rovná počtu zubov. Takže stačí nájsť najmenší spoločný násobok počtu zubov všetkých kolies. Teda treba nájsť $\mathrm{nsn}\{z_1, \ldots, z_n\}$. Platí ale $\mathrm{nsn}\{z_1, \ldots, z_n\} = \mathrm{nsn}\{\mathrm{nsn}\{z_1, z_2\}, z_3, \ldots, z_n\}$ pre $n > 2$ a $\mathrm{nsn}\{a, b\} = \frac{ab}{\mathrm{nsd}\{a,b\}}$. Preto stačí postupne po dvojiciach rátať nsn použitím druhého vzorca. Na zrátanie nsd sa dá použiť Euklidov algoritmus.

```
program prevody;
var
  f, g : text;
  Pocet, N, i, opak, Cas, Zub, d : word;

procedure vymen(var a : word; var b : word);
var
  c : integer;
begin
  c := b;
  b := a;
  a := c;
end;

function nsd(a, b : word) : word;
var
  c : integer;
begin
  if a < b then vymen(a, b);
  if b = 1 then nsd := 1
  else begin
    while b > 0 do begin
      a := a mod b;
      if a < b then vymen(a, b);
    end;
    nsd := a;
  end;
end;

begin
  assign(f, 'prevody.in');
  reset(f);
  assign(g, 'prevody.out');
  rewrite(g);
  read(f, Pocet);
  for opak := 1 to Pocet do begin
    read(f, N, Cas);
    for i := 2 to N do begin
```

```
    read(f, Zub);
    d := nsd(Cas, Zub);
    Cas := Cas * (Zub div d);
  end;
  writeln(g, 'Zadanie ', opak, ' : ');
  writeln(g, 'Cas v Kocurkove trva ',
              Cas, ' sekund.');
  writeln(g);
end;
close(f);
close(g);
end.
```

3.2.8 Slimák

Tento príklad neskrýval žiadnu hlbokú myšlienku ani vražedný algoritmus. V nasledujúcom riešení si rozrežeme slimáka po dĺžke na hornú a dolnú polovicu a stredný riadok (ten sa vyrieši zvlášť). Potom v každej polovici zvlášť popíšeme párne a nepárne riadky a striedavo ich vypisujeme.

```
program Slimak;
var
  f, g: text;
  Vek: integer;
  nic, Smer: char;
  i, Dlzka: integer;

procedure Pis(co: string;kolko: integer);
var
  i: integer;
begin
  for i := 1 to kolko do
    write(g, co);
end;

begin
  assign(f, 'slimak.in');
  reset(f);
  assign(g, 'slimak.out');
  rewrite(g);
  readln(f, Vek, nic, Smer);
  while Vek > 0 do begin
    write(g, 'Slimak ma ', Vek,
        ' mesiacov a ide smerom v');
    if Smer='P' then
      writeln(g, 'pravo : ')
    else
      writeln(g, 'lavo : ');
    writeln(g);
    Dlzka  :=  4 * (Vek - 1);
    for i := 0 to Dlzka div 2 - 1 do begin
      if Smer = 'L' then write(g, '   ');
      if odd(i) then begin
        Pis('* ', (i + 1) div 2);
        Pis(' ', Dlzka - 2 * (i + 1));
        Pis(' *', (i + 1) div 2);
      end else begin
        Pis('* ', i div 2);
        Pis('*', Dlzka - 2 * i);
        Pis(' *', i div 2);
      end;
      if Smer = 'P' then write(g, '   ');
      writeln(g);
    end;
    if Vek > 1 then begin
      if Smer = 'L' then
        write(g, '   ');
      Pis('* ', Vek-2);
      if Smer = 'L' then
        write(g, '**')
      else
        write(g, '* ');
      if Smer = 'P' then
        write(g, '**')
      else
        write(g, ' *');
      Pis(' *', Vek - 2);
      if Smer = 'P' then write(g, '   ');
      writeln(g);
    end;
    for i := Dlzka div 2 - 1 downto 2 do begin
      if Smer = 'L' then write(g, '   ');
      if odd(i) then begin
        Pis('* ', (i - 1) div 2);
        if Smer = 'P' then write(g, '* ');
        Pis('  ', Dlzka div 2 - i);
        Pis(' *', (i - 1) div 2);
        if Smer = 'L' then write(g, ' *');
      end else begin
        Pis('* ', i div 2 - 1);
        if Smer = 'P' then write(g, '* ');
        Pis('*', Dlzka - 2 * (i - 1));
        Pis(' *', i div 2 - 1);
        if Smer = 'L' then write(g, ' *');
      end;
      if Smer = 'P' then write(g, '   ');
      writeln(g);
    end;
    if Smer = 'L' then begin
      write(g, 'H');
      Pis(' ', Dlzka + 1);
      writeln(g, '*');
      write(g, 'H');
      Pis('*', Dlzka + 2);
      writeln(g);
    end else begin
      write(g, '*');
      Pis(' ', Dlzka + 1);
      writeln(g, 'H');
      Pis('*', Dlzka + 2);
      writeln(g, 'H');
    end;
    writeln(g);
    writeln(g);
    readln(f, Vek, nic, Smer);
  end;
  close(f);
  close(g);
end.
```

4.1 Domáce kolo

4.1.1 Mašinky

Tento príklad bol veľmi jednoduchý, preto hodnotenie bolo o niečo prísnejšie ako obvykle. Za program, ktorý pracoval v čase $O(n)$, ste mohli získať najviac 10 bodov. Ak váš algoritmus mal časovú zložitosť $O(n^2)$ mohlo vám byť pridelené najviac 6 bodov. V prvom prípade sa strhávali 2 body, keď program používal pomocné pole, za každé ďalšie pomocné pole sa strhával ďalší 1 bod. Ak ste zvolili zlý rozsah (menší ako obmedzenie v zadaní) a za ďalšie chyby sa strhávalo po 1 bode.

Vzorové riešenie je založené na veľmi jednoduchej myšlienke: ak pôvodná mašinka prekóduje i-te písmenko na j-te písmenko, potom naša nová mašinka musí j-te písmenko prekódovať na i-te písmenko. Túto novú mašinku môžeme veľmi ľahko konštruovať už pri načítavaní.

Najčastejšie nedostatky vstupov a výstupov, ktoré sa vo vašich riešeniach vyskytli:

- väčšina z vás krvopotne načítavala čísla do reťazcov aby ich potom pomocou pascalovského príkazu *val* previedla na čísla. V propozíciách však bolo napísané, že vstup je zadaný správne, mohli ste teda rovno čítať čísla – netreba si zbytočne komplikovať život. Snaha o "bezchybné" načítanie pripravila o body tých, ktorí si neuvedomili, že nemôžu naraz čítať celý riadok zo vstupu, keďže tento môže byť dlhší ako 255 znakov.
- väčšina z vás vypisovala výslednú permutáciu pomocou takéhoto cyklu:

  ```
  for i:=1 to n do write(f,kod[i],' ');
  writeln(f);
  ```

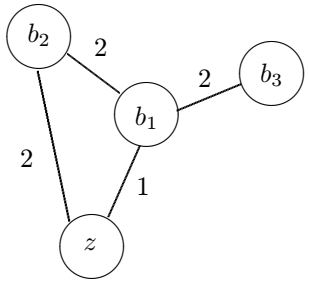
  Tu ste si ale neuvedomili, že do výstupného súboru zapisujete za posledným číslom kódu mašinky, čo je podľa zadaní a špecifikácie výstupu neprípustné. V tomto kole sme vám za to body nestrhávali, v ďalšom kole by vám však takéto riešenie nebolo uznané. Správny výpis – pozri program.

```
program masinky;
var
  fin, fout : text;
  a : array [1..9999] of integer;
  n, i, j : integer;
  ch : string[1];

begin
  assign(fin, 'masinka.in');
  reset(fin);
  assign(fout, 'masinka.out');
  rewrite(fout);
  read(fin, n);
  for i := 1 to n do begin
    read(fin, j);
    {i-te pismenko sa prekoduje na j-te,
     teda vo vysledku sa musi j-te
     prekodovat na i-te}
    a[j] := i;
  end;
  ch := '';
  for i := 1 to n do begin
    write(fout, ch, a[i]);
    ch := ' ';
  end;
  writeln(fout);
  close(fin); close(fout);
end.
```

4.1.2 Myš

V zadaní nebol uvedený príklad vstupných údajov, čo spôsobilo mnohým z Vás ťažkosti, lebo bolo treba čítať pozornejšie zadanie. Niektoré detaily, napríklad rozsah n, neboli však určené ani tam. V takom prípade si môžete určiť obmedzenia sami, ale treba ich potom uviesť aj do Vašich komentárov. V ďalšom predpokladáme, že $n \leq 100$ (inak by sme museli použiť iné dátové štruktúry). Niektorým riešeniam chýbalo uvedenie hlavnej myšlienky použitého algoritmu. Väčšinou sa to prejavilo: "... a nájdeme najkratšie cesty". Ale ako sa hľadali bolo treba namáhavo hľadať v programe. V niektorých prípadoch sa to ani pri najlepšej snahe nepodarilo. Najčastejšou chybou bolo, že ste z počiatočného brlôžku pokračovali po najlacnejšej priamej ceste do niektorého z tých, kde boli dobroty a odtiaľ opäť po najkratšej ceste do ďalšieho (1 bod). Skúste si nájsť príklad vstupných údajov, kde tento postup zlyhá. Jeden príklad je na obrázku (vtedy týmto postupom nájdete trasu b_1, b_2, b_3, alebo b_1, b_3, b_2 ale nie najkratšiu b_2, b_1, b_3).



Dobré riešenia sa dali rozdeliť do troch hlavných skupín.

1a) Použili ste štandardný algoritmus na nájdenie najkratších ciest z jedného do všetkých ostatných brlôžtekov (Dijkstrov algoritmus – je použitý aj vo vzorovom programe) $O(n^2)$. Hlavná myšlienka je nasledovná: na začiatku vieme minimálnu vzdialenosť len pre počiatočný vrchol, tá je prirodzene rovná 0. Počas celého postupu rozdeľujeme brlôžky na tie, čo už majú minimálnu vzdialenosť definitívne určenú a na zvyšné. Pre zvyšné si udržiavame minimálnu vzdialenosť, na ktorú sa k nim vieme dostať ale len cez už definitívne spracované brlôžky. Ak sa k niektorému brlohu nevieme dostať cez už spracované, jeho vzdialenosť je ∞. Zo zatiaľ nespracovaných si vyberieme ten, čo má minimálnu vzdialenosť od počiatočného brlôžka, čím ho preradíme medzi už definitívne určené. Uvedomte si, že to, čo sme určili, je ozaj najkratšia vzdialenosť od začiatku, prechádzajúca len cez už spracované vrcholy. Tým sa však mohli zmeniť vzdialenosti od počiatočného k zvyšným ešte nespracovaným brlohom. Takže musíme prepočítať, či cez naposledy spracovaný vrchol k nim nevedie kratšia cesta. Takto pokračujeme pokým nespracujeme všetky brlohy. Všimnite si, že v každom kroku sa počet ešte nespracovaných brlohov zmenší o jedna. Krokov je n a v každom musíme obnoviť vzdialenosti pre zvyšné brlohy, ktorých je postupne $n-1, n-2, \ldots, 1$, odkiaľ vidno celkovú zložitosť. Treba ešte upozorniť, že predpokladáme, že vzdialenosti medzi dvomi brlohmi sú nezáporné, čo je, vzhľadom na to, o čo sa jedná, oprávnené. Pretože chodby medzi brlôžtekmi boli obojsmerné, stačilo zistiť pomocou tohoto algoritmu vzdialenosti od ľubovoľných troch brlohov spomedzi z, b_1, b_2, b_3.

1b) Na nájdenie najkratších ciest medzi všetkými vrcholmi (Floydov algoritmus) $O(n^3)$ (10 bodov). Tento algoritmus pracuje s maticou $n \times n$, priamych vzdialeností medzi brlohmi, t.j. prvky (i,j)-ty a (j,i)-ty sú rovné alebo vzdialenosti medzi brlohmi i a j, alebo ∞, keď brlohy nemajú priame spojenie. V ďalšom maticu v každom kroku "prepočítavame" tak, aby po k tom prepočítaní prvky (i,j) a (j,i) určovali vzdialenosť najkratšej cesty medzi brlohmi i a j, ktoré prechádzajú len cez brlohy 1 až k. Tie alebo neobsahujú k-ty brloh, alebo ich vieme dostať spojením dvoch ciest z i do k a z k do j, ktoré prechádzajú len cez brlohy 1 až $k-1$. To sú ale hodnoty našej matice v predchádzajúcom kroku. Tiež si všimnite, že maticu môžeme prepočítavať po riadkoch, lebo k-ty riadok sa v k-tom kroku nezmení. V našom prípade je matica symetrická, ale tento algoritmus by pracoval aj keby cesty medzi vrcholmi boli iba jednosmerné. Dôležité si bolo uvedomiť, že treba vyskúšať, ktorá zo šiestich možných trás je najkratšia.

2a) Podobne ako 1b), ale cesty predlžovali bez podmienky, že po k-tom kroku budeme mať najkratšie vzdialenosti prechádzajúce iba cez brlohy 1 až k, ktorá je veľmi dôležitá, aby sme mohli po n krokoch skončiť. Najkratšie cesty zisťovali tak, že postupne skúmali, či sa nedá do niektorého brlôžka dostať kratšou cestou cez iný brlôžtek. Ak áno mohlo to ovplyvniť nejaký už predtým spracovaný brlôžtek, takže celý postup sa opakoval, až pokým sa už žiadnu cestu nepodarilo skrátiť $O(n^4)$. Ak by sme skončili skôr, teda napríklad bez testovania, či sa nejakú cestu podarilo skrátiť, nedostali by sme najmenšie vzdialenosti. Teda by sme nemali zaručené, že by sme na základe nich dostali najkratšiu cestu prechádzajúcu danými brlohmi. Inak povedané také riešenie je zlé.

2b) Z počiatočného vrcholu prehľadávali všetky možné cesty (backtracking). Bolo to treba robiť rozumne: keď sme dorazili do poslednej komôrky zapamätať si dĺžku, aj poradie komôrok v akom sme ich navštevovali; môžeme skúsiť inú cestu, keď je doterajšia dlhšia než najkratšia už nájdená, alebo keď prídeme do brlôžku, kde sme už boli (9 bodov).

3) Hľadali najkratšie cesty ako v 2b), ale nevyužili všetky informácie a vždy našli iba vzdialenosť medzi dvoma danými brlôžtekmi, takže museli hľadať najmenej šesť ráz (8 bodov).

Vysvetlenie hlavnej myšlienky bolo z toho za najviac 2 body. Za nešikovnosti v realizácii sa strhávali ďalšie body.

```
program Mys;
const
  MaxN = 100;
var
  n : integer;
  f, g : text;
  b : array[0..3] of integer;
  vzd : array[1..MaxN, 1..MaxN] of integer;

procedure Nacitaj;
var
  i, j, u, v : integer;
begin
  assign(f, 'Mys.in'); reset(f);
  readln(f, n, b[0], b[1], b[2], b[3]);
  for i := 1 to n do
    for j := 1 to n do
      if i <> j then
        vzd[i, j] := MaxInt
      else
        vzd[i, j] := 0;
  for i := 1 to n do begin
    read(f, v);
    repeat
      read(f, u);
      read(f, vzd[v, u]);
      vzd[u, v] := vzd[v, u]
    until eoln(f);
  end;
  close(f)
end;

procedure UrciVzdialenosti;

  procedure Dijkstra(start : byte);
  { urci najkratsie vzdialenosti z
    vrcholu start do vsetkych ostatnych }
  var
    Bol : array[1..MaxN] of boolean;
    i, j, k : integer;
  begin
    for i := 1 to n do Bol[i] := false;
    Bol[start] := true;
    for i := 1 to n - 1 do begin
      j := 1;
      while Bol[j] do inc(j);
      k := j + 1;
      { najdeme najmensi neurceny }
      while k <= n do begin
        if not Bol[k] and
           (vzd[start, k] < vzd[start, j])
          then j := k;
        inc(k)
      end;
      Bol[j] := true;
      for k := 1 to n do
        if (not Bol[k]) and
           (vzd[start, j]  <
             vzd[start, k] - vzd[j, k])
        then begin
          vzd[start, k] :=
            vzd[start, j] + vzd[j, k];
          vzd[k, start] := vzd[start, k]
        end
    end
  end;

var
  i : integer;
begin
  { staci zistit tri razy, lebo
    chodbickami sa da ist oboma smermi }
  for i := 1 to 3 do Dijkstra(i)
end;

procedure NajkratsiuVypis;
const
  { 6 moznosti ako sa daju komorky
    prejst }
  ix : array[1..6, 1..3] of byte =
    ((1, 2, 3), (1, 3, 2), (2, 1, 3),
     (2, 3, 1), (3, 1, 2), (3, 2, 1));
var
  i, DlzkaCesty, ktory : integer;
  x : longint;
begin
  DlzkaCesty := MaxInt;
  ktory := 1;
  for i := 1 to 6 do begin
    x := longint(vzd[b[0], b[ix[i, 1]]]) +
      vzd[b[ix[i, 1]], b[ix[i, 2]]] +
      vzd[b[ix[i, 2]], b[ix[i, 3]]];
    if x < DlzkaCesty then begin
      DlzkaCesty := x;
      ktory := i
    end
  end;
  assign(g, 'Mys.out'); rewrite(g);
  write(g, b[ix[ktory, 1]],
    b[ix[ktory, 2]] : 4, b[ix[ktory, 3]] : 4);
  close(g);
end;

begin
  Nacitaj;
  UrciVzdialenosti;
  NajkratsiuVypis;
end.
```

4.1.3 Počtár

Mnohí ste splnili Boleslavovi jeho dávny sen a preto Vás štedro odmenil bodíkmi. Objavili sa dva typy správnych riešení. Najjednoduchšie riešenia využívali pomocnú procedúru, ktorá pre dané a a b^2 zistila, či je b^2 vyscítateľné z a. Potom postupne skúšali za a dosadzovať čísla od 1 po b^2, až napokon našli najmenšie číslo, z ktorého sa b^2 dá vyscítať. Za takéto riešenie ste mohli získať 6-8 bodov, podľa spôsobu naprogramovania pomocnej procedúry.

