56. ročník matematické olympiády na středních školách

Kategorie P

In: Karel Horák (editor); Martin Mareš (editor); Peter Novotný (editor); Jaromír Šimša (editor); Jaroslav Švrček (editor); Pavel Töpfer (editor): 56. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané ve školním roce 2006/2007. 48. mezinárodní matematická olympiáda. 19. mezinárodní olympiáda v informatice. (Czech). Praha: Jednota českých matematiků a fyziků, 2008. pp. 89–108.

Persistent URL: http://dml.cz/dmlcz/405131

P – I – 1

Pizza kolem

Marco se rozhodl, ze zužitkuje své kulinářské i cyklistické dovednosti a založí si firmu pro výrobu a rozvoz pizzy. Firmu plánuje provozovat tak, že vždy nejdříve bude shromažďovat objednávky, a když jich bude dostatek, tak pizzy upeče, sedne na kolo a rozveze je zákazníkům. Protože Marco je lepší cyklista než kuchař, tak zatím ve své nabídce plánuje pouze jeden druh pizzy. Aby se ale odlišil od konkurence, tak přijímá objednávky i na šestinové části pizzy. Lze si u něj objednat například 1/6, 4/6 nebo 15/6 pizzy. Navíc, jako speciální službu zákazníkům, chce Marco pizzy pro každého zákazníka dodat co nejméně rozřezané (aby si zákazník sám mohl rozhodnout, jak si dále pizzu rozdělí). Proto například 4/6 pizzy chce dodat jako jeden kus příslušné velikosti a 15/6 chce dodat jako 2 celé pizzy a k nim jednu polovinu pizzy (Marco chce také dodat co nejvíce pizz vcelku, takže uspokojení této objednávky třemi kusy velikosti 5/6 nepřipadá v úvahu).

Až když Marco všude rozdal letáky propagující jeho novou firmu, tak si uvědomil, že díky jeho speciální službě zákazníkům není jednoduché zkombinovat objednávky tak, aby mu moc kusů pizzy nezbylo. Obrátil se proto na vás, abyste mu napsali program, který by mu s problémem pomohl. Pro začátek by mu stačil program, který na vstupu dostane objednávky a na výstup vypíše, kolik pizz má Marco napéct.

Formát vstupu: Na prvním řádku vstupního souboru `pizza.in` se nachází celé číslo N, $1 \leqq N \leqq 10\,000$ — počet objednávek. Ve vstupním souboru pak následuje N řádků. Každý řádek popisuje jednu objednávku a obsahuje jedno celé číslo c, $1 \leqq c \leqq 100$, které je počet objednaných šestin pizzy.

Formát výstupu: Výstupní soubor `pizza.out` bude obsahovat jedno celé číslo p, které značí nejmenší možný počet pizz, které je třeba upéct,

aby šlo splnit všechny objednávky a byla dodržena speciální služba zákazníkům.

Příklad 1:

pizza.in	pizza.out
3	2
2	(z jedné pizzy lze například uříznout dva kusy
2	o velikosti 2/6 a z druhé pizzy se uřízne kus
3	velký 3/6)

Příklad 2:

pizza.in	pizza.out
3	3
4	(kvůli požadavku na dodání co nejméně rozřeza-
5	ných kousků pizzy je třeba pro každou objed-
3	návku upéct celou pizzu)

P – I – 2

Zasypané město

Archeolog Bedřich Hrozný zkoumá nově nalezené zasypané město v poušti. Jako první krok se rozhodl, že pomocí sonaru určí, kolik místností měly všechny domy ve městě dohromady.

Kus pouště, kde město leželo, si Bedřich pokryl čtvercovou sítí o rozměrech $M \times N$. Se sonarem postupně projel všechny řádky takto vytvořené čtvercové sítě a svá měření si zaznamenal. Pro jednoduchost předpokládal, že pod každým polem této sítě se nachází buď kamení, nebo písek. Na základě získaných dat by rád určil, kolik místností (souvislých oblastí písku) v zasypaném městě bylo.

Soutěžní úloha. Na vstupu je dán popis zasypaného města, které si představujeme jako čtvercovou síť o rozměrech $M \times N$. Políčka čtvercové sítě jsou popsána po řádcích od horního ke spodnímu a na jednotlivých řádcích postupně zleva doprava. Bedřich si svá měření zapsal pomocí dvojic čísel, kde první číslo znamená počet políček s pískem a druhé počet políček s kamením. Data získaná ze sonaru tedy tvoří K dvojic nezáporných čísel (p, q). Dvojice (p, q) reprezentuje, že z následujících $p + q$ políček je prvních p políček tvořeno jen pískem a zbylých q políček je tvořeno kamením. Každý úsek takovýchto $p + q$ políček leží pouze v jednom řádku čtvercové sítě, tj. žádná dvojice (p, q) neodpovídá úseku políček na dvou či více řádcích.

Vaším úkolem je spočítat počet místností v zasypaném městě. Místností se rozumí souvislá oblast písku, která už v žádném směru nejde zvětšit. Dvě políčka čtvercové sítě považujeme za sousední, pokud mají společnou hranu, nikoliv jen vrchol.

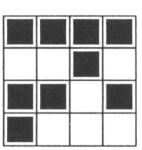
Formát vstupu: Na prvním řádku souboru `mesto.in` se nacházejí tři nezáporná čísla M, N a K — počet řádků a sloupců čtvercové sítě města a počet dvojic, které popisují její obsah. Je známo, že M i N jsou menší než 50 000 a že K je menší než 1 000 000 000. Každý z dalších K řádků obsahuje dvě *nezáporná* čísla p a q, kde p je počet políček zasypaných pískem a q je počet políček, pod kterými jsou kameny. Políčka jsou popsána po řádcích od horního řádku sítě, na jednotlivých řádcích zleva doprava. Navíc žádná dvojice (p, q) nepopisuje políčka obsažená na více řádcích.

Pokud se Vám nepodaří vyřešit úlohy s výše popsanými omezeními na M, N a K, předpokládejte, že každé M a N jsou nejvýše 500 a K je nejvýše 100 000.

