Dvadsiaty piaty ročník
Olympiády v informatike

SLOVENSKÁ INFORMATICKÁ SPOLOČNOSŤ

Odborným garantom súťaže Olympiáda v informatike je Slovenská informatická spoločnosť. Viac sa o nej dozviete na stránke `http://www.informatika.sk/`

RNDr. Michal Forišek PhD., Slovenská komisia OI
Dvadsiaty piaty ročník Olympiády v informatike
ISBN 978-80-8072-112-1

Obsah

O priebehu 25. ročníka Olympiády v informatike

V školskom roku 2009/10 prebehol jubilejný dvadsiaty piaty ročník Olympiády v informatike (OI). Podobne ako v minulom ročníku, aj v tomto prebehla súťaž v dvoch kategóriách: A a B.

Do kategórie B, určenej pre prvákov a druhákov klasických štvorročných stredných škôl, sa zapojilo 36 žiakov. Z nich bolo 29 pozvaných na krajské kolá, ktorými súťaž skončila.

Do ťažšej kategórie A, určenej pre všetkých stredoškolákov bez obmedzenia, sa zapojilo 108 žiakov. Najlepších 69 postúpilo do krajského, a z nich najlepších 30 do celoštátneho kola OI. Celoštátne kolo sa v tomto školskom roku konalo v Novom Meste nad Váhom (teoretická časť) a v Trenčíne (praktická časť).

Najlepší riešitelia kategórie A boli následne pozvaní na týždňové výberové sústredenie, kde sa určila reprezentácia Slovenska na medzinárodných súťažiach – Stredoeurópskej olympiáde v informatike (CEOI) a Medzinárodnej olympiáde v informatike (IOI).

Na celoslovenskej úrovni sa počas celého ročníka o plynulý chod OI starala Slovenská komisia OI v nasledujúcom zložení:

- doc. RNDr. Gabriela Andrejková, CSc.,
  predsedkyňa, ÚI PF UPJŠ, Košice
- RNDr. Michal Forišek, PhD., podpredseda, KI FMFI UK, Bratislava
- RNDr. Andrej Blaho, FMFI UK, Bratislava
- Mgr. Ján Katrenič, PF UPJŠ, Košice
- Mgr. Juliana Šišková, FMFI UK, Bratislava
- PaedDr. Miloslava Sudolská, PhD.,
  KI FPV UMB, krajská predsedkyňa pre BB
- RNDr. Eva Hanulová, Gym. J. Hronca, krajská predsedkyňa pre BA
- RNDr. Rastislav Krivoš-Belluš,
  Ústav informatiky PF UPJŠ, krajský predseda pre KE
- prof. Ing. Veronika Stoffová, CSc.,
  KI PF UJS, krajská predsedkyňa pre NR
- Mgr. Mária Majherová, PhD.,
  Katedra matematiky FHPV PU, krajská predsedkyňa pre PO

- Ing. Andrea Julény, KI FMech TnUAD, krajská predsedkyňa pre TN
- Mgr. Blanka Thomková, Gym. Jána Hollého, krajská predsedkyňa pre TT
- RNDr. Peter Varša, PhD., KI FRI ŽU, krajský predseda pre ZA

Zástupcom IUVENTY zabezpečujúcim organizačnú stránku súťaže bol už tradične Ing. Tomáš Lučenič.

V roku 2010 pripadla na Slovensko už po tretí krát povinnosť zorganizovať Stredoeurópsku olympiádu v informatike (CEOI). Tá sa uskutočnila v dňoch 12. až 19. júla v Košiciach. Viac sa o jej priebehu dočítate na posledných stranách tejto ročenky.

V školskom roku 2009/10 oslavovala Olympiáda v informatike svoje okrúhle výročie: od jej vzniku už prešlo štvrť storočia. Za týchto 25 rokov prešla súťaž dlhou cestou. Z pôvodnej jednej kategórie Matematickej olympiády je dnes samostatná súťaž. Z pôvodne čisto teoretickej súťaže je dnes súťaž, ktorá síce stále kladie hlavný dôraz na návrh efektívnych algoritmov, avšak zahŕňa aj úlohy, kde je po vymyslení algoritmu potrebná aj jeho bezchybná implementácia. Obtiažnosť úloh v súťaži odzrkadľuje vývoj teoretickej informatiky vo svete – tie spred vyše dvadsiatich rokov sú často len rozcvičkou pre dnešných riešiteľov. Ak nás v nasledujúcich dvadsiatich piatich rokoch čaká toľko isto zmien, máme sa na čo tešiť.

Michal Forišek, podpredseda SK OI

Zadania domáceho kola kategórie A

A-I-1 Maliar Bonifác

Mestský úrad v Kocúrkove vypísal nedávno výberové konanie na veľmi zodpovednú a dôležitú úlohu: maľovanie chodníka pred mestským úradom. Výberové konanie vyhral maliar Bonifác (jediný uchádzač (a starostov brat)).

Ako to už chodí, len čo Bonifác podpísal zmluvu, začali mu z mestského úradu prichádzať príkazy: Tento kus chodníka zelenou, tento ružovou, potom to skoro celé pretrieť na bielo... Veru tak. Netrvalo dlho a Bonifác si všimol, že niektoré príkazy sa prekrývajú. A keď si spomenul, že ho zmluva zaväzuje všetky príkazy, v tom poradí v akom ich dostal, vykonať, začali ho obchádzať mdloby.

No zrazu prišiel na myšlienku vskutku geniálnu. Keby vedel, ako má chodník vyzerať na konci, po vykonaní všetkých príkazov, mohol by ho tak vymaľovať rovno a potom sa tváriť, že on predsa všetky príkazy dodržal. A hlavne potom mestskému úradu všetko vyúčtuje podľa pôvodných príkazov a pekne sa na tom nabalí.

Súťažná úloha:

Chodník pred mestským úradom má K kocúrkovských krokov. Ten jeho koniec, ktorý je bližšie ku potravinám, bude mať súradnicu 0, opačný koniec bude mať súradnicu K. V súčasnosti má celý chodník asfaltovo modrú farbu. Bonifác má F iných farieb, očíslovaných od 1 po F. Postupne mu prišlo N príkazov, každý z nich je tvaru „a_i b_i f_i“, kde a_i a b_i sú súradnice začiatku a konca úseku a f_i je farba, ktorou ho má ofarbiť. Na ofarbenie metra chodníka potrebuje Bonifác liter farby. Pre každú z farieb spočítajte, koľko litrov jej Bonifác bude potrebovať.

Formát vstupu:

V prvom riadku vstupu sú tri medzerami oddelené celé čísla N, F a K ($1 \leq N, F \leq 100\,000$, $1 \leq K \leq 1\,000\,000\,000$). Nasleduje N riadkov, z ktorých každý popisuje jeden príkaz, a to v poradí, v akom ich Bonifác dostal. Presnejšie, i-ty z týchto riadkov obsahuje tri medzerami oddelené celé čísla a_i, b_i a f_i ($0 \leq a_i < b_i \leq K$, $1 \leq f_i \leq F$).

Pre 8 z 10 testovacích vstupov bude navyše platiť $K \leq 1\,000\,000$. Pre 6 z týchto 8 bude navyše aj $N \leq 10\,000$. Pre 3 z týchto 6 bude $N, K \leq 1\,000$.

Formát výstupu:

Pre každú z F farieb vypíšte jeden riadok a v ňom jedno celé číslo – počet litrov tejto farby, ktoré budeme potrebovať.

Príklad:

Vstup	Výstup
4 5 7 1 5 1 2 4 3 4 6 4 3 6 2	1 3 1 0 0

A-I-2 Čokoláda

Marienka bude mať čoskoro narodeniny. Jej braček Janko dlho nevedel, čo jej darovať – až kým vo svojej tajnej skrýši na povale nenarazil na zvyšky čokolády, ktorú si tam kedysi ukryl. Pravda, myši už boli vybrať svoju daň, ale aj tak jej ešte zostalo dosť. Deravé časti oláme, aby mu ostala pekná štvorcová tabuľka, a tú pekne zabalí. A zvyšok samozrejme zje.

Súťažná úloha:

Daný je pôvodný počet riadkov R a stĺpcov S, ktoré čokoláda kedysi mala, a matica $R \times S$ núl a jednotiek udávajúca, ktoré políčka z nej zostali celé. Zistite, koľkými spôsobmi môže Janko uskutočniť svoj plán. Inými slovami, spočítajte, koľkými spôsobmi sa dá na zvyšku čokolády vyznačiť štvorec bez dier. Všetky hrany štvorca musia samozrejme ležať na hranách políčok. Rovnako veľké štvorce na rôznych súradniciach považujeme za rôzne.

Formát vstupu:

V prvom riadku vstupu sú dve medzerami oddelené celé čísla R a S ($1 \leq R, S \leq 2\,500$). Nasleduje R riadkov, v r-tom z nich je S medzerami oddelených celých čísel $a_{r,1}, \ldots, a_{r,s}$. Ak je políčko (r, s) celé, je $a_{r,s} = 1$, inak $a_{r,s} = 0$.

Pre 7 z 10 testovacích vstupov bude platiť $R \leq 500$ a $S \leq 2\,500$. Pre 5 z týchto 7 bude $R, S \leq 500$, a pre 3 z týchto 5 bude $R, S \leq 20$.

Formát výstupu:

Vypíšte jeden riadok a v ňom jedno celé číslo – hľadaný počet štvorcov.

Príklad:

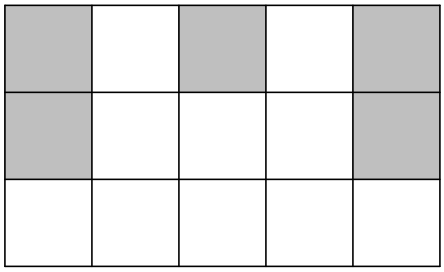
Vstup

```
3 5
0 1 0 1 0
0 1 1 1 0
1 1 1 1 1
```

Na obrázku vpravo je čokoláda popísaná vstupom. Sivou farbou sú políčka, ktoré chýbajú. Štvorec 1×1 *sa na nej dá vyznačiť desiatimi spôsobmi a štvorec* 2×2 *dvoma, to je dokopy* $10 + 2 = 12$ *spôsobov.*

výstup

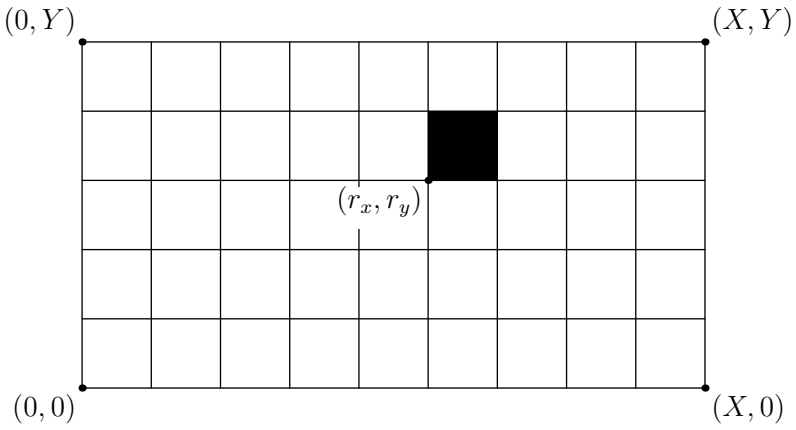
```
12
```



A-I-3 Koláč

Zlá ježibaba drží v klietke Janka a Marienku a snaží sa ich vykŕmiť. Práve pre nich upiekla za plech ježibabieho koláča. Koláč má tvar obdĺžnika. Celý je odpudivý a Jankovi s Marienkou sa doň príliš nechce. A keby to nestačilo, namiesto príslovečnej čerešničky je koláč ozdobený ohavnou pečenou ropuchou.

Keďže čokoľvek je lepšie ako musieť zjesť túto ropuchu, rozhodli sa Janko s Marienkou, že si z jedenia koláča spravia hru. Marienka na ňom lyžičkou naznačila čiary, čím ho rozdelila na $X \times Y$ rovnakých štvorcov. Celá ropucha sedí na jednom z týchto štvorcov.



Teraz budú striedavo ťahať. Hráč, ktorý je na ťahu, si vyberie niektorú z vyznačených čiar, a pozdĺž nej koláč rozreže na dve obdĺžnikové časti. Následne

tú časť, kde nie je ropucha, zje. Samozrejme, ten, kto príde na ťah v okamihu, keď už zostal len štvorec s ropuchou, prehral a musí ju zjesť. Ako prvá ťahá Marienka.

Súťažná úloha:

Dané sú rozmery koláča – kladné celé čísla X, Y. Tiež sú dané súradnice r_x, r_y ($0 \leq r_x < X$, $0 \leq r_y < Y$) ľavého dolného rohu štvorca, v ktorom je ropucha.

a) (2 body) Rozhodnite, kto vyhrá hru pre situáciu, ktorá je na obrázku – teda pre $(X, Y) = (9, 5)$ a $(r_x, r_y) = (5, 3)$ – a popíšte jednu možnú stratégiu, ktorá mu zabezpečí výhru.

b) (8 bodov) Napíšte čo najefektívnejší program, ktorý pre dané hodnoty X, Y, r_x a r_y zistí, ktoré z detí hru vyhrá, ak budú obe hrať optimálne.

Príklady:

Vstup

```
8 1 1 0
```

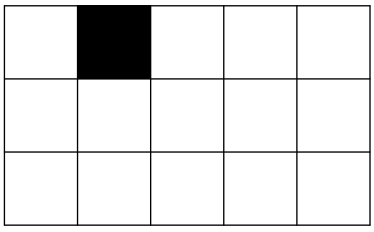
výstup

```
Vyhra Marienka.
```

V prvom ťahu spraví rez po priamke $x = 3$. Zostane ropucha a okolo nej z každej strany jeden štvorec. Janko zje jeden z nich, Marienka druhý, a Jankovi zostane ropucha.

Vstup

```
5 3 1 2
```



výstup

```
Vyhra Janko.
```

A-I-4 Počítač s gumenou rúrou

V tomto ročníku olympiády sa budeme v teoretickej úlohe stretávať so špeciálnym počítačom zvaným Kvak. V študijnom texte uvedenom za zadaním tejto úlohy je popísané, ako tento počítač funguje.

Súťažná úloha:

a) (3 body)
V rúre je jedno číslo. Napíšte program pre Kvak, ktorý vypíše 1, ak je to prvočíslo, a 0 inak.
Plný počet bodov dostanete za ľubovoľné riešenie, ktoré bude mať menej ako 100 príkazov a na ľubovoľnom vstupe spraví menej ako 10 000 krokov.

b) (4 body)
V rúre je postupnosť *kladných* čísel. Dĺžka postupnosti je menšia ako 65 000. Napíšte program pre Kvak, ktorý túto dĺžku spočíta a vypíše. Plný počet bodov dostanete za riešenie, ktoré bude mať časovú zložitosť lineárnu od dĺžky vstupnej postupnosti.

c) (3 body)
Máme pokazenú verziu počítača Kvak, ktorá sa od obyčajnej líši tým, že inštrukciu `put` už zvládne vykonať len desaťkrát a potom sa definitívne zasekne. Ľubovoľné iné inštrukcie zvláda vykonávať bez problémov.

V rúre je neprázdna postupnosť čísel, ktorá môže byť ľubovoľne dlhá. Dá sa napísať program pre pokazený Kvak, ktorý nájde a vypíše jej maximum? Ak áno, napíšte ho, ak nie, dokážte, že to nejde.

Interpreter:

Aby ste si mohli otestovať svoje riešenia, na webstránke olympiády máte k dispozícii interpreter programov pre Kvak. Nájdete ho na adrese `http://oi.sk/archiv/2009/kvak/`. Môžete ho použiť buď online, alebo si stiahnuť jeho zdrojový kód (v C++) a doma si ho skompilovať.

Tabuľka inštrukcii

príkaz	účinok príkazu
`get X`	Kvak vyberie číslo z rúry a uloží ho do registra `X`.
`put X`	Kvak vloží do rúry číslo z registra `X`.
`put` číslo	Kvak vloží dané číslo do rúry.
`print`	Kvak vyberie číslo z rúry a vypíše ho na výstup.
`add`	sčítanie: Kvak vyberie dve čísla z rúry a vloží tam ich súčet.
`sub`	odčítanie: Kvak vyberie dve čísla z rúry a vloží tam ich rozdiel (prvé mínus druhé).
`mul`	násobenie: Kvak vyberie dve čísla z rúry a vloží tam ich súčin.
`div`	delenie: Kvak vyberie dve čísla z rúry a vloží tam celú časť ich podielu (prvé lomeno druhé).
`mod`	zvyšok: Kvak vyberie dve čísla z rúry a vloží tam zvyšok, ktorý dá prvé z nich po delení druhým.
`label L`	návestie: Toto miesto v programe dostane meno `L` (kde `L` môže byť ľubovoľný reťazec).
`jump L`	skok: Kvak bude pokračovať vo vykonávaní programu od miesta, ktoré sa volá `L`.
`jz X L`	skok ak nula: Ak je v registri `X` nula, Kvak vykoná príkaz `jump L`.
`jeq X Y L`	skok ak sa rovnajú: Ak je v registroch `X` a `Y` to isté, Kvak vykoná príkaz `jump L`.
`jgt X Y L`	skok ak je väčšie: Ak je v registri `X` väčšia hodnota ako v `Y`, Kvak vykoná príkaz `jump L`.
`jempty L`	skok ak je rúra prázdna: Ak v rúre nie sú žiadne čísla, Kvak vykoná príkaz `jump L`.
`stop`	koniec: Kvak prestane vykonávať program.

Študijný text

Vedci z Kolégia Skúmania Potrubí (KSP) nedávno vyvinuli nový počítač, zvaný Kvak. Kvak má jedinú dátovú štruktúru: jednosmernú gumenú rúru.

Jediný dátový typ, ktorý Kvak pozná, sa volá `number`, a je to celé číslo z rozsahu od 0 po 65 535, vrátane.[1] Všetky matematické operácie počíta Kvak modulo 65 536. Teda napríklad hodnota výrazu $65530 + 10$ je 4.

Kvak má 26 premenných, ktoré voláme registre. Tie sú označené písmenami `a` až `z` a v každom z nich môže byť uložená jedna hodnota typu `number`. Na začiatku výpočtu sú vo všetkých registroch nuly.

Okrem registrov má Kvak, ako sme už spomínali, jednu jednosmernú gumenú rúru. S tou vie robiť dve operácie: vložiť do nej číslo z registra `X` príkazom `put X` a z opačného konca rúry vybrať číslo a uložiť ho do registra `X` príkazom `get X`.

Čísla sa v rúre samozrejme nemôžu predbiehať, Kvak ich teda bude vyberať v tom istom poradí, v akom ich tam vložil. Rúra je gumená, takže sa do nej čísel zmestí ľubovoľne veľa. Ak nie je povedané ináč, je na začiatku výpočtu rúra prázdna. Okrem rúry má Kvak ešte jeden kotúč žltej pásky, na ktorú môže písať svoj výstup.

Ak sa stane, že pri pokuse vybrať z rúry je rúra prázdna, nastane chyba. Rovnako nastane chyba, ak sa pokúsime deliť nulou, počítať zvyšok po delení nulou, alebo skočiť na neexistujúce miesto. Ak sa program dostane na koniec, Kvak korektne skončí (ako keby na konci programu bola ešte inštrukcia `stop`.)

Pre stručnosť môžeme písať viac príkazov do jedného riadku, v takomto prípade ich od seba treba oddeliť bodkočiarkou.

Príklad 1:

Nasledujúci program spočíta a vypíše súčet čísel od 1 do 20.

```
put 20 ; put 0
label start
  get n ; jz n end
  put n ; put n ; put 1 ; add ; sub
  get x ; put x
jump start
label end
print
```

[1] $65\,535 = 2^{16} - 1$, typ `number` je teda 16-bitové celé číslo bez znamienka (to isté ako `word` v Pascale, `unsigned short` v C/C++).

Vždy, keď sa Kvak pri vykonávaní programu dostane k riadku `label start`, budú v rúre práve dve čísla. Keď prvé z nich označíme n, hodnota druhého bude súčet $s = (n+1) + \cdots + 20$. Následne načítame n do registra `n`. Ak je $n = 0$, máme v rúre hľadaný súčet, môžeme ho vypísať a skončiť. V opačnom prípade chceme spraviť dve veci: Prirátať toto n k doteraz získanému súčtu, a následne zmenšiť n o jedna. Po vykonaní príkazov `put n ; put n ; put 1` máme v rúre postupne čísla: s, n, n, 1. Príkaz `add` sčíta prvé dve, po jeho vykonaní je v rúre: n, 1, $n + s$. Po vykonaní ďalšieho príkazu `sub` máme v rúre hodnoty $n + s$ a $n - 1$. To už je skoro to, čo treba, len sú v opačnom poradí. Preto jednu z nich načítame do `x` a znovu vložíme.

Príklad 2:

V rúre je neprázdna postupnosť čísel. Napíšeme program, ktorý spočíta a vypíše jej súčet. (Presnejšie, jeho zvyšok po delení 65 536.)

Dokola budeme opakovať nasledujúci postup: Zistíme, či sú v rúre aspoň dve čísla. Ak áno, tie dve, ktoré sú práve na začiatku, sčítame. Ak už zostalo len jedno, je zjavne súčtom všetkých pôvodných čísel. V programe pre Kvak môžeme túto myšlienku implementovať napríklad nasledovne:

```
label cyklus ; get a ; jempty koniec ; put a ; add ; jump cyklus
label koniec ; put a ; print
```

Na začiatku každej iterácie načítame do registra `a` číslo z rúry. Ak je tá v tomto okamihu prázdna, máme v registri `a` hľadaný súčet, stačí ho už len vypísať. Ak nie, číslo z registra `a` vrátime do rúry. V tomto okamihu sú v rúre aspoň dve čísla, a teda môžeme bez obáv zavolať inštrukciu `add`.

Časová zložitosť tohto riešenia je lineárna od počtu čísel, ktoré boli na začiatku v rúre. Totiž každá iterácia cyklu trvá len konštantne veľa krokov a zmenší nám o jedno počet čísel v rúre.

Zadania domáceho kola kategórie B

B-I-1 Majstrovstvá na rybníku

Ako sa blížila zima, bol na rybníku Čvachtáku čoraz pevnejší ľad. A keď už bol pevný natoľko, že sa ani medveď Mišo nebál po ňom prejsť, zišli sa na brehu zvieratká a začali hútať, ako by sa len zabavili.

„Už viem!“ vykríkla líška Eliška. „Spravíme majstrovstvá v krasokorčuľovaní!“

„Ale ja sa neviem korčuľovať...“ namietol had Félix.

„To nevadí,“ vyriešila to Eliška lišiacky, „ty budeš rozhodca!“

Súťažná úloha:

Zvieratká sa rozdelili na S súťažiacich a R rozhodcov. Súťažiaci predviedli svoje jazdy a každý z nich dostal od každého rozhodcu číselnú známku. Výsledné body súťažiaceho zistíme tak, že škrtneme jednu najvyššiu a jednu najnižšiu známku, ktorú dostal, a zvyšných $R - 2$ známok sčítame. Napíšte program, ktorý načíta všetky známky, vypočíta výsledné body všetkých súťažiacich a vypíše výsledkovú listinu súťaže.

Formát vstupu:

V prvom riadku vstupu sú dve medzerou oddelené celé čísla S a R – počet súťažiacich a počet rozhodcov. Môžete predpokladať, že $3 \leq S, R$ a že $S \cdot R \leq 1\,000\,000$.

Nasleduje $2S$ riadkov, vždy dva riadky popisujú jedného súťažiaceho. V prvom z nich je meno súťažiaceho. Meno je reťazec tvorený medzi 1 a 10 malými písmenami anglickej abecedy. V druhom riadku je vždy zoznam R medzerami oddelených celých čísel, predstavujúcich známky, ktoré tento súťažiaci dostal od rozhodcov. Známky sú z rozsahu od 0 po 6 000, vrátane.

V niektorých 6 testovacích vstupoch bude platiť $S \leq 1\,000$. V niektorých 4 testovacích vstupoch budú mať všetci súťažiaci navzájom rôzne výsledné body.

Formát výstupu:

Váš program má vypísať S riadkov – jeden pre každého súťažiaceho. Každý riadok má mať podobu „`B: meno`“, kde `meno` je meno súťažiaceho a `B` sú body, ktoré získal. Riadky usporiadajte podľa počtu bodov zostupne.
Súťažiacich s rovnakým skóre vypíšte usporiadaných podľa abecedy vzostupne (napr. `baran` < `had` < `hadica`).

Príklad:

Vstup

```
4 5
zajko
8 7 6 5 4
jezko
4 4 4 4 4
zabka
7 13 7 4 4
medvedik
23 19 47 11 48
```

Výstup

```
89: medvedik
18: zabka
18: zajko
12: jezko
```

Všimnite si, že `zabka` *a* `zajko` *majú rovnaké skóre, preto sú utriedení podľa abecedy.*

B-I-2 Cesta

Firma TravelEarth plánuje v novom roku do svojich GPS navigačných zariadení pridať novú funkciu. Tá by mala pomôcť vodičom presnejšie určiť dĺžku jazdy autom pozdĺž zvolenej trasy. Takýto čas dnes TravelEarth odhaduje iba na základe vzdialeností. Lenže v skutočnosti sa popri ceste nachádzajú dopravné značky, ktoré na rôznych úsekoch povoľujú rôznu maximálnu rýchlosť.

Súťažná úloha:

V našej úlohe budeme uvažovať len jednu konkrétnu cestu. Keď na nej vyznačíme miesta, kde sa mení maximálna povolená rýchlosť, rozdelíme ju tak na N úsekov. Tieto úseky si očíslujeme od 1 do N. Dĺžku úseku i označíme d_i a maximálnu povolenú rýchlosť na ňom označíme v_i.

Vašou úlohou je napísať program, ktorý načíta a spracuje popis cesty a následne bude odpovedať na otázky tvaru: „Ako dlho nám bude trvať presun, ak začíname na x-tom a končíme na y-tom kilometri cesty?“ (Samozrejme za predpokladu, že v každom úseku ideme presne jeho maximálnou povolenou rýchlosťou.)

Formát vstupu:

Prvý riadok vstupu obsahuje jedno celé číslo N ($1 \leq N \leq 200\,000$) – počet úsekov na našej ceste. Druhý riadok obsahuje N medzerami oddelených celých čísel $d_1, \ldots, d_N$ ($1 \leq d_i \leq 10\,000$) – dĺžky jednotlivých úsekov v kilometroch. Tretí riadok obsahuje N medzerami oddelených celých čísel $v_1, \ldots, v_N$ ($1 \leq v_i \leq 500$) – maximálne povolené rýchlosti v kilometroch za hodinu.

Vo štvrtom riadku nasleduje celé číslo Q ($1 \leq Q \leq 100\,000$) – počet otázok. Každý z nasledujúcich Q riadkov obsahuje dve medzerou oddelené celé čísla x a y predstavujúce jednu otázku ($0 \leq x < y \leq D$, kde D je súčet všetkých d_i, t. j. celková dĺžka cesty).

V 7 z 10 testovacích vstupov bude $D \leq 16\,000\,000$. V 5 z týchto vstupov bude tiež $N \leq 10\,000$ a $Q \leq 20\,000$.

Formát výstupu:

Pre každú otázku vypíšte jeden riadok – najkratší čas v hodinách, za ktorý sa pri dodržaní predpisov vieme dostať z x-tého kilometra cesty na y-ty.

Na ukladanie reálnych čísel odporúčame použiť dátový typ `double`. Na výpis výsledku môžete použiť príkaz `writeln(vysledok:0:7);` ak programujete v Pascale, resp. `printf("%.7f\n",vysledok);` v C/C++. Výsledok vypíšte na aspoň 7 desatinných miest. Riešenia, ktoré sa od správneho budú líšiť len drobnou zaokrúhľovacou chybou, uznáme za správne.

Príklad:

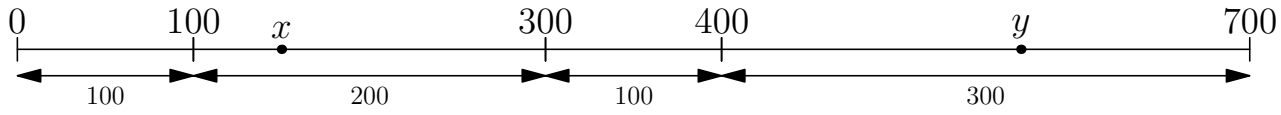
Vstup

```
4
100 200 100 300
80 90 50 120
3
300 400
0 700
150 570
```

Výstup

```
2.0000000
7.9722222
5.0833333
```

Cesta zo vstupu je znázornená na obrázku. Body x a y zodpovedajú tretej otázke. Pri ceste z x do y pôjdeme prvých 150 km rýchlosťou 90 km/h, potom 100 km rýchlosťou 50 km/h, a nakoniec 170 km rýchlosťou 120 km/h.

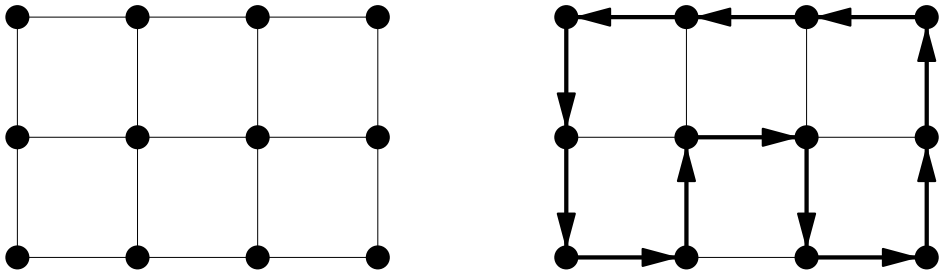


B-I-3 Squaplex

Squaplex je hlavolam, ktorý sa hrá na štvorčekovanom papieri v tvare obdĺžnika, obsahujúcom $R \times S$ mrežových bodov. Úlohou riešiteľa je položiť hrot ceruzky do ktoréhokoľvek mrežového bodu a nakresliť ťah spĺňajúci nasledujúce podmienky:

- Ceruzkou sa môžeme pohybovať len po čiarach hracieho plánu.
- Ťah musí začínať aj končiť v tom istom mrežovom bode.
- Cez každý mrežový bod na hracej ploche musíme ceruzkou prejsť práve raz. (Výnimkou je začiatočný bod, ktorý navštívime dvakrát.)

Na obrázku je hrací plán veľkosti 3×4 (vľavo) a jedno možné riešenie preň (vpravo).



Súťažná úloha:

a) (5 bodov) Zistite a dokážte, pre ktoré hodnoty R a S má tento hlavolam riešenie.
b) (5 bodov) Napíšte program, ktorý pre zadané hodnoty R a S navrhne jedno riešenie, ak existuje.

Formát vstupu:

Vstup obsahuje dve kladné celé čísla R a S, oddelené medzerou.

Môžete predpokladať, že aspoň jedno z čísel R a S je väčšie ako 1.

Formát výstupu:

Ak neexistuje pre dané R a S riešenie, vypíšte jeden riadok so správou „`Nema riesenie.`“.

V opačnom prípade vypíšte popis jedného riešenia, ktoré začína aj končí v ľavom hornom rohu. Riešenie môžeme popísať ako postupnosť znakov `H` (hore), `D` (dole), `L` (doľava) a `P` (doprava).

Príklad:

Vstup	Výstup
`3 4`	`DDPHPDPHHLLL`

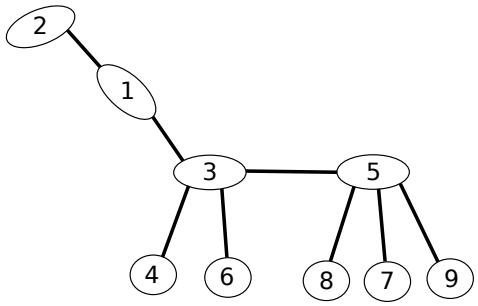
Táto postupnosť pohybov zodpovedá riešeniu na obrázku v zadaní.

B-I-4 Gaštanová žirafa

Zuzkino najobľúbenejšie ročné obdobie je jeseň. Vždy sa teší na deň, kedy dozrejú prvé gaštany, a ona si z nich bude môcť stavať rôzne zvieratká.

Určite ste už aj vy také zvieratko videli. Vyrába sa veľmi ľahko – zoberieme zápalku, oba jej konce zabodneme do dvoch gaštanov, a už máme prvú nožičku.

Zuzka si dnes nazbierala N gaštanov. Najskôr si na ne fixkou napísala čísla od 1 po N, lebo sa chcela hrať s mladším bratom Tomáškom na lotériu. Potom ale všetky gaštany zobrala a postavila z nich *veľanohú žirafu*.



Na obrázku vidíme príklad veľanohej žirafy. Každá veľanohá žirafa sa skladá z nasledujúcich gaštanov:

- 4 gaštany predstavujúce hlavu, krk, trup a zadok žirafy.
- aspoň 2 gaštany predstavujúce predné kopytá
- aspoň 2 gaštany predstavujúce zadné kopytá

Zápalkami sú prepojené nasledujúce dvojice gaštanov: hlava-krk, krk-trup, trup-zadok, trup-každé predné kopyto a zadok-každé zadné kopyto.

Zuzka vymyslela pre Tomáška nasledujúcu hru: Na začiatku Zuzka oznámi Tomáškovi hodnotu N. Tomáškovym cieľom je o každom gaštane od 1 do N zistiť, ktorú časť žirafy predstavuje. Tomáško však žirafu nevidí, lebo ju Zuzka pred ním schovala. Môže sa jej len pýtať otázky tvaru „sú gaštany a a b spojené zápalkou?“

Príklad hry:

```
Z: Počet gaštanov tvoriacich moju žirafu je 9.
T: Sú gaštany 1 a 2 spojené?        Z: Áno.
T: Sú gaštany 1 a 3 spojené?        Z: Áno.
T: Sú gaštany 3 a 4 spojené?        Z: Áno.
T: Sú gaštany 3 a 5 spojené?        Z: Áno.
T: Sú gaštany 3 a 6 spojené?        Z: Áno.
T: Sú gaštany 3 a 7 spojené?        Z: Nie.
```

```
T: Sú gaštany 5 a 8 spojené?        Z: Áno.
T: Sú gaštany 5 a 9 spojené?        Z: Áno.
T: Hlava je gaštan 2.
   Krk je gaštan 1.
   Trup je gaštan 3.
   Zadok je gaštan 5.
   Predné kopytá sú gaštany 4 a 6.
   Zadné kopytá sú gaštany 7, 8 a 9.
```

(Zuzkina žirafa v tomto príklade je tá z obrázku.)

Súťažná úloha:

Navrhnite čo najlepšiu stratégiu pre túto hru. Presnejšie, nájdite čo najefektívnejší algoritmus, ktorý bude túto hru hrať namiesto Tomáška a vždy vyhrá. Algoritmus stačí v riešení uviesť ako pseudokód (slovný popis), nie je potrebné implementovať ho v programovacom jazyku.

Na zisk 10 bodov potrebujete nájsť algoritmus, ktorého časová zložitosť bude lineárna od N. Za 7 bodov bude riešenie, ktorého časová zložitosť bude od N závisieť kvadraticky.

Pri odhade časovej zložitosti predpokladajte, že Zuzka dokáže na každú otázku odpovedať okamžite. Uvedomte si, že 10-bodové riešenie si ani zďaleka nemôže dovoliť opýtať sa Zuzky na všetky dvojice gaštanov.

Zadania krajského kola kategórie A

A-II-1 Aquapark

Manažment aquaparku sa rozhodol zistiť, ako veľmi sú využívané tobogany. Konkrétne v tomto aquaparku sú 3 tobogany. Manažéri majú k dispozícii tieto informácie:

- Ako dlho trvá jazda na každom tobogane (T_1, T_2, T_3)
- Ako dlho trvá dostať sa od koncov toboganov k začiatkom (D).
  Tobogany sú umiestnené tak, že od hocijakého konca k hocijakému začiatku to trvá rovnako dlho.
- Fotobunkou zistené časy, kedy niekto nasadol na daný tobogan (a_{ij}).

Z týchto informácií sa presný počet ľudí využívajúcich tobogany nemusí dať určiť. Vždy je ale jednoznačne určený *minimálny* počet ľudí, ktorí mohli tobogany využívať tak, aby to zodpovedalo zaznamenaným údajom.

Súťažná úloha:

Napíšte program, ktorý z daných informácií o dĺžke jazdy na toboganoch, trvaní cesty nahor a časoch jednotlivých spustení určí minimálny možný počet ľudí, ktorí mohli využívať tobogany.

Inými slovami, nájdite najmenšie číslo K také, že existuje rozvrh pre K ľudí, pri ktorom by sa na každom z toboganov v každom zo zaznamenaných časov niekto spustil. Nezabudnite, že keď niekto nasadne na tobogan i v čase T, tak ďalšiu jazdu (na hociktorom tobogane) môže začať najskôr v čase $T + T_i + D$. **Nezabudnite uviesť dôkaz správnosti vášho algoritmu.**

Formát vstupu:

V prvom riadku sú 3 čísla T_1, T_2, T_3: dĺžky trvania jázd na jednotlivých toboganoch. V ďalšom riadku je jedno číslo D: čas, ktorý trvá výstup nahor. Potom nasledujú 3 riadky. Riadok $i+2$ začína číslom N_i, udávajúcim počet jázd, ktoré sa na tobogane i uskutočnili. Potom nasleduje N_i čísel a_{ij} $(1 \leq j \leq N_i)$, ktoré vyjadrujú časy nástupov na tento tobogan. Pre každý tobogan je táto postupnosť časov usporiadaná vzostupne.

Platí: $0 \leq N_1, N_2, N_3 \leq 1\,000\,000$, $1 \leq T_1, T_2, T_3, D, a_{ij} \leq 500\,000\,000$.

Formát výstupu: Vypíšte jedno číslo: minimálny počet ľudí, pre ktorý mohla zaznamenaná situácia nastať.

Príklady:

Vstup

```
1 2 3
1
2 1 7
3 2 5 11
1 3
```

Výstup

```
2
```

Dvaja ľudia mohli všetky jazdy stihnúť nasledovne: prvý sa spustil na prvom, treťom, a opäť prvom tobogane (v časoch 1, 3 a 7). Druhý spravil všetky tri jazdy na druhom tobogane.

Vstup

```
4 5 6
10
2 2 3
3 1 7 15
1 5
```

Výstup

```
6
```

V tomto prípade žiaden návštevník nemohol stihnúť dve jazdy, preto určite bolo šesť návštevníkov.

A-II-2 Oplotenie farmy

Maroško sa dopočul, že v poľnohospodárstve sa točia veľké peniaze. Preto sa rozhodol, že v ňom začne podnikať. Netrvalo dlho a už vlastnil nádhernú veľkú farmu, na ktorej sa pestuje množstvo zaujímavých plodín. Maroškova poľnohospodárska pôda má obdĺžnikový tvar a je rozdelená na $R \times S$ rovnako veľkých štvorcových políčok. Na každom políčku je zasadená jedna poľnohospodárska plodina.

Nedávno sa farma ocitla v nebezpečí, pretože v okolí sa premnožili niektoré zvieratá, ktoré si radi pochutia na rastlinách na Maroškovej farme. Preto sa Maroško rozhodol, že postaví na farme plot, ktorý ochráni plody jeho práce.

