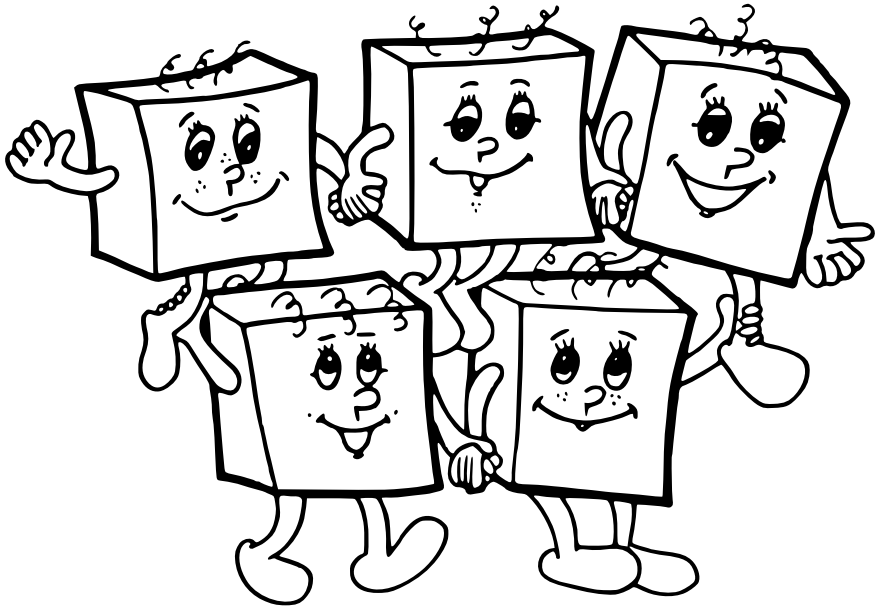
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH



dvadsiaty tretí ročník

školský rok 2007/08

riešenia krajského kola
kategória B

- **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://www.ksp.sk/oi/`.

B-II-1 Zberateľky

Ako zistiť, či sú v úseku $B[1..K]$ nejaké z hľadaných čísel? Ľahko. Stačí porovnať hodnoty $B[K]$ a $A[K]$. Ak sú rovnaké, obe hľadané čísla sú niekde ďalej v poli B. Ak je hodnota v poli B menšia, aspoň jedna z hľadaných hodnôt leží v skúmanom úseku.

Vieme toho dokonca povedať aj viac. Tým, že sme pridali dva nové prvky, sa niektoré z pôvodných prvkov posunuli „doprava“ (na pozíciu s väčším indexom), a to najviac o 2. Teda hodnota $A[i]$ je v poli B na pozícii i, $i+1$ alebo $i+2$. A podľa toho, o koľko pozícii je vpravo, vieme, koľko z hľadaných nových prvkov je v poli B naľavo od nej.

Pomocou tejto úvahy vieme ľahko zistiť, koľko hľadaných čísel je v úseku $B[1..K]$: Pozrime sa na hodnotu $B[K]$. Sú dve možnosti: Ak je aj v poli A, tak musí nutne byť na jednej z pozícii $A[K-2]$, $A[K-1]$ alebo $A[K]$. A podľa toho, na ktorej z nich je, vieme, koľko hľadaných čísel je naľavo. Ak táto hodnota v poli A nie je, je to jedno z hľadaných čísel. Zopakovaním úvahy pre $B[1..(K-1)]$ zistíme, či je v tejto časti poľa aj druhé hľadané číslo.

Podobne vieme zistiť počet hľadaných čísel v ľubovoľnom úseku $B[k..l]$ – spočítame ho jednoducho ako (počet v $B[1..l]$) mínus (počet v $B[1..(k-1)]$).

Všimnite si, že na zodpovedanie takejto otázky nám stačí konštantný počet operácií.

Vzorové riešenie bude používať myšlienku binárneho vyhľadávania.