Formát výstupu: Jediný řádek souboru `mesto.out` by mělo tvořit jediné nezáporné číslo — počet místností v zasypaném městě. Všimněte si, že pokud pod všemi políčky je jen kamení, bude toto číslo rovno nule.

Příklad:

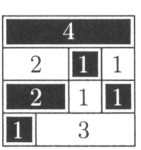
`mesto.in`	`mesto.out`
4 4 7	3
0 4	
2 1	
1 0	
0 2	
1 1	
0 1	
3 0	



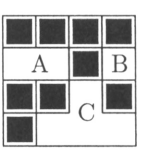
Zasypané město

■ kamení

□ písek



Zasypané město, jak ho vidí sonar



Místnosti zasypaného města označené A, B, C.

P – I – 3

Okružní jízda

Ve Stínové Praze je komplikovaný dopravní systém. Tvoří ho křižovatky, navzájem propojené ulicemi, ale na rozdíl od reálné Prahy může do jedné křižovatky vést libovolný počet ulic. Zajisté chápete, že řízení dopravy ve Stínové Praze je velmi složité a často dochází k nehodám. Radní ve Stínové Praze se rozhodli zlepšit dopravní situaci. Nejprve ze všech ulic udělali jednosměrky, a to tak, že do každé křižovatky vchází alespoň jedna ulice a z každé křižovatky vychází alespoň jedna ulice. Pak navíc na každé křižovatce zakázali jednu možnost odbočení (tedy před touto změnou se křižovatka, do níž vede k ulic a z níž vede l ulic, dala projet kl způsoby; nyní je možné ji projet jen $kl - 1$ způsoby). Stínoví Pražané si ale začali stěžovat, že se nedokážou dostat z domu do práce či naopak. Aby podobná tvrzení vyvrátili, rozhodli se radní dokázat, že se dá z každé ulice dostat do každé jiné. Dělat to pro každou dvojici ulic zvlášť by bylo pracné, proto chtějí nalézt „okružní jízdu" — tj. cyklickou posloupnost ulic takovou, že všechna odbočení v ní jsou povolená, nikde se v ní nejede v protisměru a každá ulice se v ní vyskytuje právě jednou. Všimněte si, že taková posloupnost nemusí existovat: např. pokud existuje křižovatka, do které vede méně ulic, než kolik z ní vychází.

Formát vstupu: Na prvním řádku vstupního souboru `okruh.in` jsou dvě přirozená čísla n a m, udávající počet křižovatek a počet ulic ve Stínové Praze. Křižovatky jsou očíslovány přirozenými čísly od 1 do n. Následujících m řádků popisuje ulice. Na každém z nich je dvojice čísel u a v ($1 \leqq u, v \leqq n$), znamenající, že z křižovatky u vede jednosměrná ulice do křižovatky v. Mezi dvěma křižovatkami může vést v každém směru nejvýše jedna ulice. Dále následuje n řádků, i-tý z nich popisuje, jaké odbočení je zakázáno na i-té křižovatce. Jsou-li na i-tém řádku čísla u a v ($1 \leqq u, v \leqq n$), pak jedeme-li po ulici z u do i, nesmíme odbočit do ulice z i do v.

Formát výstupu: Do výstupního souboru `okruh.out` vypište posloupnost čísel $v_1, v_2, \ldots, v_m$ ($1 \leqq v_i \leqq n$ pro každé i) takovou, že:

- z v_i do v_{i+1} (pro $1 \leqq i \leqq m$) vede ulice,
- jedeme-li ulicí z v_i do v_{i+1}, je povoleno odbočit do ulice z v_{i+1} do v_{i+2} (pro $1 \leqq i \leqq m$), a
- každá ulice je použita, tj. existuje-li ulice z u do v, pak existuje i tak, že $v_i = u$ a $v_{i+1} = v$.

Indexy počítáme cyklicky, tj. $v_{m+1} = v_1$ a $v_{m+2} = v_2$. Pokud existuje

více takových posloupností, vypište libovolnou z nich. Pokud taková posloupnost neexistuje, vypište řetězec „`Okruzni jizda neexistuje.`“.

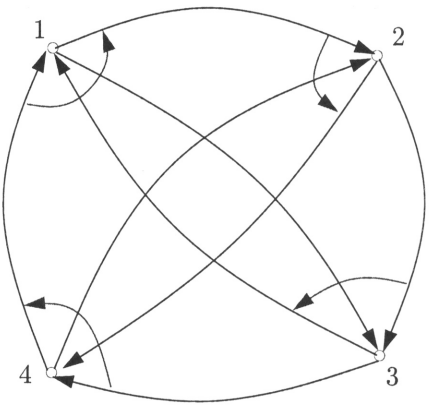
Příklad 1:

`okruh.in`

```
4 8
1 2
2 3
3 1
3 4
1 3
4 2
2 4
4 1
4 2
1 4
2 1
3 1
```

`okruh.out` (jeden ze správných výstupů)

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



Příklad 2:

`okruh.in`

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

`okruh.out`

```
Okruzni jizda neexistuje.
```

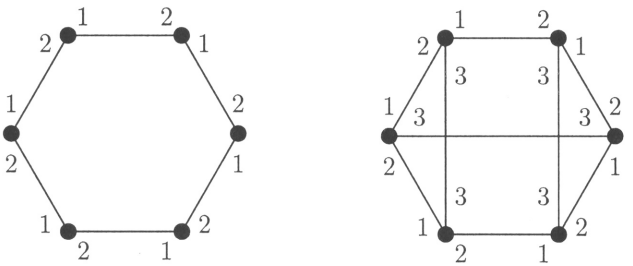
P – I – 4

Grafomat

Grafem nazveme libovolnou konečnou množinu V *vrcholů* grafu spolu s množinou E *hran*, což jsou neuspořádané dvojice vrcholů. Žádné dva vrcholy nejsou spojeny více hranami, žádná hrana nespojuje vrchol se sebou samým.