Keďže v okolí nie je veľká konkurencia v oblasti stavebníctva, podarilo sa mu zohnať len jeden kontakt: firmu Štvorce s.r.o., ktorá sa špecializuje na stavebné práce štvorcového charakteru. Firma Štvorce s.r.o. sa ponúkla, že postaví na farme plot, ktorý ochráni štvorcový úsek pozemku.

Maroško teraz rozmýšľa, kde daný plot postaviť. Plot môže ochrániť štvorcové územie ľubovoľnej veľkosti. Navyše musí viesť cez hranice medzi políčkami, takže každé políčko ochráni alebo celé, alebo vôbec. Ďalšia Maroškova požiadavka je, aby plot ochránil políčka s aspoň dvoma rôznymi plodinami. Chce

mať totiž istotu, že na trhu neostane len s jedným typom produktu. Pomôžte Maroškovi sa aspoň trochu zorientovať a zistite, koľko možností na postavenie plotu má.

Súťažná úloha:

Daný je počet riadkov R, počet stĺpcov S a počet plodín K. Platí $1 \leq R, S \leq 2\,500$, $1 \leq K \leq 1\,000\,000\,000$. Na každom z ďalších R riadkov je S čísel – popis, ktoré plodiny sú zasadené na jednotlivých políčkach. Plodiny sú očíslované od 0 do $K - 1$. Vypíšte jedno číslo – koľkými spôsobmi môže Maroško postaviť na farme plot, ktorý ohradí štvorcovú oblasť, na ktorej sa pestujú aspoň dve rôzne plodiny.

Príklad:

Vstup

```
3 6 10
1 0 0 0 1 7
2 0 0 0 7 7
3 0 0 0 7 7
```

Výstup

```
8
```

Pre daný popis farmy máme 5 možností, ako postaviť plot okolo oblasti 2×2 *a 3 možnosti, ako postaviť plot okolo oblasti* 3×3.

A-II-3 Obmedzovač rýchlosti

Spoločnosť Expresná Pošta doručuje zásielky po celej Európe. V poslednom čase jej vodiči príliš často prehliadali dopravné značky, kvôli čomu niekoľkokrát dostali pokutu za vysokú rýchlosť. Riaditeľ spoločnosti preto rozhodol do každého auta zakúpiť obmedzovač rýchlosti. Ten funguje nasledovne: vodič si pred jazdou nastaví rýchlosť v a prístroj sa automaticky postará o to, že auto počas celej jazdy neprekročí rýchlosť v. Riaditeľ spoločnosti naviac vydal predpis, podľa ktorého si vodič musí nastaviť také obmedzenie rýchlosti, aby na trase, ktorou pôjde, neprekročil žiadnu maximálnu povolenú rýchlosť.

Súťažná úloha:

Daná je cestná sieť, po ktorej jazdia vodiči spoločnosti Expresná Pošta. Táto cestná sieť obsahuje N miest, medzi ktorými vedie dokopy M rôznych ciest. Každá cesta spája práve dve mestá, pričom cesty sa mimo miest navzájom križujú len mimoúrovňovo (teda mimo mesta nie je možné odbočiť na inú

cestu). Pre každú cestu poznáme jej dĺžku (v kilometroch) a maximálnu povolenú rýchlosť (v kilometroch za hodinu).

Pre danú dvojicu miest x a y môže existovať viacero spôsobov, ako sa po cestách dostať z x do y. Vašou úlohou je napísať program, ktorý pre **všetky možné** dvojice miest x a y určí minimálny čas potrebný na cestu z mesta x do mesta y pri použití obmedzovača rýchlosti a dodržaní riaditeľovho predpisu.

Formát vstupu:

Prvý riadok obsahuje dve kladné celé čísla N, M ($1 \leq N \leq 100$, $1 \leq M \leq 5\,000$). Číslo N určuje počet miest. Mestá na vstupe sú očíslované číslami 1 až N. Číslo M určuje počet ciest medzi nimi.

Každý z nasledujúcich M riadkov popisuje jednu cestu a obsahuje štyri čísla i j d m, udávajúce, že cesta spájajúca mestá i a j má dĺžku d kilometrov a maximálnu povolenú rýchlosť m kilometrov za hodinu. Všetky cesty na vstupe sú obojsmerné. Medzi dvoma mestami môže byť postavených viacero ciest s rôznou dĺžkou, resp. maximálnou rýchlosťou.

Môžete predpokladať, že medzi každou dvojicou miest existuje aspoň jedna trasa (možno tvorená viacerými nadväzujúcimi cestami).

Všetky vzdialenosti a rýchlosti na vstupe sú uvedené s presnosťou na najviac 3 desatinné miesta. Pre každú vzdialenosť d na vstupe platí $1 \leq d \leq 10^6$. Pre každú rýchlosť m na vstupe platí $5 \leq m \leq 10^5$.

Formát výstupu:

Výstup má obsahovať N riadkov, každý riadok obsahujúci N čísel. Číslo v i-tom riadku a j-tom stĺpci určuje minimálny čas (v hodinách) potrebný na jazdu medzi mestom i a j. Výsledok stačí uviesť s presnosťou na 3 desatinné miesta.

Pri práci s reálnymi číslami v počítači môžu vznikať zaokrúhľovacie chyby. Túto skutočnosť môžete vo svojom riešení ignorovať – ako keby všetky výpočty, ktoré potrebujete, boli presné.

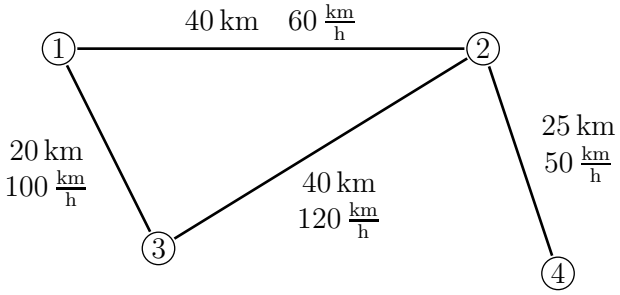
Príklad:

Vstup

```
4 4
1 2 40.0 60.0
1 3 20.0 100.0
2 3 40.0 120.0
2 4 25.0 50.0
```

Výstup

```
0.000 0.600 0.200 1.300
0.600 0.000 0.333 0.500
0.200 0.333 0.000 1.300
1.300 0.500 1.300 0.000
```



Najrýchlejší spôsob, ako sa dostať z mesta 1 do mesta 2 je cez mesto 3: pôjdeme 60 km rýchlosťou 100 km/h. Všimnite si ale, že najrýchlejší spôsob, ako sa dostať z mesta 1 do mesta 4 je po trase 1-2-4, nie 1-3-2-4.

A-II-4 Počítač s gumenou rúrou

V tomto ročníku olympiády sa v teoretickej úlohe stretávame so špeciálnym počítačom zvaným Kvak. V študijnom texte uvedenom za zadaním tejto úlohy je popísané, ako tento počítač funguje. Študijný text je až na štylistické zmeny identický s tým z domáceho kola.

Súťažná úloha:

a) (3 body)
Lucasove čísla sú definované nasledovne: $L_0 = 2$, $L_1 = 1$ a pre ľubovoľné $n \geq 2$ platí $L_n = L_{n-1} + L_{n-2}$.
V rúre je jedno číslo n. Napíšte program pre Kvak, ktorý vypočíta a vypíše hodnotu (L_n mod 65 536).

b) (3 body)
V rúre je neprázdna postupnosť *kladných* čísel. Napíšte program pre Kvak, ktorý zistí, či sa v tejto postupnosti vyskytuje číslo 47 a podľa toho vypíše buď číslo 1 (ak áno) alebo číslo 0 (ak nie).

c) (4 body)
V rúre je neprázdna postupnosť *kladných* čísel. Napíšte program pre Kvak, ktorý na výstup vypíše všetky párne čísla v nej. (Na poradí, v akom ich vypíše, nezáleží, ale každé párne číslo musí vypísať práve toľkokrát, koľkokrát sa vyskytlo na vstupe.)

Tabuľku s povolenými inštrukciami nájdete na strane 10, študijný text začína na strane 11.

Zadania krajského kola kategórie B

B-I-1 Telemánia

Nemenovaný mobilný operátor by chcel v rámci série vianočných prekvapení udeliť špeciálny bonus tomu svojmu zákazníkovi, ktorý v poslednom roku telefonoval najviac. Vašou úlohou bude napísať pre tohto mobilného operátora program, ktorý dotyčného zákazníka nájde.

K dispozícii máte dáta uložené v dvoch tabuľkách. V prvej tabuľke sú telefónne čísla a mená všetkých zákazníkov nášho operátora, druhá tabuľka obsahuje zoznam všetkých hovorov za posledný rok, v ktorých aspoň jeden z účastníkov (t. j. volajúci, volaný, alebo obaja) bol zákazníkom nášho operátora.

Súťažná úloha:

Napíšte program, ktorý načíta dáta o zákazníkoch a hovoroch za posledný rok a vypíše meno zákazníka nášho mobilného operátora, ktorý za posledný rok v súčte telefonoval najdlhší čas.

Formát vstupu:

Prvý riadok vstupu obsahuje jedno celé číslo N ($1 \leq N \leq 1\,000\,000$) – počet všetkých zákazníkov nášho mobilného operátora. Každý z nasledujúcich N riadkov obsahuje telefónne číslo a meno jedného zákazníka. Môžete predpokladať, že telefónne číslo bude obsahovať presne 10 cifier a meno bude obsahovať najviac 200 znakov. Pre jednoduchosť môžete tiež predpokladať, že meno neobsahuje medzery.

Ďalej nasleduje jeden riadok, obsahujúci jedno celé číslo M ($1 \leq M \leq 1\,000\,000$) – počet zaznamenaných hovorov. Každý z nasledujúcich M riadkov obsahuje dve telefónne čísla oddelené medzerou, za ktorými nasleduje jedno celé číslo určujúce dĺžku hovoru v sekundách.

Pozor, telefónne čísla v tabuľke hovorov nemusia byť nutne len zákazníci nášho operátora!

Formát výstupu:

Výstup má obsahovať jediný riadok a v ňom telefónne číslo a meno nášho zákazníka, ktorý podľa záznamov hovorov dohromady telefonoval najdlhší čas.

Ak je takýchto zákazníkov viac, stačí vypísať ľubovoľného jedného z nich.

Príklad:

Vstup

```
3
0972125726 Jozko
0975342583 Ferko
0972654124 Jano
5
0975342583 0972654124 80
0972125726 0955956114 137
0955956114 0975342583 293
0972125726 0972654124 36
0972125726 0975342583 54
```

Výstup

```
0975342583 Ferko
```

Ferko dohromady telefonoval 427, Jožko 227 a Jano 116 sekúnd.

Všimnite si, že medzi hovormi sa tiež nachádza číslo `0955956114`*, na ktoré sa uskutočnilo až 430 sekúnd hovorov. Toto číslo ale nepatrí žiadnemu z našich zákazníkov, preto nemôže vyhrať.*

B-II-2 Kráľovské cesty

Santoland je nádherná krajina, ale má veľmi zlú infraštruktúru. Telefóny ledva fungujú, o internete môžu len snívať a cesty sú v celej krajine len tri.

Každá z týchto troch ciest začína v hlavnom meste a žiadne dve sa už potom nikde nekrižujú. Každé mesto v Santolande (okrem hlavného) leží na práve jednej z týchto ciest. V každom meste vedia obyvatelia, cez ktoré susedné mesto sa dostanú do hlavného mesta, a taktiež vedia vzdialenosť do tohto mesta v kilometroch.

Kráľ Banto sa rozhodol, že v rámci podpory turistického ruchu zriadi telefonickú linku, na ktorú bude môcť ktokoľvek zavolať a spýtať sa na vzdialenosť ľubovoľných dvoch miest.

Ako už však odznelo, telefóny ledva fungujú a hovor nikdy nevydrží dlhšie ako minútu bez prerušenia. Preto je potrebné napísať program, ktorý bude na linke odpovedať na otázky volajúcich *veľmi* efektívne.

Súťažná úloha:

Vašou úlohou bude napísať program, ktorý načíta popis krajiny, prípadne ho nejak spracuje, a potom bude čo najrýchlejšie spracovávať jednotlivé telefonáty.

Popis krajiny sa nachádza v textovom súbore `mapa.txt`.

V prvom riadku súboru je celé číslo N, ktoré označuje počet miest v Santolade. Hlavné mesto má číslo 1 a ostatné mestá majú čísla 2, ..., N, pričom čísla miest nesúvisia s ich umiestnením na cestách.

Druhý až N-tý riadok popisujú mestá 2 až N. Riadok popisujúci mesto i obsahuje dve kladné celé čísla m_i a d_i, kde m_i je číslo najbližšieho mesta na ceste z mesta i do hlavného mesta (mesta 1) a d_i je dĺžka cesty medzi mestami i a m_i.

Môžete predpokladať, že vstup je korektný, teda medzi hodnotami m_x sa číslo 1 nachádza práve 3× (do hlavného mesta vedú 3 cesty), každé iné číslo sa tam nachádza najviac raz a z každého mesta sa dá dostať do hlavného.

Po tom, ako váš program načíta a spracuje tieto údaje, musí fungovať v nekonečnom cykle, v ktorom striedavo načíta zo štandardného vstupu dve čísla miest a a b a vypíše vzdialenosť $d_{a,b}$ medzi týmito dvomi mestami.

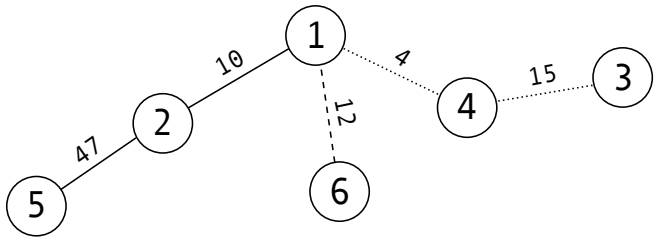
Bodovanie:

Prvoradým kritériom hodnotenia riešení bude efektívnosť spracovania jednotlivých telefonátov. (Následne prihliadneme aj k časovej zložitosti predspracovania a pamäťovej zložitosti vášho programu.)

Príklad:

`mapa.txt`

```
6
1 10
4 15
1 4
2 47
1 12
```



Obrázok znázorňuje krajinu popísanú v súbore `mapa.txt`*. Napr. keďže 5. riadok vstupu je* `2 47`*, tak vieme, že z mesta 5 vedie do mesta 2 cesta dĺžky 47. Každá cesta z hlavného mesta je kreslená iným typom čiary.*

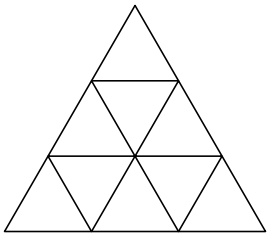
Vstup

```
5 2
1 6
1 3
5 3
5 6
...
```

Výstup

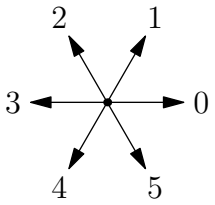
```
47
12
19
76
69
...
```

B-II-3 Triplex



Hrací plán pre hru Triplex má tvar rovnostranného trojuholníka so stranou dĺžky N, ktorý je rozdelený na jednotkové trojuholníky. Príklad hracieho plánu pre $N = 3$ vidíte na prvom obrázku vpravo.

Slovom *vrcholy* označujeme všetky vrcholy všetkých jednotkových trojuholníkov tvoriacich hrací plán. Slovom *hrany* označujeme všetky ich strany. Hrací plán na obrázku má teda 10 vrcholov a 18 hrán.



Na hracom pláne sa nachádza jedna figúrka. Na začiatku hry je táto figúrka v „hornom“ vrchole trojuholníka. Existuje 6 smerov, ktorými môžeme figúrkou hýbať. Tieto smery si očíslujeme tak, ako je znázornené na druhom obrázku vpravo.

Figúrka nesmie pri svojom pohybe opustiť hrací plán. Napr. v prvom ťahu sa teda musí pohnúť buď smerom 4, alebo smerom 5.

Súťažná úloha:

a) *(5 bodov)* Napíšte program, ktorý načíta hodnotu N a vypíše postupnosť ťahov figúrkou, pri ktorej figúrka navštívi práve raz každý vrchol a na konci sa vráti do vrcholu, kde začínala.

b) *(5 bodov)* Napíšte program, ktorý načíta hodnotu N a vypíše postupnosť ťahov figúrkou, pri ktorej figúrka prejde práve raz po každej hrane a na konci sa vráti do vrcholu, kde začínala.

Riešenie každej podúlohy bude hodnotené samostatne, môžete riešiť podúlohu b) aj ak neviete riešiť a).

Formát vstupu:

Na vstupe je najskôr písmeno (`A` alebo `B`), udávajúce, ktorú podúlohu má program riešiť, a potom kladné celé číslo N.

Formát výstupu:

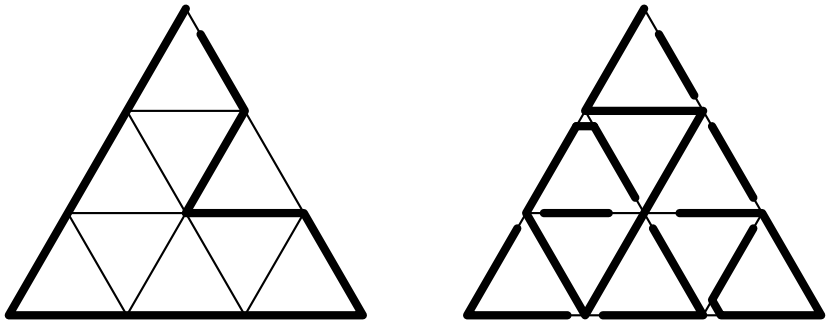
Výstup má obsahovať jediný riadok a v ňom postupnosť cifier 0 až 5 popisujúcich jednotlivé ťahy. Ak existuje viacero riešení, stačí vypísať jedno ľubovoľné.

Príklady:

Vstup	Výstup
`A 3`	`4440002312`

Vstup	Výstup
`B 3`	`4044215533100531 22`

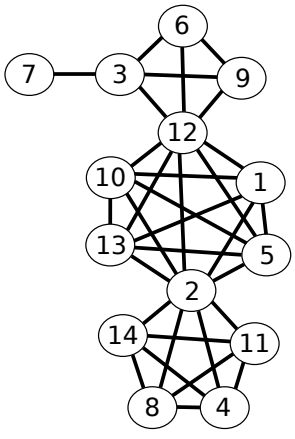
Na nasledujúcich obrázkoch sú znázornené postupnosti ťahov zodpovedajúce vstupom a výstupom z príkladu.



Čiara na obrázku vpravo je miestami prerušovaná, aby bolo lepšie vidieť, ktoré jej časti na seba nadväzujú na miestach, kde križuje samú seba.

B-II-4 Gaštanový snehuliak

Pamätáte si zo zadaní domáceho kola Zuzkinu gaštanovú žirafu? Keď prišla zima, Zuzka sa rozhodla rozobrať ju a namiesto nej postaviť z gaštanov objekt, ktorý sa k zime hodí omnoho viac – snehuliaka.



Gaštanový snehuliak sa, podobne ako klasický, skladá z troch gulí. Guľa sa z gaštanov vyrobí tak, že zoberieme niekoľko gaštanov (aspoň štyri) a každé dva spojíme zápalkou.

Aby snehuliak dobre držal pokope, vyrobila ho Zuzka tak, že stredná guľa má práve jeden gaštan spoločný s hornou a práve jeden iný gaštan s dolnou guľou.

A aby bol snehuliak dokonalý, ešte dostal nos-mrkvu. Teda nos-gaštan: jeden gaštan, ktorý je zápalkou spojený s niektorým jedným gaštanom v hornej guli. (Nie s tým, ktorý patrí aj do strednej gule.)

Príklad takéhoto snehuliaka nájdete na obrázku napravo. Horná guľa je tvorená štyrmi, stredná šiestimi a spodná piatimi gaštanmi. Spoločný gaštan pre hornú a strednú guľu má číslo 12, spoločný gaštan pre strednú a dolnú guľu má číslo 2. Gaštan predstavujúci nos má číslo 7.

Keď Zuzka dostavala snehuliaka, zavolala brata Tomáška, aby sa opäť zahrali jej obľúbenú hru. Pripomeňme si jej pravidlá: Na začiatku Zuzka oznámi Tomáškovi hodnotu N – celkový počet gaštanov, ktoré tvoria snehuliaka. Tieto gaštany sú nejako náhodne očíslované od 1 po N. Tomáškovym cieľom je o každom gaštane od 1 do N zistiť, ktorú časť snehuliaka predstavuje. Tomáško však snehuliaka nevidí, lebo ho Zuzka pred ním schovala. Môže sa jej len pýtať otázky tvaru „sú gaštany a a b spojené zápalkou?“

Príklad hry pre snehuliaka z obrázka:

```
Z: Počet gaštanov tvoriacich snehuliaka je 14.
T: Sú gaštany 1 a 2 spojené?        Z: Áno.
T: Sú gaštany 1 a 3 spojené?        Z: Nie.
T: Sú gaštany 2 a 4 spojené?        Z: Áno.
...
T: Sú gaštany 7 a 3 spojené?        Z: Áno.
T: Nos je gaštan 7, hornú guľu tvoria gaštany 3 6 9 12,
   strednú 1 2 5 10 12 13 a dolnú 2 4 8 11 14.
```

Súťažná úloha:

Navrhnite čo najlepšiu stratégiu pre túto hru. Presnejšie, nájdite čo najefektívnejší algoritmus, ktorý bude túto hru hrať namiesto Tomáška a vždy vyhrá. Algoritmus stačí v riešení uviesť ako pseudokód (slovný popis), nie je potrebné implementovať ho v programovacom jazyku.

Na zisk 10 bodov potrebujete nájsť algoritmus, ktorého časová zložitosť bude lineárna od N. Za 7 bodov bude riešenie, ktorého časová zložitosť bude od N závisieť kvadraticky.

Pri odhade časovej zložitosti predpokladajte, že Zuzka dokáže na každú otázku odpovedať okamžite. Uvedomte si, že 10-bodové riešenie si ani zďaleka nemôže dovoliť opýtať sa Zuzky na všetky dvojice gaštanov.

Zadania celoštátneho kola kategórie A

A-III-1 Čokoláda je tu zas

Janko bude mať čoskoro narodeniny. Jeho sestra Marienka si ešte dobre pamätá na olámanú čokoládu, ktorú jej daroval pred pár mesiacmi. Rozhodla sa teda, že sa mu oplatí rovnakou mincou.

Zašla do pivnice a zo svojej tajnej skrýše vybrala čokoládu, ktorú si tam kedysi ukryla. Všadeprítomné myši už aj túto čokoládu ohrýzli, to však Marienke neprekážalo – veď predsa stačí vyhryzené časti olámať.

Marienka je však šikovnejšia ako Janko a uvedomila si, že nemusí lámaním vyrobiť štvorec. Čokolády sú predsa často obdĺžnikového tvaru. A to jej ponúka množstvo nových možností ako vyrobiť darček pre Janka.

Súťažná úloha:

Daný je pôvodný počet riadkov R a stĺpcov S, ktoré čokoláda kedysi mala, a matica $R \times S$ núl a jednotiek udávajúca, ktoré políčka z nej zostali celé.

Zistite, koľkými spôsobmi môže Marienka uskutočniť svoj plán. Inými slovami, spočítajte, koľkými spôsobmi sa dá na zvyšku čokolády vyznačiť obdĺžnik bez dier. Všetky hrany obdĺžnika musia samozrejme ležať na hranách políčok. Rovnako veľké obdĺžniky na rôznych súradniciach považujeme za rôzne.

Formát vstupu:

V prvom riadku vstupu sú dve medzerami oddelené celé čísla R a S. Nasleduje R riadkov, v r-tom z nich je S medzerami oddelených celých čísel $a_{r,1}, \ldots, a_{r,s}$. Ak je políčko (r, s) celé, je $a_{r,s} = 1$, inak $a_{r,s} = 0$.

Formát výstupu:

Vypíšte jeden riadok a v ňom jedno celé číslo – hľadaný počet obdĺžnikov.

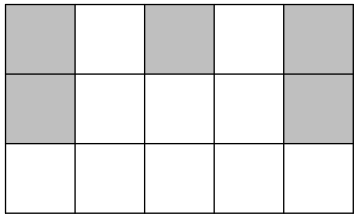
Príklad:

Vstup

```
3 5
0 1 0 1 0
0 1 1 1 0
1 1 1 1 1
```

Výstup

```
33
```



Na obrázku je čokoláda popísaná vstupom. Sivou farbou sú políčka, ktoré už myši stihli poškodiť.

Obdĺžnik 1×1 *sa na nej dá vyznačiť desiatimi spôsobmi,* 2×1 *piatimi,* 1×2 *šiestimi,* 3×1 *dvoma,* 1×3 *štyrmi,* 1×4 *dvoma,* 2×2 *dvoma,* 1×5 *jedným, a* 2×3 *tiež jedným spôsobom.*

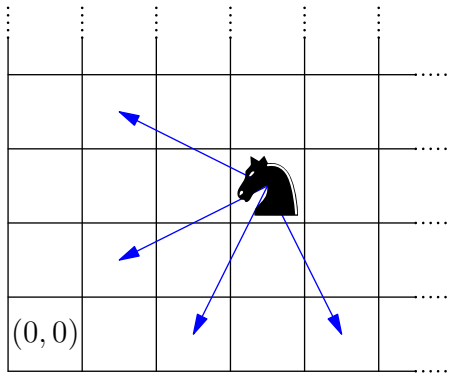
A-III-2 Odveta

„Šach-mat," s posmešným úškrnom zahlásil CéDéčko. Jeho protihráč Petržlen mal síce doteraz šach veľmi rád, ale už ho to prestáva baviť: práve s CéDéčkom prehral sedemnástu partiu za sebou. Preto si vymyslel novú, vlastnú hru, v ktorej ho určite porazí.

Hracím plánom je šachovnica, ktorá je doprava aj dohora nekonečná. Každé políčko tejto šachovnice teda vieme popísať dvomi nezápornými celými číslami.

Na tejto šachovnici je umiestnených N koní. Na začiatku aj hocikedy počas hry sa môže na jednom políčku naraz nachádzať ľubovoľne veľa koní.

Koňmi je povolené ťahať len podľa šachových pravidiel. Sú povolené len tie ťahy, pri ktorých kôň neopustí šachovnicu a priblíži sa do ľavého dolného rohu. Presnejšie, ťah z (x, y) na (x', y') je povolený len vtedy, keď $x', y' \geq 0$ a $x' + y' < x + y$.



Všetky povolené ťahy koňom na súradniciach $(3, 2)$.

Hráč, ktorý je na ťahu, si vyberie niekoľko koní (môže koľko chce, ale musí aspoň jedného) a každým z nich raz pohne. Hráči ťahajú striedavo. Prehráva ten, pre koho už neexistuje ďalší ťah (teda už nemôže pohnúť žiadnym koňom).

Petržlen si pre túto hru už napísal program, ktorý bude za neho hrať optimálne. Žiaľ, CéDéčko programovať nevie, preto by privítal vašu pomoc.

Súťažná úloha:

Je daný počet koňov N a ich umiestnenie na šachovnici na začiatku. Prvý je na ťahu CéDéčko.

Napíšte program, ktorý zistí, kto vyhrá, ak budú obaja hráči hrať optimálne. Ak má byť víťazom CéDéčko, váš program by mu mal poradiť prvý ťah ľubovoľnej vyhrávajúcej stratégie.

V prípade, že úlohu neviete riešiť pre všeobecné N, tak dostanete pár bodov aj vtedy, ak úlohu vyriešite pre jedného koníka, prípadne pre dva koníky začínajúce na súradniciach medzi $(0, 0)$ a $(100, 100)$.

Formát vstupu:

Prvý riadok vstupu obsahuje počet koní N. V každom z nasledujúcich N riadkov sú dve čísla r_i a s_i, určujúce riadok a stĺpec, v ktorom sa nachádza i-ty kôň na začiatku hry.

Formát výstupu:

Prvý riadok výstupu má obsahovať meno hráča, ktorý vyhrá (`CeDecko` alebo `Petrzlen`). Ak vyhrá CéDéčko, vypíšte pre každého koňa, ktorým má tiahnuť, riadok s číslami r_a, s_a, r_b, s_b – pôvodné a nové umiestnenie koňa. Na poradí týchto riadkov nezáleží.

Príklady:

Vstup	Výstup
`1` `0 4`	`Petrzlen`

CéDéčko musí potiahnuť na $(1, 2)$, *odtiaľ Petržlen potiahne na* $(0, 0)$ *a vyhrá.*

Vstup	Výstup

`2`
`3 1`
`2 1`
	`CeDecko`
	`3 1 1 0`
	`2 1 0 0`

Po tomto ťahu CéDéčka Petržlen rovno prehral, lebo ani jedným koňom už nevie pohnúť. Všimnite si, že keby CéDéčko koňa z $(3, 1)$ *nechal na mieste, prehrá. Tiež by prehral, ak by tohto koňa presunul na* $(1, 2)$, *bez ohľadu na to, či by druhým koňom pohol alebo nie.*

A-III-3 Počítač s gumenou rúrou

V tomto ročníku olympiády sa v teoretickej úlohe stretávame so špeciálnym počítačom zvaným Kvak. V študijnom texte uvedenom za zadaním tejto úlohy

je popísané, ako tento počítač funguje. Študijný text je až na štylistické zmeny identický s tým z domáceho kola.

Súťažná úloha:

a) (3 body) V rúre je postupnosť kladných celých čísel. Označme ich $a_1, \ldots, a_N$ v poradí, v akom sa v nej teraz nachádzajú. Napíšte program, ktorý skontroluje, či je táto postupnosť rastúca. Ak áno, mal by váš program skončiť bez toho, aby čokoľvek vypísal. Ak nie je rastúca, mal by nájsť a vypísať najmenšie i také, že $a_i \geq a_{i+1}$.

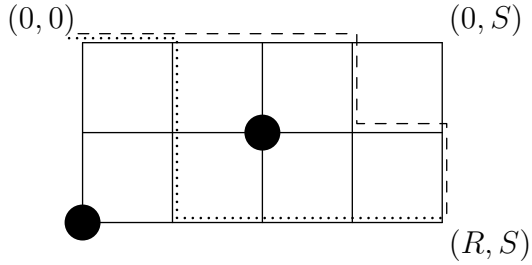
b) (7 bodov) V rúre je postupnosť kladných celých čísel, pričom viete, že jedno z týchto čísel má v rúre nadpolovičnú väčšinu – toto číslo sa teda v rúre vyskytuje viackrát ako všetky ostatné čísla dokopy. Napíšte program, ktorý toto číslo nájde a vypíše.

Tabuľku s povolenými inštrukciami nájdete na strane 10, študijný text začína na strane 11.

A-III-4 Mravce idú!

V Absurdistane sa ide konať jedna veľmi podivná súťaž. Porota si pripravila N rovnakých mriežok obdĺžnikového tvaru. Každá mriežka je tvorená $R + 1$ vodorovnými a $S + 1$ zvislými čiarami.

Každý zo súťažiacich dostane jednu mriežku a nejaký počet prekážok. Úlohou súťažiaceho je všetky prekážky, ktoré dostal, rozostaviť na rôzne mrežové body. Následne takto zablokovanú mriežku odovzdá porote.



Príklad mriežky pre $R = 2$ a $S = 4$. Vyznačené sú dve z 5 ciest mravcov.

Porota do ľavého horného rohu vypustí špeciálny druh mravcov. Tieto vedia chodiť iba dolu a doprava, a to iba po čiarach tvoriacich mriežku. A navyše sa tieto mravce nemajú rady, takže žiadne dva nepôjdu presne tou istou cestou. Preto do pravého dolného rohu príde presne toľko mravcov, koľko je rozličných ciest medzi ľavým horným a pravým dolným rohom. Keďže ale tento počet môže byť skutočne obrovský,

pri vyhodnocovaní súťaže sa berie do úvahy len zvyšok, ktorý tento počet dáva po delení číslom $10^9 + 9$.

Súťažná úloha:

Daných je N mriežok, pričom všetky majú po $R + 1$ vodorovných a $S + 1$ zvislých čiar. Pre každú mriežku je daný počet prekážok na nej a ich súradnice. Vašou úlohou je pre každú mriežku určiť hodnotu $X \bmod 1\,000\,000\,009$, kde X je počet rôznych spôsobov, ktorými sa dá dostať z ľavého horného do pravého dolného rohu dotyčnej mriežky.

Dobrá rada:

Pozor na pretečenie rozsahu premenných. Pri niektorých možných riešeniach je pri pomocných výpočtoch potrebné použiť 64-bitové celočíslené premenné (`long long` v C/C++, `int64` v Pascale).

Formát vstupu:

V prvom riadku vstupu sú tri medzerami oddelené kladné celé čísla R, S a N. Nasleduje N popisov mriežok.

Každý popis mriežky začína riadkom obsahujúcim jedno nezáporné celé číslo K, udávajúce počet prekážok na nej. Nasleduje K riadkov, v každom sú dve medzerou oddelené čísla r_i, s_i ($0 \leq r_i \leq R$, $0 \leq s_i \leq S$), kde r_i je súradnica riadku a s_i súradnica stĺpca, v ktorom leží i-ta prekážka. Žiadne dve prekážky neležia na tých istých súradniciach a žiadna prekážka neleží na súradniciach $(0, 0)$, kde mravce začínajú.

Obmedzenia veľkosti premenných:

V testovacích dátach budú premenné R, S, N a K nanajvýš rovné tým v nasledujúcej tabuľke:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	5	7	10	100	1000	1200	1000	1000	2000	15000	1000	2500	20000	100000	100000
S	5	7	10	100	1000	100000	1000	1000	2000	15000	1000	2500	10000	100000	100000
N	5	7	10	50	50	1	1000	1000	200	50	1000	200	200	2	100
K	5	7	10	15	15	15	2	10	15	10	50	100	50	30	50

Formát výstupu:

Pre každú mriežku vypíšte jeden riadok a v ňom jedno celé číslo: hodnotu $X \bmod 1\,000\,000\,009$, kde X je počet rôznych spôsobov, ktorými sa dá dostať z ľavého horného do pravého dolného rohu dotyčnej mriežky.

Príklad:

Vstup

```
2 4 2
2
1 2
2 0
2
0 1
1 1
```

Výstup

```
5
1
```

Prvá mriežka je tá nakreslená v zadaní.
V druhej jediná voľná cesta vedie cez bod $(2, 0)$.

A-III-5 Hurikán

Súostrovie Kiribati tvorí N ostrovov. Ešte predvčerom bola medzi týmito ostrovmi postavená celá sieť mostov, po ktorých sa dalo pohodlne prejsť z každého ostrova na každý iný. Včera sa ale prehnal cez Kiribati hurikán Hermano a mnohé mosty postŕhal. Zostalo ich stáť len M. A čo je ešte horšie, spomedzi týchto M mostov má K porušenú statiku. Miestny statik zistil, že každý z najbližších K dní padne práve jeden z týchto K mostov.

Miestna vláda samozrejme potrebuje zabezpečiť, aby sa obyvatelia naďalej vedeli v každý deň dostať z každého ostrova na každý iný. Jediný spôsob, ako to dosiahnuť, je zaplatiť niekoľko prievozníkov od Školenej Prievozníckej Mafie Námorných Dopravcov a Gondolierov (ŠPMNDaG). Každý prievozník vie zabezpečiť dopravu medzi jednou konkrétnou dvojicou ostrovov.

Súťažná úloha:

Váš program dostane na vstupe popis mostov, ktoré zostali stáť po hurikáne a poradie, v akom spadnú tie z nich, ktoré sú poškodené. Z týchto údajov by mal pre dnešok aj pre každý z nasledujúcich K dní spočítať, aký najmenší počet prievozníkov stačí v ten deň najať.

Formát vstupu:

Prvý riadok vstupu obsahuje dve celé čísla N a M ($2 \leq N$, $1 \leq M$), udávajúce počet ostrovov a mostov. Ostrovy sú očíslované od 1 do N.

Každý z nasledujúcich M riadkov popisuje jeden most. Most je zadaný dvojicou celých čísiel a_i b_i ($1 \leq a_i, b_i \leq N$, $a_i \neq b_i$) – čísla ostrovov, ktoré daný most spája. Mosty si tiež očíslujeme od 1 po M, v poradí, v akom sú zadané na vstupe.

Nasleduje riadok obsahujúci celé číslo K ($1 \leq K \leq M$), udávajúce počet poškodených mostov. V poslednom riadku je K medzerami oddelených čísiel – čísla poškodených mostov v poradí, v akom spadnú.

Obmedzenia veľkosti premenných:

Vo všetkých testovacích vstupoch bude platiť $M, N \leq 200\,000$. V sadách 1 až 4 bude platiť $N \leq 1000$. V sadách 1 až 7 bude platiť $K \leq 100$.

Formát výstupu:

Vypíšte $K + 1$ riadkov, pričom v i-tom riadku bude jedno celé číslo: počet prievozníkov potrebný po tom, ako spadne prvých $i - 1$ mostov.

Príklad:

Vstup

```
4 4
1 2
2 3
1 3
3 4
3
2 4 3
```

Výstup

```
0
0
1
2
```

Ešte stále sa dá po mostoch prejsť medzi každými dvoma ostrovmi. A bude sa to dať aj zajtra, po páde mosta $2 - 3$.

Po páde mosta $3 - 4$ *už bude potrebné najať prievozníka, aby nebol ostrov 4 izolovaný od ostatných.*

Po páde mosta $1 - 3$ *už zostane stáť jediný most:* $1 - 2$. *Vtedy už bude potrebné najať dvoch prievozníkov.*

Riešenia domáceho kola kategórie A

A-I-1 Maliar Bonifác

Na získanie 3 bodov stačilo robiť presne to, čo mal pôvodne robiť Bonifác: spravíme si K-prvkové pole, kde každý prvok bude predstavovať jeden meter chodníka. Teraz postupne čítame príkazy a zakaždým príslušný úsek chodníka prefarbíme, t. j., do príslušných premenných poľa priradíme novú farbu.

Toto riešenie má pamäťovú zložitosť $O(K)$ a časovú $O(NK)$. Čo môžeme zlepšiť?

Predstavme si, že Bonifác nebude najskôr nič maľovať. Vypočuje si všetky požiadavky, potom zoberie kriedu a pre každú požiadavku spraví na chodník dve čiary – na začiatku a na konci dotyčného úseku. Takto rozdelí chodník na $U \leq 2N + 1$ úsekov. A je zjavné, že každý z úsekov, ktoré takto dostaneme, bude celý jednej farby. Stačí teda pre každý z nich zistiť jeho farbu.

Takto sme sa úspešne zbavili premennej K udávajúcej dĺžku chodníka. Použitému triku sa zvykne hovoriť *kompresia súradníc*. A teraz nám teda stačí spraviť si pole dĺžky U a ofarbovať jeho prvky (predstavujúce jednotlivé úseky chodníka). Takéto riešenie má časovú zložitosť $O(N^2)$ a dalo sa zaň získať 6 bodov.

Ukážeme teraz dva rôzne prístupy, ktoré povedú k rôznym riešeniam za plný počet bodov. Prvý prístup bude založený na myšlienke, že sa prejdeme po chodníku z jedného konca na druhý a o každom mieste zistíme, akú bude mať na konci farbu. Druhý prístup bude založený na simulácii požiadaviek v takom poradí, v akom ich Bonifác dostal, ale použijeme lepšiu dátovú štruktúru ako obyčajné pole.