Druhý typ riešení bol trochu matematickejší. Využíval nasledovný vzorec na súčet k za sebou nasledujúcich čísiel: $a+(a+1)+...+(a+k-1) = a.k+k.(k-1)/2$. Keď chceme pre dané b^2 zistiť, či je vyscítateľné z nejakého čísla a, pričom počet sčítaných čísiel má byť dané k, potom musí platiť $a = (b^2 - k(k-1)/2)/2$. Hodnota výrazu na pravej strane rovnosti klesá s rastúcim k, teda ak nájdeme najväčšie také k, pre ktoré je $a = (b^2 - k(k-1)/2)/2$ prirodzené číslo, potom je a najmenšie číslo, z ktorého je b^2 vyscítateľné. Aby

bola hodnota výrazu kladné číslo musí byť $k \leq (\sqrt{1+8b^2}-1)/2$, teda za k stačí postupne dosadzovať čísla $\lfloor\sqrt{1+8b^2}-1)/2\rfloor, ..., 1$, kým nenájdeme také k, pre ktoré je hodnota výrazu počítajúceho a prirodzené číslo. Za takéto vzorové riešenie ste mohli získať 10 bodov.

```
program Boleslav;
var
  fin, fout : Text;
  n, i, b : Integer;

function Vypocitaj(x : Integer) : Integer;
var
  k : Word;
begin
  k := trunc((Sqrt(1 + 8.0 * x) - 1) / 2);
  x := x - k * (k - 1) div 2;
  while x mod k <> 0 do begin
    k := k - 1;
    x := x + k;
  end;
  Vypocitaj := x div k;
end;

begin
  Assign(fin, 'boleslav.in');
  Reset(fin);
  Assign(fout, 'boleslav.out');
  Rewrite(fout);
  Readln(fin, n);
  for i := 1 to n do begin
    Readln(fin, b);
    Writeln(fout, Vypocitaj(b));
  end;
  Close(fout); Close(fin);
end.
```

4.1.4 Vlaky

Vlak s dobou obehu m bude v každom čase, ktorý je násobkom m na tom istom mieste, odkiaľ vyštartoval. Všetky vlaky sa budú naraz nachádzať na mieste odkiaľ vyštartovali v čase, ktorý je násobkom všetkých dôb obehu. Prvýkrát táto situácia nastane v čase t, ktorý je najmenším spoločným násobkom všetkých dôb obehu. Od tohto okamihu sa už bude situácia periodicky opakovať (s periódou t). Ak sa teda všetky vlaky stretnú na hlavnej stanici, budú sa tak stretávať každých t časových jednotiek. Ak sa za prvých t časových jednotiek nestretnú, nestretnú sa už nikdy.

Majme dva vlaky A a B. Vždy, keď sa všetky vlaky stretnú na stanici, musia sa tam vyskytovať aj vlaky A a B. Vlak A príde na hlavnú stanicu vždy v čase $km + a$, kde m je doba obehu vlaku A, a je čas prvého príchodu, k ľubovoľné celé nezáporné číslo. Vlak B príde na stanicu vždy v čase $ln + b$, kde n je doba obehu, b čas prvého príchodu a l celé nezáporné číslo. Z predchádzajúcich úvah vyplýva, že ak sa oba vlaky stretnú, bude sa tak diať vždy v čase $j\,\mathrm{nsn}(m,n)+c$, kde nsn je najmenší spoločný násobok, j ľubovoľné celé nezáporné číslo a c je určíme hľadaním celočíselných riešení rovnice $km + a = ln + b$. Úpravou dostaneme $km - ln = b - a$. Nech d je najväčší spoločný deliteľ čísel m a n. Potom ak za k a l dosadíme ľubovoľné celé čísla, ľavá strana rovnice je vždy deliteľné číslom d. Preto ak má mať rovnica riešenie, musí byť aj pravá strana deliteľná číslom d. Ak je táto podmienka splnená, riešenie rovnice už ľahko nájdeme. Využijeme pri tom fakt, že pre ľubovoľné celé čísla m a n existujú celé čísla u a v také, že $um + vn = d$. Ak túto rovnosť prenásobíme číslom $(b-a)/d$, dostaneme

$$\frac{(b-a)u}{d}m + \frac{(b-a)v}{d}n = b - a$$

z čoho už vidno, že jedno z riešení rovnice je $k = \frac{(b-a)u}{d}$ a $l = -\frac{(b-a)v}{d}$. Vlak teda príde na stanicu v čase $\frac{(b-a)u}{d}m + a$ (toto číslo však ešte nemusí čas prvého príchodu, ten získame ako zvyšok po delení číslom $\mathrm{nsn}(m,n)$).

Takto sme vyjadrili čas stretávania dvoch vlakov v rovnakej forme ako čas príchodu jedného vlaku na stanicu. Preto rovnaký postup môžeme použiť aj na výpočet času, kedy sa na stanici stretnú $3, 4, \ldots n$ vlakov, pričom vždy použijeme údaje o jednom vlaku načítané zo súboru a údaje o menšej skupine vypočítané v predchádzajúcom kroku výpočtu.

Čísla u a v vypočítame súčasne s výpočtom najmenšieho spoločného deliteľa d mierne modifikovaným Euklidovým algoritmom. Majme dané a a b ($a > b$) a hľadáme u, v také, že $au + bv = d$. Ak a je deliteľné b, potom $d = b$ a platí, že $a \cdot 0 + b \cdot 1 = d$. Ak a nie je deliteľné b, rekurzívne vypočítame hodnoty d', u' a v' pre $a' = b$ a $b' = a \bmod b$. Platí, že

$$\begin{aligned} d = a'u' + b'v' &= bu' + (a - b(a \operatorname{div} b))v' \\ &= b(u' + v'(a \operatorname{div} b)) + av'. \end{aligned}$$

Z tohto vzťahu už ľahko vyjadríme u a v.

Čo sa týka bodovania, algoritmy podobné vzorovému riešeniu (ktoré dokázali vyriešiť úlohu pre dva vlaky v čase $O(\log mn)$) mohli získať 10 bodov. Iné riešenia, ktoré správne zisťovali, kedy má úloha riešenie, mohli získať najviac 8 bodov, pričom sa strhávali body za horšiu efektívnosť a nedostatočný popis. Riešenia, ktoré simulovali polohu vláčikov, až kým čas neprekročil určitú konštantu a iné nesprávne riešenia mohli získať najviac 2 body.

```
program rovnice;
const
  maxn = 100;
var
  n : integer;     {pocet trati}
  blokov, blok : integer;
  {popisy trati}
  a, m : array[1..maxn] of longint;
  fin, fout : text;  {vst. a vyst. subor}
  i : integer;
  dasa : boolean;   {ci sa stretnu}
  na, nm : longint;  {nove hodnoty a a m}

procedure nsd(a, b : longint;
              var d, u, v : longint);
{do d ulozi nsd(a, b),
 u, v su take cisla, ze au + bv = d}
var
  uu, vv : longint;
begin
  if a < b then      {vymen poradie}
    nsd(b, a, d, v, u)
  else {a >= b}
    if a mod b = 0 then begin
    {nsd(a, b) = b, b = 0 . a + 1 . b}
      d := b;
      u := 0;
      v := 1;
    end else begin
      {rekurzivne vyratame}
      nsd(b, a mod b, d, uu, vv);
      u := vv;
      v := uu - vv * (a div b);
    end;
end;

function spoj_rovnice(a, b, m, n : longint;
            var c, mn : longint) : boolean;
{hlada spolocne riesenie rovnic
 x = k . m + a,  x = l . n + b
 v tvare  x = j . mn + c,
 ak riesenie neexistuje,  vrati false}
var
  d, u, v : longint;
begin
  nsd(m, n, d, u, v);
  if (b - a) mod d <> 0 then
    spoj_rovnice := false
  else begin
    mn := m * n div d;
    c := m * (n + (b - a) * u) div d + a;
    c := c mod mn;
    if c < 0 then
      c := c + mn;
    spoj_rovnice := true;
  end;
end;

begin
  {otvorenie suborov}
  assign(fin, 'vlaky.in');
  reset(fin);
  assign(fout, 'vlaky.out');
  rewrite(fout);
  readln(fin, blokov);
  for blok := 1 to blokov do begin
    {nacitanie udajov}
    readln(fin, n);
    for i := 1 to n do
      readln(fin, m[i], a[i]);
    {spajanie rovnic}
    dasa := true;
    while (n > 1) and dasa do begin
      dasa := spoj_rovnice(a[n], a[n-1],
        m[n], m[n-1], na, nm);
      if dasa then begin
        n := n - 1;
        a[n] := na;
        m[n] := nm;
      end;
    end;
    {vypis vysledku}
    if dasa then
      writeln(fout, a[1])
    else
      writeln(fout, 'Vlaky sa nestretnu.');
  end;
  close(fin);
  close(fout);
end.
```

4.1.5 Stavbári

Riešenia tohto príkladu možno rozdeliť na fungujúce a nefungujúce. Početnú skupinu nefungujúcich riešení tvoria také, ktorých autori si neuvedomili, že jeden diel môže zaberať viac stĺpcov. Potom rôznym spôsobom spočítavali výšku v jednotlivých stĺpcoch nezávisle a vypísali maximum. Za takéto algoritmy sa dalo získať maximálne 0 bodov.

Ďalšia skupina riešení boli také, ktoré sa snažili vypočítať výšku na základe vhodného ofarbenia – políčku patriacemu nejakému predmetu sa zväčša priraďovala jeho "lokálna výška". Spoločný problém týchto riešení je v tom, že nedokážu zistiť "viacnásobné" posuny – keď sa pod danou časťou nachádza časť, ktorá ešte bude klesať a zatiaľ nebola farbená. Maximálne 4 body.

Fungujúce riešenia simulovali posúvanie častí domov. Najprv sa označia jednotlivé časti domov rôznymi farbami a potom sa tie, ktoré sa ešte dajú posunúť posúvajú. Nakoniec sa nájde maximálna výška. Za tieto

riešenia sa dalo získať 7–10 bodov, v závislosti od implementácie. Niektorí ste zbytočne veľakrát prefarbovali už ofarbené časti, naopak mnohí ste sa pokúšali implementovať rôzne zefektívnenia ako posúvať sa naraz o viac riadkov (pri nešikovnom zápise to môže priniesť viac škody ako osohu), preskakovať prázdne časti a pod. Častou chybou bolo, že ste používali veľa rekurzie – v poli rozmerov 100×100 môže byť rádovo 5000 malých objektov, alebo objekty s rádovo 10000 políčkami a keď máte rekurzívnu procedúru s množstvom lokálnych premenných, tak zásobník dostáva zabrať. Za neefektívnu rekurziu sa strhával 1 bod.

Popis je podstatná súčasť riešenia, ktorú mnohí podceňujete. Častokrát je to buď jednoriadkový komentár v programe, alebo výpis premenných s komentárom typu "toto je premenná cyklu", "túto premennú používam na rôzne veci". Popis by mal objasňovať myšlienku programu tak, aby bolo zrejmé ako pracuje aj bez čítania kódu. Za chýbajúci alebo nevhodný popis sa strhávali 2 body.

Nasledujúce riešenie simuluje padanie častí. Každá časť sa označí inou farbou. Máme dve špeciálne farby: -1 označuje časti, ktoré už dopadli a ďalej sa nehýbu, -2 označuje časti, ktoré práve dopadli a ešte ich treba dokončiť. Ostatné farby predstavujú padajúce objekty. Prechádzame poľom zospodu nahor po riadkoch a políčka prislúchajúce padajúcim objektom prepisujeme o riadok nižšie. Ak týmto dopadla časť, označíme -1 tie jej políčka, ktoré už sú na správnom mieste a -2 ostatné. Takéto políčka potom ešte raz presunieme a prepíšeme na -1. Nakoniec zistíme maximálnu výšku.

```
program Stavbari;
var
  f, g : text;
  A : array[0..101, 0..101] of integer;
  { M, N sú rozmery úlohy,
    c je počet blokov}
  M, N, c, vyska : integer;
  i, j : integer;
  este : boolean;
  stack : array[1..20001, 1..2] of shortint;

{ vyplní súvislý objekt na pozícii [x, y] tak,
  že farbu z nahradí na1 resp. na2 podľa toho,
  či dané políčko má výšku väčšiu alebo
  menšiu ako level }
procedure fill(x, y, z, level,
               na1, na2 : integer);
var
  i, j, top : integer;
begin
  top := 1;
  stack[1, 1] := x; stack[1, 2] := y;
  while top > 0 do begin
    {vyberieme zo zásobníka políčko}
    x := stack[top, 1];
    y := stack[top, 2]; dec(top);
    {spracujeme}
    if a[x, y] = z then begin
      if x < level then
        a[x, y] := na1
      else
        a[x, y] := na2;
      {všetkých susedov dáme do zásobníka}
      for i := -1 to 1 do
        for j := -1 to 1 do
          if (i * j = 0) and
             (a[x + i, y + j] <> 0) then begin
            inc(top);
            stack[top, 1] := x + i;
            stack[top, 2] := y + j;
          end;
    end;
  end;
end;

{hlavný program}
begin
  assign(f, 'stavba.in');
  reset(f);
  assign{g, 'stavba.out');
  rewrite(g);
  readln(f, m, n);
  {načítame vstup }
  for i := 1 to M do
    for j := 1 to N do
      read(f, a[i, j]);
  { podlaha je zarážka }
  for i := 1 to N do begin
    a[0, i] := 0;
    a[M + 1, i] := -1;
  end;
  for i := 1 to M do begin
    a[i, 0] := 0;
    a[i, m + 1] := 0;
  end;
  {označíme bloky}
  c := 1;
  for i := 1 to M do
    for j := 1 to N do
      if a[i, j] = 1 then begin
        inc(c);
        fill(i, j, 1, -1, c, c);
      end;
  {kým sa niečo môže posúvať}
  este := true;
  while (este) do begin
    este := false;
    for i := M downto 1 do
      for j := 1 to N do begin
        {prepisujeme políčka}
        case a[i - 1, j] of
          -2     :
            a[i, j] := -1;
          -1, 0  :
            if a[i, j] <> -1 then
              a[i, j] := 0;
            else begin
              este := true;
              a[i, j] := a[i-1, j];
              {keď dopadla časť,  označíme }
              if a[i + 1, j] = -1 then
                fill(i, j, a[i, j], i, -2, -1);
            end;
        end;
      end;
  end;
```

```
    vyska := 0;
    for j := 1 to N do begin
      i := 0;
      while a[i, j] = 0 do inc(i);
      if M - i + 1 > vyska then
        vyska := M - i + 1;
    end;
    writeln(g, 'Vyska mesta je ', vyska, '.');
    close(f); close(g);
end.
```

4.2 Finále

4.2.1 O zaujímavom trojuholníku

Tento príklad je veľmi jednoduchý, stačí si uvedomiť dve veci:

- Keďže platí vzťah $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ pre $n, k > 0$, vieme každý ďalší riadok efektívne vyrátať z hodnôt riadku predchádzajúceho sčítavaním dvoch susedných čísel. Nie je však potrebné pamätať si celý trojuholník, stačia nám z neho vždy iba dva riadky.
- Kombinačné čísla veľmi rýchlo rastú. My však nepotrebujeme vedieť hodnotu kombinačného čísla, ale iba či je párne alebo nie. Ak vieme, či sú nejaké dve čísla a a b párne alebo nepárne, vieme zistiť, či bude ich súčet párny alebo nie. Preto si budeme pamätať iba zvyšky kombinačných čísel po delení dvoma.

