58. ročník Matematické olympiády – 2008/2009

Úlohy krajského kola kategorie P

Krajské kolo 58. ročníku MO kategorie P se koná v úterý 13. 1. 2009 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.

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-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 řešení úlohy **P-II-4** je nutnou součástí řešení program pro zásobníkový počítač.

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

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

P-II-1 Lesník Jehlička

Pan Jehlička kdysi vysázel krásný a veliký les, stromy rostoucí v přesných řadách se ani jehličkou neodchylovaly od dokonalosti. Přišel čas, kdy les již vyrostl, a začalo se s těžbou dřeva. Každý z dřevorubců začal kácet na jiném okraji lesa a vysekal do něj pořádnou paseku. Když se pan Jehlička přišel na les podívat, objevil jen jeho zbytek, s hlubokými výseky od těžby. Pana Jehličku by teď zajímalo, kolik stromů mu vlastně zbylo. Pomůžete mu s tím?

Soutěžní úloha

Stromy v lese pana Jehličky jsou vysázeny tak, že rostou v mřížových bodech obdélníkové mřížky se stranami tvořenými D a S body. Polohu každého stromu lze tedy popsat dvojicí celočíselných souřadnic (x, y), kde $1 \leq x \leq D$, $1 \leq y \leq S$. To,

co z lesa zbylo, je vymezeno mnohoúhelníkem (který může být i nekonvexní) tak, že vrcholy mnohoúhelníka jsou stromy a na všech mřížových bodech uvnitř i na hranici mnohoúhelníka stále ještě stojí strom. Vaším úkolem je spočítat, kolik mřížových bodů leží uvnitř tohoto mnohoúhelníka, včetně jeho hranice.

Formát vstupu

První řádek obsahuje přirozená čísla D, S, N – velikost mřížky $2 \leq D, S \leq 10^9$ a počet vrcholů mnohoúhelníka $3 \leq N \leq 100\,000$.

Na následujících N řádcích jsou popsány vrcholy mnohoúhelníka. Na i-tém z nich je dvojice celočíselných souřadnic $1 \leq x_i \leq D$, $1 \leq y_i \leq S$ udávající polohu i-tého vrcholu. Vrcholy jsou zadané v pořadí, v jakém leží na hranici mnohoúhelníka při obcházení po nebo proti směru hodinových ručiček.

Formát výstupu

Na jediný řádek výstupu vypište jedno celé číslo představující počet mřížových bodů uvnitř mnohoúhelníka včetně jeho hranice.

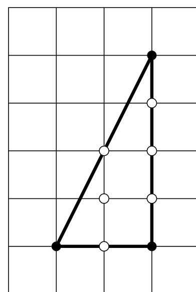
Příklady

Vstup:

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

Výstup:

```
9
```



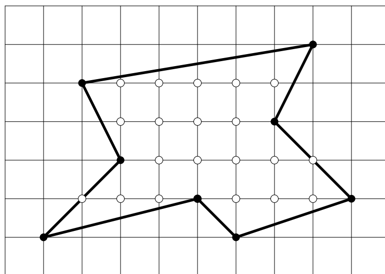
Tři mřížové body tvoří vrcholy trojúhelníka, pět mřížových bodů leží na hranách a jeden mřížový bod leží uvnitř.

Vstup:

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

Výstup:

```
28
```



Osm mřížových bodů tvoří vrcholy mnohoúhelníka, dva mřížové body leží na hranách a osmnáct mřížových bodů leží uvnitř.

P-II-2 Čínsky nebo česky?

Děd Vševěd zestárl. Nelze se tedy divit tomu, že na všetečné dotazy už neodpovídá tak rád a ochotně jako dříve. Navíc ho začala zlobit paměť, takže se tu a tam stává, že něco neví, což je v jeho profesi velmi nemilé. Proto učinil moudré rozhodnutí – předat své poslání někomu mladšímu, a to rovnou svému vnukovi.

Vnuk Vševěd je na svůj věk neskonale moudrý, chytrý a navíc přečetl všechny encyklopedie i wikipedie. Bohužel se ale nestihl za tak krátkou dobu naučit dostatek cizích jazyků.

Což o to, s tím by, díky programu TheBestTranslatorOfTheWorld 5.0, neměl být žádný problém. Nepříjemné je, že tento program sám od sebe neumí zjistit, z jakého jazyka má překládat, takže je nutné vstupní jazyk nastavit ručně.

Vyzkoušet všech 10^9 jazyků, které program umí, by zabralo neúnosně dlouhou dobu, takže před kanceláří vnuka Vševěda by se tvořily fronty. Proto by potřeboval ještě jeden program, který by uměl zjistit, v jakém jazyku je daný text napsaný.

Soutěžní úloha

Vaším úkolem je navrhnout program, který určí jazyk, jehož slova se v daném textu vyskytují nejčastěji. Váš program bude mít k dispozici dva soubory: `slovniky.in`, ve kterém jsou uložena slova jednotlivých jazyků, a `text.in`, ve kterém je text, který má program analyzovat. Do souboru `jazyk.out` váš program zapíše jméno jazyka, jehož slova jsou v textu nejčastěji obsažena. V případě, že takových jazyků bude více, měl by program vypsat všechny takové jazyky. Opakované výskyty téhož slova v souboru `text.in` se počítají do počtu slov opakovaně.

Formát vstupu – soubor slovniky.in

Na prvním řádku souboru je číslo N ($1 \leq N \leq 10\,000$), které udává počet slovníků v souboru. Dále následuje N slovníků.

Každý ze slovníků začíná dvěma řádky. Na prvním je uvedeno jméno jazyka a na druhém číslo S, které udává počet slov v tomto slovníku. Poté následuje S řádků, přičemž na každém z nich je právě jedno slovo daného jazyka (každé slovo je kratší než 255 znaků a je tvořeno pouze malými písmeny anglické abecedy). Tato slova nejsou uvedena v žádném určitém pořadí, ale jsou navzájem různá. Celkový počet slov ve všech slovnících dohromady je nejvýše 100 000.

Jméno každého jazyka je tvořeno nejvýše 255 malými písmeny anglické abecedy. Můžete předpokládat, že jména jazyků jsou navzájem různá. Různé slovníky mohou obsahovat stejné slovo.

Formát vstupu – soubor text.in

V tomto souboru je uložen text určený k analýze. Text je tvořen slovy kratšími než 255 znaků, přičemž slovem nazýváme souvislý úsek malých písmen anglické abecedy. Slova jsou oddělena mezerami nebo konci řádků. Text nebude obsahovat více než 1 000 000 slov. Tato slova nemusí být navzájem různá.

Formát výstupu – soubor jazyk.out

Pokud existuje jeden jazyk, jehož slova se vyskytují v zadaném textu nejčastěji, program vypíše do výstupního souboru jméno tohoto jazyka. V případě, že takových jazyků je více, program vypíše jména všech takových jazyků na samostatné řádky (v libovolném pořadí).

Příklad

slovniky.in

```
3
cestina
4
text
bagr
ahoj
zebra
anglictina
4
zebra
by
hello
car
nemcina
3
hallo
auto
sagt
```

text.in

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

jazyk.out

```
cestina
anglictina
```

P-II-3 Cyklistický závod

Po úspěchu horského maratonu se na vás jeho organizátoři obrátili s prosbou o pomoc při organizaci cyklistického závodu. Trasa závodu by měla vést z cíle horského maratonu, Vyšných Háků, do hlavního města Velkého Sumce. Cyklistický závod bude tvořen několika etapami a organizátoři již určili možné dvojice měst, mezi kterými by mohly vést jednotlivé etapy závodu. Pro každou takovou dvojici navíc odhadli počet diváků, kteří by se na závod přišli podívat. Protože rozpočet celého závodu je omezený, organizátoři by rádi trasu závodu navrhli tak, aby měl co nejméně etap, a přitom ho vidělo co nejvíce diváků.

Soutěžní úloha

Váš program obdrží seznam dvojic měst, mezi kterými by mohly vést etapy závodu, a pro každou dvojici odhad počtu diváků, kteří by se na danou etapu přišli podívat. Jednotlivé etapy, které tvoří trasu závodu, musí na sebe v koncových městech navazovat. Program by měl nalézt trasu závodu vedoucí z Vyšných Háků do Velkého Sumce, která má co nejméně etap, a mezi všemi takovými trasami určit tu, kterou shlédne největší počet diváků (počty diváků jednotlivých etap závodu se sčítají). Pokud je i takových tras více, program může vypsat libovolnou z nich.

Formát vstupu

První řádek obsahuje dvě přirozené čísla N a M, $2 \leq N \leq 1\,000\,000$ a $1 \leq M \leq 1\,000\,000$; N je počet měst a M je počet dvojic měst, mezi kterými by mohla jedna z etap závodu vést. Města jsou očíslována čísly mezi 1 a N, přičemž Vyšné Háky mají číslo 1 a Velký Sumec má 2. Na každém z M následujících řádků je trojice čísel A, B a D ($1 \leq A, B \leq N$ a $0 \leq D \leq 1\,000$) popisující jednu z dvojic měst, mezi kterými by mohla vést jedna etapa závodu: etapa závodu by mohla vést mezi městy s čísly A a B a očekávaný počet diváků pro tuto etapu je D. Etapa závodu může vést z města A do města B nebo naopak.

Můžete předpokládat, že zadané dvojice měst pro etapy závodu umožňují sestavit aspoň jednu trasu závodu. Vstup navíc neobsahuje dva různé řádky, které by popisovaly etapu mezi stejnou dvojicí měst.

Formát výstupu

Na první řádek výstupu vypiště dvě čísla, nejmenší možný počet P etap závodu z města s číslem 1 do města s číslem 2 a největší počet diváků, který by mohl shlédnout takový závod tvořený P etapami. Na druhý řádek vypište $P + 1$ čísel měst, která tvoří optimální trasu.

Příklad

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
```

P-II-4 Stackal

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

Soutěžní úloha

Napište program pro zásobníkový počítač, který vyhodnotí zadaný logický výraz a vypíše jeho hodnotu na výstup. Program by měl používat nejmenší možný počet zásobníků.

Logické výrazy zapisujeme pomocí znaků `0`, `1`, `&`, `|`, `(` a `)`. Znaky `0` a `1` slouží jako konstanty (nepravda a pravda), `&` je logický součin (`0&0` = `0&1` = `1&0` = `0`, `1&1` = `1`), `|` je logický součet (`0|0` = `0`, `0|1` = `1|0` = `1|1` = `1`) a závorky fungují běžným způsobem. Pokud závorky neuvedeme, součin má přednost před součtem, tedy `1|0&0|0` = `1|(0&0)|0` = `1|0|0` = `1`.

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.