Zametanie:

V tomto riešení pôjdeme po chodníku, pričom si budeme v každom okamihu pamätať množinu čísel príkazov, ktoré obsahujú miesto, kde práve sme. Samozrejme, aktuálne miesto má farbu podľa toho z nich, ktorý bol posledný – a teda má najväčšie číslo. A kedy sa množina príkazov zmení? Vtedy, keď prídeme na miesto, kde nejaký príkaz začína alebo končí.

Celé riešenie bude teda vyzerať nasledovne: Do poľa uložíme všetky začiatky a konce príkazov. Toto pole utriedime. Takto rozdelíme chodník na $2N + 1$ úsekov. O každom z nich vieme rovno povedať jeho dĺžku. (Ak na niektorom

mieste začínalo alebo končilo viacero príkazov, budú mať niektoré úseky dĺžku nulovú, ale to nám nevadí.) A keď ich budeme spracúvať postupne, vieme si udržiavať množinu aktívnych príkazov.

V našej implementácii sme na uloženie množiny „aktívnych“ príkazov použili `set`. Celé toto riešenie sa však dá ľahko implementovať aj v Pascale – vystačíme si napríklad s dvoma haldami. V jednej budú uložené všetky začiatky a konce príkazov, ktoré ešte treba spracovať, v druhej budú čísla príkazov, ktoré sú momentálne aktívne.

Takto dostávame riešenie s časovou zložitosťou $O(N \log N)$ a pamäťovou zložitosťou $O(N)$.

Listing programu:

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;

class udalost {
  public:
    int pozicia, prikaz; bool zacina;
    udalost(int po=0, int pr=0, bool z=true):pozicia(po),prikaz(pr),zacina(z) {}
};
bool skor(const udalost &a, const udalost &b) { return a.pozicia < b.pozicia; }

int N, F, K;
set<int> aktivne;           // cisla momentalne aktivnych prikazov
vector<udalost> udalosti;   // zaciatky a konce prikazov
vector<int> farby;          // ku kazdej farbe pocet litrov ktore treba
vector<int> farby_prikazov; // ku kazdemu prikazu jeho farba

int main() {
  // nacitame vstup
  cin >> N >> F >> K;
  for (int i=0; i<N; i++) {
    int z,k,f;
    cin >> z >> k >> f;
    farby_prikazov.push_back(f);
    udalosti.push_back(udalost(z,i,true));
    udalosti.push_back(udalost(k,i,false));
  }

  // utriedime udalosti
  sort( udalosti.begin(), udalosti.end(), skor );

  // spracujeme udalosti
  farby.resize(F+1,0);
```

```
  aktivne.insert( udalosti[0].prikaz );
  for (int i=1; i<2*N; i++) {
    int farba = 0, dlzka = udalosti[i].pozicia - udalosti[i-1].pozicia;
    if (!aktivne.empty()) farba = farby_prikazov[ *aktivne.rbegin() ];
    farby[farba] += dlzka;
    if (udalosti[i].zacina) aktivne.insert( udalosti[i].prikaz );
    else aktivne.erase( udalosti[i].prikaz );
  }

  // vypiseme vysledok
  for (int f=1; f<=F; f++) cout << farby[f] << endl;
}
```

Simulácia – prvá možnosť:

Potrebujeme teraz vymyslieť efektívny spôsob, ako si pamätať ofarbenie chodníka – ešte lepší ako ten v 6-bodovom riešení. Existuje hneď viacero možností, ako na to, ale všetky majú spoločné jednu vec – potrebujeme sa vedieť efektívne „zbaviť“ úsekov, ktoré práve spracúvaným príkazom celé prekryjeme.

Ak máme k dispozícii vyvažovaný binárny strom (napr. `set` v STL), nepotrebujeme ani strácať čas kompresiou súradníc. Stačí si jednoducho v strome pamätať všetky jednofarebné úseky aktuálne ofarbeného chodníka, utriedené podľa začiatku.

Napríklad pre $K = 20$ budeme po spracovaní prikazov „od 3 po 7 farbou 1“ a „od 4 po 5 farbou 2“ mať päť jednofarebných úsekov: od 0 po 3 farbou 0, od 3 po 4 farbou 1, atď. Pre každý z týchto úsekov by sme mali jeden záznam v strome.

Ako teraz spracujeme novú požiadavku? Začneme tým, že nájdeme v strome posledný úsek, ktorý začína skôr ako ona, a prvý úsek, ktorý neskôr končí. (Toto vieme spraviť v $O(\log N)$, keďže náš strom je vyvážený a má $O(N)$ vrcholov. V `set`e na to slúži napr. metóda `lower_bound`.)

Oba tieto úseky skrátime po hranicu novej požiadavky. (Pozor, mohlo sa stať, že ide o ten istý úsek, ktorý sa nám týmto rozdelil na dva.) Následne zo stromu postupne vyhádžeme všetky úseky medzi nimi – tie naša nová požiadavka celé prekryje – a úplne na záver vložíme do stromu úsek zodpovedajúci novej požiadavke.

Mohlo by sa zdať, že takéto riešenie nemusí byť efektívne – môže sa predsa stať, že tých prekrytých intervalov bude niekedy strašne veľa, nie?

Môže sa to stať, ale nie často. Trik je v tom, že dokopy do stromu vložíme $O(N)$ intervalov, a vyhodiť ich môžeme len toľko, koľko sme ich predtým vložili. Preto dokopy spravíme s naším stromom $O(N)$ operácií. A keďže každú

potrebnú operáciu vie vyvažovaný strom spraviť v čase $O(\log N)$, má aj toto riešenie časovú zložitosť $O(N \log N)$.

Listing programu:

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;

class zaznam {
  public:
    int zac, kon, farba;
    zaznam(int zac=0, int kon=0, int farba=0) : zac(zac),kon(kon),farba(farba) {}
};
bool operator< (const zaznam &a, const zaznam &b) { return a.zac < b.zac; }

int N, F, K;
set<zaznam> S;

int main() {
  cin >> N >> F >> K;
  S.insert(zaznam(-1,K+1,0));
  while (N--) {
    zaznam cur, next;
    cin >> cur.zac >> cur.kon >> cur.farba;

    // spracuje interval obsahujuci lavy koniec poziadavky
    set<zaznam>::iterator left = S.lower_bound(cur);
    left--;
    if (left->kon > cur.kon) S.insert(zaznam(cur.kon,left->kon,left->farba));
    if (left->kon > cur.zac) {
      next = zaznam(left->zac,cur.zac,left->farba);
      S.erase(left);
      S.insert(next);
    }

    // pravy koniec poziadavky
    next = zaznam(cur.kon,K+1,0);
    set<zaznam>::iterator right = S.upper_bound(next);
    right--;
    if (right->zac < cur.kon) {
      S.insert(zaznam(cur.kon,right->kon,right->farba));
      S.erase(*right);
    }

    // vymaze vsetky intervaly medzi nimi
    left = S.lower_bound(cur);
    right = S.upper_bound(next); right--;
    S.erase(left,right);
```

```
    // a vlozi novy
    S.insert(cur);
  }
  vector<int> farby(F+1,0);
  for (set<zaznam>::iterator it=S.begin(); it != S.end(); ++it)
    farby[ it->farba ] += (it->kon)-(it->zac);
  for (int f=1; f<=F; f++) cout << farby[f] << endl;
}
```

Simulácia – druhá možnosť:

(Riešenie podľa Tomáša Belana)

Predchádzajúce riešenie vieme pomocou STL (alebo podobnej sady knižníc v inom jazyku) implementovať ešte jednoduchšie. Namiesto toho, aby sme si pamätali celé intervaly, budeme si pamätať len usporiadanú množinu záznamov tvaru „na pozícii x začína úsek farby f“. Na toto použijeme dátovú štruktúru `map<int,int> zaciatky`, pričom záznam (x, f) uložíme príkazom `zaciatky[x]=f;`.

Čo presne musíme spraviť, keď nám príde nový interval (x, y, f)? V prvom rade zistíme, akej farby bude interval, ktorý bude od tejto chvíle začínať na pozícii y. Toto spravíme tak, že pomocou metódy `upper_bound` nájdeme posledný záznam v mape `zaciatky`, ktorého kľúč je menší alebo rovný ako y. Jeho farba a odteraz bude začínať na pozícii y. Teraz do mapy `zaciatky` uložíme záznamy (x, f) a (y, a). A na záver pomocou metódy `erase` zmažeme všetky záznamy, ktoré už nie sú aktuálne – teda tie s kľúčmi medzi x a y.

Pre každý interval vložíme do mapy dva nové záznamy, a každý záznam z mapy najviac raz vymažeme. Každá z týchto operácií prebehne v čase $O(\log N)$, preto spracovanie všetkých záznamov má časovú zložitosť $O(N \log N)$. Po spracovaní posledného záznamu už len v lineárnom čase prejdeme cez všetky záznamy, ktoré máme na konci v mape, a spočítame odpoveď. Výsledná implementácia je až neuveriteľne stručná:

Listing programu:

```
#include <iostream>
#include <vector>
#include <map>
using namespace std;

int N, F, K, zac, kon, farba;
map<int, int> zaciatky;
vector<int> farby;
```

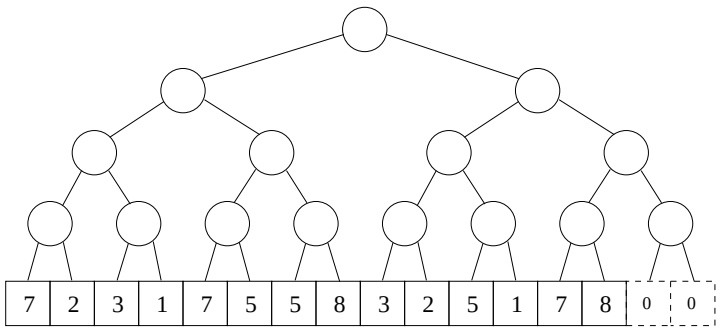
```
int main() {
  cin >> N >> F >> K;
  zaciatky[0] = zaciatky[K] = 0;
  while (N--) {
    cin >> zac >> kon >> farba;
    int after = (--zaciatky.upper_bound(kon))->second;
    zaciatky[zac] = farba;
    zaciatky[kon] = after;
    zaciatky.erase( zaciatky.upper_bound(zac), zaciatky.lower_bound(kon) );
  }

  farby.resize(F+1,0);
  map<int,int>::iterator it = zaciatky.begin();
  while (it->first < K) {
    farba = it->second, zac = it->first, kon = (++it)->first;
    farby[farba] += kon-zac;
  }
  for (int f=1; f<=F; f++) cout << farby[f] << endl;
}
```

Simulácia – tretia možnosť:

Na záver si ukážeme dátovú štruktúru, ktorá sa dá rozumne ľahko implementovať aj v Pascale a umožní nám robiť simuláciu rovnako efektívne.

Začneme tým, že rovnako ako v 6-bodovom riešení spravíme kompresiu súradníc, čím rozdelíme chodník na U úsekov. Pre jednoduchosť budeme predpokladať, že U je mocnina dvoch. (Ak by nebolo, zväčšíme ho na najbližšiu mocninu dvoch. Uvedomte si, že tým sa zväčší menej ako na dvojnásobok pôvodnej hodnoty, a teda naďalej $U = O(N)$.)

Použijeme dátovú štruktúru známu pod menom *intervalový strom*. Ten bude vyzerať takto:

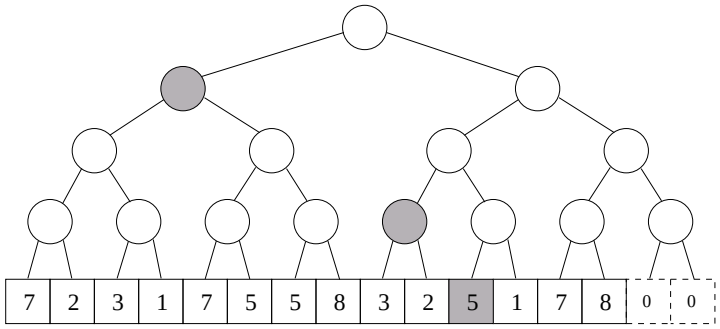


Listy intervalového stromu zodpovedajú jednotlivým úsekom chodníka a budeme si v nich samozrejme ukladať ich farby. Všimnite si, že vnútorný vrchol,

ktorý je k úrovní nad listami, zodpovedá intervalu obsahujúcemu 2^k po sebe idúcich úsekov. Tie intervaly, ktoré zodpovedajú vrcholom nášho stromu, budeme volať *jednoduché*.

Načo je intervalový strom dobrý? Ukážeme, že ľubovoľný interval úsekov vieme šikovne „poskladať“ z jednoduchých intervalov.

Zaoberajme sa najskôr intervalom, ktorý obsahuje úseky od 1 po k. Tvrdíme, že tento vieme zložiť z najviac $\lg N$ jednoduchých intervalov. Toto dokážeme tak, že budeme z jeho ľavej strany odkrajovať čo najväčšie jednoduché intervaly, až kým sa neminie. Najjednoduchšie je to vidieť na príklade. Napr. interval „od 1 po 11“ vieme rozdeliť na „od 1 po 8“, „od 9 po 10“ a „od 11 po 11“.

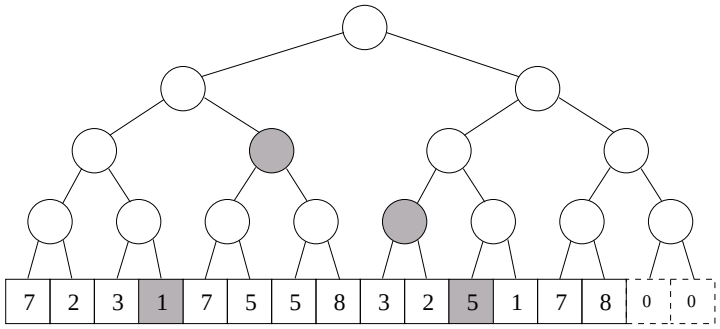


Vyznačené vrcholy zodpovedajú jednoduchým intervalom, ktoré dokopy tvoria interval „od 1 po 11“.

Odhad počtu použitých intervalov vyplýva napríklad z toho, že v každom kroku odkrojíme viac ako polovicu intervalu.

Všimnite si, že postup krájania zodpovedá schádzaniu z koreňa dole po intervalovom strome. V každom vrchole sa pozrieme, či nerozkrájaná časť zadaného intervalu leží celá v ľavom podstrome. Ak áno, nič nekrájame a zlezieme doň. Ak nie, odkrojíme interval zodpovedajúci ľavému podstromu a zlezieme do pravého.

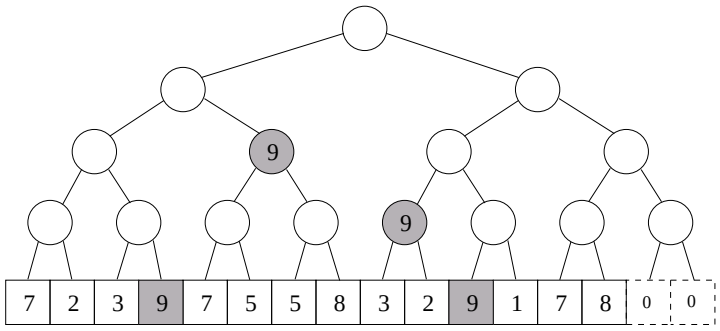
Vo všeobecnej situácii, keď chceme naskladať interval úsekov od k po l, budeme na tom podobne, vystačíme si s $2 \lg N$ jednoduchými intervalmi. Dôkaz je podobný, pôjdeme dole intervalovým stromom. Akonáhle niekedy zistíme, že zadaný interval zasahuje do oboch podstromov, rozkrojíme ho na dve časti. No a každá z častí už teraz zodpovedá jednoduchšiemu prípadu, ktorý sme rozobrali vyššie.



Všeobecný prípad: interval „od 4 po 11“.

Ukázali sme teda, že v čase $O(\log N)$ vieme ľubovoľný interval rozdeliť na niekoľko častí, ktoré zodpovedajú vrcholom stromu.

Načo to bude dobré? Vnútorné vrcholy stromu použijeme na to, aby sme vedeli rýchlo simulovať Bonifácove požiadavky. Dohodneme sa, že ak je vo vnútornom vrchole stromu iná hodnota ako nula, znamená to, že celý zodpovedajúci interval má príslušnú farbu – bez ohľadu na to, aké hodnoty sú uložené vo vrcholoch v tomto podstrome. Toto nám umožní spracovať každú požiadavku v čase $O(\log N)$: nájdeme v strome vrcholy zodpovedajúce dotyčnému intervalu a zaznačíme si do nich príslušnú farbu.

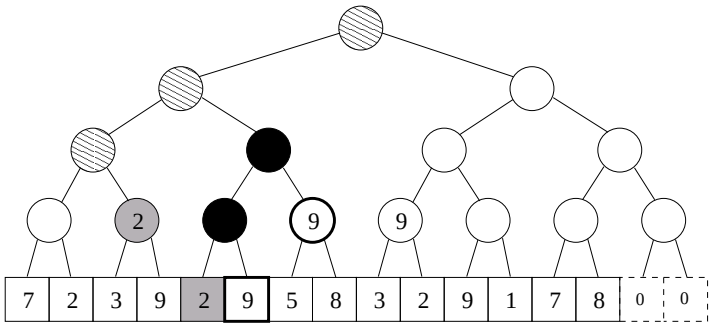


Stav po prefarbení intervalu „od 4 po 11“ na farbu 9. V prázdnych vrcholoch sú nuly.

Potrebujeme si ešte rozmyslieť, čo presne sa stane, ak počas spracúvania požiadavky prechádzame v strome cez vrchol, ktorý je momentálne celý zafarbený. To, že cez tento vrchol prechádzame, znamená, že časť jeho podstromu ideme prefarbiť. Preto odteraz bude v tomto vrchole nula – už nebude jednofarebný. Namiesto toho (skôr, ako sa pohneme hlbšie) zafarbíme jeho farbou oba vrcholy pod ním.

Na nasledujúcom obrázku je znázornené, ako sa zmenia údaje v strome z predchádzajúceho obrázku po spracovaní požiadavky „prefarbi interval od 3 po 5 na farbu 2“. Šedé pozadie majú, rovnako ako na predchádzajúcich obrázkoch,

vrcholy zodpovedajúce aktuálnej požiadavke. Šrafované vrcholy sú tie, ktoré počas spracúvania požiadavky navštívime, ale nič sa v nich neudeje. Zaujímavé veci sa začnú diať, ked dorazíme k hornému čiernemu vrcholu, v ktorom bola doteraz hodnota 9. Tú presunieme do jeho synov, a následne to isté ešte raz zopakujeme v druhom čiernom vrchole. Oba čierne vrcholy teda teraz obsahujú nuly. Hodnota 9 sa presunula z horného čierneho vrchola do dvoch hrubo orámovaných vrcholov. Všimnite si, že tieto presne zodpovedajú tej časti pôvodného intervalu, ktorú sme neprefarbili.



Kompresiu súradníc vieme spraviť v čase $O(N \log N)$, napríklad pomocou triedenia a binárneho vyhľadávania. (Tak to robíme v nižšie uvedenom programe.) Úvodné nastavenie hodnôt v strome spravíme v čase $O(N)$. Každý z N príkazov spracujeme v čase $O(\log N)$, a na záver v čase $O(N)$ prejdeme strom a spočítame výsledok. Celková časová zložitosť tohto riešenia je teda $O(N \log N)$ a pamäťová zložitosť je $O(N)$.

(Náš intervalový strom má menej ako $4N$ listov, a každá vyššia vrstva má polovičný počet vrcholov oproti tej bezprostredne pod ňou. Preto náš intervalový strom obsahuje nanajvýš $8N$ záznamov, a teda je pamäťová zložitosť skutočne lineárna.)

Listing programu:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int N, F, K, D, L;
struct poziadavka { int zac, kon, farba; };
struct vrchol { int farba, dlzka; };
vector<poziadavka> P; // pole s poziadavkami
vector<vrchol> T; // intervalovy strom
vector<int> farby;
```

```
// nacitanie vstupu
void load() {
  cin >> N >> F >> K;
  poziadavka tmp;
  for (int i=0; i<N; i++) {
    cin >> tmp.zac >> tmp.kon >> tmp.farba;
    P.push_back(tmp);
  }
}

// najdenie indexu prvku x v utriedenom poli
int najdi(int x, const vector<int> &pole) {
  int lo=0, hi=pole.size();
  while (hi-lo > 1) { int med=(hi+lo)/2; if (pole[med]<=x) lo=med; else hi=med; }
  return lo;
}

void build_tree() {
  // utriedime vsetky hranice poziadaviek
  vector<int> hranice;
  hranice.push_back(0); hranice.push_back(K);
  for (int i=0; i<N; i++) {
    hranice.push_back( P[i].zac );
    hranice.push_back( P[i].kon );
  }
  sort( hranice.begin(), hranice.end() );
  // vyhadzeme duplikaty
  D=1;
  for (int i=1; i<2*N+2; i++)
    if (hranice[i]!=hranice[i-1]) swap(hranice[i],hranice[D++]);
  hranice.resize(D);
  D--;
  // vyrobime strom
  for (int i=1; ; i*=2) if (i>D+7) { L=i; break; }
  T.resize(2*L);
  for (int i=0; i<D; i++) T[L+i].dlzka = hranice[i+1]-hranice[i];
  for (int i=L-1; i>=1; i--) T[i].dlzka = T[2*i].dlzka + T[2*i+1].dlzka;
  // prepiseme zaciatky a konce poziadaviek do noveho cislovania
  for (int i=0; i<N; i++) {
    P[i].zac = najdi( P[i].zac, hranice );
    P[i].kon = najdi( P[i].kon, hranice );
  }
}

void color(int x, int kde=1, int left=0, int length=L) {
  if (P[x].kon <= left || P[x].zac>=left+length) return; // mimo
  if (P[x].zac <= left && left+length <= P[x].kon) { // dnu
    T[kde].farba=P[x].farba; return;
  }
  if (kde<L && T[kde].farba!=0) {
```

```
    T[2*kde].farba=T[2*kde+1].farba=T[kde].farba;
    T[kde].farba=0;
  }
  color(x,2*kde,left,length/2);
  color(x,2*kde+1,left+length/2,length/2);
}

void evaluate(int kde) {
  if (kde>=L || T[kde].farba>0) farby[ T[kde].farba ] += T[kde].dlzka;
  else { evaluate(2*kde); evaluate(2*kde+1); }
}

int main() {
  load();
  build_tree();
  for (int i=0; i<N; i++) color(i);
  farby.resize(F+1,0);
  evaluate(1);
  for (int f=1; f<=F; f++) cout << farby[f] << endl;
}
```

A-I-2 Čokoláda

Najskôr ukážeme riešenie s časovou zložitosťou $O(R^2S)$. Bude založené na jednoduchej myšlienke: vyskúšame všetky dvojice riadkov, a pre každú dvojicu v $O(S)$ spočítame všetky štvorce, ktoré práve tam majú svoj horný a dolný riadok.

Keď sme si už zvolili horný a dolný riadok, máme pás políčok. Niektoré jeho stĺpce sú celé, tie môžeme použiť. A niektoré obsahujú aspoň jedno chýbajúce políčko, a tie použiť nemôžeme.

Ak by sme vedeli, ktoré stĺpce použiť môžeme, a ktoré nie, dostávame nasledujúcu úlohu: Máme dané číslo K (dĺžku strany štvorca) a pole $A[1..S]$ obsahujúce len nuly a jednotky (zlé a dobré stĺpce). Koľko existuje v poli A úsekov dĺžky K, ktoré obsahujú samé jednotky?

Túto úlohu vieme ľahko vyriešiť v lineárnom čase od dĺžky poľa A. Existuje viacero spôsobov, ukážeme jeden z nich. Všimneme si, že úsek je dobrý práve vtedy, ak je jeho súčet rovný K. Spravíme si nové pole $B[0..S]$, kde $B[0] = 0$ a pre všetky $i > 0$ je $B[i] = A[i] + B[i-1]$. Zjavne platí, že $B[i]$ je súčet prvých i prvkov poľa A. (Hodnoty v poli B voláme *prefixovými sumami* poľa A.)

Potom ale úsek, ktorý začína na pozícii p, je dobrý práve vtedy, ak $B[p+K-1] - B[p-1] = K$. (Rozmyslite si, že hodnota $B[p+K-1] - B[p-1]$ predstavuje súčet prvkov poľa A na pozíciách p až $p+K-1$.)

Pole B vieme spočítať v čase $O(S)$, a následne vieme o každom z $S-K+1$ úsekov dĺžky K v konštantnom čase zistiť jeho súčet, a podľa toho povedať, či je tento úsek dobrý alebo nie. Celkovo teda našu jednorozmernú úlohu vieme riešiť v čase $O(S)$.

Zostáva doriešiť, odkiaľ vezmeme informáciu o tom, ktoré stĺpce môžeme použiť a ktoré nie. Na to nám stačí spracúvať dvojice riadkov v systematickom poradí. Pre dvojicu (r_1, r_1) túto informáciu máme „zadarmo“ priamo vo vstupe. A keď pre dvojicu (r_1, r_2) vieme, ktoré stĺpce sa ešte dajú použiť, pre dvojicu (r_1, r_2+1) túto informáciu ľahko zistíme – sú to tie stĺpce, ktoré sa dali použiť pre (r_1, r_2), a zároveň majú jednotku aj v riadku r_2+1.

Listing programu:

```
#include <cstdio>
using namespace std;

int R, S;
int A[5012][5012];
int zije[5012], sucet[5012];

int main() {
  scanf("%d %d ",&R,&S);
  for (int r=0; r<R; r++) for (int s=0; s<S; s++) scanf("%d",&A[r][s+1]);
  long long result = 0;
  for (int r1=0; r1<R; r1++) {
    for (int s=1; s<=S; s++) zije[s]=1;
    for (int r2=r1; r2<R; r2++) {
      for (int s=1; s<=S; s++) zije[s] &= A[r2][s];
      sucet[0]=0;
      for (int s=1; s<=S; s++) sucet[s]=sucet[s-1]+zije[s];
      for (int d=r2-r1+1, zac=1; zac+d-1<=S; zac++)
        if (sucet[zac+d-1]-sucet[zac-1] == d)
          result++;
    }
  }
  printf("%Ld\n",result);
  return 0;
}
```

Lepšie riešenie:

Podobný trik ako sme robili pri jednorozmernej úlohe v predchádzajúcom riešení vieme spraviť aj v dvojrozmernom prípade. Začneme teda tým, že pre každý riadok aj pre každý stĺpec zadanej matice si predpočítame prefixové sumy. Vďaka nim vieme o ľubovoľnom kuse riadku alebo stĺpca v konštantnom čase povedať, či je celý dobrý. Na toto predpočítanie nám stačí čas $O(RS)$.

Teraz postupne pre každé políčko zodpovieme otázku: „Aký najväčší štvorec má pravý dolný roh na tomto políčku?“

Ak je na danom políčku 0, odpoveď je zjavne 0, inak je odpoveď aspoň 1. Budeme teraz postupne zväčšovať veľkosť strany štvorca, až kým „nenarazíme“ – nezistíme, že už náš štvorec obsahuje nejakú dieru, prípadne prekročil hranice pôvodného obdĺžnika.

Predstavme si, že skúmame pravý dolný roh na súradniciach (r, s), a už vieme, že tam má pravý dolný roh štvorec veľkosti K. Ako zistiť, či tam máme aj štvorec veľkosti $K+1$? Jednoducho – je tam práve vtedy, ak sú v stĺpci $s-K$ dobré všetky políčka od $r-K$ po r, a zároveň v riadku $r-K$ musia byť dobré všetky políčka od $s-K$ po s. Obe veci vieme overiť v konštantnom čase vďaka predpočítaným prefixovým sumám.

Takto dostávame riešenie, ktorého časová zložitosť je $O(RS + X)$, kde X je počet štvorcov vo vstupe. V najhoršom prípade bude toto riešenie rovnako rýchle ako to predchádzajúce, ale na „riedkych“ vstupoch (s veľa dierami) bude výrazne rýchlejšie.

Vzorové riešenie:

Vidíme, že ak chceme lepšiu časovú zložitosť, nesmieme štvorce počítať po jednom. Použijeme podobný prístup ako v predchádzajúcom riešení. Nech $K(r, s)$ je veľkosť najväčšieho plného štvorca, ktorý má pravý dolný roh na políčku (r, s). Potom hľadanú odpoveď získame jednoducho ako súčet hodnôt $K(r, s)$ pre všetky prípustné r a s.

Ukážeme teraz, ako hodnoty $K(r, s)$ šikovne spočítať. Ak je na políčku (r, s) diera, tak zjavne $K(r, s) = 0$. Predpokladajme teda, že na (r, s) diera nie je.

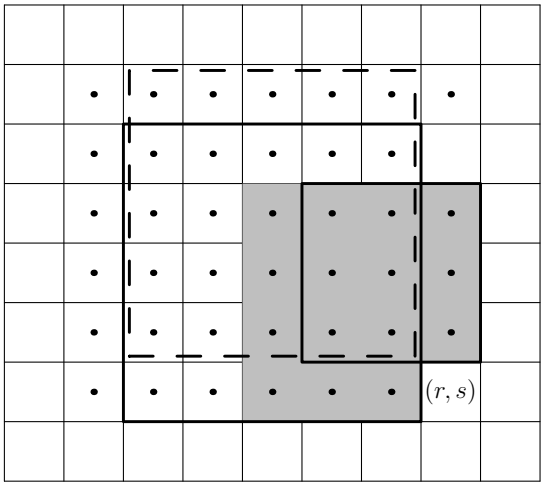
V prvom rade si uvedomme, že platia nasledujúce nerovnosti:

- $K(r, s) \leq K(r-1, s) + 1$
- $K(r, s) \leq K(r, s-1) + 1$
- $K(r, s) \leq K(r-1, s-1) + 1$

Totiž ak má na (r, s) pravý dolný roh štvorec veľkosti x, tak keď zoberieme štvorec veľkosti $x-1$ s pravým dolným rohom na niektorom z políčok $(r-1, s)$, $(r, s-1)$, alebo $(r-1, s-1)$, tak tento menší štvorec bude celý vo vnútri v tom pôvodnom – a teda bude určite dobrý. Dostávame teda, že platí:
$K(r, s) \quad \leq \quad 1 + \min(K(r-1, s), K(r, s-1), K(r-1, s-1))$.

Dokážeme teraz, že v skutočnosti v predchádzajúcom vzťahu vždy platí rovnosť. Nech totiž $\min(K(r-1, s), K(r, s-1), K(r-1, s-1)) = x$. Keď zoberieme

zjednotenie štvorcov, ktoré majú veľkosť x a pravý dolný roh na týchto troch políčkach, dostaneme presne štvorec veľkosti $x+1$ s rohom na (r,s) – s jedinou výnimkou, ktorou je samotné políčko (r,s). O tom sme ale predpokladali, že je dobré, a teda naozaj máme štvorec veľkosti $x+1$.



Na obrázku je príklad situácie z dôkazu. Bodkami sú vyznačené dobré políčka. Máme $K(r-1,s)=3$ a $K(r,s-1)=K(r-1,s-1)=5$. Prvé dva zodpovedajúce štvorce sú vyznačené hrubou čiarou, tretí čiarkovanou. Dostávame $x=\min(3,5,5)=3$. Zjednotenie príslušných troch štvorcov 3×3 je vyplnené. Všimnite si, že spolu s políčkom (r,s) dostávame štvorec 4×4.

Každú hodnotu $K(r,s)$ teda vieme spočítať v konštantnom čase z predchádzajúcich hodnôt. Časová zložitosť tohto riešenia je teda $O(RS)$. Za zmienku ešte stojí, že pamäťová zložitosť sa dá zredukovať na $O(R)$, keďže nám vždy stačí pamätať si hodnoty K pre predchádzajúci a aktuálny riadok.

Listing programu:

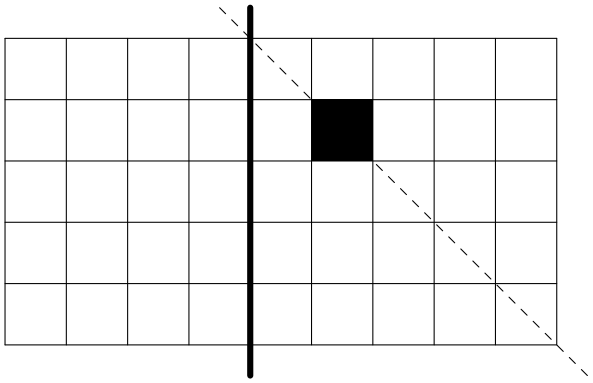
```
#include <algorithm>
#include <cstdio>
using namespace std;

int R, S, A;
long long result = 0;
int K[2][5012];

int main() {
  scanf("%d%d",&R,&S);
  for (int r=1, cur=0; r<=R; r++, cur=1-cur)
    for (int s=1; s<=S; s++) {
      scanf("%d",&A);
      if (A==0) K[cur][s]=0;
```

```
      else K[cur][s] = 1 + min(K[cur][s-1],min(K[1-cur][s],K[1-cur][s-1]));
      result += K[cur][s];
    }
  printf("%Ld\n",result);
  return 0;
}
```

A-I-3 Koláč

Podúloha a):

Hru vyhrá Marienka. V prvom ťahu spraví rez po priamke $x = 4$ (na obrázku hrubou čiarou). Takto vyrobí štvorec 5×5, ktorý je, vrátane polohy ropuchy, symetrický vzhľadom na uhlopriečku (na obrázku čiarkovane).



Od tohto okamihu bude Marienka ťahať symetricky s Jankom – nech ten spraví rez, aký chce, Marienka spraví jeho zrkadlový obraz (podľa čiarkovanej osi). Teda ak Janko reže vodorovne, tak Marienka zvisle a naopak. Marienka zjavne vždy bude môcť spraviť svoj ťah, a vždy po jej ťahu zostane koláč symetrický. Preto nutne tým, kto už nebude môcť rezať, bude Janko.

Podúloha b):

Riešenie začneme tým, že vysvetlíme základné pojmy z teórie kombinatorických hier. Stav hry, teda aktuálne rozmery obdĺžnika a polohu ropuchy na ňom, budeme volať *pozícia*. *Vyhrávajúca stratégia* je postup, ktorý nám zaručí, že hru vyhráme, bez ohľadu na to, ako bude ťahať protihráč. Pozícia je *vyhrávajúca*, ak hráč, ktorý je práve na ťahu, má vyhrávajúcu stratégiu. Ostatné pozície voláme *prehrávajúce*.

Prezeranie stromu hry:

Najjednoduchšie riešenie, ktoré sa dá použiť pre ľubovoľnú konečnú kombinatorickú hru, je založené na dvoch jednoduchých myšlienkach:

- Ak z danej pozície všetky ťahy vedú do vyhrávajúcich pozícií, tak je táto pozícia prehrávajúca.
- Ak z danej pozície existuje ťah do prehrávajúcej, tak je táto pozícia vyhrávajúca.

(Ak všetky ťahy vedú do vyhrávajúcich pozícií, nech si vyberieme ktorýkoľvek, vždy tým dostaneme súpera do vyhrávajúcej pozície. A ak sa potom bude súper držať nejakej vyhrávajúcej stratégie, hru prehráme. Preto takáto pozícia je prehrávajúca. Naopak, ak existuje ťah do prehrávajúcej pozície, spravíme ho, a tým dostaneme súpera do tejto, pre neho prehrávajúcej pozície.)

Túto myšlienku ľahko prepíšeme do rekurzívnej funkcie, ktorá nám o pozícii povie, či je vyhrávajúca alebo prehrávajúca.

Dynamické programovanie / memoizácia:

Problém predchádzajúceho prístupu spočíva v tom, že je príliš pomalý. Hlavný dôvod je ten, že pri rekurzívnych volaniach vlastne skúša všetky možné priebehy hry, a pri tom mnohé pozície vyhodnotí veľa krát.

Tu je samozrejme ľahká pomoc – akonáhle o nejakej pozícii zistíme, či je vyhrávajúca, zapíšeme si to do pomocného poľa. Takto dosiahneme to, že každú pozíciu budeme spracúvať práve raz.

Koľko je rôznych pozícii, do ktorých sa hra môže dostať? Každá pozícia je určená štyrmi súradnicami: ľavý, pravý, horný a dolný okraj aktuálneho obdĺžnika. Ľavý okraj musí ležať medzi 0 a r_x, vrátane, pravý medzi $r_x + 1$ a X, analogicky pre zvyšné dva.

Dokopy máme teda $(r_x + 1)(X - r_x)(r_y + 1)(Y - r_y)$ pozícii, čo je $O(X^2Y^2)$. Taká bude aj pamäťová zložitosť nášho riešenia. Možných ťahov je v každej pozícii $O(X + Y)$, takže časová zložitosť tohto riešenia je $O(X^2Y^2(X + Y))$.

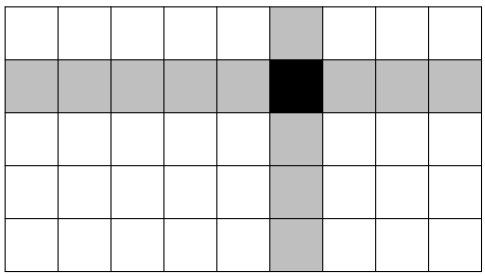
Na toto riešenie sa môžeme dívať aj z opačnej strany: Keby sme napríklad pozície spracúvali zoradené podľa plochy, tak by platilo, že v okamihu, keď vyhodnocujeme nejakú pozíciu, už vieme o všetkých pozíciách, do ktorých môžeme ťahať, či sú vyhrávajúce alebo prehrávajúce. Takto implementované riešenie má rovnakú časovú aj pamäťovú zložitosť ako to predchádzajúce.

Listing programu:

```
#include <iostream>
#include <cstring>
using namespace std;

int X,Y,rx,ry;
int memo[50][50][50][50];

int winning(int x1, int y1, int x2, int y2) {
  // ak uz sme tuto poziciu riesili, rovno vrat hodnotu
  if (memo[x1][y1][x2][y2] >= 0) return memo[x1][y1][x2][y2];
  // ak nie, inicializuj ju na prehravajucu
  memo[x1][y1][x2][y2]=0;
  // a ak najdes tah do prehravajucej, zmen sucasnu na vyhravajucu
  for (int x=x1+1; x<x2; x++) {
    int nx1, nx2;
    if (x<=rx) nx1=x, nx2=x2; else nx1=x1, nx2=x;
    if (!winning(nx1,y1,nx2,y2)) return memo[x1][y1][x2][y2]=1;
  }
  for (int y=y1+1; y<y2; y++) {
    int ny1, ny2;
    if (y<=ry) ny1=y, ny2=y2; else ny1=y1, ny2=y;
    if (!winning(x1,ny1,x2,ny2)) return memo[x1][y1][x2][y2]=1;
  }
  return 0;
}

int main() {
  cin >> X >> Y >> rx >> ry;
  memset(memo,-1,sizeof(memo));
  cout << (winning(0,0,X,Y) ? "Marienka" : "Janko") << " vyhra." << endl;
}
```

Optimálne riešenie:

Všimnime si štyri pásiky políčok, ktoré sú na nasledujúcom obrázku šedé. Bez ohľadu na to, ako budeme rezať, vždy skrátime práve jeden z nich. A naopak, ak si povieme, ktorý pásik a o koľko chceme skrátiť, vždy vieme taký rez urobiť.[2] Každú pozíciu môžeme teda popísať dĺžkami týchto štyroch pásikov (a, b, c, d).



