Ako sa vyhnúť častým chybám?

Ahojte, tento text je tu preto, že veľa z vás robí stále tie isté chyby v riešeniach, dokonca niektorí aj opakovane tie isté. A boli by sme radi, keby ste sa toho v budúcnosti vyvarovali, aby sme vám nemuseli stále strhávať body. Aj nás viac teší čítať dokonalé riešenia, aj vás bude určite viac tešiť, keď dostanete plný počet bodov :).

Povinné čítanie, ešte predtým ako budete čítať toto, je určite: `http://www.ksp.sk/wiki/Seminar/Riesenie`. Keď to už máte naštudované, môžeme začať :). Všetky príklady popisov budú riešeniami úlohy Zmedvedieva sa, ktorej zadanie nájdete tu `http://www.ksp.sk/wiki/uploads/Zadania/ps312.pdf` a vzorák k nej tu `http://www.ksp.sk/wiki/uploads/Zadania/vzor312.pdf`.

Čo to vlastne je ten popis myšlienky a ako to má vyzerať?

Predstavte si, že sa snažíte niekomu, kto vôbec nevie programovať povedať, čo váš program robí. Nepoužijete ani slovíčko cyklus (ak to nie je nutné), ani slovo premenná, ani nič podobné. Nebudete ho zaťažovať dĺžkami polí a podobnými technickými detailami.

Ako by ste mu popísali vaše riešenie úlohy Zmedvedieva sa? (Berme do úvahy riešenie s pamäťou $O(r \cdot s)$.) Asi nejak takto: zapamätám si mapku, pozriem sa na 4 najväčšie obdĺžniky, ktoré neobsahujú medveďa a v každom zrátam počet miest na stavanie stanov. Vyberiem z nich ten, kde je ich najviac a počet miest na stany v ňom je riešením úlohy. Jednoduché, zrozumiteľné.

Ako veľmi podrobný má byť popis algoritmu a dátových štruktúr?

Zo skúseností môžeme povedať, že pravdepodobne menej podrobný ako ten, ktorý ste napísali :). Pokiaľ v programe používate dva vnorené for cykly, naozaj stačí povedať, že tam sú dva for cykly, v ktorých napríklad prechádzate mapu. My si program vieme pozrieť a detaily pochopíme. Nemusíte písať, že ste si vytvorili premennú i, ktorú ste si inicializovali na nulu a potom vám ide for cyklus od nula po $n - 1$, kde n je počet prvkov.

Ak v programe hľadáte maximum z n prvkov, stačí napísať napríklad, že v cykle si nájdem maximum. Znova nemusíte písať, že máte premennú max, ktorú si najskôr inicializujete na nulu a potom máte for cyklus, v ktorom ju postupne porovnávate s každým prvkom a na konci vypíšete. To sú všetko veci, ktoré za vás hovorí váš kód. Nezabudnite, že opravovateľ má prístup k programu, ktorý ste odovzdali.

Mali by ste sa skôr sústrediť na veci, ktoré nie sú z kódu zjavné. Ak máte v programe nejaké funkcie alebo zložitejšie dátové štruktúry je fajn popísať aspoň v skratke, čo robia. Ale naozaj stačí v skratke. Napríklad: „Funkcia `spocitaj(r1, s1, r2, s2)` spočíta počet jednotiek v políčkach, ktoré sú na riadkoch r1 až r2 vrátane a v stĺpcoch s1 až s2 vrátane.“

Prečo musím písať dôkaz správnosti? A ako má ten dôkaz vyzerať?

Pod dôkazom sa nemyslí nejaký strašne formálny matematický dôkaz s kopou gréckych symbolov a logických spojok, ale skôr dôvod, pre ktorý veríte, že vaše riešenie funguje. Písať by ste ho mali z dvoch dôvodov:

Jednak preto, aby bolo jasné, že ste len nevyskúšali na testovači prvú vec, ktorá vám napadla, ale naozaj aj rozumiete, prečo a ako ten váš program funguje.

Po druhé, pri dokazovaní správnosti si sami preveríte či je váš program správny a odhalíte prípadnú chybu v myšlienke skôr, ako sa natrápite programovaním zlého algoritmu. Dávajte si pozor najmä pri tvrdeniach typu „oplatí sa nám zobrať menšie číslo, oplatí sa nám odseknúť vrchol s najväčším počtom susedov...“ Napríklad ak by vaše riešenie úlohy Zmedvedieva sa vyzeralo „spomedzi všetkých obdĺžnikov bez medveďa vyberiem ten s najväčším obsahom“, tak nie je korektné, napríklad na vstupe `XXMS` by nefungovalo.

Správny dôkaz vzorového riešenia tejto úlohy by mohol vyzerať napríklad takto:

Pokiaľ nejaký obdĺžnik môžeme rozšíriť (alebo predĺžiť) do ľubovoľnej strany bez toho, aby sme vyšli z mapy alebo zobrali políčko s medveďom, tak nový obdĺžnik nemôže mať menej políčok S ako pôvodný. (Políčka sme iba pridávali.) Z toho dôvodu nám stačí uvažovať len také obdĺžniky, ktoré sa už nedajú rozšíriť (a predĺžiť). Také obdĺžniky sú len 4, všetky sa z troch strán dotýkajú okraju mapy a z jednej strany medveďa a líšia sa tým, z ktorej strany sa dotýkajú medveďa. Teda nám stačí skontrolovať spomenuté 4 obdĺžniky. (Teraz je fajn ešte pridať ilustračný obrázok, čo ale väčšina z vás v tejto úlohe urobila a aj ste za to boli patrične pochválení :).) A toto úplne ako dôkaz stačí. Že to nevyzerá až tak hrozne?

Čoho sa naozaj pri popise treba vyvarovať?

Najhorší popis, aký existuje je copy-paste zdrojového kódu a ku každému riadku napísaný komentár, čo sa v tom riadku robí. Napríklad: `int x=0; do premennej x som priradil nulu`. TOTO NAOZAJ NE-

ROBTE!!! Od opravovateľa môžete očakávať aspoň minimálny stupeň inteligencie a to, že už niekedy videl kód v `C++/Pascale/Pythone`. Ak chcete skopírovať kód do popisu, pokojne to urobte, ale najskôr napíšte nejaký popis a až pod to skopírujte kód. Prosíme nekombinovať a nemixovať :). Ak nechcete, vôbec kód v popise neuvádzajte. My si ho vieme pozrieť aj tak. (Samozrejme, pokiaľ nejde o nejakú špeciálnu teoretickú úlohu, kde sú ale presné pokyny zvyčajne napísane v zadaní.)

Prečo zložitosť algoritmu nie je konštantná

Niektorí z vás sa dopúšťajú nasledovnej chyby: V programe máme jediné pole `mapa[2000][2000]`, a v zadaní máme napísané, že r, s sú najviac 2 000. Na základe toho prehlásime, že pamäťová zložitosť je konštantná, lebo je to stále zhruba 4 000 000 `int`ov. Rovnako si povieme že program spraví najviac 4 000 000 krokov a to je predsa tiež konštanta.

To je ale zle. Vy totiž máte odhadovať zložitosť samotného **algoritmu**, nie **programu**. A váš algoritmus by mal fungovať pre ľubovoľné r a s. Áno, váš teoretický algoritmus, tak ako ste popísali jeho myšlienku v popise, musí fungovať aj pre $r = 1\,000\,000\,000$, aj keď by sa nezmestil do pamäte a bežal by pol roka. Preto zatiaľ čo vo vašom kóde máte pole veľkosti 2000×2000, tak prvý z algoritmov vo vzoráku úlohy Zmedvedieva sa musí používať pole `mapa[r][s]`. Jeho pamäťová zložitosť bude teda $O(r \cdot s)$. Druhý algoritmus vo vzoráku má analogicky zložitosť $O(s)$, lebo si vystačí s poľom dĺžky s.

Limity na veľkosť vstupu v zadaní slúžia len pre váš program, aby ste nemuseli používať dynamické štruktúry, keď ich netreba a aby ste vedeli odhadnúť, aký dobrý musí byť váš program, aby dostal plný počet bodov za prax. Vo vašom kóde môžete naďalej predpokladať platnosť týchto obmedzení.

V popise algoritmu však považujte premenné spomenuté v zadaní naozaj za ľubovoľne veľké. (Dolné limity zvyčajne platia naďalej, napríklad aj v úlohe Zmedvedieva môžete aj v popise predpokladať, že r a s sú aspoň 1.)

Prečo nie je moja časová zložitosť $O(n^2)$?

Ak sa pozrieme na úlohu Zmedvedieva sa, môžete si všimnúť, že v zadaní sa nikde premenná n nevyskytuje. V zadaní sú definované iba premenné r a s. A vy aj tak do popisu napíšete, že časová zložitosť je $O(n^2)$. Lenže čo je podľa vás n? Je $n = r \cdot s$? Alebo $n^2 = r \cdot s$? Pokiaľ to nenapíšete, my na to nemáme ako prísť a preto za takýto popis časovej zložitosti nedostanete body. Treba používať premenné, ktoré boli definované v zadaní, prípadne ak používate vlastné, treba napísať, čo vaša premenná predstavuje. Napríklad: moja časová zložitosť je $O(\ell \cdot n)$, kde n predstavuje počet slov na vstupe a ℓ dĺžku najdlhšieho slova. Takisto netreba zabudnúť napísať, prečo je vaša zložitosť taká, aká je, a od čoho závisí.

Sú niekde moje opravené riešenia?

Áno, každé vaše riešenie, ktoré malo aj popis, si vedúci pozrú a nielenže ho obodujú, ale zvyčajne vám k nemu napíšu aj komentár, kde vás pochvália alebo vám napíšu, kde máte chybu. Dokonca, ak je to v ich silách, napíšu vám aj prečo ste nedostali body z testovača.

Tieto opravené riešenia nájdete presne tam, kde ste ich predtým odovzdali (konkrétne: kliknete na odovzdané riešenia, vyberiete si sadu a nájdete daný popis). Ak sa skončí séria a na stránke je už oznam, že riešenia sú opravené, môžete si vaše popisy znova stiahnuť a tentoraz tam budete mať pravdepodobne niečo dopísané :).

Záverečné rady

Čítajte si komentáre opravovateľov k svojim riešeniam, zvyčajne sa tam dozviete kopu užitočných vecí, ktoré sa zameriavajú presne na to, kde robíte chyby. Čítajte aj vzoráky, ak už pre nič iné, aspoň si trochu odkukáte štýl, akým písať popisy. Samozrejme, nechceme od vás aby ste rozoberali v popise všetky možné riešenia úlohy, stačí jedno, ale poriadne :).

Korešpondenčný seminár z programovania
XXXI. ročník, 2013/14
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Úlohy 2. kola letnej časti

Termín odoslania riešení tejto série je pondelok **26. mája 2014**.

1. Zatváračka matriošiek

kat. Z; 0 b za popis, 10 b za program

Už ste niekedy počuli ako sa vyrábajú matriošky? Zoberiete jednu veľkú krabicu (vyrábačku), z ktorej naraz vypadáva celá sada desiatich matriošiek, a druhú (zatváračku), ktorá takúto sadu zoberie a poskladá. Servác si takéto dve krabice kúpil. Postavil ich na opačné konce svojej dlhej výrobnej haly a zistil, že matriošky treba medzi koncami haly aj prenášať. Prepojil ich teda desiatimi pásovými linikami označenými `A - J`. Celý natešený spustil výrobu, počkal pol hodinku, kým sa po páse prepravila prvá sada matriošiek k zatváračke (každá matrioška mala, samozrejme, vlastný pás), tá s nimi chvíľu šramotila a následne zakapala.

