58. ročník Matematické olympiády – 2008/2009

Úlohy ústředního kola kategorie P – 1. soutěžní den

Na řešení úloh máte 4,5 hodiny čistého času.

Řešení každého příkladu musí obsahovat:

- **Popis řešení,** to znamená slovní popis použitého 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.
- **Program.** V úlohách **P-III-1** a **P-III-2** je třeba uvést dostatečně podrobný zápis algoritmu, např. ve tvaru pseudokódu nebo zdrojového textu nejdůležitějších částí programu v jazyce Pascal nebo C. Ze zápisu můžete vynechat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů apod. V řešení úlohy **P-III-3** je nutnou součástí řešení program pro zásobníkový počítač.

Hodnotí se nejen správnost programu, ale také kvalita popisu řešení a efektivita zvoleného algoritmu.

P-III-1 Lesník Jehlička II

V krajském kole pan Jehlička s vaší pomocí a s hrůzou zjistil, že z jeho kdysi krásného a velikého lesa zbývá jen drobný háj. Aby zabránil jeho dalšímu zmenšování, najal tlupu strážných trollů. Trollové jsou známí pro svou sílu, méně pak již pro svou pronikavou inteligenci. Pan Jehlička se proto rozhodl příkazy pro trolly co možná nejvíce zjednodušit. Každý troll má přiděleny dva body a mezi nimi pochoduje po rovné čáře.

Během prvního dne bylo však nutné ošetřit takřka všechny trolly s drobnými zraněními. Pan Jehlička pozapomněl, že se úsečky, po nichž trollové pochodují, mohou protínat, a ve svých instrukcích nezmínil nutnost vyhýbat se srážkám s ostatními trolly. Pokus přidat trollům tento příkaz narazil na jejich zapomnětlivost. Po několika opakováních obtížného příkazu „na konci úsečky se otoč“ trollům přetekl zásobník a příkaz „vyhýbej se srážkám“ se ztratil, s neblahými důsledky pro rozpočet ošetřovny.

Pan Jehlička se proto rozhodl všechna křížení označit dopravní značkou „Pozor, troll!“. Vaším úkolem je zjistit, kam tyto značky umístit.

Soutěžní úloha:

Úsečka, po níž se i-tý troll pohybuje, je zadána dvojicí jejích krajních bodu se souřadnicemi (a_i, b_i), (c_i, d_i), kde a_i, b_i, c_i a d_i jsou celá čísla. Krajní body úsečky

neleží na žádné jiné úsečce. Úlohou je vypsat souřadnice průsečíků těchto úseček. Každý průsečík vypište pouze jednou, i když by se v něm protínalo tři nebo více úseček. Předpokládejte, že průsečíků je málo – podstatně méně než N^2.

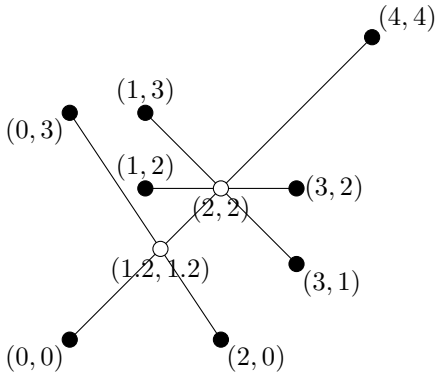
Formát vstupu:

První řádek obsahuje přirozené číslo N, udávající počet trollů, $0 \leq N \leq 10\,000$. Na následujících N řádcích jsou popsány úsečky, po nichž se trollové pohybují. Na i-tém řádku se nachází čtveřice celých čísel a_i, b_i, c_i a d_i, souřadnice krajních bodů (a_i, b_i) a (c_i, d_i) úsečky.

Formát výstupu:

Na každý řádek výstupu vypište souřadnice jednoho z průsečíků zadaných úseček. Pořadí průsečíků může být libovolné.

Příklady:



Vstup:

```
4
0 0 4 4
0 3 2 0
1 3 3 1
1 2 3 2
```

Výstup:

```
1.2 1.2
2 2
```

P-III-2 Kouzelník Pecivális

Při poklesu zájmu o jeho magická představení v důsledku hospodářské krize se starý kouzelník Pecivális rozhodl využít nově nabytého času k procestování vzdálených koutů Země. Klesající tržby však mají ten nepříjemný důsledek, že si nemůže dovolit konvenční způsoby přepravy a nezbývá než se spolehnout na (ne)osvědčené metody předků ...

Během pátrání v zapomenutých koutech svého obydlí měl Pecivális štěstí a podařilo se mu najít potřebné svazky zaklínadel. Jejich použití však není úplně jednoduché – podle nejlepších soudobých poznatků v oblasti cestovní magie (které se za posledních pár set let příliš nezměnily) může sice začít i skončit zaříkání na libovolném místě v knize, není však záhodno mezitím nějaký text přeskočit a navíc musí magickými slovy vyvolaná energie dosáhnou některé z potřebných hladin pro kýžený cíl přesunu. Čím větší bude počet přečtených slov, tím větší je pak šance úspěšného zakončení. Jelikož však není starý mág v této oblasti čarování příliš zběhlý, bude potřebovat vaši pomoc.