Celý algoritmus potom spočíva už iba v postupnom generovaní a vypisovaní požadovaného počtu riadkov Pascalovho trojuholníka.

```
program O_zaujimavom_trojuholniku;
var
  riadok, novyriadok :
     array[0..100] of byte;
  fin, fout : text;
  n : integer;
  i, j : integer;
begin
  assign(fin, 'pascal.in');
  assign(fout, 'pascal.out');
  reset(fin); rewrite(fout);
  readln(fin, n);
  while n > 0 do begin
    {vypis n riadkov trojuholnika}
    for i := 0 to n - 1 do begin
      {vypocet hodnot riadku z hodnot
       predchadzajuceho riadku}
      novyriadok[0] := 1;
      for j := 1 to i - 1 do
        {spocitam ostatne hodnoty riadku}
        novyriadok[j] :=
          (riadok[j - 1] + riadok[j]) mod 2;
      novyriadok[i] := 1;
      {vypis riadku a kopirovanie poli}
      for j := 0 to i do begin
        riadok[j] := novyriadok[j];
        if riadok[j] = 0 then
          write(fout, '.')
        else
          write(fout, '*');
      end;
      writeln(fout);
    end;  {for i}
    {nacitanie noveho n}
    readln(fin, n);
  end; {while}
  close(fin); close(fout);
end.
```

4.2.2 Kôň

Tento príklad patrí medzi ľahšie. Na jeho úspešné vyriešenie si stačí pozorne prečítať zadanie a dôsledne ho naprogramovať, aby program správne rozlíšil dané štyri možnosti zadanej trasy koňa. Trasa mohla byť nekorektná, vtedy sa vypísalo `Nekorektna` a b, kde a a b sú riadok a stĺpec prvého políčka trasy, kde sa porušili pravidlá. Keď bola trasa korektná, t.j. spĺňala pravidlá pohybu šachového koňa, bolo treba rozlíšiť, či sa skočilo dvakrát na to isté políčko a keď nie, či sa prešli všetky políčka.

Program sa dal spraviť dvoma spôsobmi:

- každú trasu koňa sme prečítali dvakrát. V prvom prechode sme zistili, či je nekorektná a keď nebola, v druhom prechode sme zistili, ktorý z troch prípadov nastal,
- trasu čítame iba raz pričom môžeme byť v niektorom zo štyroch stavov.

Vzorový program je urobený druhým spôsobom. Kľúčová je funkcia *Este*, ktorá modifikuje argumenty *Ok* a *NieDvakrat* a jej hodnota je *true* pokým čítam jednu trasu koňa. *Ok* je `false` keď je trasa nekorektná. *NieDvakrat* je *true* keď trasa neprechádza dvakrát tým istým políčkom. Pokým má *Ok* hodnotu *true*, zisťujeme či trasa neprechádza dvakrát jedným políčkom. V prípade, že áno zapamätáme si ktoré to je a nastavíme *NieDvakrat* na *false*. Keď nájdeme nekorektný skok, zapamätáme si políčko kam sa nedá skočiť a nastavíme *Ok* na *false* a zvyšok ťahu už iba dočítame bez kontroly.

```
program kon;
const
  MaxVelkost = 100;
var
  f, g : text;
  m, n, p, r, px, py, x, y, i, j,
  vx, vy, Navstivene : integer;
  Ok, NieDvakrat : Boolean;
  Volno : array[1..MaxVelkost,
                1..MaxVelkost] of boolean;

  function Este(var Ok, NieDvakrat : Boolean;
                x, y : integer) : Boolean;
  begin
    if (x <> 0) and (y <> 0) then begin
      Este := true;
      if Ok then begin
        if abs(px - x) *
           abs(py - y) = 2 then begin
          if NieDvakrat then
            if Volno[x, y] then
              Volno[x, y] := false
            else begin
              NieDvakrat := false;
              vx := x; vy := y
            end
        end else begin
          Ok := false;
          vx := x; vy := y
        end
      end;
      inc(Navstivene)
    end else
      Este := false;
    px := x; py := y
  end;

begin
  assign(f, 'kon.in'); reset(f);
  assign(g, 'kon.out'); rewrite(g);
  read(f, m, n);
  while (m <> 0) and (n <> 0) do begin
     for i := 1 to m do
       for j := 1 to n do
         Volno[i, j] := true;
     read(f, p, r); px := p; py := r;
     Volno[p, r] := false;
     Ok := true;
     NieDvakrat := true;
     Navstivene := 1;
     repeat
       read(f, x, y)
     until not Este(Ok, NieDvakrat, x, y);
     if Ok and NieDvakrat then
       if Navstivene = m * n then
         writeln(g, 'OK')
       else
         writeln(g, 'Neuplna')
     else begin
       if Ok then
         write(g, 'Dvakrat ')
       else
         write(g, 'Nekorektna ');
       writeln(g, vx, ' ', vy)
     end;
     read(f, m, n)
  end;
  close(g); close(f)
end.
```

4.2.3 Krížové odkazy

Tento príklad bol vo svojej podstate veľmi jednoduchý. Celá jeho zložitosť spočívala v nevyhnutnosti použiť dynamické dátové štruktúry, pretože zo zadania bolo zrejmé, že inak by sa nám nemuseli údaje zmestiť do pamäti. Najjednoduchšie bolo vytvoriť si pole smerníkov na identifikátory a v druhom poli si pamätať k identifikátorom ich čísla a začiatky zoznamov výskytov. Šikovnejšie bolo vytvoriť pole smerníkov na štruktúry obsahujúce trojice: identifikátor, jeho číslo a smerník na začiatok zoznamu jeho výskytov.

Vstupný súbor prečítame v dvoch etapách. V prvej prečítame prvú časť vstupného súboru a vytvoríme si tabuľku identifikátorov s ich číslami. V druhej etape postupne čítame zvyšné časti a príslušným identifikátorom komponentov pridáme do zoznamu výskytov číslo daného okna. Využijeme pri tom tabuľku skonštruovanú v prvej etape. Výhodné je v prvej etape vytvoriť tabuľku utriedenú podľa identifikátorov, aby sa nám v druhej etape ľahko identifikátory hľadali. Pred výpisom je zase výhodné utriediť tabuľku identifikátorov podľa ich čísiel. Zoznamy výskytov identifikátorov vytvárame tak, aby boli utriedené podľa čísiel.

```
program Krizove_odkazy;
const
  MaxN = 1000;
type
  PVyskyt = ^Tvyskyt;
  TVyskyt = record
    Dalsi : PVyskyt;
    Id : integer;
  end;
  PStruktura = ^TStruktura;
  TStruktura = record
    Ident : string[100];
    Cislo : integer;
    Vyskyty : PVyskyt;
  end;
  TTriedenie = (PodlaIdent, PodlaCisla);
var
  f, g : text;
  Odkazy : array[1..MaxN] of PStruktura;
  n, i, Kluc : integer;

procedure Utried(Ako : TTriedenie);

  function Kluc(i : integer) : Pointer;
  begin
    if Ako = PodlaCisla then
      Kluc := @(Odkazy[i]^.Cislo)
    else
```

```
      Kluc := @(Odkazy[i]^.Ident)
  end;

  function JeMenej(k1, k2 : Pointer) : Boolean;
  type
    pinteger = ^integer;
    pstring  = ^string;
  begin
    if Ako = PodlaCisla then
      JeMenej := pinteger(k1)^ < pinteger(k2)^
    else
      JeMenej := pstring(k1)^ < pstring(k2)^
  end;

  procedure Tried(l, r: Integer);
  var
    i, j, k : integer;
    x : pointer;
    y : PStruktura;
  begin
    i := l; j := r; x := Kluc((l+r) div 2);
    repeat
      while JeMenej(Kluc(i), x) do
        i := i + 1;
      while JeMenej(x, Kluc(j)) do
        j := j - 1;
      if i <= j then
      begin
        y := Odkazy[i];
        Odkazy[i] := Odkazy[j];
        Odkazy[j] := y;
        i := i + 1; j := j - 1;
      end;
    until i > j;
    if j - l < r - i then begin
      k := r; r := j; j := k;
      k := l; l := i; i := k
    end;
    if l < j then Tried(l, j);
    if i < r then Tried(i, r)
  end;

begin
  Tried(1, n)
end;

procedure CitajIdentifikatory;
var
  Buf : string;
  Medzera, TmpCislo, Kod : integer;
begin
  Kluc := 1; n := 0;
  readln(f, Buf);
  while buf <> '' do begin
    for i := 1 to Length(Buf) do
      Buf[i] := UpCase(Buf[i]);
    inc(n); new(Odkazy[n]);
    Medzera := pos(' ', Buf);
    with Odkazy[n]^ do begin
      Ident := copy(Buf, 1, Medzera - 1);
      Val(Copy(Buf, Medzera, length(Buf)),
          TmpCislo, Kod);
      Cislo := TmpCislo;
      Vyskyty := nil
    end;
    readln(f, Buf)
  end;
  Utried(PodlaIdent)
end;

procedure CitajOkna;

  function Hladaj(var Buf : string) :
    Integer;
  {binrne vyhadvanie}
  var
    i, d, h, s : integer;
  begin
    for i := 1 to Length(Buf) do
      Buf[i] := UpCase(Buf[i]);
    d := 1; h := n;
    repeat
      s := (d + h) div 2;
      if Odkazy[s]^.Ident < Buf then
        d := s + 1
      else
        h := s - 1
    until (Odkazy[s]^.Ident = Buf) or (d > h);
    Hladaj := s
  end;

  procedure Vloz(var Ptr : PVyskyt;
                 Id : integer);
  var
    Novy, TmpPtr, Predchodca : PVyskyt;
  begin
    New(Novy);
    Novy^.Id := Id; Novy^.Dalsi := nil;
    if (Ptr = nil) or
       ((Ptr <> nil) and
        (Id < Ptr^.Id)) then begin
      Novy^.Dalsi := Ptr;
      Ptr := Novy
    end else begin
      TmpPtr := Ptr^.Dalsi;
      Predchodca := Ptr;
      while (TmpPtr <> nil) and
            (TmpPtr^.Id < Id) do begin
        Predchodca := TmpPtr;
        TmpPtr := TmpPtr^.Dalsi
      end;
      Novy^.Dalsi := TmpPtr;
      Predchodca^.Dalsi := Novy;
    end
  end;

var
  Buf : string;
  Co : integer;
begin
  readln(f, Buf);
  while not eof(f) do begin
    Co := Odkazy[Hladaj(Buf)]^.Cislo;
    readln(f, Buf);
    while Buf <> '' do begin
      Vloz(Odkazy[Hladaj(Buf)]^.Vyskyty, Co);
      readln(f, Buf)
    end;
    readln(f, Buf)
  end
end;

procedure VypisVysledky;
var
  i : integer;
  ptr : PVyskyt;
begin
  assign(g, 'odkazy.out'); rewrite(g);
  Utried(PodlaCisla);
  for i := 1 to n do begin
    writeln(g, Odkazy[i]^.Cislo);
    ptr := Odkazy[i]^.Vyskyty;
```

```
    if ptr <> nil then begin
      write(g, ptr^.Id);
      ptr := ptr^.Dalsi;
      while ptr <> nil do begin
        write(g, ' ', ptr^.Id);
        ptr := ptr^.Dalsi
      end
    end;
    writeln(g)
  end;
  close(g)
end;

begin
  assign(f, 'odkazy.in'); reset(f);
  for i := 1 to MaxN do Odkazy[i] := nil;
  CitajIdentifikatory;
  CitajOkna;
  close(f);
  VypisVysledky;
end.
```

4.2.4 O Bonifácových číslach

Označme si Bonifácove číslo $X = x_m...x_1$. Teraz hľadajme $B = b_n...b_1$ z ktorého mohlo X vzniknúť. V prípade, že X začína jednotkou (t.j. $x_m = 1$) musel pri sčítaní nastať prenos cez najvyššie rády a teda platí $n = m - 1$. Pre $n \leq 2$ nie je problém priamo nájsť číslo B, alebo zistiť, že vhodné B neexistuje. Pre $n > 2$ rozlíšme nasledujúce prípady:

V prípade $x_{m-1} = 0$ a $x_1 = 9$ musí byť $b_1 + b_n = 9$ a číslo $1x_{m-2}...x_2$ muselo vzniknúť z čísla $b_{n-1}...b_2$. (Toto už môžme hľadať rekurzívne.)

- V prípade $x_{m-1} = x_1$ musí byť $b_1 + b_n = 10 + x_1$ a číslo $x_{m-1}...x_2 - 1$ muselo vzniknúť z čísla $b_{n-1}...b_2$. Zrejme musí byť $x_1 < 9$ a $x_{m-1}...x_2 > 0$.
- V prípade $x_{m-1} = x_1 + 1$ musí byť $b_1 + b_n = 10 + x_1$ a číslo $1x_{m-1}...x_2 - 1$ muselo vzniknúť z čísla $b_{n-1}...b_2$. Zrejme musí byť $x_1 < 9$.
- Ak nenastal žiaden z predchádzajúcich prípadov, potom vhodné B neexistuje

V prípade, že X nezačína jednotkou, prenos pri sčítaní cez najvyššie rády nenastal a teda platí $n = m$. Pre $n \leq 2$ môžme priamo zistiť číslo B, alebo zistiť, že neexistuje. Pre $n > 2$ rozlíšime nasledujúce prípady:

- V prípade $x_{m-1} = x_1$ musí byť $b_1 + b_n = x_1$ a číslo $x_{m-1}...x_2$ muselo vzniknúť z čísla $b_{n-1}...b_2$. Zrejme musí byť $x_1 > 1$.
- V prípade $x_{m-1} = x_1 + 1$ musí byť $b_1 + b_n = x_1$ a číslo $1x_{m-1}...x_2$ muselo vzniknúť z čísla $b_{n-1}...b_2$. Zrejme musí byť $x_1 > 1$.
- Ak nenastal žiaden z predchádzajúcich prípadov, potom vhodné B neexistuje

Aplikáciou týchto myšlienok a zlúčením spoločných vlastností jednotlivých prípadov dostaneme nasledovný program:

```
program Cisla;
var
  x : array[0..100] of integer;
  n, i, j : integer;
  f, g : Text;
  c : Char;

function Zmensi(a, b : integer) : Boolean;
begin
  while (b <= a) and (x[b] = 0) do begin
    x[b] := 9; b := b + 1
  end;
  x[b] := x[b] - 1;
  Zmensi := b <= a;
end;

function Over(a, b : integer) : Boolean;
var
  p, c : integer;
begin
  Over := false;
  if x[a] = 1 then begin
    p := 1; a := a - 1;
  end else
    p := 0;
  while a > b + 1 do begin
    p := p * 10 + x[a] - x[b];
    c := x[b];
    if p >= 10 then begin
      c := c + 10;
      x[a] := p - 10;
      if Not Zmensi(a, b + 1) then exit;
      p := x[a];
    end;
    if (c < 2) or (c > 18) or
       (p < 0) or (p > 1) then exit;
    a := a - 1; b := b + 1
  end;
  if (x[a] <= 1) and (p = 0) then exit;
  if a = b then
    Over := (x[a] Mod 2 = 0)
  else
    Over := (x[a] = x[b] + p)
end;

begin
  Assign(f, 'cisla.in'); Reset(f);
  Assign(g, 'cisla.out'); Rewrite(g);
  Readln(f, n);
  while n > 0 do begin
    i := 100;
    while not Eoln(f) do begin
      Read(f, c);
      x[i] := Ord(c) - Ord('0');
```

```
    i := i - 1
  end;
  Readln(f);
  if Over(100, i + 1) then
    Writeln(g, 'ANO')
  else
    Writeln(g, 'NIE');
  n := n - 1;
end;
Close(g); Close(f)
end.
```

4.2.5 O námestí

Existencia prechádzky zrejme nezávisí od začiatočnej polohy starostu. Keď je dĺžka jednej strany námestia $1m$, potom je prechádzka možná iba v prípade, že dĺžka druhej strany je najviac $3m$ a kocúr stojí na kraji námestia.