Kde bol problém? Zatváračka (prirodzene) očakávala, že matriošky k nej prídu po pásoch utriedné, ale vyrábačka bola kúpená za akciovú cenu a teda vyrobné matriošky vypľúvala na pásy ľubovoľne. Jediným riešením tohto problému je postupné vymieňanie matriošiek medzi pásmi, počas dlhej cesty k zatváračke. Na to slúžia vymieňače. Vymieňač umiestníte nad nejaké dva pásy a ak zistí, že matrioška, ktorá prechádza pod pravým ramenom je menšia, ako tá ktorá prechádza pod ľavým, tak ich bleskurýchle vymení. Ak vhodne umiestnite dostatočne veľa vymieňačov za seba, zabezpečíte, aby na konci boli matriošky utriedené zľava doprava tak, ako to zatváračka očakáva. Najmenšia matrioška má prísť po páse `A`, druhá najmenšia po páse `B`, atď. až po najväčšiu, ktorá má prísť po páse `J`.

A presne to bude Vašou úlohou. Navrhnite, ako má Servác umiestniť vymieňačky tak, aby matriošky vždy utriedili a bolo ich čo najmenej.

Úloha

Celé sa to dá predstaviť aj takto: Máme 10 premenných `A - J`, v ktorých je uložených 10 rôznych čísel. Máte k dispozícii jednu jedinú operáciu:
`Pozri sa na dve premenné a ak v tej s abecedne skorším písmenom je väčšie číslo, tak vymeň ich obsahy.` Nájdite postupnosť takýchto operácií, ktorá vždy utriedi čísla v premenných (v poradí podľa abecedy), bez ohľadu na to, aké čísla boli v premenných na začiatku.

Bodovanie

Body dostávate, iba ak je vaša postupnosť správna. Počet bodov závisí od počtu použitých operácií, podľa nasledovnej tabuľky:

Počet operácií	Body
0 - 38	10
39 - 41	9
42 - 44	8
45 - 47	7
48 - 50	6
51 - 60	5
61 - 80	4
81 - 100	3
101 - 200	2
201 - 1000	1

Odovzdávanie

V tejto úlohe odovzdávate jeden `.txt` súbor, v ktorom budú po riadkoch popísané jednotlivé operácie. Každý riadok má mať tvar `<prvá premenná ako jedno veľké písmeno><medzera><druhá premenná ako jedno veľké písmeno><koniec riadku>`. Počet bodov ako aj prípadné chyby sa dozviete hneď po submitnutí.

Príklad

Príklad uvediem pre prípad, že by premenné boli iba 4.

```
A B
B C
C D
A B
B C
C D
A B
B C
C D
```

Po prvých troch operáciách je najväšie číslo v D, po druhých troch je aj na páse C správna matrioška. Na konci sú matriošky utriedené. Štyri premenné sa dajú utriediť aj na menej operácii, odporúčame vám skúsiť si to.

2. (Z)lomový starosta Eman

kat. Z; 7 b za popis, 3 b za program

Eman je zaujímavá osôbka. Napriek tomu, že niektorí nemenovaní vedúci si o ňom mysleli, že vlastne vôbec neexistuje, zvládol kandidovať za starostu a byť dôstojným nasledovníkom legendárneho účastníka Jodíka (nech je mu brnenská akademická pôda ľahká). Eman má však aj mnohé nedostatky. Hlavne vedomostné. Napriek tomu, že je z Ruska, nevie, kto bola Lajka alebo čo to je Transsibírska magistrála. Ako nádejný kandidát na starostu by tiež mohol aspoň tušiť, čo je to inflácia[1]. Takisto aj v matematike má Emo „menšie“ medzery.

Náhodou sa na sústredení ocitol na prednáške o rastúcich postupnostiach. Keď ku koncu prednášky zistil, čo to tá postupnosť je, tabuľu zotreli, prednáška skončila a Emo je teraz bezradný. Rád by zistil začiatky niektorých postupností, ale tie už sú skoro celé zotreté. Z každej postupnosti mu zostali na tabuli napísané už len dve čísla, o ktorých vie, že boli v postupnosti blízko pri sebe (buď hneď za sebou, alebo medzi nimi bolo jedno číslo). Pomôžte mu teraz zrekonštruovať ich začiatky...

Úloha

Na vstupe dostanete dve kladné celé čísla, ktoré sú členmi nejakej rastúcej postupnosti kladných celých čísel. (To, že je rastúca, okrem iného znamená, že každé dva členy tejto postupnosti sú navzájom rôzne.) Čísla, ktoré dostanete na vstupe, sa v postupnosti vyskytujú buď hneď za sebou alebo sa medzi nimi vyskytuje ešte práve jedno číslo.

O všetkých číslach v tejto postupnosti (okrem prvých dvoch) platí, že každé číslo je súčtom dvoch predošlých. Teda napríklad tretie číslo je súčtom prvého a druhého. Štvrté je súčtom druhého a tretieho.

Nájdite najmenšiu dvojicu kladných celých čísel, ktoré mohli túto postupnosť začínať. (Prvé číslo musí byť čo najmenšie, v prípade remízy aj druhé.)

Formát vstupu

Na vstupe dostanete dve kladné celé čísla n a m, môžete predpokladať, že $(1 \leq n < m \leq 10^{16})$. Na uloženie týchto premenných budete potrebovať premennú typu `int64` v Pascale alebo `long long int` v C++.

Formát výstupu

Na výstup vypíšte dve medzerou oddelené kladné celé čísla, prvý a druhý prvok postupnosti.

Príklady

vstup	výstup
`7 11`	`1 3`

Táto postupnosť by vyzerala takto: 1, 3, 4, 7, 11

vstup	výstup
`4 11`	`1 3`

Znova by najlepšia postupnosť vyzerala takto: 1, 3, 4, 7, 11, medzi čísla na vstupe sme ešte vložili 7.