Soutěžní úloha:

Mějme posloupnost slov, z nichž každé je ohodnoceno celým číslem, které představuje jeho energii. Hledaná energetická hladina je reprezentována (všemi) násobky daného přirozeného čísla K. Vaším úkolem je na základě těchto informací najít nejvhodnější začátek a konec zaříkání, to jest nejdelší souvislý úsek slov, jejichž energie budou v součtu násobkem čísla K. Předpokládejte, že číslo K je obvykle řádově menší než počet slov v knize zaříkadel.

Formát vstupu:

Na prvním řádku vstupu jsou dvě přirozená čísla N a K, $1 \leq N \leq 1\,000\,000$ a $1 \leq K \leq 50\,000$, oddělená mezerami; číslo N udává počet slov v knize zaříkadel. Jak již bylo řečeno, K je obvykle mnohem menší než N. Na druhém řádku je pak mezerami oddělený seznam N nezáporných celých čísel $a_1, \ldots, a_n$, $0 \leq a_i \leq 1\,000\,000\,000$, která představují energie jednotlivých slov v knize.

Formát výstupu:

Program vypíše dvě čísla i a j ($1 \leq i \leq j \leq N$), přičemž součet $a_i + a_{i+1} + \ldots + a_j$ musí být násobkem čísla K a rozdíl $j - i$ největší možný mezi všemi dvojicemi i a j, pro něž je součet $a_i + a_{i+1} + \ldots + a_j$ násobkem K. Je-li možných dvojic i a j více, můžete vypsat libovolnou z nich. Pokud naopak žádná taková dvojice neexistuje, vypište „`Nelze zaklínat.`“.

Příklad 1:

Vstup:

```
8 5
1 2 8 6 3 4 4 9
```

Výstup:

```
3 7
```

Příklad 2:

Vstup:

```
5 8
1 1 1 1 1
```

Výstup:

```
Nelze zaklínat.
```

P-III-3 Stackal

Studijní text, který je stejný jako v domácím a krajském kole, následuje po zadání soutěžní úlohy.

Soutěžní úloha:

Na vstupu je zadán řetězec obsahující pouze znaky '`a`', '`b`', '`c`' a '`d`'. Napište program pro zásobníkový počítač, který zjistí, zda má v tomto řetězci každý ze znaků '`a`' až '`d`' stejný počet výskytů. Pokud ano, program na výstup zapíše jedničku, jinak nulu.

Zaměřte se na použití co nejmenšího počtu zásobníků, byť za cenu vyšší časové složitosti.

Příklad:

Na vstupy '`abcddcba`' a '`aaabbbcccddd`' je správná odpověď '`1`'.

Na vstup '`badbadc`' je správná odpověď '`0`' (znaky '`a`', '`b`' a '`d`' se vyskytují dvakrát, ale znak '`c`' pouze jednou).

Studijní text:

V letošním ročníku olympiády se budeme setkávat se *zásobníkovými počítači.* To jsou výpočetní stroje, jejichž paměť je tvořena několika *zásobníky.* Každý zásobník obsahuje posloupnost *hodnot* a umí s nimi provádět tyto tři operace: přidat hodnotu na konec posloupnosti (uložit ji do zásobníku), odebrat hodnotu z konce posloupnosti (vybrat ji ze zásobníku) a konečně zjistit, zda v zásobníku ještě nějaké hodnoty jsou. Mimo to má náš počítač ještě pevný počet obyčejných proměnných. Hodnoty uložené v zásobnících i v proměnných musí mít pevný rozsah *nezávislý na velikosti vstupu.*

Zásobníkové počítače budeme programovat v jazyku Stackal. To je jazyk podobný Pascalu, ovšem upravený podle možností našich strojů. V následujících odstavcích popíšeme, v čem se od klasického Pascalu liší.

Proměnné ve Stackalu mohou být pouze těchto typů: `boolean` (logický typ, může nabývat hodnot `true` a `false`), `char` (znak z nějaké konečné množiny znaků, které budeme říkat abeceda), $a..b$ (celá čísla z intervalu od a do b; jak a, tak b musí být nezáporné konstanty menší než 100) a `stack of` t, což je zásobník hodnot typu t (jiného než `stack`). Počáteční hodnoty proměnných při spuštění programu nejsou definovány, výjimku tvoří zásobníky, které jsou na počátku vždy prázdné.

Vstup a výstup programu jsou vždy posloupnosti znaků (neboli řetězce). Funkce `read`(c) přečte další znak ze vstupu, uloží ho do proměnné c a vrátí `true`. Pokud by již na vstupu žádné další znaky nebyly, vrátí `false` a proměnnou c nezmění. Na výstup se zapisuje příkazem `write`(x), kde x je buďto znak nebo proměnná typu `char`. Ve vstupu ani výstupu se není možné vracet ani znaky přeskakovat.

Zásobníky je možno ovládat pomocí speciálních příkazů a funkcí: Je-li S zásobník, pak příkaz `push`(S, x) uloží do S hodnotu x (hodnota samozřejmě musí být správného typu), funkce `pop`(S) vybere hodnotu ze zásobníku a vrátí ji jako svůj výsledek a booleovská funkce `empty`(S) vrátí `true`, pokud je zásobník S prázdný, jinak `false`. Funkci `pop` je také možné volat jako proceduru, pokud nás odebraná

hodnota nezajímá. Použití funkce `pop` na prázdný zásobník není povoleno a způsobí zastavení programu s běhovou chybou. Žádným jiným způsobem nelze se zásobníky manipulovat.

Příkazy Pascalu máme k dispozici všechny, jediným omezením je, že nesmíme používat přiřazovací příkaz `:=` na zásobníky. Také můžeme v programu definovat procedury a funkce, není ovšem dovoleno používat rekurzi a zásobníky mohou být použity jako parametry pouze tehdy, jsou-li předávány odkazem.

Časovou a paměťovou složitost programů definujeme obdobně jako na normálním počítači. Čas budeme měřit počtem provedených příkazů, paměť největším počtem hodnot, které si program pamatuje v jednom okamžiku ve svých proměnných a všech zásobnících. Často se budeme snažit o to, aby program používal co nejmenší počet zásobníků, byť by kvůli tomu byl pomalejší.

Příklad: Napište program, který zjistí, zda se v zadaném řetězci vyskytuje stejný počet znaků '`a`' jako znaků '`b`' a podle toho vypíše buď '`1`' (když to je pravda) nebo '`0`' (když ne).

Řešení: Jelikož hodnoty proměnných jsou omezené stovkou, nemůžeme si jednoduše počítat výskyty znaků v celočíselné proměnné. Místo toho využijeme dva zásobníky: do jednoho budeme ukládat `a`-čka, do druhého `b`-čka. Až vstup skončí, budeme vybírat znaky vždy z obou zásobníků současně a odpovíme `1` právě tehdy, když se oba současně vyprázdnily.

```
program rovnost;
var a, b: stack of char;              { dva zásobníky na znaky }
    c: char;                          { právě zpracovávaný znak }
begin
   while read(c) do begin             { čteme ze vstupu, dokud to jde }
      if c='a' then push(a, c);       { znak uložíme do správného zásobníku }
      if c='b' then push(b, c);
      end;
   while not empty(a) and not empty(b) do begin
      pop(a);                         { odebíráme znaky z obou zás. současně }
      pop(b);
      end;
   if empty(a) and empty(b) then      { vyšly oba prázdné? }
      write('1')
   else
      write('0');
end;
```

Tento program má lineární časovou i paměťovou složitost a potřebuje dva zásobníky.

Druhé řešení: Ukážeme si, jak jeden zásobník ušetřit a stále zachovat lineární časovou složitost. Místo jednotlivých počtů znaků budeme do zásobníku ukládat, o kolik víc jsme viděli `a`-ček než `b`-ček. Pokud je tento rozdíl kladný (`a`-ček je více), zapamatujeme si příslušný počet znaků '`+`'. Záporné rozdíly budeme ukládat pomocí znaků '`-`'.

Rozmysleme si tedy, co se stane, když program přečte znak ‘a’. Tehdy by měl k rozdílu přičíst jedničku. Proto zkontroluje, jaká hodnota se nachází na vrcholu zásobníku – to je hodnota, kterou by přečetl následující pop. Pokud to je ‘-’, tak ho pouze odstraníme. V opačném případě (‘+’ nebo prázdný zásobník) přidáme nové ‘+’. Znak ‘b’ se zpracovává obdobně, pouze se k oběma znaménkům chováme opačně.

```
program rovnost_podruhe;

{ Pomocná funkce, která zjistí, co je na vrcholu zásobníku }
function look(var s:stack of char): char;
var c: char;
begin
   if empty(s) then c := '0'          { pokud je prázdný, vrátíme nulu }
   else begin
      c := pop(s);                    { jinak odebereme prvek ze zásobníku }
      push(s, c);                     { a ihned ho vrátíme zpět }
      end;
   look := c;
end;

var r: stack of char;                 { zde je uložen rozdíl a-b }
    c: char;                          { právě zpracovávaný znak }
begin
   while read(c) do begin
      if c='a' then begin             { přečetli jsme ‘a’ => zvyšujeme rozdíl }
         if look(r)='-' then pop(r)
                        else push(r, '+');
         end;
      if c='b' then begin             { přečetli jsme ‘b’ => snižujeme rozdíl }
         if look(r)='+' then pop(r)
                        else push(r, '-');
         end;
      end;
   if empty(r) then write('1')        { je rozdíl nulový? }
               else write('0');
end;
```

Na zpracování každého znaku potřebujeme konstantně mnoho příkazů, takže časová složitost je stále lineární. Na jediném použitém zásobníku se objeví nejvýše tolik znamének, kolik je znaků na vstupu, takže paměťová složitost je taktéž lineární.