Ďalej sa budeme zaoberať prípadmi, keď sú obe strany dlhšie ako $1m$. Ofarbime si námestie ako šachovnicu, pričom kachličku v ľavom hornom rohu nafarbíme na bielo. Keď prechádzka existuje, potom sa starosta striedavo nachádza na čiernych a bielych políčkach a teda ich na námestí musí byť rovnaký počet (políčko na ktorom je kocúr nerátame). Z toho vyplýva, že pre námestia, kde je rôzny počet čiernych a bielych políčok prechádzka neexistuje. Teda pre prípady, keď je plocha námestia párna, alebo je plocha nepárna a kocúr stojí na čiernom políčku vieme, že prechádzka neexistuje.

Ďalej uvažujme iba námestia, ktoré sme týmto testom nevylúčili a ukážeme návod na dôkaz, že prechádzka vždy existuje. Keď máme nejaký, kocúra neobsahujúci, obdĺžnik so stranou nepárnej dĺžky, potom existuje cesta z rohu do protiľahlého rohu prechádzajúca cez všetky políčka obdĺžnika. Ak je na námestí kocúr v strede, potom môžme námestie pokryť štyrmi takýmito obdĺžnikmi tak, aby zloženie ciest cez obdĺžniky tvorilo hľadanú prechádzku. Podobne možno nájsť prechádzky pre ostatné polohy kocúra (nakreslite si to).

```
program Namestie;
var
  n, s, d, i, j, kx, ky : Integer;
  f, g : Text;
  c : Char;

begin
  Assign(f, 'namestie.in'); Reset(f);
  Assign(g, 'namestie.out'); Rewrite(g);
  readln(f, n);
  while n > 0 do begin
    readln(f, s, d);
    for i := 1 to d do begin
      for j := 1 to s do begin
        read(f, c);
        if c = 'K' then begin
          kx := i; ky := j;
        end;
      end;
      readln(f);
    end;
    if (s = 1) or (d = 1) then
      if ((kx + ky = 2) or (kx + ky = s + d))
         and (s + d <= 4) then
        writeln(g, 'ANO')
      else
        writeln(g, 'NIE')
    else
      if (s * d mod 2 = 0) or
         ((kx + ky) mod 2 = 1) then
        writeln(g, 'NIE')
      else
        writeln(g, 'ANO');
    n := n - 1;
  end;
  Close(g); Close(f);
end.
```

4.2.6 O optickom prístroji

V tomto príklade bolo treba nájsť sled, ktorý v každom zrkadle, cez ktoré prechádza mení smer a žiadna hrana sa v ňom nevyskytuje dvakrát. To, že musí v každom zrkadle meniť smer je zrejmé z toho, že lúč sa od zrkadla odrazí. Treba ešte zaručiť aby počas celého behu lúča boli zrkadlá nastavené rovnako. Nech niektoré zrkadlo sa počas behu lúča preklopí. Potom ak naňho lúč nenarazil dvakrát pri rôznych preklopeniach, tak to nevadí. Ak naňho ale narazil dvakrát pri rôznych preklopeniach, potom jednoduchou analýzou prípadov zistíme, že po niektorej hrane sa prešlo dvakrát. Teda sled, v ktorom sa dvakrát prejde po tej istej hrane nevyhovuje. Naopak, ak máme sled, ktorý neprejde dvakrát po tej istej hrane, tento je vyhovujúci.

V programe si do zásobníka ukladáme zrkadlá príslušného sledu spolu s aktuálnym otočením. Vyberieme zo zásobníka položku a skúsime predĺžiť lúč. Ak sa nám to nepodarí (lúč vyletel, prešlo sa dvakrát po tej istej hrane), skúsime natočiť zrkadlo ináč. Ak sa to nedá, vrátime sa o jedno späť.

```
program zrkadla;
const
  maxs = 100;
  sx : array[0..3] of integer =
```

```
    (0, 1, 0, -1);
  sy : array[0..3] of integer =
    (-1, 0, 1, 0);
var
  f, g : text;
  m, n, i, j : integer;
  a, b : array[1..maxs, 1..maxs] of byte;

function zisti(x, y, k : integer) : boolean;
var
  i : integer;
begin
  zisti := true;
  if (y = m) and (x = n + 1) then exit;

  if (y >= 1) and (y <= m) and
     (x >= 1) and (x <= n) then begin
    if a[y, x] = 0 then begin
      if zisti(x + sx[k], y + sy[k], k) then
        exit;
    end else begin
      if b[y, x]=0 then begin
        for i := 1 to 2 do begin
          b[y, x] := i;
          k := k xor (b[y, x] * 2 - 1);
          if zisti(x + sx[k],
                   y + sy[k], k) then exit;
          k := k xor (b[y, x] * 2 - 1);
        end;
        b[y, x] := 0;
      end else begin
        k := k xor (b[y, x] * 2 - 1);
        if zisti(x + sx[k],
                 y + sy[k], k) then exit;
        k := k xor (b[y, x] * 2 - 1);
      end;
    end;
  end;
  zisti := false;
end;

var
  cislo  : integer;
begin
  assign(f, 'zrkadla.in'); reset(f);
  assign(g, 'zrkadla.out'); rewrite(g);
  cislo := 0;
  readln(f, m, n);
  while m > 0 do begin
    {nacitame mriezku}
    for i := 1 to m do
      for j := 1 to n do begin
        read(f, a[i, j]);
        b[i, j] := 0;
      end;
    inc(cislo);
    write(g, 'Vstup cislo ', cislo, ': ');
    {zistime,  ci sa da prejst}
    if zisti(1, 1, 1) then
      writeln(g, 'DA SA.')
    else
      writeln(g, 'NEDA SA.');
    readln(f, m, n);
  end;
  close(g); close(f);
end.
```

4.2.7 Tucty

Tento príklad bol veľmi ľahký, len si bolo treba uvedomiť niekoľko vecí:

1. So skloňovaním sa nemusíme zaoberať, keďže na prvé tri písmenká slov nemá skloňovanie vplyv a podľa nich môžeme všetky slová rozlíšiť.
2. Celé číslo stačí prejsť raz zľava do prava. Evidujeme si dve premenné, pri načítaní ľubovoľnej časti čísla túto "prinásobíme" ku prvej premennej a pri načítaní "A" alebo "." prvú premennú prirátame ku druhej. Výsledkom bude obsah druhej premennej.
3. Dĺžka čísla nebola obmedzená, takže sa nedalo načítať celé číslo do reťazca (ten má totiž iba 255 znakov); bolo treba postupovať po slovách.
4. Pri nevhodnom počítaní mohli niektoré medzivýsledky presiahnuť 32767!

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

FILE *f,*g;

int porovnaj(char *a,char *b) {
  int i,j;

  j=1;
  for (i=0;(i<strlen(a))&&(i<strlen(b));i++)
    j=j&&(a[i]==b[i]);
  return j;
}

int spracujjedno() {
  char c[50],d;
  int suc1,suc2,del;

  fscanf(f," %s",c);
  if (c[0]=='.') return 0;
  suc1=1; suc2=0; del=1;
  do {
    d=c[strlen(c)-1];
    if ((del>1) && (suc1>1))
      {suc1=suc1/del; del=1;}
    if (porovnaj(c,"A"))
      {suc2+=(suc1/del); suc1=1; del=1;}
    if (porovnaj(c,"PAR")) suc1*=2;
    if (porovnaj(c,"TUC")) suc1*=12;
    if (porovnaj(c,"KOP")) suc1*=60;
    if (porovnaj(c,"VEL")) suc1*=144;
    if (porovnaj(c,"STV")) del=4;
    if (porovnaj(c,"POL")) del=2;
    if (porovnaj(c,"TRE")) del=3;
    if (porovnaj(c,"DES")) del=10;
    if (d!='.') fscanf(f," %s",c);
  } while (d!='.');
  suc2+=(suc1/del);
  return suc2;
```

```
}

void main() {
  int u;

  f=fopen("TUCTY.IN","rt");
  g=fopen("TUCTY.OUT","wt");
  while ((u=spracujjedno())!=0) {
    fprintf(g,"%d\n",u);
  }
  fclose(f); fclose(g);
}
```

4.2.8 Mestá

Tento príklad patril medzi tie ťažšie, aj keď nebol ťažký tým, ako ho naprogramovať, ale tým, ako prísť na jednoduchý a správny algoritmus.

Najjednoduchším spôsobom, ako prísť na riešenie, bolo pouvažovať nad doplnkom grafu zadaného zadaním (doplnok grafu je graf, kde tam, kde v pôvodnom grafe viedla hrana, hrana nevedie a naopak). Ak A a B sú v pôvodnom grafe úplne poprepájané, potom v doplnku nesmie medzi nimi viesť žiadna hrana (a naopak, ak medzi A a B nevedie v doplnku žiadna hrana, potom v pôvodnom grafe sú A a B úplne poprepájané). Naša úloha teda vedie na hľadanie počtu súvislých komponentov v doplnku pôvodného grafu. Táto úloha je už veľmi jednoduchá a ak neviete, ako ju riešiť, ľahko to zistíte, ak sa pozriete do programu.

```
#include <stdio.h>
#include <stdlib.h>
FILE *f,*g;
int m,n,a[101][101],k[101];

int nacitaj() {
  int i,j,x,y;
  fscanf(f," %d",&n);
  if (n<0) return 0;
  // vyjednickuj pole
  for (i=1;i<=n;i++) {
    for (j=1;j<=n;j++) {
      a[i][j]=1;
    }
  }
  // nacitaj nehrany
  fscanf(f," %d",&m);
  for(i=0;i<m;i++) {
    fscanf(f," %d %d",&x,&y);
    a[x][y]=0;a[y][x]=0;
  }
  return 1;
 }

void prehladajkomponent(int u) {
  int i;
  k[u]=1;
  for (i=1;i<=n;i++) {
    if((!k[i])&&(a[u][i]))
      prehladajkomponent(i);
  }
}

int pocetkomponentov() {
  int i,pk=0;
  for(i=1;i<=n;i++) k[i]=0;
  for(i=1;i<=n;i++) {
    if (!k[i]) {
      pk++; prehladajkomponent(i);
    }
  }
  return pk;
}

void main() {
  f=fopen("MESTA.IN","rt");
  g=fopen("MESTA.OUT","wt");
  while (nacitaj()) {
    fprintf(g,"%d\n",pocetkomponentov());
  }
  fclose(f);
  fclose(g);
}
```

5.1 Domáce kolo

5.1.1 O horolezcovi Pištovi

V tomto príklade bolo treba prehľadávať veľkú maticu. Takú veľkú, že zaberala viac ako 64KB, čo je limit pre statické dáta v TurboPascale. Vymysleli ste dve riešenia: buď dynamicky naalokujem dve polia, alebo si pamätám menej informácie (aby sa jedno políčko zmestilo do 1 byte). Algoritmus prehľadávania je jednoduchý – prechádzam po políčkach a pamätám si tie, na ktorých som už bol (tam nepôjdem) a tie, ktoré s nimi susedia (na tie môžem ísť, takže si z nich v každom kroku jedno vyberiem). Problém bol opäť v tom, že ak si všetky potenciálne políčka mám pamätať, môže toho byť veľa (napr. pri prehľadávaní do hĺbky preteká zásobník). Niektorí ste to riešili tak, že ste vždy prechádzali celú maticu, aby ste našli suseda. Lepšie riešenie bolo použiť dynamicky alokovanú frontu (alebo zásobník) alebo si pamätať návratovú informáciu priamo v matici.

Vzorové riešenie si v matici pamätá byty s nasledovným významom: spodné štyri bity - či sa dá prejsť na susedné políčko v príslušnom smere, nasledujúci bit určuje, či je toto políčko maximum v riadku, ďalšie dva

bity určujú číslo smeru, z ktorého sme pri prehľadávaní prišli na toto políčko a posledný bit určuje, či sme už toto políčko prehľadali. Pre každý riadok si pamätáme hodnotu maxima, aby sme vedeli nájsť maximálny prvok.

Pre tých, čo sa im chce ešte trochu pošpekulovať prezradíme, že sa dá napísať riešenie, v ktorom sa matica prečíta iba raz a v pamäti budú vždy iba jej dva susedné riadky. Časová zložitosť takéhoto riešenia bude trochu väčšia, ale pri dobrej implementácii nie o veľa.

```
#include <stdio.h>
#include <stdlib.h>

#define BOL 128
#define MAXVAL 16

int si[]={-1,0,1,0};   // vektory smerov
int sj[]={0,1,0,-1};

int M,N;
char A[202][202];      // matica
// maximalny prvok v riadku
int max[202],Max;

void main() {
 FILE *f,*g;
 int riadok[2][202];   // vstupny buffer
 int i,j,r,mx,ti,tj,sm;

 f=fopen("pista.in","r");
 g=fopen("pista.out","w");
 fscanf(f,"%d %d ",&M,&N);
 for(i=0;i<N;i++)
   fscanf(f,"%d",riadok[0]+i);
 r=0;Max=0;
 for(i=0;i<M;i++) {     // vyrobime maticu
   for(j=0,mx=0;j<N;j++)
     if(riadok[r][j]>mx)
       mx=riadok[r][j];
   max[i]=mx;
   if(mx>Max) Max=mx;
   for(j=0;j<N;j++) {
     if (riadok[r][j]==mx)
       A[i][j]|=MAXVAL;
     if (i<M-1)
       fscanf(f,"%d",riadok[1-r]+j);
     if ((j>0)&&(abs(riadok[r][j-1]
                     -riadok[r][j])<2))
       A[i][j]|=8;
     if ((j<N-1)&&(abs(riadok[r][j+1]
                       -riadok[r][j])<2))
       A[i][j]|=2;
     if ((i<M-1)&&
         (abs(riadok[r][j]
              -riadok[1-r][j])<2)) {
           A[i][j]|=4; A[i+1][j]|=1;
     }
   }
   r=1-r;
 }
 fscanf(f,"%d %d",&ti,&tj);
 ti--;tj--;
 i=ti;j=tj;sm=0;
 while(1) {               // prechadzame
   if ((A[i][j]&MAXVAL)&&(max[i]==Max)) {
     fprintf(g,"ANO\n");
     goto end;
   }
   A[i][j]=(A[i][j]&BOL)+sm*32+
           (A[i][j]&127)%32;
   A[i][j]|=BOL;
   for(sm=0;sm<4;sm++)    // ako dalej ?
     if ((A[i][j]&(1<<sm))&&
         !(A[i+si[sm]][j+sj[sm]]&BOL)) {
       i+=si[sm];
       j+=sj[sm];
       break;
     }
   if(sm==4)           // treba sa vratit
   if ((i==ti)&&(j==tj)) break;
   else {              // vratime sa
    sm=(((A[i][j]&127)/32)%4+2)%4;
    i+=si[sm];
    j+=sj[sm];
    sm=((A[i][j]&127)/32)%4;
   }
 }
 fprintf(g,"NIE\n");
 end:;
 fclose(f);
 fclose(g);
}
```

5.1.2 Digitálne korytnačky

Medzi Vašimi riešeniami príkladu o korytnačkách sa vyskytli dva druhy správnych riešení. Zriedkavejšie riešenia sa pokúšali riešiť sústavu kongruencii ("rovníc o zvyškoch"). Častejšie, jednoduchšie a lepšie riešenia digitálne simulovali pohyb korytnačiek a v každom kroku zisťovali, či štartom v danom kroku možno prejsť na druhý breh. Riešeniu stačilo simulovať kroky až kým sa zopakoval stav zo začiatku (tento sa zopakuje po $2.\,\mathrm{nsd}(A[1],...,A[n])$ krokoch), alebo sa podarilo prejsť na druhý breh. Za správne riešenie ste mohli získať 6-10 bodov podľa kvality. Naše vzorové riešenie je tiež založené na simulácii pohybu korytnačiek.

```
program Digitalne_Korytnacky;

var
  Fin, Fout :Text;
  n, i :Integer;
  a, b :array[1..500] of Integer;
  Rie, Kon :Boolean;

begin
  Assign(Fin,'koryt.in');
  Reset(Fin);
  Assign(Fout,'koryt.out');
  Rewrite(Fout);
  Read(Fin,n);
```

```
while n > 0 do begin
  for i := 1 to n do begin
    read(Fin,a[i]);
    b[i] := 0;
  end;
  repeat
    Rie := true;
    Kon := true;
    for i := 1 to n do begin
      b[i] := (b[i] + 1) mod (2 * a[i]);
      Kon := Kon and (b[i] = 0);
      Rie := Rie and ((b[i] + i - 1) mod
                                    (2 * a[i]) < a[i]);
    end;
  until Rie or Kon;
  if Rie then
    Writeln(fout,'ANO')
  else
    Writeln(fout,'NIE');
  Read(Fin,n);
end;
Close(Fout);
Close(Fin);
end.
```

5.1.3 Nové kúzlo

Najprv niečo o bodovaní. Za zlé riešenia (napríklad také, ktoré dali dobrý výsledok iba v prípade, že všetky karty v kope sú rôzne) ste mohli získať 1 až 2 body. Riešenia, ktoré generovali všetky možnosti, mohli získať 5 až 6 bodov podľa časovej zložitosti, pričom za exponenciálnou pamäťovú zložitosť som strhávala ďalší bod. Za riešenia, ktoré (podobne ako vzorové riešenie) pracovali v čase $O(n^2)$, ste mohli dostať 10 bodov, riešenia s časovou zložitosťou $O(n^3)$ dostali 9 bodov.

Vzorové riešenie je pomerne jednoduché. Nech je počet kariet n a nech ich čísla sú $a_1, a_2, \ldots a_n$. Postupne budem vypočítavať počet možností pre väčšiu a väčšiu časť kopy. Označme M_i množinu všetkých postupností čísel kariet, aké môžu vzniknúť po odobratí niekoľkých kariet z kopy $a_1, a_2, \ldots a_i$. Ďalej nech s_i je veľkosť množiny M_i. Výsledkom práce algoritmu je teda číslo s_n.

Množina M_0 obsahuje iba jedinú postupnosť (dĺžky 0), a teda $s_0 = 1$. Množina M_1 obsahuje postupnosť dĺžky 0 a postupnosť (a_1), takže $s_1 = 2$. Predpokladajme teraz, že máme vypočítané hodnoty $s_0, s_1, \ldots s_{i-1}$ a chceme vypočítať hodnotu s_i. Každá z postupností obsiahnutých v M_{i-1} patrí aj do M_i. Okrem nich do M_i patria postupnosti, ktoré vznikli z niektorej postupnosti v M_{i-1} pridaním karty s číslom a_i na koniec. Nech M'_i je množina týchto postupností, t.j. $M'_i = \{(b_1, b_2, \ldots b_k, a_i) : (b_1, b_2, \ldots b_k) \in M_{i-1}\}$. Potom platí $M_i = M_{i-1} \cup M'_i$. Pre veľkosti množín platí, že $|M'_i| = s_{i-1}$ a

$$\begin{aligned} s_i = |M_i| &= |M_{i-1} \cup M'_i| \\ &= |M_{i-1}| + |M'_i| - |M_{i-1} \cap M'_i| \\ &= 2s_i - |M_{i-1} \cap M'_i|. \end{aligned}$$

Ak chceme vypočítať s_i, zostáva iba určiť veľkosť množiny $M_{i-1} \cap M'_i$, čo je počet takých postupností, ktoré sa nachádzajú v M_{i-1} a ktoré majú na konci kartu s číslom a_i. Môžu nastať dva prípady:

1. Žiadna karta s číslom a_i sa medzi kartami $a_1, a_2, \ldots a_{i-1}$ nevyskytuje. V žiadnej postupnosti z M_{i-1} sa teda karta s číslom a_i nevyskytuje a $|M_{i-1} \cap M'_i| = 0$.
2. Medzi kartami $a_1, a_2, \ldots a_{i-1}$ sa vyskytuje aspoň jedna karta s číslom a_i. Nech j je najväčšie číslo také, že $1 \leq j < i$ a $a_j = a_i$. Každá postupnosť z M_{i-1}, ktorá sa končí kartou s číslom a_i, patrí aj do M_j. Každá postupnosť z M_j, ktorá sa končí kartou s číslom a_i, mohla vzniknúť pridaním tejto karty na koniec niektorej postupnosti z M_{j-1}. Teda platí $M_{i-1} \cap M'_i = M'_j$, čiže $|M_{i-1} \cap M'_i| = s_{j-1}$.

Program teda funguje tak, že postupne vypočítava hodnoty $s_1, s_2, \ldots s_n$. Pri výpočte s_i prechodom poľa a zistí, ktorý z uvažovaných prípadov nastal, prípadne nájde hodnotu j potrebnú v druhom prípade a na základe predtým vypočítaných hodnôt vypočíta s_i. Časová zložitosť je $O(n^2)$, pamäťová $O(n)$.

Pre labužníkov ešte dodám, že pomocou triedenia alebo nejakých vhodných dátových štruktúr nie je ťažké znížiť časovú zložitosť algoritmu na $O(n \log n)$ – pre každý prvok poľa a potrebujeme iba rýchlo vedieť zistiť polohu predchádzajúceho prvku s rovnakou hodnotou.

```
program kuzlo;
const
  maxn = 100;
var
  {cisla kariet}
  a : array[1..maxn] of integer;
  {pocty moznosti}
  s : array[0..maxn] of longint;
  fin, fout : text;
  kop, k, n, i, j : integer;

begin
  assign(fin, 'kuzlo.in');
  assign(fout, 'kuzlo.out');
```

```
  reset(fin);
  rewrite(fout);
  {nacitame pocet kop kariet}
  readln(fin, kop);
  for k := 1 to kop do begin
    {nacitame pocet kariet}
    readln(fin, n);
    for i := 1 to n do  {nacitame kopu}
      read(fin, a[i]);
    s[0] := 1;
    for i := 1 to n do begin
      j := i - 1; {hladame rovnaku kartu}
      while (j > 0) and (a[j] <> a[i]) do
        dec(j);
      if j = 0 then
        {rovnaka karta neexistuje}
        s[i] := s[i - 1] * 2
      else {a[i] = a[j]}
        s[i] := s[i - 1] * 2 - s[j - 1];
    end;
    {vypis pre kopu k}
    writeln(fout, s[n]);
  end;
  close(fin);
  close(fout);
end.
```

5.1.4 Rozbal

Tento príklad bol podľa nás ľahký, napriek tomu ho riešilo iba 43 družstiev a ešte si niektorí zle prečítali zadanie. Ako ste mnohí konštatovali v komentároch k vašim riešeniam, algoritmus bol popísaný v zadaní. Podľa toho ako ste zvládli jeho implementáciu ste boli odmenení bodmi. Vyskytli sa nasledovné typy riešení:

- (do 7b) Prepisovanie do pomocného poľa. Nešikovné posúvanie – na miesto jedného posunu o $d+1$ miest, $d+1$ posunutí o 1 jedno miesto.
- (do 8b) Presne podľa zadania. Posunutie poľa v cykle alebo využitím *memmove*.
- (do 9b) Ak ste si uvedomili, že pole netreba fyzicky posúvať, ale stačí si pamätať, kde je jeho začiatok. Pole malo veľkosť podľa zadania, t.j. 512 bytov.
- (do 10b) Ako predchádzajúce riešenie a keď ste využili, že stačí pole veľkosti 256 bytov, lebo $0 \leq s < 256$. Takéto riešenie uvádzame ako vzorové.

Body sa strhávali za technické nešikovnosti a chýbajúci popis. Najčastejšou chybou bolo, že ste sa pokúšali načítať celý vstup do pamäti, jeho dĺžka však nebola obmedzená. A podobne aj rozbaliť v pamäti a potom vypísať.

```
program rozbal;
const
  {N moze byt lubovolne}
  N = 256;
var
  f, g : file of byte;
  d, s, z : byte;
  zac, i : integer;
  r : array[0..255] of byte;

begin
  assign(f, 'rozbal.in'); reset(f);
  assign(g, 'rozbal.out'); rewrite(g);
  zac := 0;
  fillchar(r, SizeOf(r), 0);
  while not eof(f) do begin
    read(f, d);
    if d > 0 then
      read(f, s);
    read(f, z);
    i := (s + zac) mod N;
    while d > 0 do begin
      r[zac] := r[i];
      write(g, r[zac]);
      zac := (zac + 1) mod N;
      i := (i + 1) mod N;
      dec(d)
    end;
    r[zac] := z;
    write(g, z);
    zac := (zac + 1) mod N;
  end;
  close(f); close(g)
end.
```

5.1.5 List

Riešenia tohoto príkladu boli štyroch typov: zlé riešenia založené na nejakom chybnom predpoklade (maximálne 3 body), riešenia skúšajúce náhodné postupnosti ťahov (3 body), riešenia systematicky prehľadávajúce všetky dosiahnuteľné konfigurácie (maximálne 5 bodov) a riešenie podobné vzorovému (maximálne 10 bodov).

Vzorové riešenie je založené na jednoduchom pozorovaní: každý štvorček rubikovho listu sa môže ocitnúť v štyroch rôznych polohách – symetrických k jeho polohe podľa stredových osí a podľa stredu rubikovho listu. Takto možno celý list rozdeliť na štvorice políčok (každá štvorica má štvorčeky na súradniciach (i,j), $(n-i+1,j)$, $(i,n-j+1)$, $(n-i+1,n-j+1)$). Štvorčeky nemôžu byť otočené ľubovoľne, ľahko ukážeme,

Z Z Z Z C C

že prípustné kombinácie sú: CC, ZZ, CC a ich rotácie.

Dostali sme teda nutnú podmienku pre to, aby list bolo možné poskladať. Ukážeme, že táto podmienka je aj postačujúca. Ukážeme postup, pomocou ktorého možno pre prvú z uvedených prípustných kombinácií preskladať list tak, aby všetky políčka tejto štvorice mali rovnakú farbu a súčasne sa nezmenila farba ostatných políčok listu. Ak je naša štvorica (i, j), $(n - i + 1, j)$, $(i, n - j + 1)$, $(n - i + 1, n - j + 1)$, potom stačí použiť tento postup: otočiť i-ty riadok, otočiť j-ty stĺpec, otočiť i-ty riadok a otočiť znova j-ty stĺpec. V ostatných prípadoch ľahko možno nájsť podobné postupy.

Všimnite si ešte, že naša podmienka je ekvivalentná s podmienkou, že každý štvorček má rovnakú farbu ako niektorý zo štvorčekov s polohami symetrickými podľa stredových osí. Táto podmienka sa ľahšie testuje.

```
program list;
const
  MAXN = 20;
var
  n : integer;
  a : array[1..MAXN, 1..MAXN] of integer;
  f, g : text;

procedure nacitaj;
var
  i, j : integer;
  c : char;
begin
  readln(f,n);
  for i := 1 to n do begin
    for j := 1 to n do begin
      read(f,c);
      if c = 'C' then
        a[i,j] := 0
      else
        a[i,j] := 1;
    end;
    readln(f);
  end;
end;

function kontroluj : boolean; var
  i, j : integer;
begin
  for i := 1 to n do begin
    for j := 1 to n do begin
      if ((a[i, j] <> a[n - i + 1, j]) and
          (a[i, j] <>
             a[i, n - j + 1])) then begin
        kontroluj := false;
        exit;
      end;
    end;
  end;
  kontroluj := true;
end;

begin
  assign(f, 'list.in'); reset(f);
  assign(g, 'list.out'); rewrite(g);
  nacitaj;
  if not kontroluj then
    write(g, 'ne');
  writeln(g, 'da sa');
  close(f); close(g);
end.
```

5.2 Finále

5.2.1 Olympiáda

Myšlienka je jednoduchá: ak zotriedime celý zoznam súťažiacich podľa názvu krajiny, budú vo výsledku všetci súťažiaci z jednej krajiny za sebou. Teraz teda stačí raz prejsť celé pole, sčítať body za krajiny a potom zotriediť podľa týchto bodov.

V programu používame C-čkovú funkciu *qsort*, v Pascale ste mohli použiť quicksort z príkladov.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

FILE *f, *g;

struct VYSL {
  char kraj[21];
  int body;
} vysledky[400];

int n;

void nacitaj() {
  char c[21];

  fscanf(f," %d",&n);
  for (int i=0; i<n; i++) {
    fscanf(f," %s %s %d",c,
      vysledky[i].kraj,
      &(vysledky[i].body));
  }
}

int porovnaj_krajiny(const void *x,
                     const void *y) {
  return strcmp(((VYSL *)x)->kraj,
                ((VYSL *)y)->kraj);
}

int porovnaj_body(const void *x,
                  const void *y) {
  if (((VYSL *)x)->body ==
      ((VYSL*)y)->body)
    return porovnaj_krajiny(x,y);
  else
    return -((VYSL *)x)->body +
            ((VYSL*)y)->body;
```

```
}

void zotried_podla_krajin() {
  qsort((void *)vysledky, n, sizeof(VYSL),
        porovnaj_krajiny);
}

void scitaj() {
  int m=0;

  for(int i=1;i<n;i++) {
    if (strcmp(vysledky[i].kraj,
               vysledky[m].kraj)) {
      m++;
      strcpy(vysledky[m].kraj,
             vysledky[i].kraj);
      vysledky[m].body = vysledky[i].body;
    } else {
      vysledky[m].body += vysledky[i].body;
    }
  }
  n=m+1;
}

void zotried_podla_bodov() {
  qsort((void *)vysledky, n, sizeof(VYSL),
        porovnaj_body);
}

void vypis() {
  for(int i=0; i<n; i++) {
    fprintf(g,"%s %d\n",
            vysledky[i].kraj,
            vysledky[i].body);
  }
}

int main() {
  f=fopen("OLYMP.IN","rt");
  g=fopen("OLYMP.OUT","wt");
  nacitaj();
  zotried_podla_krajin();
  scitaj();
  zotried_podla_bodov();
  vypis();
  fclose(f);
  fclose(g);
  return 0;
}
```

5.2.2 Kód

Pri riešení tohoto príkladu si bolo treba uvedomiť, čo hovoria podmienky 1 a 2 v zadaní. Podmienka 1 zaručovala, že kratšie kódy budú mať vždy menšiu hodnotu než dlhšie. Hľadanie kódu jednotlivých znakov môžeme rozdeliť do troch etáp.

V prvej etape spočítame, koľko znakov majú mať kódy s dĺžkou 1, 2, ..., 30. Počet znakov s dĺžkou kódu l bude uložený v $Dlzka[l]$.

V druhej etape určíme pre každú dĺžku kódu l jeho najmenšiu hodnotu h_l. Na základe 2. podmienky zo zadania vieme, že kódy s rovnakou dĺžkou budú mať po sebe idúce hodnoty. Na základe 1. podmienky vieme, že najmenšiu hodnotu kódu bude mať prvý znak s najkratšou dĺžkou kódu, nech je táto dĺžka d. Kódy majú len nezáporné hodnoty, takže najmenšia hodnota h_d bude 0. Aká bude najmenšia hodnota kódu s dĺžkou $d+1$? $h_{d+1} = 2(h_d + Dlzka[d])$. Inak povedané h_{d+1} musí byť o $Dlzka[d]$ väčšia aby bola splnená 1. podmienka a navyše kód je dlhší o jeden bit, preto násobenie dvomi. Je zrejmé že kód je prefixový. Hodnoty h_l sú v programe uložené v poli $HodnotaKodu[l]$.

V tretej etape už stačí iba jednotlivým znakom priradiť ich kódy. Aby sme dodržali 2. podmienku zo zadania, postupne pre všetky znaky v poradí ich ASCII kódov im priradíme najmenší, ešte nepoužitý, kód ich dĺžky. Výsledná hodnota kódu i-teho znaku je uložená v $Kod[i]$.

```
program PrefixovyKod;

const
  MaxDlzka = 30;
  MaxKodov = 256;
type
  TMaxDlzka = 0..MaxDlzka;
  TMaxKodov1 = 0..MaxKodov - 1;
var
  x, n : integer;
  f, g : text;
  Dlzka : array[TMaxKodov1] of integer;
  Kod : array[TMaxKodov1] of longint;
  Pocet : array[TMaxDlzka] of integer;
  HodnotaKodu : array[TMaxDlzka] of longint;

procedure Init;
var
  i : integer;
begin
  for i := 0 to MaxKodov - 1 do
    Dlzka[i] := 0;
  for i := 0 to MaxDlzka do
    Pocet[i] := 0
end;

procedure VytvorKody;
var
  i, Hodnota : integer;
  d : TMaxDlzka;
begin
  {zistime pocet kodov s rovnakou dlzkou}
  for i := 0 to MaxKodov - 1 do
    inc(Pocet[Dlzka[i]]);
  Pocet[0] := 0;
  {urcime pre kazdu dlzku
   najmensiu hodnotu kodu}
  HodnotaKodu[0] := 0;
  for i := 1 to MaxDlzka do
    HodnotaKodu[i] :=  2 *
```

```
      (HodnotaKodu[i - 1] + Pocet[i - 1]);
  {znakom priradime kody}
  for i := 0 to MaxKodov - 1 do begin
    d := Dlzka[i];
    if d > 0 then begin
      Kod[i] := HodnotaKodu[d];
      inc(HodnotaKodu[d])
    end;
  end;
  {kody vypiseme}
  for i := 0 to n do
    writeln(g, i, ' ', Dlzka[i],
                  ' ', Kod[i]);
  writeln(g)
end;
```

```
begin
  assign(f, 'kod.in'); reset(f);
  assign(g, 'kod.out'); rewrite(g);
  repeat
    n := -1;
    Init;
    repeat
      read(f, x);
      inc(n); Dlzka[n] := x
    until eoln(f);
    if x > -1 then VytvorKody
  until x < 0;
  close(f); close(g);
end.
```

5.2.3 Hra

Určite si viete predstaviť, ako by sme simulovali hru, ak by sme si šachovnicu reprezentovali ako dvojrozmerné pole, pričom 0 znamená nezafarbené políčko a 1 zafarbené. Pri každom ťahu potom stačí nájsť požadovaný najväčší obdĺžnik a vyfarbiť ho jedničkami.