[1] Ďalšie informácie o Emovej kandidatúre nájdete tu: `http://people.ksp.sk/~roman/sustr/kampane/eman/`

3. Zoskok

kat. Z; 8 b za popis, 4 b za program

Super Máriovi sa zunoval jeho pôvodný, idealistický a hlavne absolútne nefyzikálny 2D svet. Najviac sa mu nepáčil, spôsob akým skákal a padal. Predsa len, kto to kedy videl, aby človek počas pádu niekoľkokrát zmenil smer, ktorým letí? To predsa nedáva zmysel! Rýchlo zbalil kufre, rozlúčil sa s korytnačkami a princeznami, a zdúchol na výpravu do paralelných vesmírov. Nakoniec našiel presne taký, ako hľadal.

Nová realita mu však dala silnú facku. Zatiaľčo predtým nemal problém kamkoľvek doskočiť, v tomto novom svete je gravitácia tak silná, že nielenže nemôže skákať vôbec, ale keď začne padať, tak padá kolmo nadol, až kým do niečoho nenarazí. Nový svet je taktiež 2D a obsahuje kusy pôdy vznášajúcej sa vo vzduchu, na ktorých môže Mário stáť. Na spodku krajiny je naviac pevná solídna zem.

A aby toho nebolo málo, nášho superhrdinu ani v novom svete neobišla láska. Lenže čo si počať, keď jediný chybný krok môže znamenať doživotné odlúčenie od osoby, ktorú má tak rád? Preto je veľmi dôležité vybrať si správne miesto na rande. Také miesto nemôže byť úplne hocikde. Musí sa naň vedieť dostať[2]. Naviac, miesto stretnutia sa musí nachádzať na zemi vo výške 0 a nie na kuse lietajúcej pôdy.

Pomôžte superhrdinovi v núdzi, odmena od takého boháča[3] bude isto veľmi lákavá.

Úloha

Na vstupe dostanete popis bočného pohľadu na Máriov svet ako mriežku prázdnych a plných políčok. Ak Mário stojí na zemi (políčko pod ním je plné), môže sa pohnúť doprava alebo doľava, ak tam je prázdne políčko. Keď je pod ním prázdne políčko, padá dole.

Začínať môže na ľubovoľnom voľnom políčku vrchného riadka. Zistite, na ktoré voľné políčka spodného riadka sa vie dostať.

Formát vstupu

V prvom riadku sú čísla r a s ($1 \leq r, s \leq 1000$), rozmery krajiny. Ďalej nasleduje r riadkov po s znakoch, popis krajiny, kde # (mriežka) znamená, že dané políčko je plné, a . (bodka) znamená, že je dané políčko prázdne. Pod najspodnejším riadkom sa nachádzajú už len plné políčka, Mário teda nemôže vypadnúť z mapy.

Formát výstupu

Vypíšte čísla všetkých políčok zo spodného riadka, na ktoré sa Mário vie dostať. Tieto čísla vypíšte usporiadane, od najmenšieho po najväčšie. Políčka na spodnom riadku číslujeme zľava doprava číslami od 1 po s.

Príklad

vstup

```
4 4
.#..
.###
..#.
.#..
```

výstup

```
1
```

Druhé políčko je obsadené a na zvyšné dve sa nemá ako dostať.

vstup

```
4 4
##.#
##.#
##.#
#...
```

výstup

```
2 3 4
```

Keď dosiahne zem, vie sa po nej pohybovať doprava i doľava.

vstup

```
2 2
##
..
```

výstup

```

```

□

Na žiadne spodné políčko sa nevie dostať, výstupom je prázdny riadok (Nezabudnite, že aj keď sa žiadne číslo nevypíše, musí byť tento riadok zalomený!)

[2] Ako sa tam dostane jeho vysnená princezná, je už jej problém.

[3] Jeho hlava proti zberaniu majetku celý čas výrazne protestovala.

4. Zranenie

kat. Z a O; 10 b za popis, 5 b za program

Všade okolo mňa bola tma. Bežal som po ošarpanej poľnej cestičke s cieľom čo najrýchlejšie zapáliť sviečku. Jediné, na čom záležalo, bolo dostať ju horiacu na koniec cesty a čas sa krátil. Nohy sa mi rýchlo prepletali, vietor svišťal popri ušiach a ani som nevnímal terén, po ktorom som bežal. A to bola chyba. Znenazdajky sa mi ľavá noha zachytila o vyčnievajúci kameň a ja som letel v ústrety zemi.

Prvá vec, ktorú si pamätám, bola, že sedím na zemi a cítim strašnú bolesť v ľavom pleci. Skúsim ním opatrne pohnúť, ale prebehne mi ním taká bolesť, že skoro zamdliem. Okolo sa zbiehajú ľudia a pýtajú sa, čo sa stalo. Ja mám zlé tušenie, že viem, prečo ma to plece tak strašne bolí – mám zlomenú kľúčnu kosť.

Všetci okolo mňa bezradne stoja, netušia ako mi pomôcť. Pozriem sa smerom k vychádzajúcemu mesiacu. A tam neďaleko na kopci stojí on. Zdravotník Žaba s vejúcim plášťom lemovaný mesačným svetlom. Keď sa rozbehne smerom ku mne viem, že záchrana sa blíži a že sa mi už nič zlé nestane.

Keď pribehne, hneď volá sanitku a profesionálne mi zaväzuje ruku do trojcípej šatky. S rukou fixovanou pri páse sa mi hýbe hneď lepšie. Napriek tomu ma však čaká dlhá cesta ku chate, kam má prísť sanitka, a ja mám pocit, že odpadnem. Našťastie si je Žaba plne vedomý tohoto nebezpečia a zadáva mi nasledujúcu úlohu, aby som si rozptýlil myslenie a sústredil sa na niečo iné ako bolesť.

Úloha

Dostanete zadané jedno číslo n. Z neho môžete vyrábať nové čísla tak, že poprehadzujete (spermutujete) jeho cifry – žiadnu ale nesmiete pridať ani zahodiť. Napríklad z čísla 198 môžete vyrobiť 189, 198, 819, 891, 918 alebo 981. Vzniknuté číslo môže dokonca začínať nulami alebo sa rovnať pôvodnému číslu.

Vašou úlohou je nájsť dve čísla, ktoré sa dajú vyrobiť z čísla n a ich súčet končí čo najväčším počtom núl.

Formát vstupu

Na prvom riadku je jediné číslo n, ktoré má najviac 10^5 cifier a nezačína prebytočnými nulami.

V dvoch z piatich testovacích sád navyše platí, že číslo n **neobsahuje** cifru 0.

Formát výstupu

Vypíšte dve čísla, ktoré vzniknú spermutovaním cifier čísla n tak, aby súčet týchto čísiel mal čo najviac núl na konci. Ak niektorá permutácia obsahuje na začiatku prebytočné nuly, vypíšte aj tie.

Príklad

vstup

```
198
```

výstup

```
981
819
```

Súčet týchto dvoch čísiel je 1800 *a má na konci dve nuly. Väčší počet nie je možné dosiahnuť.*

vstup

```
500
```

výstup

```
500
500
```

V tomto príklade je optimálne dvakrát použiť identickú permutáciu, čím dostaneme číslo 1000 *obsahujúce tri nuly na konci.*

5. Ono to rieši samo! (diel 4: celočíselné LP)

kat. Z a O; 16 b za popis, 0 b za program

K tejto úlohe si budeš potrebovať od nás stiahnuť súbor s testovacími dátami. Nájdeš ho na adrese `http://www.ksp.sk/wiki/uploads/Zadania/ksp31-4-5-vstupy.zip`.

V minulej sérii sme sa zoznámili s lineárnym programovaním a ukázali sme si, ako pomocou neho riešiť niekoľko úloh. Všetky mali „spoločného menovateľa“: počas riešenia sme používali reálne čísla. V niektorých z úloh sme mali aj ako výsledok reálne číslo, v iných sme si navyše ukázali, že optimálny výsledok vždy vyjde celočíselný.

Dnes sa spolu pozrieme na *celočíselné* lineárne programy (integer linear programs, ILP). Podobne ako v *reálnych* budeme mať aj tu sadu premenných, a program bude vlastne len sada lineárnych rovníc a nerovníc,

ktoré majú tieto premenné spĺnať (a navyše jedna lineárna funkcia, ktorej hodnotu sa snažíme maximalizovať). Celý rozdiel bude v tom, že tentokrát ako hodnoty premenných povolíme len nezáporné *celé* čísla.[4]

Okamžite sa však ukáže, že táto zdanlivo drobná zmena bude mať prevratné následky. Na jednej strane nám obmedzenie na celé čísla umožní modelovať omnoho viac problémov, na druhej strane už však vo všeobecnosti nepôjde tieto problémy riešiť efektívne. ILP solvery teda budú za nás robiť kopu temnej mágie a optimalizácií, podobne ako tomu bolo napríklad u SAT solverov, ktoré sme stretli v prvej sérii jesennej časti KSP.

Boolovské premenné a logické spojky

V ILP môžeme priamočiaro používať boolovské premenné: stačí si na premennú `x` pridať obmedzenie `x <= 1` a ajhľa, máme čo sme chceli – premennú s dvomi hodnotami: 0 (false) a 1 (true).

Menej očividné asi je, že s týmito premennými vieme následne robiť všetky potrebné logické operácie. Zjavná je negácia: negáciou premennej `x` je premenná `y` pre ktorú platí `y = 1 - x`. Ešte potrebujeme logický and a logický or. Ako na tie?

Majme teda dve boolovské premenné `x` a `y`. Zoberme si boolovskú premennú `z`, pre ktorú navyše platí `z >= x`, `z >= y` a `z <= x+y`. Ak `x` aj `y` sú obe rovné 0, dostaneme z týchto podmienok jediné riešenie `z=0`. A ak je aspoň jedna z premenných `x` a `y` rovná 1, dostaneme jediné riešenie `z=1`. Takto sme teda len pomocou lineárnych nerovností definovali novú premennú `z`, ktorej hodnota je logickým or-om hodnôt `x` a `y`.

Podobne vieme definovať aj logický and: tým bude premenná `w` spĺňajúca `w <= x`, `w <= y` a `w >= x+y-1`.

No a teraz by už malo byť jasné, že vieme zobrať ľubovoľný SAT program a prepísať ho do ekvivalentného ILP. A tu už začíname aj tušiť, prečože je to riešenie ILP vo všeobecnosti také ťažké.

Problém batoha

Niektoré problémy sa výrazne ľahšie ako SAT programami formulujú v ILP podobe. Pozrime sa napríklad na problém batoha. V jeho najjednoduchšej podobe tento problém vyzerá nasledovne: máme n vecí s kladnými celočíselnými hmotnosťami $w_1, \ldots, w_n$ a batoh s dostatočným objemom a nosnosťou z. Chceli by sme vybrať takú podmnožinu vecí, aby sme batoh čo najlepšie využili – súčet hmotností vybraných vecí sa musí čo najviac priblížiť k hodnote z, nesmie ju však prekročiť.

V jazyku ILP sa toto zapíše úplne triviálne. Predpokladajme napríklad, že máme batoh s nosnosťou 174974 a dvadsať veci s hmotnosťami 217, 236, 2039, 7620, 9796, 12047, 12587, 14046, 16848, 17822, 18760, 20945, 21645, 22100, 23352, 25367, 26364, 26513, 26904 a 32470. Vyrobíme si teda nasledovný ILP:[5]

```
max: zobral;
zobral <= 174974;
zobral = 217 * y0 + 236 * y1 + ... + 32470 * y19;
y0 <= 1;
...
y19 <= 1;
int zobral, y0, y1, ..., y19;
```

Spustíme ILP solver (napríklad náš starý známy `lp_solve`) a už len čítame riešenie: optimálna hodnota `zobral` je `174974`, batoh teda vieme naplniť na chlp presne. A hodnoty premenných `y0` až `y19` nám hovoria, ktoré veci zobrať (1) a ktoré nie (0).

