63. ročník Matematické olympiády – 2013/2014

Úlohy ústředního kola kategorie P – 1. soutěžní den

Na řešení úloh máte 4,5 hodiny čistého času. Řešení každé úlohy pište na samostatný list papíru. Při soutěži je zakázáno používat jakékoliv pomůcky kromě psacích potřeb (tzn. knihy, kalkulačky, mobily, apod.).

Řešení každého příkladu musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, argumenty zdůvodňující jeho správnost, diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není povoleno odkazovat se na Vaše řešení předchozích kol, opravovatelé je nemají k dispozici; na autorská řešení se odkazovat můžete.
- **Zápis algoritmu.** V úlohách **P-III-1** a **P-III-2** je třeba uvést zápis algoritmu, a to buď ve tvaru zdrojového textu nejdůležitějších částí programu v jazyce Pascal nebo C/C++, nebo v nějakém dostatečně srozumitelném pseudokódu. Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod. V úloze **P-III-3** je nutnou součástí řešení program, ve kterém budete využívat volání funkcí „mimozemských počítačů KSP“.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu (včetně zdůvodnění správnosti) a efektivita zvoleného algoritmu. Algoritmy posuzujeme zejména podle jejich časové složitosti, tzn. podle závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce.

P-III-1 Buridan a kavárny

Buridanův osel je známý filozofický myšlenkový experiment. Představte si, že hladového osla postavíte přesně doprostřed mezi dvě kupky sena. Osel by si sice dal seno, ale jelikož je situace dokonale symetrická, nemá se podle čeho rozhodnout, zda jít doleva nebo doprava. A tak zůstane stát na místě, až chudák zdechne hlady.

Davida už znáte z domácího kola. Stále si hledá ubytování na Manhattanu. Nyní si ale uvědomil, že mu hrozí stejný problém jako Buridanovu oslovi: pokud by existovaly dvě kavárny, které jsou od jeho bytu stejně vzdáleny, mohlo by se mu stát, že se mezi nimi nebude umět rozhodnout a nedopadne to dobře.

Pro jednoduchost si Manhattan představíme jako čtvercovou síť, po níž se lze pohybovat pouze čtyřmi základními směry. V některých mřížových bodech (tzn. na některých křižovatkách) jsou kavárny – v každém bodě nejvýše jedna.

Soutěžní úloha

Vstupem programu je mapa Manhattanu. Pro každou křižovatku určete, zda tam David může bydlet – tzn. zda neexistují žádné dvě kavárny, které by byly stejně vzdáleny od dané křižovatky.

Popis vstupu

Na prvním řádku je rozměr n čtvercové sítě, která představuje Manhattan – síť tvoří n vodorovných a n svislých cest. Máme tedy přesně n^2 křižovatek. Zbytek vstupu tvoří n řádků, na každém z nich je n čísel oddělených mezerami: 1 představuje křižovatku s kavárnou, 0 křižovatku bez kavárny.

Popis výstupu

Vypište n řádků a na každém z nich n znaků oddělených mezerami: pro každou křižovatku buď '`A`', pokud tam David může bydlet, nebo '`N`', pokud tam bydlet nemůže.

Hodnocení

Plných 10 bodů dostanete za řešení s asymptoticky optimální časovou složitostí. Za pomalejší správné řešení můžete dostat 3 až 9 bodů podle konkrétní časové složitosti.

Příklady

Vstup:
```
5
1 0 0 1 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 1
0 0 0 0 0
```

Výstup:
```
A A A A A
A A A A N
N N N N A
A A A A A
A A A A A
```

Vstup:
```
3
1 0 1
0 0 0
1 0 1
```

Výstup:
```
N N N
N N N
N N N
```

Vstup:
```
2
0 1
0 0
```

Výstup:
```
A A
A A
```

P-III-2 Přetahování lanem

V Šikmé Lhotě se koná tradiční turnaj v přetahování lanem. Jelikož ale v této obci není nikde žádná rovina, turnaj probíhá na úbočí kopce. Sklonu kopce odpovídá koeficient $q > 1$. Do turnaje se přihlásilo n hráčů. V pořadí od nejmladšího k nejstaršímu dostali čísla 1 až n. Sílu hráče i označíme jako s_i.

Na začátku turnaje jsou všichni hráči aktivní. Turnaj se skládá z několika zápasů. Každého zápasu se účastní všichni dosud aktivní hráči a na zápas se rozdělí do dvou týmů. Horní tým (na vrchu kopce) tvoří k nejstarších aktivních hráčů, dolní tým (na spodku kopce) tvoří všichni ostatní. Tým tahající lano směrem vzhůru je zjevně v nevýhodě: vyhraje právě tehdy, když součet jejich sil je ostře větší než q-násobek součtu sil týmu tahajícího lano směrem dolů. Turnaj končí, jakmile v některém zápasu zvítězí horní družstvo.

Po každém zápasu jeden z hráčů přestane být aktivní. Je to vždy buď nejmladší aktivní hráč (který již musí jít domů), nebo nejstarší aktivní hráč (který odejde do hospody). Hodnota k se nemění, takže když po zápasu odejde nejstarší hráč, v následujícím zápasu bude za horní tým hrát i nejstarší z hráčů, kteří dosud hráli za dolní tým.

Soutěžní úloha

Na vstupu dostanete počet hráčů n, jejich síly $s_1, \ldots, s_n$, koeficient strmosti kopce q a velikost horního týmu k. Napište program, který vypočítá, kolik nejvýše zápasů se může v celém turnaji uskutečnit. Jinými slovy, váš program by měl nalézt takové pořadí odcházejících hráčů, při němž první výhra horního týmu nastane co nejpozději.

Popis vstupu a výstupu

Na prvním řádku vstupu jsou zadána celá čísla n, k a racionální číslo q. Můžete předpokládat, že $1 \leq k \leq n$, že $q > 1$ a že všechny vaše výpočty s číslem q dávají přesné výsledky.

Na druhém řádku je n kladných celých čísel $s_1, \ldots, s_n$: síly jednotlivých hráčů. Můžete předpokládat, že se jejich součet vejde do běžné celočíselné proměnné.

Na výstup vypište jediný řádek a na něm jedno celé číslo – maximální možný počet zápasů v turnaji.

Hodnocení

Za řešení s exponenciální časovou složitostí vzhledem k n získáte nejvýše 3 body. Za libovolné správné řešení s polynomiální časovou složitostí vzhledem k n dostanete alespoň 5 bodů. Časovou složitost optimálního řešení vám neprozradíme. Nezapomeňte zdůvodnit správnost svého algoritmu!

Příklady

Vstup:	*Výstup:*
`5 2 1.01` `40 70 1000 30 20`	`4`

Tento kopec je skoro rovina, horní tým téměř není zvýhodněn. Nahoře začínají hráči se silou $30+20=50$ *a dole hráči se silou* $40+70+1000=1110$. *Jelikož* $1110\times 1.01>50$, *dolní tým první zápas snadno vyhraje.*

Turnaj bude trvat nejdéle tehdy, když po prvním zápasu odejde nejmladší hráč (se silou 40) a po druhém zápasu odejde opět nejmladší hráč (se silou 70). Po třetím zápasu je už jedno, kdo odejde – na čtvrtý zápas totiž zůstanou nahoře dva hráči a dole nikdo, takže horní tým jistě vyhraje.

Vstup:	*Výstup:*
`12 7 2.0` `2 4 9 5 3 3 4 8 8 6 1 6`	`4`

Jeden možný optimální průběh zápasů: po prvním zápasu odejde nejmladší (síla 2), po druhém nejstarší (síla 6), po třetím zase nejmladší (síla 4). Ke čtvrtému zápasu na vrchu kopce nastoupí hráči se silou $3+3+4+8+8+6+1=33$, *zatímco dole budou jen hráči se silou* $9+5=14$. *Protože* $14\times 2.0<33$, *čtvrtý zápas už vyhraje horní tým a turnaj skončí.*

P-III-3 Mimozemské počítače

K této úloze se vztahuje studijní text uvedený na následujících stranách. Studijní text je identický se studijním textem z domácího a krajského kola. V katalogu problému uvádíme pouze hamiltonovskou cestu a kružnici, jiné problémy nebudou v tomto kole potřeba.

Soutěžní úloha

a) *(1 bod)* Mimozemšťané nám dodali sálový KSP, který rozhoduje problém existence hamiltonovské cesty obsahující předepsanou hranu. Tento KSP má tedy funkci `cesta_s_hranou(n,E,u,v)`, která dostane jako parametry počet n vrcholů grafu, seznam E jeho hran a dvě čísla vrcholů u a v. Jestliže v daném grafu existuje hamiltonovská cesta, na níž vrcholy u a v následují bezprostředně po sobě, KSP rozsvítí zelené světlo, jinak rozsvítí červené světlo. Tuto funkci můžete použít pouze jednou a jejím zavoláním výpočet vašeho programu skončí.