K-graf budeme říkat takovému grafu, ve kterém s každým vrcholem sousedí právě K hran a konce těchto hran jsou očíslovány přirozenými čísly od 1 do K. Oba konce jedné hrany přitom mohou být očíslovány růz-

ně. Pokud budeme hovořit o hranách vycházejících z nějakého vrcholu v, budeme zmiňovat *místní* čísla hran (to jsou čísla konce, kterým je v) a čísla *protější* (to jsou ta zbývající). Pro každý vrchol jsou místní čísla všech jeho hran navzájem různá. Následující obrázek ukazuje příklad 2-grafu a 3-grafu:



Ohodnocením grafu nazveme přiřazení prvků nějaké konečné množiny vrcholům grafu — tedy například rozdělení vrcholů na černé a bílé nebo označení vrcholů čísly od 1 do 5.

Grafomat je zařízení pro automatické řešení grafových úloh. Jeho vstupem je libovolný K-graf G spolu s jeho ohodnocením; výstupem je nějaké další ohodnocení téhož grafu. Samotný výpočet je vykonáván *automaty* umístěnými v jednotlivých vrcholech grafu. Každý automat má svou paměť a řídí se programem. Programy všech automatů jsou identické, zatímco paměť má každý automat svoji a mimo to ještě může nahlížet do pamětí svých grafových sousedů.

Paměť automatu je tvořena konečným množstvím proměnných, které si můžeme představit jako pascalské proměnné typu interval. Obsahují tedy přirozená čísla v nějakém pevném rozsahu, který nezávisí na velikosti vstupu. Mimo to je také možné používat pole intervalových proměnných, jejichž indexy jsou opět z pevných intervalů. Žádné jiné typy proměnných (neomezeně velká čísla, ukazatele, ...) použít nelze.

Zvláštní roli hrají proměnné x a y. Proměnná x na počátku výpočtu obsahuje vstupní ohodnocení toho vrcholu grafu, ke kterému patří, hodnota proměnné y na konci výpočtu určí výstupní ohodnocení vrcholu. Všechny proměnné s výjimkou proměnné x mají svou počáteční hodnotu pevně určenu. Deklarace proměnných vypadá například takto:

```
var x: 1..5;       { číslo od 1 do 5, na počátku vstup }
    y: 1..5 = 3;   { číslo od 1 do 5, na počátku 3,
                     na konci výstup }
    z: array [1..2] of 3..4 = (3, 4);  { pole dvou čísel }
```

Řídící program automatu si můžeme představit jako pascalský program, v němž si zakážeme používat rekurzi a který bude manipulovat pouze s proměnnými v paměti automatu a případně i automatů sousedních. Na své vlastní proměnné se automat odkazuje jejich jmény, jako by to byly obyčejné pascalské globální proměnné, na proměnné sousedů pak konstrukcí $S[i].p$. Zde i je celočíselný výraz s hodnotou $1 \ldots K$, jenž značí, o kolikátého souseda se jedná, tedy místní číslo hrany, kterou je soused připojen; p je jméno libovolné proměnné. Proměnné sousedů je možné pouze číst.

Aby mohl program dávat do souvislostí své hrany s hranami svých sousedů, má k dispozici ještě proměnné $P[1], \ldots, P[K]$, které jsou pevně nastaveny tak, že $P[i]$ obsahuje protější číslo hrany s místním číslem i. Výraz $S[i].S[P[i]].x$ je tedy totéž jako samotné x. (Pozor, zatímco druhé S je odkaz na proměnnou patřící sousedovi, proměnná P v indexu je opět místní.)

Výpočet grafomatu probíhá v taktech, a to následovně: V nultém taktu se proměnné všech automatů nastaví na počáteční hodnoty a proměnné x na vstupní ohodnocení jednotlivých vrcholů. V každém dalším taktu se pak vždy jednou spustí program každého automatu, přičemž proměnné svých sousedů vidí program ve stavu, v jakém byly na začátku taktu. Ačkoliv tedy jednotlivé automaty běží současně, nemůže se stát, že by jeden četl z proměnné, do které právě druhý zapisuje.

Výpočet pokračuje tak dlouho, dokud v nějakém taktu všechny automaty neprovedou příkaz `stop`. Pak se výpočet zastaví a z proměnných y grafomat přečte výstupní ohodnocení grafu. Pokud příkaz `stop` provedou jen některé automaty, výpočet pokračuje, a to i na těchto automatech. Struktura grafu, jakož i obsah proměnných P zůstává po celou dobu výpočtu konstantní.

Za *časovou složitost* výpočtu budeme považovat počet taktů, které uběhnou do zastavení. Nijak tedy nezávisí na rychlosti programů jednotlivých automatů. Podobně jako u časové složitosti klasických algoritmů nebudeme hledět na multiplikativní konstanty a bude nás zajímat pouze asymptotické chování složitosti, tedy zda je lineární, kvadratická, atd. Případy, kdy výpočet neskončí, nebudeme připouštět, pro úplnost ale dodejme, že tehdy se nutně musí hodnoty proměnných periodicky opakovat.

Příklad 1: Je dán 3-graf a v něm vyznačen jeden vrchol v, a to tak, že jeho proměnná x bude inicializována jedničkou, zatímco všem ostatním

vrcholům nulou. Napište program pro grafomat, který označí všechny vrcholy z vrcholu v dosažitelné po hranách, a to tak, že jejich proměnná y bude na konci výpočtu rovna jedné, zatímco u nedosažitelných vrcholů bude nulová.

Řešení: Inspirujeme se prohledáváním grafu do šířky. V každém taktu se každý vrchol podívá, zda některý z jeho sousedů je již označen a pokud ano, také se sám označí. Pokud se označení nezmění, vrchol voláním `stop` souhlasí se zastavením. Průběh výpočtu tedy bude vypadat tak, že v i-tém taktu budou označeny ty vrcholy, jejichž vzdálenost od v je menší nebo rovna i. Výpočet zastaví, jakmile se hodnoty proměnných přestanou měnit, tj. po nejvýše N taktech. Proto je časová složitost našeho programu lineární v počtu vrcholů (na rozdíl od klasického průchodu do šířky nezávisí na počtu hran).