Podúloha A: batoh s cenami (4 body)

V súboroch `batoh01.in` až `batoh04.in` máte štyri vstupy pre problém batoha s cenami. Vstupný súbor má v prvom riadku počet vecí n, v druhom riadku maximálnu nosnosť batoha z a následne v každom z n ďalších riadkov popis jednej veci: jej váhu w_i a jej cenu c_i. Ku každému vstupu zostrojte (ručne alebo programom) ILP, ktorý zistí, ktoré veci nabrať do batoha tak, aby sme neprekročili jeho nosnosť a dosiahli najväčšiu možnú celkovú cenu. Pomocou nejakého ILP solvera následne vaše ILP vyriešte.

Podúloha B: scheduling (6 bodov)

Máme procesor so 4 jadrami a n procesov. Ku každému procesu vieme čas potrebný na jeho vykonanie. Každý proces sa musí celý vykonať na jednom z jadier. Našou úlohou je procesy rozdeliť na jadrá tak, aby sme

[4] Technický detail: len hodnoty *premenných* sú obmedzené na nezáporné čísla, hodnoty výrazov, ktoré pomocou nich počítame, pokojne môžu byť záporné.

[5] Syntaktická poznámka: `lp_solve` vyžaduje v programe najskôr cieľ (to čo minimalizujeme/maximalizujeme), potom všetky nerovnosti a až nakoniec deklaráciu toho, ktoré premenné sú celočíselné.

boli čo najskôr hotoví – teda aby uplynulo čo najmenej času od okamihu kedy začneme po okamih kedy dobehne posledný proces.

V súboroch `proc01.in` až `proc03.in` máte tri vstupy pre tento problém: vždy najskôr n a potom n celých čísel: jednotlivé časy. Nájdite pomocou ILP solvera ich optimálne riešenia. (Najskôr si treba vhodne zvoliť, čo budú naše premenné. A potom prísť na to, ako sformulovať a zapísať to, že chceme vlastne minimalizovať najväčší zo štyroch súčtov časov.)

Podúloha C: obchodný cestujúci (6 bodov)

Na záver sa pozrieme na problém obchodného cestujúceho. V súboroch `tsp01.in` až `tsp05.in` máte päť vstupov pre tento problém. V prvom riadku vstupu je počet miest v krajine n. Mestá sú očíslované 0 až $n-1$. Nasleduje $n-1$ riadkov, v i-tom z nich je $n-i$ celých čísel: vzdialenosti z mesta $i-1$ postupne do miest i až $n-1$. Vašou úlohou je nájsť okružnú cestu, ktorá práve raz navštívi každé mesto a má najmenšiu možnú celkovú dĺžku.

Pomôcka: Existujú síce spôsoby, ako rozumne malým ILP zapísať všetky potrebné podmienky, sú však veľmi komplikované a LP solveru sa s nimi ťažko robí. Odporúčame vám radšej ísť iným, omnoho praktickejším smerom. Predstavte si jednoduchý ILP, ktorý obsahuje len podmienky „z každého mesta vyberieme dve cesty“. To zjavne nie je úplne to, čo chceme – totiž optimálnym riešením takéhoto ILP môže byť nielen jedna okružná cesta cez všetky mestá, ale aj sada kratších okružných ciest, z ktorých každá ide cez menej miest. Čo spraviť v tom druhom prípade? Predsa pridať vhodnú podmienku, ktorá dotyčnú možnosť (a zároveň aj mnohé iné jej podobné) zakáže – napr. podmienku typu „niekedy musím prejsť z tohto chotára do hentoho“. A potom to skúsiť znova, až kým nedostaneme situáciu, kedy najlepším riešením je už okružná cesta ktorú hľadáme.

Inštrukcie k odovzdávaniu riešení

Vaše riešenia budeme bodovať ručne. Ku každej podúlohe by ste mali uviesť stručný slovný popis riešenia, pomocné programy a výstup lineárneho programu. Ideálne z toho všetkého vyrobte a odovzdajte jedno veľké PDF.

6. Ostrieľaný hráč

kat. O; 12 b za popis, 8 b za program

Nikto nevie ako a prečo, ale Žabe sa podarilo nájsť v Dote[6] skrytý item, ktorý doteraz nikto nikdy nevidel. Item sa volá Kasparov Knights, je pomerne drahý a dá sa použiť len raz za hru, no pri správnom použití vie byť veľmi užitočný.

Item funguje nasledovne: Celú mapu si predstavíme ako obrovskú šachovnicu, na niektorých políčkach sú nepriateľské jednotky. Následne hráč na šachovnicu rozmiestni ľubovoľný počet šachových koní. (Môže aj na políčka s nepriateľskými jednotkami.) Za každého umiestneného koňa spotrebuje item určité množstvo many, ktoré závisí od jeho pozície. Nakoniec hráč aktivuje item a všetky jednotky, ktoré sú ohrozené nejakým z koní, sú zničené.

Pomôžte Žabe nájsť najlepšie rozmiestnenie koní, aby sa mohol sústrediť na hru.

Úloha

Na šachovnici rozmerov $r \times s$ máme rozmiestnené nepriateľské jednotky. Našou úlohou je umiestniť na šachovnicu niekoľko koní tak, aby boli **všetky** nepriateľské jednotky ohrozené aspoň jedným koňom a zároveň **aby množstvo spotrebovanej many bolo čo najmenšie**.

Každý kôň ohrozuje 8 políčok okolo seba (alebo menej, ak by išlo o políčka mimo šachovnice), pre ktoré platí, že sa od pozície koňa líšia o 1 v jednej súradnici a o 2 v druhej súradnici (čiže ako obyčajný šachový kôň).

