62. ročník Matematické olympiády – 2012/2013

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* (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 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 zápis log-space programu.

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 podle jejich časové složitosti, tzn. podle závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. V zadání každé úlohy najdete limity na velikost vstupních dat. Můžete je využít k odhadu, jak dobré je vaše řešení. Na počítači pracujícím rychlostí miliarda instrukcí za sekundu dokončí vzorové řešení výpočet s libovolnými povolenými vstupními daty nejvýše za několik sekund.

P-III-1 Vodovodní síť

V zemi za devatero horami a devatero řekami úspěšně zavedli elektřinu a nyní by chtěli napojit všechny domácnosti na zdroj kvalitní pitné vody. Ve vytipovaných městech postavili vodárny a nyní hodlají vybudovat vodovod mezi městy.

Poučena ze zmatků, které nastaly při budování elektrické sítě, vrchní plánovací komise nejprve všech N měst v zemi očíslovala od 1 do N a nechala vyprojektovat vodovod, který by spojil i-té a $(i+1)$-ní město, kde $i = 1, \ldots, N-1$. Když však sečetli náklady, zjistili, že na postavení všech těchto vodovodů státní pokladna nestačí.

Komise se na vás proto obrátila s prosbou, abyste navrhli, které z vodovodů vynechat, ovšem tak, aby bylo stále zajištěno zásobování všech měst pitnou vodou.

Soutěžní úloha

Je dáno N měst, která jsou očíslována od 1 do N. U měst, ve kterých stojí vodárna, je dáno, kolik vody může tato vodárna vyprodukovat navíc oproti spotřebě daného města. U ostatních měst je pak zadáno, jaká je jejich spotřeba vody. Pro každou dvojici měst s čísly i a $i+1$, kde $i = 1, \ldots, N-1$, je určeno, kolik stojí vybudovat vodovod mezi těmito městy.

Vaším úkolem je nalézt množinu vodovodů, které zajistí dostatečné zásobování všech měst vodou a zároveň cena na jejich vybudování bude co nejmenší. Vámi nalezená množina vodovodů tedy musí splňovat následující: pokud nebude postaven vodovod mezi městy $i-1$ a i a mezi městy j a $j+1$ ($1 < i \leq j < N$), pak množství vody, kterou vyprodukují vodárny ve městech s čísly i až j, je alespoň součet spotřeby vody v těchto městech.

Popis vstupu

První řádek vstupu obsahuje jedno celé číslo N, které udává počet měst. Dalších $N-1$ řádků obsahuje každý dvě celá čísla a_i a b_i, $i = 1, \ldots, N-1$. Poslední řádek obsahuje jedno celé číslo a_N. Pokud $a_i \geq 0$, pak vodárna v i-tém městě může vyprodukovat a_i jednotek vody navíc oproti spotřebě tohoto města. Pokud $a_i < 0$, pak v i-tém městě není vodárna a jeho spotřeba vody je $-a_i$. Cena vybudování vodovodu mezi i-tým a $(i+1)$-ním městem je b_i, vždy platí $b_i \geq 1$.

Můžete předpokládat, že platí $1 \leq N \leq 1\,000\,000$, $\sum_{i=1}^{N} |a_i| \leq 2\,000\,000\,000$, $\sum_{i=1}^{N} a_i \geq 0$ a $\sum_{i=1}^{N-1} b_i \leq 2\,000\,000\,000$.

Povšimněte si, že podmínka $\sum_{i=1}^{N} a_i \geq 0$ garantuje existenci řešení (při propojení všech měst vodovody).

Popis výstupu

Výstup obsahuje jeden řádek s jedním celým číslem, které udává nejmenší možnou cenu, kterou je třeba zaplatit za vybudování vodovodů tak, aby byly splněny podmínky uvedené v soutěžní úloze.

Hodnocení

Až 2 body získá řešení, které dokáže úlohu efektivně vyřešit pro $N = 20$. Až 6 bodů získá řešení, které dokáže úlohu efektivně vyřešit pro $N = 1\,000$. Plných 10 bodů získá řešení, které dokáže úlohu efektivně vyřešit pro $N = 1\,000\,000$.

Příklad

Vstup:	*Výstup:*
5 -2 2 10 2 -2 3 -5 4 5	7

Nejlevněji lze vybudovat vodovodní síť tak, že se propojí první až čtvrté město; páté město zůstane izolováno.

P-III-2 Vdavky

Král Chytrolín II. se rozhodl provdat svou dceru. Protože ale nechtěl vypadat jako jeho otec Hamoun IV., řekl, že ji nedá nejbohatšímu princi, nýbrž K-tému nejbohatšímu princi. Princezna Nádhernína byla vyhlášená svou krásou, a tak se sešly obrovské zástupy zájemců a zbývalo jen vybrat vhodného prince.

Jelikož má král k dispozici N rádců, rozdělil prince do N skupin, každému rádci přidělil jednu skupinu a přikázal mu, ať všechny prince ve své skupině seřadí sestupně podle jejich bohatství. Bohužel ale zjistil, že i když má prince rozdělené a seřazené, stále není jednoduché vybrat K-tého nejbohatšího. Chce dostát svému jménu, takže potřebuje ukázat, že jeho rozhodnutí rozdělit a seřadit prince bylo nadmíru chytré a že najít ženicha pro svou dceru je už jenom maličkost. Bohužel jméno je jenom jméno, a proto mu nezbylo nic jiného, než se obrátit na vás, samozřejmě pod přísahou naprosté diskrétnosti.

Soutěžní úloha

Navrhněte algoritmus, který králi pomůže po rozdělení princů do skupin a seřazení jednotlivých skupin sestupně podle bohatství najít K-tého nejbohatšího prince, tj. takového, že právě $K-1$ princů je bohatších (předpokládejte, že žádní dva princové nejsou stejně bohatí). Protože princů je opravdu hodně a jejich majetek je nevyjádřitelný čísly, je možné se pouze dotazovat rádců, zda je bohatší princ A nebo B.

Popis vstupu

Vstupem pro váš algoritmus jsou dvě čísla N a K, kde N je počet posloupností a K označuje, kolikátého nejbohatšího nápadníka hledáme. Dále máte k dispozici funkci

```
int bohatsi(int s1, int i1, int s2, int i2);
```

případně

```
function bohatsi(s1, i1, s2, i2: integer): boolean;
```

simulující rádce. Tato funkce vrací `1`, resp. `true`, právě když je princ na pozici `i1` ve skupině `s1` bohatší než princ na pozici `i2` ve skupině `s2`. Jednotlivé skupiny i princové v jednotlivých skupinách jsou číslováni vždy od 1. V každé skupině je alespoň K princů (jelikož jsou v rámci skupin seřazeni sestupně dle bohatství, princové na pozicích $K+1$ a vyšších nejsou pro výsledek úlohy důležití).

Popis výstupu

Program by měl vypsat dvě čísla s a i, která určují skupinu s a pozici i, na které se nachází K-tý nejbohatší princ.

Hodnocení

Až 2 body získá řešení, které dokáže úlohu efektivně vyřešit pro $N = 2$ a $K = 100\,000$. Až 6 bodů získá řešení, které dokáže úlohu efektivně vyřešit pro $N = 10\,000$ a $K = 1\,000\,000$. Plných 10 bodů získá řešení, které dokáže úlohu efektivně vyřešit pro $N = 10\,000$ a $K = 1\,000\,000\,000$.

Příklad

Mějme následující skupiny princů (pro názornost uvádíme i jejich bohatství vyjádřené penězi):

1. skupina: 20 Kč, 15 Kč, 10 Kč
2. skupina: 18 Kč, 16 Kč, 8 Kč
3. skupina: 14 Kč, 7 Kč, 4 Kč

Budeme hledat 3. nejbohatšího prince, tedy $N = 3$ a $K = 3$.

Pokud bychom zavolali např. `bohatsi(1,2,2,3)`, dostali bychom `1`, resp. `true` (15 Kč > 8 Kč), pokud `bohatsi(3,3,2,1)` dostali bychom `0`, resp. `false` (4 Kč < 18 Kč).

Správný výsledek je $s = 2$ a $i = 2$ (16 Kč).

P-III-3 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 i krajského kola. Připomeňme, že v této úloze nebudeme hodnotit časovou složitost vašich řešení, nemusíte ji proto ani určovat.

Soutěžní úloha

a) *(4 body) Permutace* n čísel je posloupnost $P[1], \ldots, P[n]$, v níž se každé z čísel $1, \ldots, n$ vyskytuje právě jednou. Na takovou permutaci můžeme pohlížet jako na orientovaný graf, jehož vrcholy jsou čísla $1, \ldots, n$ a hrana vede vždy z vrcholu i do $P[i]$.

To znamená, že každému vrcholu přiřadíme právě jednu hranu vedoucí dovnitř a právě jednu vedoucí ven. Takové grafy jsou tvořené disjunktními *cykly,* přičemž za cyklus počítáme i hranu, která začíná i končí v tomtéž vrcholu.

Napište log-space program, který pro zadanou permutaci zjistí, kolika cykly je tvořena. Vstupem je pole `P[1..n]` reprezentující permutaci, výstupem je jediná celočíselná proměnná `c` – počet cyklů permutace `P`.

Příklad: Nechť $n = 6$ a $P[1 \ldots 6] = (3, 6, 5, 4, 1, 2)$. Tato permutace je tvořena cykly $(1, 3, 5)$, $(2, 6)$ a (4), takže výsledkem programu je $c = 3$.

b) *(6 bodů) Strom* je souvislý neorientovaný graf bez cyklů. Napište log-space program, který pro zadaný strom a dva jeho vrcholy u a v vypočítá *vzdálenost* z u do v, tedy počet hran na nejkratší cestě stromem z u do v.

Vstup tvoří tři celočíselné proměnné n, u, v (kde $n \geq 0$, $1 \leq u < v \leq n$) a dvě pole `A[1..n-1]` a `B[1..n-1]`. Zde n udává počet vrcholů stromu. Vrcholy jsou číslovány od 1 do n. Pole A a B popisují hrany stromu – pro každé $i = 1, \ldots, n-1$ jsou vrcholy $A[i]$ a $B[i]$ spojeny hranou. Výstup tvoří celočíselná proměnná d, do které přiřaďte vypočtenou vzdálenost mezi u a v.

Příklad: Nechť $n = 5$, $A[1 \ldots 4] = (1, 4, 3, 3)$, $B[1 \ldots 4] = (4, 2, 4, 5)$. Jestliže $u = 3$ a $v = 5$, pak výsledek je $d = 1$, neboť mezi vrcholy 3 a 5 vede hrana. Jestliže $u = 5$ a $v = 1$, pak výsledek je $d = 3$, neboť z vrcholu 5 do vrcholu 1 vede nejkratší cesta přes vrcholy 3 a 4.

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.