My si však nemôžeme dovoliť pole 1000×1000. Všimnime si však, že ak niektorý obdĺžnik má ľavý dolný roh (x_1, y_1) a pravý horný roh $(x_2 - 1, y_2 - 1)$, potom sa x_1 aj x_2 vyskytlo medzi x-ovými súradnicami v zadaní a y_1 aj y_2 medzi y-ovými súradnicami v zadaní. Preto nám stačí pamätať si iba šachovnicu s nerovnako veľkými políčkami, pričom hranicami políčok sú súradnice zo vstupu úlohy (z predchádzajúceho vyplýva, že ľubovoľný najväčší obdĺžnik zaberá vždy celé políčko takejto šachovnice). Takýchto políčok je však vždy najviac 101×101 a to si už dovoliť môžeme.

```
program hra;
type
  polesuradnic = array [0..120] of integer;
var
  px, py : polesuradnic;
    {px[i-1], py[j-1], px[i], py[j]
     su hranice obdlznika (i, j)}
  a : array [0..120, 0..120] of integer;
  body : array [1..120, 1..2] of integer;
  n : integer;
  rx, ry : integer;
  maxplocha : longint;
  pocet, i : integer;
  pocetvstupov : integer;
  f, g : text;

procedure zotried(var p : polesuradnic;
                  kolko : integer);
var
  i, j, k, pom : integer;
begin
  for i := 0 to kolko do begin
    k := i;
    for j := i + 1 to kolko do
      if p[j] < p[k] then
        k := j;
    pom := p[i]; p[i] := p[k]; p[k] := pom;
  end;
end;

procedure zhusti(var p : polesuradnic;
                 var kolko : integer);
var
  i, pom : integer;
begin
  pom := 0;
  for i := 1 to kolko do begin
    if p[i] <> p[i - 1] then begin
      pom := pom + 1;
      p[pom] := p[i];
    end;
  end;
  kolko := pom;
end;

procedure nacitaj_body;
var
  vx, vy, i, j : integer;
begin
  read(f, vx, vy, maxplocha, n);
  n := n * 2;
  px[0] := 0; py[0] := 0;
  {nacitaj body}
  for i := 1 to n do begin
    read(f, body[i, 1], body[i, 2]);
    px[i] := body[i, 1];
    py[i] := body[i, 2];
  end;
  px[n+1] := vx; py[n+1] := vy;
  {zotriedit jednotlive suradnice}
  zotried(px, n + 1);
  zotried(py, n + 1);
  {zhusti (vymaz rovnakych hodnot)}
  rx := n + 1; ry := n + 1;
  zhusti(px, rx);
  zhusti(py, ry);
  {vynulovanie pola a}
  for i := 1 to rx do for j := 1 to ry do
      a[i, j] := 0;
  {zarazky - okolo urobit jednicky}
  for i := 1 to rx do begin
    a[i, 0] := 1;
    a[i, ry + 1] := 1;
  end;
  for i := 1 to ry do begin
    a[0, i] := 1;
```

```
    a[rx + 1, i] := 1;
  end;
end;

function priradpolicko(var p : polesuradnic;
          suradnica : integer) : integer;
{priradi policko danej suradnici}
var
  i : integer;
begin
  i := 0;
  while suradnica >= p[i] do inc(i);
  priradpolicko := i;
end;

procedure najvacsiobdlznik(x, y : integer;
            var rozx, rozy : integer;
            var plocha : longint);
{vrati pocet policok v x-ovom
 a y-ovom smere najvacsieho
 obdlznika; ak existuje viac - vrati
   -1,  hocico;
 ak je policko uz vyfarbene - vrati
   -2,  hocico;}
var
  minx, i, j : integer;
  aktplocha : longint;
begin
  if a[x, y] > 0 then begin
    rozx := -2;
    exit;
  end;
  minx := 100; i := 0;
  plocha := -1;
  while a[x + i, y] = 0 do begin
    j := 0;
    while (a[x + i, y + j] = 0) and
          (j <= minx) do inc(j);
    dec(j);
    minx := j;
    {i, j udavaju rozmery najvacsieho
     obdlznika, ktory ma pravy okraj
     na konci stlpca x+i a lavy dolny roh
     mu zacina na policku x, y}
    aktplocha :=
      longint(px[x + i] - px[x - 1]) *
             (py[y + j] - py[y - 1]);
    {provname ho s doteraz najvacsim
     najdenym obdlznikom}
    if aktplocha > plocha then begin
      rozx := i; rozy := j;
      plocha := aktplocha;
    end else if aktplocha = plocha then
      rozx := -1;
    {zvacsit i a znovu}
    inc(i);
  end;
end;
```

```
procedure vyfarbiobdlznik(x, y,
            rozx, rozy : integer);
var
  i, j : integer;
begin
  for i := 0 to rozx do
    for j := 0 to rozy do
      a[x + i, y + j] := 1;
end;

procedure sledujhru;
var
  plochy : array [0..1] of longint;
  plocha : longint;
  x, y : integer;
  rozx, rozy : integer;
  i : integer;
begin
  plochy[0] := 0; plochy[1] := 0;
  for i := 1 to n do begin
    x := priradpolicko(px, body[i, 1]);
    y := priradpolicko(py, body[i, 2]);
    najvacsiobdlznik(x, y, rozx, rozy, plocha);
    if rozx = -2 then begin
      writeln(g, 'Vyfarbene policko');
      exit;
    end;
    if rozx = -1 then begin
      writeln(g, 'Viacero obdlznikov');
      exit;
    end;
    vyfarbiobdlznik(x, y, rozx, rozy);
    if plocha > maxplocha then begin
      writeln(g, 'Privelka plocha');
      exit;
    end;
    plochy[i mod 2] := plochy[i mod 2] + plocha;
  end;
  if plochy[0] > plochy[1] then
    writeln(g, 'Janko')
  else if plochy[1] > plochy[0] then
    writeln(g, 'Marienka')
  else
    writeln(g, 'Remiza');
end;

begin
  assign(f, 'HRA.IN'); reset(f);
  assign(g, 'HRA.OUT'); rewrite(g);
  read(f, pocetvstupov);
  for i := 1 to pocetvstupov do begin
    nacitaj_body;
    sledujhru;
  end;
  close(f); close(g);
end.
```

5.2.4 Škatule

Ak máme škatuľu s rozmermi $a \times b \times c$, jej povrch bude $P = 2(ab + bc + ac)$. Ak sú rozmery celočíselné, musí platiť, že povrch je párny. Pre nepárne čísla P teda neexistuje žiadna škatuľa s týmto povrchom. Nech je teda P párne. Označme ako a najmenší z rozmerov, c najväčší. Keďže $a \leq b, c$, platí, že $6a^2 \leq P$, t.j. $1 \leq a \leq \sqrt{P/6}$. Stačí nám teda skúšať všetky celočíselné a z tohoto intervalu.

Pre dané a potrebujeme ešte zistiť počet všetkých b a c celých ($b \leq c$), takých, že $ab + bc + ac = \frac{P}{2}$. Pričítajme k obom stranám tejto rovnice a^2 a dostávame $(a + b)(a + c) = \frac{P}{2} + a^2$. Označme hodnotu pravej

strany k, keďže P aj a už máme pevné, toto číslo vieme vypočítať. Nech $l = a + b$. Z rovnice vyplýva, že l je deliteľom k a keďže $a \leq b \leq c$, platí, že $2a \leq l \leq \sqrt{k}$. Naopak pre každé l, ktoré delí k a spĺňa nerovnosť $2a \leq l \leq \sqrt{k}$, môžeme určiť rozmery b a c takto: $b = l - a$ a $c = \frac{k}{l} - a$, pričom takto vypočítané b a c budú celé a bude platiť, že $a \leq b \leq c$. Stačí teda nájsť počet deliteľov čísla k z intervalu $[2a, \sqrt{k}]$. Časová zložitosť programu je $O(n)$, pamäťová $O(1)$.

```
program kvadre;
var
  f, g : text;
  p, p2 : longint; {povrch a jeho polovica}
  poc : longint;  {pocet moznosti}
  k, a, l : longint;
begin
  assign(f, 'skatule.in'); reset(f);
  assign(g, 'skatule.out'); rewrite(g);
  readln(f, p);
  while p > 0 do begin
    poc := 0;
    if p mod 2 = 0 then begin
      p2 := p div 2;
      a := 1;        {pre vsetky a}
      while 3 * a * a <= p2 do begin
        k := p2 + a * a;
        l := 2 * a; {pre vsetky l=a+b}
        while l * l <= k do begin
          if k mod l = 0 then inc(poc);
          inc(l);
        end;
        inc(a);
      end; {while a}
    end; {if}
    writeln(g, poc);
    readln(f, p);
  end;
  close(f); close(g);
end.
```

5.2.5 Záhon

Najľavejší z najvrchnejších krížikov v mape budeme volať L. Ak sa na políčku napravo od L nachádza tiež krížik, potom ak chceme pokryť záhrady tvarmi ŠIN a ŠON musíme na políčko L položiť ŠON. Ak sa napravo od L nenachádza krížik, potom musíme na políčko L položiť ŠIN. Teda na to, aby sme zistili, či sa záhrady dajú pokryť záhonmi v štýle štvorizmu stačí ísť po políčkach po riadkoch zhora dolu, v riadku zľava doprava a každý nájdený nepokrytý krížik pokryť tvarom, ktorý je jednoznačne určený. Ak sa nám niekedy nepodarí tento tvar položiť, potom záhrady nie je možné pokryť. Pri takomto postupe nepotrebujeme mať naraz načítanú v pamäti celú mapu, ale vždy najviac tri riadky.

```
program zahon;
var
  fin, fout : text;
  a : array[0..2, 1..10000] of char;
  i, j, v, s : integer;
  ok : boolean;

function priloz(i, j : integer) : boolean;
var
  k, l : integer;
begin
  if (j + 2 <= s) and
     (a[i, j+1] = '#') then begin
    {skusime prilozit SON}
    for k := 0 to 2 do
      for l := 0 to 2 do
        if a[(i+k) mod 3, j+l] = '#' then
          a[(i+k) mod 3, j+l] := '.'
        else begin
          priloz := false;
          exit;
        end;
    priloz := true;
  end else
    if (j - 1 >= 1) and
       (j + 1 <= s) then begin
      {skusime prilozit SIN}
      for k := 0 to 2 do
        for l := -k mod 2 to k mod 2 do
          if a[(i+k) mod 3, j+l] = '#' then
            a[(i+k) mod 3, j+l] := '.'
          else begin
            priloz := false;
            exit;
          end;
      priloz := true;
    end else
      priloz := false;
end;

begin
  assign(fin, 'zahon.in');
  reset(fin);
  assign(fout, 'zahon.out');
  rewrite(fout);
  readln(fin, v, s);
  while v > 0 do begin
    ok := true;
    for i := 1 to v do begin
      for j := 1 to s do
        read(fin, a[i mod 3, j]);
      readln(fin);
      if i > 2 then
        for j := 1 to s do
          if a[(i - 2) mod 3, j] = '#' then
            if ok then
              ok := priloz((i - 2) mod 3, j);
    end;
    for i := v - 1 to v do
      for j := 1 to s do
        if a[i mod 3, j] = '#' then
          ok := false;
    if ok then
      writeln(fout, 'ANO')
    else
      writeln(fout, 'NIE');
```

```
    readln(fin, v, s);
  end;
```

```
  close(fout); close(fin);
end.
```

5.2.6 Slová

Počet všetkých slov dĺžky k je 26^k. Tieto slová budeme chápať ako zápisy čísel v sústave so základom 26, pričom `a` je cifra s hodnotou 0 a `z` je cifra s hodnotou 25. Ak usporiadame všetky slová dĺžky k podľa abecedy, im zodpovedajúce čísla v sústave so základom 26 budú 0 až $26^k - 1$.

Majme teda dané slovo w dĺžky k, pričom chceme určiť jeho evidenčné číslo. Prevedieme najprv toto slovo z čísla v 26-kovej sústave na číslo v desiatkovej sústave. Toto číslo určuje počet slov dĺžky k, ktoré majú evidenčné číslo menšie ako w. Zostáva ešte pripočítať tie slová, ktoré sú kratšie ako slovo w. Týchto je $\sum_{i=1}^{k-1} 26^i$. Takto sme dostali počet všetkých slov s evidenčným číslom menším ako má slovo w, takže po pričítaní 1 dostávame evidenčné číslo w (V programe namiesto pripočítavania 1 počítame uvedenú sumu od $i = 0$ — $26^0 = 1$).

Naopak, ak prevádzame číslo na slovo, odčítame najprv všetky kratšie slová. Dĺžku však vopred nepoznáme, preto odčítavame čísla 26^i pre $i = 0, 1, \ldots$ až dovtedy, kým príslušná mocnina 26^k nebude väčšia ako číslo, od ktorej ju odčítavame. V tom prípade k je hľadaná dĺžka a číslo, ktoré sme odčítavaním dostali, stačí previesť do sústavy so základom 26.

Jedinou nepríjemnosťou je, že čísla, s ktorými pracujeme, sú veľmi veľké, a preto je potrebné uchovávať ich cifry v poli a naprogramovať procedúry na vykonávanie základných aritmetických operácií.

```
program slova;
const
  max = 30;
type
  cislo = array[1..max] of integer;
var
  f, g : text;
  mocniny : array[0..20] of cislo;
  vystup, vstup : string;

procedure nuluj(var x : cislo);
{vynuluje cislo x}
var
  i : integer;
begin
  for i := 1 to max do x[i] := 0;
end;

function vacsie(var x, y : cislo) : boolean;
{porovna x a y}
var
  i : integer;
begin
  i := max;
  while (i > 1) and (x[i] = y[i]) do
    dec(i);
  vacsie := x[i] > y[i]
end;

procedure odcitaj(x, y : cislo; var z : cislo);
{z := x-y}
var
  i, zv : integer;
begin
  zv := 0;
  for i := 1 to max do begin
    z[i] := x[i] - (y[i] + zv);
    zv := 0;
    if z[i] < 0 then begin
      zv := 1;
      z[i] := z[i] + 10;
    end
  end;
end;

procedure scitaj(x, y : cislo; var z : cislo);
{z := x+y}
var
  i, zv, j : integer;
begin
  zv := 0;
  for i := 1 to max do begin
    j := x[i] + y[i] + zv;
    zv := j div 10;
    z[i] := j mod 10;
  end;
end;

procedure nasob(x : cislo; y : integer;
                var z : cislo);
{z := x*y}
var
  i, zv, j : integer;
begin
  zv := 0;
  for i := 1 to max do begin
    j := y * x[i] + zv;
    zv := j div 10;
    z[i] := j mod 10;
  end;
end;

procedure vydel(x : cislo; y : integer;
                var z : cislo;
                var zv : integer);
{z := x div y,  zv := x mod y}
var
  i, j : integer;
begin
  zv := 0;
  for i := max downto 1 do begin
    j := 10 * zv + x[i];
    zv := j mod y;
    z[i] := j div y;
  end;
end;
```

```
procedure vytvor_mocniny;
{mocniny[i] := 26^i}
var
  i : integer;
begin
  nuluj(mocniny[0]);
  mocniny[0, 1] := 1;
  for i := 1 to 20 do begin
    nasob(mocniny[i - 1], 26, mocniny[i]);
  end;
end;

procedure vyrataj_cislo;
var
  x, y : cislo;
  dlzka, i, j : integer;
begin
  nuluj(x);
  {scitame kratsie slova}
  dlzka := length(vstup);
  for i := 0 to dlzka - 1 do
    scitaj(x, mocniny[i], x);
  {premena z 26 do 10 sustavy}
  for i := 1 to dlzka do begin
    j := ord(vstup[i]) - ord('a');
    nasob(mocniny[dlzka - i], j, y);
    scitaj(x, y, x);
  end;
  {cislo z pola do retazca}
  i := max;
  while x[i] = 0 do dec(i);
  vystup := '';
  while i > 0 do begin
    vystup := vystup +
              chr(x[i] + ord('0'));
    dec(i);
  end;
end;

procedure vyrataj_slovo;
var
  x : cislo;
  i, j, dlzka : integer;
  znak : char;
begin
  {zistenie dlzky slova}
  nuluj(x);
  dlzka := length(vstup);
  for i := 1 to dlzka do
    x[i] := ord(vstup[dlzka - i + 1]) -
            ord('0');
  dlzka := 0;
  while not vacsie(mocniny[dlzka], x) do begin
    odcitaj(x, mocniny[dlzka], x);
    inc(dlzka);
  end;
  {premena z 10 do 26 sustavy}
  vystup := '';
  for i := 1 to dlzka do begin
    vydel(x, 26, x, j);
    znak := chr(j + ord('a'));
    vystup := znak + vystup;
  end;
end;

begin
  assign(f, 'slova.in'); reset(f);
  assign(g, 'slova.out'); rewrite(g);
  vytvor_mocniny;
  readln(f, vstup);
  while vstup[1] <> '*' do begin
    if vstup[1] in ['a'..'z'] then
      vyrataj_cislo
    else
      vyrataj_slovo;
    writeln(g, vystup);
    readln(f, vstup);
  end;
  close(f); close(g);
end.
```

5.2.7 Ferdo

Konce rebríkov, štart a cieľ sú význačné body v mrakodrape. Ferdo totiž môže prebehnúť po mrakodrape práve vtedy, keď môže prebehnúť po grafe z týchto bodov. Takže si najprv vytvoríme taký graf. V poli G máme pre každý vrchol jeho troch susedov. Sused 0 je vľavo, sused 1 vpravo a sused 2 zvislo. Ak sused neexistuje, je v poli G hodnota -1. Vrcholy sa vytvárajú takto: načítame jeden riadok. Pre každý spodný koniec rebríka vytvoríme nový párny vrchol a pre každý horný koniec nepárny vrchol. Spojíme nové vrcholy zvislo. Zároveň máme všetky informácie, aby sme pospájali body na tomto podlaží (máme spodné konce rebríkov z práve načítaného riadku a prípadné horné konce z predchádzajúceho riadku). Takto postupujeme (pamätáme si naraz vždy dva riadky), kým nevytvoríme celý graf. Nakoniec pridáme štartovací a cieľový vrchol.