Kôň v i-tom riadku a j-tom stĺpci spotrebuje $2^{s \cdot i + j}$ many (riadky aj stĺpce číslujeme od 0). Napríklad na šachovnici rozmerov 4×6 by jednotlivé pozície mali nasledovné ceny:

	0	1	2	3	4	5
0	1	2	4	8	16	32
1	64	128	256	512	2^{10}	2^{11}
2	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
3	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}

[6] Dota 2 je Žabova obľúbená počítačová hra, ktorou zabíja svoj voľný čas.

Formát vstupu

Na prvom riadku vstupu sú dve čísla r a s ($1 \leq r, s \leq 200\,000$), udávajúce počet riadkov a stĺpcov šachovnice. Na druhom riadku je počet nepriateľských jednotiek n ($1 \leq n \leq 200\,000$).

Nasleduje n riadkov a na i-tom sú dve čísla, r_i a s_i ($0 \leq r_i < r$, $0 \leq s_i < s$) udávajúce pozíciu (riadok a stĺpec) i-tej nepriateľskej jednotky. Rôzne jednotky **môžu** byť na rovnakom políčku.

Rozumné množstvo bodov dostanete, ak budete predpokladať, že rozmery šachovnice sú do $2\,000 \times 2\,000$ resp. 50×50 resp. 5×5.

Formát výstupu

Na prvom riadku výstupu vypíšte počet rozmiestnených koní. Následne pre každého koňa vypíšte na jeden riadok dve čísla udávajúce jeho pozíciu na šachovnici. Kone vypisujte v poradí podľa ceny.

V prípade, že je viac možností, ako dosiahnuť optimálnu cenu, si vyberte ľubovoľnú z nich.

Príklad

vstup

```
4 6
3
3 1
1 2
3 3
```

výstup

```
2
0 0
1 2
```

Prvý kôň ohrozuje políčko $(1, 2)$, *druhý kôň ohrozuje políčka* $(3, 1)$ *a* $(3, 3)$. *Spolu minieme* $2^0 + 2^8 = 257$ *many.*

7. Ošarpanejší matfyz

kat. O; 12 b za popis, 8 b za program

War. War never changes.
– Fallout

Nerf guny prevrátili život KSPákov z nôh na hlavu. Ani deň neprešiel bez toho, aby si nedali poriadnu prestrelku: raz v T2, inokedy na chodbe pred akvárkami[7], priamo v akvárkach, v počítačových halách...

Venovali tomu každú voľnú chvíľu, a tak si ani nevšimli, že sa už začali blížiť termíny odovzdávania domácich úloh, projektov, bakalárskych prác a iných čas žerúcich vecí. Bola položená otázka: dobré známky, alebo nerf guny? Prehral, samozrejme, spánok.

Aj naďalej behali po celom matfyze, strieľajúc po všetkom, čo aspoň zhruba pripomínalo vedúceho Trojstenu. Boli riadne unavení, a tak občas strelili mimo, náboje triafali steny, okná, cvičiaceho predmetu Vybrané kapitoly z teoretickej informatiky a iné drevené kusy nábytku.

Matfyz však má už svoje roky, a tak po dvoch týždňoch zúrivého boja nevydržal a celá budova sa začala rúcať. Máte pocit, že to zastavilo KSPákov? Samozrejme, že nie.

Ale bezpečnosť nadovšetko! A tak si stanovili pravidlá pre pohyb po týchto zrúcaninách. Každá chodba má teraz vyznačený smer, v ktorom sa v nej má pohybovať. Navyše, každá križovatka má **najviac jednu** chodbu, po ktorej sa dá z nej odísť.

V zápale boja Zygro skúma, kto z vedúcich sa ako daleko môže dostať. Totiž vedúci, ktorý nemá kam ísť, je oveľa lepší terč. Nemá na to ale čas, lebo musí zbierať náboje. Pomôžte mu!

Úloha

Matfyz (okrem samých skríň a dverí) pozostáva z n križovatiek a nanajvýš n jednosmerných chodieb, z ktorých každá spája nejaké dve križovatky, pričom každá križovatka má **najviac jednu** chodbu smerom von. Žiadna chodba nemá tvar slučky (t.j., nemá na začiatku a konci tú istú križovatku).

Na vstupe dostanete popis toho, ako sa postupne rozpadali chodby, a taktiež veľa otázok. Najskôr môžete načítať celý vstup a až potom zodpovedať všetky otázky. Každá otázka je tvaru „kam najďalej sa v danom okamihu vedel dostať človek z danej križovatky?“ Občas sa samozrejme stane, že odpoveď na takúto otázku neexistuje – to vtedy, keď dotyčný príde na cyklus, po ktorom môže chodiť do nekonečna.

Formát vstupu

V prvom riadku je číslo n ($1 \leq n \leq 300\,000$), udávajúce počet križovatiek. Tie sú očíslované od 1 po n.

[7] akvárka sú presklené učebne v pavilóne matematiky

V druhom riadku je n čísel a_1 až a_n, kde číslo a_i hovorí, že na začiatku chodba z križovatky i viedla na križovatku a_i. Ak je a_i rovné nule, tak z tejto krížovatky nikdy neviedla von žiadna chodba.

Tretí riadok obsahuje číslo q ($1 \leq q \leq 300\,000$), udávajúce počet udalostí vo zvyšku vstupu. Každá udalosť je buď otázka alebo oznam o rozpadnutí chodby. Presnejšie, každý z nasledujúcich q riadkov je buď tvaru 1 x alebo 2 x, kde 1 x predstavuje otázku, kam najďalej sa dá dostať z križovatky x, a 2 x nám hovorí, že sa práve zrútila chodba ktorá doteraz vychádzala z križovatky x. Je garantované, že táto chodba pred zrútením naozaj existovala.

Formát výstupu

Pre každý z riadkov v tvare 1 x (a to v poradí, v akom sú uvedené na vstupe) vypíšte jeden riadok a v ňom číslo križovatky, na ktorej by sa v danej chvíli zasekol človek začínajúci na križovatke x. Ak taká križovatka neexistuje (teda ak človek začínajúci na x príde na cyklus, po ktorom môže behať do nekonečna), vypíšte namiesto toho reťazec `CYKLUS`.