Program vypadá následovně:

```
var x: 0..1;                { byl vrchol označen ve vstupu? }
    y: 0..1 = 0;            { je označen teď? }
    prev: 0..1 = 0;         { předchozí stav }
    i: 1..3;
begin
  prev := y;      { zapamatujeme si, jestli už byl označen }
  if x=1 then y := 1;       { přeneseme označení ze vstupu }
  for i := 1 to 3 do        { podívejme se na všechny sousedy }
     if S[i].y <> 0 then { je-li i-tý soused označen, }
        y := 1;             { označ i sebe sama }
  if y = prev then stop; { pokud se nic nemění,
                           můžeme končit }
end.
```

Příklad 2: Mějme 2-graf složený z jediného cyklu sudé délky (tj. z vrcholů očíslovaných $0 \ldots N-1$, přičemž vrchol i je spojen hranou označenou 1 s vrcholem $(i+1) \bmod N$ a hranou označenou 2 s vrcholem $(i-1) \bmod N$; příklad takového grafu pro $N = 6$ najdete na obrázku na začátku tohoto textu). V tomto grafu je vyznačen jeden vrchol v. Napište program pro grafomat, který označí vrchol protilehlý k v, tedy vrchol s číslem $(v + N/2) \bmod N$.

Řešení: Vyšleme „signál" putující z vrcholu v ve směru jedničkových hran rychlostí 1 vrchol za takt a druhý signál putující stejnou rychlostí opačným směrem. Jakmile nějaký vrchol zjistí, že do něj přišly oba signály, označí se a signály již dál nepředává.

```
var x: 0..1;          { vstupní značka u vrcholu }

    y: 0..1 = 0;      { výstupní značka }
    l, r: 0..1 = 0; { už tímto vrcholem prošel signál
                      doleva a doprava? }
begin
   if x=1 then        { začínáme posílat }
      begin x := 0; l := 1; r := 1; end
   else if (S[2].l=1) and (S[1].r=1) then
                      { signály se v tomto vrcholu potkaly }
      begin y := 1; stop; end
   else if (S[2].l=1) and (l=0) then l := 1
                      { předáme signál doleva }
   else if (S[1].r=1) and (r=0) then r := 1
                      { předáme signál doprava }
   else stop;         { nic se neděje => můžeme končit }
end.
```

Soutěžní úloha. Napište program pro grafomat, který v zadaném 3-grafu s vyznačenými dvěma vrcholy nalezne nejkratší cestu vedoucí mezi nimi a vyznačí vrcholy ležící na této cestě. Můžete předpokládat, že cesta vždy existuje. Pokud je nejkratších cest více, vyberte si libovolnou z nich.

Vstup bude tvořen proměnnou x, která bude v prvním ze zadaných vrcholů rovna jedné, v druhém dvěma a ve všech ostatních vrcholech nulová.

Výstupem programu bude proměnná y ve vrcholech nejkratší cesty jedničková, jinde nulová.

Pokuste se nalézt takový program, jehož časová složitost bude záviset pouze na délce sestrojené cesty a ne na velikosti celého grafu.

P – II – 1

Zasypané město

Archeolog Bedřich Hrozný již s vaší pomocí zmapoval zasypané město, které objevil. Nyní by rád započal s vykopávacími pracemi. Požádal o pomoc místní univerzitu, která mu dala k dispozici N studentů prvního a N studentů druhého ročníku. Každý ze studentů, které má k dispozici, studuje jednu z následující tří specializací: starověká historie, středověká historie nebo archeologie. Bedřich Hrozný by rád rozdělil studenty do N dvojic tak, aby v každé dvojici byl jeden student prvního a jeden student

druhého ročníku. Navíc chce, aby studenti v každé dvojici měli různé specializace, a tak se jejich znalosti vzájemně doplňovaly.

Pokuste se najít řešení, jehož časová složitost je co nejmenší, pokud odhlédnete od času potřebného na načtení vstupu.

Formát vstupu: Vstupní soubor se jmenuje `mesto.in`. Jeho první řádek obsahuje číslo N, které udává počet studentů prvního (a druhého) ročníku. Každý ze zbývajících $2N$ řádků obsahuje jedno číslo a řetězec `starovek`, `stredovek` nebo `archeologie`, které udávají ročník a specializaci jednotlivých studentů. Číslo a řetězec jsou vždy odděleny jednou mezerou.

Formát výstupu: Výstupní soubor se jmenuje `mesto.out`. Pokud studenty nelze rozdělit do dvojic dle požadavků Bedřicha Hrozného, výstupní soubor obsahuje jediný řádek s textem „`Studenty nelze rozdelit do dvojic.`“. V opačném případě soubor obsahuje N řádků, z nichž každý obsahuje dva řetězce, které jsou `starovek`, `stredovek` a `archeologie`. Tyto řetězce jsou odděleny jednou mezerou a udávají specializace studentů prvního a druhého ročníku (v tomto pořadí) v jednotlivých dvojicích.

Příklad:

`mesto.in`	`mesto.out`
`3`	`starovek stredovek`
`1 starovek`	`starovek archeologie`
`1 starovek`	`stredovek starovek`
`2 archeologie`	
`1 stredovek`	
`2 stredovek`	
`2 starovek`	

P – II – 2

Okružní jízda

Z domácího kola si zajisté vzpomínáte na Stínovou Prahu a její problémy s dopravou. Přesto si ale její popis připomeneme. Stínovou Prahu tvoří křižovatky, navzájem propojené ulicemi. Na rozdíl od reálné Prahy může do jedné křižovatky vést libovolný počet ulic větší než tři. Na začátku letošního ročníku se radní rozhodli zvýšit bezpečnost dopravy následujícím způsobem: Nejprve ze všech ulic udělali jednosměrky, a to tak, že do každé křižovatky vchází stejný počet ulic, jaký z ní vychází. Pak

navíc na každé křižovatce zakázali jednu možnost odbočení (tedy před touto změnou se křižovatka, do níž a z níž vedlo k ulic, dala projet k^2 způsoby; po změně to bylo možné jen $k^2 - 1$ způsoby). Přes odpor Stínových Pražanů se tuto vyhlášku podařilo prosadit a dnes dorazily ze statistického úřadu první výsledky: počet dopravních nehod se zdvojnásobil.

