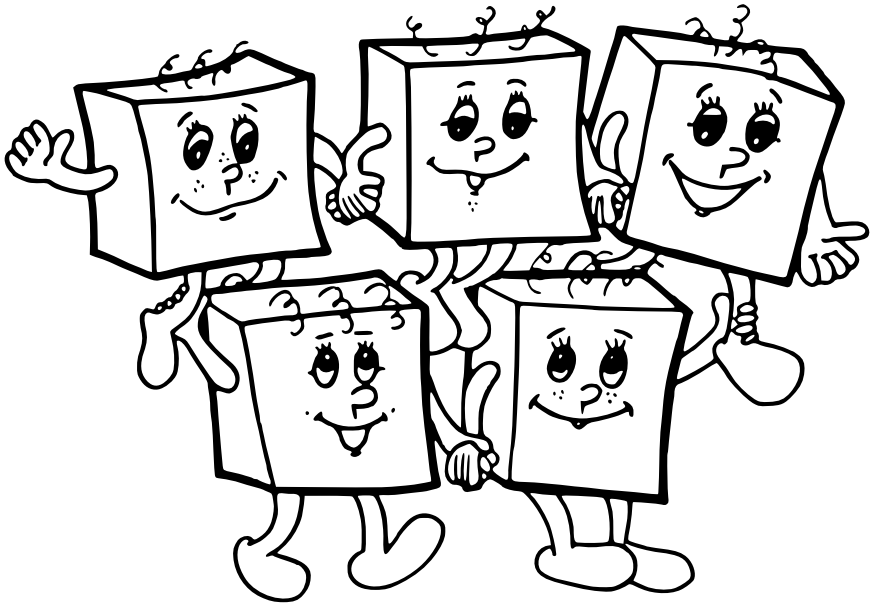
OLYMPIÁDA V INFORMATIKE
NA STREDNÝCH ŠKOLÁCH



dvadsiaty štvrtý ročník
školský rok 2008/09

zadania celoštátneho kola
kategória A

1. súťažný deň

- **Olympiáda v informatike** je od školského roku 2006/07 samostatnou súťažou. Predchádzajúcich 21 ročníkov tejto súťaže prebiehalo pod názvom **Matematická olympiáda, kategória P** (programovanie).
- Oficiálnu **webstránku** súťaže nájdete na `http://oi.sk/`.

Informácie a pravidlá

Celoštátne kolo 24. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 25. – 28. marca 2009. Na riešenie úloh prvého, teoretického dňa majú súťažiaci 4,5 hodiny čistého času. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

Pokiaľ nie je v zadaní povedané ináč, základným kritériom hodnotenia riešenia je správnosť a efektívnosť navrhnutého algoritmu. Hodnotí sa aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.

Súčasťou riešenia by mali teda byť:

- **Popis riešenia,** to znamená slovný popis použitého algoritmu, argumenty zdôvodňujúce jeho správnosť (prípadne dôkaz správnosti algoritmu), diskusiu o efektivite vášho riešenia (časová a pamäťová zložitosť). Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného zápisu algoritmu (do programu).
- **Program.** V úlohách **A-III-1** a **A-III-2** treba uviesť dostatočne podrobný zápis algoritmu, najlepšie v tvare zdrojového textu najdôležitejších častí programu v jazyku Pascal alebo C/C++. Zo zápisu môžete vynechať jednoduché operácie ako vstupy, výstupy, implementáciu jednoduchých matematických vzťahov a pod.

  V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu vysvetlite, ako sa časti programu, ktoré obsahujú volania knižnice, dajú implementovať bez nej bez zhoršenia časovej zložitosti.

  V úlohe **A-III-3** je namiesto klasického programu potrebné uviesť program pre zásobníkový počítač.

Zadania kategórie A

A-III-1 Horár Jedlička II

V krajskom kole pán Jedlička s vašou pomocou a s hrôzou zistil, že z jeho kedysi krásneho a veľkého lesa zostáva len drobný háj. Aby zabránil jeho ďalšiemu zmenšovaniu, najal si tlupu strážnych trollov. Trollovia sú známi svojou silou, menej však už svojou prenikavou inteligenciou. Pán Jedlička sa rozhodol príkazy pre trollov čo najviac zjednodušiť. Každý troll teda dostal pridelené 2 body a má chodiť po úsečke, ktorá ich spája, tam a späť.

Bohužial počas prvého dňa bolo nutné ošetriť skoro všetkých trollov kvôli drobným zraneniam. Pán Jedlička zabudol, že sa úsečky, po ktorých trollovia pochodujú, môžu pretínať, a vo svojích inštrukciách zabudol uviesť, aby sa trollovia vyhýbali zrážkam s ostatnými trollmi. Pokus pridať trollom tento príkaz narazil na ich zábudlivosť. Po niekoľkých opakovaniach ťažkého príkazu „na konci úsečky sa otoč“ trollom pretiekol zásobník a príkaz „vyhýbaj sa zrážkam“ sa stratil, čo malo nepriaznivé dôsledky na rozpočet ošetrovne.

Pán Jedlička sa preto rozhodol všetky križovatky označiť dopravnou značkou „Pozor troll!“. Vašou úlohou je zistiť, kam má tieto značky umiestniť.

Súťažná úloha

Úsečka, po ktorej sa i-ty troll pohybuje, je zadaná dvojicou jej krajných bodov so súradnicami $(a_i, b_i), (c_i, d_i)$, kde a_i, b_i, c_i a d_i sú celé čísla. Krajné body úsečky neležia na žiadnej inej úsečke. Vašou úlohou je vypísať

súradnice priesečníkov týchto úsečiek. Každý priesečník vypíšte iba raz, aj keby sa v ňom pretínali 3 a viac úsečiek. Predpokladajte, že priesečníkov je málo – podstatne menej ako N^2.

Formát vstupu

Prvý riadok obsahuje prirodzené číslo N, udávajúce počet trollov, $0 \leq N \leq 10\,000$. Na nasledujúcich N riadkoch je popis úsečiek, po ktorých sa trollovia pohybujú. Na i-tom riadku sa nachádza štvorica celých čísel a_i, b_i, c_i a d_i, súradnice krajných bodov (a_i, b_i) a (c_i, d_i) úsečky. Môžete predpokladať, že $(a_i, b_i) \neq (c_i, d_i)$.

Formát výstupu

Na každý riadok výstupu vypíšte súradnice jedného z priesečníkov zadaných úsečiek. Poradie priesečníkov môže byť ľubovoľné.

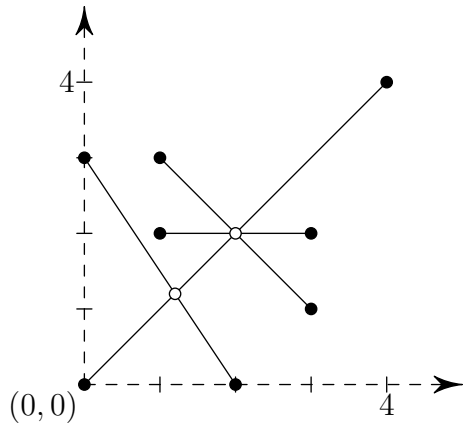
Príklad

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
```



A-III-2 Magické cestovanie

V dôsledku finančnej krízy značne poklesol záujem o magické vystúpenia kúzelníka Davida Aluminumfielda. Preto sa starý kúzelník rozhodol, že čas, ktorý takto nadobudol, využije na precestovanie vzdialených kútov zeme. Klesajúce tržby majú ale za následok, že si nemôže dovoliť konvenčné spôsoby prepravy a musí sa spoľahnúť na osvedčené metódy svojich kúzelníckych predkov.

Začal teda pátrať na povale svojho obydlia a medzi veľkou hromadou zbytočných a starých vecí sa mu podarilo nájsť knihu cestovných kúzel. Jej použitie ale nie je úplne jednoduché. Funguje nasledovne: David si môže vybrať, kde v knihe začne čítať, musí však prečítať súvislý úsek textu – nesmie preskakovať slová ani sa vracať späť. Každé slovo v knihe je nabité nejakou magickou energiou. Zaklínadlo sa podarí len vtedy, ak je celková magická energia prečítaných slov násobkom magickej konštanty K. Zaklínadlo bude tým silnejšie, čím viac slov ho tvorí. (Na veľkosti celkovej magickej energie nezáleží.)

Súťažná úloha

Daná je magická konštanta K, počet slov N v knihe kúzel, a pre každé slovo jedno celé číslo udávajúce jeho magickú energiu. Vašou úlohou bude na základe týchto informácii nájsť najvhodnejší začiatok a koniec zaklínadla – t. j. najdlhší súvislý úsek slov, pre ktorý platí, že súčet ich energických hodnôt je násobkom K.

Číslo K je obyčajne oveľa menšie ako počet slov v knihe.

Formát vstupu

V prvom riadku vstupu sú dve prirodzené čísla – N a K, $1 \leq N \leq 1\,000\,000$ a $1 \leq K \leq 50\,000$, oddelené medzerou. Ako bolo povedané, K je obvykle omnoho menšie ako N. V druhom riadku vstupu je medzerami oddelených N nezáporných čísel $a_1, \ldots, a_N, 0 \leq a_i \leq 1,000,000,000$, ktoré predstavujú magickú energiu jednotlivých slov.

Formát výstupu

Program vypíše dve čísla – i a $j (1 \leq i \leq j \leq N)$, pričom súčet $a_i + a_{i+1} + \ldots + a_j$ musí byť násobkom čísla K a rozdiel $j - i$ najvyšší možný spomedzi všetkých takýchto dvojíc. Ak je možných dvojíc viac, vypíšte ľubovoľnú z nich. Naopak, ak neexistuje žiadna taká dvojica, vypíšte `Neda sa zaklinat`.

Príklad 1

vstup

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

výstup

```
3 7
```

Takisto výstup „1 5“ by bol správny.

Príklad 2

vstup

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

výstup

```
Neda sa zaklinat
```

A-III-3 Zásobníkové počítače

Súťažná úloha

Na vstupe je zadaný reťazec obsahujúci výhradne znaky `a`, `b`, `c` a `d`. Napíšte program pre zásobníkový počítač, ktorý zistí, či má každý zo znakov `a` až `d` rovnaký počet výskytov. Ak áno, nech váš program na výstup vypíše 1, ak nie, nech vypíše 0.

Zamerajte sa na požitie čo najmenšieho počtu zásobníkov aj za cenu vyššej časovej zložitosti.

Príklad

Na vstupy `abcddcba` a `aaabbbcccddd` je správna odpoveď 1.

Na vstup `badbadc` je správna odpoveď 0 (znaky `a`, `b` a `d` se vyskytujú dvakrát, ale znak `c` len raz).

Študijný text

(Študijný text je rovnaký ako v domácom a krajskom kole.)

V tomto ročníku olympiády si predstavíme zásobníkové počítače. Pamäť týchto počítačov je tvorená niekoľkými zásobníkmi. Každý zásobník obsahuje postupnosť hodnôt, s ktorou vie robiť tri operácie: Pridať novú hodnotu na koniec postupnosti (vložiť ju na vrch zásobníka), odstrániť hodnotu z konca postupnosti (vybrať ju zo zásobníka) a zistiť, či je postupnosť v zásobníku prázdna. Okrem zásobníkov vieme mať už len konštantný počet obyčajných premenných. Hodnoty uložené v zásobníkoch aj premenných musia mať vopred určený rozsah, ktorý nezávisí od veľkosti vstupu.

Naše zásobníkové počítače budeme programovať v jazyku Stackal. Ide o odrodu jazyka Pascal, upravenú podľa možností našich počítačov. V nasledujúcich odsekoch popíšeme, v čom sa náš jazyk od klasického Pascalu líši.

Premenné môžu byť týchto typov:

- `boolean` (logická premenná nadobúdajúca hodnoty `true` a `false`),
- `char` (premenná obsahujúca jeden znak),
- `a..b` (celočíselná premenná nadobúdajúca hodnoty od a po b vrátane, pričom $0 \leq a \leq b \leq 100$) a
- `stack of T`, kde `T` je typ iný ako `stack` (zásobník obsahujúci postupnosť hodnôt typu `T`).

Zásobníky sú na začiatku behu programu prázdne, obsah ostatných premenných je nedefinovaný.

Vstupom aj výstupom programu je postupnosť znakov – inými slovami, reťazec. Vstup čítame volaním funkcie `read(c)`, kde `c` je premenná typu `char`. Ak sme ešte vstup nedočítali, táto funkcia uloží do premennej `c` ďalšie písmeno zo vstupu a vráti `true`. V opačnom prípade vráti `false` a hodnotu premennej `c` nezmení. Na výstup zapíšeme znak z premennej `c` volaním procedúry `write(c)`. Na vstupe ani výstupe sa nie je možné vracať ani znaky preskakovať, program musí čítať vstup aj zapisovať výstup zaradom, „zľava doprava“.

Na prácu so zásobníkmi budeme mať špeciálne procedúry a funkcie. Nech `x` je premenná typu `T` a `S` je zásobník obsahujúci hodnoty tohto typu (teda `S` je premenná typu `stack of T`). Volaním procedúry `push(S,x)` vložíme do zásobníka `S` obsah premennej `x`. Funkcia `pop(S)` vyberie hodnotu zo zásobníka a vráti ju ako svoju návratovú hodnotu. Túto funkciu môžeme použiť aj ako procedúru, ak nás vybratá hodnota nezaujíma. Ak je pri volaní funkcie `pop(S)` zásobník `S` prázdny, program skončí s chybou – toto teda robiť nesmieme. Posledná užitočná funkcia je `empty(S)`, ktorá vráti `true` ak je zásobník `S` prázdny a `false` ak nie je. Žiadnym iným spôsobom nevieme s obsahom zásobníkov manipulovať.

K dispozícii máme všetky príkazy klasického Pascalu, môžeme si definovať aj vlastné pomocné procedúry a funkcie, avšak musíme dodržať niekoľko obmedzení. Je zakázané používať rekurziu. Priradzovací príkaz `:=` nesmieme použiť na zásobníky. Zásobník smieme dať ako parameter funkcii len odkazom, teda použitím kľúčového slova `var`.

(Najjednoduchšie riešenie, ako obmedzenia dodržať: Deklarujte všetky použité zásobniky na začiatku programu ako globálne premenné. Potom postupne definujte pomocné funkcie tak, aby každá z nich volala len funkcie definované skôr.)

Časovú a pamäťovú zložitosť definujeme podobne ako u klasických programov. Čas meriame počtom vykonaných príkazov, použitá pamäť je rovná maximálnemu počtu hodnôt, ktoré si program niekedy počas svojho behu pamätal v premenných a zásobníkoch.

Často bude našim cieľom minimalizovať počet použitých zásobníkov, a to aj za cenu horšej časovej zložitosti.

Príklad 1

Napíšte program, ktorý zistí, či je v zadanom reťazci rovnako veľa písmen `a` a `b`, a podľa toho vypíše na výstup buď znak `1` alebo znak `0`.

Riešenie 1a

Keďže rozsah celočíselných premenných je obmedzený, nemôžeme si počítať výskyty znakov `a` a `b` v premenných. Môžeme na to však ľahko využiť dva zásobníky. Do jedného si budeme ukladať `a`-čka a do druhého `b`-čka. Po dočítaní vstupu budeme súčasne vyberať znaky z oboch zásobníkov, a zistíme, či sa oba vyprázdnia naraz.

```
var a, b: stack of char;                    { dva zasobniky na znaky }
    c: char;                                { prave spracuvany znak }
begin
   while read(c) do begin                   { citame zo vstupu, kym sa da }
      if c='a' then push(a, c);             { znak vlozime do spravneho zasobniku }
      if c='b' then push(b, c);
   end;
   while not empty(a) and not empty(b) do begin
      pop(a); pop(b);                       { vyberame znaky z oboch zasobnikov }
   end;
   if empty(a) and empty(b) then write('1')  { su prazdne oba? }
   else write('0');
end;
```

Tento algoritmus má lineárnu časovú aj pamäťovú zložitosť a potrebuje dva zásobníky.

Riešenie 1b

Na riešenie tejto úlohy stačilo použiť len jeden zásobník. Budeme si v ňom pamätať, o koľko viac znakov `a` ako znakov `b` sme doteraz prečítali. Presnejšie, ak bol tento rozdiel kladný, budeme mať v zásobníku príslušný počet znakov `+`, inak tam budeme mať príslušný počet znakov `-`.

Všimnime si napríklad, čo sa stane, keď zo vstupu prečítame ďalšie `a`. Rozdiel sa zvýšil o 1, potrebujeme teda upraviť zásobník. Skontrolujeme, aký znak je na jeho vrchu. Ak je to `-`, len ho odstránime. Ak je tam `+` alebo je zásobník prázdny, pridáme nové `+`. Znak `b` spracujeme opačne.

```
{ Pomocna funkcia, ktora zisti znak na vrchu zasobnika }
function peek(var s:stack of char): char;
var c: char;
begin
   if empty(s) then c := '0'                    { ak je prazdny, vratime napr. nulu }
   else begin
      c := pop(s);                              { inak odoberieme prvok zo zasobnika }
      push(s, c);                               { vlozime ho hned spat }
   end;
   peek := c;                                   { a odovzdame ho ako navratovu hodnotu }
end;

var r: stack of char;                           { tu je ulozeny rozdiel a-b }
    c: char;                                    { prave spracuvany znak }
begin
   while read(c) do begin
      if c='a' then                             { zvysujeme rozdiel }
         if peek(r)='-' then pop(r) else push(r, '+');
      if c='b' then                             { znizujeme rozdiel }
         if peek(r)='+' then pop(r) else push(r, '-');
   end;
   if empty(r) then write('1') else write('0');  { je rozdiel nulovy? }
end;
```

Na spracovanie každého znaku nám stačí konštantný počet príkazov, preto je časová zložitosť naďalej lineárna. V zásobníku bude najviac toľko znamienok, koľko je znakov na vstupe, preto je aj pamäťová zložitosť lineárna.

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE

DVADSIATY ŠTVRTÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX