62. ročník Matematické olympiády – 2012/2013

Úlohy krajského kola kategorie P

Krajské kolo 62. ročníku MO kategorie P se koná v úterý 22. 1. 2013 v dopoledních hodinách. Na řešení úloh máte 4 hodiny čistého času. V krajském kole MO-P se neřeší žádná praktická úloha, pro zajištění rovných podmínek řešitelů ve všech krajích je použití počítačů při soutěži zakázáno. Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (tzn. knihy, výpisy programů, kalkulačky, mobilní telefony).

Řešení každé úlohy vypracujte na samostatný list papíru.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), 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 vhodné odkazovat se na vaše řešení úloh domácího kola, opravovatelé je nemusí mít k dispozici; na autorská řešení se odkazovat můžete.
- **Zápis algoritmu.** V úlohách **P-II-1**, **P-II-2** a **P-II-3** 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-II-4** je nutnou součástí řešení zápis log-space programu.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. V zadání každé úlohy najdete přibližné limity na velikost vstupních dat. Efektivním vyřešením úlohy rozumíme to, že váš program spuštěný s takovými daty na současném běžném počítači dokončí výpočet během několika sekund.

Vzorová řešení úloh naleznete krátce po soutěži na webových stránkách olympiády *http://mo.mff.cuni.cz/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů postupujících do ústředního kola a také popis prostředí, v němž se budou v ústředním kole řešit praktické úlohy.

P-II-1 Cesta

Slovutný král Filip vyráží na cestu po svém království. S ohledem na státnické povinnosti si už vybral, která města a v jakém pořadí navštíví.

Výpravu ale komplikuje to, že král Filip striktně dodržuje dietu předepsanou svými královskými lékaři a astrology. Ti mu doporučili, kolik dní má cesta trvat, a předložili několik možných plánů, co má jíst každý den. Král Filip je gurmán a v každém městě chce jíst pouze místní specialitu (ta je v každém městě právě jedna, nicméně každá specialita může být k dostání i ve více městech). Nemusí se ale na jídlo zastavit v každém z měst. Pomozte mu rozhodnout, které z jeho dietních plánů lze realizovat.

Doprava mezi městy je velmi rychlá, není tedy žádné omezení na vzdálenost mezi městy, v nichž se král Filip zastaví na jídlo. Města ale musí navštívit v předem určeném pořadí, nemůže se nikdy vracet, ani dvakrát po sobě jíst ve stejném městě (aby se zástupci ostatních měst neurazili).

Popis vstupu

První řádek vstupu obsahuje přirozená čísla N, M, K a L, kde N udává počet měst ve Filipově cestě, M je počet druhů jídel, K je počet dietních plánů a L je jejich délka (počet jídel v každém z nich). Druhy jídel jsou očíslovány přirozenými čísly od 1 do M. Druhý řádek vstupu obsahuje N přirozených čísel popisujících speciality v jednotlivých městech v pořadí, v němž je král Filip během své cesty navštíví. Na i-tém z K následujících řádků je L přirozených čísel, popisujících i-tý dietní plán; j-té z těchto čísel udává jídlo, které Filip musí jíst při své j-té zastávce na cestě.

Popis výstupu

Výstup se skládá z K řádků. Na i-tý z nich vypište `ano`, lze-li během cesty realizovat i-tý dietní plán, a jinak `ne`. Dietní plán lze realizovat, jestliže vznikne z posloupnosti specialit odstraněním některých čísel.

Příklad

Vstup:	*Výstup:*
`10 4 4 3`	`ano`
`1 2 1 3 4 1 2 1 3 4`	`ne`
`1 4 2`	`ano`
`4 3 2`	`ne`
`1 1 1`	
`4 4 4`	

Hodnocení

Plných 10 bodů získáte za řešení, které úlohu efektivně vyřeší pro N i $K \cdot L$ řádově statisíce.

Až 7 bodů získáte za řešení, které přitom navíc bude předpokládat $M \leq 10$.

Až 4 body získáte za řešení, které úlohu efektivně vyřeší za předpokladu, že $K \cdot (N + L) \leq 100\,000$.

P-II-2 Vyvážená strava

Slovutný král Filip se rozhodl, že od nynějška bude jíst výhradně vyváženou stravu. Král je hrdým majitelem ovocného sadu, ve kterém rostou vzácné binární jabloně. Nařídil proto svým sadařům, aby v něm začali pěstovat pouze vyvážená jablka. Taková binární jabloň je velice specifický strom. Skládá se z uzlů, větví, listů a jablek. Z každého uzlu vedou směrem nahoru přesně dvě větve a každá větev končí buď v nějakém uzlu nebo v listu. Z listů už nevedou vzhůru žádné větve. Jabloň je navíc samozřejmě strom (tj. souvislý graf neobsahující kružnice). V každém listu může růst libovolný počet jablek.

Aby binární jablka byla bezvadně vyvážená, je potřeba, aby celá jabloň byla vyvážená. Takovou vyváženost zaručí jedině následující vlastnost, která musí být splněna pro každý uzel ve stromě: „Když se podíváme na obě větve, které z tohoto uzlu vedou směrem nahoru, a spočítáme všechna jablka v celých takto určených podstromech, tak nám musejí vyjít dva stejné počty.“ Sadaři chtějí samozřejmě co nejlépe vyhovět všem manýrům svého krále a zaručit mu co nejvíce vyvážených jablek. Rozhodli se proto co nejméně jablek otrhat tak, aby se stromy staly vyváženými.

Soutěžní úloha

Nedůvěřivý král by chtěl kontrolovat, jestli ho sadaři nepodvádějí. Žádá vás, abyste pro něj napsali program, který obdrží popis binární jabloně a odpoví, kolik nejméně jablek je potřeba ze stromu otrhat tak, aby zbylý strom byl vyvážený.

Popis vstupu

Pro jednoduchost nazývejme nadále uzly i listy jako vrcholy. První řádek obsahuje jediné přirozené číslo N, udávající počet vrcholů stromu. Vrcholy jsou číslované přirozenými čísly od 1 do N. Následujících N řádků popisuje vrcholy stromu, kde i-tý z nich se vztahuje k i-tému vrcholu stromu. Pokud řádek popisuje uzel, tak obsahuje znak `U` a dvě přirozená čísla oddělená mezerami. Tato čísla vyjadřují, do kterých vrcholů vedou větve nahoru z tohoto uzlu. Řádek popisující list začíná znakem `L` a za mezerou následuje jedno kladné celé číslo, které říká, kolik jablek v tomto listu roste. Můžete přepokládat, že váš programovací jazyk umí pracovat s čísly řádově tak velkými, jako je celkový počet jablek, a provádět s nimi základní aritmetické operace v konstantním čase.

Popis výstupu

Program vypíše jediné číslo znamenající nejmenší počet jablek, který je potřeba ze stromu otrhat tak, aby byl zbylý strom vyvážený. Všimněte si, že řešení vždycky existuje, protože když otrháme všechna jablka, tak je strom podle definice vyvážený.

Příklady

Vstup:	*Výstup:*

`3`
`L 3`
`U 1 3`
`L 5`
	`2`

Vstup:

```
5
U 2 3
L 1
U 4 5
L 2
L 3
```

Výstup:

```
6
```

Hodnocení

Plných 10 bodů získáte za řešení, které zvládne efektivně vyřešit vstup s $N \leq 1\,000\,000$. Až 6 bodů získáte za řešení, které zvládne efektivně vyřešit vstup s $N \leq 1\,000$ za předpokladu, že strom obsahuje celkem nejvýše 1 000 jablek.

P-II-3 Zaokrouhlování

Princ Honzík, syn slovutného krále Filipa, si rád hraje s maticemi, a tak se nelze divit, že jednu pěknou matici dostal jako dárek k narozeninám. Bohužel ale Honzíkovi rodiče nevěděli, že má raději celá čísla než desetinná, takže při výběru dárku se nijak neomezovali a vybrali matici plnou desetinných čísel. Co víc, v darované matici nebylo vůbec žádné celé číslo! Honzík ví, že darovanému koni se na zuby nehledí, přesto ho ale trochu mrzelo, že z matice nemá takovou radost, jakou by mohl. Všiml si ale jedné věci – součet čísel v každém řádku i v každém sloupci darované matice byl vždy celé číslo. A tak ho napadlo, že by si matici mohl zkusit upravit tak, aby byla celočíselná a přitom součet jednotlivých řádků i sloupců zůstal stejný jako v původní matici. A protože ji chce změnit co nejméně, jedinou operaci, kterou chce při úpravě použít, je zaokrouhlení desetinného čísla směrem nahoru nebo směrem dolů.

Dlouho si s tím ale nedokázal poradit, a proto požádal o pomoc vás.

Soutěžní úloha

Napište program, který dostane matici s M řádky a N sloupci tvořenou kladnými necelými desetinnými čísly. Pro tuto matici platí, že součet čísel v každém řádku a v každém sloupci je vždy celé číslo. Váš program pro každé číslo v matici určí, zda se má zaokrouhlit nahoru nebo dolů, tak aby se součet jednotlivých řádků ani součet jednotlivých sloupců nezměnil. Případně určí, že úloha nemá řešení.

Popis vstupu

První řádek vstupu obsahuje dvě celá čísla M a N, která určují rozměry matice. Každý z následujících M řádků obsahuje právě N kladných necelých desetinných čísel, s přesností na 2 desetinná mista, která určují jednotlivé prvky matice.

Popis výstupu

Pokud úloha řešení má, bude výstup obsahovat právě M řádků, přičemž každý řádek bude obsahovat N znaků. Znak `N` říká, že se má odpovídající číslo v matici zaokrouhlit směrem nahoru, a znak `D`, že se má zaokrouhlit směrem dolů. Pokud řešení neexistuje, bude výstup tvořen jedním řádkem, který obsahuje text `NEEXISTUJE`.

Příklad

Vstup:	*Výstup:*
`3 4`	`DDND`
`5.31 2.39 7.13 0.17`	`NNDD`
`3.33 4.43 1.92 2.32`	`DDNN`
`10.36 1.18 6.95 4.51`	

Hodnocení

Plných 10 bodů získáte za řešení, které zvládne efektivně vyřešit vstup s $N \cdot M \leq 1\,000\,000$. Až 6 bodů získáte za řešení, které zvládne efektivně vyřešit vstup s $N \cdot M \leq 1\,000$. Až 3 body získáte za řešení, které zvládne efektivně vyřešit vstup s $N \cdot M \leq 20$.

P-II-4 Log-space programy

V této úloze budeme pracovat s programy s logaritmickou prostorovou složitostí neboli log-space programy. Ve studijním textu uvedeném za zadáním úlohy je popsáno, jak takové programy fungují. Studijní text je identický s textem z domácího kola. Připomeňme, že v této úloze nebudeme hodnotit časovou složitost vašich řešení, nemusíte ji proto ani určovat.

Soutěžní úloha

a) *(4 body)* Napište log-space program, který vynásobí dvě dlouhá čísla. Vstupem jsou pole `A[1..n]` a `B[1..n]`, kde `A[i]` resp. `B[i]` je i-tá číslice od konce v desítkovém zápisu zadaného čísla. Výsledek násobení, tedy číslo

$$\left(\sum_{i=1}^{n} A[i] \cdot 10^{i-1}\right) \cdot \left(\sum_{i=1}^{n} B[i] \cdot 10^{i-1}\right),$$

zapište ve stejném formátu do výstupního pole `C[1..(2*n)]`.

Příklad: Je-li $n = 2$, $A[1] = 3$, $A[2] = 1$, $B[1] = 2$, $B[2] = 1$, pak chceme vynásobit $13 \cdot 12 = 156$, do výstupního pole tedy zapíšeme $C[1] = 6$, $C[2] = 5$, $C[3] = 1$ a $C[4] = 0$.

b) *(6 bodů)* *Les* je neorientovaný graf bez cyklů, tedy takový graf, že každá jeho komponenta souvislosti je strom. Napište log-space program, který rozhodne, zda dva vrcholy leží ve stejné komponentě souvislosti zadaného lesa, tedy zda mezi nimi existuje cesta po jeho hranách. Vstup tvoří čtyři celočíselné proměnné n, m, u, v (kde $0 \leq m \leq n-1$ a $1 \leq u < v \leq n$) a dvě pole `A[1..m]` a `B[1..m]`. Zde n udává počet vrcholů grafu a m počet jeho hran. Vrcholy jsou číslovány od 1 do n. Pole A a B popisují hrany lesa – pro každé $i = 1, \ldots, m$ jsou vrcholy $A[i]$ a $B[i]$ spojeny hranou. Výstup tvoří celočíselná proměnná s, do které přiřaďte 1, jestliže mezi vrcholy u a v existuje v zadaném lesu cesta, a 0 jinak.

Příklad: Nechť $n = 5$, $m = 3$, $A[1] = 1$, $B[1] = 2$, $A[2] = 4$, $B[2] = 3$, $A[3] = 4$, $B[3] = 5$. Jestliže $u = 3$ a $v = 5$, pak výsledek je $s = 1$, neboť mezi vrcholy 3 a 5 vede cesta přes vrchol 4. Jestliže $u = 3$ a $v = 1$, pak výsledek je $s = 0$.

Studijní text

Známe-li více algoritmů řešících tutéž úlohu, obvykle považujeme za lepší ten, který má menší časovou složitost. Prostorovou složitost používáme až jako vedlejší kriterium, alespoň dokud nejsou paměťové nároky programu absurdně vysoké. Zkusme tentokrát pořadí důležitosti obrátit a zajímat se především o prostorové nároky algoritmu.

Budeme psát *log-space programy.* Tak budeme říkat obyčejným programům zapsaným v Pascalu či Céčku, které splňují následující podmínky:

- Každá proměnná v programu je *celočíselného typu* (`int`, `integer`, apod.) a její hodnota je *polynomiální ve velikosti vstupu.* Tím myslíme, že absolutní hodnota čísla nepřesáhne $c \cdot n^k$, kde n je velikost vstupu programu (počet čísel na vstupu) a c a k nějaké konstanty nezávislé na vstupu. Můžeme tedy používat například hodnoty z rozsahu $-n \ldots n$, $-3n^5 \ldots 3n^5$ nebo $-10 \ldots 10$, ale už ne $0 \ldots 2^n$.
- Pascalské typy `char` a `boolean` budeme také považovat za celočíselné.
- Žádné další typy (např. pole a ukazatele) nejsou povoleny.
- Jedinou výjimku tvoří *vstup a výstup.* Vstup programu bude dostupný v určených proměnných (obvykle polích), ze kterých smí program pouze číst. Podobně výstup uložíme do smluvených proměnných a máme do nich povoleno pouze zapisovat (speciálně tedy nesmíme výstupní proměnnou zvýšit o 1 pomocí `++` nebo `inc`, protože tyto operace kombinují zápis se čtením). Čísla na vstupu budou vždy polynomiálně velká, aby bylo možné ukládat je do pracovních proměnných.
- Program nepoužívá rekurzi.

Příklad 1: Maximum z pole čísel můžeme hledat následujícím log-space programem:

```
var n: integer;                          { vstup: počet čísel }
    A: array [1..n] of integer;          { vstup: zadaná čísla }
    m: integer;                          { výstup: pozice maxima }
    i, j: integer;                       { pracovní proměnné }
begin
   j := 1;
   for i := 2 to n do
      if A[i] > A[j] then
         j := i;
   m := j;
end;
```

Proměnné `i` a `j` se pohybují v rozsahu $1 \ldots n$, ostatní proměnné jsou vstupní, popřípadě výstupní. Požadavky definice log-space programu jsou tedy splněny.

Příklad 2: Následující log-space program nalezne v poli číslo, které se tam vyskytuje nejčastěji:

```
var n: integer;                          { vstup: počet čísel }
    A: array [1..n] of integer;          { vstup: zadaná čísla }
```

```
    m: integer;                          { výstup: pozice nejčetnějšího }
    i, j, c, cmax: integer;              { pracovní proměnné }
begin
   cmax := 0;
   for i := 1 to n do
       begin
          { Spočítáme, kolikrát se vyskytuje A[i] }
          c := 0;
          for j := 1 to n do
             if A[j] = A[i] then
                c := c+1;
          { Je to víc než dosavadní maximum? }
          if c > cmax then
             begin
                cmax := c;
                m := i;
             end;
       end;
end;
```

Opět si snadno rozmyslíme, že hodnoty pracovních proměnných `i`, `j`, `c` a `cmax` nepřesáhnou n.

Motivace

Jistě vás napadne otázka, proč jsme zavedli log-space programy právě takto. Jsou totiž věrným modelem algoritmů s logaritmickým množstvím pracovní paměti, tedy s pamětí $\mathcal{O}(\log n)$, kde n je velikost vstupu. *Prostorovou složitost* programů ovšem měříme přesněji, než je v olympiádě obvyklé, totiž v *bitech.*

Jeden bit paměti nabývá 2 možných stavů, pomocí 2 bitů můžeme rozlišit 4 různé stavy a obecně pomocí k bitů 2^k stavů. Například 8-bitová proměnná tedy může nabývat $2^8 = 256$ různých hodnot, takže do ní můžeme ukládat čísla v rozsahu $0 \ldots 255$, nebo třeba $-100 \ldots 155$ – počátek intervalu si můžeme zvolit libovolně. A obráceně: pokud proměnná v programu nabývá hodnot $1 \ldots k$, potřebujeme na její uložení $\lceil \log_2 k \rceil$ bitů paměti; pro rozsah $j \ldots k$ je to $\lceil \log_2 (k - j + 1) \rceil$ bitů. Jelikož $\log_2 n^k = k \log_2 n$, polynomiálně velká čísla jsou přesně ta, která se vejdou do logaritmického množství paměti.

Pole by se teoreticky do logaritmického prostoru mohlo vejít, ale bylo by potřeba, aby součet velikostí všech položek byl logaritmický. To splňuje například pole $\log n$ čísel konstantního rozsahu, nebo naopak konstatní počet položek logaritmické velikosti. Možnosti jsou tedy značně omezené, a proto jsme pro jednoduchost pole v našich log-space programech vůbec nepovolili.

Ještě nesmíme zapomenout na to, že paměť potřebujeme i k *volání podprogramů* (procedur a funkcí). Musíme si totiž uložit, kam se z podprogramu vrátit (k tomu stačí rozlišit konstantně mnoho možností), a zapamatovat si lokální proměnné podprogramu. Pokud program není rekurzivní, nebudou paměťové nároky vyšší, než kdyby všechny proměnné byly globální. Pokud bychom ovšem rekurzi použili, museli bychom vzít v úvahu, že jedna proměnná může současně existovat ve více instancích, takže se množství paměti vynásobí hloubkou rekurze. Rekurze by

se tedy vešla do logaritmické paměti jen ve speciálních případech, takže ji v zájmu jednoduchosti také nepovolujeme.

Na závěr si všimněme, že k řešení obou našich příkladů by menší než logaritmické množství paměti nestačilo. Jelikož vstup je zadán v poli, potřebujeme umět indexovat prvky tohoto pole, tedy vytvářet indexy v rozsahu $1 \ldots n$, a k uložení těchto indexů je logaritmický prostor potřeba.