58. ročník matematické olympiády na středních školách

Kategorie P

In: Karel Horák (editor); Martin Mareš (editor); Peter Novotný (editor); Martin Panák (editor); Jaromír Šimša (editor); Jaroslav Švrček (editor); Pavel Töpfer (editor): 58. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané ve školním roce 2008/2009. 50. mezinárodní matematická olympiáda. 21. mezinárodní olympiáda v informatice. (Czech). Praha: Jednota českých matematiků a fyziků, 2011. pp. 103–120.

Persistent URL: http://dml.cz/dmlcz/405173

P – I – 1

Učebnice

„Píšeme dějiny, týhletý krajiny, jaký to příběh je...“ prozpěvoval si král Zloburtus a rozverně upravoval učebnice dějepisu. Přece jen byly napsány za jeho předchůdce krále Pravdomila III. a ten měl na historii příliš upjaté názory. Třeba tvrdil, že rod Pravdomilů byl starší a urozenější než rod Zloburtusů. Což ovšem nebyla pravda a bylo potřeba vše napravit. To je spousta práce, a tak vás královským rozkazem požádal o program, který by dokázal nahrazovat nevhodná slova slovy vhodnějšími.

Soutěžní úloha. Pro zadaný seznam nevhodných slov a jejich náhrad upravte vstupní text tak, že každý výskyt nevhodného slova v textu nahradíte jeho vhodnějším „ekvivalentem“.

Formát vstupu: Vstupní soubor se jmenuje `ucebnice.in`. Na prvním řádku souboru se nachází přirozené číslo N ($1 \leqq N \leqq 100\,000$), které udává počet nevhodných slov. Následuje N řádků, přičemž na každém z nich se nacházejí vždy dvě slova oddělená mezerou, která jsou tvořena pouze velkými písmeny. První slovo na každém z těchto N řádků je „nevhodné“ a druhé je jeho vhodnější náhrada. Od $(N + 2)$-ého řádku až do konce souboru pak následuje samotný text učebnice. Text je tvořen pouze velkými písmeny anglické abecedy a mezerami (a konci řádků), přičemž souvislé úseky písmen tvoří jednotlivá slova. Můžete předpokládat, že žádné slovo nebude delší než 255 písmen a že N nevhodných slov je navzájem různých.

Formát výstupu: Výstupní soubor se jmenuje `ucebnice.out`. Výstupem programu je text, ve kterém byla všechna nevhodná slova nahrazena jejich vhodnějšími ekvivalenty. Ostatní slova, tj. zbytek souboru, musí zůstat beze změny. Výstupní text musí též zachovávat mezery mezi slovy a odřádkování podle vstupního souboru.

Příklad:

Vstupní soubor `ucebnice.in`:

```
5
PRAVDOMIL␣ZLOBURTUS
ZLOBURTUS␣PRAVDOMIL
DRACKA␣DRAKA
ZROUNA␣MRACKA
ZBYTECNA␣DVOJICE
A PAK␣HRDINNY␣PRAVDOMIL␣PRAVDOMILUJICNE␣SRAZIL␣K ZEMI␣DRACKA␣ZROUNA
VSE␣LZE␣NAJIT␣VE␣FILMU␣␣HISTORIE␣RODU␣PRAVDOMILU

V TETO␣UCEBNICI␣BUDE␣HANEN␣ZLOBURTUS
```

Výstupní soubor `ucebnice.out`:

```
A PAK␣HRDINNY␣ZLOBURTUS␣PRAVDOMILUJICNE␣SRAZIL␣K ZEMI␣DRAKA␣MRACKA
VSE␣LZE␣NAJIT␣VE␣FILMU␣␣HISTORIE␣RODU␣PRAVDOMILU

V TETO␣UCEBNICI␣BUDE␣HANEN␣PRAVDOMIL
```

P – I – 2

Egyptské pyramidy

Amon a Thespis kontrolují bezpečnost pyramid před lupiči ve starověkém Egyptě. Amona stavitelé vpustí do nové pyramidy a čekají, jestli se dostane k některému z ukrytých pokladů. Thespis zůstane venku se stavebním plánem a voláním Amona naviguje. Ani jeden z nich bohužel není žádný génius a Thespis často netuší, kam by měl Amona poslat. U vaší firmy na děrování hliněných destiček si proto objednali návrh algoritmu na hledání cest v pyramidách.

Pyramidy i jejich interiéry jsou podle odvěké tradice zorientovány podle světových stran a plány se obvykle kreslí na čtverečkovaný papyrus, přičemž jeden čtvereček odpovídá délce 1 m. Podle nejnovějších bezpečnostních trendů se v pyramidě též vyskytují uzamykatelné dveře a klíče. Každé z políček plánu pyramidy je chodba, stěna, poklad, dveře nebo místnost, kde se nachází klíč. Dveře a klíče jsou pro odlišení označeny červenou, zelenou, modrou a fialovou barvou.

Aby se Amon v pyramidě neztratil, otáčí se jen o násobky devadesáti stupňů a každý jeho krok měří přesně 1 m, tj. Amon se může pohybovat pouze vodorovně nebo svisle o jedno pole. Na pole může Amon vstoupit, pokud je prázdné, je to poklad, místnost s klíčem, nebo pokud jsou to dveře, ke kterým už má příslušný klíč. Klíč Amon sebere, jakmile vstoupí na pole, kde se klíč nachází, a klíč mu už poté zůstává po celý zbytek pohybu v pyramidě. Jedním klíčem může Amon odemknout libovolně mnoho dveří barvy shodné s barvou klíče.

Díky magickým formulím vyřčeným faraónskými kouzelníky platí, že pokud Amon udělá jeden krok doleva z políčka v nejlevějším sloupci plánku, ocitne se na políčku ve stejném řádku v nejpravějším sloupci. Na druhou stranu, krok doprava z nejpravějšího sloupce ho přivede do nejlevějšího sloupce. Podobně krok nahoru z prvního řádku ho přemístí do posledního řádku a naopak. Jakmile se Amon dostane na políčko s pokladem, dá to hlasitým voláním najevo, a Thespidův úkol navigování uvnitř pyramidy končí.

Vaším úkolem je napsat program, který pro zadanou mapu pyramidy najde cestu k některému z pokladů nebo určí, že žádná taková cesta není možná. Pamatujte, že v pyramidě nemusí být vůbec žádný poklad ukryt, mohou existovat dveře bez příslušných klíčů a také klíče, ke kterým neexistují žádné dveře stejné barvy. Také se může vyskytnout více pokladů nebo dveří či klíčů stejné barvy. Amonova výchozí pozice je ale na plánu vždy právě jedna.

Formát vstupu: Vstupní soubor se jmenuje `pyramida.in`. Na jeho prvním řádku jsou uvedeny rozměry plánu pyramidy jako dvě přirozená čísla R (počet řádků) a S (počet sloupců) oddělená jednou mezerou. Můžete předpokládat, že $1 \leqq R \times S \leqq 1\,000\,000$. Na každém z následujících R řádků je vždy právě S znaků popisujících mapu pyramidy s následujícími významy:

`#`	zeď
`.`	volné políčko
`*`	Amonova výchozí pozice
`CZMF`	červené, zelené, modré nebo fialové dveře
`czmf`	červený, zelený, modrý nebo fialový klíč
`$`	poklad

Mapa pyramidy je ve vstupním souboru orientována jako běžné mapy, tj. její horní okraj je severní.

Formát výstupu: Výstupní soubor `pyramida.out` obsahuje jediný řádek, který popisuje některou z nejkratších posloupností kroků, jimiž se Amon může dostat k některému z pokladů. Posloupnost Amonových kroků k pokladu je zadána jako posloupnost znaků ‚`S`', ‚`J`', ‚`V`' a ‚`Z`', které udávají, na kterou světovou stranu má Amon udělat následující krok. Pokud žádná taková posloupnost neexistuje, tj. poklad není možné získat, výstupní soubor obsahuje pouze slovo ‚`NELZE`'.

Příklady:

Vstup:

```
3 5
#####
#*F$#
#####
```

Výstup:

```
NELZE
```

Vstup:

```
3 19
###################
#$F.zMc.*.Cm.Z.ZZf#
###################
```

Výstup:

```
ZZVVVVVZZZZZZZVVVVVVVVVVVVVZZZZZZZZZZZZZZZZZ
```

Vstup:

```
3 1
*
.
$
```

Výstup:

```
S
```

(Přejdeme přes severní okraj.)

P – I – 3

Horský maraton

V jedné hornaté zemi si všimli, že ve velkoměstech získávají na popularitě všelijaké maratony, půlmaratony a podobné běžecké závody, které se odehrávají v jejich ulicích. Rozhodli se, že podobný závod uspořádají ve svých velehorách. Uběhnout více než 42 kilometrů ve vysoké nadmořské výšce ovšem není žádná legrace. Chtějí proto postavit závod na jiném principu — nebude tak dlouhý, ale jeho náročnost bude spočívat v rozdílu mezi nadmořskou výškou startu a cíle.

Protože podobné akce jsou často využívány k propagaci, pořadatelé stanovili, že trasa tohoto „maratonu" by měla vést po nedávno otevřené turistické značce, která prochází celým pohořím, a start i cíl by měly být u některého z rozcestí. Aby však závod měl stále ještě něco společného s maratonem, potřebují vybrat takovou trasu, aby rozdíl nadmořských výšek cíle a startu byl pokud možno co nejblíže předem dané hodnotě X, což bude vhodně zvolený násobek oné maratonské délky 42,195 km.

Organizátoři vás požádali o pomoc s výběrem trasy závodu. Trasa turistické značky se skládá z N úseků mezi rozcestími. Pro každý z úseků je zadáno celé číslo a_i — rozdíl nadmořských výšek konce a začátku úseku. Pokud je a_i kladné, pak trasa stoupá, pokud je záporné, tak trasa klesá, a pokud je nulové, tak trasa v tomto úseku vede po rovině. Vaším úkolem

je nalézt čísla i a j, $1 \leqq i \leqq j \leqq N$, taková, že rozdíl mezi nadmořskou výškou konce j-tého úseku a začátku i-tého úseku je co nejbližší číslu X. Jinými slovy, vaším úkolem je nalézt v zadané posloupnosti souvislou podposloupnost $a_i, a_{i+1}, \ldots, a_j$, jejíž součet je S a $|X - S|$ je co nejmenší možné.

Při řešení úlohy nepředpokládejte, že by velikost čísel v posloupnosti byla omezená (přeci jen nevíte, jak přesné údaje vám pořadatelé zadají), ale můžete předpokládat, že všechny aritmetické operace vyžadují čas $O(1)$.

Příklad: Pro posloupnost $3, 5, -2, 5, 4$ délky $N = 5$ a pro $X = 7$ je hledanou podposloupností $-2, 5, 4$ (tedy $i = 3$ a $j = 5$) se součtem 7. Pro $X = 10$ jsou možným řešením podposloupnost $3, 5, -2, 5$ se součtem 11 a jiným možným řešením je podposloupnost $5, 4$ se součtem 9. Váš program nemusí nalézt všechna nejlepší řešení — stačí, když vypíše libovolné z nich.

P – I – 4

Stackal

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 během 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.

Soutěžní úloha. Napište program pro zásobníkový počítač, který o zadaném řetězci rozhodne, zda je symetrický. Tak se říká těm řetězcům, které se pozpátku čtou stejně jako popředu, jinými slovy první znak je stejný jako poslední, druhý jako předposlední a tak dále.

a) Snažte se o co nejlepší časovou složitost. *(5 bodů)*

b) Použijte nejmenší možný počet zásobníků. *(5 bodů)*

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. Polohu každého stromu lze tedy popsat dvojicí celočíselných souřadnic (x, y). To, co z lesa zbylo, je vymezeno mnohoúhelníkem (který může být i nekonvexní) tak, že vrcholy mnohoúhelníku jsou stromy a na všech mřížových bodech uvnitř i na hranici mnohoúhelníku stále ještě stojí strom. Vaším úkolem je spočítat, kolik mřížových bodů leží uvnitř tohoto mnohoúhelníku, včetně jeho hranice.

Formát vstupu: První řádek obsahuje přirozené číslo N — počet vrcholů mnohoúhelníku ($3 \leqq N \leqq 100\,000$).

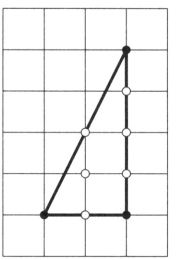
Na následujících N řádcích jsou popsány jednotlivé vrcholy mnohoúhelníku. Na i-tém z nich je dvojice celočíselných souřadnic $1 \leqq x_i \leqq 10^9$,

$1 \leqq y_i \leqq 10^9$ udávající polohu i-tého vrcholu. Vrcholy jsou zadané v pořadí, v jakém leží na hranici mnohoúhelníku při obcházení po nebo proti směru hodinových ručiček.

Formát vstupu: Na jediný řádek výstupu vypište jedno celé číslo představující počet mřížových bodů uvnitř mnohoúhelníku včetně jeho hranice.

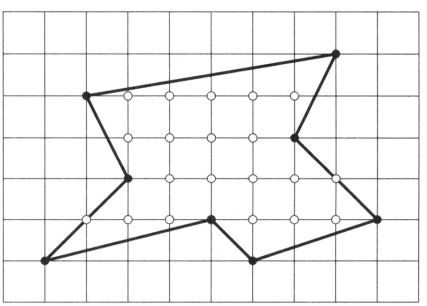
Příklady:

Vstup:	*Výstup*:
3	9
1 1	
3 5	
3 1	



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

Vstup:	*Výstup*:
8	28
2 5	
3 3	
1 1	
5 2	
6 1	
9 2	
7 4	
8 6	



Osm mřížových bodů tvoří vrcholy mnohoúhelníku, 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 ($1 \leqq N \leqq 10\,000$), 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 vý-

stupního 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, $2 \leqq N \leqq 1\,000\,000$ a $1 \leqq M \leqq 1\,000\,000$; 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 ($1 \leqq A, B \leqq N$ a $0 \leqq D \leqq 1\,000$) 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 vypište 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

K úloze se vztahuje studijní text z úlohy P–I–4.

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`.

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.

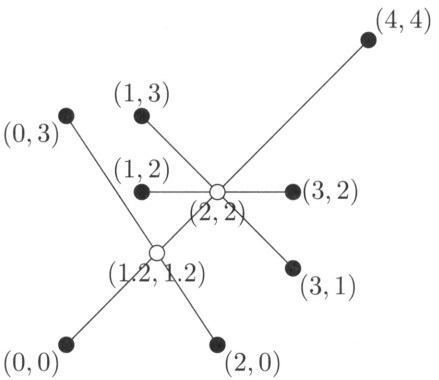
Pan Jehlička se proto rozhodl všechna křížení označit dopravní značkou „Pozor, troll!“. Vaším úkolem je zjistit, kam tyto značky umístit.

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, $0 \leqq N \leqq 10\,000$. 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:	*Výstup*:
4	1.2 1.2
0 0 4 4	2 2
0 3 2 0	
1 3 3 1	
1 2 3 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, $1 \leqq N \leqq 1\,000\,000$ a $1 \leqq K \leqq 50\,000$, 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 $a_1, \ldots, a_n$, $0 \leqq a_i \leqq 1\,000\,000\,000$, která představují energie jednotlivých slov v knize.

Formát výstupu: Program vypíše dvě čísla i a j ($1 \leqq i \leqq j \leqq N$), 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:	*Výstup*:
`8 5` `1 2 8 6 3 4 4 9`	`3 7`

Příklad 2:

Vstup:	*Výstup*:
`5 8` `1 1 1 1 1`	`Nelze zaklínat.`

P – III – 3

Stackal

K úloze se vztahuje studijní text z úlohy P–I–4.

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).

P – III – 4

Výlet do Švýcarska

Program: `swiss.pas` / `swiss.c` / `swiss.cpp`
Vstup: `swiss.in`
Výstup: `swiss.out`

Tibor je vášnivý cyklista. Kromě cyklů v grafech má však také rád hory. Rozhodl se, že se o prázdninách vypraví s kamarády na kola do Švýcarska. Chtěli by podniknout čtyřdenní výlet a spát budou vždy v hotelu v některém ze švýcarských měst. Vlakem dojedou do jednoho z měst a odtud vyrazí do strmých kopců a hlubokých údolí. Na internetu si našli seznam jednodenních tras mezi švýcarskými městy, dokonce i s bohatým hodnocením od ostatních cyklistů. Stejně jako mnoho dalších věcí v Tiborově životě, i výlet musí tvořit cyklus. Po čtyřech dnech se tedy musí vrátit zase zpět do města, ze kterého vyrazili. Teď jen naplánovat tu nejlepší trasu. Tibor však zjistil, že to není tak jednoduché a rád by, abyste mu pomohli.

Soutěžní úloha. Švýcarská města si pro jednoduchost očíslujeme čísly od 1 do N a počet tras nalezených na internetu si označíme M. Každá trasa vždy spojuje dvě různá města a žádné dvě trasy nespojují stejnou dvojici měst. Navíc má každá trasa ohodnocení, což je přirozené číslo menší nebo rovné 256. Všechny trasy lze projet oběma směry a ohodnocení trasy je pro oba směry stejné. Z každého města vede nejvýše 100 tras.

Tibor přirozeně nechce jet žádnou z tras dvakrát. Vaším úkolem je najít čtyři na sebe navazující trasy takové, že první z nich začíná ve stejném městě, kde poslední končí, a tyto trasy mají největší součet ohodnocení. Pokud existuje více řešení, můžete vybrat libovolné z nich.

Formát vstupu: První řádek vstupního souboru `swiss.in` obsahuje přirozená čísla N a M, počet měst $4 \leqq N \leqq 10\,000$ a počet tras $4 \leqq \leqq M \leqq 1\,000\,000$.

Na následujících M řádcích jsou popsány jednotlivé trasy. Na i-tém řádku jsou přirozená čísla x_i, y_i a h_i, kde $1 \leqq x_i \leqq N$ a $1 \leqq y_i \leqq N$ jsou čísla měst, které spojuje i-tá trasa, a $1 \leqq h_i \leqq 256$ je ohodnocení i-té trasy. Žádné město se nevyskytuje ve více než 100 trasách.

Formát výstupu: Na první řádek výstupního souboru `swiss.out` vypište nejvyšší součet hodnocení čtyř tras, které splňují podmínky zadání. Na druhý řádek vypište pět čísel navštívených měst v pořadí, v jakém je cyklisté projedou. Dle zadání úlohy musí být první a poslední z těchto čísel stejná.

Pokud žádné takové čtyři trasy neexistují, výstup bude tvořen jedním řádkem se slovem „`NEEXISTUJE`“.

Příklady:

Vstup:
```
6 9
1 2 10
2 5 11
3 1 10
6 3 7
1 4 3
2 6 15
5 3 10
4 5 5
4 6 9
```

Výstup:
```
43
2 6 3 5 2
```

Vstup:
```
7 9
1 2 1
2 3 1
1 3 1
3 4 1
3 5 1
5 4 1
5 6 1
6 7 1
2 7 1
```

Výstup:
```
NEEXISTUJE
```

P – III – 5

Záchranná akce

Program: `akce.pas` / `akce.c` / `akce.cpp`
Vstup: `akce.in`
Výstup: `akce.out`

Do lunární stanice na Měsíci narazil neznámý předmět a jedinou šancí na záchranu je vyslat teleportem robota s prosbou o pomoc do některé z dalších stanic na Měsíci. Nejbližší takovou stanicí je stanice Alfa, která je však kromě jednoho skladiště zcela odstíněna proti teleportaci.

Mapa skladiště je obdélníková mřížka čtvercových polí, kde je každé pole buď celé zabráno překážkou (a tedy je neprůchozí) nebo je pro robota celé průchozí. Robot je malý primitivní model, kterého lze ovládat jen tak, že se mu zadá posloupnost jednoduchých příkazů a on ji neustále opakuje. Příkazy jsou čtyři: pro otočení robota vlevo a vpravo (o 90°) a pro pohyb vpřed a vzad (na sousední pole).

Robot se po teleportaci může ocitnout na libovolném políčku na mapě skladiště, kde není překážka, a je natočen směrem na sever. Pokud se robot pokusí pohnout na místo s překážkou, zůstane na místě (zarazil se o ni) a pokračuje následujícím příkazem ze své posloupnosti. Za opuštění skladiště se považuje překročení libovolného okraje mapy.

Soutěžní úloha. Vaším úkolem je napsat testovací software, který na základě příkazové sekvence pro robota a mapy skladu ve tvaru čtvercové sítě spočítá, z kolika políček lze sklad opustit prováděním zadané posloupnosti příkazů. Připomeňme, že po provedení posledního příkazu této posloupnosti robot začne znovu provádět příkazy posloupnosti od začátku. Navíc pro každé políčko, ze kterého robot skladiště opustí, program určí počet příkazů, které robot provede do opuštění skladiště (do tohoto počtu se započítávají i ty příkazy, které robot nemohl vykonat kvůli překážce v cestě).

Formát vstupu: Na prvním řádku vstupního souboru `akce.in` je délka L ($1 \leqq L \leqq 500$) sekvence příkazů pro robota; samotná sekvence příkazů pro robota je pak uvedena na druhém řádku. Druhý řádek tedy obsahuje posloupnost čítající L následujících znaků:

- ▷ `L` = otočení robota o 90 stupňů doleva,
- ▷ `R` = otočení robota o 90 stupňů doprava,
- ▷ `+` = posun robota o jedno políčko vpřed vzhledem ke směru jeho natočení,
- ▷ `-` = posun robota o jedno políčko vzad vzhledem ke směru jeho natočení.

Na třetím řádku vstupu jsou pak čísla S a V, oddělená jednou mezerou. S ($1 \leqq S \leqq 500$) udává šířku mapy, tedy počet políček od západu k východu (vodorovně), a V ($1 \leqq V \leqq 500$) reprezentuje výšku mapy, tj. počet políček od severu k jihu (svisle).

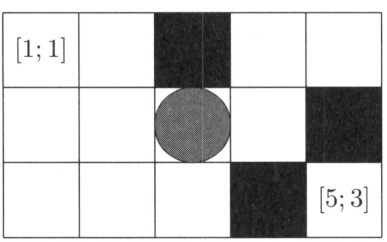
Na dalších V řádcích následuje samotná mapa. Každý řádek obsahuje S znaků tečka (`.`) nebo mříž (`#`), kde tečky představují volná políčka a mříže překážky. První znak prvního řádku mapy odpovídá jejímu severozápadnímu rohu.

Formát výstupu: Na první řádek výstupního souboru `akce.out` vypište počet políček, ze kterých robot skladiště opustí. Každý z následujících V řádků pak obsahuje S čísel: j-té číslo na i-tém řádku udává počet robotem vykonaných příkazů do okamžiku, kdy robot opustí skladiště. Pokud z odpovídajícího políčka nelze skladiště opustit nebo se na něm nachází překážka, je toto číslo rovno nule.

Příklad:

Vstup:	*Výstup*:
`8`	`9`
`+R++LL+R`	`1 1 0 1 1`
`5 3`	`9 9 0 4 0`
`..#..`	`17 0 0 0 3`
`....#`	
`...#.`	

Plán skladiště z příkladu:



Z políčka se šedou značkou robot neunikne, pokud dostane posloupnost příkazů uvedenou v příkladu.