Majme najskôr jednoduchší problém: Máme úsek poľa B, o ktorom vieme, že v ňom je práve jedno hľadané číslo. Ako ho nájsť?

Ak má úsek dĺžku 1, už sme ho našli. Ak je dlhší, rozdeľme ho približne na polovice. Teraz si vieme v konštantnom čase spočítať, v ktorej polovici hľadané číslo je. Dostali sme tú istú úlohu, len na úseku polovičnej dĺžky. Tento postup teda opakujeme dovtedy, kým nám nezostane už len úsek dĺžky 1.

Ak sme teda začali s úsekom dĺžky K, po približne $\log_2 K$ krokoch nájdeme hľadané číslo.

A čo teraz s pôvodným problémom? Majme teda úsek poľa B, o ktorom

vieme, že sú v ňom obe hľadané čísla. Opäť ho rozdelíme na polovice. Môžu nastať dva prípady:

Ak sú obe čísla v tej istej polovici, máme pôvodný problém a úsek polovičnej dĺžky, pokračujeme ďalej v delení na polovice.

Ak je každé číslo v inej polovici, dostali sme dva jednoduchšie problémy, ktoré už vieme riešiť.

Časová zložitosť tohoto riešenia je $O(\log N)$ – v najhoršom prípade rozdelíme kus poľa na dve časti $2 \log_2 N$ krát.

Listing programu:

```
var A,B : array[1..10000] of longint;
    N : longint;


{ spocita kolko hladanych cisel je v B[1..(kon-1)] }
function spocitaj(kon: longint) : longint;
begin
   if (kon=1) then begin spocitaj:=0; exit; end;
   if (kon>2) and (A[kon-3]=B[kon-1]) then begin spocitaj:=2; exit; end;
   if (kon>1) and (kon<N+3) and (A[kon-2]=B[kon-1]) then
      begin spocitaj:=1; exit; end;
   if (kon<N+2) and (A[kon-1]=B[kon-1]) then begin spocitaj:=0; exit; end;
   { ak sa dostal sem, B[kon-1] je jedno z hladanych cisel }
   spocitaj := 1 + spocitaj(kon-1);
end;

{ spocita kolko hladanych cisel je v B[zac..(kon-1)] }
function spocitaj(zac, kon: longint) : longint;
begin spocitaj := spocitaj(kon) - spocitaj(zac); end;

procedure najdi_jeden(zac, kon: longint; var x : longint);
var stred,prvy : longint;
begin
   if (kon-zac=1) then begin x:=B[zac]; exit; end;
   stred:=(zac+kon) div 2;
   prvy:=spocitaj(zac,stred);
   if (prvy=0) then najdi_jeden(stred,kon,x) else najdi_jeden(zac,stred,x);
end;

procedure najdi_dva(zac, kon: longint; var x,y : longint);
var stred,prvy : longint;
begin
   stred:=(zac+kon) div 2;
   prvy:=spocitaj(zac,stred);
   if (prvy=0) then begin najdi_dva(stred,kon,x,y); exit; end;
   if (prvy=2) then begin najdi_dva(zac,stred,x,y); exit; end;
   najdi_jeden(zac,stred,x);
```

```
   najdi_jeden(stred,kon,y);
end;

{ pre nazornost este priklad programu, ktory nasu proceduru vola }
var i,x,y : longint;
begin
   read(N);
   for i:=1 to N do read(A[i]);
   for i:=1 to N+2 do read(B[i]);
   najdi_dva(1,N+3,x,y);
   writeln(x,' ',y);
end.
```

B-II-2 Krtkovou norou tam a späť

Použijeme postup z domáceho kola. Do každej chodbičky pridáme nové brlôžky, ktoré ju rozdelia na úseky dĺžky 1.