```
1 =      1
3 =     11
5 =    101
3 =     11
```

[2]Pre tých, čo už sú v kombinatorickej teórii hier doma: Týmto sme práve dokázali, že naša hra je izomorfná so 4-kopovým NIMom, pričom dĺžky pásikov zodpovedajú veľkostiam kôp.

Dokážeme teraz nasledujúce tvrdenie: Majme pozíciu (a, b, c, d). Zoberme čísla a, b, c, d, preveďme ich do dvojkovej sústavy a zapíšme pod seba. Tvrdíme, že pozícia je prehrávajúca práve vtedy, ak je v každom stĺpci párny počet jednotiek.[3]

Ak toto chceme dokázať, potrebujeme ukázať, že platia obe vlastnosti, ktoré majú prehrávajúce pozície mať. Presnejšie, musíme ukázať, že:

- Z pozície, kde je v každom stĺpci párny počet jednotiek, každý ťah vedie do pozície, kde to neplatí.
- V každej inej pozícii existuje ťah, po ktorom bude v každom stĺpci párny počet jednotiek.

Prvé tvrdenie očividne platí. Ak spravíme ťah, zmeníme tým práve jedno z čísel. Keď sa teraz pozrieme na binárne zápisy našich čísel, vidíme, že sa nám zmenil práve jeden riadok. Vyberme si niektorý bit v ňom, ktorý sa zmenil. Potom v jeho stĺpci sa nutne zmenila parita počtu jednotiek, a teda je ich tam teraz nepárny počet.

Druhé tvrdenie dokážeme nasledovne: Všimnime si najľavejší stĺpec, kde je nepárny počet jednotiek. Vyberme si ľubovoľný riadok, ktorý má v tomto stĺpci jednotku. Tú zmeníme na nulu. Následne zmažeme zvyšok tohto riadku a znova ho dopočítame tak, aby v každom stĺpci bol párny počet jednotiek. Keďže najľavejší bit, ktorý sme menili, sme zmenili z jednotky na nulu, bez ohľadu na to, ako sme menili ostatné bity, hodnotu čísla sme zmenšili, a teda ide o platný ťah.

Ukážeme si to celé na príklade. Majme pozíciu $(58, 9, 43, 22)$. Keď si ju zapíšeme v dvojkovej sústave, zistíme, že v 4., 3. a 2. stĺpci sprava je nepárny počet jednotiek. Vyberieme si číslo 43, ktoré má v stĺpci 4 jednotku, a túto zmeníme na nulu. Následne dopočítame hodnoty ostatných bitov (tieto bity sú v strednom stĺpci dočasne nahradené bodkami).

```
       v
58 = 111010        111010        111010 = 58
 9 =   1001          1001          1001 =  9
43 = 101011        100...        100101 = 37
22 =  10110         10110         10110 = 22
```

Ak teda v začiatočnej pozícii našej hry platí, že v niektorom stĺpci binárnych zápisov dĺžok pásikov je nepárny počet jednotiek, hru vyhrá Marienka. Stačí jej

[3] Inými slovami, vtedy, keď $a \oplus b \oplus c \oplus d = 0$, kde $\oplus$ je bitový xor.

vždy nájsť a spraviť taký ťah, po ktorom bude všade počet jednotiek párny. Už sme ukázali, že taký ťah vždy existuje. A naopak Janko bude vždy na ťahu v pozícii, v ktorej je všade počet jednotiek párny, a nech ťahá, ako chce, vždy toto poruší.

Keďže v koncovej pozícii sú všetky dĺžky pásikov rovné nule, znamená to, že v nej je všade počet jednotiek párny, a teda hráčom na ťahu (ktorý práve prehral) je Janko.

Naopak, ak je pre začiatočnú pozíciu v každom stĺpci binárnych zápisov dĺžok pásikov párny počet jednotiek, hru vyhrá Janko – po tom, ako Marienka spraví prvý ťah, bude Janko v pozícii s nepárnym počtom jednotiek v niektorom stĺpci a môže použiť tú istú stratégiu ako v opačnom prípade mala Marienka.

Práve sme teda ukázali riešenie, ktoré dokáže o ľubovoľnej pozícii v konštantnom čase povedať, či je vyhrávajúca. (Rozmyslite si, že dokonca vieme pre vyhrávajúcu pozíciu nájsť v konštantnom čase všetky možné vyhrávajúce ťahy.)

A-I-4 Počítač s gumenou rúrou

Podúloha a):

Ak celé číslo N nie je prvočíslo, tak má nejakého deliteľa d takého, že $1 < d < N$. A nie len to. Číslo $e = N/d$ je celé, a tiež je deliteľom N (lebo $N/e = d$, pochopiteľne). Navyše tiež platí, že $1 < e < N$. Ak by aj d, aj e bolo viac ako $\sqrt{N}$, tak máme $N = d \cdot e > \sqrt{N} \cdot \sqrt{N} = N$, čo je spor. Preto je jedno z nich nanajvýš rovné $\sqrt{N}$.

Práve sme teda dokázali tvrdenie: Ak celé číslo N nie je prvočíslo, tak má deliteľa d, pre ktorého platí $1 < d \leq \sqrt{N}$.

Budeme teda postupne skúšať všetky hodnoty d z tohto rozsahu. Ak nájdeme nejakú, ktorá delí N, vypíšeme, že nie je prvočíslo. Ak ani jedna z nich N nedelí, je to prvočíslo. Na začiatku ešte samozrejme ošetríme špeciálne prípady $N = 0$ a $N = 1$.

```
get n
put 1 ; get j
jz n bad
jeq n j bad
put 2 ; get d

label cyklus
```

```
  put n ; put d ; div ; get e
  jgt d e good
  put n ; put d ; mod ; get e
  jz e bad
  put d ; put 1 ; add ; get d
jump cyklus

label good ; put 1 ; print ; stop
label bad ; put 0 ; print ; stop
```

V cykle vždy najskôr porovnáme hodnoty $\lfloor N/d \rfloor$ a d. Ak už je prvá menšia, znamená to, že určite $d > \sqrt{N}$ a môžeme prestať skúšať. Následne spočítame hodnotu N mod d a ak je 0, tak d delí N, a tiež môžeme prestať, len s opačným výsledkom. V opačnom prípade zvýšime d a ideme na ďalšiu iteráciu cyklu.

Najhorším vstupom pre tento program je samozrejme veľké prvočíslo. Najväčšie prvočíslo, ktoré môžeme na vstupe dostať, je 65 521. Preň tento program spraví niečo vyše 4 000 krokov.

Podúloha b):

Najjednoduchšie riešenie je zmeniť každé číslo v rúre na 1, a následne použiť program z Príkladu 2 v študijnom texte, ktorý tieto jednotky sčíta a vypíše ich súčet – ktorý je zároveň ich počtom.

Prvý krok spravíme tak, že využijeme, že čísla v rúre sú kladné. Vložíme si teda na koniec rúry nulu. Potom v cykle opakujeme: prečítame číslo z rúry. Ak to nie je 0, namiesto neho vložíme do rúry 1. Ak to 0 je, už sme spracovali všetky čísla, v rúre máme namiesto každého z nich jednotku, a môžeme ísť sčítať.

(Drobný detail: aby náš program fungoval aj ak mal na začiatku rúru prázdnu, vložíme teraz do rúry ešte jednu nulu. Tá nám súčet nezmení, a rúra prestane byť prázdna.)

Celý program môže vyzerať napríklad takto:

```
put 0
label prepisuj
  get a ; jz a dalej ; put 1
jump prepisuj
label dalej
put 0
label cyklus
  get a ; jempty koniec ; put a ; add
jump cyklus
```

```
label koniec
put a ; print
```

Časová zložitosť je lineárna od počtu čísel, ktoré sú na začiatku v rúre, lebo najskôr raz prejdeme celú rúru, a potom použijeme lineárny program na zistenie súčtu.

Podúloha c):

Aj táto úloha je riešiteľná. Porovnávať hodnoty vieme, stačí ich načítať do rôznych registrov. Problém, ktorý potrebujeme vyriešiť, je, že akonáhle načítame hodnotu do registra, nevieme ju už presunúť do iného. Potrebovali by sme si teda pamätať, kde máme doteraz nájdené maximum. Trik je v tom, že toto si pamätať vieme – pozíciou v programe. Na začiatku máme maximum v registri `a` a nové čísla načítavame do `b`. Ak niekedy do `b` načítame väčšiu hodnotu ako tú, čo máme v `a`, skočíme do inej časti programu, kde sa správame opačne – maximum máme v `b` a nové čísla načítavame do `a`. A ak sa v `a` opäť vyskytne väčšie číslo ako je teraz v `b`, skočíme späť do prvej časti programu. A tak dokola, až kým nedočítame celú postupnosť.

```
get a

label maximum_je_v_a
  jempty koniec_a
  get b ; jgt b a maximum_je_v_b
jump maximum_je_v_a

label maximum_je_v_b
  jempty koniec_b
  get a ; jgt a b maximum_je_v_a
jump maximum_je_v_b

label koniec_a ; put a ; print ; stop
label koniec_b ; put b ; print ; stop
```

Riešenia domáceho kola kategórie B

B-I-1 Majstrovstvá na rybníku

Pre každého súťažiaceho vieme jeho výsledné skóre zistiť rovno pri načítaní vstupu. Najjednoduchšie je spraviť si pomocné pole veľkosti R, do ktorého načítame všetky známky daného súťažiaceho. Potom pre toto pole postupne zistíme súčet, maximum aj minimum a z nich už ľahko určíme výsledné skóre.

(Za zmienku stojí, že sa zaobídeme aj bez takéhoto pomocného poľa. Vystačíme si s tromi premennými. V dvoch z nich si budeme pamätať doteraz najmenšiu a doteraz najväčšiu známku, ktorú aktuálny súťažiaci dostal, a v tretej bude súčet ostatných už spracovaných známok.)

Po dočítaní vstupu teda máme S usporiadaných dvojíc (skóre, meno), ktoré máme utriediť.

V takejto situácii je väčšinou dobré (kvôli prehľadnosti programu) „zabaliť“ si údaje, ktoré ideme triediť, do vhodného dátového typu – napr. `record` v Pascale alebo `struct` v C/C++.

Následne na utriedenie týchto záznamov použijeme nejaký štandardný triediaci algoritmus. V C máme k dispozícii funkciu `qsort`, v C++ funkciu `sort`. Obe tieto funkcie implementujú dobré algoritmy na triedenie – utriedia N prvkov v čase $O(N \log N)$. V Pascale bolo potrebné naprogramovať si vlastné triedenie. V tomto riešení si popíšeme HeapSort – triedenie pomocou haldy.

Listing programu:

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <string>
using namespace std;

struct sutaziaci { string meno; int body; };

bool operator< (const sutaziaci &A, const sutaziaci &B) {
  if (A.body != B.body) return A.body > B.body; else return A.meno < B.meno;
}

int S, R, lo, hi, cur;

int main() {
```

```
  cin >> S >> R;
  vector<sutaziaci> vysledky(S);
  for (int s=0; s<S; ++s) {
    vysledky[s].body = 0;
    cin >> vysledky[s].meno >> lo >> hi;
    if (lo > hi) swap(lo,hi);
    for (int r=0; r<R-2; ++r) {
      cin >> cur;
      if (cur < lo) swap(lo,cur);
      if (cur > hi) swap(hi,cur);
      vysledky[s].body += cur;
    }
  }
  sort( vysledky.begin(), vysledky.end() );
  for (int s=0; s<S; ++s)
    cout << vysledky[s].body << ": " << vysledky[s].meno << endl;
}
```

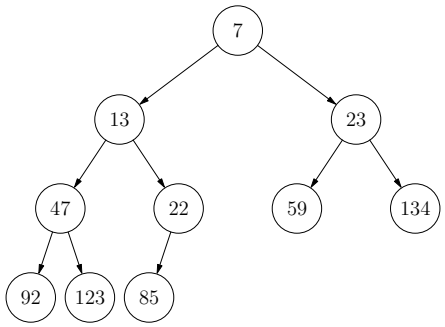
Halda:

Predstavme si, že máme čiernu krabičku s tlačidlom, do ktorej vieme vhadzovať čísla a z ktorej po každom stlačení tlačidla vypadne najmenšie z čísel, ktoré sú práve v nej.

Takáto krabička vie byť celkom užitočná. Pomocou nej by sa nám napríklad ľahko triedilo – nahádžeme všetky čísla do nej, no a potom už len stláčame tlačidlo, kým postupne nevypadnú všetky v utriedenom poradí.

Jednou šikovnou implementáciou takejto krabičky je halda. Je to vlastne binárny strom, ktorý má každé poschodie úplne plné, možno okrem toho posledného, najspodnejšieho. Každý prvok, okrem tých najspodnejších, má teda práve dvoch synov. Prvky musia byť usporiadané tak, aby platilo, že hodnota uložená v ľubovoľnom vrchole je menšia alebo rovná ako každá z hodnôt uložená v jeho synoch (ak nejakých má).

Takto môže vyzerať halda:



Všimnime si zaujímavú vlastnosť haldy: najmenšie číslo je určite v jej koreni. O ostatných číslach už toho tak veľa nevieme povedať – všimni si, že napríklad 22 je hlbšie ako 23.

Na to, že halda je strom, môžeme zase šťastne zabudnúť. Haldu si totiž vieme úplne jednoducho pamätať v poli. Stačí si vrcholy očíslovať po vrstvách. Na políčku s číslom x teda bude prvok, ktorý by sme prečítali ako x-tý, keby sme haldu „čítali po riadkoch“.

Všimnime si niektoré vlastnosti takto uloženej haldy:

- Ak je v halde N prvkov, v poli budú na políčkach 1 až N.
- Koreň haldy, teda najmenší prvok v nej, je na políčku s číslom 1.
- K-ta vrstva haldy sa v poli začína na políčku s indexom 2^{K-1}.
- Synovia vrcholu s číslom x majú vždy čísla $2x$ a $2x + 1$.
- A opačne, otec vrcholu s číslom x má číslo $\lfloor x/2 \rfloor$.

Halda z obrázku by vyzerala v poli takto:

1	2	3	4	5	6	7	8	9	10	11	12	
7	13	23	47	22	59	134	92	123	85			

Ukážeme teraz, ako do haldy efektívne pridávať nové čísla a ako z nej efektívne vyberať najmenšie.

Vloženie prvku:

Ako teda vložíme nejaký prvok do haldy? Zaradíme ho do poľa na prvé voľné miesto. Jediné, čo nám teraz môže kaziť „haldovitosť“, je, že tento prvok môže byť menší od prvku nad ním. V tom prípade túto dvojicu prvkov vymeníme. Rozmyslite si, že opäť môže nastať jediný problém: nový prvok môže byť menší aj od svojho nového otca. V takomto prípade postup opakujeme. (Tomuto sa hovorí „bublanie prvku dohora“.)

Tento postup určite skončí, prinajhoršom vtedy, ked sa nový prvok dostane na úplný vrch haldy – do koreňa. Po jeho skončení je opäť celá halda v poriadku, úspešne sme teda vložili nový prvok.

Výber najmenšieho prvku:

A čo s vybratím najmenšieho prvku? To bude fungovať podobne. Vieme, že najmenší prvok je ten na vrchu haldy. Tentokrát ho ale nemôžeme len tak

odstrániť, vznikla by nám tam totiž diera. No a tú treba niečím zaplniť. Najjednoduchšie riešenie: Zoberieme posledný prvok v poli (t. j. najpravejší list v poslednej vrstve) a presunieme ten na začiatok poľa.

Touto zmenou sme opäť dosiahli, že čísla máme uložené na prvých niekoľkých políčkach poľa. Nemusí to ale ešte byť korektná halda. To, čo nám ju môže kaziť, je práve presunutý prvok. Preto zopakujeme niečo podobné, ako pri vkladaní. Tentokrát ale presunutý prvok môže byť len priveľký, preto ho budeme musieť „prebublať“ dodola. Toto bublanie treba robiť trochu šikovnejšie. Tentokrát totiž náš „zlý“ prvok môže mať až dvoch synov a byť väčší od každého z nich. Hravo ale zistíme, že riešenie je jednoduché: stačí ho vymeniť s menším z oboch synov.

Opäť, tento postup je konečný. Skončíme, ak už sú obidvaja aktuálni synovia väčší alebo rovní presunutému prvku, prípadne ak sa náš prvok prebublal až do najspodnejšej vrstvy. V každom prípade máme opäť korektnú haldu.

Odhad zložitosti:

Všimnime si, čo sa deje pri jednom prebublaní prvku. Pri vkladaní prebubleme v najhoršom z poslednej vrstvy až do koreňa, pri vyberaní minima naopak, z koreňa až po najhlbšiu vrstvu. V obidvoch prípadoch je počet operácií úmerný hĺbke stromu, teda počtu vrstiev. A aká je tá hĺbka? Na zaplnenie K vrstiev haldy potrebujeme $2^K - 1$ prvkov, preto halda s N prvkami má približne $\log_2 N$ vrstiev. Každá operácia s haldou, v ktorej je N prvkov, má teda časovú zložitosť $O(\log N)$.

Triedenie pomocou haldy:

Ako sme už spomínali na začiatku, pomocou haldy vieme ľahko napísať triedenie, známe pod menom HeapSort. (Heap je halda po anglicky.) Pri triedení N prvkov najskôr N-krát vložíme prvok do haldy, potom odtiaľ N-krát vyberieme najmenší. Počas celého tohto procesu nie je nikdy v halde viac ako N prvkov, preto časová zložitosť každej operácie s ňou je $O(\log N)$. Týchto operácií je $2N$, preto je výsledná časová zložitosť HeapSortu $O(N \log N)$. Pamäťová zložitosť haldy aj triedenia pomocou nej je samozrejme $O(N)$.

Listing programu:

```
type sutaziaci = record
                   meno: string[12];
                   body: longint;
                 end;

var S, R : longint;
```

```
    vysledky : array[1..1000047] of sutaziaci;
    i, j, max, min, cur : longint;

procedure swap(var x, y : longint);
var z : longint;
begin z:=x; x:=y; y:=z; end;

procedure swap(var x, y : sutaziaci);
var z : sutaziaci;
begin z:=x; x:=y; y:=z; end;

function horsi(var x, y : sutaziaci) : boolean;
begin
   if x.body <> y.body then horsi:=(x.body < y.body)
                       else horsi:=(x.meno > y.meno);
end;

procedure insert(i : longint);
begin
   while (i>1) and (horsi(vysledky[i],vysledky[i div 2])) do begin
      swap( vysledky[i div 2], vysledky[i] );
      i := i div 2;
   end;
end;

procedure extract(pocet : longint);
var i, j : longint;
begin
   swap( vysledky[1], vysledky[pocet] );
   dec(pocet);
   i := 1;
   while true do begin
      j := i;
      if (2*i <= pocet) and (horsi(vysledky[2*i],vysledky[j])) then j:=2*i;
      if (2*i+1 <= pocet) and (horsi(vysledky[2*i+1],vysledky[j])) then j:=2*i+1;
      if j=i then break;
      swap( vysledky[i], vysledky[j] );
      i := j;
   end;
end;

begin
   { nacitame pocet sutaziacich a pocet rozhodcov }
   readln(S,R);

   { pre kazdeho sutaziaceho:
     * nacitame znamky ktore dostal
     * pocas nacitavania si pamatame doteraz najmensiu,
       doteraz najvacsiu a sucet ostatnych }
   for i:=1 to S do begin
```

```
      readln(vysledky[i].meno);
      vysledky[i].body := 0;
      read(max);
      read(min);
      if (max < min) then swap(max,min);
      for j:=3 to R do begin
         read(cur);
         if (cur > max) then swap(max,cur);
         if (cur < min) then swap(min,cur);
         vysledky[i].body := vysledky[i].body + cur;
      end;
      readln;
   end;

   { utriedime a vypiseme vysledky }
   for i:=2 to S do insert(i);
   for i:=S downto 2 do extract(i);
   for i:=1 to S do writeln(vysledky[i].body,': ',vysledky[i].meno);
end.
```

B-I-2 Cesta

Najjednoduchším riešením tejto úlohy bolo odpovedať na každú otázku simuláciou: Prejdeme po celej ceste od začiatku po koniec, úsek po úseku. Keď nájdeme úsek, kde leží x, začať merať čas a pri y zase prestaneme a vypíšeme odpoveď. Takéto riešenie má časovú zložitosť $O(NQ)$ – pre každú z Q otázok prejdeme v najhoršom prípade všetkých N úsekov cesty.

Lepšie riešenie je založené na nasledujúcom pozorovaní: čas cesty z bodu x do bodu y vieme vypočítať ako rozdiel dvoch časov: času cesty z 0 do y a času cesty z 0 do x.

Ak je celková dĺžka celej cesty D malá, môžeme spraviť jednoduchý trik. Rozdelíme si cestu na D úsekov dĺžky 1 a následne postupne pre každé q od 0 po D spočítame hodnotu c_q: čas, za ktorý sa vieme dostať z bodu 0 do bodu q. Tieto hodnoty spočítame ľahko – zjavne $c_0 = 0$, no a ak poznáme c_q, tak vieme vypočítať c_{q+1} ako $c_q + 1/v_q$, kde v_q je maximálna povolená rýchlosť na úseku medzi bodmi q a $q + 1$.

Keď máme spočítané hodnoty c_q, vieme na každú otázku odpovedať v konštantnom čase – pre ľubovoľné x, y vieme čas cesty z x do y vyjadriť ako $c_y - c_x$.

Takéto riešenie má časovú zložitosť $O(N + D + Q)$.

Toto riešenie vieme pre obmedzenia dané v zadaní „hackersky“ vylepšiť do podoby, v ktorej získa 9 alebo 10 bodov. Vieme si napríklad cestu namiesto úsekov dĺžky 1 rozdeliť na úseky dĺžky 1000, pre každý úsek si zapamätať zmeny rýchlosti, ktoré sa na ňom udejú, a hodnoty c_q si pamätať len pre násobky 1 000. V takto upravenom riešení nám na zodpovedanie ľubovoľnej otázky vystačí nanajvýš pár tisíc operácií.

Vzorové riešenie, ktoré teraz popíšeme, je pravdepodobne o niečo ľahšie na implementáciu. Opäť budeme používať myšlienku o rozdiele dvoch časov. Tentokrát však nebudeme cestu nijak deliť, a jednoducho spočítame hodnoty $T[i]$: čas potrebný na prechod od začiatku cesty po koniec i-teho úseku.

Čo robiť, ak miesto, v ktorom chceme skončiť, neleží presne na hranici úseku? Stačí nám vedieť, v ktorom úseku sa koniec našej cesty nachádza a aká je vzdialenosť medzi začiatkom tohto úseku a koncom našej cesty.

Ešte raz a poriadnejšie. Položme $p_0 = t_0 = 0$ a pre všetky i nech je $p_{i+1} = p_i + d_i$ a $t_{i+1} = t_i + d_i/v_i$. Teda p_i je vzdialenosť od začiatku cesty po koniec i-teho úseku a t_i je čas, za ktorý na toto miesto vieme doraziť. Hodnoty p_i a t_i vieme spočítať v čase $O(N)$ použitím vyššie uvedeného vzťahu.

Teraz ukážeme, ako pre daný bod x zistiť čas, za ktorý vieme prísť od začiatku cesty k nemu. Toto spravíme v dvoch krokoch. V prvom kroku nájdeme úsek, v ktorom x leží – teda nájdeme i, pre ktoré $p_{i-1} < x \leq p_i$. V druhom kroku spočítame samotný čas cesty ako $t_{i-1} + (x - p_{i-1})/v_i$.

Ako efektívne realizovať prvý krok? Použijeme *binárne vyhľadávanie*. Vieme, že hľadané i je z množiny $\{1, \ldots, n\}$. Túto množinu budeme dokola deliť na polovicu, až kým nezostane len jedna hodnota – tá správna.

Predpokladajme, že vieme, že pre nejaké a a b platí $p_a < x \leq p_b$, pričom $b - a > 1$. V takejto situácii zoberieme $c = \lfloor (a + b)/2 \rfloor$ a porovnáme p_c a x. Ak zistíme, že $p_c < x$, dostávame nový vzťah $p_c < x \leq p_b$, v opačnom prípade dostávame $p_a < x \leq p_c$. V oboch prípadoch sme zmenšili interval možností približne na polovicu.

Keďže v našom prípade má cesta N úsekov, vieme ten správny binárnym vyhľadávaním nájsť v čase $O(\log N)$. Zvyšok výpočtov už vieme realizovať v konštantnom čase. Preto vieme na každú otázku odpovedať v čase $O(\log N)$, a celková časová zložitosť nášho vzorového riešenia je $O(N + Q \log N)$.

Listing programu:

```
var N, Q, i, x, y : longint;
    d, v, p : array[0..200047] of longint;
```

```
    t : array[0..200047] of double;

function cas(x : longint) : double;
var i, j, k : longint;
begin
   if x=0 then cas:=0 else begin
      i := 0; j := N;
      while (j-i > 1) do begin
         k := (i+j) div 2;
         if p[k]<x then i:=k else j:=k;
      end;
      cas := t[i] + (x-p[i]) / v[i];
   end;
end;

begin
   read(N);
   for i:=0 to N-1 do read(d[i]);
   for i:=0 to N-1 do read(v[i]);
   t[0] := 0;
   for i:=0 to N-1 do t[i+1] := t[i] + (d[i] / v[i]);
   p[0] := 0;
   for i:=0 to N-1 do p[i+1] := p[i] + d[i];
   read(Q);
   while Q>0 do begin
      dec(Q);
      read(x,y);
      writeln((cas(y)-cas(x)):0:10);
   end;
end.
```

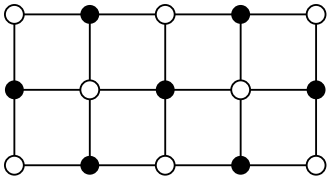
B-I-3 Squaplex

Pre $R = S = 1$ má úloha triviálne riešenie: položíme ceruzku na papier a hneď ju aj zdvihneme.

Ak je práve jedno z R a S rovné 1, úloha zjavne riešenie nemá – akonáhle opustíme začiatočný mrežový bod, použijeme na to jedinú hranu, ktorá z neho vychádza, a teda sa už nemáme ako vrátiť doň späť.

Predpokladajme teda, že $R, S > 1$.

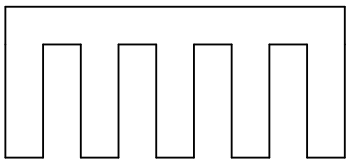
Najprv si ukážeme, že úloha nemôže mať riešenie, ak sú oba čísla R a S nepárne. Uvažujme ofarbenie mrežových bodov dvoma farbami „šachovnicovým“ spôsobom znázorneným na nasledujúcom obrázku:



Každý ťah spĺňajúci podmienky zadania teraz musí striedavo prechádzať čiernymi a bielymi bodmi. Presnejšie, ak si postupne zaznačíme farby navštívených bodov (Č=čierna, B=biela), tie musia tvoriť postupnosť BČBČB...BČB. Táto postupnosť musí začínať aj končiť bielou, keďže začíname aj končíme v ľavom hornom rohu. To ale znamená, že celková dĺžka postupnosti je určite nepárna.

Lenže ak sú oba rozmery šachovnice nepárne, je celkový počet bodov nepárny. Ak navštívime každý z nich práve raz a následne sa vrátime do bodu, kde sme začínali, nutne dostaneme postupnosť párnej dĺžky.

V prípade, že aspoň jedno z čísel R a S je párne, úloha má riešenie. Ťah možno zostrojiť napríklad spôsobom znázorneným na nasledujúcom obrázku:



Je zrejmé, že ak hodnota S je párna, riešenie z predchádzajúceho obrázku funguje pre akúkoľvek výšku, t.j. hodnotu R. V prípade, že S je nepárne a R je párne, stačí predchádzajúci obrázok preklopiť, t. j. vymeniť x-ovú a y-ovú os.

Listing programu:

```
var R, S, i : longint;

procedure chod(kolko : longint; ako : char);
var j : longint;
begin
   for j:=1 to kolko do write(ako);
end;

begin
   readln(R,S);
   if (R=1) and (S=1) then begin writeln; halt; end;
   if (R=1) or (S=1) then begin writeln('Nema riesenie.'); halt; end;
   if (R mod 2=1) and (S mod 2=1) then begin
      writeln('Nema riesenie.');
      halt;
   end;
   if (S mod 2=0) then begin
```

```
      chod(R-1,'D');
      for i:=1 to S-1 do begin
         chod(1,'P');
         if (i mod 2=1) then chod(R-2,'H') else chod(R-2,'D');
      end;
      chod(1,'H');
      chod(S-1,'L');
   end else begin
      chod(S-1,'P');
      for i:=1 to R-1 do begin
         chod(1,'D');
         if (i mod 2=1) then chod(S-2,'L') else chod(S-2,'P');
      end;
      chod(1,'L');
      chod(R-1,'H');
   end;
   writeln;
end.
```

B-I-4 Gaštanová žirafa

Začnime tým, že si skúsime spísať niekoľko jednoduchých pozorovaní:

Najľahšie vieme gaštany priradiť častiam tela žirafy podľa toho, koľko z nich vedie zápaliek. Z hlavy aj z každého kopyta je to práve jedna, z krku sú dve, zo zadku aspoň tri, a z trupu aspoň 4.

Trup od zadku vieme odlíšiť podľa toho, že z neho vedie zápalka do krku.

Ak poznáme trup aj zadok, vieme nájsť hlavu – je to jediný gaštan, ktorý nesusedí ani s jedným z nich.

Tieto pozorovania nám už stačia na návrh efektívnej stratégie:

1. ak ešte nevieme nič

- Vyberieme si niektoré 4 gaštany (napr. tie s číslami 1, 2, 3 a 4).
- Pre každý z nich sa opýtame na všetky ostatné gaštany, aby sme vedeli, s ktorými je spojený a s ktorými nie.
- Keďže v celej žirafe sú len 3 gaštany, z ktorých ide viac ako jedna zápalka, aspoň jeden z našich gaštanov má len jednu zápalku. Jeden taký si vyberme a označme ho a.
- Už poznáme aj gaštan, ktorý je spojený s gaštanom a, označme ho b.
- Opýtame sa na všetky dvojice (b, iný gaštan). Ak zistíme, že z b vedú dve zápalky, je a hlava a b krk. V opačnom prípade je b buď trup alebo zadok.

2. ak vieme, že gaštan b je trup alebo zadok

- Určite existujú aspoň 3 gaštany, s ktorými b nie je spojený. Vyberieme si niektoré 3 takéto gaštany.
- Pre každý z nich zistíme, koľko zápaliek z neho vedie.
- V celej žirafe sú len 3 gaštany, z ktorých ide viac ako jedna zápalka, a jeden z týchto troch je b. Preto aspoň jeden z našich troch gaštanov má len jednu zápalku. Jeden taký si vyberme a označme ho c.
- Gaštan spojený s c označme d.
- Opýtame sa na všetky dvojice (d, iný gaštan). Ak zistíme, že z d vedú dve zápalky, je c hlava a d krk. V opačnom prípade je d buď trup alebo zadok.

3. ak vieme, že gaštany b a d sú trup a zadok

- Jediný gaštan, ktorý nie je spojený ani s trupom, ani so zadkom, je hlava. Keďže už poznáme čísla gaštanov spojených s b aj s d, to jediné, ktoré medzi nimi nie je, je číslo hlavy.

4. ak už vieme, ktorý gaštan je hlava

- Jediný gaštan, ktorý je spojený s hlavou, je krk.
- Jediný ďalší gaštan, ktorý je spojený s krkom, je trup.
- Všetky gaštany, ktoré nie sú spojené s hlavou, krkom ani trupom, sú zadné kopytá.
- Ak ešte nevieme, ktorý gaštan je zadok, nájdeme ho tak, že zoberieme ľubovoľné zadné kopyto a nájdeme gaštan, s ktorým susedí.
- Všetky ostatné gaštany sú predné kopytá.

Ľahko spočítame, že ak sa budeme držať tejto stratégie, určite budeme potrebovať menej ako $10N$ otázok.

Riešenia krajského kola kategórie A

A-II-1 Tobogany

Keby sme mali celú situáciu riešiť ručne a nie programom, dalo by sa postupovať napríklad nasledovne:

Zoženieme si dostatočne veľa dobrovoľníkov. Na začiatku budú všetci len tak sedieť pri vstupe na tobogany a budú všetci pasívni. Teraz budeme postupne spracúvať nasadnutia na tobogan v poradí, v akom sa naozaj odohrali. Čo sa stane, ak máme niekoho poslať dole toboganom? Ak máme hore nejakého aktívneho dobrovoľníka, jedného z nich (je jedno, ktorého) pošleme dole toboganom. Ak nie, zoberieme pasívneho dobrovoľníka, prehlásime ho za aktívneho a pošleme dole toboganom. Aktívny dobrovoľník má samozrejme za úlohu vybehnúť späť na štart, len čo zo svojho toboganu vypadne. No a tvrdíme, že počet aktívnych dobrovoľníkov na konci dňa je rovný minimálnemu počtu ľudí, ktorí mohli v daný deň tobogany používať.

Sedliackemu rozumu by sa mohlo zdať, že takéto riešenie vyzerá celkom dobre – koniec koncov, nového aktívneho dobrovoľníka pridáme len keď musíme. Toto však samo o sebe nie je dôkazom. Zlé jazyky by sa mohli opýtať: „A nešlo by to predsa len lepšie tak, že by sme niekedy skôr pridali viac aktívnych dobrovoľníkov a tým niekedy neskôr ušetrili?“

Správnosť nášho riešenia preto radšej dokážeme poriadnejšie:

Predpokladajme, že máme ľubovoľné optimálne riešenie O a že máme riešenie N vyrobené naším algoritmom. Ukážeme, že O sa dá konečným počtom krokov prerobiť na N, pričom nezmeníme počet ľudí, ktorých potrebujeme.

Všimnime si prvú udalosť, kedy sa O a N líšia – t. j. prvýkrát pošlú v konkrétnom čase dole konkrétnym toboganom rozličné osoby. Ako sa to mohlo stať? A čo s tým spraviť? Rozoberieme niekoľko prípadov:

- V riešení N sme „vyrobili“ nového aktívneho dobrovoľníka len vtedy, ak práve nebol žiaden aktívny dobrovoľník k dispozícii. Keďže riešenie O sa doteraz s N zhodovalo, v takomto prípade tiež nemáme nikoho aktívneho k dispozícii, a teda nutne spravíme to isté ako v N. Tento prípad teda nenastal.

- Vieme teda, že v riešení N sme dole toboganom poslali niektorého aktívneho dobrovoľníka x, ktorý práve čakal na štarte. Ak sme v O poslali

iného aktívneho dobrovoľníka y, upravíme O na O' tak, že od tohto okamihu všetky jazdy x robí y a naopak.

- Zostáva teda posledný prípad: V riešení N sme dole toboganom poslali niektorého aktívneho dobrovoľníka x, zatiaľ čo v našom konkrétnom optimálnom riešení O sme vyrobili nového aktívneho dobrovoľníka z a poslali toho. (Inými slovami, v O prišiel človek, ktorý sa v ten deň ešte nespustil.)

  V tomto prípade opäť môžeme upraviť O na O' jednoducho tak, že vymeníme z a x.

Ukázali sme, že ak sa O a N líšia, tak bez ohľadu na to, ktorý prípad nastal, vždy vieme O upraviť tak, aby sme dostali rovnako dobré riešenie, ktoré sa s N líši až v neskoršom kroku. Po konečne veľa opakovaniach vyššie uvedeného postupu teda z riešenia O nutne vyrobíme naše riešenie N, čím sme dokázali, že aj naše riešenie N je nutne optimálne.

Ako to implementovať? Program sme písali tak, aby sa navyše podľa neho dal priamo aj jeden rozvrh zostrojiť. Vstup reprezentujeme ako 3 fronty (pre každý tobogan máme jednu). Taktiež výstupy z toboganov (t. j. časy, keď z neho vystúpia dobrovoľníci, ktorí sa ním práve vezú) si udržujeme v ďalších 3 frontách. Nasledujúcu udalosť vždy zistíme ako minimum z hodnôt vo vstupných frontách. Pre každú udalosť sa pozrieme na situáciu na výstupe. Nájdeme tam najskoršie vystupujúceho dobrovoľníka. Ak stíha túto jazdu, vyberieme ho z patričnej fronty. Ak nie, musíme pridať nového človeka.

Časová aj pamäťová zložitosť tohto riešenia je $O(N)$, kde N je celkový počet jázd.

Pomalšie riešenia:

Vyššie uvedené riešenie sa dá zostrojiť aj ináč – tak, že budeme uvažovať sekvenčne. Zoberieme prvého človeka, pustíme ho na úplne prvú jazdu, potom na prvú ďalšiu čo stíha, atď. Ak nám ešte zostali nespravené jazdy, tak pridáme druhého, tretieho, atď., kým nepokryjeme všetky jazdy.

Dôkaz správnosti tohto riešenia vyzerá podobne ako pri našom vzorovom riešení. (Presnejšie, ak by sme v našom vzorovom riešení pridali podmienku „ak máš k dispozícii viacero aktívnych dobrovoľníkov, pošli toho s najmenším poradovým číslom“, zostrojili by sme úplne to isté riešenie ako týmto postupom.)

Priamočiara implementácia má v najhoršom možnom prípade časovú zložitosť $O(N^2)$. Existuje ale aj implementácia tohto riešenia s časovou zložitosťou $O(N \log N)$.

Za zmienku taktiež stojí pomalšie riešenie, ktoré je ale univerzálnejšie a dalo by sa napríklad použiť aj v situácii, kedy rôzne tobogany začínajú a končia na rôznych miestach, a teda pešie presuny medzi ich koncami a začiatkami trvajú rôzne dlho.

Prevedieme úlohu na hľadanie maximálneho párovania v bipartitnom grafe. Jednu partíciu budú tvoriť časy konca jázd, druhú časy začiatku, a hrana znamená, že po danom konci sa stíha daný začiatok. Každému platnému rozvrhu zodpovedá nejaké párovanie v tomto grafe a naopak. Totiž každá hrana, ktorú vyberieme do párovania, vlastne hovorí, že človek, ktorý práve dokončil jednu jazdu, má ako nasledujúcu spraviť príslušnú druhú jazdu. Zjavne každou vybranou hranou ušetríme jedného človeka, preto hľadaný minimálny počet ľudí vieme vypočítať ako N mínus veľkosť maximálneho párovania v našom grafe.

Listing programu:

```
#include <cstdio>
#include <queue>

using namespace std;

#define MAX_TIME 2000000100

int main()
{
        int t[3], d, n[3], a;
        queue<int> nastup[3];
        queue<int> vystup[3];
        scanf("%d %d %d %d", &t[0], &t[1], &t[2], &d);
        for(int i = 0; i < 3; i++)
        {
                scanf("%d", &n[i]);
                for(int j = 0; j < n[i]; j++)
                {
                        scanf("%d", &a);
                        nastup[i].push(a);
                }
                nastup[i].push(MAX_TIME); //vlozime fiktivny posledny nastup
        }
        int prvyNastupPos, prvyNastupCas;
        int prvyVystupCas, prvyVystupPos;
        int pocetDeti = 0;
        while(1)
        {
                prvyNastupCas = MAX_TIME;
                prvyNastupPos = -1;
                for(int i = 0; i < 3; i++) //najdeme prvy nespracovany nastup
                {
```

```
                        if(nastup[i].front() < prvyNastupCas)
                        {
                                prvyNastupCas = nastup[i].front();
                                prvyNastupPos = i;
                        }
                }
                if(prvyNastupCas == MAX_TIME) break; //vycerpane vsetko, koncime
                prvyVystupCas = MAX_TIME;
                for(int i = 0; i < 3; i++) //najdeme prvy nepouzity vystup
                {
                        if(vystup[i].size() == 0) continue;
                        if(vystup[i].front() < prvyVystupCas)
                        {
                                prvyVystupCas = vystup[i].front();
                                prvyVystupPos = i;
                        }
                }
                if(prvyVystupCas <= prvyNastupCas) //mozeme tento vystup pouzit
                        vystup[prvyVystupPos].pop();
                else
                        pocetDeti++;
                nastup[prvyNastupPos].pop();
                vystup[prvyNastupPos].push(prvyNastupCas+t[prvyNastupPos]+d);
        }
        printf("%d\n", pocetDeti);
}
```

A-II-2 Oplotenie farmy

Vzorové riešenie tohto príkladu bude využívať postup, ktorý sme použili v úlohe o čokoláde z domáceho kola. Každý štvorcový úsek farmy môžeme jednoznačne identifikovať bodom, v ktorom sa nachádza jeho pravý dolný roh a dĺžkou jeho strany. Základnou myšlienkou riešenia je, že namiesto počítania pestrých štvorcov môžeme spočítať štvorce všetky a od nich odčítať počet jednofarebných.

Keby sme pre každý bod (r, s) vedeli počet jednofarebných štvorcov, ktorých pravý dolný roh je (r, s), potom by sme už vedeli spočítať hľadaný výsledok. Totiž počet všetkých štvorcov (jednofarebných a pestrých dokopy), ktoré majú pravý dolný roh na (r, s) je $\min(r, s)$. A teda vieme počet pestrých štvorcov pre (r, s) vypočítať ako $\min(r, s)$ mínus počet tých, ktoré sú jednofarebné.

Pozrime sa bližšie na štvorce rôznych veľkostí ktoré majú pravý dolný roh na (r, s). Zjavne platí, že akonáhle je niektorý z nich pestrý, sú pestré aj všetky väčšie od neho – lebo ho celý obsahujú.

Označme $K(r,s)$ dĺžku strany najväčšieho štvorca, ktorý má pravý dolný roh na (r,s) a v ktorom sú všetky plodiny rovnaké. Keď sa teraz pozrieme na štvorce, ktoré majú pravý dolný roh na (r,s), tak zjavne tie, ktoré majú dĺžku strany od 1 po $K(r,s)$ sú jednofarebné a všetky väčšie sú pestré. Preto $K(r,s)$ je zároveň rovné počtu jednofarebných štvorcov, ktorých pravý dolný roh je (r,s).

A ako vypočítať hodnotu $K(r,s)$? Uvažujme políčka $(r,s-1)$, $(r-1,s)$ a $(r-1,s-1)$. Ak je v aspoň jednom z týchto políčok iná plodina ako na políčku (r,s), potom je $K(r,s)$ rovné jednej, pretože štvorec s dĺžkou strany dva už je pestrý. V prípade, že sa plodiny na týchto políčkach zhodujú s plodinou na políčku (r,s), potom platí:

$$K(r,s) = 1 + \min(\, K(r-1,s),\, K(r,s-1),\, K(r-1,s-1)\,)$$

Dôkaz tejto rovnosti je zhodný s dôkazom v riešeniach domáceho kola.

Keďže na výpočet $K(r,s)$ stačí poznať hodnoty $K(r-1,s), K(r,s-1), K(r-1,s-1)$, môžeme postupovať po riadkoch od hora dole a v rámci riadku zľava doprava a každú hodnotu získať v konštantnom čase s využitím hodnôt, ktoré už poznáme. Preto má toto riešenie časovú zložitosť $O(RS)$. Ak by sme si načítali celý vstup, mali by sme aj pamäťovú zložitosť $O(RS)$. Avšak, podobne ako v domácom kole, môžeme túto zložitosť zlepšiť na $O(R)$, keď si uvedomíme, že stačí v pamäti držať len dva riadky vstupu a tiež dva riadky hodnôt $K(r,s)$, pretože predchádzajúce riadky sú už pre nás zbytočné.

Listing programu:

```
#include <cstdio>
using namespace std;

int R,S,K;
int M[2][2500],D[2][2550];
long long res;

int min(int a, int b){return a < b ? a : b;}

int main(){
  scanf("%d %d %d ",&R,&S,&K);
  //uplne hore a uplne vlavo si domyslime imaginarne policka
  //s plodinou cislo K, aby sa nam lahsie pocitalo
  for(int i=0;i<=S;i++) D[0][i] = K;
  for(int i=1;i<=R;i++){
    int novy = i%2;
    int stary = (novy+1)%2;
    D[novy][0] = K;
    for(int j=1;j<=S;j++) scanf("%d ",&D[novy][j]);
```

```
    for(int j=1;j<=S;j++){
      if (D[novy][j-1] != D[novy][j]) M[novy][j] = 1;
      else if (D[stary][j] != D[novy][j]) M[novy][j] = 1;
      else if (D[stary][j-1] != D[novy][j]) M[novy][j] = 1;
      else M[novy][j] = 1 + min( min(M[novy][j-1],M[stary][j]), M[stary][j-1] );
      res += min(i,j) - M[novy][j];
    }
  }
  printf("%Ld\n",res);
  return 0;
}
```

A-II-3 Obmedzovač rýchlosti

Prvé pozorovanie tejto úlohy je, že nech si vyberieme akúkoľvek trasu, optimálne nastavenie obmedzovača rýchlosti je rovné minimu z obmedzení rýchlosti na našej trase – menej sa neoplatí, viac nesmieme.

Dôsledkom je, že pre dosiahnutie najnižšieho času cesty medzi dvoma mestami má zmysel nastavovať obmedzovač rýchlosti len na jednu z rýchlostí, ktoré sú na vstupe.

Pomalšie riešenie:

Uvažujme problém, kedy už máme nastavené obmedzenie na nejakú rýchlosť v. Za aký čas sa teraz vieme dostať z mesta x do mesta y?

Nech G_v je podgraf pôvodného grafu, ktorý obsahuje len hrany, kde je rýchlosť v ešte povolená. T. j. ak máme nastavené obmedzenie na rýchlosť v, potom môžeme cestovať len po hranách grafu G_v. Keďže rýchlosť jazdy už máme určenú, zrejme najlepší čas jazdy dosiahneme vtedy, ak si zvolíme najkratšiu možnú trasu (v kilometroch).

Tým sa nám ponúka nasledujúce riešenie:

```
pre každú rýchlosť v zo vstupu
    vytvoríme graf G_v;
    riešime problém (dĺžkovo) najkratšej cesty v grafe G_v;
    pre každú dvojicu miest x a y si zapamätáme riešenie
        s časom dist_v(x,y)/v;
```

kde $dist_v(x, y)$ určuje dĺžku (v kilometroch) najkratšej cesty medzi mestami x, y v grafe G_v.

Ako je to s časovou zložitosťou takéhoto riešenia? Podľa zadania a nášho úvodného pozorovania stačí pre v uvažovať všetky možné rýchlosti na vstupe, ktorých je dohromady M. Graf G_v vieme zakaždým ľahko zostrojiť v čase $O(M)$

tak, že prejdeme cez všetky hrany pôvodného grafu. Otázkou už iba ostáva, ako vyriešiť „klasický“ problém najkratšej cesty v grafe G_v medzi každou dvojicou vrcholov. Dva možné prístupy:

- Algoritmus Floyd-Warshall, ktorý tento problém rieši v čase $O(N^3)$.
- N-krát spustený Dijkstrov algoritmus (raz z každého vrcholu). Tým vieme získať opäť časovú zložitosť $O(N^3)$, alebo v prípade využitia haldy získame riešenie v čase $O(NM \log N)$.

Nakoľko celý cyklus sa spúšťa pre každú rýchlosť na vstupe (t.j. najviac M-krát), celková časová zložitosť takéhoto riešenia je $O(MN^3)$, resp. $O(M^2N \log N)$ a pamäťová zložitosť $O(N^2)$.

Vzorové riešenie:

Naše vzorové riešenie využíva podobnú myšlienku ako algoritmus Floyd-Warshall. Začneme tým, že zoradíme všetky hrany zo vstupu do postupnosti $e_1, e_2, \ldots, e_m$ tak, aby pre ich maximálne povolené rýchlosti platilo $v_1 \geq v_2 \geq v_3 \geq \ldots \geq v_m$.

Úlohu teraz vyriešime metódou dynamického programovania. Postupne pre každé k spočítame všetky hodnoty $d_k(x, y)$ – dĺžku najkratšej cesty (v kilometroch) medzi mestami x a y, ak sa smieme pohybovať len po hranách $e_1, e_2, \ldots, e_k$. (Resp. $d_k(x, y) = \infty$, ak taká cesta neexistuje.)

Ekvivalentne, podľa definície z predchádzajúceho riešenia môžeme povedať, že $d_k(x, y)$ je dĺžka najkratšej cesty medzi mestami x a y v grafe G_{v_k}. Ako sme konštatovali skôr, všetko, čo by sme potrebovali, je vyriešiť tento problém pre každú z hodnôt $k = 1, 2, \ldots, M$.

Našou ideou teraz bude v k-tom kroku „sprístupniť“ hranu e_k a prerátať hodnoty $d_k(x, y)$ pre všetky dvojice miest x, y. Pri určení hodnoty $d_k(x, y)$ uvažujeme dve možnosti.

- V prípade, že najkratšia cesta hranu e_k nepoužíva, je $d_k(x, y) = d_{k-1}(x, y)$.
- V opačnom prípade sa dá najkratšia cesta z x do y rozdeliť na 3 úseky: prídeme z x do niektorého vrcholu hrany e_k, prejdeme ňou a z jej druhého konca prídeme do y. Aj v prvom, aj v treťom úseku sa už vyskytujú len hrany s číslom menším ako k, a teda už vieme ich optimálnu dĺžku.

  Formálne, označme u_{k1} a u_{k2} vrcholy spojené hranou e_k. Potom platí buď $d_k(x, y) = d_{k-1}(x, u_{k,1}) + |e_k| + d_{k-1}(u_{k,2}, y)$ alebo $d_k(x, y) = d_{k-1}(x, u_{k,2}) + |e_k| + d_{k-1}(u_{k,1}, y)$ – podľa toho, ktorým smerom naša trasa prechádza hranou e_k.

My vždy máme na výber, či hranu e_k použiť alebo nie, a ak áno, tak v ktorom smere. Vyberieme si samozrejme najlepšiu z uvedených troch možností. Preto $d_k(x, y)$ je vždy rovné minimu z uvedených troch možností.

Tým sme získali rekurzívny vzťah pre hodnoty $d_k(x, y)$. Každú z týchto MN^2 hodnôt pomocou neho spočítame v konštantnom čase, takéto riešenie má teda časovú zložitosť $O(MN^2)$.

Pre zníženie pamäťovej zložitosti si stačí všimnúť, že hodnoty d_k závisia len od hodnôt d_{k-1}, a teda si vystačíme s pamäťou veľkosti $O(N^2)$.

Listing programu:

```
#include<cstdio>
#include<algorithm>
using namespace std;

struct hrana{
   double m,d;
   int x,y;
   hrana(){}
   hrana(int x, int y, double m, double d): x(x), y(y), m(m), d(d) {}
};

int operator < (const hrana &h1, const hrana &h2){
   return h1.m > h2.m;
}

const double INF=1e50;
const int maxM=5000, maxN=100;
int n,m;
hrana h[maxM];
double vysl[maxN][maxN];//tu si pamatame celkovy vysledok, tj. casy medzi mestami
double d[maxN][maxN];   //v d si pamatame vzdialenost pouzitim prvych k-1 hran

int main(){
   scanf("%d %d",&n,&m);
   for(int k=0;k<m;k++){
      scanf("%d %d %lf %lf",&h[k].x,&h[k].y,&h[k].d,&h[k].m);
      h[k].x--; h[k].y--;
   }
   sort(h,h+m*sizeof(hrana));
   for(int i=0;i<n;i++)
      for(int j=0;j<n;j++) d[i][j]=vysl[i][j]=(i==j)?0.0:INF;

   for(int k=0;k<m;k++){
      double v=h[k].m;
      for(int i=0;i<n;i++)
         for(int j=0;j<n;j++){
            d[i][j]=min( d[i][j] , d[i][h[k].x] + h[k].d + d[h[k].y][j]);
```

```
            d[i][j]=min( d[i][j] , d[i][h[k].y] + h[k].d + d[h[k].x][j]);
            vysl[i][j]=min(vysl[i][j], d[i][j]/v);
         }
   }
   for(int i=0;i<n;i++){
      for(int j=0;j<n;j++) printf("%.3lf ",vysl[i][j]);
      printf("\n");
   }
}
```

A-II-4 Počítač s gumenou rúrou

Podúloha a: Lucasove čísla:

(V celom riešení tejto podúlohy ticho predpokladáme, že všetky sčítania sú sčítaniami modulo 65 536.)

Použijeme dva registre a a b. Na začiatku do nich uložíme hodnoty $L_0 = 2$ a $L_1 = 1$.

Predstavme si teraz, že v a máme hodnotu L_{x-2}, v b hodnotu L_{x-1} a rúra je prázdna. Teraz vieme spočítať hodnotu L_x: vložíme do rúry hodnotu z a, vložíme tam aj hodnotu z b a vykonáme príkaz `add`.

Následne do rúry vložíme ešte raz hodnotu z registra b. V rúre teda teraz máme za sebou najskôr hodnotu $L_{x-2} + L_{x-1} = L_x$ a potom hodnotu L_{x-1}. Prvú z nich načítame do registra b a druhú do registra a.

Tým sme sa dostali do „o jedno posunutej“ situácie: začínali sme s hodnotami L_{x-2} a L_{x-1}, skončili sme s hodnotami L_{x-1} a L_x.

A teraz už môžeme ľahko napísať celý program riešiaci súťažnú úlohu: Číslo z rúry uložíme do registra n. Inicializujeme registre a a b na L_0 a L_1. Vyššie uvedený postup n-krát zopakujeme, čím dostaneme v registroch a a b hodnoty L_n a L_{n+1}. A na záver hodnotu z registra a vypíšeme na výstup.

```
get n
put 2 ; get a
put 1 ; get b
label loop
jz n koniec
put n ; put 1 ; sub ; get n
put a ; put b ; add
put b
get b
```

```
get a
jump loop
label koniec
put a
print
```

Podúloha b: hľadanie výskytu čísla 47:

Testovať, či je číslo rovné 47, vieme napríklad pomocou inštrukcie `jeq` – skok v prípade rovnosti hodnôt v dvoch registroch. Na to, aby sme ju mohli používať, však musíme dostať do nejakého registra hodnotu 47. A to nevieme spraviť len tak, že ju vložíme do rúry a hneď načítame do registra – lebo rúru už máme plnú vstupu.

Tento problém vyriešime tak, že obsah rúry „pretočíme“. Začneme tým, že si do rúry vložíme hodnotu 0. Táto bude slúžiť ako „zarážka“ za vstupnou postupnosťou kladných čísel. Za túto 0 vložíme hodnotu 47.

Teraz budeme dokola opakovať: načítame hodnotu z rúry do registra, a ak to nie je nula, vložíme ju naspäť do rúry. Takto sa bude obsah rúry postupne otáčať, až kým niekedy do registra nenačítame našu nulu. V tomto okamihu vieme, že prvé číslo v rúre je naša 47 a za ňou nasleduje celá vstupná postupnosť. Hodnotu 47 si teda načítame do registra z.

Teraz sme presne v rovnakej situácii ako na začiatku, s jediným rozdielom – v registri z máme číslo 47. S tým teraz každú hodnotu v zadanej postupnosti porovnáme. Ak sa nám rúra vyprázdni, vložíme do nej hodnotu 0, vypíšeme ju na výstup a skončíme. Ak niekedy narazíme na hodnotu 47, najskôr v cykle vyprázdnime zvyšok rúry, potom do nej vložíme hodnotu 1, vypíšeme ju na výstup a skončíme.

```
put 0
put 47
label pretoc
get a ; jz a dalej ; put a
jump pretoc

label dalej
get z
label testuj
jempty nenasiel
get a
```

```
jeq a z nasiel
jump testuj

label nasiel
jempty pis1
get x
jump nasiel

label pis1
put 1 ; print
jump koniec

label nenasiel
put 0 ; print
jump koniec

label koniec
```

Pre zaujímavosť uvedieme ešte jednu možnosť, ako riešiť druhú polovicu úlohy. Za postupnosť si opäť vložíme nulu ako zarážku. Postupne spracúvame zadanú postupnosť a vždy, keď nájdeme hodnotu 47, do rúry vložíme číslo 1. Po tom, ako z rúry načítame nulu-zarážku, do rúry vložíme ešte jednu 0.

V tomto okamihu teda máme v rúre najskôr niekoľko jednotiek (a to presne toľko, koľkokrát bola v zadanej postupnosti hodnota 47) a následne jednu nulu. Prvé číslo z rúry vypíšeme na výstup a skončíme.

Podúloha c: výpis párnych čísel:

Keď chceme zistiť, či je nejaké číslo párne, môžeme spočítať zvyšok, aký dáva po delení 2, a overiť, že je 0. Zvyšok vieme spočítať inštrukciou `mod`. Zároveň si ale musíme niekde „odložiť“ aj pôvodnú hodnotu čísla, lebo `mod` svoje vstupy z rúry odstráni.

V prvej fáze nášho riešenia si pripravíme obsah rúry tak, aby sme mohli použiť inštrukciu `mod` na vypočítanie zvyškov po delení dvoma. Použijeme rovnaký trik ako v predchádzajúcej podúlohe – začneme tým, že za vstupnú postupnosť vložíme 0 ako zarážku. Teraz budeme po jednom čítať vstupnú postupnosť a za každé prečítané číslo a do rúry postupne vložíme hodnoty 1, a, 2, a. Hodnota 1 nám bude hovoriť „ešte nasleduje ďalší prvok“, nasledujúce dve hodnoty a a 2 použijeme ako vstupy pre `mod`, a posledná hodnota a zostane zachovaná.

Druhú fázu nášho riešenia opäť začneme tým, že na koniec postupnosti pridáme 0 ako zarážku. Teraz dokola opakujeme: Načítame z rúry hodnotu do registra. Ak je to 0, už sme spracovali celú postupnosť a prejdeme na tretiu fázu. Ak nie, vykonáme inštrukciu `mod`. Tá zoberie zo začiatku rúry hodnoty a a 2 a namiesto nich vloží do rúry a mod 2. A následne načítame z rúry hodnotu do registra a vložíme ju späť.

Ak sme na začiatku mali v rúre postupnosť $(s_1, \ldots, s_k)$, tak po skončení druhej fázy nášho riešenia tam máme postupnosť $(s_1 \bmod 2, s_1,\ s_2 \bmod 2, s_2, \ldots,$ $s_k \bmod 2, s_k)$. No a v tretej, poslednej fáze použijeme práve spočítané zvyšky na to, aby sme určili, ktoré čísla vypísať a ktoré nie. Prečítame zvyšok, ak je 1, nasledujúce číslo zahodíme, ak je 0, nasledujúce číslo vypíšeme na výstup. Keď sa rúra vyprázdni, skončíme.

```
put 0
label mark
get a ; jz a druha_faza
put 1 ; put a ; put 2 ; put a
jump mark

label druha_faza
put 0
label dalej
get a ; jz a vypis
mod
get a ; put a
jump dalej

label vypis
jempty koniec
get b ; jz b chceme
get a
jump vypis

label chceme
print
jump vypis

label koniec
```

Riešenia krajského kola kategórie B

B-II-1 Telemánia

Najjednoduchšie riešenie tejto úlohy je priamočiare. Postupne pre každý z M hovorov spracujeme každého jeho aktéra – teda prezrieme všetkých N zákazníkov, či je to jeden z nich, a ak ho nájdeme, tak pripočítame práve prevolaný počet sekúnd k jeho celkovému času volania. Toto riešenie má časovú zložitosť $O(MN)$.

V tomto riešení najskôr popíšeme niekoľko riešení, ktoré sú lepšie ako toto triviálne, ale nie sú optimálne. Optimálne riešenie nájdete popísané v časti s nadpisom „Vzorové riešenie“.

Myšlienka lepšieho riešenia:

Ako naše priamočiare riešenie zlepšiť? Všimnime si, že pre každý hovor vždy znova a znova prechádzame celý zoznam zákazníkov. Keby sme na začiatku tento zoznam upravili do nejakej vhodnejšej podoby, mohli by sme potom vedieť ľahšie zisťovať, či konkrétne telefónne číslo patrí nášmu zákazníkovi (a ak áno, ktorému).

Veľmi dobrý nápad je napríklad všetkých zákazníkov usporiadať podľa telefónneho čísla.

Ako nám toto pomôže? Keď nás teraz bude zaujímať, komu patrí konkrétne telefónne číslo, môžeme použiť binárne vyhľadávanie: Pozrieme sa do prostriedku zoznamu. Ak tam je hľadané číslo, skončili sme, ak nie, vieme, či ho hľadať v prvej alebo v druhej polovici. A tento postup opakujeme, kým číslo buď nenájdeme, alebo nezistíme, že v zozname zákazníkov nie je.

Implementácia lepšieho riešenia:

Na triedenie zoznamu zákazníkov môžeme použiť ľubovoľné efektívne triedenie, napr. HeapSort. Toto triedenie je popísané vo vzorových riešeniach domáceho kola. Časová zložitosť tohto algoritmu je $O(N \log N)$, kde N je počet triedených záznamov – v našom prípade teda zákazníkov.

Pozrime sa teraz na jedno binárne vyhľadávanie. Zoznam, v ktorom hľadáme, má N prvkov. Pri binárnom vyhľadávaní v každom kroku zmenšíme počet možností na polovicu. Preto počet krokov, ktoré spravíme, bude rádovo $\log_2 N$.

V našom algoritme spravíme binárne vyhľadávanie $2M$-krát: raz pre každého z účastníkov každého z M hovorov. Táto časť riešenia má teda časovú zložitosť

$O(M \log N)$.

Celková časová zložitosť tohto riešenia je teda $O((M + N) \log N)$.

Listing programu:

```
type zakaznik = record
   cislo : string[12];
   meno : string;
   cas : longint;
end;

var N : longint;
    zakaznici : array[1..1000047] of zakaznik;

procedure swap(var x, y : zakaznik);
var z : zakaznik;
begin z:=x; x:=y; y:=z; end;

function neskor(var x, y : zakaznik) : boolean;
begin neskor := x.cislo > y.cislo; end;

procedure insert(i : longint);
begin
   while (i>1) and (neskor(zakaznici[i],zakaznici[i div 2])) do begin
      swap( zakaznici[i div 2], zakaznici[i] );
      i := i div 2;
   end;
end;

procedure extract(pocet : longint);
var i, j : longint;
begin
   swap( zakaznici[1], zakaznici[pocet] );
   dec(pocet);
   i := 1;
   while true do begin
      j := i;
      if (2*i <= pocet) and (neskor(zakaznici[2*i],zakaznici[j]))
         then j:=2*i;
      if (2*i+1 <= pocet) and (neskor(zakaznici[2*i+1],zakaznici[j]))
         then j:=2*i+1;
      if j=i then break;
      swap( zakaznici[i], zakaznici[j] );
      i := j;
   end;
end;

procedure nacitaj_zakaznikov;
var c : char;
    i, j : longint;
```

```
begin
   readln(N);
   for i:=1 to N do begin
      zakaznici[i].cislo:='';
      for j:=1 to 10 do begin
         read(c);
         zakaznici[i].cislo := zakaznici[i].cislo+c;
      end;
      read(c);
      readln(zakaznici[i].meno);
      zakaznici[i].cas := 0;
   end;
end;

procedure utried_zakaznikov;
var i : longint;
begin
   for i:=2 to N do insert(i);
   for i:=N downto 2 do extract(i);
end;

procedure spracuj_hovor(cislo: string; cas: longint);
var lo, hi, med : longint;
begin
   if cislo < zakaznici[1].cislo then exit;
   { binarnym vyhladavanim najdeme cislo v zozname zakaznikov }
   lo:=1; hi:=N+1;
   while hi-lo>1 do begin
      med:=(lo+hi) div 2;
      if zakaznici[med].cislo <= cislo then lo:=med else hi:=med;
   end;
   if zakaznici[lo].cislo <> cislo then exit; { nenasiel }
   inc( zakaznici[lo].cas, cas );
end;

procedure spracuj_hovory;
var c : char;
    M, i, j, cas : longint;
    cislo1, cislo2 : string;

begin
   readln(M);
   for i:=1 to M do begin
      cislo1:=''; for j:=1 to 10 do begin read(c); cislo1:=cislo1+c; end;
      read(c);
      cislo2:=''; for j:=1 to 10 do begin read(c); cislo2:=cislo2+c; end;
      read(c);
      readln(cas);
      spracuj_hovor(cislo1,cas);
      spracuj_hovor(cislo2,cas);
```

```
   end;
end;

procedure vypis_najlepsieho;
var i, vitaz : longint;
begin
   vitaz := 1;
   for i:=2 to N do if zakaznici[i].cas > zakaznici[vitaz].cas then vitaz:=i;
   writeln(zakaznici[vitaz].cislo+' '+zakaznici[vitaz].meno);
end;

begin
   nacitaj_zakaznikov;
   utried_zakaznikov;
   spracuj_hovory;
   vypis_najlepsieho;
end.
```

Alternatívne, rovnako dobré riešenie:

Niektoré programovacie jazyky nám ponúkajú dátovú štruktúru známu pod názvom *asociatívne pole*. Ide o niečo podobné ako klasické pole, ale indexovať do neho môžeme nie len číslami od 0 po nejaké k, ale ľubovoľnými objektmi, ktoré vieme usporiadať.

Ak máme k dispozícii takéto asociatívne polia, môžeme použiť dve: v jednom si ku každému zákazníckemu číslu uložíme meno zákazníka, v druhom jeho počet prevolaných sekúnd.

Konkrétny príklad: v C++ máme v knižnici STL k dispozícii dátovú štruktúru `map`, ktorá predstavuje asociatívne pole.[4] Táto štruktúra je implementovaná pomocou vyvažovaného binárneho stromu, každá operácia s ňou teda trvá čas priamo úmerný logaritmu počtu uložených záznamov.

Riešenie využívajúce `map` má teda tiež časovú zložitosť $O((M+N)\log N)$.

Listing programu:

```
#include <iostream>
#include <string>
#include <map>
#include <vector>
using namespace std;

int main() {
  // nacitame telefonny zoznam
  int N;
```

[4] Presnejšie, asociatívne pole, ktoré je navyše usporiadané podľa kľúčov, teda objektov, ktoré používame ako indexy doň.

```
  cin >> N;
  vector<string> cisla(N);
  map<string,string> mena;
  for (int n=0; n<N; ++n) { cin >> cisla[n]; cin >> mena[cisla[n]]; }

  // nacitavame hovory a zaznamenavame si ich casy
  int M;
  cin >> M;
  map<string,int> casy;
  string cislo1, cislo2;
  int cas;
  for (int m=0; m<M; ++m) {
    cin >> cislo1 >> cislo2 >> cas;
    casy[cislo1] += cas;
    casy[cislo2] += cas;
  }

  // najdeme vitaza
  string vitaz = cisla[0];
  for (int n=1; n<N; ++n) if (casy[cisla[n]] > casy[vitaz]) vitaz = cisla[n];
  cout << vitaz << " " << mena[vitaz] << endl;
}
```

Hešovanie:

(Táto časť vzorového riešenia je „nepovinná“ a zaoberá sa komplikovanou metódou, netrpezlivý čitateľ môže preskočiť rovno na časť Vzorové riešenie, kde nájde popis jednoduchšieho a efektívnejšieho riešenia.)

Ešte efektívnejšie riešenie môžeme dosiahnuť použitím hešovania. Trik je v tom, že nepotrebujeme nutne zákazníkov utriediť.

Keby napríklad telefónne čísla boli len 6-ciferné, mohli by sme úlohu ľahko riešiť pomocou obyčajného poľa, do ktorého by sme indexovali priamo telefónnymi číslami.

Základná myšlienka hešovania je v tom, že toto isté chceme robiť aj pre veľké čísla. Ak by sme napríklad našli funkciu f, ktorá pre každé z našich N zákazníckych čísel vráti číslo trebárs od 0 po $3N$ a navyše bude platiť, že pre žiadne dve zákaznícke čísla nedostaneme ten istý výstup, tak sme vyhrali – vždy, keď spracúvame hovor, zoberieme telefónne číslo, funkciou f ho „preložíme“ (zahešujeme) na číslo z malého rozsahu a toto číslo (hešovaciu hodnotu) použijeme ako index do poľa.

Dobrá funkcia f v praxi môže napríklad vyzerať nasledovne: načítame N, nájdeme náhodné prvočíslo p medzi $2N$ a $4N$ a zadefinujeme $f(x) = x \bmod p$.

Napríklad pre vstup zo zadania by sme mohli zvoliť $p = 11$, potom hešovacia hodnota Jožkovho čísla bude $f(0975342583) = 975342583 \bmod 11 = 0$, Ferko bude mať hešovaciu hodnotu 6 a Jano 2.

Ak by všetko fungovalo tak, ako sme to práve popísali, tak máme riešenie v čase $O(N + M)$: Najskôr pre každé z N zákazníckych čísel spočítame jeho hešovaciu hodnotu. Potom pre každého volajúceho aj volaného spočítame hešovaciu hodnotu h jeho čísla. Ak žiadne zákaznícke číslo nemalo hodnotu h, nič nerobíme. Ak ju nejaké malo, pozrieme sa, či sa zhoduje s práve spracúvaným, a ak áno, zvýšime mu prevolaný čas.

Samozrejme, v praxi sa môže stať, že si zvolíme hešovaciu funkciu a následne zistíme, že niektoré dve zákaznícke čísla majú tú istú hešovaciu hodnotu. Toto sa volá *kolízia* a je potrebné to ošetriť – napríklad tak, že pre každú možnú hešovaciu hodnotu si budeme pamätať zoznam všetkých zákazníckych čísel, ktoré ju majú.

Takéto riešenie má, pri dobrej voľbe hešovacej funkcie, *očakávanú* časovú zložitosť $O(N + M)$.

Vzorové riešenie:

Teraz ukážeme vzorové riešenie, ktoré bude mať vždy časovú zložitosť $O(N + M)$. Toto riešenie bude založené na nasledujúcej myšlienke: Keďže telefónne čísla majú konštantný počet cifier, môžeme to využiť a utriediť ich v lineárnom čase. Týmto vylepšením dostaneme algoritmus s časovou zložitosťou $O(N + M \log N)$.

Ale ako sa zbaviť binárneho vyhľadávania? Jednoducho – nahradíme ho ďalším triedením. Prejdeme celý zoznam hovorov a pre každý hovor „číslo1 číslo2 čas“ pridáme do nového poľa dva záznamy: „číslo1 čas“ a „číslo2 čas“. Toto nové pole následne utriedime podľa telefónneho čísla.

Pre príklad zo zadania by toto utriedené pole vyzeralo nasledovne:

```
0955956114 137
0955956114 293
0972125726 137
0972125726 36
0972125726 54
0972654124 36
0972654124 80
0975342583 293
0975342583 54
0975342583 80
```

No a teraz, keď máme dve utriedené polia (zákazníkov aj hovory), vieme už

ľahko zistiť celkové časy hovorov našich zákazníkov. Stačí si všimnúť, že v poli hovorov tvoria hovory každého zákazníka súvislý úsek, a navyše ich čísla sú v tom istom poradí ako v poli so zákazníkmi.

Zostáva ukázať, ako budeme triediť telefónne čísla. Použijeme triedenie známe pod názvom RadixSort. Postupne spravíme 10 prechodov, pričom po k-tom prechode budeme mať čísla utriedené podľa ich *posledných* k cifier.

Pozrime sa bližšie, ako bude vyzerať k-ty prechod. Na začiatku máme čísla utriedené podľa posledných $k-1$ cifier. Pozrieme sa teraz na k-te cifry všetkých čísel, ktoré triedime, a spočítame si, že je medzi nimi c_0 núl, c_1 jednotiek, atď. Teraz vieme, že v poradí utriedenom podľa posledných k cifier budú čísla s nulou na pozíciách 1 až c_0, čísla s jednotkou na pozíciách c_0+1 až c_0+c_1, atď. Tak už len prejdeme zaradom všetky čísla a každé umiestnime na správne miesto do nového poradia.

(Všimnite si, na čo slúžil predpoklad, že čísla už sú utriedené podľa posledných $k-1$ cifier: keď ich ukladáme na nové miesta, tak čísla, ktoré majú na k-tej pozícii rovnakú cifru spracúvame v správnom poradí.)

Každý prechod zjavne prebehne v lineárnom čase, preto je časová zložitosť RadixSortu pre 10-ciferné čísla lineárna od ich počtu.

Celkovo teda toto vzorové riešenie potrebuje $O(N)$ operácií na utriedenie zákazníkov, $O(M)$ na utriedenie hovorov a následne $O(N+M)$ na spracovanie hovorov a vypočítanie výsledku. Celková časová zložitosť je teda $O(N+M)$, čo je zjavne optimálne.

Listing programu:

```
{$H+} { fpc direktiva ktora zapne AnsiStringy, aby zaznamy nezrali zbytocne vela
pamate }

type zakaznik = record
   cislo : string[12];
   meno : string;
   cas : longint;
end;

var N, M : longint;
    zakaznici, hovory, pomocne : array[0..2000047] of zakaznik;

procedure radix_sort(var co: array of zakaznik; kolko, cifra: longint);
var i : longint;
    c : array[0..9] of longint;
begin
   for i:=0 to 9 do c[i]:=0;
   for i:=0 to kolko-1 do inc( c[ ord(co[i].cislo[cifra])-48 ] );
```

```
   for i:=1 to 9 do c[i]:=c[i-1]+c[i];
   for i:=9 downto 1 do c[i]:=c[i-1]; c[0]:=0;
   for i:=0 to kolko-1 do begin
      pomocne[ c[ ord(co[i].cislo[cifra])-48 ] ] := co[i];
      inc( c[ ord(co[i].cislo[cifra])-48 ] );
   end;
   for i:=0 to kolko-1 do co[i] := pomocne[i];
end;

procedure nacitaj_zakaznikov;
var c : char;
    i, j : longint;
begin
   readln(N);
   for i:=0 to N-1 do begin
      zakaznici[i].cislo:='';
      for j:=1 to 10 do begin
         read(c);
         zakaznici[i].cislo := zakaznici[i].cislo+c;
      end;
      read(c);
      readln(zakaznici[i].meno);
      zakaznici[i].cas := 0;
   end;
   for i:=10 downto 1 do radix_sort(zakaznici,N,i);
end;

procedure nacitaj_hovory;
var c : char;
    i, j, cas : longint;

begin
   readln(M);
   for i:=0 to M-1 do begin
      hovory[2*i].cislo:='';
      for j:=1 to 10 do begin
         read(c);
         hovory[2*i].cislo:=hovory[2*i].cislo+c;
      end;
      read(c);
      hovory[2*i+1].cislo:='';
      for j:=1 to 10 do begin
         read(c);
         hovory[2*i+1].cislo:=hovory[2*i+1].cislo+c;
      end;
      read(c);
      readln(cas);
      hovory[2*i].cas := cas;
      hovory[2*i+1].cas := cas;
   end;
```

```
   for i:=10 downto 1 do radix_sort(hovory,2*M,i);
end;

procedure spracuj;
var z, h : longint;
begin
   z := 0;
   for h:=0 to 2*M-1 do begin
      while (z < N) and (zakaznici[z].cislo < hovory[h].cislo) do inc(z);
      if z = N then break;
      if zakaznici[z].cislo = hovory[h].cislo then
         inc( zakaznici[z].cas, hovory[h].cas );
   end;
end;

procedure vypis_najlepsieho;
var i, vitaz : longint;
begin
   vitaz := 0;
   for i:=1 to N-1 do if zakaznici[i].cas > zakaznici[vitaz].cas then vitaz:=i;
   writeln(zakaznici[vitaz].cislo+' '+zakaznici[vitaz].meno);
end;

begin
   nacitaj_zakaznikov;
   nacitaj_hovory;
   spracuj;
   vypis_najlepsieho;
end.
```

Alternatívne vzorové riešenia:

Namiesto RadixSortu od najmenej významnej cifry ho môžeme implementovať aj od najvýznamnejšej pomocou rekurzie. Alebo aj bez rekurzie na dva prechody: rozdelíme si každé číslo na prvých 5 a druhých 5 cifier. Najskôr utriedime čísla podľa ich prvých 5 cifier rovnako, ako v jednej fáze nášho RadixSortu, následne nájdeme úseky s rovnakými prvými 5 ciframi a každý z nich utriedime ešte raz podľa zvyšných 5 cifier. (Rovnako sa dal aj RadixSort zo vzorového riešenia upraviť na 2 prechody namiesto 10.)

Úplne iný spôsob, ako dosiahnuť optimálnu časovú zložitosť, je použiť na zapamätanie zoznamu zákazníkov *písmenkový strom* (trie). Táto dátová štruktúra je popísaná v riešení úlohy A-I-1 z minulého ročníka OI.

B-II-2 Kráľovské cesty

Táto úloha sa dala správne vyriešiť niekoľkými spôsobmi, prejdeme si postupne od menej bodovaných po viac bodované.

Lineárny čas pre jeden telefonát:

Toto riešenie bolo asi najjednoduchšie vymyslieť. Spomenieme len stručnú myšlienku. Stačilo si mestá reprezentovať ako graf, kde mestá sú vrcholy a cesty medzi mestami sú hrany patričnej dĺžky a potom pri každom telefonáte celý graf prehľadať a nájsť cestu medzi danými mestami.

O niečo lepšie riešenie sa dalo dosiahnuť tak aby sme pri prehľadávaní nepozreli celý graf, ale iba to, čo musíme (teda prejdeme len rádovo toľko vrcholov, koľko ich je na ceste z jedného mesta do druhého). To sa dalo spraviť tak, že z oboch miest pustíme bežcov smerom do hlavného mesta, pričom bežci budú rátať prejdenú vzdialenosť. Ak jeden z bežcov dorazí do druhého mesta, tak skončíme, lebo vzdialenosť miest je práve taká, akoľko prešiel ten bežec. Ak sa bežci stretnú v hlavnom meste, tak je výsledkom súčet prejdených vzdialeností. Toto riešenie má síce v najhoršom prípade časovú zložitosť $O(N)$, ale prakticky je lepšie ako predchádzajúce, lebo často sa stane, že bude potrebovať pozrieť na oveľa menej miest.

Konštantný čas pre jeden telefonát:

Konštantný čas sa dal dosiahnuť viacerými spôsobmi, ktoré sa líšili rýchlosťou predspracovania a pamäťovou zložitosťou. Prvý z nich má jednoduchú myšlienku. Vyrobíme si tabuľku $N \times N$, v ktorej budeme mať pre každú dvojicu miest zaznačenú vzdialenosť medzi nimi. Ak už máme tabuľku, tak pri každom telefonáte sa do nej pozrieme a povieme vzdialenosť medzi mestami.