Teraz zistíme, ako sa bude šíriť oheň. To, čo potrebujeme vedieť je pre každý vrchol grafu, kedy sa k nemu oheň dostane. Preto pri načítavaní grafu nastavíme vzdialenosti na hranách na d/O, kde d je dĺžka a O je rýchlosť ohňa. V poli $Dist[i]$ nastavíme 0 ak vo vrchole i vznikol požiar na začiatku a -1 inak. Vo funkcii *KedyZhori* prehľadávame graf ako v Dijkstrovom algoritme (t.j. v cykle nájdeme vrchol s najmenším $Dist$ a obnovíme jeho susedov). Na konci máme pre každý vrchol, kedy sa k nemu dostane. Teraz si treba určiť, kedy sa môže Ferdo dostať ku každému vrcholu. Na to je funkcia *Prebehni*, ktorá pracuje rovnako ako *KedyZhori*, ale vzdialenosti sú d/F a pri prehľadávaní môžeme obnoviť suseda, iba ak tam Ferdo dobehne skôr ako oheň. Nakoniec máme v poli $Time$ príslušné časy a stačí pozrieť na cieľový vrchol, či tam Ferdo môže dobehnúť.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

FILE *f,*g;
int N,M,F,O;

int G[1000][3],V;
double W[1000][3],Dist[1000];
int Riadok[2][1000],Nr[2];

void NacitajRiadok(int i) {
  int j;
  char ohen;

  Riadok[i][0]=-1;
  fscanf(f,"%d ",Nr+i);
  for (j=0;j<Nr[i];j++) {
     fscanf(f,"%d%c",Riadok[i]+j,&ohen);
     if (ohen=='*')
       Riadok[i][j]=-Riadok[i][j]-1;
     else Riadok[i][j]++;
  }
}

void RobRiadok(int r,int &num) {
   int l,i,j,ri,rj,n,pos[2],ni,nj;

   n=num+2*Nr[1-r];
   // zvisle
   for (i=0;i<Nr[r];i++) {
      if (Riadok[r][i]<0) Dist[n+2*i]=0;
      else Dist[n+2*i]=-1;
      Dist[n+2*i+1]=-1;
      Riadok[r][i]=abs(Riadok[r][i])-1;
      G[n+2*i][2]=n+2*i+1;
      W[n+2*i][2]=1/(double)O;
      G[n+2*i+1][2]=n+2*i;
      W[n+2*i+1][2]=1/(double)O;
   }
   // vodorovne
   i=0; pos[0]=pos[1]=0;
   if ((Nr[1-r]>0)&&
       (Riadok[1-r][0]<Riadok[r][0])) {
      G[num+1][0]=-1; ri=1-r;
   } else {
      G[n][0]=-1;ri=r;
   }
   pos[ri]=1;
   for (l=0;l<Nr[0]+Nr[1]-1;l++) {
      if ((Nr[1-ri]>pos[1-ri])&&
          ((pos[ri]==Nr[ri])||
           (Riadok[1-ri][pos[1-ri]]<
              Riadok[ri][i+1]))) {
         j=pos[1-ri]++;
         rj=1-ri;
      } else {
         j=pos[ri]++;
         rj=ri;
      }
      if (ri==r) ni=n+2*i;
      else ni=num+2*i+1;
      if (rj==r) nj=n+2*j;
      else nj=num+2*j+1;
      G[ni][1]=nj;
      G[nj][0]=ni;
      W[ni][1]=W[nj][0]=
        abs(Riadok[ri][i]-
              Riadok[rj][j])/(double)O;
      i=j;
      ri=rj;
   }
   if (ri==r) ni=n+2*i; else ni=num+2*i+1;
   G[ni][1]=-1;
   num=n;
}

void KedyZhori() {
   int OK[1000],Nok;
   int i,j,mij;

   for (i=0;i<V;i++) OK[i]=0;
   for (Nok=0;Nok<V;Nok++) {
      mij=-1;
      for (j=0;j<V;j++)
         if (!OK[j]&&
             ((Dist[j]>=0)&&
              ((mij==-1)||
               (Dist[j]<
                 Dist[mij])))) mij=j;
      OK[mij]=1;
      for (j=0;j<3;j++)
         if  ((G[mij][j]!=-1)&&
              ((Dist[mij]+W[mij][j]<
                  Dist[G[mij][j]])||
               (Dist[G[mij][j]]==-1)))
            Dist[G[mij][j]]=Dist[mij]+W[mij][j];
   }
}

int Prebehni() {
   int OK[1000],Nok;
   double Time[1000];
   int i,j,mij;

   for (i=0;i<V;i++) {
      OK[i]=0;
      Time[i]=-1;
   }
   Time[V-2]=0;
   for (Nok=0;Nok<V;Nok++) {
      mij=-1;
      for (j=0;j<V;j++)
         if (!OK[j]&&
             ((Time[j]>=0)&&
              ((mij==-1)||
               (Time[j]<
                  Time[mij])))) mij=j;
      if (mij==-1) break;
      OK[mij]=1;
      for (j=0;j<3;j++)
 if ((G[mij][j]!=-1)&&
             (Time[mij]+W[mij][j]<
                Dist[G[mij][j]])&&
    ((Time[mij]+W[mij][j]<
                Time[G[mij][j]])||
              (Time[G[mij][j]]==-1)))
    Time[G[mij][j]]=Time[mij]+W[mij][j];
   }
   return Time[V-1]!=-1;
}

void main() {
  int PocetZadani,Zadanie;
  int i,r,j,Prv;

  f=fopen("ferdo.in","r");
  g=fopen("ferdo.out","w");
  fscanf(f,"%d ",&PocetZadani);
  for (Zadanie=1;
       Zadanie<=PocetZadani;
       Zadanie++) {
```

```
  fprintf(g,"Zadanie %d:",Zadanie);
  fscanf(f,"%d %d %d %d ",&N,&M,&F,&O);
  Nr[0]=0; r=1; V=0;
  for (i=0;i<N-1;i++) {
    NacitajRiadok(r);
    if (i==0) Prv=abs(Riadok[r][0])-1;
    RobRiadok(r,V);
    r=1-r;
  }
  Nr[r]=1;Riadok[r][0]=M+1;
  RobRiadok(r,V);
  G[V][2]=-1;
  V++;
  //zaciatok
  G[V][0]=G[V][2]=-1;
  G[V][1]=0;
  Dist[V]=-1;
  G[0][0]=V;
  W[V][0]=W[0][0]=Prv/(double)O;
  V++;
  KedyZhori();
  for (i=0;i<V;i++)
      for (j=0;j<3;j++)
        W[i][j]*=O/(double)F;
   if (Prebehni())
     fprintf(g,"prejde\n");
    else fprintf(g,"zhori\n");
 }
 fclose(f); fclose(g);
}
```

5.2.8 Čísla

Predstavme si, že všetky čísla napísané na papieri zväčšíme o jednotku, ale hru budeme hrať s pôvodnými nezväčšenými číslami. Teda ak sú na papieri napísané čísla $A+\underline{1}$ a $B+\underline{1}$ tak môžme podľa pravidla hry pripísať aj číslo $AB+A+B+\underline{1}=(A+\underline{1})(B+\underline{1})$. Vidíme, že z pohľadu zväčšených čísel je pravidlo jednoduchšie: ak sú na papieri napísané nejaké dve čísla, potom tam možno pripísať aj ich súčin. Teda na to, aby sme v pôvodnej hre zistili, či môžme dosiahnuť číslo C stačí zistiť, či je $C+1$ súčinom niektorých o jedna zväčšených začiatočných čísel. To ale nemusí byť také jednoduché (napr. dá sa 768 získať súčinom niektorých z 4,6,16,64,96?). Na to, aby sme zistili, či sa nejaké D dá napísať ako súčin niektorých z čísel $X_1,...,X_n$ si budeme postupne vytvárať usporiadaný zoznam deliteľov D, ktorých vieme získať ako súčin niektorých z čísel $X_1,...,X_j$ pre $j=1$ potom pre $j=2$ atď. až po $j=n$. Keď pridávame ďalšie číslo X_{j+1}, stačí pozerať prvky v zozname od najmenšieho a a pridávať delitele vzniknuté vynásobením X_{j+1}. Takýto zoznam bude mať najviac 1344 prvkov, lebo každé číslo menšie ako 10^9 má najviac toľkoto deliteľov (734134400 ich má presne 1344).

```
program cisla;
const
  max_del = 1344;
var
  fin, fout : text;
  {spajany zoznam dosiahnutych delitelov}
  a : array[1..max_del] of longint;
  d : array[1..max_del] of integer;
  n, i, m, p1, p2 : integer;
  c, x, y : longint;
  ano : boolean;

begin
  assign(fin, 'cisla.in');
  reset(fin);
  assign(fout, 'cisla.out');
  rewrite(fout);
  read(fin, n);
  for i := 1 to n do begin
    read(fin, c);
    c := c + 1;
    m := 1; a[1] := 1; d[1] := 0;
    ano := false;
    while not eoln(fin) do begin
      read(fin, x);
      x := x + 1;
      if not ano then begin
        p1 := 1;
        p2 := 1;
        while p2 > 0 do begin
          if (c div a[p2]) mod x = 0 then begin
            {nasli sme nejaky delitel}
            y := a[p2] * x;
            ano := ano or (y = c);
            while (d[p1] > 0) and
                  (a[d[p1]] < y) do
              p1 := d[p1];
            if (d[p1] = 0) or
               (a[d[p1]] > y) then begin
              {pridame ho do zoznamu - je novy}
              m := m + 1;
              a[m] := y;
              d[m] := d[p1];
              d[p1] := m;
            end else
              if p2 = 1 then break;
          end;
          p2 := d[p2];
        end;
      end;
    end;
    if ano then
      writeln(fout, 'ANO')
    else
      writeln(fout, 'NIE');
  end;
  close(fout); close(fin);
end.
```

Pravidlá

Pre autentickejší dojem uvedieme pravidlá v takej forme, ako ich dostali súťažiaci v 5. ročníku súťaže.

Pravidlá korešpondenčného kola v roku 1998

Do korešpondenčného kola sa prihlasujú dvojčlenné družstvá. Členovia družstva riešia úlohy korešpondenčného kola spolu! Príklady korešpondenčného kola pošlite spolu s vyplnenou prihláškou na adresu KVI MFF UK, Mlynská Dolina, 842 15 Bratislava najneskôr do **1. marca 1998**. Riešenie každého príkladu píšte na samostatný papier. Riešenie musí obsahovať výpis programu (môže byť aj rukou), vstupné údaje, na ktorých ste program skúšali, a výsledky. **Ku každému programu pripojte stručné vysvetlenie významu použitých premenných a hlavnú myšlienku Vášho algoritmu**. Dôraz kladťte na efektívnosť a zdôvodnenie Vášho algoritmu. Nezabudnite sa na každý papier čitateľne podpísať. Po vyhodnotení prvého kola Vám pošleme vaše riešenia spolu so vzorovými riešeniami a výsledkovou listinou.

Pravidlá súťaže Programujem s COFAXom '98

- Programátorské teamy pozostávajú z dvoch študentov tej istej strednej školy.
- Každá stredná škola môže vyslať len jeden reprezentačný team.
- Počas súťaže trvajúcej 4 hodiny tento team rieši sadu najmenej šiestich programátorských úloh na počítači 386 alebo 486 v prostredí Borland Pascal alebo Borland C v MS DOS.
- Každý team má počas celej súťaže k dispozícii jeden počítač, na ktorom môže úlohy riešiť.
- V prípade poruchy na tomto počítači, team dostane náhradný počítač, na ktorom pokračuje v riešení úloh. Stratený čas pri výmene počítačov sa nezohľadňuje vo výsledkoch.
- Každý team si sám zodpovedá za vlastné riešenia (programy a dáta) na svojom počítači. Má k dispozícii dve diskety, na ktorých si môže archivovať riešenia. V prípade vážnej poruchy na počítači (napr. harddisk), organizátori neakceptujú protesty vyplývajúce zo straty dát na pokazenom počítači.
- Nie je možné používať žiaden Vami prinesený programový produkt, žiadne Vaše diskety. Je zakázané ľubovoľným spôsobom komunikovať s členmi iného teamu (ústne, písomne, vymieňať si papiere, prípadne literatúru, diskety a pod.). Rovnako je zakázané hocakým spôsobom znemožniť prácu inému teamu (prerušenie napájania a pod.). Za porušenie tohoto pravidla sa team okamžite diskvalifikuje.
- Každý team je jednoznačne identifikovaný číslom, podľa ktorého sa poznajú aj jeho diskety a riešenia.
- Každá úloha má stanovené meno (max. 8 ascii znakov) a je označená písmenom. Súťažiaci pri riešení vytvárajú program "meno"`.pas`, resp. "meno"`.cpp`, ktorý číta vstupné dáta zo súboru "meno"`.in` a zapisuje výsledky do súboru "meno"`.out`. Písmeno, ktorým je úloha označená slúži len na lepšiu orientáciu pre súťažiacich.
- Každá úloha má presne špecifikovaný vstup a výstup. Vstupný aj výstupný súbor sú textové súbory, kde každý riadok končí neviditeľnými znakmi `#13#10` (súbor typu `text` pre pascalistov, resp. `"t"` pre c-čkarov). Za posledným viditeľným znakom v riadku nesmú byť žiadne medzery.
- Všetky úlohy majú taký charakter, že vo vstupnom súbore je viacero vstupných dávok, ktoré treba Vašim algoritmov spracovať a vypísať opäť viacero výstupných dávok do výstupného súboru. Jednotlivé dávky sú oddelené tak, aby sa dali rozlíšiť pri čítaní, rovnako tiež, aby bolo možné zistiť koniec vstupných dávok. Príklad: nech riešená úloha je na tému triedenie postupnosti čísiel. Nech vstupná neutriedená postupnosť je popísaná dĺžkou a jednotlivými prvkami (napr. 7 4 2 7 3 5 7 8). Vstupný súbor potom má tvar viacerých vstupných dávok do Vášho programu. Napríklad:

```
sort.in:
4 5 3 2 5
10 6 3 5 7 9 4 2 5 5 7
2 3 55
3 5 4 3
0
```

Presný formát vstupu a požadovaný formát výstupu budú vždy precízne popísané v každej úlohe. V našom ilustračnom príklade vidíme štyri vstupné dávky, ktoré treba utriediť a vypísať do výstupného súboru. Preto, jedna z možností, aký môže byť požadovaný formát výstupu, je:

```
sort.out:
```

```
2 3 5 5
2 3 4 5 5 5 6 7 7 9
3 55
3 4 5
```

Takže Váš program musí mať nasledujúci tvar (pre Pascal):

```
...
var f, g :text;
...
assign(f, 'sort.in'); reset(f);
assign(g, 'sort.out'); rewrite(g);
read(f, pocet)
while pocet > 0 do begin
  read(f, pocet);
  for i := 1 to pocet do read(f, ...); readln(f);
  .... triedenie
  for i := 1 to pocet do write(g, ...); writeln(g);
  read(f, pocet)
 end;
close(f); close(g);
```

Toto je len ilustračný program, aby ste mali predstavu, že okrem 'čistého' algoritmu budete musieť naprogramovať v každom riešení čítanie vstupného súboru a výpis do výstupného súboru v presne stanovenom formáte. Preto vám odporúčame si okrem čisto logického myslenia precvičiť aj tieto veci. Môže sa stať, že pre niektorých z Vás to bude náročnejšie ako naprogramovanie samotného algoritmu.

- Riešenie žiadnej úlohy nesmie obsahovať direktívy kompilátora, príkazy zapisujúce priamo do pamäti mimo dátových štruktúr (napríklad `move`, resp. `memcpy` a `memmove`) a nesmú používať funkcie z knižníc `crt` resp. `conio.h`.
- Pri opravovaní budú Vaše riešenia kompilované riadkovým kompilátorom `tpc` resp. `tcc` s parametrami `$B-`, `$I+`, `$R+`, `$X+`, `$S+` v PASCALe a `-N -ml` v C++.
- Počas celej súťaže majú súťažiaci právo písomnou formou zadávať porote otázky týkajúce sa zadania úloh. Otázky však musia byť takého tvaru, aby sa na ne dalo odpovedať `ÁNO`, `NIE`. V prípade, že otázka nebude vhodne formulovaná, alebo sa nebude týkať textu zadania (prípadne odpoveď bude napísaná v zadaní) odpoveď bude `BEZ KOMENTÁRA`. Ak sa počas súťaže vyskytne problém, ktorý by sa mali dozvedieť všetci súťažiaci, bude tento problém a jeho riešenie zverejnený písomnou alebo ústnou formou.
- Na odovzdávanie riešení slúži program `hotovo.exe`, ktorý bude na každom počítači. Keď súťažiaci budú chcieť dať nejaké riešenie otestovať, spustia tento program. Ten si od nich vyžiada základné informácie o tíme, úlohe a programovacom jazyku, riešenie zakóduje, pridá k nemu hlavičku obsahujúcu informácie pre testovací program a skopíruje ho na disketu pod menom `*.rie`. Ak chcú súťažiaci dať testovať viac úloh naraz, opätovne spustia `hotovo.exe`. Po vykonaní týchto úkonov ohlásia súťažiaci niektorému porotcovi, že odovzdávajú úlohou. Porotca odnesie riešenie(a) na diskete na testovanie. Po otestovaní sa disketa vráti súťažiacim, pričom na nej budú namiesto súborov `*.rie` súbory `*.pro`, ktoré budú obsahovať správy o výsledku testovania.
- Pre každú úlohu sa stanovuje čas, ktorý je súčtom trestného času a času od začiatku súťaže do odovzdania správneho riešenia. Za každé neúspešné testovanie sa trestný čas zvyšuje o 20 minút. Celkový čas tímu je súčet časov za správne vyriešené úlohy.
- Poradie tímu sa určuje na základe počtu správne vyriešených úloh. V prípadne rovnosti tohoto počtu rozhoduje celkový čas tímu: čím menší tým lepšie.
- Sporné záležitosti rieši odborná porota súťaže a jej rozhodnutie je definitívne.

Výsledky

Výsledkové listiny v tejto časti z priestorových dôvodov nie sú úplné. Význam bodov je uvedený v pravidlách a rovná sa celkovému času teamu.

1994

Poradie	Škola	Súťažiaci		Riešenia	Body
1.	G. Trebišov	Miroslav Dudík	Vojtech Varga	3	292
2.	G. Košice, Šrobárova	Tomáš Vinař	Luboš Pustý	3	328
3.	G. Košice, Trebišovská	Ján Karaffa	Ján Rusz	3	393
4.	G. Levice	Marek Nagy		3	403
5.	G. V.P-T Martin	Ondrej Gregor	Zdeno Ondák	2	113
6.	G. Žilina, Hlinská	Jozef Svrček	Martin Valent	2	129
7.	G. Bratislava, Hubeného	Peter Fodrek	Daniel Janec	2	166
8.	G. Košice, Poštová	Peter Helcmanovský	Rasťo Krivoš-Belluš	2	249
9.	G. P.Horova Michalovce	Martin Domány	Marek Ďurina	2	279
10.	G. Piešťany, SNP 9	Radovan Mikuš	Tomáš Daniška	2	285
11.	G. J. Hronca Bratislava	Patrik Sučanský	Bronislava Brejová	2	288

Por.	A'	A*	B'	B*	C'	C*	D'	D*	E'	E*	F'	F*	Príkl.	Čas
1.	15	0	216	0			61	0					3	292
2.	17	0	178	0		1		2	113	1			3	328
3.	93	0			117	1			143	1			3	393
4.	30	1	220	2			93	0					3	403
5.	38	0				1	75	0					2	113
6.	25	0		1	104	0		1					2	129
7.	102	0					64	0					2	166
8.	31	1						1	178	1			2	249
9.	49	1					170	2					2	279
10.	95	2		1			130	1					2	285
11.	19	0					209	3		1			2	288
12.	172	1					193	2				4	2	425
13.	168	3					193	1					2	441
14.	240	0			240	2				2		1	2	520
15.	238	5	201	3				2					2	599
16.	31	0											1	31
17.		4		1			54	0					1	54
18.	68	0						1					1	68
19.	72	0						2					1	72
20.	117	0						1					1	117
21.	123	1		1		1		2					1	143
22.		1	135	1									1	155

Por.	A'	A*	B'	B*	C'	C*	D'	D*	E'	E*	F'	F*	Príkl.	Čas
23.							162	1					1	182
24.	162	2											1	202
25.	238	4											1	318
		1											0	0
		3						2					0	0
		3											0	0
		1						3					0	0
								2					0	0
		8											0	0
		3											0	0
		3						6					0	0
		3				2		3		1			0	0
		2				2		2					0	0
		4						1					0	0
								3					0	0
								3					0	0
		4						3					0	0
		5						3					0	0
		2											0	0
		2				2		1					0	0
								1		2			0	0
		1											0	0
		1						7					0	0
				1						1			0	0

A' znamená čas, kedy bol príklad správne odovzdaný
A* znamená čas počet zlých riešení príkladu A

1995

Poradie	Škola	Súťažiaci		Riešenia	Body
1.	G. Trebišov	Miroslav Dudík	Vojtech Varga	4	487
2.	G. Sučany, Ang-Sl	Stanislav Funiak	Peter Vaníček	4	656
3.	G. Prievidza, VBN	Róbert Macho	Peter Leškovský	3	289
4.	G. Šamorín, maď.	Ladislav Szabó	Albert Diósi	3	654
5.	G. J. Hronca Bratislava	Dušan Lacko	Michaela Ačová	3	700
6.	G. Košice, Poštová	Peter Helcmanovský	Rasťo Krivoš - Beluš	2	88
7.	G. Košice, Šrobárova	Miroslav Paulík	Ľuboš Pustý	2	100
8.	G. Piešťany	Radovan Mikuš	Kamil Poturnaj	2	150
9.	G. Snina	Eva Mariničová	Ľubomír Marcin	2	232
10.	SPŠ elektrotech. Prešov	Martin Kobyľan	Miroslav Kuzmišin	2	260
11.	G. Myjava	Miroslav Zervan	Miroslav Škarítka	2	279
12.	G. Košice, Trebišovská	Ján Rusz	Ján Karaffa	2	287
13.	G. Bratislava, Grösslingova	Martin Pekár	Rastislav Krútky	2	347

Por.	A'	A*	B'	B*	C'	C*	D'	D*	E'	E*	F'	F*	G'	G*	Príkl.	Čas
1.	101	0	175	1	158	0	33	0							4	487
2.	203	1	39	0	180	1	154	2		1					4	656
3.	52	0	89	0		3	148	0		2					3	289
4.	220	1		3	239	2	135	0		2					3	654
5.	213	1		1		1	210	2	177	2					3	700
6.	47	0		2		2	41	0		3					2	88
7.	62	0				3	38	0		1					2	100
8.	113	0		1		1	37	0		1					2	150
9.	155	2		1			37	0		1					2	232
10.	105	0		3			155	0							2	260
11.	195	1				2	64	0		2					2	279
12.	184	2		2		0	63	0		1					2	287
13.	47	0				3	220	4							2	347
14.	58	0				5		4		2					1	58
15.	60	0						4		1					1	60
16.	88	0						2							1	88
17.			90	0				3							1	90
18.		3					217	1		3					1	237
19.	179	3						2		3					1	239
		2		1				2	239						1	239
21.		3						1		1					0	0
		1						1		2					0	0
								2		1					0	0
		1						3							0	0
						6		4							0	0
						1		1							0	0
		2													0	0
		1				4		2							0	0
						1		1		1					0	0
						3		2		2					0	0
				4		2		4							0	0
						2		2							0	0
				1		1									0	0

A' znamená čas, kedy bol príklad správne odovzdaný
A* znamená čas počet zlých riešení príkladu A

1996

Poradie	Škola	Súťažiaci		Riešenia	Body
1.	G. Trebišov	Dudík Miroslav	Varga Vojtech	7	789
2.	G. Košice, Poštova	Helcmanovský Peter	Krivoš-Belluš Rasťo	7	983
3.	G. Sereď	Války Ján	Krivošík Bohuslav	7	1257
4.	G. Bratislava, Grösslingova	Šipöcz Tomáš	Bezák Dušan	6	725

Poradie	Škola	Súťažiaci		Riešenia	Body
5.	G. Sučany	Vaníček Peter	Funiak Stanislav	6	839
6.	G. J. Hronca, Bratislava	Jandačka Martin	Kováč Radoslav	5	468
7.	G. Prievidza, VBN	Macho Róbert	Leškovský Peter	5	688
8.	G. Prešov, Konštantínova	Revický Marek	Mihaľák Matúš	4	601
9.	G. P.Horova Michalovce	Nistor Ľubomír	Vasil Peter	4	603
10.	G. Snina	Mariničová Eva	Marcin Ľubomír	4	703
11.	G. Košice, Trebišovská	Rusz Ján	Lacko Stanislav	3	191
12.	G. Lučenec	Gábor Miroslav	Kľuka Ján	3	305
13.	G. Matky Alexie BA	Svitok Ján	Wagner Juraj	3	365
14.	G. Šaľa	Kúkel Martin	Horal Stanislav	3	377
15.	G. Košice, Kováčska	Mojžiš Marcel	Mittelmann Michal	3	505
16.	G. Humenné	Habala Rastislav	Matula Jozef	3	599

Por.	A'	A*	B'	B*	C'	C*	D'	D*	E'	E*	F'	F*	G'	G*	H	H*	Príkl.	Čas
1.	99	0			78	0	198	0	63	0	179	0	34	0	138	0	7	789
2.	14	0	210	1	30	0	234	2			227	1	74	1	74	1	7	983
3.	64	0	160	2	145	0	177	0	199	0	239	2	153	2			7	1257
4.	11	0	238	2	32	0	89	0		2			169	0	126	1	6	725
5.			240	1	22	0	97	0	200	0	184	0	76	0			6	839
6.	81	0		1	44	0	176	0		3			30	0	137	0	5	468
7.	77	1	156	1	97	0	166	1				4	132	0			5	688
8.	46	1		1	143	2	212	1					120	0			4	601
9.		3			147	0	83	0				1	148	1	205	0	4	603
10.	189	1			52	0	184	0					198	3			4	703
11.	57	1		4	21	0		2					73	1			3	191
12.	49	0		2	159	0						2	77	1			3	305
13.	103	0			63	0							179	1			3	365
14.	16	0			57	0							224	4			3	377
15.	94	0			157	0		1					214	2			3	505
16.					40	0			200	2			239	4			3	599
17.	28	0				1							129	0			2	157
18.				2	50	0				3			108	0			2	158
19.	55	0			108	0								7			2	163
20.	33	0			167	1						3		3			2	220
21.	39	0			187	1								1			2	246
22.	98	0			209	0											2	307
23.		2					60	0						1			1	60
24.		2				1							110	1			1	130
25.		5			115	2								1			1	155
26.	177	0				3											1	177
27.					164	1											1	184

Por.	A'	A*	B'	B*	C'	C*	D'	D*	E'	E*	F'	F*	G'	G*	H	H*	Príkl.	Čas
28.		3															0	0
		7				1											0	0
		6				11											0	0
						7								4			0	0

A' znamená čas, kedy bol príklad správne odovzdaný
A* znamená čas počet zlých riešení príkladu A

1997

Poradie	Škola	Súťažiaci		Riešenia	Body
1.	G. Sučany	Peter Vaníček	Stanislav Funiak	5	559
2.	G. Poprad, Popr. nábr.	Martin Lang	Michal Forišek	5	975
3.	G. Humenné	Jozef Matula	Rastislav Habala	4	591
4.	G. Piešťany	Dušan Janiška	Roman kapusta	3	265
5.	SPŠ elektrotechnická, Košice	Ján Hudák	Ján Senko	3	279
6.	G. Prešov, Konštantínova	Slavomír Nemšák	Branislav Saxa	3	283
7.	G. B. Bystrica, Tajovského	Peter Rafaj	Ján Lunter	3	373
8.	G. Dubnica, sv. J. Bosca	Dušan Zervan	Kristián Berec	2	155
9.	G. Bratislava, L. Novomeského	Peter Trebatický	Matúš Horváth	2	174
10.	G. Nitra, Golianova	Peter Novák	Pavol Žibrita	2	180
11.	G. Sereď	Roman Varga	Andrej Salay	2	193
12.	G. Galanta, maďarské	Juraj Frivolt	Robert Vincze	2	255
13.	G. P. Bystrica	Martin Hajduch	Radoslav Balvan	2	262
14.	G. Žilina, V. Okružná	Dana Biernacká	Matej Lučenič	2	278
15.	G. N. Mesto nad Váhom	Marián Trúsik	Ján Nemšák	2	290
16.	G. Prievidza	Tomáš Ižo	Martin Kopál	2	392

Por.	A'	A*	B'	B*	C'	C*	D'	D*	E'	E*	F'	F*	G'	G*	H	H*	Príkl.	Čas
1.	11	0	75	1	97	0						6	233	0	123	0	5	559
2.	13	0	178	4	154	0			232	5					218	0	5	975
3.	39	0	92	1	141	0							239	3			4	591
4.	22	0	79	0		2							164	0	1		3	265
5.	12	0	34	0	173	3		3		4		4					3	279
6.	21	0	92	1	130	1											3	283
7.	64	0	134	2	115	1						5					3	373
8.	38	0		2	117	0						3					2	155
9.	67	0	107	0		3						1					2	174
10.	35	1		3	125	0								3			2	180
11.	46	1	127	0		1											2	193
12.	68	0		3	167	1											2	255
13.	67	1		11	135	2		11									2	262
14.	81	1		1	177	0											2	278
15.	75	2		9	155	1											2	290
16.	130	0		2	222	2											2	392

Por.	A'	A*	B'	B*	C'	C*	D'	D*	E'	E*	F'	F*	G'	G*	H	H*	Príkl.	Čas
17.	16	0		3		2		1									1	16
18.	18	0		4						4							1	18
19.	29	0		2		3				1							1	29
20.	38	0								6							1	38
21.	25	1		2		10				6							1	45
22.	32	1		3								4					1	52
23.	119	2															1	159
24.	142	1		4		2						2					1	162
25.	171	0										3					1	171
26.		2		2													0	0
				3				2									0	0
		7				2											0	0

A' znamená čas, kedy bol príklad správne odovzdaný
A* znamená čas počet zlých riešení príkladu A

1998

Poradie	Škola	Súťažiaci		Riešenia	Body
1.	G. Sereď	Ján Války	Roman Varga	3	451
2.	G. Poprad, Popr. nábr.	Martin Lang	Michal Forišek	3	494
3.	G. Pezinok	Ľubomír Čech	Filip Číž	2	157
4.	G. Žilina, Hlinská	Michal Matoušek	Róbert Pojtek	2	175
5.	G. Nitra, Parovská	Ronald Filo	Martin Štefček	2	245
6.	G. Trnava, J.Hollého	Augustín Osuský	Dušan Suchoň	2	290
7.	G. Žilina, V. Okružná	Peter Varša	Dana Biernacká	2	311
8.	G. Prievidza, VBN	Vladimír Wiedermann	Zuzana Dzuráková	2	337
9.	G. J. Hronca, BA	Vladinír Koutný	Richard Kráľovič	2	434
10.	G. Piešťany, SNP 9	Weissensteiner Július	Čúzy Ján	1	35