Na krizové poradě se radní rozhodli pro nový pokus: zruší se jednosměrky (každou ulici tedy lze projet oběma směry), zato se na každé křižovatce zakáží tři možnosti odbočení. Zákaz je také „obousměrný", tj. je-li zakázáno na dané křižovatce odbočit z a-té ulice do b-té ulice, je také zakázáno odbočit z b-té ulice do a-té ulice. Křižovatku t ulic lze tedy projet $t(t-1) - 6$ způsoby. Vzhledem k historii Stínové Prahy je t vždy sudé a $t \geqq 4$.

Aby se radní tentokrát vyhnuli fámám o autech duchů, donekonečna jezdících v ulicích Stínové Prahy bez možnosti dosáhnout cíle, požádali vás opět o nalezení „okružní jízdy" — cyklické posloupnosti ulic takové, že všechna odbočení v ní jsou povolená a každá ulice se v ní vyskytuje právě jednou.

Formát vstupu: Na prvním řádku vstupního souboru `okruh.in` jsou dvě přirozená čísla n a m, která udávají počet křižovatek a počet ulic ve Stínové Praze. Křižovatky jsou očíslovány přirozenými čísly od 1 do n. Následujících m řádků popisuje ulice. Na každém z nich je dvojice čísel u a v ($1 \leqq u, v \leqq n$), znamenající, že mezi křižovatkami u a v vede ulice. Mezi dvěma křižovatkami může vést nejvýše jedna ulice. Dále následuje n řádků, i-tý z nich popisuje, jaká odbočení jsou zakázána na i-té křižovatce a obsahuje šest přirozených čísel. Pokud jsou na něm čísla u_1, v_1, u_2, v_2, u_3 a v_3 ($1 \leqq u_j, v_j \leqq n$), říkají, že jedeme-li po ulici z u_j-té křižovatky, $1 \leqq j \leqq 3$, na i-tou křižovatku, nesmíme odbočit do ulice vedoucí k v_j-té křižovatce, a naopak, přijíždíme-li z v_j-té křižovatky, nesmíme pokračovat směrem k u_j-té.

Formát výstupu: Do výstupního souboru `okruh.out` vypište posloupnost čísel $v_1, v_2, \ldots, v_m$ ($1 \leqq v_i \leqq n$ pro každé i) takovou, že:

- z v_i do v_{i+1} (pro $1 \leqq i \leqq m$) vede ulice,
- jedeme-li ulicí mezi v_i-tou a v_{i+1}-tou křižovatkou, je povoleno odbočit do ulice mezi v_{i+1}-tou a v_{i+2}-tou křižovatkou, $1 \leqq i \leqq m$, a
- každá ulice je projeta právě jednou, tj. existuje-li ulice mezi u-tou a v-tou křižovatkou, pak existuje i takové, že $v_i = u$ a $v_{i+1} = v$ nebo $v_i = v$ a $v_{i+1} = u$.

Indexy počítáme cyklicky, tj. $v_{m+1} = v_1$ a $v_{m+2} = v_2$. Pokud existuje

více takových posloupností, vypište libovolnou z nich. Pokud taková posloupnost neexistuje, vypište řetězec „`Okruzni jizda neexistuje.`“.

Příklad:

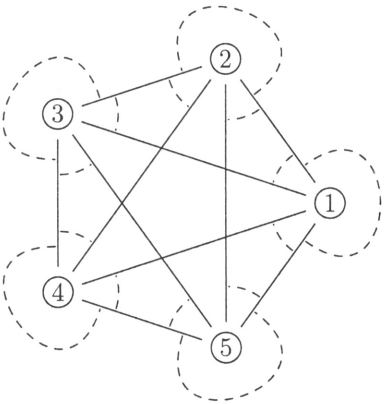
`okruh.in`

```
5 10
1 2
2 3
3 4
4 5
5 1
1 3
2 4
3 5
4 1
5 2
5 2 2 3 4 5
1 3 1 5 3 4
2 4 2 1 4 5
3 5 2 3 1 5
4 1 1 2 3 4
```

`okruh.out`

```
1 2 4 5 2 3 5 1 3 4
```

(jeden ze správných výstupů)



Obrázek města z příkladu.
Čárkované čáry označují zakázaná odbočení.

P – II – 3

Pizza kolem

Marcova firma Pizza kolem slaví díky vaší pomoci v minulém kole olympiády velký úspěch. Marco se proto rozhodl firmu rozšířit. Plánuje mít několik poboček na hranici města a obrátil se na vás, abyste mu pomohli rozhodnout, kde by bylo nejlepší nové pobočky otevřít.

Pro jednoduchost si budeme hranici města představovat jako kružnici, kterou rozdělíme na N stejně dlouhých úseků. Marco ví, kolik jeho zákazníků v každém z těchto úseků žije. Mimo to je nutné, aby vždy celý úsek byl přiřazen téže pizzerii Marcovy firmy a aby úseky přiřazené jedné pizzerii následovaly po sobě. Jedna pizzerie dokáže obsloužit K zákazníků, takže je nutné, aby součet zákazníků v úsecích přiřazených jedné pizzerii byl nejvýše K. Marco by byl rád, kdyby náklady na rozšíření jeho firmy byly co nejnižší, a proto chce pokrýt všechny úseky kružnice co nejmenším počtem pizzerií.

Formát vstupu: Jméno vstupního souboru je `pizza.in`. První řádek obsahuje dvě celá kladná čísla N a K, která udávají počet úseků, na které je kružnice (hranice města) rozdělena, a počet zákazníků, které dokáže jedna pizzerie obsloužit. Každý z následujících N řádků obsahuje jedno celé číslo A_i, $i = 1, \ldots, N$, které udává, kolik zákazníků žije v i-tém úseku kružnice. Můžete předpokládat, že $1 \leqq A_i \leqq K$ pro všechna $i = 1, \ldots, N$.

Formát výstupu: Jméno výstupního souboru je `pizza.out`. První řádek obsahuje číslo M, které udává minimální počet pizzerií, které musí Marco otevřít. Každý z následujících M řádků obsahuje dvě čísla S_i a T_i, $i = 1, \ldots, M$, určující úseky, které budou pokryty i-tou pizzerií. Pokud $1 \leqq S_i \leqq T_i \leqq M$, pak i-tá pizzerie bude obsluhovat úseky $S_i, S_i + 1, \ldots, T_i$. Pokud $1 \leqq T_i < S_i \leqq M$, pak bude tato pizzerie obsluhovat úseky $T_i, T_i + 1, \ldots, S_i$. Oblasti, které jsou obsluhovány jednotlivými pizzeriemi, musí být navzájem disjunktní a každý úsek musí být obsluhován některou z pizzerií. Navíc součet počtů zákazníků, kteří žijí v úsecích obsluhovaných jednou pizzerií, musí být nejvýše K.

Příklad:

`pizza.in`	`pizza.out`
`7 11`	`4`
`4`	`6 1`
`4`	`2 3`
`7`	`4 4`
`6`	`5 5`
`6`	
`4`	
`2`	

P – II – 4

Grafomat na lovu

Definice grafomatu je uvedena v textu úlohy P–I–4.

Soutěžní úloha. Král Lamželezo XXVI. tuze rád organizoval lovy. Příčilo se mu ale zabíjení čehokoliv živého, ať už to byla zvířata nebo nešikovní honci navzájem, a tak si pořídil robotické lovce a posílal je lovit mechanickou zvěř. Lovci pokaždé utvořili kruhovou formaci okolo nory, každý lovec se propojil s oběma sousedními a ledva si libovolný z nich všiml, že zvíře vystrčilo anténky, předal zprávu ostatním a všichni naráz

vypálili. Vaším úkolem je napsat program pro grafomat, který bude lovce řídit. Můžete předpokládat, že zvíře zahlédne vždy jen jediný z lovců.

Mějme 2-graf složený z jediného cyklu sudé délky, tj. z vrcholů očíslovaných od 0 do $N-1$, přičemž vrchol i je spojen hranou označenou 1 s vrcholem $(i+1) \bmod N$ a hranou označenou 2 s vrcholem $(i-1) \bmod N$ (tedy stejně, jako na prvním obrázku ve studijním textu). Váš program se má chovat následovně: pokud dostane $x = 0$ ve všech vrcholech, ihned se zastaví s $y = 0$ (lovci nic nevidí, a proto nestřílí); pokud dostane v jednom vrcholu $x = 1$ a v ostatních $x = 0$, má se po nějakém konečném počtu kroků zastavit s $y = 1$ ve všech vrcholech, přičemž ve všech předchozích krocích musí být y nulové (jeden lovec zvěř zahlédl, takže po čase všichni současně vystřelí). Pro ostatní kombinace vstupů se program může chovat libovolně.

Pokud vám to pomůže, můžete předpokládat, že počet vrcholů grafu je v nějakém vhodném tvaru (třeba mocnina dvojky, druhá mocnina apod.).

P – III – 1

Pizza vrací úder

Rozmach Marcovy firmy narazil na konec roku, přesněji řečeno na daňové přiznání. Během rozvážení pizz a zřizování nových poboček Marco neměl čas zabývat se účetnictvím a nyní s úděsem zjistil, že si u svých zběžných poznámek zapomněl zapsat, co jsou příjmy a co výdaje. Nicméně nezpanikařil, vzpomněl si na příběhy, které vykládal jeho dědeček (bývalý kápo italské mafie), a jal se účetní záznamy doplnit (zfalšovat).

Aby výsledek vypadal co nejdůvěryhodněji, rozhodl se použít čísla ze svých poznámek a pouze si u nich zvolit, které jsou příjmy a které výdaje. Navíc se rozhodl, že když falšovat, tak pořádně. Rád by vypadal jako úspěšný obchodník, a tedy nekončil ve ztrátě. Na druhou stranu, chce platit co nejmenší daně, tedy jeho zisk by měl být co nejmenší. Při řešení tohoto nelehkého úkolu by mu mohlo pomoci to, že velikosti čísel v jeho poznámkách jsou podstatně menší než jejich počet.

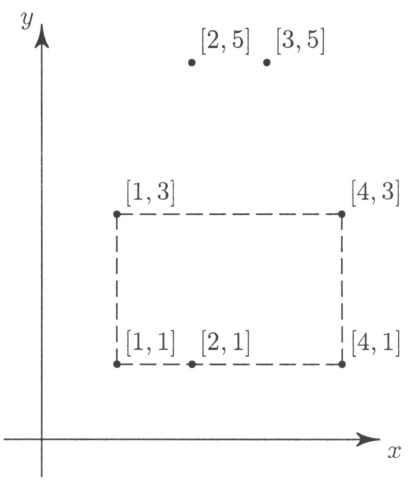
Soutěžní úloha. Je dáno n přirozených čísel $a_1, \ldots, a_n$ z rozsahu 1 až k; hodnota k je typicky řádově menší než n. Vaším úkolem je nalézt čísla $s_i \in \{+1, -1\}$ taková, že součet $z = s_1 a_1 + s_2 a_2 + \ldots + s_n a_n$ je nezáporný a zároveň nejmenší možný. Např. pro $n = k = 4$ a $a_i = i$ pro $i = 1, 2, 3, 4$ je optimální řešení $s_1 = s_4 = +1$ a $s_2 = s_3 = -1$ s velikostí součtu $z = 0$.

P – III – 2

Obdélník

V rovině je dáno n vesměs různých bodů $B_1, \ldots, B_n$. Vaším úkolem je navrhnout algoritmus, který nalezne obdélník $A_1A_2A_3A_4$, jehož vrcholy jsou některé ze zadaných bodů, tj. $A_1 = B_i$ pro nějaké i, $1 \leqq i \leqq n$, analogicky pro A_2, A_3 a A_4, a počet bodů B_i ležících uvnitř obdélníku $A_1A_2A_3A_4$ je maximální možný. Navíc se požaduje, aby hrany obdélníku $A_1A_2A_3A_4$ byly rovnoběžné s osami, tj. x-ové souřadnice vrcholů A_1 a A_4 a vrcholů A_2 a A_3 byly stejné a rovněž y-ové souřadnice vrcholů A_1 a A_2 a vrcholů A_3 a A_4 byly stejné. Body, které leží na hranách obdélníku $A_1A_2A_3A_4$, považujeme za body ležící uvnitř obdélníku. Pokud zadané body netvoří žádný obdélník $A_1A_2A_3A_4$, který by vyhovoval podmínkám zadání, program vypíše vhodnou zprávu.

Jedno z možných zadání a příklad řešení (v tomto případě jediného optimálního) jsou na obr. 30. Je zde $n = 7$ bodů se souřadnicemi $[1, 1]$, $[2, 1]$, $[4, 1]$, $[1, 3]$, $[4, 3]$, $[2, 5]$ a $[3, 5]$. Optimálním řešením je obdélník s rohy $[1, 1]$, $[4, 1]$, $[4, 3]$ a $[1, 3]$, který obsahuje pět bodů.



Obr. 30

P – III – 3

Grafomat a šamani

Na indiánské vesnice kmene Grafomatonů se zvolna snáší soumrak. Příštího dne začne největší ze slavností tohoto léta, na níž se sejdou všichni členové kmene. Indiáni se v tichém očekávání chystají ulehnout do svých teepee, pouze šamani postávají u signálních ohňů a pilně vysílají kouřové signály do sousedních vesnic. Rituál totiž vyžaduje, aby lidé z právě poloviny vesnic přišli pomalováni červeně a z druhé poloviny zeleně. Jen šamani vědí, jak se na tom dokáží domluvit — možných signálů je totiž pomálu a každá vesnice vidí jen na tři sousední. Jak to mohou dělat? Jak? ...

Když jste se ze sna probudili, napadlo vás, že indiánské domlouvání docela přesně odpovídá této úloze pro grafomat:

Soutěžní úloha. Napište program pro grafomat, který v zadaném 3-grafu s jedním označeným vrcholem označí právě polovinu vrcholů a skončí. Předpokládejte, že graf je souvislý (z každého vrcholu jde po hranách dojít do každého) a že má sudý počet vrcholů, takže rozdělení na poloviny je vždy možné. (Sudý počet vrcholů mají ve skutečnosti všechny 3-grafy, ale to zde nebudeme dokazovat.)

Vstup bude tvořen proměnnou x, která bude v označeném vrcholu rovna jedné a všude jinde nulová.

Výstupem programu bude proměnná y, nulová v právě polovině vrcholů a jedničková ve zbývajících.

Nápověda: Zkuste si nejdříve rozmyslet, jak by se úloha dala řešit, pokud by namísto obecného 3-grafu byl zadán strom (tj. souvislý graf bez cyklů).

Definice grafomatu je uvedena v textu úlohy P–I–4. Nadto si dovolujeme připomenout, že počet stavů každého automatu musí být konečný, takže nelze používat proměnné, jejichž rozsah hodnot závisí na velikosti vstupu.

P – III – 4

Policie zasahuje

Program:	`policie.pas` / `policie.c` / `policie.cpp`
Vstup:	`policie.in`
Výstup:	`policie.out`

I v tomto kole se na vás radní Stínové Prahy obracejí s dalším, ještě naléhavějším, problémem. Ve městě se totiž usídlila mafie a její řádění překročilo únosnou mez. Proto byla městská policie pověřena učinit řádění mafie přítrž. Jak už to ale bývá, není dostatek důkazů o činnosti mafie, a tak se policisté rozhodli nějakou dobu sledovat, jak se mafiáni mezi sebou stýkají. Mafiáni jsou však prohnaní a nechodí jen po ulicích, ale využívají ke svým přesunům i kanalizační systém města. Do kanalizace je tedy třeba rozestavit policejní hlídky tak, aby bylo zamezeno tajným kontaktům mezi mafiány. Přesněji, je potřeba, aby na každé cestě mezi domy dvou mafiánů byla alespoň jedna policejní hlídka.

Tento nelehký úkol naštěstí zjednodušuje fakt, že kanalizačním systémem Stínové Prahy lze mezi každými dvěma domy projít právě jedním způsobem. Takže pokud se má mafián dostat kanalizací z jednoho místa na druhé, má jen jedinou možnost, kudy kanalizací projít (pokud nechce jít žádným místem dvakrát). Speciálně to tedy znamená, že kanalizace má „acyklickou" strukturu a je souvislá, jako např. kanalizační systém na obrázku.

Vaším úkolem je napsat program, který pro daný popis kanalizačního systému a seznam domů ve vlastnictví mafiánů určí minimální počet hlídek, které je nutné do kanalizačního systému rozmístit tak, aby na každé cestě mezi dvěma domy mafiánů byla alespoň jedna hlídka. Hlídky lze umísťovat pouze do míst větvení, tj. hlídka nemůže být umístěna uprostřed stoky. Speciálně hlídka, která je umístěna ve větvení, kam je napojen dům některého z mafiánů, odděluje tento dům od všech ostatních domů.

Vstup: Na prvním řádku vstupního souboru `policie.in` jsou dvě celá čísla n ($3 \leqq n \leqq 100\,000$) a p ($2 \leqq p < n$) oddělená jednou mezerou. Číslo n udává počet větvení v kanalizačním systému — *větvením* rozumíme buď slepý konec nějaké stoky (napojený na dům, který ale nemusí patřit mafiánovi) nebo křižovatku, ze které vedou alespoň dvě stoky. Kanalizační systém města je pak tvořen $n - 1$ stokami, z nichž každá spojuje dvě větvení. Číslo p udává počet domů, které jsou ve vlastnictví mafiánů.

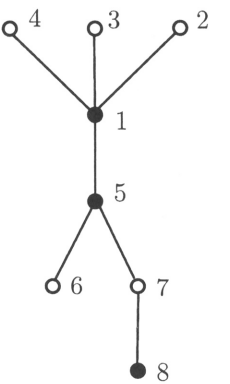
Větvení v kanalizaci jsou očíslována přirozenými čísly od 1 do n. Domy jsou do kanalizace připojeny jen v místech větvení. Na každém z následujících $n - 1$ řádků vstupního souboru jsou dvě celá čísla a_i a b_i ($1 \leqq a_i, b_i \leqq n$), která určují větvení spojená i-tou stokou.

Posledních p řádků vstupního souboru obsahuje vždy jedno celé číslo od 1 do n. Tato čísla jsou navzájem různá a určují čísla větvení v kanalizaci, kam jsou napojeny domy mafiánů.

Výstup: Váš program má do výstupního souboru `policie.out` vypsat nejmenší možný počet míst, která je třeba obsadit hlídkou, aby na cestě mezi každými dvěma domy mafiánů byla alespoň jedna hlídka.

Příklad:

```
policie.in      policie.out
8 5             2
1 2
1 3             (Hlídky je možné umístit
1 4             na větvení s čísly 1 a 5.)
1 5
5 6
5 7
7 8
2
3
4
6
7
```



P – III – 5

Rybka

Program: `rybka.pas / rybka.c / rybka.cpp`
Vstup: `rybka.in`
Výstup: `rybka.out`

Za horkých bezmračných letních dní se voda v rybníce Blaťáku někdy skoro vaří. Slunce nemilosrdně žhne a malé rybky se spěchají skrýt do hlubších a chladnějších částí rybníka. Jen rybka Julka se opozdila za ostatními a už skoro umdlévá.

Taková choulostivá malá rybka, jako je Julka, snáší jen určitý rozsah teplot, řekněme t_1 až t_2 (včetně krajních hodnot). Ráno mají různé části rybníka různou teplotu a jakmile vyjde slunce, všechny části rybníka se ohřívají stejně rychle, a to o 1 stupeň za 1 časovou jednotku. Rybník Blaťák je obdélníkový a je rozdělen na $m \times n$ stejně velkých čtvercových polí. Pole na pozici $[i, j]$ má ráno teplotu $T[i, j]$ stupňů ($1 \leqq i \leqq m$, $1 \leqq j \leqq n$). Julka je ráno na pozici $[J_x, J_y]$ a potřebovala by se dostat do bezpečí na pozici $[C_x, C_y]$. Julka se za každou časovou jednotku posune

na jedno ze čtyř přes hranu sousedících čtvercových polí nebo zůstane stát. Pak se vždy teplota všech polí zvýší o 1 stupeň.

Samozřejmě se rybička cestou nesmí přehřát ani nachladit, tj. pole, kde se Julka vyskytuje, musí mít teplotu T, $t_1 \leqq T \leqq t_2$. Tato teplota musí být dodržena jak v okamžiku, když Julka na pole vplouvá, tak když jej opouští (mezi čímž se teplota zvýšila alespoň o 1 stupeň). Vyjímku tvoří jen cílové pole, na které smí vplout nezávisle na jeho teplotě a tím se dostat do bezpečí. Rybka Julka nesmí samozřejmě na své cestě opustit rybník.

Napište program, který dostane na vstupu Julčin teplotní rozsah $\langle t_1, t_2 \rangle$ ($0 \leqq t_1 < t_2 \leqq 10^6$), velikost Blaťáku $m \times n$ ($1 \leqq m, n \leqq 1\,000$), počáteční a cílovou pozici Julky $[J_x, J_y]$ a $[C_x, C_y]$ ($1 \leqq J_x, C_x \leqq m, 1 \leqq J_y, C_y \leqq n$) a počáteční teplotu každého pole rybníka $T[i,j]$ ($0 \leqq T[i,j] \leqq 10^6$) a najde pro naši malou rybku cestu do bezpečí respektující teplotní omezení či zjistí, že taková cesta neexistuje. Nalezená cesta má být nejrychlejší možná, tj. má trvat co nejméně časových jednotek. Pokud existuje více nejrychlejších cest, program může vypsat libovolnou z nich. Můžete předpokládat, že všechny teploty t_1, t_2 a $T[i,j]$ jsou celá čísla. Navíc také můžete předpokládat, že rybka je na počátku v poli s teplotou pro ni přijatelnou a že počáteční pole je různé od cílového.

Vstup: Na prvním řádku vstupního souboru `rybka.in` jsou čtyři celá čísla m, n, t_1 a t_2 oddělená mezerami, na druhém řádku jsou pak souřadnice J_x, J_y, C_x a C_y. Všechna čísla m, n, t_1, t_2, J_x, J_y, C_x a C_y splňují výše uvedená omezení. Rybník Blaťák je orientován tak, že pole se souřadnicemi $[i,j]$ sousedí severní hranou s polem se souřadnicemi $[i, j-1]$, jižní hranou s polem se souřadnicemi $[i, j+1]$, západní hranou s polem $[i-1, j]$ a východní hranou s polem $[i+1, j]$. Posledních n řádků vstupního souboru popisuje počáteční teploty jednotlivých částí rybníka, na řádku $j+2$ se tedy nacházejí čísla $T[1,j], T[2,j], T[3,j], \ldots, T[m,j]$ vzájemně oddělená mezerami.

Výstup: Do souboru `rybka.out` vypište Julčinu cestu jako posloupnost pokynů pro pohyb rybky oddělených mezerami. Pokyn je buď jeden znak `S`, `J`, `V` nebo `Z`, který určuje směr pohybu rybky, nebo kladné číslo udávající počet časových jednotek, po které se rybka nepohybuje. Jednotlivé pokyny jsou odděleny jednou mezerou. Výstupní soubor nesmí obsahovat dvě čísla za sebou a poslední pokyn, který obsahuje, musí být pokynem k pohybu rybky. V případě, že cesta neexistuje, vypište do výstupního souboru řádek s textem „`Chudak Julka!`“.

Příklad:

`rybka.in`
3 4 10 20
2 2 3 4
9 7 13
0 18 15
1 15 19
2 3 0

`rybka.out` *(jedna z více možností)*
V S 1 Z Z 5 J J J V V

`rybka.in`
3 2 20 30
1 1 3 1
25 13 25
27 24 0

`rybka.out`
`Chudak Julka!`