Ako vypočítať takú tabuľku? Najjednoduchší spôsob je pre každú dvojicu miest vypočítať ich vzdialenosť pomocou algoritmu z predchádzajúceho odseku. Takých dvojíc je približne N^2, jedno hľadanie trvá $O(N)$, takže predspracovanie bude trvať $O(N^3)$. Pamäťová zložitosť bude $O(N^2)$, lebo vzdialenosti všetkých dvojíc miest si musíme pamätať.

Tento algoritmus sa dá ľahko vylepšiť tak, že predspracovanie bude trvať $O(N^2)$: stačí pre každý začiatok cesty spustiť prehľadávanie len raz, a keď skončí, pre každé iné mesto si zapamätáme vzdialenosť doň.

Konštantný čas pre telefonát, lineárny pre predspracovanie:

V tomto riešení využijeme to, že všetky cesty v kráľovstve začínajú v hlavnom meste a potom sa už vôbec nekrižujú. Predstavme si, že by sme o každom

meste vedeli, na ktorej z týchto ciest leží a zároveň by sme vedeli vzdialenosť mesta od hlavného mesta. Teda pre každé mesto a si budeme pamätať hodnotu `a.vzd`, čo je vzdialenosť mesta a od hlavného mesta a hodnotu `a.cesty`, čo je označenie cesty, na ktorej a leží. Ako bude vyzerať jeden telefonát? Dostaneme teraz dve mestá a a b a chceme vedieť ich vzdialenosť. Ak ležia na rôznych cestách (`a.cesta`$\neq$`b.cesta`), tak ak chceme ísť z a do b musíme ísť cez hlavné mesto a preto vzdialenosť miest je súčtom vzdialeností miest a a b od hlavného mesta, čo vieme vypočítať v konštantnom čase.

Ak a aj b ležia na tej istej ceste (`a.cesta=b.cesta`), tak ich vzdialenosť bude rozdielom ich vzdialeností od hlavného mesta, čo vieme tiež vypočítať v konštantnom čase. Jedinou otázkou je, ako získať tieto údaje.

Začneme tým, že nájdeme tri mestá, v ktorých jednotlivé cesty končia – toto sú práve tie tri mestá, ktoré nie sú ani raz spomenuté na vstupe (lebo jedine oni nemajú žiadneho suseda, ktorý by od nich bol ďalej).

Keď teraz poznáme mesto, ktorým cesta končí, vieme, že na nej leží aj jeho sused, za ním je sused toho suseda, a tak ďalej, až kým sa nedostaneme ku hlavnému mestu. Toto spravíme pre každú z ciest samostatne.

A keď už pre cestu poznáme všetky mestá, ktoré na nej ležia, môžeme po nej prejsť ešte raz – tentokrát v opačnom smere – a pre každé z nich zistiť, ako je ďaleko od hlavného mesta.

Listing programu:

```
type mesto = record
   sused, vzdialenost : longint; { udaje zo vstupu }
   cesta, do_hlavneho : longint; { cislo cesty a vzdialenost do hl. m. }
   koniec : boolean;             { konci v tomto meste cesta? }
end;

var N : longint;
    mesta : array of mesto;
    dlzky : array[1..3] of longint;          { dlzky jednotlivych ciest }
    cesty : array[1..3] of array of longint; { cisla miest na nich }
    f : text;
    i,x,y : longint;

begin
   { nacitame mesta }
   assign(f,'mapa.txt'); reset(f);
   readln(f,N);
   setlength(mesta,N);
   mesta[0].do_hlavneho := 0;
   for i:=1 to N-1 do mesta[i].koniec := true;
   for i:=1 to N-1 do begin
```

```
      readln(f,x,y);
      mesta[i].sused := x-1;
      mesta[i].vzdialenost := y;
      mesta[x-1].koniec := false;
   end;

   { zistime kde koncia cesty a zostrojime ich }
   for i:=1 to 3 do setlength(cesty[i],N);
   x := 0;
   for i:=1 to N-1 do
      if mesta[i].koniec then begin
         inc(x);
         cesty[x][0] := i;
         y := 1;
         while true do begin
            cesty[x][y] := mesta[ cesty[x][y-1] ].sused;
            if cesty[x][y] = 0 then break;
            inc(y);
         end;
         dlzky[x] := y;
      end;

   { prejdeme po kazdej ceste od hlavneho mesta a zistime do_hlavneho }
   for x:=1 to 3 do begin
      for y:=dlzky[x]-1 downto 0 do begin
         mesta[ cesty[x][y] ].cesta := x;
         mesta[ cesty[x][y] ].do_hlavneho :=
            mesta[ cesty[x][y+1] ].do_hlavneho
            + mesta[ cesty[x][y] ].vzdialenost;
      end;
   end;

   { odpovedame na otazky }
   while not eof do begin
      readln(x,y); dec(x); dec(y);
      if mesta[x].cesta = mesta[y].cesta then
         writeln( abs( mesta[x].do_hlavneho - mesta[y].do_hlavneho ) )
      else
         writeln( mesta[x].do_hlavneho + mesta[y].do_hlavneho );
   end;
end.
```

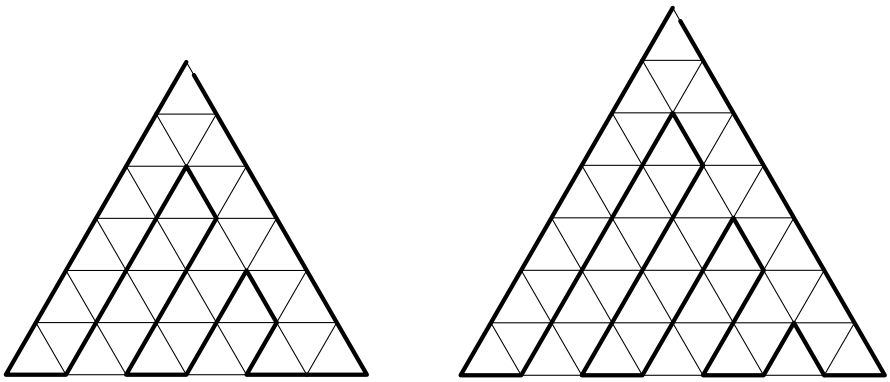
B-II-3 Triplex

Každú podúlohu budeme riešiť samostatne, obe riešenia ale budú mať jedno spoločné: Kľúčom k úspechu bude zvoliť si z mnohých možných riešení také, ktoré sa nám bude čo najjednoduchšie programovať.

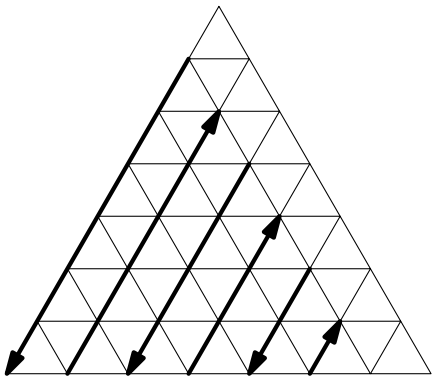
Podúloha a: prejsť všetky vrcholy:

V teórii grafov sa trasa, ktorú má naša figúrka prejsť (teda trasa prechádzajúca práve raz cez každý vrchol) volá *Hamiltonovská kružnica*. Vo všeobecnosti je jej hľadanie veľmi ťažký problém a nie je preň známy žiaden efektívny algoritmus. My ale našťastie máme veľmi špecifický hrací plán, pre ktorý to vždy ide.

Riešenie, ktoré sme si vybrali, si najskôr ukážeme pre $N = 6$ a $N = 7$:



Toto riešenie vieme veľmi ľahko zovšeobecniť pre ľubovoľné N. Všimnime si na nasledujúcom obrázku, ako presnejšie funguje „cik-cak“ časť našej cesty:



Znázornené šípky sú postupne: $N - 1$ krokov dole vľavo, $N - 2$ krokov hore vpravo, $N - 3$ krokov dole vľavo, a tak ďalej. Toto ľahko prepíšeme do cyklu.

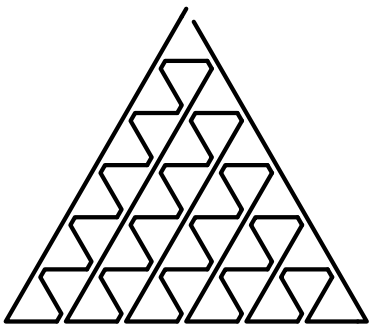
Jediné, na čo si ešte potrebujeme dať pozor, je úplný koniec. Všimnite si, že v okolí pravého dolného rohu sa riešenia pre $N = 6$ a $N = 7$ mierne líšia.

Rovnako ako $N = 6$ bude vyzerať riešenie pre každé párne N a rovnako ako $N = 7$ bude vyzerať riešenie pre každé nepárne N.

Listing programu:

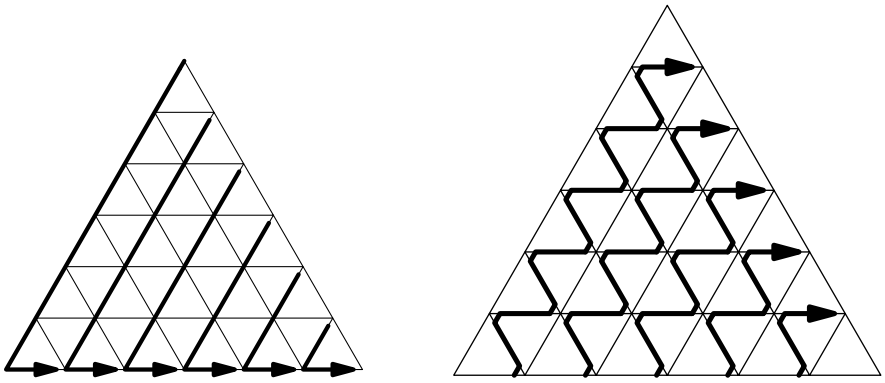
```
var N, i, j : longint;
begin
   readln(N);
   write(4);
   for i:=1 to N-1 do begin
      if i mod 2 = 1 then begin
         for j:=1 to N-i do write(4);
         write(0);
      end else begin
         for j:=1 to N-i do write(1);
         write(5);
      end;
   end;
   write(0);
   for i:=1 to N do write(2);
   writeln;
end.
```

Podúloha b: prejsť všetky hrany:



V teórii grafov sa tento druh trasy volá *Eulerovský ťah*. Na rozdiel od Hamiltonovskej kružnice, na hľadanie Eulerovského ťahu existujú všeobecné efektívne algoritmy. Jedna možná myšlienka je, že začneme s nejakou trasou, ktorá ide každou hranou najviac raz (napríklad prázdnou), a kým neobsahuje všetky hrany, tak dokola opakujeme: nájdeme nejaký vrchol, v ktorom ešte máme nepoužitú hranu. Tam našu trasu „rozpojíme“ a vložíme do nej nový kus, ktorý začína aj končí v uvedenom vrchole.

Nebudeme tu ale zachádzať do detailov, pretože pre našu úlohu existuje aj jednoduchšie riešenie – opäť si stačí zvoliť vhodnú, pravidelnú trasu. Napríklad tú, ktorá je znázornená na obrázku vpravo. V tejto trase sa striedajú presuny priamo doľava dole a presuny „cik-cak“ naspäť doprava hore. Lepšie to uvidíme, keď si trasu prekreslíme do dvoch obrázkov:



Listing programu:

```
var N, i, j : longint;
begin
   readln(N);
   for i:=N downto 1 do begin
      for j:=1 to i do write(4);
      write(0);
      for j:=1 to i-1 do write(20);
   end;
   for i:=1 to N do write(2);
   writeln;
end.
```

B-II-4 Gaštanový snehuliak

Podobne ako v domácom kole ukážeme, že pre Tomáška existuje stratégia, pri ktorej bude počet otázok, ktoré mu stačia, lineárny od počtu gaštanov N.

Základné úvahy:

Základný „stavebný kameň“ nášho riešenia bude taktiež rovnaký: Tomáško si vyberie nejaký gaštan a opýta sa $N-1$ otázok, aby zistil, s koľkými a ktorými inými gaštanmi je ten jeho gaštan spojený.

Gaštan nazveme *obyčajný*, ak patrí do práve jednej z troch gulí nášho snehuliaka a nie je spojený s nosom. Sú práve štyri gaštany, ktoré nie sú obyčajné: jeden je nos, druhý je jeho jediný sused, tretí je spoločný gaštan hornej a strednej a štvrtý je spoločný gaštan strednej a dolnej gule. Tieto štyri gaštany budeme volať *špeciálne*.

Pre daný gaštan g nazveme *okolím* g množinu obsahujúcu gaštan g a všetky gaštany, ktoré sú s ním spojené zápalkou. Zistiť ako vyzerá okolie gaštanu g vieme položením $N-1$ otázok.

Oplatí sa, skôr než začneme hrať hru, spraviť nasledujúce pozorovania:

- Žiadne dva špeciálne gaštany nemajú rovnaké okolie.
- Žiaden obyčajný gaštan nemá rovnaké okolie ako žiaden špeciálny gaštan.
- Dva obyčajné gaštany majú rovnaké okolie práve vtedy, ak patria do tej istej gule.

Riešenie:

Budeme postupovať nasledovne. Najskôr nájdeme jednu guľu:

- Vyberieme si ľubovoľných 8 gaštanov. Ku každému z nich nájdeme jeho okolie.
- Vyberieme spomedzi našich gaštanov dva, ktoré majú rovnaké okolie A. (Keďže len 4 gaštany sú špeciálne, medzi našimi 8 gaštanmi sú aspoň 4 obyčajné. A keďže snehuliak má len tri gule, z niektorej gule máme aspoň dva obyčajné gaštany. A tieto dva gaštany majú určite rovnaké okolie.)
- Okolie A je presne jedna z gulí nášho snehuliaka.

Na $8(N-1)$ otázok sme teda našli prvú guľu. Nevieme ešte, ktorá to je, ale vieme, že určite obsahuje aspoň jeden špeciálny gaštan. Na nájdenie druhej gule nám teda bude stačiť aj menej otázok:

- Vyberieme si ľubovoľných 6 gaštanov, ktoré nepatria do A. Ku každému z nich nájdeme jeho okolie.
- Vyberieme spomedzi našich gaštanov dva, ktoré majú rovnaké okolie B.
- Okolie B je druhá z gulí nášho snehuliaka.

A pre veľký úspech postup zopakujeme aj do tretice.

- Vyberieme si ľubovoľné 4 gaštany, ktoré nepatria do A ani do B. Ku každému z nich nájdeme jeho okolie.
  (V tomto kroku môže nastať jeden špeciálny prípad: ak sú A a B horná a dolná guľa, a ak je stredná guľa tvorená len 4 gaštanmi, ostali nám už len tri neidentifikované gaštany. V takomto prípade vyberieme len tieto tri.)
- Vyberieme spomedzi našich gaštanov dva, ktoré majú rovnaké okolie C.
- Okolie C je tretia z gulí nášho snehuliaka.

Podľa toho, ktoré dvojice okolí A, B a C majú neprázdny prienik, vieme povedať, ktorá guľa je stredná – tá, ktorá má spoločný gaštan s každou zo zvyšných dvoch.

A ešte nám ostal jeden gaštan, ktorý nepatrí do A, B, ani C. Tento gaštan je nos. A tá množina spomedzi A, B a C, ktorá obsahuje jeho suseda, je horná guľa.

Pre ľubovoľne veľké N si náš algoritmus vystačí s menej ako $18N$ otázkami, je teda lineárny od počtu gaštanov.

Alternatívne riešenie:

Existujú aj riešenia, ktoré si vystačia s o niečo menej otázkami ako to naše. Výhodou nášho riešenia ale je, že sa ľahko vysvetľuje a neobsahuje (takmer) žiadne špeciálne prípady.

Uvedieme ešte jeden iný spôsob, ako sa dalo túto hru vyhrať s lineárnym počtom otázok. Základná myšlienka tohto spôsobu je, že najskôr nájdeme nos, a od nosa potom postupne zostrojíme snehuliaka.

Najskôr popíšeme druhú časť riešenia. Akonáhle niekedy vieme, ktorý gaštan je nos, môžeme postupovať nasledovne: Jeho jediný sused patrí do hornej gule. Hornú guľu tvorí jeho okolie okrem nosa. Teraz vyberieme dva zo zvyšných gaštanov a ku každému nájdeme jeho okolie. Aspoň jeden z nich je obyčajný, a teda jeho okolie bude niektorá iná guľa. Podľa toho, či má spoločný gaštan s hornou, vieme, či je to stredná alebo dolná guľa. V oboch prípadoch si na záver vyberieme jeden ľubovoľný zo zvyšných gaštanov a nájdeme jeho okolie, aby sme presne vedeli aj tretiu guľu.

Jediné, čo ešte treba vymyslieť, je ako nájsť snehuliakov nos. Zjavne akonáhle nájdeme okolie nejakého gaštanu a zistíme, že má len jedného suseda, vieme, že je to nos. Rozoberieme teda len situácie, kedy žiaden z gaštanov, ktoré si vyberieme, nebude nos.

Nos budeme hľadať nasledovne: Vyberieme si prvý gaštan, nájdeme jeho okolie. Vyberieme si druhý gaštan, ktorý neleží v okolí prvého, a nájdeme jeho okolie.

Teraz mohli nastať tri prípady:

1. Obe okolia dokopy pokrývajú všetky gaštany okrem jedného. Vtedy ten jeden gaštan musí byť nos.
2. Obe okolia dokopy pokrývajú úplne všetky gaštany. Toto mohlo nastať jedine tak, že sme trafili dva špeciálne gaštany – suseda nosa a gaštan spoločný pre strednú a spodnú guľu. Vyberieme teda namiesto nich dva iné gaštany a už máme istotu, že tento prípad nenastal.

3. Ostalo viac gaštanov, ktoré do ani jedného z nájdených okolí nepatria. Tento prípad rozoberieme nižšie.

Teraz vyberieme jeden zo zvyšných vrcholov a nájdeme jeho okolie. A už sú len dva možné prípady. V tom lepšom zostal jeden gaštan, ktorý nesusedí so žiadnym z našich troch, a to musí byť nos. V tom horšom prípade naše tri okolia pokrývajú úplne všetky gaštany, lebo jeden z nich je sused nosa. Ak teda nastal tento prípad, tak postupne každý z troch našich gaštanov skúsime nahradiť iným, ktorý neleží v okolí zvyšných dvoch.

Riešenia celoštátneho kola kategórie A

A-III-1 Čokoláda je tu zas

Najskôr ukážeme riešenie s časovou zložitosťou $O(R^2S)$. Bude založené na jednoduchej myšlienke: vyskúšame všetky dvojice riadkov, a pre každú dvojicu v $O(S)$ spočítame všetky obdĺžniky, ktoré práve tam majú svoj horný a dolný riadok.

Keď sme si už zvolili horný a dolný riadok, máme pás políčok. Niektoré jeho stĺpce sú celé biele, tie môžeme použiť. A niektoré obsahujú aspoň jedno sivé políčko, a tie použiť nemôžeme.

Ak by sme vedeli, ktoré stĺpce použiť môžeme, a ktoré nie, dostávame vlastne jednorozmernú verziu pôvodnej úlohy: máme riadok s nulami a jednotkami a chceme spočítať počet úsekov, ktoré sú tvorené len jednotkami.

To spravíme tak, že pre každé miesto zistíme počet úsekov jednotiek, ktoré na tom mieste končia, a všetky tieto počty sčítame. Počet úsekov jednotiek, ktoré na danom mieste končia, je zjavne rovný počtu jednotiek, ktoré uvidíme, keď pôjdeme z daného miesta doľava po najbližšiu nulu (alebo začiatok). A tento počet si vieme priebežne počítať, ako spracúvame čísla v riadku – vždy, keď spracujeme jednotku, zvýšime si počítadlo, a vždy, keď spracujeme nulu, počítadlo vynulujeme.

Takto teda celý riadok spracujeme v čase lineárnom od jeho dĺžky, čiže $O(S)$.

Zostáva doriešiť, odkiaľ vezmeme informáciu o tom, ktoré stĺpce môžeme použiť a ktoré nie. Na to nám stačí spracúvať dvojice riadkov v systematickom poradí. Pre dvojicu (r_1, r_1) túto informáciu máme „zadarmo“ priamo vo vstupe. A keď pre dvojicu (r_1, r_2) vieme, ktoré stĺpce sa ešte dajú použiť, pre dvojicu $(r_1, r_2 + 1)$ túto informáciu ľahko zistíme – sú to tie stĺpce, ktoré sa dali použiť pre (r_1, r_2), a zároveň majú jednotku aj v riadku $r_2 + 1$.

Listing programu:

```
#include <iostream>
using namespace std;

int R, S, A[5012][5012], zije[5012];

int main() {
  cin >> R >> S;
```

```
  for (int r=0; r<R; r++) for (int s=0; s<S; s++) cin >> A[r][s];
  long long result = 0;
  for (int r1=0; r1<R; r1++) {
    for (int s=0; s<S; s++) zije[s]=1;
    for (int r2=r1; r2<R; r2++) {
      for (int s=0; s<S; s++) zije[s] &= A[r2][s];
      int run=0;
      for (int s=0; s<S; s++) if (zije[s]) result += (++run); else run=0;
    }
  }
  cout << result << endl;
}
```

Vzorové riešenie:

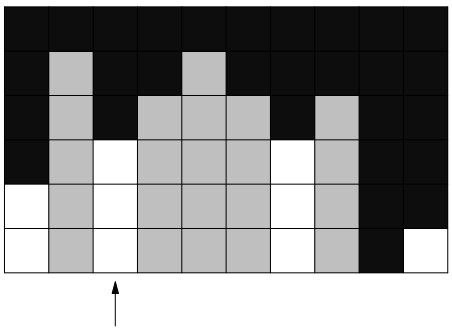
Teraz ukážeme jedno možné riešenie s optimálnou časovou zložitosťou $O(RS)$.

Naše riešenie bude založené na nasledujúcej základnej myšlienke: Postupne pre každý riadok spočítame všetky obdĺžniky, ktoré práve v ňom majú svoj dolný riadok.

Keď spracúvame nejaký riadok, budeme o každom jeho políčku potrebovať vedieť, akú má *výšku* – t. j. aká dlhá dohora idúca súvislá postupnosť jednotiek na ňom začína. Výšky pre nový riadok si vieme spočítať z výšok pre o jedno vyšší riadok rovnako, ako sme si počítali v predchádzajúcom riešení to, ktoré stĺpce ešte môžeme použiť.

Predstavme si, že teraz zoberieme všetkých S stĺpcov a utriedime ich od najvyššieho po najnižší. V tomto poradí ich teraz budeme spracúvať. Vždy, keď spracujeme stĺpec, zarátame všetky obdĺžniky, ktoré práve pribudli – teda tie, ktoré majú dolnú stranu na práve spracúvanom riadku, obsahujú aspoň jedno políčko tohto stĺpca, a celé ležia v už spracovaných stĺpcoch.

Zjavne takto každý obdĺžnik zarátame práve raz – vtedy, keď spracujeme posledný zo stĺpcov, v ktorých leží. Zostáva už len jediné – prísť na to, ako tieto počty obdĺžnikov efektívne určiť.



Situácia uprostred spracúvania riadku s výškami $2, 5, 3, 4, 5, 4, 3, 4, 0, 1$.
(Svetlou sivou sú už spracované stĺpce, šípkou je označený práve spracúvaný.)

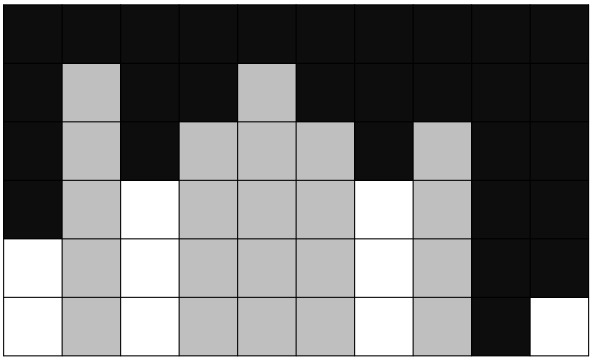
Keď spracúvame nejaký stĺpec, potrebujeme vedieť povedať počet jemu priradených obdĺžnikov. Keďže každý z nich má už pevne zvolenú spodnú stranu, je každý určený tromi hodnotami: výškou a tým, ako ďaleko doľava a ako ďaleko doprava od práve spracúvaného stĺpca siaha.

To, ako ďaleko doľava a doprava môže siahať, je samozrejme nanajvýš rovné počtu už spracovaných stĺpcov bezprostredne naľavo a napravo. Napríklad v situácii na obrázku by pri spracúvaní stĺpca označeného šípkou mohli obdĺžniky siahať až 1 políčko doľava a 3 doprava.

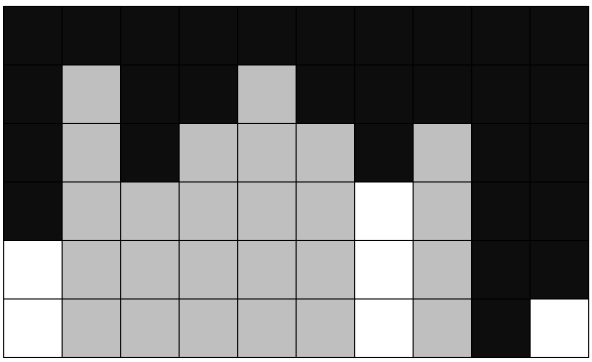
Uvedomme si teraz, že všetky už spracované stĺpce sú aspoň tak vysoké ako ten aktuálny. A teda každej prípustnej kombinácii hodnôt (výška, počet políčok doľava, počet políčok doprava) naozaj zodpovedá platný obdĺžnik. (Toto je dôvod, prečo stĺpce spracúvame práve usporiadané podľa výšky a nie v inom poradí.) A teda počet obdĺžnikov priradených aktuálnemu stĺpcu vieme zistiť jednoducho vynásobením týchto troch hodnôt.

Zostáva doriešiť, odkiaľ efektívne zistíme, ako ďaleko doľava a doprava môžu siahať obdĺžniky pre aktuálny stĺpec. To sa dá spraviť jednoducho. Všimnime si, že v každom okamihu tvoria už spracované stĺpce niekoľko súvislých úsekov. Ku každému súvislému úseku si budeme pamätať dve čísla: na jeho ľavom okraji ako ďaleko doprava, a na jeho pravom okraji ako ďaleko doľava siaha. Takto sa pri spracúvaní stĺpca v konštantnom čase dozvieme informácie, ktoré potrebujeme (sú uložene v jeho bezprostredných susedoch), a takisto po jeho spracovaní vieme tieto informácie v konštantnom čase upraviť – poznáme začiatok aj koniec úseku obsahujúceho práve spracovaný stĺpec, tak do jeho koncov zapíšeme jeho novú dĺžku.

Na nasledujúcich obrázkoch nájdete pamätané informácie pred spracovaním a po spracovaní šípkou označeného stĺpca z predchádzajúceho obrázku.



dolava		1				3		1	
doprava		1		3				1	



Ešte raz teda zhrnieme naše riešenie. Postupne pre každý z R riadkov spravíme nasledujúce veci: Najskôr v čase $O(S)$ spočítame výšky pre tento riadok. Potom utriedime všetky stĺpce podľa výšky – keďže výšky sú celé čísla od 0 po R, toto vieme spraviť v čase $O(R+S)$ napríklad priehradkovým triedením (bucket sort). Následne pre každý stĺpec v konštantnom čase zistíme počet jemu priradených obdĺžnikov, a tiež v konštantnom čase upravíme pamätané dĺžky súvislých úsekov. Aj táto fáza spracovania riadku teda beží v čase $O(S)$.

Celková časová zložitosť je teda $O(R(R+S))$. A keďže môžeme predpokladať, že $R \leq S$ (inak vstup transponujeme, čo počet obdĺžnikov nezmení), je toto rovné $O(RS)$. Táto zložitosť je zjavne optimálna, keďže musíme načítať takto veľký vstup.

Listing programu:

```
#include <algorithm>
#include <cstdio>
using namespace std;

int R, S;
int A[5012][5012], vedierka[5012][5012];
int vyska[5012], dolava[5012], doprava[5012], pocty[5012];

int main() {
  scanf("%d %d ",&R,&S);
  for (int r=1; r<=R; r++) for (int s=1; s<=S; s++) scanf("%d",&A[r][s]);
  long long result = 0;
  for (int r=1; r<=R; r++) {
    for (int s=1; s<=S; s++) if (A[r][s]) vyska[s]++; else vyska[s]=0;
    fill(dolava,dolava+S+2,0);
    fill(doprava,doprava+S+2,0);
    fill(pocty,pocty+r+2,0);
    for (int s=1; s<=S; s++) vedierka[ vyska[s] ][ pocty[vyska[s]]++ ] = s;
    for (int v=r; v>0; v--) for (int ss=0; ss<pocty[v]; ss++) {
```

```
      int s = vedierka[v][ss];
      result += v * (dolava[s-1]+1) * (doprava[s+1]+1);
      int vlavo=s-dolava[s-1], vpravo=s+doprava[s+1], dlzka=vpravo-vlavo+1;
      doprava[vlavo]=dolava[vpravo]=dlzka;
    }
  }
  printf("%Ld\n",result);
  return 0;
}
```

A-III-2 Odveta

Ako každá matematická hra[5], aj naša hra s koníkmi je v každom momente v nejakej pozícii (stave). Pozíciu vieme jednoznačne popísať tak, že uvedieme aktuálne súradnice všetkých koníkov.

Pod *vyhávajúcou stratégiou* budeme rozumieť postup, ktorý hráčovi zaručí víťazstvo bez ohľadu na ťahy protihráča. Pozícia je *vyhávajúca*, ak hráč, ktorý je na ťahu, má vyhávajúcu stratégiu. Pozícia, ktorá nie je vyhávajúca, je *prehrávajúca*. Zjavne každá pozícia je buď vyhrávajúca alebo prehrávajúca.

Pozíciu, z ktorej neexistuje ťah, nazývame *koncová*. V našej hre sú podľa pravidiel všetky koncové pozície prehrávajúce.

Keďže v súťažnej úlohe máme proti sebe optimálne ťahajúceho protihráča, zaujíma nás presne to, či je začiatočná pozícia pre nás vyhrávajúca, alebo nie. Ako to určiť?

Pri charakterizácii pozícii si môžeme pomôcť nasledovnými myšlienkami:

1. Ak je pozícia koncová, je prehrávajúca.
2. Ak z danej pozície všetky ťahy vedú do vyhrávajúcich pozícií, tak je táto pozícia prehrávajúca.
3. Ak z danej pozície existuje ťah do prehrávajúcej, tak je táto pozícia vyhrávajúca.

(Ak všetky ťahy vedú do vyhrávajúcich pozícií, nech si vyberieme ktorýkoľvek, vždy tým dostaneme súpera do vyhrávajúcej pozície. A ak sa potom bude súper držať nejakej vyhrávajúcej stratégie, hru prehráme. Preto takáto pozícia je prehrávajúca. Naopak, ak existuje ťah do prehrávajúcej pozície, spravíme ho, a tým dostaneme súpera do tejto, pre neho prehrávajúcej pozície.)

[5] Presnejšie, konečná kombinatorická hra s úplnou informáciou.

Túto myšlienku ľahko prepíšeme do rekurzívnej funkcie, ktorá nám o pozícii povie, či je vyhrávajúca alebo prehrávajúca.

Hra s jedným alebo dvomi koníkmi:

Podľa nenápadnej rady v zadaní sa teraz zamyslime nad jednoduchšou verziou hry a uvažujme, že na šachovnici je len jeden koník.

Problém predchádzajúceho prístupu spočíva v tom, že je príliš pomalý. Hlavný dôvod je ten, že pri rekurzívnych volaniach vlastne skúša všetky možné priebehy hry, a pri tom mnohé pozície vyhodnotí veľa krát.

Tu si môžeme pomôcť takzvanou memoizáciou – akonáhle o nejakej pozícii zistíme, či je vyhrávajúca alebo prehrávajúca, zapíšeme si to do pomocného poľa. Takto dosiahneme to, že každú pozíciu budeme spracovávať práve raz a riešenie vykoná pre každú spracovávanú pozíciu konštantný počet operácii.

A koľko pozícií budeme spracovávať? Všimnime si, že každým ťahom sa priblížime k políčku $(0, 0)$. Presnejšie, súčet súradníc sa nám zníži aspoň o jedna. To znamená, že počet spracovávaných stavov môžeme zhruba zhora odhadnúť počtom bodov, ktoré majú súčet súradníc menší alebo rovnaký ako počiatočná pozícia. V prípade, že počiatočná pozícia je (X, Y), potom je týchto pozícií $O((X + Y)^2)$, a taká je aj časová zložitosť tohto algoritmu.

Uvedené myšlienky sa dajú veľmi jednoducho rozšíriť aj viacero koníkov. Ak máme N koníkov a každý začína na súradniciach so súčtom nanajvýš S, tak je počet dosiahnuteľných pozícií rádovo rovný S^{2N}. Takéto riešenie je teda použiteľné len pre veľmi malé hodnoty N.

Rýchlejšia charakterizácia pozícií pre jedného koníka:

Vráťme sa k najjednoduchšej možnej hre s jediným koníkom.

Ak chceme vedieť rýchlejšie povedať, ktorá pozícia je vyhrávajúca a ktorá prehrávajúca, potom by sme mali nájsť charakterizáciu pozícii, ktorá nevyužíva vedomosti o pozíciach naokolo. Veľmi často pomáha skúsiť si vypočítať pre niekoľko najmenších pozícií, či sú vyhrávajúce, a hľadať nejakú závislosť.

Ak by sme v našom prípade vyhodnotili všetky pozície, ktorých súčet súradníc nepresahuje 13, dostali by sme nižšie uvedenú tabuľku.

Políčko $(0, 0)$ sa nachádza v jej ľavom dolnom rohu. Nulou sme označili prehrávajúce pozície, jednotkou vyhrávajúce. Nie je obtiažne spozorovať vzorku – celá šachovnica je zložená z dlaždíc veľkosti 4×4, na ktorých sú 4 políčka vľavo dole prehrávajúce a zvyšné vyhrávajúce. Preto môžeme vysloviť nasledujúcu hypotézu: Pozícia (X, Y) je prehrávajúca práve vtedy, ak $X \bmod 4 \in \{0, 1\}$ a $Y \bmod 4 \in \{0, 1\}$.

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

Túto hypotézu dokážeme matematickou indukciou podľa súčtu $X + Y$. Pre malé hodnoty sme už toto tvrdenie dokázali zostrojením vyššie uvedenej tabuľky, stačí sa teda sústrediť na indukčný krok.

Majme teda nejakú pozíciu (X, Y), pričom o všetkých pozíciách, ktoré majú súčet súradníc menší ako $X + Y$, už vieme, že naša hypotéza pre ne platí. Dokážeme, že platí aj pre (X, Y). Rozoberieme niekoľko možností podľa toho, aké zvyšky po delení 4 dávajú X a Y.

- Obe súradnice dávajú zvyšok 0 alebo 1:

  Nech potiahneme ľubovoľne, jednu súradnicu vždy zmeníme o 2 a druhú o 1. No a tá súradnica, ktorú sme zmenili o 2, bude teraz dávať po delení 4 zvyšok 2 alebo 3. A teda podľa indukčného predpokladu bude táto nová pozícia pre hráča na ťahu vyhrávajúca.

  Tým sme zdôvodnili, že nech potiahneme akokoľvek, vždy dostaneme súpera do vyhrávajúcej pozície. Naša pozícia je teda prehrávajúca, čo sme aj chceli dokázať.

- Jedna súradnica dáva zvyšok 2 alebo 3, druhá 0 alebo 1:

  Potiahneme tak, že prvú súradnicu zmenšíme o 2.
  Ak druhá dáva zvyšok 1, tak ju o 1 zmenšíme, inak ju o 1 zväčšíme.

- Obe súradnice dávajú zvyšok 2: Zmenšíme X o 2 a Y o 1.

- Obe súradnice dávajú zvyšok 3: Zmenšíme X o 2 a zväčšíme Y o 1.
- Jedna súradnica dáva zvyšok 3 a druhá 2: Prvú súradnicu zmenšíme o 2 a druhú o 1.

  Vo všetkých štyroch prípadoch sme ukázali ťah, ktorým súpera dostaneme do pozície, ktorá je podľa indukčného predpokladu prehrávajúca. Z čoho vyplýva, že naša pozícia musela zakaždým byť vyhrávajúca, č.b.t.d.

Pre hru s jedným koníkom teda vieme v konštantnom čase povedať, či je pozícia vyhrávajúca alebo nie.

Všeobecná hra:

Čo ale v prípade, ak máme koníkov viac? Pri hľadaní všeobecného riešenia pomôže, ak sa budeme pozerať na vzor vyššie a skúšať sa po ňom hýbať dvoma, prípadne iným malým počtom koníkov. Takto môžeme objaviť nasledujúcu charakterizáciu pozícií:

- Ak sú všetky koníky na políčku s nulou, tak je pozícia prehrávajúca.
- Ak existuje koník, ktorý je na políčku s jednotkou, tak je pozícia vyhrávajúca.

Dôkaz: Pozíciu, kde sú všetky koníky na políčkach s nulou, budeme volať modrá, ostatné pozície budeme volať červené. Ľahko nahliadneme, že platia nasledujúce tri tvrdenia:

1. Všetky koncové pozície hry sú modré.
2. Ak sme v modrej pozícii, tak ľubovoľným ťahom z nej vyrobíme červenú pozíciu.

   Musíme totiž pohnúť aspoň jedného koníka. A keďže v hre s jedným koníkom všetky ťahy z prehrávajucej pozície (t. j. políčka s nulou) vedú do vyhrávajúcich pozícií (na políčka s jednotkou), koník, ktorého pohneme, skončí na políčku s jednotkou, a teda nová pozícia bude červená.
3. Ak sme v červenej pozícii, tak z nej vieme platným ťahom vyrobiť modrú pozíciu.

   Vyberieme všetky koníky, ktoré stoja na políčkach s jednotkou. Keďže tieto políčka dostali jednotky, z každého z nich existuje skok na nejaké políčka s nulou. Takéto skoky porobíme, čím dosiahneme, že výsledná pozícia bude modrá.

Teraz vidíme, že modré a červené pozície presne spĺňajú našu definíciu vyhrávajúcich a prehrávajúcich pozícií. Nutne sú teda všetky modré pozície prehrávajúce a všetky červené vyhrávajúce.

Keď máme situáciu s N koníkmi, potom stačí len pre každého v konštantnom čase zistiť, či je na políčku s nulou alebo na políčku s jednotkou. V prípade, že existuje koník na políčku s jednotkou, pozícia je vyhrávajúca. V takom prípade vieme o každom jazdcovi na jednotke zistiť v konštantom čase, kam ho treba presunúť. Časová zložitosť tohto riešenia je teda $O(N)$. Všimnite si, že vieme koníky spracúvať po jednom, a teda nám stačí konštantná pamäť.

Listing programu:

```
#include <iostream>
using namespace std;
int N,x,y;
int dx[4] = {-2,-2,-1,1}, dy[4] = {1,-1,-2,-2}; // povolene tahy kona

//metoda zisti, ci je konik na prehravajucej pozicii v hre s 1 konom
bool prehravajuci(int x, int y){
  return  (x%4 < 2) && (y%4 < 2);
}

int main(){
  cin >> N;
  int prehra=1;
  for (int i=0;i<N;i++){
    cin >> x >> y;
    if ( !prehravajuci(x,y) ){
      if (prehra == 1) cout << "Cedecko" << endl;
      prehra = 0;
      //treba pohnut kazdym konikom, ktory nie je na prehravajucej pozicii
      cout << x << " " << y << " ";
      for (int j=0;j<4;j++){
        int nx = x + dx[j], ny = y + dy[j];
        if ( nx < 0 || ny < 0 ) continue;
        if ( prehravajuci(nx,ny) ){
          cout << nx << " " << ny << endl;
          break;
        }
      }
    }
  }
  if (prehra == 1) cout << "Petrzlen" << endl;
  return 0;
}
```

A-III-3 Počítač s gumenou rúrou

Podúloha a: Kontrola, či je postupnosť rastúca:

Začneme obľúbeným trikom: na koniec postupnosti si ako zarážku vložíme číslo 0. Teraz načítame do registra a prvé číslo postupnosti a za zarážku vložíme číslo 1.

Od tohto okamihu ďalej budeme čísla z rúry striedavo načítavať do registrov b a a. Vždy, keď načítame nové číslo, porovnáme ho s predchádzajúcim. Ak je to nové väčšie, vložíme do rúry ďalšie číslo 1. Ak nie, našli sme práve hľadanú prvú chybu. Všimnime si, že počet 1, ktoré sme doteraz do rúry vložili, je presne rovný indexu, ktorý máme vypísať. Stačí teda vyprázdniť z rúry všetko po zarážku a následne všetky 1 sčítať, aby sme dostali odpoveď, a túto vypíšeme.

V opačnom prípade, teda ak sa nám podarí dočítať celú zadanú postupnosť až po zarážku, jednoducho program ukončíme.

Celý program môže vyzerať napr. nasledovne:

```
get a ; put 0
jz a koniec
jump citaj_b

label citaj_b
  put 1 ; get b
  jz b koniec
  jgt b a citaj_a
jump vyprazdni

label citaj_a
  put 1 ; get a
  jz a koniec
  jgt a b citaj_b
jump vyprazdni

label vyprazdni
  get a ; jz a scitaj
jump vyprazdni

label scitaj
  get a ; jempty vypis ; put a ; add
```

```
jump scitaj

label vypis
  put a ; print
jump koniec

label koniec
```

Podúloha b: Hľadanie majoritného prvku:

Prvé jednoducho naprogramovateľné riešenie bude založené na nasledujúcej myšlienke: Nech a a b sú dva prvky zadanej postupnosti, ktoré sú navzájom rôzne. Potom ak oba tieto prvky vynecháme, dostaneme opäť postupnosť, ktorá má majoritný prvok (a je to ten istý prvok ako predtým).

Prečo? Jednoducho preto, že sme vyhodili nanajvýš jeden z prvkov, ktorých bola väčšina, a aspoň jeden z prvkov, ktorých väčšina nebola.

Náš program bude teda dokola opakovať nasledujúci postup: Číta postupnosť a overuje, či sú všetky prvky rovnaké. Ak sa dočíta na koniec a zistí že sú, jeden z nich vypíše a skončí. V opačnom prípade nájde dvojicu rôznych prvkov. Túto odstráni, postupnosť dočíta do konca a začne odznova.

Tento program bude mať časovú zložitosť kvadratickú od dĺžky zadanej postupnosti. Totiž každú iteráciu uvedeného postupu vieme vykonať v čase lineárnom od aktuálnej dĺžky postupnosti a platí, že jedným jej vykonaním zmenšíme dĺžku postupnosti o 2.

(Síce by sme mohli postupne vyhadzovať aj viac dvojíc, ak ich nájdeme postupne, ale v najhoršom možnom prípade bude aj tak takéto riešenie potrebovať kvadraticky veľa krokov – napr. pre postupnosť obsahujúcu k jednotiek a následne $k+1$ dvojok. Preto radšej zostaneme pri ľahšie napísateľnom programe.)

```
put 0

label kolo
  get a
  label loop
    get b ; jz b koniec
    jeq a b dalej
  jump docitaj

  label dalej
```

```
    put b
  jump loop

  label docitaj
    get a ; jz a docital ; put a
  jump docitaj

  label docital
    put 0
jump kolo

label koniec
put a ; print
```

Existuje však aj lepšie riešenie. Uvažujme najskôr situáciu, kedy je celková dĺžka n našej postupnosti párna. Rozdelíme našu postupnosť na $n/2$ párov. V niektorých z nich sú obe čísla rovnaké, v niektorých sú rôzne. Už sme si zdôvodnili, že každý pár, kde sú čísla rôzne, môžeme zahodiť. Zostane nám teda niekoľko párov, v ktorých sú čísla rovnaké. Čo s nimi? Z každého páru jedno číslo necháme a to druhé zahodíme.

Rozmyslite si, že touto operáciou nič nepokazíme – ak mal nejaký prvok väčšinu pred ňou, tak to znamená, že nadpolovičný počet dvojíc bol tvorený týmto prvkom. A teda po „vydelení dvoma“ bude mať tento prvok stále nadpolovičný počet výskytov. Tým sme teda ukázali, že ak je n párne, vieme v lineárnom čase našu úlohu previesť na úlohu nanajvýš polovičnej veľkosti.

Ako to bude vyzerať pre nepárne n? Označme x prvok, ktorý má v našom poli väčšinu. Dvojicu prvkov nazveme dobrá, ak sú oba rovné x, a zlá, ak sú oba rovnaké, ale iné od x. Posledný prvok označme p a zatiaľ ho odložme nabok. Ostatných $2m$ prvkov rozdeľme rovnako ako predtým na m dvojíc.

Ak $p \neq x$, sú všetky výskyty x (ktorých je aspoň $m+1$) rozdelené v týchto dvojiciach. To je presne predchádzajúci prípad, vieme teda, že dobrých dvojíc máme viac ako zlých.

Ak $p = x$, najhoršie, čo sa môže stať, je že máme presne m výskytov x v dvojiciach a že tie sú rozložené tak, že je dobrých a zlých dvojíc presne rovnako. (Dôkaz že dobrých nemôže byť menej ako zlých: Predstavme si, že zoberieme ľubovoľné nie-x a vymeníme ho s p. Dostaneme situáciu, v ktorej je ostro viac dobrých ako zlých dvojíc, a pritom sme zmenili len jednu dvojicu.)

Rozlíšme teraz dva prípady. Ak je dokopy dobrých a zlých dvojíc nepárny počet, už sme vyhrali. Totiž spolu s informáciou, že dobrých je aspoň toľko ako zlých, z toho vieme povedať, že dobrých je viac. Môžeme teda spraviť to isté ako pre párne n (a posledný prvok p jednoducho zahodiť).

Ak je dobrých a zlých dvojíc dokopy párny počet, necháme z každej z nich po jednom prvku *a navyše* ku nim pridáme posledný prvok p. Čo tým dostaneme? Ak bolo dobrých dvojíc viac ako zlých, bolo ich nutne viac aspoň o dve (aby bol celkový počet párny), a teda pridanie p nič nepokazí. A ak bolo dobrých dvojíc rovnako ako zlých, tak vieme, že $p = x$, a teda po pridaní p bude mať x opäť väčšinu.

Ešte raz teda zhrnieme celý algoritmus pre $n = 2m + 1$:

Postupne vyrobíme m dvojíc. Každú, v ktorej sú prvky rôzne, zahodíme, a z každej, v ktorej sú prvky rovnaké, jeden necháme a druhý zahodíme. Pritom si pamätáme paritu počtu prvkov, ktoré sme už nechali. Keď sa dostaneme k poslednému prvku p, ktorý už nemá pár, podľa zapamätanej parity sa rozhodneme, či ho nechať alebo zahodiť.

Vyššie popísaný algoritmus teda v lineárnom čase spracuje postupnosť, ktorú má v rúre, a vyrobí v nej novú postupnosť približne polovičnej dĺžky. Pritom platí, že nová postupnosť má ten istý majoritný prvok ako tá pôvodná. Ľahko spočítame, že opakovaným použitím tohto algoritmu (až kým nám nezostane jediný prvok) dostaneme odpoveď v lineárnom čase.

```
label kolo
get a ; jempty koniec
put a ; put 0
jump even

label even
  get a ; jz a kolo
  get b ; jz b posledne
  jeq a b pis_odd
jump even

label odd
  get a ; jz a kolo
  get b ; jz b kolo
```

```
  jeq a b pis_even
jump odd

label pis_odd  ; put a ; jump odd
label pis_even ; put a ; jump even
label posledne ; put a ; jump kolo
label koniec   ; put a ; print ; stop
```

A-III-4 Mravce idú

V tejto úlohe bolo cieľom spočítať zvyšok, ktorý dáva akési veľké číslo po delení číslom $M = 10^9 + 9$. Na úvod tohto riešenia preto spomenieme základy práce so zvyškami.

Majme dve celé čísla A a B. Vieme, že ich môžeme jednoznačne zapísať v tvare $A = a_1M + a_0$ a $B = b_1M + b_0$, kde čísla a_0, a_1, b_0, b_1 sú celé a $0 \leq a_0, b_0 < M$. Hodnoty a_0 a b_0 sú zvyšky, ktoré dávajú A a B po delení M. Všimnime si teraz, že $A \pm B = (a_1 \pm b_1)M + (a_0 \pm b_0)$ a $AB = (a_1b_1M + a_1b_0 + a_0b_1)M + a_0b_0$. Preto platí: $A \pm B$ dáva po delení M rovnaký zvyšok ako $a_0 \pm b_0$ a AB dáva po delení M rovnaký zvyšok ako a_0b_0.

Toto pozorovanie nám v našom riešení umožní vyhnúť sa veľkým číslam – sčítame, odčítame a násobíme, čo potrebujeme, a vždy, keď nejaká priebežná hodnota prekročí M, môžeme namiesto nej pracovať len so zvyškom, ktorý dáva po delení M.

Kvôli prehľadnosti budeme tento krok vo zvyšku riešenia vynechávať. Keď teda napr. uvedieme v riešení „do c priraď $a + b$“, myslíme tým „do c priraď $((a + b) \bmod M)$“.

Základné dynamické programovanie:

Poďme si spočítať počet rôznych ciest z miesta $(0, 0)$ na všetky ostatné miesta mriežky. Počet ciest do miesta (r, s) označme $f(r, s)$.

Zjavne ak (r, s) leží mimo našej mriežky, tak sa tam nedá dostať, a teda $f(r, s) = 0$. Rovnako $f(r, s) = 0$ pre miesta, kde je prekážka. Pre ostatné miesta môžeme postupovať nasledovne: Všimnime si, že keď chceme ísť do (r, s), tak tam musíme prísť z $(r - 1, s)$ alebo z $(r, s - 1)$. A keďže cesty cez tieto miesta sú určite navzájom rôzne, tak počet týchto možností stačí spočítať. Teda platí $f(r, s) = f(r - 1, s) + f(r, s - 1)$.

Takýmto spôsobom každú mapu vyriešime s časovou zložitosťou $O(RS)$, dostávame teda riešenie s celkovou časovou zložitosťou $O(NRS)$.

Cesty bez prekážok:

Niekedy nie je zlé sa na úlohu pozrieť z opačného konca. Totiž prekážky nám spôsobujú, že niektoré cesty budú nepoužiteľné. Preto najprv spočítame počet všetkých ciest ignorujúc prekážky a následne odpočítame zlé cesty.

Označme $c(r, s)$ počet ciest takých, že sa v jednom smere posunieme o r a v druhom o s. Inými slovami, $c(r, s)$ bude počet ciest z $(0, 0)$ do (r, s), pričom nemáme žiadne prekážky. Cesta, ktorá prejde v jednom smere vzdialenosť r a v druhom vzdialenosť s má určite presne $r + s$ krokov. Keď chceme vybrať konkrétnu z týchto ciest, musíme vybrať, ktorých r spomedzi týchto $r+s$ krokov povedie smerom dodola. Každému možnému výberu zodpovedá práve jedna cesta, a teda platí $c(r, s) = \binom{r+s}{r}$.

Ako túto hodnotu rýchlo spočítať si ukážeme neskôr. Dodefinujme ešte $c(r, s) = 0$, ak $r < 0$ alebo $s < 0$.

Inklúzia a exklúzia:

Predstavme si teraz, že máme práve jednu prekážku, a to na súradniciach (r_1, s_1). Potom celkový počet ciest vedúcich cez prekážku bude $c(r_1, s_1) \cdot c(R - r_1, S - s_1)$, keďže máme $c(r_1, s_1)$ spôsobov ako sa dostať z $(0, 0)$ ku prekážke a následne $c(R - r_1, S - s_1)$ spôsobov ako pokračovať ďalej.

Počet všetkých ciest, ktoré cez túto prekážku *nevedú*, vieme teda zistiť tak, že od $c(R, S)$ odčítame počet ciest, ktoré cez prekážku vedú.

Teraz jedno dôležité pozorovanie: Keď máme ľubovoľný počet prekážok, môžeme ich usporiadať podľa hodnoty $r_i + s_i$. Inými slovami, môžeme vo zvyšku riešenia predpokladať, že platí $r_1 + s_1 \leq r_2 + s_2 \leq \cdots$. Potom bude platiť nasledujúce tvrdenie: každá cesta, ktorá prechádza prekážkou číslo i, môže predtým ísť len cez prekážky s číslom menším ako i. Prečo? Jednoducho preto, že každým krokom sa o 1 zvýši súčet súradníc miesta, na ktorom práve stojíme. A teda každé políčko, cez ktoré sme doteraz šli, má menší súčet súradníc ako to, kde práve sme.

Vráťme sa k riešeniu pôvodnej úlohy. Čo ak by boli prekážky práve dve: na (r_1, s_1) a na (r_2, s_2)? V súlade s vyššie uvedeným pozorovaním predpokladáme, že $r_1 + s_1 \leq r_2 + s_2$.

Všetkých ciest je $c(R, S)$. Od nich odčítame cesty, ktoré vedú cez prvú prekážku, tých je $c(r_1, s_1) \cdot c(R - r_1, S - s_1)$, a následne aj cesty, ktoré vedú cez druhú prekážku, tých je $c(r_2, s_2) \cdot c(R - r_2, S - s_2)$.

Ešte stále však nemáme správny výsledok. Cesty, ktoré vedú postupne cez obe prekážky, sme totiž odčítali dvakrát. Aby sme dostali správny výsledok,

musíme zistiť, koľko je takýchto ciest, a tento počet k aktuálnemu výsledku pripočítať. No ale to je ľahké: ciest, ktoré postupne prechádzajú cez (r_1, s_1) a (r_2, s_2) je $c(r_1, s_1) \cdot c(r_2 - r_1, s_2 - s_1) \cdot c(R - r_2, S - s_2)$. Všimnite si, že vďaka tomu, ako sme dodefinovali c pre záporné vstupy, tento vzťah funguje aj v prípadoch, že takéto cesty neexistujú.

Vyššie uvedené úvahy pre jednu a dve prekážky vieme zovšeobecniť aj pre viac prekážok, čím dostaneme tzv. princíp inklúzie a exklúzie (alebo tiež „zapojenia a vypojenia“).[6]

Formálne môžeme toto riešenie sformulovať nasledovne: Nech $p(X)$ je počet ciest, ktoré idú cez každú prekážku v množine X (a možno aj cez iné prekážky). Pre ľubovoľnú množinu X vieme tento počet spočítať v čase úmernom $|X|$: stačí vynásobiť počet ciest z $(0, 0)$ k prvej prekážke, počet ciest od prvej prekážky ku druhej, atď., až počet ciest od poslednej prekážky na (R, S).

Aby sme dostali správny výsledok, potrebujeme zobrať všetky cesty. Od nich odčítame cesty vedúce cez ľubovoľnú jednu prekážku, pripočítame cesty cez dvojice prekážok, zase odpočítame cesty cez trojice prekážok, atď. Stručne to môžeme zapísať nasledovne: hľadaný počet ciest je rovný sume všetkých hodnôt $(-1)^{|X|} p(X)$, kde sčitujeme cez všetky $X \subseteq \{1, 2, \ldots, K\}$.

Implementáciou tohto vzťahu dostávame riešenie, ktoré vie ľubovoľnú mriežku s K prekážkami vyhodnotiť v čase $O(K \cdot 2^K)$ – za predpokladu, že má k dispozícii všetky potrebné kombinačné čísla.

Vzorové riešenie, prvá časť:

Označme $z(i)$ počet spôsobov, ako sa dostať ku prekážke i tak, aby sme nešli cez žiadnu prekážku s číslom menším ako i.

Určite platí $z(1) = c(r_1, s_1)$ – totiž cesta na prvú prekážku nemá ako prechádzať cez iné prekážky, preto každá možná cesta je dobrá.

Predpokladajme teraz, že už poznáme hodnoty $z(1)$ až $z(k-1)$. Ako určiť hodnotu $z(k)$? Všetky cesty ku prekážke k vieme rozdeliť do nasledujúcich navzájom disjunktných množín:

1. Cesty, ktoré vedú cez prekážku 1.
2. Cesty, ktoré nevedú cez prekážku 1 a vedú cez prekážku 2.
3. Cesty, ktoré nevedú cez prekážky 1 ani 2 a vedú cez prekážku 3.

   ...

k. Cesty, ktoré nevedú cez žiadnu z prekážok 1 až $k - 1$.

[6] `http://sk.wikipedia.org/wiki/Princíp_zapojenia_a_vypojenia`

Všetkých ciest ku prekážke k dokopy je $c(r_k, s_k)$. Všimnime si teraz cesty i-teho z vyššie uvedených typov (pre nejaké $i < k$). Koľko ich je? Máme $z(i)$ spôsobov, ako sa dostať k i-tej prekážke tak, aby sme netrafili žiadnu z predchádajúcich, a $c(r_k - r_i, s_k - s_i)$ spôsobov, ako sa odtiaľ, už ľubovoľnou cestou, dostať ku k-tej prekážke. Preto platí:

$$z(k) = c(r_k, s_k) - \sum_{i<k} z(i) \cdot c(r_k - r_i, s_k - s_i)$$

Keď už poznáme hodnoty $z(i)$, analogickou úvahou by sme vedeli spočítať počet ciest, ktoré neprechádzajú žiadnou prekážkou. Pri implementácii si ale môžeme pomôcť jednoduchým trikom: pridáme prekážku číslo $K + 1$ na súradniciach (R, S). Potom ako $z(K + 1)$ dostaneme presne počet hľadaných ciest.

Toto riešenie spracuje jednu mriežku v čase $O(K^2)$ – opäť za predpokladu, že máme k dispozícii všetky potrebné kombinačné čísla.

(Pred)počítanie kombinačných čísel:

Jednou možnou cestou je predpočítať si všetky kombinačné čísla pomocou vzorca $\binom{a+1}{b+1} = \binom{a}{b} + \binom{a}{b+1}$ alebo ináč povedané Pascalovho trojuholníka. Kľúčové je uvedomiť si, že kombinačné čísla sa nemenia. Stačí ich predpočítať raz, na začiatku behu programu, a následne ich využívať počas behu.

Najjednoduchšie riešenie je uložiť si ich jednoducho v dvojrozmernom poli. Pri pamäťovom limite 64 MB sa nám zmestí do pamäte približne 16 miliónov celých čísel, čo zodpovedá zhruba tabuľke 4000×4000.

Šikovnejšie riešenie je založené na pozorovaní, že väčšinu kombinačných čísel nikdy nevyužijeme. Lepšie je teda na začiatku načítať celý vstup a zistiť, ktoré kombinačné čísla potrebovať budeme. Následne budeme kombinačné čísla počítať pomocou vyššie uvedeného vzťahu a vždy, keď narazíme na také, ktoré potrebujeme, si jeho hodnotu zapamätáme. Pomocou tohto triku sa určite dalo získať aspoň 13 bodov.

Inou možnou cestou (ktorá okrem iného stačila aj na testovací vstup 14) je počítanie hodnôt $\binom{a}{b}$ cez ich prvočíselný rozklad: pre ľubovoľné prvočíslo p ľahko spočítame, koľkokrát delí každé z čísel $a!$, $b!$ a $(a - b)!$.

Delenie modulo M:

Kombinačné čísla vieme počítať aj priamo, podľa vzorca $\binom{a}{b} = \frac{a!}{b!(a-b)!}$. Problém s týmto prístupom je ale v delení. Zatiaľ čo sčítať, odčítať a násobiť modulo M je ľahké, s delením do vôbec nie je také jasné.

V prvom rade, delenie pri operáciách so zvyškami vo všeobecnosti nemusí fungovať. Napríklad čísla 12 a 22 dávajú po delení 10 rovnaký zvyšok, ale $12/2 = 6$ a $22/2 = 11$ už rovnaký zvyšok nedávajú. No a ak rátame modulo 10, tak napríklad číslo 7 vôbec nevieme vydeliť dvomi – teda neexistuje také x, aby $2x$ dávalo rovnaký zvyšok ako 7.

V našej situácii sme však na tom dobre, lebo číslo M, modulo ktoré rátame, je prvočíslo. A v takejto situácii vieme „deliť“ – teda ku každému a a každému b (takému, že $0 < b < M$) existuje práve jeden možný zvyšok x taký, že $bx \equiv a \pmod{M}$.

(Prečo je tomu tak? Uvažujme hodnoty 0, b, $2b$, …, $(M-1)b$. Žiadne dve z týchto hodnôt nemôžu po delení M dávať rovnaký zvyšok, lebo potom by M delilo ich rozdiel $(j-i)b$. To ale nemôže, lebo oba činitele sú menšie ako M a zároveň M je prvočíslo.)

Špeciálne teda ku každému b existuje práve jedno x také, že $bx \equiv 1 \pmod{M}$. Takéto x budeme volať inverzným prvkom k b a budeme ho značiť b^{-1}.

Teraz ľahko nahliadneme, že platí: ak a/b je celé číslo, tak dáva po delení M rovnaký zvyšok ako $a \cdot b^{-1}$. Ak teda potrebujeme vedieť hodnotu $\binom{a}{b}$ modulo M, zistíme ju ako $\big(a! \cdot (b!)^{-1} \cdot ((a-b)!)^{-1}\big)$.

Ak teda predpočítame pre každé n z rozsahu od 1 po 100 000 hodnoty $n!$ a $(n!)^{-1}$, budeme vedieť ľubovoľné potrebné kombinačné číslo zistiť v konštantnom čase.

Inverzné prvky:

Ako nájsť inverzný prvok k danému číslu b? Toto číslo vieme efektívne zistiť napríklad rozšíreným Euklidovým algoritmom, kde hľadáme nejaké celočíselné riešenie rovnice $bx + My = 1$.

Inou, jednoduchšou možnosťou je použiť malú Fermatovu vetu. Tá hovorí, že ak M je prvočíslo, tak pre ľubovoľné b $(0 < b < M)$ platí $b^{M-1} \equiv 1 \pmod{M}$. No a b^{M-1} môžeme prepísať ako $b \cdot b^{M-2}$, odkiaľ vidíme, že inverzným prvkom k b je $b^{M-2} \bmod M$. Túto hodnotu vieme šikovným umocňovaním vypočítať v čase $O(\log M)$.

Celé vzorové riešenie:

Začneme tým, že pre každé n predpočítame hodnotu $n! \bmod M$ a následne k nej pomocou malej Fermatovej vety zistíme aj inverzný prvok. Následne postupne spracúvame mriežky zo vstupu použitím postupu s počítaním čísel $z(i)$. Takéto riešenie má časovú zložitosť $O((R+S)\log M + NK^2)$.

Listing programu:

```
// riesenie pouzivajuce inverzne prvky a dynamicke programovanie
#include <algorithm>
#include <vector>
#include <cstdio>
using namespace std;

long long MODEXP(long long number, long long power, long long modulus) {
  if (power==0) return 1LL % modulus;
  if (power==1) return number % modulus;
  long long tmp = MODEXP(number,power/2,modulus);
  tmp = (tmp*tmp) % modulus;
  if (power&1) tmp = (tmp*number) % modulus;
  return tmp;
}

bool distless(const pair<int,int> &A, const pair<int,int> &B) {
  return A.first + A.second < B.first + B.second;
}

int main() {
  int R, S, N, K, P=1000000009;
  scanf("%d%d%d",&R,&S,&N);

  vector<int> fact(R+S+2);
  fact[0]=fact[1]=1;
  for (int i=2; i<R+S+2; ++i) fact[i] = (1LL * fact[i-1] * i) % P;
  vector<int> inverse(R+S+2);
  for (int i=0; i<R+S+2; ++i) inverse[i] = MODEXP(fact[i],P-2,P);

  while (N--) {
    scanf("%d",&K);
    vector< pair<int,int> > prekazky;
    for (int k=0; k<K; ++k) {
      int x,y;
      scanf("%d%d",&x,&y);
      prekazky.push_back( make_pair(x,y) );
    }
    prekazky.push_back( make_pair(R,S) );
    sort( prekazky.begin(), prekazky.end(), distless );
    vector<long long> G(K+1);
    for (int k=0; k<=K; ++k) {
      int dx = prekazky[k].first, dy = prekazky[k].second;
      G[k] = (((1LL * fact[dx+dy] * inverse[dx]) % P) * inverse[dy]) % P;
    }
    for (int k=0; k<K; ++k)
      for (int l=k+1; l<=K; ++l) {
        int dx = prekazky[l].first - prekazky[k].first,
            dy = prekazky[l].second - prekazky[k].second;
        if (dx<0 || dy<0) continue;
```

```
        long long C = (((1LL*fact[dx+dy]*inverse[dx]) % P) * inverse[dy]) % P;
        G[l] -= G[k] * C;
        G[l] %= P;
        if (G[l] < 0) G[l] += P;
      }
    printf("%Ld\n",G[K]);
  }
}
```

A-III-5 Hurikán

Ľahko vidno, že kiribatské ostrovy a aktuálne stojace mosty medzi nimi predstavujú neorientovaný graf – keď spadne niektorý z mostov, z grafu zmizne príslušná hrana.

Medzi vrcholmi, ktoré sú v jednom komponente súvislosti grafu, existuje spojenie po mostoch. Zostáva zabezpečiť dostatok prievozníkov na dopravu medzi rôznymi komponentmi. Keď má graf S komponentov, najmenší postačujúci počet prievozníkov je $S - 1$ (napríklad budú premávať medzi prvým komponentom a všetkými ostatnými). Preto nám stačí zistiť pre každý deň, z koľkých komponentov sa skladá graf.

Priamočiare riešenie: Počet komponentov grafu sa dá zistiť prehľadávaním do hĺbky alebo do šírky. Začneme v prvom vrchole a ofarbujeme všetky tie vrcholy, do ktorých sa dá z neho dostať. Ak ešte ostali nejaké neofarbené vrcholy, niektorý z nich si vyberieme a začneme z neho znova prehľadávať. Toto opakujeme, kým neofarbíme celý graf. Počet komponentov je rovný práve počtu, koľkokrát sme museli začínať prehľadávať.

Na tomto je založené nasledujúce riešenie: postupne pre každý deň od začiatku odstránime z grafu hranu mosta, ktorý práve spadol, a prehľadávaním zisťujeme aktuálny počet komponentov. To znamená, že potrebujeme práve $(K + 1)$-krát prehľadať celý graf. Časová zložitosť toho riešenie je preto $O(K \cdot (M + N))$.

Prístup odzadu: Pozrime sa na úlohu od konca. Začneme s grafom, v ktorom chýba všetkých K mostov s porušenou statikou. Budeme postupovať od posledného dňa k prvému a postupne pridávať tieto mosty do grafu. Zatiaľ je toto riešenie rovnako dobré ako predchádzajúce, akurát dostávame výsledky v opačnom poradí.

Ďalšou zmenou je, že tentoraz nám do grafu hrany pribúdajú. Pridaním hrany sa môžu dva komponenty spojiť do jedného (ak táto hrana viedla medzi

vrcholmi z dvoch rôznych komponentov) alebo sa rozdelenie na komponenty vôbec nezmení (ak hrana viedla medzi vrcholmi z toho istého komponentu).

Na rýchlejšie riešenie budeme preto potrebovať dátovú štruktúru, ktorá nám umožní efektívne zisťovať, či sú dva vrcholy v rovnakom komponente, a tiež zlučovať dvojicu komponentov.

Union-find: Na chvíľu zabudnime, že ide o graf a riešme všeobecnejšiu úlohu. Komponent nazvime množinou; vrcholy v ňom budú prvkami tejto množiny.

Množinu prvkov si reprezentujeme stromom. Zavesíme ho za koreň, ktorý označme ako *reprezentanta* množiny. Z ostatných prvkov vedie vždy jedna hrana smerom hore, do *nadriadeného* prvku. Nadriadený prvok je bližšie ku koreňu alebo je to dokonca koreň.

Keď začneme v ľubovoľnom prvku množiny a budeme prechádzať v hierarchii čoraz vyššie (stále do nadriadeného prvku), musíme skončiť v reprezentantovi množiny. To znamená, že dva prvky sú v rovnakej množine práve vtedy, keď z oboch dôjdeme do toho istého reprezentanta.

Dve množiny zlúčime do jednej tak, že reprezentanta jednej z nich zavesíme pod reprezentanta druhej (nastavíme mu ho ako nadriadeného).

Spájaním množín môžu vznikať dlhé reťaze prvkov. Vtedy bude mať hľadanie reprezentantov až lineárnu časovú zložitosť od veľkosti množiny. Ukážeme si dve úpravy, ktoré prinesú zrýchlenie. Ako prvé spravíme to, že pri zlučovaní dvoch množín zavesíme vždy strom s menšou hĺbkou pod hlbší strom. To zabezpečí, že vzniknuté stromy budú mať hĺbku rádovo logaritmickú od počtu prvkov. Ďalšou možnosťou je zvoliť si náhodne, ktorý strom zavesíme pod ktorý, vďaka čomu budú v očakávanom prípade tiež vznikať stromy s malou hĺbkou. Druhým trikom je kompresia cesty. Po nájdení reprezentanta ho nastavíme všetkým prvkom, ktorými sme na ceste nahor prešli, ako priameho nadriadeného.

Pomocou pokročilých matematických metód sa dá dokázať, že priemerná časová zložitosť hľadania reprezentanta s využitím oboch zrýchlení bude $O(\alpha(n))$, kde n je veľkosť množiny a α je inverzná Ackermannova funkcia. Pre prakticky využiteľné čísla je jej hodnota najviac 4, preto ju môžeme pokladať za konštantu. Celková časová zložitosť tohto algoritmu je teda $O(N + M\alpha(N))$, prípadne $O(N + M + K\alpha(N))$ ak najskôr všetky stabilné mosty spracujeme jedným prehľadávaním. V listingu programu na konci tohto riešenia používame namiesto spájania podľa hĺbok spájanie náhodné – ľahšie sa implementuje a očakávaná časová zložitosť je rovnaká.

Iný spôsob ako napísať riešenie, ktoré by malo získať plný počet bodov:

Rovnako ako v predchádzajúcom riešení budeme mosty spracúvať od konca a zisťovať počet komponentov súvislosti. To budeme robiť jednoducho tak, že ku každému vrcholu si budeme pamätať číslo jeho komponentu a ku každému komponentu zoznam jeho vrcholov. Vždy, keď pridávame hranu, pozrieme sa, či sú jej konce v tom istom komponente. Ak áno, nič sa nedeje, ak nie, „prefarbíme“ celý menší komponent – teda každému vrcholu v ňom zmeníme jeho číslo na číslo väčšieho komponentu.

Všimnite si, že vždy, keď vrchol prefarbíme, dostane sa tým do komponentu aspoň dvakrát takej veľkosti ako dovtedy. Preto každý vrchol prefarbíme nanajvýš $\log_2 N$ krát, a teda má toto riešenie časovú zložitosť $O(M + N \log N)$.

Listing programu:

```
#include <cstdio>
#include <cstdlib>
using namespace std;
#define MAX 1234567

int N, M, K, components, E[MAX][2], boss[MAX], D[MAX], damaged[MAX], answer[MAX];

int sef(int x) {
    if (x==boss[x]) return x; else return boss[x]=sef(boss[x]);
}

void join(int x, int y) {
    x=sef(x); y=sef(y); if (x==y) return;
    --components;
    if (rand()&1) boss[x]=y; else boss[y]=x;
}

int main() {
    scanf("%d%d",&N,&M);
    components = N;
    for (int n=1; n<=N; ++n) boss[n]=n;
    for (int m=1; m<=M; ++m) scanf("%d%d",&E[m][0],&E[m][1]);
    scanf("%d",&K);
    for (int k=1; k<=K; ++k) { scanf("%d",&D[k]); damaged[D[k]]=1; }
    for (int m=1; m<=M; ++m) if (!damaged[m]) join(E[m][0],E[m][1]);
    answer[0] = components-1;
    for (int k=K; k>=1; --k) {
        join( E[D[k]][0], E[D[k]][1] );
        answer[K+1-k] = components-1;
    }
    for (int k=K; k>=0; --k) printf("%d\n",answer[k]);
}
```

Výsledky krajských kôl kategórie B

V kategórii B súťaž končí krajským kolom, nižšie uvedené sú výsledky tohto kola v siedmich z ôsmich krajov. (V Banskobystrickom kraji sa do kategórie B nezapojil žiaden riešiteľ.)

Výsledky neboli koordinované, je teda možné, že v rôznych krajoch boli použité mierne odlišné bodovacie škály.

Bratislavský kraj:

Meno	Ročník, škola	1.	2.	3.	4.	Σ
1. Martin Fikar	1 Gym. Jura Hronca BA	4	5	10	7	26
2. Richard Kakaš	1 Gym. Jura Hronca BA	3	5	5	6	19
3. Rastislav Rabatin	1 Gym. Jura Hronca BA	4	4	4	6	18
4. Matej Krajčovič	1 Gym. Jura Hronca BA	5		5	7	17
5. Richard Trebichavský	2 Gym. Jura Hronca BA	3	3	7	0	13
6. Ján Ondráš	0 Gym. Grösslingová BA	4		7		11

Košický kraj:

Meno	Ročník, škola	1.	2.	3.	4.	Σ
1. Jakub Šafin	1 Gym. P. Horova Michalovce	4	7	10	7	28
2. Miroslava Klučárová	2 Gym Javorová Spiš. Nová Ves	4	0	10	7	21
3. Radovan Hurčík	2 Gym. Sečovce	4	1	5	9	19
4. Ondrej Tokár	2 Gym. Sečovce	4	1	5	6	16
5. Martin Bajtoš	1 Gym. Školská Spiš. Nová Ves	4	0	4	6	14
6. Oto Király	2 Gym. Sečovce	4	0	4	1	9
7. Tomáš Pham	2 Gym Javorová Spiš. Nová Ves	3	0	3	0	6
8. Miloš Tutko	2 SOŠ Moldavská cesta Košice	1	0	0	0	1
9. Andrej Štofa	1 SOŠ Moldavská cesta Košice	0	0	0	0	0

Nitriansky kraj:

Meno	Ročník, škola	1.	2.	3.	4.	Σ
1. Juraj Papp	2 Gym. Piaristická Nitra	6	9	9		24
2. Marek Šuppa	1 SKŠ Nitra, Gym. sv. Cyrila a Metoda	4	7	5		16
3. Juraj Karlubík	2 Gym. Golianova Nitra	4	0	6	4	14
4. Gabriel Takács	2 Gym. Fándlyho Šaľa	3	0	6	4	13
Martin Macák	2 Gym. Golianova Nitra	3	1	9	0	13
6. Michal Porubsky	-2 SKŠ Nitra, Gym. sv. Cyrila a Metoda	3		1	5	9

Prešovský kraj:

Meno	Ročník, škola	1.	2.	3.	4.	Σ
1. Pavol Šatala	2 Gym. Kežmarok	7	8	8	9	32
2. Jerguš Greššák	1 Gym. Mudroňova Prešov	4	10	10	6	30
3. Patrik Pekarčík	1 Gym. Kežmarok	6	6	3	0	15

Trenčiansky kraj:

Meno	Ročník, škola	1.	2.	3.	4.	Σ
1. Roman Kučera	2 Gym. Nedožerského Prievidza	3	5	9		17
2. Milan Mikuš	2 Gym. Ul. 1. mája Trenčín	4	8	4		16
3. Ivan Ševčík	2 Gym. Ľudovíta Štúra Trenčín	3	4	1		8

Trnavský kraj:

Meno	Ročník, škola	1.	2.	3.	4.	Σ
1. Márton Bartal	1 Súkromné gym. s vjm. Dunajská Streda	4	4			8

Žilinský kraj:

Meno	Ročník, škola	1.	2.	3.	4.	Σ
1. Richard Molnár	1 Ev. gym. J. Tranovského Lipt. Mikuláš	7	8	0	7	22

Výsledky celoštátneho kola kategórie A

Celoštátne kolo kategórie A sa v 25. ročníku Olympiády v informatike uskutočnilo v dňoch 24. až 27. marca 2010 v Novom Meste nad Váhom. Praktickú časť súťažiaci riešili v priestoroch Fakulty mechatroniky Trenčianskej univerzity.

Meno	Ročník, škola	1.	2.	3.	4.	5.	Σ
1. Belan Tomáš	4. ŠpMNDaG, Teplická 7, BA	8	9	6	11	15	49
2. Hudec Tobiáš	4. G. Komenského 2, Partizánske	10	9	10	0	15	44
3. Balog Matej	3. G. Grösslingová 18, Bratislava	7	10	6	5	15	43
Hozza Ján	3. G. Jura Hronca, Bratislava	7	9	7	5	15	43
5. Bačo Ladislav	4. G. Poštová 9, Košice	7	10	8	6	9	40
6. Sládek Filip	4. G. A. Bernoláka, Námestovo	7	5	8	2	15	37
7. Špano Marek	3. G. Jura Hronca, Bratislava	6	8	6		15	35
8. Hruška Eugen	2. G. I. Kupca, Hlohovec	9	9	7	5	1	31
9. Hagara Michal	4. G. Jura Hronca, Bratislava	0	10	8	6	3	27
10. Sebechlebský Ján	3. G. Žiar n/Hronom	4	9	6	3	3	25
11. Jančo Tomáš	3. G. Ľ. Štúra, Trenčín	5	9	4	3	3	24
Konečný Jakub	4. G. Grösslingová 18, Bratislava	6	7	5	4	2	24
13. Pitoňák Martin	4. G. J. G. Tajovského, BB	2	5	3	3	10	23
14. Anderle Michal	3. G. B. S. Timravy, Lučenec	1	7	8	6	0	22
15. Guričan Pavol	3. G. Jura Hronca, Bratislava	5	10	3	3		21
Krejčír Andrej	4. G. V. B. Nedožerského, PD	0	8	7	3	3	21
17. Vavrík Boris	3. G. Jura Hronca, Bratislava	2	8	6	3	1	20
18. Pinter Martin	3. G. Jura Hronca, Bratislava	7	0	9	3	0	19
19. Majerech Ferdinand	4. G. Školská 7, Sp. N. Ves	6	0	1		9	16
20. Babej Tomáš	3. G. Poštová 9, Košice	5	2	3	5	0	15
Cuc Bruno	4. G. Grösslingová 18, Bratislava	3	5	7	0	0	15
22. Horňák Marián	2. G. Párovská 1, Nitra	2	0	8		4	14
Morvay Tomáš	3. G. Golianova 68, Nitra	0	0	6	3	5	14
24. Korbaš Rafael	3. G. F. G. Lorcu, Bratislava	4	3	3	3	0	13
25. Gandžala Jozef	4. G. J. G. Tajovského, BB	0	6	3	3		12
Novella Tomáš	4. G. Alejová 1, Košice	3	0	2	3	4	12
27. Jurenka Vladimir	4. G. Grösslingová 18, Bratislava	5	0	2	4		11
Livora Tomáš	3. G. Javorová 16, Sp. N. Ves	4	0	3	3	1	11
29. Strapko Martin	3. G. Jura Hronca, Bratislava	4	0	1	3	0	8
30. Rábara Radoslav	3. G. J. Hollého, Trnava	1	0	3	3	0	7

Výsledky výberového sústredenia

V dňoch 13. až 19. mája 2010 sa v Bratislave konalo výberové sústredenie. Na toto sústredenie boli pozvaní najlepší riešitelia celoštátneho kola OI, kategórie A. Štyria najlepší riešitelia výberového sústredenia majú možnosť reprezentovať Slovensko na Medzinárodnej olympiáde v informatike. Na základe výberového sústredenia taktiež vyberá SK OI reprezentačný tím pre Stredoeurópsku olympiádu v informatike a pre prípravné sústredenia. Vzhľadom na to, že Stredoeurópska olympiáda v informatike bola v roku 2010 na Slovensku, až ôsmi súťažiaci dostali príležitosť zúčastniť sa jej.

V nasledujúcej tabuľke sú postupne uvedené body za celoštátne kolo OI, za „domáce úlohy“ a za sedem súťažných dní výberového sústredenia.

	Meno	$\sum$	CK	doma	št	pi	so	ne	po	ut	st
1.	Tomáš Belan	639.9	49	18.5	60	160	76	44	83.9	100	48.5
2.	Ján Hozza	444.2	43	15.5	52.5	74	67.5	35	85.2	35.5	36
3.	Ladislav Bačo	407.9	40	15	35	76.5	69.5	33	78.4	44.5	16
4.	Matej Balog	351.6	43	20.5	31	66.5	36	27	43.1	49.5	35
5.	Filip Sládek	343.0	37	18.5	51.5	61.5	58.5	17.5	53.5	27.5	17.5
6.	Michal Anderle	235.8	22	12	24	68.5	19.5	18.5	29.3	24	18
7.	Ján Sebechlebský	204.3	25	3	20.5	53	29.5	8	24.8	22	18.5
8.	Michal Hagara	182.8	27	4.5	20.5	43.5	31.5	6	22.3	13	14.5
9.	Marek Špano	179.0	35	9	17	32	10	6	13.5	41.5	15
10.	Martin Pitoňák	174.0	23	7.5	29	61.5	17	1	16.5	18.5	0
11.	Tomáš Jančo	166.9	24	2	19.5	38	21.5	16.5	22.4	14	9

Slovensko-švajčiarske prípravné sústredenie

Vďaka blízkej spolupráci medzi národnými olympiádami v informatike na Slovensku a vo Švajčiarsku mali vybraní riešitelia KSP a OI možnosť zúčastniť sa prípravného sústredení vo švajčiarskom Davose v dňoch 8. až 13. februára 2010. Výsledky sústredenia uvádzame v nasledujúcej tabuľke.

Meno	kraj.	day 1	day 2	day 3	day 4	$\sum$
1. Tomáš Belan	SVK	300	250	350	332	1232
2. Ján Hozza	SVK	210	210	300	272	992
3. Ladislav Bačo	SVK	200	210	200	272	882
4. Timon Manfred Gehr	SUI	200	200	270	172	842
5. Josef Ziegler	SUI	170	200	100	272	742
6. Andrej Mariš	SVK	170	150	100	106	526
7. Sämi Grütter	SUI	130	150	200	0	480
8. Alain Vaucher	SUI	110	150	140	73	473
9. Lazar Todorović	SUI	100	100	100	40	340
10. Martin Jehli	SUI	50	80	100	73	303
11. Alexander Kayed	SUI	10	100	100	73	283
12. Sofia Balicka	SUI	40	0	110	106	256
13. Adrian Stauffer	SUI	10	90	0	0	100
14. Adrian Aulbach	SUI	10	30	0	43	83

Česko-poľsko-slovenské prípravné sústredenie

V poradí dvanáste súťažné stretnutie najlepších stredoškolákov z Čiech, Poľska a Slovenska sa uskutočnilo v Prahe v dňoch 21. až 25. júna 2010. Ako býva zvykom, najlepšie sa darilo poľským súťažiacim.

Meno, krajina		day1	day2	day3	day4	$\sum$
1. Anna Piekarska	POL	205	300	128	80	713
2. Dawid Dąbrowski	POL	255	200	68	160	683
Michał Zgliczyński	POL	195	200	88	200	683
4. Tomáš Belan	SVK	80	185	157	200	622
5. Igor Adamski	POL	275	100	29	160	564
6. Adrian Jaskółka	POL	85	300	69	10	554
7. Łukasz Jocz	POL	90	240	40	180	550
8. Jan Kanty Milczek	POL	170	200	67	100	537
9. Vlastimil Dort	CZE	175	200	56	100	531
10. David Klaška	CZE	150	155	60	155	520
11. Ján Hozza	SVK	25	200	178	105	508
12. Hynek Jemelík	CZE	100	180	116	100	496
13. Filip Hlásek	CZE	130	195	48	0	373
14. Matej Balog	SVK	150	100	96	20	366
15. Filip Sládek	SVK	105	145	49	10	309
16. Jan Polášek	CZE	85	85	0	100	270
17. Štěpán Šimsa	CZE	105	55	25	30	215
18. Michal Anderle	SVK	5	50	35	40	130
19. Ján Sebechlebský	SVK	0	15	50	20	85
Jiří Setnička	CZE	75	10	0	0	85
21. Lukáš Folwarczný	CZE	0	50	10	20	80
22. Michal Mojzík	CZE	15	10	10	10	45

Stredoeurópska olympiáda v informatike

V roku 2010 sa po tretíkrát za svoju 17-ročnú históriu konala Stredoeurópska olympiáda v informatike (CEOI) na Slovensku. Rovnako ako v roku 2002, kedy sme CEOI organizovali u nás naposledy, boli hosťujúcim mestom Košice a samotná súťaž prebehla v priestoroch Univerzity Pavla Jozefa Šafárika.

Súťaž sa konala v dňoch 12. až 19. júla 2010. Zúčastnili sa jej ôsmi stredoškoláci zo Slovenska a po štyroch ďalších z Bulharska, Českej republiky, Chorvátska, Maďarska, Nemecka, Poľska, Rumunska a Švajčiarska. Webstránku súťaže nájdete na adrese `http://ceoi2010.ics.upjs.sk/`.

Priebeh jednotlivých dní bol nasledovný: V utorok 13. júla bola súťaž slávnostne otvorená. V rámci slávnostného otvorenia odznela prednáška prof. Gefferta na tému Multiway in-place merging. V stredu a piatok boli dva súťažné dni. V každý z nich dostali súťažiaci 5 hodín času na vyriešenie troch veľmi náročných praktických úloh. Voľný čas po súťaži bol v piatok vyplnený prezentáciou humanoidného robota Nao. Vo štvrtok a sobotu mali súťažiaci, ako aj ich sprievod, možnosť oddýchnuť si na výletoch a spoznať krásy Slovenska. Pri štvrtkovom výlete boli na výber trasy na Popradské pleso a na Téryho chatu, traja najrýchlejší turisti dokonca stihli aj prejsť Priečnym sedlom. Sobotňajší výlet smeroval do Krásnej Hôrky a odtiaľ do Zádielskej doliny. Nedeľné dopoludnie strávili účastníci v Múzeu letectva, a potom sa už dočkali záverečného vyhodnotenia.

Na hladkom chode CEOI 2010 sa podieľalo množstvo ľudí. Určite sa nám ich nepodarí vymenovať všetkých, pokúsime sa uviesť väčšinu z nich.

Autori súťažných úloh

- RNDr. Michal Forišek, PhD. (chair)
- Daniel Bundala
- Peter Fulla
- RNDr. Ján Katrenič
- Mgr. Michal Nánási
- Mgr. Lukáš Poláček
- František Simančík
- Mgr. Monika Steinová
- Mgr. Juliana Šišková
- Mgr. Marek Zeman

Organizačný výbor a garanti akcie
- doc. RNDr. Gabriela Andrejková, CSc. (chair)
- prof. RNDr. Pavol Sovák, CSc.
- prof. RNDr. Andrej Bobák, DrSc.
- doc. RNDr. Dana Pardubská, PhD.
- prof. RNDr. Branislav Rovan, PhD.
- doc. RNDr. Gabriel Semanišin, PhD.
- RNDr. František Galčík, PhD. (www-pages)
- RNDr. Rastislav Krivoš-Belluš (guide leader)
- Jana Boháčová
- PhDr. Svetlana Libová
- doc. RNDr. Božena Mihalíková, CSc.
- Mgr. Ladislav Mikeš (trips)
- Mgr. Peter Mlynárčik (trips)

Technické zabezpečenie
- Mgr. Martin Rejda (chair)
- Ing. Marián Andrejko
- Eduard Dvorný
- Mgr. Ľudovít Hvizdoš
- Michal Petrucha

Sprievodcovia tímov
- Zuzana Bedécsová (POL)
- Mária Dolinská (BUL)
- Ivana Ďurčová (GER)
- Mária Harčarufková (SVK 1, 2)
- Natália Iriasová (CRO)
- Matúš Jaraba (SUI)
- Ján Jerguš (ROM)
- Eva Maďarošová (HUN)
- Mária Piatnicová (CZE)

Reportéri
- Maroš Andrejko
- Gabriela Hucovičová
- Samuel Kupka
- Pavol Rajzák

Slovensko na CEOI 2010 reprezentovali títo žiaci:

- Michal Anderle z Gymnázia B. S. Timravy v Lučenci
- Ladislav Bačo z Gymnázia Poštová v Košiciach
- Matej Balog z Gymnázia Grösslingová v Bratislave
- Tomáš Belan zo Školy pre mimoriadne nadané deti v Bratislave
- Ján Hozza z Gymnázia Jura Hronca v Bratislave
- Martin Pitoňák z Gymnázia J. G. Tajovského v Banskej Bystrici
- Ján Sebechlebský z Gymnázia v Žiari nad Hronom
- Filip Sládek z Gymnázia Antona Bernoláka v Námestove

Výsledková listina CEOI 2010

meno	kraj.	prvý deň			druhý deň			Σ	medaila
1. Anna Piekarska	POL	55	90	60	60	95	100	460	gold
2. Anton Anastasov	BUL	55	60	50	40	100	100	405	gold
Stjepan Glavina	CRO	25	40	40	100	100	100	405	gold
4. Ivan Katanić	CRO	55	90	50	40	100	55	390	silver
Ivica Kičić	CRO	55	90	50	0	95	100	390	silver
6. Tomáš Belan	SVK	55	100	0	100	100	20	375	silver
Bogdan-Cristian Tătăroiu	ROM	35	80	50	10	100	100	375	silver
8. Vladislav Haralampiev	BUL	100	10	50	10	100	100	370	silver
9. Jan Milczek	POL	100	20	20	20	100	100	360	silver
10. Vlad-Alexandru Gavrilă	ROM	15	60	40	20	100	45	280	silver
11. Rumen Hristov	BUL	5	10	50	70	100	40	275	bronze
12. Adrian Satja Kurdija	CRO	35	30	50	10	100	45	270	bronze
13. Adrian Jaskółka	POL	100	0		10	100	55	265	bronze
14. Simon Bürger	GER	65		0	0	90	100	255	bronze
15. Igor Adamski	POL	25	10	50	10	100	45	240	bronze
Victor-Cristian Ionescu	ROM		40	50	40	15	95	240	bronze
17. Filip Hlásek	CZE	15	10	0	40	100	45	210	bronze
18. Ján Hozza	SVK	25	40	0	40	70	20	195	bronze
19. Mihail Kovachev	BUL	55	0	50	0	30	55	190	bronze
20. Richárd Palincza	HUN	0	40	40	0	90	0	170	bronze

meno	kraj.	prvý deň			druhý deň			Σ	medaila
21. Štěpán Šimsa	CZE	25		0	40	15	45	125	
22. Ágoston Weisz	HUN	15	10	0	10	65	20	120	
23. Alexandru Tache	ROM		0		10	0	100	110	
24. Filip Sládek	SVK		20	0	0	65	20	105	
25. Josef Ziegler	SUI		30	0	0	50	20	100	
26. Matej Balog	SVK	15	0	10	10	40	20	95	
Lukáš Folwarczný	CZE	10	0	0	0	70	15	95	
Michal Mojzík	CZE	0	20	0	0	55	20	95	
29. Aaron Montag	GER	15	20	50	0	5	0	90	
30. Lazar Todorović	SUI	0	20	50	10	5		85	
31. Nikola Djokić	SUI		40		0	40	0	80	
32. Patrik Adrián	HUN	15	40	0		15		70	
33. Zoltán Szenczi	HUN		0	0	40	15	10	65	
34. Patrick Klitzke	GER	0	0		0	15	40	55	
35. Marek Špano	SVK			0		30	20	50	
36. Ján Sebechlebský	SVK		20	0	0	15	10	45	
37. Michal Anderle	SVK		10	0	0	25	0	35	
38. Klaas-Hendrik Poelstra	GER		0			30		30	
39. Martin Pitoňák	SVK		0			15	10	25	
40. Thomas Leu	SUI		10	0	0	0	0	10	

Víťazi paralelne bežiacej online súťaže

meno	kraj.	prvý deň			druhý deň			Σ
1. Tiancheng Lou	CHN	100	100	100	100	100	60	560
2. Tomasz Kulczyński	POL	100	90	100	20	100	100	510
3. Jiang Zhongtian	CHN	100	80	100	20	100	45	445
Feng Qiwei	CHN	55	100	50	40	100	100	445
5. Bai Chi	???	100	20	50	70	100	100	440
6. Gao Xin	CHN	100	30	100	0	100	100	430
7. rpb	???	100	100	0	80	100	45	425

Zadania prvého súťažného dňa

Pakty

Kde bolo tam bolo, bol raz jeden veľký ostrov obdĺžnikového tvaru. Strany ostrova boli dlhé presne R míľ a C míľ a celý ostrov bol rozdelený na $R \times C$ oblastí usporiadaných do mriežky. Niektoré oblasti boli neobývané, zvyšok bol zastavaný osadami rozprávkových bytostí: elfov, ľudí, trpaslíkov a hobitov. Každá oblasť obsahuje nanajvýš jednu osadu. Dve osady považujeme za susedné, ak zodpovedajúce oblasti zdieľajú spoločnú hranu mriežky.

V ostatnom čase sa osadníci začali báť Veľkého Temna, ktoré zosadlo na krajinu. Aby sa cítili bezpečnejšie, rozhodli sa uzavrieť pakty s niektorými zo svojich susedov. Pakt sa vždy uzatvára medzi dvoma susednými osadami a je to vzájomná a symetrická dohoda.

Osadníci v závislosti od svojej rasy považujú za bezpečné nasledujúce konfigurácie paktov:

- Elfovia si veria, takže potrebujú uzavrieť pakt s presne jedným susedom.
- Ľudské osady musia mať pakt práve s dvoma susedmi. Navyše nie je múdre nechávať dve protiľahlé strany osady nechránené. Preto títo dvaja spojenci nemôžu byť susedmi na dvoch protiľahlých stranách osady.
- Trpaslíci potrebujú pakty s práve troma susedmi.
- Hobiti sú strachopudi, a preto chcú mať pakty so všetkými štyrmi susedmi.

Inak povedané, okrem ľudských osád, všetky osady chcú mať pakty s určitým presným počtom svojich susedov, ale nezaujíma ich, ktorí susedia to budú. Ľudské osady majú obmedzenie navyše: spojenci nesmú byť na protiľahlých stranách osady.

Tieto podmienky musia byť splnené bez ohľadu na pozíciu osady na mape. Napríklad trpasličia osada chce mať tri pakty. Ak sa teda nachádza na pobreží, má len troch susedov a musí uzavrieť pakt so všetkými. Ak sa trpasličia osada nachádza v rohu ostrova má smolu – osadníci sa nikdy nebudú cítiť bezpečne.

Súťažná úloha:

Pre daný popis ostrova je vašou úlohou rozhodnúť, či je možné uzavrieť pakty takým spôsobom, aby sa osadníci vo všetkých osadách cítili bezpečne. V prípade kladnej odpovede navyše navrhnite jednu možnú konfiguráciu paktov. V prípade viacerých riešení vyberte ľubovoľné z nich.

Vstup:

Prvý riadok vstupu obsahuje dve čísla R a C určujúce veľkosť ostrova. Nasledujúcich R riadkov obsahuje popis ostrova. Každý riadok pozostáva z C čísel medzi 0 a 4 oddelených medzerami: 0 znamená neobývanú oblasť, 1 znamená elfiu osadu, 2 znamená ľudskú osadu, 3 znamená trpasličiu osadu, 4 znamená hobitiu osadu. (Všimnite si, že toto číslo vždy zodpovedá počtu paktov, ktoré príslušná osada potrebuje uzavrieť.)

Ohraničenia:

Vo všetkých testovacích vstupoch $1 \leq R, C \leq 70$.

V testovacích vstupoch za 55 bodov $\min(R, C) \leq 10$. Z toho v testovacích vstupoch za 15 bodov $R \cdot C \leq 20$.

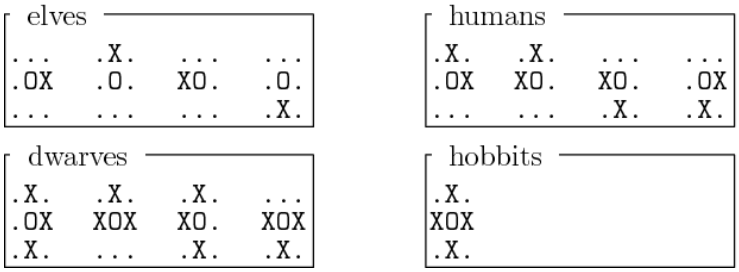
Všetky testy v jednom balíku za 10 bodov obsahujú vstupy, kde ostrov obsahuje iba neobývané oblasti a ľudské osady. (Týchto 10 bodov sa neprekrýva s vyššie spomínanými 55 bodmi.)

Výstup:

Ak neexistuje riešenie, vypíšte jeden riadok s reťazcom „`Impossible!`“ (bez úvodzoviek). V opačnom prípade vypíšte jednu možnú konfiguráciu paktov v nasledujúcom formáte.

Každá oblasť sa vo výstupe bude vyskytovať ako matica znakov 3×3. Ak je oblasť neobývaná, príslušná časť vstupu bude vyplnená znakmi „`.`“ (bodkami). Ak oblasť obsahuje osadu, tak v strede matice bude znak „`O`“ (veľké písmeno abecedy) a na susedných políčkach reprezentujúcich osady, s ktorými má aktuálna osada uzavreté pakty, má byť znak „`X`“ (veľké písmeno abecedy). Zvyšok matice 3×3 má byť vyplnený znakmi „`.`“ (bodkami).

Nasledujúci obrázok ukazuje pre každý druh osady možné konfigurácie paktov.



(elves = elfovia, humans = ľudia, dwarves = trpaslíci, hobbits = hobiti)

Príklady:

Vstup

```
3 4
2 3 2 0
3 4 3 0
2 3 3 1
```

Výstup

```
............
.OXXOXXO....
.X..X..X....
.X..X..X....
.OXXOXXO....
.X..X..X....
.X..X..X....
.OXXOXXOXXO.
............
```

Vstup

```
1 2
2 1
```

Výstup

```
Impossible!
```

Aritmetický obdĺžnik

Keďže sa práve učili aritmetické postupnosti, Peter dostal od učiteľa takúto domácu úlohu: v štvorcovej sieti $R \times C$ štvorčekov sú v niektorých políčkach zapísané čísla. Ostatné políčka sú prázdne. Petrovou úlohou je zapísať do prázdnych políčok čísla tak, aby vznikol aritmetický obdĺžnik. V aritmetickom obdĺžniku by čísla v každom riadku aj v každom stĺpci mali tvoriť aritmetickú postupnosť.

Napríklad, toto je 3×5 aritmetický obdĺžnik:

```
1  3  5  7  9
2  2  2  2  2
3  1 -1 -3 -5
```

Peter je lenivý riešiť túto úlohu ručne, a preto sa rozhodol, že napíšete program, ktorý to urobí za neho.

Súťažná úloha:

Daná je matica **celých čísel**, pričom niektoré z nich sú nahradené bodkami. Treba zistiť, či je možné nahradiť všetky bodky ľubovoľnými **racionálnymi** číslami tak, aby vznikol aritmetický obdĺžnik. Ak existuje takéto riešenie, nájdite ho.

Poznámka: Aritmetickou postupnosťou rozumieme takú postupnosť čísel, v ktorej je rozdiel každých dvoch nasledujúcich prvkov konštantný.

Vstup:

Prvý riadok vstupu obsahuje dve kladné celé čísla R a C: veľkosť matice. Nasledujúcich R riadkov obsahuje po C prvkov, ktoré sú navzájom oddelené medzerou. Každý prvok je buď bodka alebo celé číslo.

Ohraničenia:

Každé zadané číslo v štvorcovej sieti je z intervalu -100 a 100. Za každý z 10 testovacích balíkov sa dá získať do 10 bodov. Pre všetky testy v balíkoch 1 až 9 platí $1 \leq R, C \leq 6$. Pre jednotlivé testovacie balíky platí:

- Balík 1. Všetky čísla v matici sú už vyplnené.
- Balík 2. V každom teste platí buď $R = 1$, alebo $C = 1$.
- Balík 3. V každom teste platí $R = C = 2$.
- Balík 4. Každý z testov má jediné riešenie, ktoré sa dá nájsť postupom naznačeným v prvom príklade uvedenom nižšie.
- Balík 5. Každý z testov má jediné riešenie a toto obsahuje len celé čísla.
- Balík 6. Každý z testov má jediné riešenie.
- Balík 7. Každý z testov má buď jediné riešenie a toto riešenie obsahuje iba celé čísla, alebo nemá riešenie.
- Balík 8. Každý z testov má buď jediné riešenie, alebo nemá riešenie.
- Balík 9. Ľubovoľné testy.
- Balík 10. Ľubovoľné testy, kde $1 \leq R, C \leq 50$.

Výstup:

Ak úloha nemá riešenie, výstupom je jeden riadok s textom „`No solution.`“ (bez úvodzoviek). Ak má úloha viac riešení, vypíšte ľubovoľné z nich.

Výpis riešenia znamená vypísanie R riadkov, pričom každý obsahuje C racionálnych čísel navzájom oddelených medzerou.

Každé racionálne číslo treba vypísať v tvare „N/D“, kde D je kladné a N a D sú navzájom nesúdeliteľné. Ak sa D rovná 1, vynechajte časť „$/D$“.

Jednotlivé čísla vo výstupe by nemali mať viac ako 20 cifier. (Tejto podmienke by nemal byť problém vyhovieť. Dôvodom je to, aby sa zjednodušila kontrola vašich výstupov.)

Príklady:

Vstup

```
3 5
. . 3 . 5
. . . 5 .
. . . . 7
```

Výstup

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

Tento príklad sa môže riešiť nasledovnou metódou: ako prvé si všimnite, že druhé číslo v poslednom stĺpci musí byť 6. Potom vyplníte čísla v riadkoch 1 a 2 a na záver v stĺpcoch 1 až 4.

Vstup

```
1 6
4 . . 0 . .
```

Výstup

```
4 8/3 4/3 0 -4/3 -8/3
```

Vstup

```
1 4
1 2 . 2
```

Výstup

```
No solution.
```

Vstup

```
3 3
1 . .
. 2 .
. . 3
```

Výstup

```
1 2 3
1 2 3
1 2 3
```

Pre tento vstup je to jedno z mnohých riešení.

Ochrankári

Stretli ste už niekedy niekoho z desiatich európsky kráľovských dvorov, argentínsky futbalový tím vrátane ich trénera Diega Maradonu, alebo víťazov Turingovej ceny, či Fieldsovej medaily? Na vyhodnotenie CEOI 2010 sme pozvali množstvo celebrít z celého sveta. Žiaľ, len veľmi málo z nich nám odpovedalo a aj tí nám napísali, že neprídu. Aj napriek tomu si nezabudnite zobrať na vyhodnotenie foťák – kto vie, kto sa tam ukáže?

Ako si viete predstaviť, bezpečnosť takýchto dôležitých hostí je vždy na prvom mieste. Problém je, ako usadiť všetkých ochrankárov do hľadiska tak, aby sme dosiahli čo najväčšiu bezpečnosť.

Hľadisko obsahuje množstvo sedadiel usporiadaných do mriežky. Bezpečnostný technik na základe predpisov určil, koľko ochrankárov má sedieť v každom rade a v každom stĺpci hľadiska.

Súťažná úloha:

Pre každý rad a stĺpec hľadiska je daný požadovaný počet ochrankárov. Táto informácia je zapísaná skomprimovaným spôsobom popísaným nižšie. Zistite, či je možné umiestniť ochrankárov takým spôsobom, že každý rad a každý stĺpec bude obsahovať práve požadovaný počet ochrankárov.

Predpokladajte, že hľadisko je na začiatku prázdne, t.j. ochrankári môžu sedieť hocikde. Na každé sedadlo sa zmestí iba jeden ochrankár.

Vstup:

Vstup začína popisom radov. Prvý riadok vstupu obsahuje jediné kladné číslo R: počet skupín radov. Nasleduje R riadkov, každý z nich obsahuje dve celé čísla: požadovaný počet ochrankárov (ktorý je rovnaký v každom rade skupiny) a počet radov tvoriacich skupinu.

Vstup pokračuje popisom stĺpcov. Nasledujúci riadok obsahuje jediné kladné celé číslo C: počet skupín stĺpcov. Nasleduje C riadkov, každý z nich obsahuje dve kladné celé čísla: požadovaný počet (ktorý je rovnaký v každom stĺpci skupiny) a počet stĺpcov tvoriacich skupinu.

Ohraničenia:

Môžete predpokladať, že celkový počet ochrankárov požadovaných špecifikáciou radov je rovnaký ako celkový počet ochrankárov požadovaných špecifikáciou stĺpcov. Môžete predpokladať, že tento počet je nanajvýš 10^{18}.

Môžete predpokladať, že všetky čísla sú kladné a celé a nepresahujú 10^9.

Ďalej môžete predpokladať, že $1 \leq R, C \leq 200\,000$.

Niekoľko testovacích balíkov v celkovej hodnote 50 bodov spĺňa nasledujúce kritériá: celkový počet radov vo všetkých skupinách bude nanajvýš 2 000; celkový počet stĺpcov vo všetkých skupinách bude nanajvýš 2 000; celkový počet ochrankárov bude nanajvýš 1 000 000.

V testovacom balíku v hodnote ďalších 10 bodov bude v každom testovacom vstupe $R, C \leq 100$.

Výstup:

Vypíšte jediný riadok s číslom „1“ ak je možné splniť obmedzenia, v opačnom prípade vypíšte číslo „0“ (bez úvodzoviek).

Príklady:

Vstup

```
2
2 1
1 2
1
2 2
```

```
XX
X.
.X
```

výstup

```
1
```

Máme dve skupiny radov: prvá skupina má jeden rad, ktorý musí mať dvoch ochrankárov, druhá skupina má dva rady, každý z nich po jednom ochrankárovi. Ďalej máme jednu skupinu stĺpcov: každý z dvoch stĺpcov v skupine musí mať dvoch ochrankárov. Jeden možný spôsob usadenia ochrankárov je zobrazený naľavo pod vstupom.

Vstup

```
2
3 2
1 1
2
3 2
1 1
```

Výstup

```
0
```

Dva rady musia byť úplne vyplnené ochrankármi. Takže v každom stĺpci musia byť aspoň dvaja ochrankári. Avšak posledný stĺpec má mať iba jedného ochrankára, čo je spor.

Zadania druhého súťažného dňa

MP3 prehrávač

Georgov nový MP3 prehrávač má veľa zaujímavých fičúrií. Jednou z nich je tá, že tlačidlá sa môžu zalokovať. Všetky tlačidlá sa automaticky zalokujú po viac ako T sekundách neaktívnosti. Potom, ako je zapnuté zalokovanie, jed-

notlivé tlačidlá nevykonávajú svoju pôvodnú funkciu, ale stlačenie ľubovoľného tlačidla toto zalokovanie vypne.

Napríklad predpokladajme, že $T = 5$ a prehrávač je momentálne zalokovaný. Georg stlačí tlačidlo A, čaká 3 sekundy, stlačí tlačidlo B, čaká 5 sekúnd, stlačí C, čaká 6 sekúnd a stlačí D. V tomto prípade len tlačidlá B a C vykonajú svoje funkcie. Všimnite si, že tlačidlá sa medzi stlačením C a D opäť zablokovali.

Hlasitosť MP3 prehrávača sa mení tlačidlami + a −, ktoré zvyšujú, resp. znižujú úroveň hlasitosti o 1. Úroveň hlasitosti je celé číslo medzi 0 a V_{max}. Stlačenie + pri hlasitosti V_{max} ani stlačenie − pri hlasitosti 0 hlasitosť nezmení.

Súťažná úloha:

Georg nepozná hodnotu T a rozhodol sa ju zistiť nasledujúcim experimentom. Začínajúc so zalokovaným prehrávačom, postláčal postupnosť tlačidiel + a − dĺžky N, a potom si na displeji prehrávača pozrel výslednú hodnotu hlasitosti. Nanešťastie si zabudol zaznačiť počiatočnú hodnotu hlasitosti ešte pred prvým stlačením tlačidla. Neznámu počiatočnú hlasitosť si označme ako V_1 a známu výslednú hlasitosť označme V_2.

Máme danú hodnotu V_2 a zoznam tlačidiel v poradí, v akom ich Georg stláčal. Pre každé tlačidlo dostávame typ (+ alebo −) a počet sekúnd, ktoré uplynuli od začiatku experimentu až po moment stlačenia tlačidla. Úlohou je zistiť najväčšiu možnú **celočíselnú** hodnotu T, ktorá zodpovedá priebehu experimentu.

Vstup:

Prvý riadok obsahuje tri medzerou oddelené celé čísla N, V_{max} a V_2 ($0 \leq V_2 \leq V_{max}$). Každý z nasledujúcich N riadkov obsahuje popis jedného tlačidla postupnosti: znak + alebo −, medzeru a celé číslo C_i ($0 \leq C_i \leq 2 \cdot 10^9$), ktoré vyjadruje počet sekúnd od začiatku experimentu. Môžete predpokladať, že stláčania sú usporiadané vzostupne a všetky časy sú rôzne (t. j. $C_i < C_{i+1}$ pre všetky $1 \leq i < N$).

Ohraničenia:

Môžete predpokladať, že $2 \leq N \leq 100\,000$ a $2 \leq V_{max} \leq 5\,000$.

V testovacích vstupoch za 40 bodov $N \leq 4\,000$.

V testovacích vstupoch za 70 bodov $N \cdot V_{max} \leq 400\,000$.

Výstup:

Ak T môže byť ľubovoľne veľké, vypíšte riadok so slovom „`infinity`“ (bez úvodzoviek).

Inak vypíšte jeden riadok s dvoma celými číslami T a V_1 oddelenými jednou

medzerou.

Tieto hodnoty musia byť také, že ak by experiment so zalokovávacím časom T začal s počiatočnou hodnotou hlasitosti V_1, tak by sme danou postupnosťou stlačení dosiahli výslednú hlasitosť V_2. Ak existuje viac možných odpovedí, vypíšte tú, v ktorej je T najväčšie; ak aj takýchto existuje viac, vypíšte odpoveď s najväčším V_1.

(Všimnite si, že vždy existuje aspoň jedno riešenie: pre $T = 0$ žiadne tlačidlo nevykoná žiadnu funkciu, a teda dostávame $V_1 = V_2$.)

Príklady:

Vstup

```
6 4 3
- 0
+ 8
+ 9
+ 13
- 19
- 24
```

Výstup

```
5 4
```

Pre $T = 5$ tlačidlá vykonajú tieto akcie: odlokuj, odlokuj, +, +, odlokuj, −.
Pre každé $V_1 \in \{2, 3, 4\}$ dostávame $V_2 = 3$. Všimnite si, že vo výstupe je uvedené najväčšie možné V_1.
Pre $T \geq 6$ by obidve posledné stlačenia boli aktívne, a teda by bolo nemožné dosiahnuť $V_2 = 3$.

Vstup

```
3 10 10
+ 1
+ 2
+ 47
```

Výstup

```
infinity
```

Ak $V_1 = 10$, potom pre každé T dostaneme $V_2 = 10$.

PIN

Martin sa zamestnal ako administrátor vo veľkej firme. Firma má autorizačný systém, ktorý sa nezmenil od 80-tych rokov. Každý zamestnanec ma štvormiestny PIN pozostávajúci z písmen a číslic, ktorým sa prihlasuje do systému.

Martinovi sa toto vôbec, ale vôbec nepáči. Predstavte si napríklad, že sú dvaja ľudia, ktorých PINy sa odlišujú iba na jedinom mieste, napríklad `61ab`

a `62ab`. Ak prvý z nich stlačí `2` namiesto `1`, systém ho stále bez problémov prihlási. Martin sa rozhodol, že urobí štatistiku o tom, koľko párov v súčasnosti používaných PINov sa líši na jednej, dvoch, troch a štyroch pozíciách a týmito číslami presvedčí šéfa, že veci treba zmeniť.

Súťažná úloha:

Daný je zoznam PINov a celé číslo D. Nájdite počet párov PINov, ktoré sa líšia presne na D pozíciách.

Vstup:

Prvý riadok vstupu obsahuje dve celé čísla oddelené medzerou, N a D. N je počet PINov a D je počet rozdielov. Každý z nasledujúcich N riadkov obsahuje jeden PIN.

Ohraničenia:

Môžete predpokladať, že vo všetkých testoch $2 \leq N \leq 50\,000$ a $1 \leq D \leq 4$.

Každý PIN má 4 znaky, pričom každý znak je buď číslica, alebo malé písmeno anglickej abecedy. Môžete tiež predpokladať, že všetky PINy na vstupe sú navzájom rôzne.

V testoch hodných 15 bodov $N \leq 2000$.

V testoch hodných 60 bodov $D \leq 2$. Spomedzi nich pre testy hodné 30 bodov platí $D = 1$.

V testoch za 75 bodov každý PIN obsahuje iba číslice a písmená od „`a`“ do „`f`“, takže sa na ne možno pozerať ako na hexadecimálne čísla.

Výstup:

Vypíšte jediný riadok s jediným číslom: počtom párov PINov, ktoré sa líšia presne na D pozíciách.

Príklady:

Vstup	Výstup

`4 1`
`0000`
`a010`
`0202`
`a0e2`
	`0`

V tomto prípade sa každé dva PINy líšia na dvoch alebo viacerých pozíciách.

Vstup

```
4 2
0000
a010
0202
a0e2
```

Výstup

```
3
```

Tri páry PINov sa líšia presne na dvoch pozíciách: (0000,a010), (0000,0202) a (a010,a0e2).

Obrovitánska veža

Starí Babylončania sa rozhodli postaviť obrovitánsku vežu. Veža pozostáva z N kockatých blokov postavených jeden na druhom. Babylončania sa zo svojho predchádzajúceho neúspešného pokusu naučili, že ak pri stavbe položia na menší blok oveľa väčší blok, veža sa zrúti.

Súťažná úloha:

Každé dva kockaté bloky vyzerajú rôzne, aj keď sú rovnakej veľkosti. Pre každý blok je daná dĺžka jeho strany. Dané je tiež celé číslo D, ktoré má nasledovný význam: Ak má blok A stranu o viac ako D dlhšiu ako blok B, tak nesmieme položiť blok A priamo na blok B.

Vypočítajte počet rôznych spôsobov, ktorými možno postaviť vežu s použitím **všetkých** kockatých blokov. Keďže tento počet môže byť veľmi veľký, vypíšte iba výsledok modulo $10^9 + 9$.

Vstup:

Prvý riadok vstupu obsahuje dve kladné celé čísla N a D: počet kockatých blokov a toleranciu.

Druhý riadok obsahuje N celých čísel oddelených medzerami. Každé z týchto čísel reprezentuje veľkosť jedného kockatého bloku.

Ohraničenia:

Všetky čísla na vstupe sú kladné a celé a nepresahujú 10^9.

N je vždy aspoň 2.

V testovacích prípadoch hodných 70 bodov bude N nanajvýš 70.

Z toho v testovacích prípadoch hodných 45 bodov bude N nanajvýš 20.

Z toho v testovacích prípadoch hodných 10 bodov bude N nanajvýš 10.

V testovacích prípadoch za 30 bodov celkový počet možných veží nepresiahne 1 000 000.

V posledných šiestich testovacích prípadoch za 30 bodov je hodnota N vyššia ako 70. V týchto prípadoch neudávame horné obmedzenie na N.

Výstup:

Vypíšte jeden riadok s jediným číslom: počtom veží ktoré možno postaviť, modulo 1 000 000 009.

Príklady:

Vstup

```
4 1
1 2 3 100
```

Výstup

```
4
```

Prvé tri bloky možno usporiadať ľubovoľne okrem poradí 2,1,3 alebo 1,3,2. Posledný blok musí byť na spodku.

Vstup

```
6 9
10 20 20 10 10 20
```

Výstup

```
36
```

Blok o veľkosti 20 nemôžeme postaviť na blok o veľkosti 10. Bloky o veľkosti 10 možno usporiadať šiestimi spôsobmi, bloky o veľkosti 20 možno takisto usporiadať šiestimi spôsobmi.

Medzinárodná olympiáda v informatike

V roku 2010 sa Medzinárodná olympiáda v informatike (IOI) konala na kanadskej University of Waterloo v rovnomennom meste (približne hodinu od Toronta). Ako lídri zastupujúci našu krajinu pri hlasovaniach sa tejto IOI zúčastnili doc. RNDr. Gabriela Andrejková, CSc. z PF UPJŠ v Košiciach a Mgr. Juliana Lipková z FMFI UK v Bratislave. Ako člen medzinárodnej odbornej komisie (ISC) sa tejto IOI zúčastnil RNDr. Michal Forišek, PhD. z FMFI UK v Bratislave.

Našu krajinu tento rok reprezentovala táto štvorica stredoškolákov: Ladislav Bačo z Gymnázia Poštová, Košice, Matej Balog z Gymnázia Grösslingová, Bratislava, Tomáš Belan zo Školy pre mimoriadne nadané deti, Bratislava, a Ján Hozza z Gymnázia Jura Hronca, Bratislava. Výsledky našich súťažiacich uvádzame zhrnuté v tabuľke.

Hlavnou atrakciou v rámci sprievodného programu bola tento rok návšteva Niagarských vodopádov. Súťažiacich tiež čakal voľný deň, ktorý strávili v zábavnom parku Canada's Wonderland.

Meno	1. deň				2. deň				$\sum$	medaila
Ján Hozza	100	77	79	100	50	51	100	100	657	strieborná
Tomáš Belan	60	98	81	100	50	50	100	100	639	strieborná
Ladislav Bačo	60	72	25	100	50	55	100	100	562	bronzová
Matej Balog	40	31	50	100	50	73	100	100	544	bronzová

RNDr. Michal Forišek PhD., Slovenská komisia OI
Dvadsiaty piaty ročník Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava, 2010
144 strán, náklad 500 výtlačkov
Neprešlo jazykovou úpravou
ISBN 978-80-8072-112-1