Rozhledy matematicko-fyzikální

58. ročník Matematické olympiády, úlohy domácího kola kategorie P

Rozhledy matematicko-fyzikální, Vol. 83 (2008), No. 2, 37–45

Persistent URL: http://dml.cz/dmlcz/146248

58. ročník Matematické olympiády, úlohy domácího kola kategorie P

Od nadcházejícího 58. ročníku MO se mění způsob odevzdávání úloh domácího kola kategorie P. Úlohy P-I-1 a P-I-2 jsou prakticky zaměřené a vaším úkolem v nich je vytvořit a odladit efektivní program v jazyce Pascal, C nebo C++. Řešení těchto dvou úloh budete odevzdávat ve formě zdrojového kódu přes webové rozhraní přístupné na stránce `http://mo.mff.cuni.cz/submit/`, kde též naleznete další informace. Odevzdaná řešení budou automaticky vyhodnocena pomocí připravených vstupních dat a výsledky vyhodnocení se krátce po odevzdání dozvíte. Pokud váš program nezíská plný počet bodů, můžete své řešení opravit a znovu odevzdat.

Úlohy P-I-3 a P-I-4 jsou teoretické. Vaším úkolem je nalézt efektivní algoritmus řešící zadaný problém. Řešení úlohy se tedy skládá z popisu navrženého algoritmu, zdůvodnění jeho správnosti (funkčnosti) a odhadu časové a paměťové složitosti. Součástí řešení je i zápis algoritmu ve formě zdrojového kódu nebo pseudokódu v úloze P-I-3 a v jazyce zásobníkových počítačů v úloze P-I-4. Svá řešení můžete odevzdat ve formě souboru typu PDF přes výše uvedené webové rozhraní nebo zaslat poštou na adresu:

Matematická olympiáda – kategorie P
KSVI MFF UK
Malostranské náměstí 25
118 00 Praha 1

Řešení všech úloh můžete odevzdávat do 15. listopadu 2008. Opravená řešení úloh P-I-3 a P-I-4 dostanete zpět prostřednictvím krajských komisí MO. Seznam postupujících do krajského kola najdete na webových stránkách olympiády na adrese `http://mo.mff.cuni.cz/`, kde jsou také k dispozici další informace o kategorii P.

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 ($1 \leq N \leq 100\,000$), 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 Thespisův 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 $1 \leq R \times S \leq 1\,000\,000$. 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:
```
ZZVVVVVZZZZZZZZVVVVVVVVVVVVVZZZZZZZZZZZZZZZZZ
```

Vstup:	Výstup:

`3 1`
`*`
`.`
`$`
	`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, $1 \leq i \leq j \leq N$, 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 $\mathcal{O}(1)$.

Příklad: Pro posloupnost $3, 5, -2, 5, 4$ délky $N = 5$ a pro $X = 7$ je hledanou podposloupností $5, -2, 5$ (tedy $i = 2$ a $j = 4$) se součtem 8. Jiným možným řešením je podposloupnost $3, 5$. Váš program nemusí nalézt všechna nejlepší řešení – stačí, když vypíše libovolné z nich.

P-I-4 Stackal

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 zjišťující, 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)*

* * * * *

Štyri farby stačia

Ofarbovanie jednotlivých súvislých štátov na mape tak, aby susedné štáty boli vyfarbené vždy inou farbou, je zvyčajne detská hra. Ak si to vyskúšate, zistíte, že vždy vám vystačia štyri farby. Ale viete to dokázať? Matematici si z toho urobili problém teórie grafov. Farebné mapy previedli na farbenie uzlov grafu tým, že z každého štátu spravia uzly grafu, ktorého hrany dostaneme tak, že spojíme dva uzly práve vtedy, ak sú príslušné štáty susedné. Tým sa problém štyroch farieb stal úlohou teórie grafov. Od roku 1852, keď sa dôkazom začali zaoberať vážení matematici (napr. A. Morgan, W. R. Hamilton, C. S. Peirce, A. Cayley, neskôr aj A. B. Kempe, P. J. Heawood, P. G. Tait, G. D. Birkhoff, O. Veblen a ďalší) sa nedarilo urobiť presvedčivý matematický dôkaz. Zdanlivých dôkazov pravdivosti hypotézy o štyroch farbách, v ktorých sa neskôr našli chyby, nebolo málo. Až roku 1976 K. Appel a W. Haken z univerzity Illinois, po tri a polročnej príprave metód i počítačového programu a riešení za pomoci elektronických počítačov (asi 1 200 hodín strojového času), keď rozlíšili asi 2 000 rôznych prípadov, presvedčili matematikov, že ich postup je dôkazom. Niektorí ortodoxní matematici majú však k uplatneniu počítačového programu v dôkazových matematických postupoch trvalé výhrady. Matematika však nemôže vylúčiť informatické postupy computer science. Svet matematických disciplín sa rozšíril do oblasti počítačových programov.

D. Jedinák