Na vstupu dostanete neorientovaný graf G. Napište program s polynomiální časovou složitostí, který rozsvítí zelené nebo červené světlo podle toho, zda G obsahuje aspoň jednu hamiltonovskou cestu.

b) *(5 bodů)* Mimozemšťané nám dodali kufříkový KSP, který rozhoduje problém existence hamiltonovské cesty. Tento KSP má tedy funkci `cesta(n,E)`, která dostane jako parametry počet n vrcholů grafu a seznam E jeho hran. Na výstupu tato funkce vrací `True` nebo `False` podle toho, zda v grafu existuje aspoň jedna hamiltonovská cesta. Funkci `cesta` můžete volat vícekrát.

Na vstupu dostanete neorientovaný graf G, který obsahuje hamiltonovskou cestu. Napište pomocí tohoto kufříkového KSP program, který v polynomiálním čase (nepočítaje přitom volání funkce `cesta`) najde v G jednu hamiltonovskou cestu.

Pozor! Počet bodů, které dostanete, bude záviset na tom, kolik řádově volání funkce `cesta` vaše řešení v nejhorším případě potřebuje.

c) *(4 body)* Mimozemšťané nám dodali sálový KSP, který rozhoduje problém existence hamiltonovské cesty. Tento KSP má tedy funkci `cesta(n,E)`, která dostane jako parametry počet n vrcholů grafu a seznam E jeho hran. Jestliže v daném grafu existuje hamiltonovská cesta, KSP rozsvítí zelené světlo, jinak rozsvítí červené světlo. Tuto funkci můžete použít pouze jednou a jejím zavoláním výpočet vašeho programu skončí.

Na vstupu dostanete *orientovaný* graf G. Každá hrana tohoto grafu má tedy určen jeden směr, ve kterém po hraně můžeme jít. Napište program s polynomiální časovou složitostí, který rozsvítí zelené nebo červené světlo podle toho, zda G obsahuje aspoň jednu *orientovanou* hamiltonovskou cestu. (Tedy hamiltonovskou cestu, která každou svoji hranu použije ve správném směru.)

Programovací jazyky

Ve svých řešeních můžete používat libovolný strukturovaný programovací jazyk. Vhodně si zvolte potřebné datové struktury. Například v Pascalu by funkce z podúlohy a) mohla vypadat následovně:

```
type hrana = array[0..1] of longint;
procedure cesta_s_hranou(n, m: longint; E: array of hrana; u, v: longint);
```

V jazyce C++ můžeme použít buď pole:

```
void cesta_s_hranou(int n, int m, int E[][2], int u, int v);
```

Nebo třeba vektor dvojic:

```
void cesta_s_hranou(int n, vector< pair<int,int> > E, int u, int v);
```

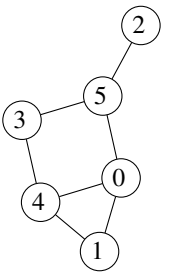
Studijní text

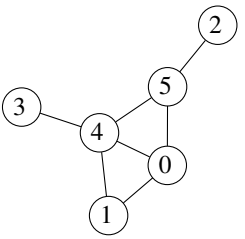
Studijní text uvedený na dalších stranách se odkazuje na několik problémů, jako například „existence hamiltonovské kružnice“. Seznámíme se proto nejprve se stručnou definicí těchto problémů.

Neorientovaný graf je uspořádaná dvojice (V, E), kde V je konečná množina objektů zvaných *vrcholy* grafu a E je konečná množina neuspořádaných dvojic vrcholů; její prvky nazýváme *hrany* grafu. Graf si můžeme představit jako silniční síť: V je množina měst a E je množina dvojic měst spojených silnicemi. Počet vrcholů budeme značit n, počet hran označíme m. Jednotlivé vrcholy budeme číslovat od 0 do $n - 1$.

Problém: Hamiltonovská cesta z u do v

Je dán neorientovaný graf G a jeho dva různé vrcholy u a v. Existuje v grafu G cesta, která začíná ve vrcholu u, končí ve vrcholu v a navštíví každý vrchol grafu G právě jednou?



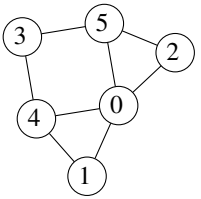


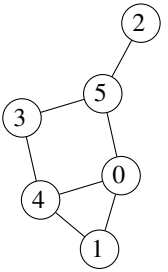
Graf vlevo obsahuje hamiltonovskou cestu z 1 do 2: můžeme jít postupně přes vrcholy 1, 0, 4, 3, 5 a 2.

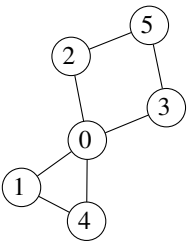
Graf vpravo hamiltonovskou cestu z 1 do 2 neobsahuje: když chceme navštívit vrchol 3, půjdeme dvakrát přes vrchol 4.

Problém: Hamiltonovská kružnice

Je dán neorientovaný graf G s $n \geq 3$ vrcholy. Existuje v grafu G kružnice, která navštíví každý vrchol grafu G právě jednou?







Graf vlevo obsahuje hamiltonovskou kružnici: začneme třeba v 1 a postupně jdeme do 0, 2, 5, 3, 4 a zpět do 1. Graf uprostřed ani graf vpravo hamiltonovskou kružnici neobsahují.

KSP – konkrétní semiorganické počítače

Píše se rok 2113. Naši Zemi před několika lety kontaktovala vyspělá mimozemská rasa z planety Žbluňk. Mimozemšťané nás zásobují svými konkrétními semiorganickými počítači (KSP), které umí řešit různé konkrétní výpočetní problémy. Naším cílem je integrovat tyto KSP do normálních počítačů a přinutit je tak řešit naše problémy.

Mimozemským počítačům ale vůbec nerozumíme, všechny snahy o jejich rozebrání byly neúspěšné. Můžeme je využít jediným způsobem – zadat jim vstupní data a počkat na výsledek. Zajímavé je, že u KSP nezáleží na velikosti vstupních dat ani na konkrétním problému, který daný KSP řeší. Když libovolný KSP spustíme s libovolnými vstupními daty, výsledek dostaneme vždy přesně za 47 setin sekundy. (Při odhadování časové složitosti programů toto považujeme za konstantu.)

Kufříkový KSP

Mimozemšťané nám dodávají dva druhy KSP: *kufříkové* a *sálové*.

Kufříkový KSP má tvar kufříku se dvěma porty. Do jednoho připojíme kabel se vstupem, do druhého naopak kabel, po němž nám KSP pošle svůj výstup. Kufříkový KSP tedy můžeme použít v našem programu libovolně mnohokrát s různými vstupními hodnotami.

Příklad

Mimozemšťané nám dodali kufříkový KSP, který rozhoduje problém existence hamiltonovské cesty z u do v. Tento KSP počítá funkci `cesta(n,E,u,v)`, která dostává jako parametry počet n vrcholů grafu, seznam E jeho hran a dvě čísla vrcholů u a v. Parametr E je seznam m dvojic čísel, každé z rozsahu od 0 do $n-1$. Na výstupu tato funkce vrací `True` nebo `False` podle toho, zda zadaný graf obsahuje příslušnou hamiltonovskou cestu.

Naším úkolem je napsat program, který bude v polynomiálním čase řešit problém existence hamiltonovské kružnice. Na vstupu dostaneme neorientovaný graf s $n \geq 3$ vrcholy a máme zjistit, zda obsahuje hamiltonovskou kružnici.

Analýza zadání

1. Program na vstupu dostane n (počet vrcholů grafu) a E (seznam hran grafu).
2. Vykoná nějaké výpočty, při nichž může libovolně používat funkci `cesta`.
3. Vrátí na výstupu `True` nebo `False` podle toho, zda vstupní graf obsahuje hamiltonovskou kružnici.

Řešení

```
def kruznice(n,E):
    for (x,y) in E:
        # pro každou hranu (x,y) v seznamu hran E:
        newE = [ (u,v) for (u,v) in E if (u,v) != (x,y) ]
        # NewE obsahuje všechny hrany kromě (x,y)
        if cesta(n,newE,x,y): return True
    return False
```

Postupně si pro každou hranu (x, y) zadaného grafu položíme otázku: „Existuje hamiltonovská kružnice procházející hranou (x, y)?“ Kdy taková kružnice existuje? Právě tehdy, když ve zbytku grafu existuje hamiltonovská cesta z x do y. Vytvoříme si tedy nový seznam hran `newE`, do kterého vložíme všechny hrany kromě (x, y), a na takto upravený graf zavoláme funkci `cesta` našeho kufříkového KSP.

Jestliže v původním grafu nějaká hamiltonovská kružnice existuje, náš program ji najde a vrátí odpověď `True`, jakmile vyzkoušíme některou z hran tvořících kružnici jako (x, y). Naopak, jestliže náš program odpoví `True`, pak v původním grafu existuje hamiltonovská cesta z nějakého vrcholu x do nějakého vrcholu y. Tato cesta společně s hranou (x, y) tvoří hledanou kružnici. Náš program tedy skutečně dělá to, co má.

Časová složitost popsaného programu je $\Theta(m^2)$, kde m je počet hran zadaného grafu. Protože v neorientovaném grafu platí $m \leq n(n-1)/2$, můžeme naši časovou složitost shora odhadnout jako $\mathcal{O}(n^4)$.

Poznámka

Existuje i efektivnější řešení. Stačí si uvědomit, že před hledáním hamiltonovské cesty z x do y vůbec nemusíme odstraňovat hranu (x, y) z grafu. Hamiltonovská cesta z x do y v grafu s $n \geq 3$ vrcholy ji stejně nemůže obsahovat. Stačí tedy pro každou hranu (x, y) zjistit, zda platí `cesta(n,E,x,y)`.

Sálový KSP

Sálový KSP je obrovský a jeho jediným výstupem jsou dvě světla – červené a zelené. S tímto výstupem dále nepracujeme.

Příklad

Mimozemšťané nám dodali sálový KSP, který rozhoduje problém 3-partitnosti. Tento KSP má funkci `tri_partitnost(X)`, která rozsvítí zelené nebo červené světlo podle toho, zda posloupnost X je 3-partitní – tedy zda se její prvky dají rozdělit do *tří* skupin se shodným součtem.

Na vstupu dostanete posloupnost kladných celých čísel A. Napište program s polynomiální časovou složitostí, který rozsvítí zelené nebo červené světlo podle toho, zda je posloupnost A 2-partitní (tzn. zda můžeme prvky posloupnosti A rozdělit do *dvou* skupin se shodným součtem).

Analýza zadání

1. Program na vstupu dostane posloupnost čísel A.
2. Vykoná nějaké výpočty a vytvoří nějakou novou posloupnost čísel X.
3. Na konci (každé možné větve) výpočtu jednou zavolá funkci `tri_partitnost(X)`, která rozsvítí správné světlo.

Řešení

```
def dva_partitnost(A):
    s = sum(A)
    if s%2 == 0:                # s%2 je zbytek po dělení s dvěma
        X = A + [ s//2 ]        # // je celočíselné dělení
    else:
```

```
    X = [1]                 # [1] je posloupnost,
                            # která určitě není 3-partitní
tri_partitnost(X)
```

Nechť jsme dostali vstupní posloupnost $A = (a_1, \ldots, a_n)$. Spočítáme si její součet s.

Je-li s sudé, přidáme na konec vstupní posloupnosti ještě jeden prvek s hodnotou $s/2$. Takto upravenou posloupnost pošleme do KSP. Ten nám odpoví, zda je tato posloupnost 3-partitní. Jenže upravená posloupnost je 3-partitní právě tehdy, když byla původní posloupnost 2-partitní. KSP tedy za nás právě vyřešil zadanou úlohu.

Proč platí výše uvedená ekvivalence? V nové posloupnosti X je součet všech prvků roven $3s/2$. Jestliže je tedy X 3-partitní, pak každá ze skupin, na něž X rozdělíme, má součet rovný přesně $s/2$. Potom ale nutně jednu z těchto tří skupin tvoří samotný přidaný prvek s hodnotou $s/2$. Naše nová posloupnost je proto 3-partitní právě tehdy, když je možné ostatní prvky rozdělit do dvou skupin se stejným součtem – tzn. právě tehdy, když je původní posloupnost A 2-partitní.

Je-li s liché, víme rovnou, že musíme odpovědět NE. Do KSP proto chceme poslat libovolný vstup, pro který dá KSP zápornou odpověď. Takovým vstupem je třeba $X = (1)$ nebo $X = (1, 1, 2)$.

Časová složitost tohoto programu je lineární vzhledem k délce posloupnosti X.