Nezabudnite pri tom, že všetky zmeny sú kumulatívne – teda ak práve odpovedáte na otázku, ktorá je r-tou udalosťou na vstupe, musíte počítať s tým, že sa už rozpadli všetky chodby o ktorých sa hovorí v udalostiach 1 až $r - 1$.

Príklady

vstup

```
3
2 3 1
7
1 1
1 2
2 1
1 2
1 1
2 2
1 2
```

výstup

```
CYKLUS
CYKLUS
1
1
2
```

Podrobný popis jednotlivých udalostí:

1. *Začíname na križovatke 1, ktorá priamo leží na cykle:* $1 \to 2 \to 3 \to 1 \to \ldots$
2. *Začíname na križovatke 2, ktorá leží na tom istom cykle.*
3. *Rozpadla sa chodba z križovatky 1 (na križovatku 2).*
4. *Človek začínajúci na križovatke 2 teraz pôjde* $2 \to 3 \to 1$ *a tam sa zasekne.*
5. *Človek začínajúci na križovatke 1 teraz nemá kam ísť.*
6. *Rozpadla sa chodba z križovatky 2 (na križovatku 3). Zostáva jediná chodba: z 3 do 1.*
7. *Človek začínajúci na križovatke 2 teraz už tiež nemá kam ísť.*

vstup

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

výstup

```
1
CYKLUS
4
3
```

Všimnite si, že na druhú otázku je odpoveď `CYKLUS`*. Križovatka 2, na ktorej začíname, na dotyčnom cykle neleží – na ten prídeme až po prejdení prvou chodbou.*

8. Organizmy

kat. O; 14 b za popis, 8 b za program

V bioinformatike sa často pracuje s fylogenetickými stromami, ktoré znázorňujú vzťahy rôznych druhov organizmov a ich vývoj v čase. Na posudzovanie podobnosti organizmov sa často využíva DNA alebo nejaké charakteristické znaky.

My sa pozrieme na tieto fylogenetické stromy bližšie. Predstavíme si ich ako zakorenené informatické stromy. Listy stromu potom predstavujú jednotlivé živočíšne druhy, napríklad mačka domáca alebo krokodíl nílsky môžu byť listy takéhoto stromu. Vnútorné vrcholy predstavujú skupiny organizmov, ktoré majú nejaké spoločné znaky.

To ako fylogenetický strom vyzerá, môže závisieť od toho, ktoré spoločné znaky práve skúmame.

Napríklad v jednom strome môže byť v spoločnej skupine delfín, veľryba a kosatka, zatiaľ čo netopier bude v úplne inej skupine. Pokiaľ ich ale začneme skúmať z pohľadu orientácie v priestore, tak delfín a netopier skončia v rovnakej skupine (oba druhy majú podobne vyvinutý sonar), zatiaľ čo veľryba a kosatka budú inde.

Viac sa o týchto stromoch dočítate napríklad na Wikipédii[8]. Nájdete tam aj obrázky takýchto stromov. K úspešnému vyriešeniu tejto úlohy však nebudú potrebné.

Úloha

V tejto úlohe máme dva fylogenetické stromy a chceme ich porovnať. To znamená, že chceme zistiť, ktoré skupiny organizmov (reprezentované vrcholom stromu) sa nachádzajú v oboch stromoch.

Na vstupe dostanete dva stromy skladajúce sa z n vrcholov očíslovaných od 0 po $n-1$. Oba stromy sú zakorenené vo vrchole 0. Čísla listov predstavujú jednotlivé živočíšne druhy. Všimnite si, že oba stromy môžu obsahovať rôznu množinu čísiel, ktoré tvoria listy.

Každému vrcholu stromu priradíme skupinu organizmov, ktorá pozostáva zo všetkých organizmov (listov), ktoré sa nachádzajú v podstrome pod ním. Zistite, ktoré skupiny organizmov sa nachádzajú v jednom aj v druhom strome. To znamená, že chcete nájsť také vrcholy v v prvom strome, že existuje vrchol u v druhom strome, že vrcholy v a u majú vo svojich podstromoch rovnakú množinu listov.

Poznámka: Plný počet bodov môže dostať iba riešenie, ktoré sa nespolieha na náhodu. Riešenie, ktoré sa spolieha na náhodu, môže ale stále dostať dosť bodov.

Formát vstupu

Na prvom riadku vstupu je číslo $1 \leq n \leq 2 \cdot 10^5$ – počet vrcholov stromov. Na druhom a treťom riadku sú popisy našich fylogenetických stromov. Strom je popísaný postupnosťou čísel $a_1, a_2, \ldots a_{n-1}$, kde a_i predstavuje otca vrcholu i.

Formát výstupu

Vypíšte utriedený zoznam vrcholov z prvého stromu, pre ktoré existuje vrchol v druhom strome, ktorý obsahuje rovnakú podmnožinu listov.

Príklad

vstup

```
7
0 1 1 1 4 4
0 4 4 0 4 4
```

výstup

```
0
1
2
3
5
6
```

V prvom strome do skupiny vrchola 1 *patria druhy číslo* 2, 3, 5 a 6. *V druhom strome sa nachádza skupina vrchola* 4, *do ktorej patria tiež vrcholy* 2, 3, 5 a 6. *Pre vrchol* 4 *prvého stromu však nevieme nájsť zodpovedajúci vrchol druhého stromu pod ktorým sú len listy* 5 a 6.

vstup

```
6
0 0 0 3 3
0 1 1 2 2
```

výstup

```
3
4
5
```

[8] https://en.wikipedia.org/wiki/Phylogenetic_tree

Zadania kategórie T

Nájdete ich onedlho na našej stránke `http://www.ksp.sk/wiki/Zadania/Zadania`. Nezabudnite sa na ne pozrieť, pribudne ďalších **päť** zaujímavých úloh rôznych obtiažností.