61. ročník Matematické olympiády – 2011/2012

Úlohy domácího kola kategorie P

Úlohy P-I-1 a P-I-2 jsou praktické, vaším úkolem v nich je vytvořit a odladit efektivní program v jazyce Pascal, C nebo C++. Řešení těchto dvou úloh odevzdávejte ve formě zdrojového kódu přes webové rozhraní přístupné na stránce *http://mo.mff.cuni.cz/submit/,* kde také naleznete další informace. Odevzdaná řešení budou automaticky vyhodnocena pomocí připravených vstupních dat a výsledky vyhodnocení se dozvíte krátce po odevzdání. Pokud váš program nezíská plný počet bodů, můžete své řešení opravit a znovu odevzdat.

Úlohy P-I-3 a P-I-4 jsou teoretické, vaším úkolem je nalézt efektivní algoritmus řešící zadaný problém. Řešení úlohy se skládá z popisu navrženého algoritmu, zdůvodnění jeho správnosti (funkčnosti) a odhadu časové a paměťové složitosti. Součástí řešení je i zápis algoritmu ve formě zdrojového kódu nebo pseudokódu v úloze P-I-3 a výsledný zlomkový program v úloze P-I-4. Řešení těchto dvou úloh odevzdávejte ve formě souboru typu PDF přes výše uvedené webové rozhraní.

Řešení všech úloh můžete odevzdávat do 15. listopadu 2011. Opravená řešení úloh a seznam postupujících do krajského kola najdete na webových stránkách olympiády na adrese *http://mo.mff.cuni.cz/,* kde jsou také k dispozici další informace o kategorii P.

P-I-1 Bezpečná planeta

Ve vesmíru existují tři typy planet:

- *typ 0: neobyvatelné* planety, na nichž člověk okamžitě zahyne,
- *typ 1: nehostinné* planety, na nichž člověk může chvíli přežít, ale nemůže tam žít dlouhodobě,
- *typ 2: přívětivé* planety, na nichž člověk může žít dlouhodobě.

Mezi jednotlivými planetami cestujeme vesmírem pomocí *jednosměrných* teleportů.

Planetu nazveme *bezpečnou,* jestliže je přívětivá a zároveň jsou přívětivé také všechny planety, na které se z ní můžeme dostat pomocí teleportů (třeba i několika po sobě). Pokud tedy vysadíte člověka na bezpečné planetě, máte jistotu, že může spokojeně žít všude, kam z ní docestuje.

Planetu označíme jako *snesitelnou,* jestliže sice není bezpečná, ale můžeme se z ní pomocí teleportů (třeba i několika po sobě) dostat na bezpečnou planetu, aniž abychom museli cestou navštívit nějakou neobyvatelnou planetu. Pokud tedy vysadíte člověka na snesitelné planetě a dáte mu vhodné instrukce, jak má cestovat, po čase dorazí živý a zdravý na nějakou bezpečnou planetu. Všimněte si, že žádná neobyvatelná planeta nemůže být snesitelná.

Soutěžní úloha

Je dán počet planet n, počet teleportů m, pro každou planetu její typ a pro každý teleport odkud kam vede. Určete všechny bezpečné a všechny snesitelné planety.

Formát vstupu

První řádek standardního vstupu obsahuje dvě celá čísla n (počet planet) a m (počet teleportů). Všechny planety si očíslujeme od 1 do n. Druhý řádek vstupu obsahuje n mezerami oddělených celých čísel z množiny $\{0, 1, 2\}$ – postupně pro každé i je to typ planety číslo i. Zbytek vstupu tvoří m řádků, z nichž každý popisuje jeden teleport. Popis teleportu je tvořen dvěma čísly planet – odkud a kam tento teleport vede.

Ve vstupech, za které lze získat celkem 5 bodů, bude $1 \leq n \leq 500$ a $0 \leq m \leq 10\,000$.

V ostatních vstupech bude $1 \leq n \leq 100\,000$ a $0 \leq m \leq 200\,000$.

Formát výstupu

Program vypíše na standardní výstup dva řádky. První z nich začíná řetězcem „`bezpecne:`“ a za ním následuje posloupnost čísel všech bezpečných planet. Čísla planet vypište v rostoucím pořadí a před každým z nich vypište jednu mezeru. Druhý řádek výstupu začíná řetězcem „`snesitelne:`“, za kterým následuje posloupnost čísel všech snesitelných planet ve stejném formátu, jako na prvním řádku. Je-li některá z posloupností prázdná, pak bezprostředně za dvojtečkou bude následovat konec řádku.

Příklad

Vstup:

```
7 8
2 1 2 2 2 2 0
1 2
2 3
3 4
4 5
5 3
2 6
6 7
7 6
```

Výstup:

```
bezpecne: 3 4 5
snesitelne: 1 2
```

P-I-2 Naložená loď

Na planety, které ještě nemají vybudovány teleportové terminály, létají meziplanetární kosmické lodě s lidskou posádkou. A ta potřebuje jíst. Vaší úlohou bude přesně naplnit celý potravinový nákladní prostor kosmické lodě balíčky s jídlem.

Problém spočívá v tom, že balíčky s jídlem se připravují na Zemi, zatímco loď čeká na oběžné dráze. Balíčky je pak třeba dopravit ze Země na kosmickou loď.

Vezme se vždy několik balíčků, naloží se do dopravní kapsle, ta se vystřelí ze Země na oběžnou dráhu a tam už si ji posádka lodě odchytí.

Existuje n typů dopravních kapslí. Kapsle i-tého typu má kapacitu přesně a_i balíčků. Nevejde se do ní více balíčků, ale zároveň jich nesmí být ani méně kvůli správnému vyvážení a doletu. Máme k dispozici dostatečné množství kapslí všech typů.

Soutěžní úloha

Napište program, který na základě zadaných typů kapslí a jejich kapacit spočítá *nejmenší možný počet* kapslí potřebný k úplnému naložení kosmické lodě. Součet kapacit použitých kapslí musí tedy být *přesně roven* kapacitě k nákladního prostoru lodě.

Formát vstupu

První řádek standardního vstupu obsahuje jedno celé číslo n – počet typů kapslí. Druhý řádek obsahuje n celých čísel $a_1, \ldots, a_n$ oddělených mezerami – kapacity kapslí jednotlivých typů. Třetí řádek obsahuje jedno celé číslo k – kapacitu nákladního prostoru lodě.

Můžete předpokládat, že $a_1 < \ldots < a_n$ a že $a_1 = 1$ (tedy vždy existuje způsob, jak lze přesně celou loď naložit).

- Ve vstupech za 3 body bude platit $n \leq 5$, $a_n \leq 2\,000$ a $k \leq 30$.
- Ve vstupech za další 4 body bude platit $n \leq 30$, $a_n \leq 2\,000$ a $k \leq 1\,000\,000$.
- Ve vstupech za zbývající 3 body bude platit $n \leq 30$, $a_n \leq 2\,000$ a $k \leq 10^{18}$.

Všimněte si, že pro uložení hodnoty k musíte použít 64-bitovou proměnnou: `long long` v C/C++, `int64` v Pascalu.

Formát výstupu

Na první řádek standardního výstupu program vypíše nejmenší počet kapslí potřebný k přesnému naložení naší kosmické lodě.

Na druhý řádek vypíše jedno libovolné řešení, které využívá tento počet kapslí. Řádek bude obsahovat n celých čísel $b_1, \ldots, b_n$ oddělených mezerami, přičemž b_i udává počet použitých kapslí velikosti a_i.

Příklady

Vstup:	*Výstup:*

`4`
`1 10 100 1000`
`147`
	`12`
	`7 4 1 0`

Použijeme jednu kapsli s kapacitou 100, čtyři kapsle s kapacitou 10 a sedm kapslí s kapacitou 1. To je cekem $1 + 4 + 7 = 12$ *kapslí.*

Vstup:	*Výstup:*

`3`
`1 8 10`
`16`
	`2`
	`0 2 0`

Optimální řešení využívá dvě kapsle s kapacitou 8.

P-I-3 Skladník

Ignác byl skladníkem. Pil alkohol, sprostě nadával a v pracovní době většinou spal. Není divu, že mu šéf už dlouhá léta vyhrožoval, že ho nahradí počítačem. Až jednoho dne ta chvíle opravdu nastala. Šéf koupil počítač, posadil ho na vrátnici skladu a Ignáce propustil. Tím dosáhl přesně stejného stavu jako dosud, jenom počítači už nemusel platit žádnou mzdu.

Netrvalo dlouho a šéf si uvědomil, že počítač dokonce může zapnout a používat. Potřebuje k tomu ale od vás napsat vhodný program.

a) *(4 body)* Navrhněte co nejefektivnější algoritmus, který bude spravovat inventář skladu. Na vstup algoritmu budou přicházet informace o tom, kolik kusů jakého druhu zboží přibylo nebo ubylo. Po každém načtení nové informace musí váš algoritmus vypsat dva údaje: celkový počet kusů zboží ve skladu a počet různých druhů zboží ve skladu.

b) *(6 bodů)* Napište ještě druhý algoritmus, který bude rovněž spravovat inventář skladu. Jeho vstup bude vypadat stejně jako vstup prvního algoritmu. Po každém načtení nové informace musí tento algoritmus vypsat dva údaje: název druhu zboží, od kterého je momentálně ve skladu největší počet kusů, a tento počet kusů. Pokud existuje více takových druhů zboží, algoritmus vypíše ten z nich, jehož název je první v abecedním pořadí.

Formát vstupu

Na každém řádku vstupu je uvedeno nejprve jedno nenulové celé číslo δ_i a za ním jeden řetězec s_i tvořený nejvýše ℓ znaky anglické abecedy. Tento řádek znamená, že počet kusů zboží s_i ve skladu se změnil o δ_i.

Při návrhu algoritmů můžete předpokládat, že $\ell \leq 50$, že počet druhů zboží ve skladu nikdy nepřekročí 100 000 a že celkový počet kusů zboží ve skladu nikdy nepřekročí 10^{18}. Můžete také předpokládat, že nikdy nedostanete pokyn, který by počet kusů nějakého zboží ve skladu změnil na záporný.

Příklad

Vstup:	*Výstup pro podúlohu a:*	*Výstup pro podúlohu b:*
`+3 koberec`	`kusů: 3, typů: 1`	`koberec 3`
`+2 buldozer`	`kusů: 5, typů: 2`	`koberec 3`
`+1 buldozer`	`kusů: 6, typů: 2`	`buldozer 3`
`+7 zebrik`	`kusů: 13, typů: 3`	`zebrik 7`
`-3 buldozer`	`kusů: 10, typů: 2`	`zebrik 7`
`-5 zebrik`	`kusů: 5, typů: 2`	`koberec 3`

Hodnocení

Každá podúloha se hodnotí samostatně.

Ve svém řešení uveďte odhad, jak *nejdéle* (tj. v nejhorším možném případě) může každému z vašich programů trvat zpracování n-tého pokynu. (V tomto odhadu můžete použít proměnnou t pro aktuální počet druhů zboží ve skladu, proměnnou k pro aktuální počet kusů zboží ve skladu a proměnnou ℓ pro maximální délku názvu

zboží.) Tato časová složitost bude hlavním kritériem hodnocení. Snažte se tedy, aby váš program zpracoval co nejrychleji každý pokyn zadaný na vstupu.

Připomínáme, že pokud používáte ve svém programu netriviální algoritmy nebo datové struktury (např. různé součásti STL v C++), potom váš popis algoritmu musí obsahovat také dostatečný popis jejich implementace, případně popis implementace příbuzné datové struktury se stejnou časovou složitostí operací.

P-I-4 Zlomkové programy

V letošním ročníku olympiády se budeme v každém soutěžním kole setkávat se zlomkovými programy. Ve studijním textu uvedeném za zadáním této úlohy je popsáno, jak zlomkové programy fungují.

Soutěžní úloha

a) *(5 bodů)* Na vstupu je číslo n tvaru $2^x 3^y 5$, přičemž $x, y \geq 0$. Napište zlomkový program, který ho převede na číslo 5, jestliže $x = y$, resp. na číslo 7, jestliže $x \neq y$.

b) *(5 bodů)* Na vstupu je číslo n tvaru $2^x 3$, přičemž $x \geq 0$. Napište zlomkový program, který ho převede na číslo tvaru 2^y, kde $y = \lfloor x/2 \rfloor$.

Interpret

Dříve než svoje řešení odevzdáte, můžete si ho otestovat. Na webových stránkách olympiády (*http://mo.mff.cuni.cz/*) najdete odkaz na interpret zlomkových programů.

Studijní text

Zlomkové programy slouží k výpočtu některých funkcí na přirozených číslech. Zlomkový program má velmi jednoduchý zápis – je to konečná posloupnost zlomků, tedy kladných racionálních čísel $(z_1, \ldots, z_k)$.

Výpočet zlomkového programu probíhá po jednotlivých krocích. Během výpočtu si udržujeme jediné celé číslo a, tzv. *aktuální hodnotu.* Na začátku výpočtu se tato hodnota rovná hodnotě na vstupu. V každém kroku výpočtu najdeme nejmenší i takové, že $a \cdot z_i$ je celé číslo, a změníme aktuální hodnotu na $a \cdot z_i$. Pokud takové i neexistuje, výpočet končí.

Příklad 1: Ukážeme si program, který pro vstup $n = 2^x$ (kde $x \geq 0$) dává výstup 3^y, kde $y = x \bmod 3$.

Jedním takovým programem je posloupnost $(1/8, 9/4, 3/2)$. Slovně si popíšeme průběh výpočtu tohoto programu: Dokud to jde, zmenšuj x o 3. Když už to nejde, máme v a číslo 1, 2, nebo 4, a převedeme ho tedy snadno na 1, 3, nebo 9.

Například pro $n = 1024 = 2^{10}$ se hodnota a bude měnit takto:

$$2^{10} \xrightarrow{1} 2^7 \xrightarrow{1} 2^4 \xrightarrow{1} 2 \xrightarrow{3} 3.$$

(Číslo nad šipkou je pořadovým číslem zlomku, kterým jsme a v daném kroku násobili.)

Pro přesnost dodejme, že velmi záleží na pořadí jednotlivých zlomků. Například program $(9/4, 3/2, 1/8)$ by z čísla 2^x vyrobil číslo 3^x. Zlomek $1/8$ by se při výpočtech tohoto programu nikdy nepoužil.

Jiné vyhovující programy jsou $(1/8, 3/2)$, $(3/2, 1/27)$ a $(1/27, 3/2)$. Rozmyslete si, proč každý z nich řeší správně Příklad 1.

Příklad 2: Co spočítá program $(2/2)$ pro vstup $n = 4$? A co spočítá pro vstup $n = 7$?

Častou chybou je odpovědět, že pro vstup $n = 4$ se tento program zacyklí, ale pro vstup $n = 7$ se výpočet programu zastaví, neboť 7 není dělitelné dvěma. Správnou odpovědí ale je, že v *obou* případech poběží výpočet programu do nekonečna. Zajímají nás totiž jenom hodnoty zlomků, nikoli jejich zápis. Zlomek $2/2$ představuje racionální číslo 1, takže $7 \cdot (2/2) = 7 \cdot 1$ je celé číslo.

Abyste se ve svých řešeních takto nespletli, doporučujeme vám psát všechny zlomky v základním tvaru. Pro posloupnost zlomků zapsaných v základním tvaru platí, že v každém kroku výpočtu hledáme nejmenší i takové, že jmenovatel i-tého zlomku dělí beze zbytku aktuální hodnotu.

Příklad 3: Ukážeme si program, který pro vstup $n = 2^{x+1}$ (kde $x \geq 0$) dává výstup 3^{x+2}.

Číslo zadané na vstupu je určitě sudé a není dělitelné žádným prvočíslem různým od 2. Použijeme prvočíslo 11 jako příznak, že už nejsme na začátku výpočtu. V prvním kroku tedy přepíšeme číslo 2^{x+1} na $2^x 3^2 \cdot 11$. Toho dosáhneme zlomkem $3^2 \cdot 11/2 = 99/2$.

Jakmile se objeví v prvočíselném rozkladu aktuální hodnoty prvočíslo 11, znamená to, že první krok výpočtu máme úspěšně za sebou. Můžeme tedy dále klidně „měnit dvojky na trojky“. To můžeme zajistit například posloupností zlomků $(3 \cdot 13)/(2 \cdot 11)$ a $11/13$. Všimněte si, že nestačí použít jeden zlomek $33/22$. Není-li vám jasné proč, přečtěte si ještě jednou Příklad 2.

Postupně se takto dostaneme k číslu $3^{x+2}11$. V této situaci už stačí jenom vydělit aktuální hodnotu číslem 11 a můžeme skončit.

Musíme dát pozor na to, abychom ve zlomkovém programu výše popsané zlomky správně seřadili: $(39/22, 11/13, 1/11, 99/2)$. V prvním kroku výpočtu se jistě použije poslední zlomek. Od této chvíle je aktuální hodnota dělitelná číslem 11 nebo 13. Střídavě se proto používají první dva zlomky, dokud se nedostaneme do situace $a = 3^{x+2}11$. Pak už se první ani druhý zlomek použít nedá. Použije se proto třetí zlomek, čímž získáme hodnotu $a = 3^{x+2}$ a výpočet zjevně skončí.

Poznámka: V zápisu podobných řešení jako v příkladu 3 není nutné čitatele a jmenovatele zlomků roznásobovat. Své řešení můžete uvést ve tvaru

$$\left(\frac{3 \cdot 13}{2 \cdot 11}, \frac{11}{13}, \frac{1}{11}, \frac{3^2 \cdot 11}{2}\right).$$