Prehľadávaním do šírky z brlôžku 1 teraz vieme v čase $O(M+N)$ pre každý brlôžok x spočítať dĺžku $d(1,x)$ najkratšej cesty doň. Okrem iného sa teda dozvieme dĺžku $d(1,N)$ cesty, ktorou má Vítek ísť.

Spustíme ešte jedno prehľadávanie do šírky, tentokrát z brlôžku N. Tým sa dozvieme dĺžky najkratších ciest medzi brlôžkom N a ľubovoľným brlôžkom x.

Ako teraz zistiť, či vie Vítek ísť cez konkrétny brlôžok x, ak ide najkratšou cestou do N? Jednoducho. Pozrime sa, aká najkratšia môže byť jeho cesta, ak pôjde cez x. V prvom rade musí prísť najkratšou cestou z 1 do x. No a následne musí prejsť, opäť najkratšou cestou, z x do N.

Ak teda Vítek pôjde najkratšou cestou, ktorá vedie cez brlôžok x, prejde trasu dĺžky $d(1,x)+d(x,N)$. Ak je táto hodnota rovná $d(1,N)$, Vítek cez x ísť môže. A naopak, ak je dĺžka cesty cez x väčšia od $d(1,N)$, Vítek cez x ísť nemôže.

Takto vieme teda o každom brlôžku v konštantnom čase zistiť, či cez neho môže Vítek ísť. Táto časť algoritmu má teda časovú zložitosť $O(N)$, a keďže prehľadávania trvajú dlhšie, celková časová zložitosť je $O(M+N)$.

Listing programu:

```
var G : array[1..1000,1..1000] of longint; { pre kazdy vrchol zoznam susedov }
    stupne : array[1..1000] of longint; { stupne vrcholov }
    N,M : longint; { pocet vrcholov a hran }
```

```
    dist1, distN : array[0..1000] of longint; { vzdialenosti od 1 a od N }

procedure nacitaj;
var i,a,b,h,kde,dalsi : longint;
begin
   readln(N,M);
   dalsi := N+1; { cislo, ktore dostane dalsi brlozok }
   for i:=1 to M do begin
      readln(a,b,h);
      kde := a;
      while (h>1) do begin { kopeme novy brlozok }
         inc(stupne[kde]); inc(stupne[dalsi]);
         G[kde][ stupne[kde] ] := dalsi;
         G[dalsi][ stupne[dalsi] ] := kde;
         kde := dalsi; inc(dalsi);
         dec(h);
      end;
      inc(stupne[kde]); inc(stupne[b]);
      G[kde][ stupne[kde] ] := b;
      G[b][ stupne[b] ] := kde;
   end;
end;

{ spusti prehladavanie z brlozka "odkial", v poli "vzdialenost" je vystup }
procedure prehladaj(odkial : longint; var vzdialenost : array of longint);
var fronta : array[1..1000] of longint;
    bol : array[1..1000] of boolean;
    zac,kon,kde,i,kam : longint;
begin
   zac:=1; kon:=2; fronta[1]:=odkial;
   fillchar(bol,sizeof(bol),0);
   bol[odkial]:=true; vzdialenost[odkial]:=0;
   while (zac < kon) do begin
      kde := fronta[zac]; inc(zac);
      for i:=1 to stupne[kde] do begin
         kam := G[kde][i];
         if (not bol[kam]) then begin
            bol[kam]:=true; vzdialenost[kam]:=vzdialenost[kde]+1;
            fronta[kon]:=kam; inc(kon);
         end;
      end;
   end;
end;

var i : longint;
begin
   nacitaj;
   prehladaj(1,dist1);
   prehladaj(N,distN);
   for i:=1 to N do
      if dist1[i]+distN[i] = dist1[N] then writeln(i);
end.
```

B-II-3 Opravovanie XML

Na pamätanie si otvorených tagov môžeme použiť, podobne ako v riešeniach domáceho kola, zásobník. Hlavným problémom bude, ako pri spracúvaní zatváracieho tagu zistiť, či nastal prípad 2 (máme niekde v zásobníku zodpovedajúci otvárací tag) alebo prípad 3 (takýto otvárací tag nemáme).

Základné riešenie

Každý otvárací tag vložíme na zásobník. Keď nám príde zatvárací tag, prezrieme obsah zásobníka, a podľa toho, či ho tam nájdeme, buď upravíme obsah zásobníka, alebo zahodíme spracúvaný zatvárací tag.

Ak je na vstupe N tagov, takéto riešenie môže spraviť až rádovo N^2 porovnaní tagov, teda jeho časová zložitosť je $O(N^2)$. (Príklad zlého vstupu: najskôr $N/2$ otváracích tagov, potom $N/2$ zatváracích tagov, ktoré nemajú zodpovedajúci otvárací tag.)

Lepšie riešenie

Aby sme vedeli toto riešenie zefektívniť, potrebujeme lepší spôsob, ako zistiť, či je daný tag práve otvorený. Existuje viacero spôsobov, ako na to.

Jednou možnosťou napríklad je pamätať si otvorené tagy okrem zásobníka aj vo vyváženom binárnom vyhľadávacom strome. Takto vieme každý tag spracovať v čase $O(\log N)$. Takéto riešenie je však dosť náročné na implementáciu. Ukážeme si preto aj jednoduchší spôsob, ako túto časovú zložitosť dosiahnuť.

Všetky názvy tagov z dokumentu (otváracích aj zatváracích) si uložíme do poľa. Pole utriedime a vynecháme z neho opakujúce sa záznamy. Teraz si môžeme tagy očíslovať číslami od 1 do nejakého M (kde určite $M \leq N$) – každému tagu priradíme index, na ktorom je jeho názov v tomto utriedenom poli. Kedykoľvek, keď budeme chcieť z názvu tagu určiť jeho číslo, vieme ho zistiť v čase $O(\log N)$ binárnym vyhľadávaním.

Keď sme sa takto pripravili, môžeme začať samotné spracúvanie tagov. Budeme si pamätať otvorené tagy v zásobníku, a navyše si budeme v pomocnom poli ku každému názvu tagu pamätať, koľkokrát je práve otvorený.

Otvárací tag spracujeme ľahko: Zistíme si jeho číslo, zvýšime príslušnú hodnotu v poli a vložíme ho na vrch zásobníka.

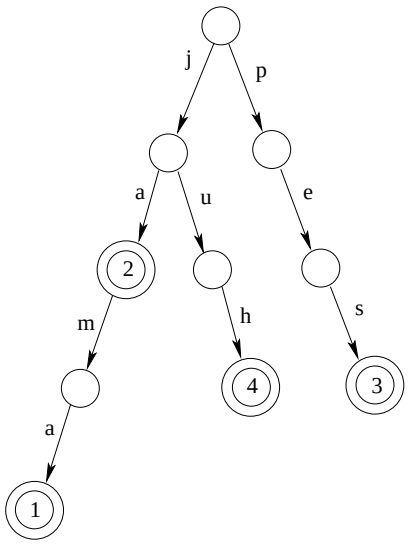
A ako je to so zatváracími tagmi? Zistíme si jeho číslo a pozrieme na príslušné miesto do pomocného poľa. Ak je tam nula, takýto tag nie je otvorený, práve spracúvaný tag teda zahodíme. Ak tam nie je nula, ideme zatvárať otvorené tagy, kým nezavrieme aj tento.

Ako spočítať časovú zložitosť tohoto postupu? Každý otvárací tag raz vložíme na zásobník a raz ho odtiaľ vyberieme. Toto je nanajvýš $2N$ operácií. každú z nich vieme spraviť v čase $O(\log N)$, lebo musíme vždy upraviť aj hodnotu v pomocnom poli, a na to potrebujeme „preložiť“ tag na číslo.

Okrem toho už len každý zatvárací tag potrebujeme preložiť na číslo, keď chceme zistiť, či máme v zásobníku k nemu pár. Takto budeme prekladať najviac N tagov. Preto celková časová zložitosť tohoto riešenia je $O(N \log N)$.

Vzorové riešenie

Zatiaľ sme nijako nevyužili, že názvy tagov sú reťazce, ktoré sa skladajú z písmeniek. Vďaka tomu existujú aj šikovnejšie spôsoby, ako si pamätať množinu reťazcov. Popíšeme jeden z nich, tzv. písmenkový strom (po anglicky *trie*). Písmenkový strom je zakorenený strom, v ktorom každý vrchol má najviac 26 synov, a hrany do synov sú označené rôznymi písmenkami (od `a` po `z`). Každá cesta z koreňa nadol zodpovedá slovu, ktoré si „prečítame“ na hranách, po ktorých ideme.



Písmenkový strom pre `jama`, `ja`, `pes` *a* `juh`.

Písmenkový strom sa dá použiť na uloženie množiny slov. Začneme so stromom, ktorý obsahuje len koreň a nič viac. Teraz budeme postupne po jednom

pridávať slová. Pri pridávaní slova vytvoríme všetky vrcholy, ktoré treba na to, aby sme si mohli na ceste z koreňa dole „prečítať“ toto slovo. (Teda pri vkladaní „`jama`“ pridáme 4 nové vrcholy, pri následnom vkladaní „`ja`“ žiadny.) Vo vrchole, kde slovo končí, si zapíšeme jeho poradové číslo.

Vzorové riešenie teda bude vyzerať nasledovne:

- Načítame zo vstupu postupnosť tagov.
- Názvy tagov postupne vkladáme do písmenkového stromu. Vždy, keď vložíme nový názov (ktorý tam dovtedy nebol), priradíme mu nové číslo.
- Postupne spracúvame postupnosť tagov ako v predchádzajúcom riešení, pričom na preklad tagov na čísla používame náš písmenkový strom.

Čas potrebný na vloženie nového reťazca do písmenkového stromu je priamo úmerný dĺžke dotyčného reťazca. Podobne aj čas potrebný na nájdenie čísla priradeného reťazcu. V našom prípade (názvy tagov majú dĺžku nanajvýš 8) vieme teda každý tag spracovať v konštantnom čase.

Preto je celková časová zložitosť vzorového riešenia $O(N)$.

Listing programu:

```
program oprava_xml_pismenkovy_strom;

type pvrchol = ^tvrchol;
     tvrchol = record
        cislo : longint;
        deti : array['a'..'z'] of pvrchol;
     end;

procedure inicializuj(co: pvrchol);
var ch : char;
begin co^.cislo:=-1; for ch:='a' to 'z' do co^.deti[ch]:=nil; end;

{ zacni vo vrchole "koren", zlez dole cestou zodpovedajucou "slovo" }
function zlez(slovo: string; koren: pvrchol) : pvrchol;
var i : integer;
begin
   for i:=1 to length(slovo) do begin
      if (koren^.deti[slovo[i]]=nil) then begin
         new( koren^.deti[slovo[i]] );
         inicializuj( koren^.deti[slovo[i]] );
      end;
      koren := koren^.deti[slovo[i]];
   end;
   zlez := koren;
end;
```

```
procedure vloz(slovo: string; koren : pvrchol; var N : longint);
begin
   koren := zlez(slovo,koren);
   if koren^.cislo >= 0 then exit;
   koren^.cislo := N;
   inc(N);
end;

function preloz(slovo: string; koren : pvrchol) : longint;
begin
   koren := zlez(slovo,koren);
   preloz := koren^.cislo;
end;

{ skusi nacitat dalsi tag, ak sa podarilo, vyplni premenne a vrati true }
function nacitaj_tag(var nazov : string; var zaciatok : boolean) : boolean;
var ch : char;
begin
   ch:='?';
   while ch<>'<' do begin
      if eof then begin nacitaj_tag:=false; exit; end;
      read(ch);
   end;
   nazov:=''; zaciatok:=true;
   read(ch);
   if (ch='/') then begin zaciatok:=false; read(ch); end;
   while true do begin
      if (ch='>') then break;
      nazov := nazov + ch;
      read(ch);
   end;
   nacitaj_tag:=true;
end;

{ nacita vstup do danych premennych }
procedure load(var N : longint; var tagy : array of string;
                                var typy : array of boolean);
var tag : string;
    typ : boolean;
begin
   N := 0;
   while (nacitaj_tag(tag,typ)) do
      begin tagy[N]:=tag; typy[N]:=typ; inc(N); end;
end;

var N,M,Z,i,cislo,cislo2,vysledok : longint;
    tagy, zasobnik : array[0..1000] of string;
    typy : array[0..1000] of boolean;
    pocty : array[0..1000] of longint;
    strom : pvrchol;

begin
   load(N,tagy,typy);
   new(strom); inicializuj(strom); M:=0;
   for i:=0 to N-1 do vloz(tagy[i],strom,M);
```

```
   Z := 0;
   vysledok := 0;
   for i:=0 to N-1 do begin
      cislo:=preloz(tagy[i],strom);
      if (typy[i]) then begin { otvaraci tag, vlozime na zasobnik }
         zasobnik[Z]:=tagy[i];
         inc(Z);
         inc(pocty[cislo]);
      end else begin { zatvaraci tag }
         if (pocty[cislo]>0) then begin { mame ho, ideme vyberat }
            while true do begin
               cislo2:=preloz(zasobnik[Z-1],strom);
               dec(pocty[cislo2]);
               dec(Z);
               if (cislo=cislo2) then break;
               inc(vysledok);
            end;
         end else inc(vysledok); { nemame ho, zahodime }
      end;
   end;
   inc(vysledok,Z); { vyhadzeme co ostalo v zasobniku }
   writeln(vysledok);
end.
```

B-II-4 Rákosie sa vracia

Podúloha a)

Použijeme podobný postup ako v jednom z príkladov v študijnom texte. Genetické pravidlá nášho rákosia budú najskôr generovať živé bunky A, B a C, pričom počet C bude stále rovnaký ako počet A a B dokopy. Tieto živé bunky si budú môcť ľubovoľne vymieňať miesta. A keď sa už bunka rozhodne, že na nejakom mieste chce skončiť, zmení sa na zodpovedajúcu mŕtvu bunku.

Pravidlá teda budú vyzerať napríklad takto:

$$Z \to ACZ \mid BCZ \mid \varepsilon \tag{1}$$

$$AB \to BA, \qquad BA \to AB \tag{2}$$

$$AC \to CA, \qquad CA \to AC \tag{3}$$

$$BC \to CB, \qquad CB \to BC \tag{4}$$

$$A \to a, \qquad B \to b, \qquad C \to c \tag{5}$$

Zjavne pre každé steblo, ktoré môže narásť, platí, že počet a plus počet b (veľkých aj malých dokopy) je rovný počtu c (opäť veľkých aj malých dokopy).

Toto je preto, že každé pravidlo túto rovnosť zachováva. Pravidlami (2) až (4) nové bunky nepribúdajú, pravidlami (5) sa len živá bunka zmení na zodpovedajúcu mŕtvu, čo rovnosť zjavne nepokazí. No a pri použití pravidla $Z \to ACZ$ alebo $Z \to BCZ$ pribudne jedno c a jedno iné písmenko, čo rovnosť tiež nepokazí. Preto určite každé mŕtve steblo tejto odrody obsahuje rovnako veľa c ako a a b dokopy.

Naopak, majme konkrétny reťazec písmen a, b, c, v ktorom je rovnako veľa c ako a a b dokopy. Nech α je počet písmen a a β počet písmen b v ňom. Keď použijeme α krát pravidlo $Z \to ACZ$, potom β krát pravidlo $Z \to BCZ$ a potom raz $Z \to \varepsilon$, dostaneme reťazec, v ktorom sú už počty A, B a C také, aké majú byť počty a, b a c na konci. Pomocou pravidiel 2-7 vieme všetky A, B a C preusporiadať do správneho poradia. Na záver ich pravidlami 8-10 prepíšeme na malé písmená a sme hotoví. Preto určite vieme mať mŕtve steblo s ľubovoľným požadovaným reťazcom.

Podúloha b)

Myšlienka bude nasledovná: Vygenerujeme si reťazec písmen X a reťazec písmen Y, každý z nich dĺžky aspoň 2. Teraz by sme chceli ich dĺžky „vynásobiť“. To spravíme takto: Škrtneme jedno X a za každé Y pridáme do reťazca jedno a. Toto opakujeme, kým sa nám všetky X neminú. Na záver už len zmažeme všetky Y.

Aby sme sa v tom celom nezamotali, použijeme ešte niekoľko pomocných symbolov ako zarážky – B (ako begin) na začiatku, E (ako end) na konci.

$$Z \to BXXP \tag{1}$$
$$P \to XP \mid YYQ \tag{2}$$
$$Q \to YQ \mid SE \tag{3}$$

Pomocou postupného aplikovania pravidiel 1, 2 a 3 zo symbolu Z postupne dostaneme reťazec tvaru $BX \ldots XY \ldots YSE$. Vieme vyrobiť ľubovoľný taký reťazec, v ktorom sú aj X, aj Y aspoň dve.

$$aS \to Sa \tag{4}$$
$$YS \to SY \tag{5}$$

$$XS \rightarrow T \quad (6)$$
$$Ta \rightarrow aT \quad (7)$$
$$TY \rightarrow YaT \quad (8)$$
$$TE \rightarrow SE \quad (9)$$

S (ako späť) je symbol, ktorý sa vracia späť (pravidlá 4 a 5) na začiatok nájsť ďalšie X. Ak sa mu to podarí, použitím pravidla 6 ho vymaže a zmení sa na T (ako tam). Symbol T použitím pravidiel 7 a 8 presúvame doprava, pričom vždy, keď stretneme Y, pridáme do reťazca nové a.

$$BS \rightarrow F \quad (10)$$
$$Fa \rightarrow aF \quad (11)$$
$$FY \rightarrow F \quad (12)$$
$$FE \rightarrow \varepsilon \quad (13)$$

Ak S prejde celým reťazcom až ku B a žiadne X už nenájde, násobenie sa skončilo a začína upratovanie. Zmeníme použitím pravidla 10 S na F (a jedným krokom zahodíme už nepotrebnú zarážku B). Použitím pravidiel 11 a 12 prechádza F doprava reťazcom, písmená a necháva na pokoji a zahadzuje všetky Y. Keď F príde na koniec slova, pravidlom 13 zahodíme posledné dva pomocné symboly F a E a skončili sme.

Zjavne keď si zvolíme ľubovoľné zložené číslo $z = mn$, vieme reťazec a^z vyrobiť tak, že pravidlami 1-3 vyrobíme reťazec BX^mY^nSE a odsimulujeme vyššie popísaný postup.

Na druhej strane, v okamihu, kedy použijeme pravidlo $Q \rightarrow SE$, je celé pokračovanie jednoznačne určené. V každom okamihu môžeme spraviť práve jednu jedinú zmenu reťazca. Tento postup, ako sme už vyššie ukázali, skončí s tým, že máme reťazec a^z pre nejaké zložené číslo z. Preto v našej odrode rákosia určite nič zlé nenarastie.

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE

DVADSIATY TRETÍ ROČNÍK OLYMPIÁDY V INFORMATIKE

Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX