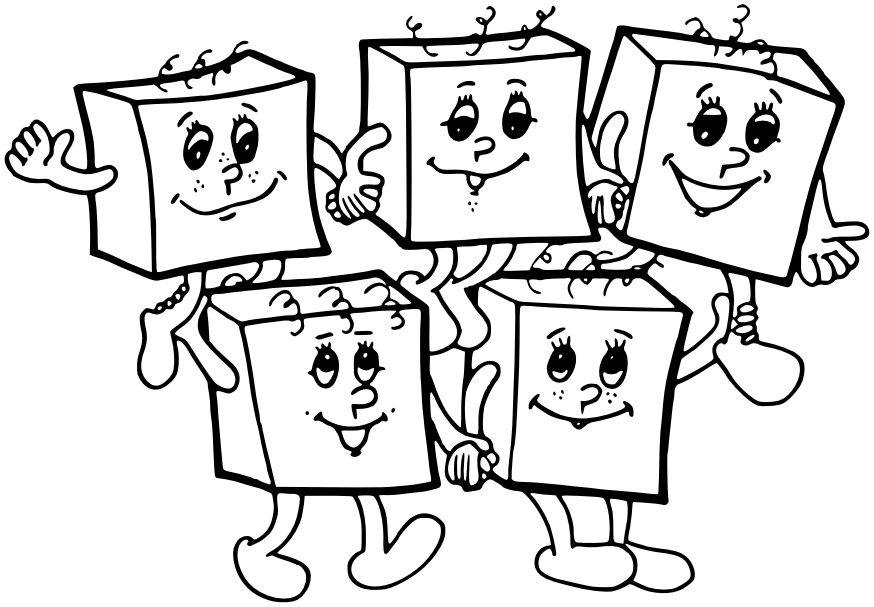
OLYMPIÁDA V INFORMATIKE
NA STREDNÝCH ŠKOLÁCH



dvadsiaty štvrtý ročník
školský rok 2008/09

zadania krajského kola
kategória A

- **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/`.

Priebeh krajského kola

Krajské kolo 24. ročníka Olympiády v informatike, kategórie B, sa koná 13. januára 2008 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci 4 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?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného zápisu algoritmu (do programu).
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v Pascale alebo C/C++.
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení

Za každú úlohu môžete získať od 0 do 10 bodov.

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.

Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov.

V zadaní úlohy môžu byť uvedené limity na veľkosť premenných. Tieto môžete použiť na odhad toho, ako dobré vaše riešenie je. Na počítači, ktorý vykoná miliardu inštrukcií za sekundu, zrieši vzorové riešenie ľubovoľný povolený vstup do niekoľko sekúnd.

Zadania kategórie A

A-II-1 Horár Jedlička

Pán Jedlička kedysi dávno vysadil krásny a veľký les, stromy rástli v radách rozmiestnené s dokonalou presnosťou. Nastal čas, keď les vyrástol a začalo sa s ťažbou dreva. Každý drevorubač začal klátiť stromy na inom okraji lesa, a vysekal do neho poriadnu paseku. Keď sa pán Jedlička prišiel pozrieť na les, objavil už len jeho zvyšok, s hlbokými výsekmi od ťažby. Pána Jedličku by teraz zaujímalo, koľko stromov mu vlastne ostalo. Pomôžete mu s tým?

Súťažná úloha

Stromy v lese pána Jedličku sú zasadené tak, že rastú v mrežových bodoch obdĺžnikovej mriežky. Polohu každého stromu môžeme teda popísať dvojicou celočíselných súradníc (x, y). To, čo z lesa zostalo, je vymedzené mnohouholníkom (ktorý môže byť aj nekonvexný) tak, že vrcholy mnohouholníka sú stromy, a na všetkých mrežových bodoch vnútri alebo na hranici mnohouholníka stále ešte stojí strom. Vašou úlohou je spočítať, koľko mrežových bodov leží vnútri tohoto mnohouholníka, vrátane jeho hranice.

Formát vstupu

Prvý riadok obsahuje prirodzené číslo N – počet vrcholov mnohouholníka ($3 \leq N \leq 100\,000$).

Na nasledujúcich N riadkoch sú popísané jednotlivé vrcholy mnohouholníka. Na i-tom z nich sa nachádza dvojica celočíselných súradníc $1 \leq x_i \leq 10^9$, $1 \leq y_i \leq 10^9$ udávajúca polohu i-teho vrcholu. Vrcholy sú zadané v poradí, v akom ležia na hranici mnohouholníka pri obchádzaní v smere alebo proti smeru hodinových ručičiek.

Formát výstupu

Na jediný riadok výstupu vypíšte jedno celé číslo predstavujúce počet mrežových bodov vnútri mnohouholníka vrátane jeho hranice.

Príklady

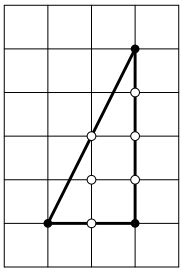
vstup

```
3
1 1
3 5
3 1
```

výstup

```
9
```

Tri mrežové body tvoria vrcholy trojuholníka, päť mrežových bodov leží na hranách a jeden mrežový bod leží vo vnútri.



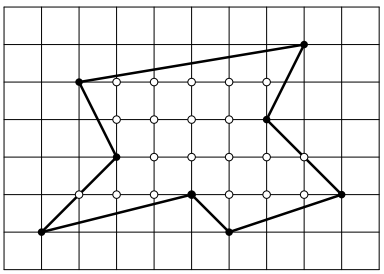
vstup

```
8
2 5
3 3
1 1
5 2
6 1
9 2
7 4
8 6
```

výstup

```
28
```

Osem mrežových bodov tvoria vrcholy mnohouholníka, dva mrežové body ležia na hranách a osemnásť mrežových bodov leží vo vnútri.



A-II-2 Babylonská kríza

V Babylone žila žena, ktorá vraj rozprávala všetkými jazykmi sveta. Vyučovala všetkých Babylončanov aby sláva Babylonu neupadla. Keď sa na svete objavil nový jazyk, tak ho táto žena, tiež Enochiou zvaná, znenazdania vedela tiež. Avšak ako šli roky, tak Enochia starla a čím ďalej tým viac sa bála toho, že by mohla začať jazyky zabúdať – a čo by sa potom stalo s jej Babylonom? Určite by aj veža padla z toľkého nešťastia.

Tak sa teda Enochia rozhodla, že by bolo načase spoľahnúť sa na dáku „techniku“. Sadla si, zobrala dláto a kladivko, a ku každému jazyku čo vedela spísala *wordlist* – zoznam všetkých jeho slov. Potom zvolala vynálezcov, alchymistov, najlepších z najlepších, ale nik si nevedel rady, ako zrealizovať nápad, ktorý mala Enochia. Chcela totiž, aby vynašli spôsob, či snáď prístroj, ktorý by vedel zobrať ľubovoľný text a odhaliť, akýmže to jazykom je písaný.

Súťažná úloha

Máte k dispozícii pomocný súbor `wordlisty.in`, v ktorom je pre každý známy jazyk uložený zoznam jeho slov. Presný formát tohto súboru je popísaný nižšie.

Vašou úlohou je navrhnúť program, ktorý po spustení načíta obsah pomocného súboru a užívateľom zadaný text a určí jazyk, v ktorom je daný text napísaný.

Presnejšie, vašou úlohou je nájsť ten jazyk, do ktorého patrí najviac slov zadaného textu. (Ak sa nejaké slovo vyskytne v texte viackrát, počíta sa samostatne každý jeho výskyt.)

Formát súboru wordlisty.in

V prvom riadku je číslo N ($1 \leq N \leq 10\,000$), ktoré určuje počet wordlistov v súbore. Ďalej nasleduje N wordlistov.

Každý z wordlistov začína dvomi riadkami. V prvom je meno jazyka a v druhom číslo S, ktoré udáva počet slov v tomto wordliste. Potom nasleduje S riadkov, pričom v každom z nich je práve jedno slovo daného jazyka. (Každé slovo je kratšie než 255 znakov a je zložené len z malých písmen anglickej abecedy. Slová v jazyku sú navzájom rôzne a nie sú uvedené v žiadnom konkrétnom poradí.)

Celkový počet slov vo všetkých wordlistoch je najviac 100 000. Meno každého jazyka je tvorené najviac 255 malými písmenami anglickej abecedy. Môžete predpokladať, že mená jazykov sú navzájom rôzne. Rôzne wordlisty môžu obsahovať rovnaké slová.

Formát vstupu

Na štandardnom vstupe je text, ktorý chce Enochia zanalyzovať. Text sa skladá zo slov. Opäť, každé slovo je zložené len z malých písmen anglickej abecedy a má najviac 255 znakov. Slová sú oddelené medzerami alebo koncami riadkov. Text nebude obsahovať viac než 1 000 000 slov. Slová v texte nemusia byť navzájom rôzne.

Formát výstupu

Vypíšte (v ľubovoľnom poradí) všetky jazyky, v ktorých sa nachádza najviac slov zadaného textu.

Príklad

wordlisty.in

```
3
slovencina
4
text
bager
ahoj
zebra
anglictina
4
zebra
by
hello
car
nemcina
3
hallo
auto
sagt
```

vstup

```
tento text
by nepochopila
ani zebra
co ma zebra auto
```

výstup

```
slovencina
anglictina
```

(Aj v slovenčine, aj v angličtine sa vyskytujú po tri slová textu, v nemčine len jedno.)

A-II-3 Cyklistické preteky

Po úspechu horského maratónu sa na vás jeho organizátori obrátili s prosbou o pomocou pri organizovaní cyklistických pretekov. Trasa pretekov by mala viesť z cieľa horského maratónu, Výšných Hákov, do hlavného mesta, Veľkého Sumca. Cyklistické preteky budú tvorené niekoľkými etapami a organizátori už určili možné dvojice miest, medzi ktorými by mohli viesť jednotlivé etapy pretekov. Pre každú takúto dvojicu naviac odhadli počet divákov, ktorí by sa prišli pozrieť na preteky. Pretože rozpočet celých pretekov je obmedzený, organizátori by radi trasu pretekov navrhli tak, aby mala čo najmenej etáp, a pritom by ich videlo čo najviac divákov.

Súťažná úloha

Váš program dostane zoznam dvojíc miest, medzi ktorými by mohli viesť etapy pretekov, a pre každú dvojicu odhad počtu divákov, ktorí by sa na danú etapu prišli pozrieť. Jednotlivé etapy, ktoré tvoria trasu pretekov, musia na seba v koncových mestách nadväzovať. Program by mal nájsť trasu pretekov vedúcu z Vyšných Hákov do Veľkého Sumca, ktorá má čo najmenej etáp a medzi všetkými takými trasami určiť takú, ktorú uvidí čo najväčší počet divákov (počty divákov jednotlivých etáp sa sčítavajú). Pokiaľ je takých trás viac, program môže vypísať ľubovoľnú z nich.

Prvý riadok vstupu obsahuje dve prirodzené čísla N a M, $2 \leq N \leq 100\,000$ a $1 \leq M \leq 1\,000\,000$; N je počet miest a M je počet dvojíc miest, medzi ktorými by mohla viesť jedna etapa pretekov. Mestá sú očíslované od 1 po N, pričom Vyšné Háky majú číslo 1 a Veľký Sumec má 2. Na každom z M nasledujúcich riadkov je trojica čísel A, B a D ($1 \leq A, B \leq N$ a $0 \leq D \leq 1\,000$) popisujúca jednu z dvojice miest, medzi ktorými by mohla viesť jedna etapa: etapa by mohla viesť medzi mestami s číslami A a B a očakávaný počet divákov pre túto etapu je D. Etapa pretekov môže ísť z mesta A do B alebo aj opačným smerom.

Môžete predpokladať, že existuje aspoň jedna možná trasa pretekov. Vstup naviac neobsahuje dva rôzne riadky, ktoré by opisovali etapu medzi rovnakou dvojicou miest.

Na prvý riadok výstupu vypíšte dve čísla: najmenší počet etáp P pretekov z 1 do 2 a najväčší počet divákov, ktorý by mohli pozerať takéto preteky. Na druhý riadok vypíšte $P+1$ čísel miest, ktoré tvoria optimálnu trasu. Ak je optimálnych trás, vyberte si ľubovoľnú jednu z nich.

Príklad

vstup

```
6 9
1 6 10
1 3 8
4 6 2
4 3 7
5 6 0
5 3 4
4 5 100
2 4 1
2 5 2
```

výstup

```
3 16
1 3 4 2
```

A-II-4 Zásobníkové počítače

Súťažná úloha

Napíšte program pre zásobníkový počítač, ktorý vyhodnotí zadaný logický výraz a vypíše jeho hodnotu na výstup. Program by mal používať čo najmenší počet zásobníkov.

Logické výrazy zapisujeme pomocou znakov `0`, `1`, `&`, `|`, `(` a `)`. Znaky `0` a `1` sú logické konštanty (`0` je nepravda, `1` je pravda), `&` je logický súčin (spojka a: `0&0=0&1=1&0=0`, `1&1=1`), `|` je logický súčet (spojka alebo: `0|0=0`, `0|1=1|0=1|1=1`) a zátvorky fungujú bežným spôsobom. Ak nie sú uvedené zátvorky, tak súčin má prednosť pred súčtom, a teda `1|0&0|1 = 1|(0&0)|0 = 1|0|0 = 1`.

Študijný text

(Študijný text je rovnaký ako v domácom 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