MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

54. ročník, školský rok 2004/2005

Riešenia úloh 3. kola kategórie P

2. súťažný deň

P-III-4

Jednotlivým písmenám v zaklínadle priradíme čísla zľava doprava podľa nasledujúceho magického receptu:

- Písmeno α, dostane číslo 1, ak sa pred ním v zaklínadle nevyskytuje iné písmeno α, ani písmená α^- a α^+, kde $\alpha^\pm$ je písmeno v abecede tesne pred/za písmenom α. Napríklad písmeno c dostane číslo 1, ak sa pred ním v zaklínadle nevyskytuje žiadne z písmen b, c, d.
- V opačnom prípade písmenu α priradíme maximum z čísel priradených písmenám α^-, α a α^+, ktoré sa pred ním vyskytujú v zaklínadle, zväčšené o 1. Napríklad písmeno c dostane číslo 4, ak sa pred ním vyskytuje písmeno b s číslom 3 a žiadne z písmen c a d.

V našom algoritme bude kľúčové nasledujúce pozorovanie:

Tvrdenie: Dve zaklínadlá sú ekvivalentné práve vtedy, keď čísla priradené každému z písmen $a \dots z$ tvoria rovnakú postupnosť.

Skôr ako si toto tvrdenie dokážeme, demonštrujme si jeho platnosť na zaklínadlách zo zadania úlohy:

```
121314   111234
abraka   krabaa
```

Postupnosti čísel priradených písmenám a, b, k a r sú nasledujúce: 134, 2, 1 a 1. Podľa tvrdenia, ktoré sme zatiaľ nedokázali, sú preto obe zaklínadlá ekvivalentné.

```
11213   12131
dabra   badar
```

Čísla priradené písmenu a v prvom zaklínadle tvoria postupnosť 13, zatiaľ čo v druhom tvoria postupnosť 23. Zaklínadlá teda podľa nášho tvrdenia nie sú ekvivalentné.

Teraz si vyššie uvedené tvrdenie dokážeme:

Dôkaz: Ak v zaklínadle vymeníme dve susedné písmená, ktoré nie sú v abecede tesne vedľa seba, čísla priradené týmto písmenám se nezmenia. Teda v ekvivalentných zaklínadlách musia byť postupnosti čísel priradených každému z písmen rovnaké.

Obrátenú implikáciu, t.j., že ak čísla priradené každému z písmen $a \ldots z$ tvoria rovnakú postupnosť, tak sú obe zaklínadlá ekvivalentné, dokážeme indukciou od dĺžky zaklínadla. Tvrdenie zrejme platí pre jedno- a dvojpísmenové zaklínadlá. Predpokladajme teraz, že obe zaklínadlá sú tvorené aspoň troma písmenami. Všimnime si v prvom zaklínadle písmeno, ktoré má priradené najväčšie číslo, ak je takých viac, tak to z nich, ktoré je prvé v abecede. Označme ho α. Podobne v druhom zaklínadle nájdeme písmeno β. Pretože postupnosti čísel priradených rovnakým písmenám sú zhodné v oboch zaklínadlách, musí platiť $\alpha = \beta$.

Pretože α je písmeno v prvom zaklínadle s najväčším číslom, nenasleduje za ním už žiadne z písmen α^-, α a α^+, a môžeme ho teda postupnosťou výmien premiestniť na koniec zaklínadla. Podobne môžeme na koniec druhého zaklínadla premiestniť písmeno β. Keďže výmeny dvoch susedných písmen, ktoré nie sú v abecede tesne vedľa seba, nemenia čísla priradené jednotlivým písmenám, sú postupnosti čísel priradených jednotlivým písmenám v oboch nových zaklínadlách zhodné. Táto vlastnosť ostane zachovaná i po odobratí písmen α a β. Na takto získané kratšie zaklínadlá použijeme indukčný predpoklad. Pretože sú skrátené zaklínadlá ekvivalentné, sú ekvivalentné i zaklínadlá s písmenami α a β na konci, a teda i pôvodné zaklínadlá.

Práve dokázané tvrdenie nám dáva návod, ako otestovať, či sú dve zaklínadlá ekvivalentné. Najskôr podľa vyššie uvedených pravidiel priradíme každému písmenu v zaklínadlách číslo a potom skontrolujeme, či sa postupnosti čísel priradených rovnakým písmenám zhodujú. Aby sme sa pri tejto kontrole vyhli viacnásobnému prechodu zadanými zaklínadlami, pri priraďovaní čísel si zároveň vytvoríme postupnosti pre jednotlivé písmená. Naviac, aby sme sa vyhli (pomalej) dynamickej alokácií pamäti, budeme si v pomocnom poli uchovávať odkazy na nasledujúce rovnaké písmeno v zadanom zaklínadle. Časová i pamäťová zložitosť takéhoto algoritmu je pre dvojicu N-písmenových zaklínadiel $O(N + A)$, kde A je počet písmen, ktoré sa môžu v zaklínadlách vyskytovať (v našom prípade $A = 26$).

Listing programu:

```
program zaklinadla;

const MAX=1000000;
var zaklinadla:array[1..2,1..MAX] of char; { zadané zaklínadlá }
```

```
    delka:longint; { dĺžka zaklínadiel }
    cisla:array[1..2,1..MAX] of longint;
      { čísla priradené jednotlivým písmenám v zaklínadlách }
    prvni:array[1..2,'a'..'z'] of longint;
      { prvý výskyt daného písmena v zaklínadle (0=nie je tam) }
    dalsi:array[1..2,1..MAX] of longint;
      { odkaz na ďalší výskyt rovnakého písmena v zaklínadle }
    vstup,vystup:text; { vstupný a výstupný súbor }

procedure nacti;
var i:longint;
    c:char;
begin
  for i:=1 to delka do
    begin
      read(vstup,zaklinadla[1][i],c,zaklinadla[2][i]);
      readln(vstup);
    end;
end;

procedure spocitej(z: integer);
var pozice:array['a'..'z'] of longint;
    i:longint;
    max:longint;
    c:char;
begin
  for c:='a' to 'z' do
    begin
      prvni[z][c]:=0;
      pozice[c]:=0;
    end;
  for i:=1 to delka do
    begin
      max:=0;
      if pozice[zaklinadla[z][i]]<>0 then
        max:=cisla[z][pozice[zaklinadla[z][i]]];
      if zaklinadla[z][i]<>'a' then
        if pozice[pred(zaklinadla[z][i])]<>0 then
          if cisla[z][pozice[pred(zaklinadla[z][i])]]>max then
            max:=cisla[z][pozice[pred(zaklinadla[z][i])]];
      if zaklinadla[z][i]<>'z' then
        if pozice[succ(zaklinadla[z][i])]<>0 then
          if cisla[z][pozice[succ(zaklinadla[z][i])]]>max then
            max:=cisla[z][pozice[succ(zaklinadla[z][i])]];
      cisla[z][i]:=max+1;
```

```
      if pozice[zaklinadla[z][i]]=0 then
        begin
          prvni[z][zaklinadla[z][i]]:=i;
          pozice[zaklinadla[z][i]]:=i;
        end
      else
        begin
          dalsi[z][pozice[zaklinadla[z][i]]]:=i;
          pozice[zaklinadla[z][i]]:=i;
        end;
      dalsi[z][i]:=0;
    end;
end;

function porovnej:boolean;
var c:char;
    i1,i2:longint;
begin
  spocitej(1);
  spocitej(2);
  for c:='a' to 'z' do
    begin
      i1:=prvni[1][c]; i2:=prvni[2][c];
      while (i1<>0) and (i2<>0) do
        begin
          if cisla[1][i1]<>cisla[2][i2] then
            begin
              porovnej:=false; exit
            end;
          i1:=dalsi[1][i1];
          i2:=dalsi[2][i2];
        end;
      if (i1<>0) or (i2<>0) then
        begin
          porovnej:=false; exit
        end;
    end;
  porovnej:=true;
end;

begin
  assign(vstup,'magia.in');
  assign(vystup,'magia.out');
  reset(vstup);
  rewrite(vystup);
```

```
  readln(vstup,delka);
  while delka<>0 do
    begin
      nacti;
      if not porovnej then write(vystup,'nie ');
      writeln(vystup,'su ekvivalentne');
      readln(vstup,delka);
    end;
  close(vstup);
  close(vystup);
end.
```

P-III-5

Úlohu si najskôr preformulujeme do reči teórie grafov: máme daný graf G a chceme zistiť, či (a ako) sa dajú jeho hrany rozdeliť do dvojíc tak, aby hrany v každej dvojici mali spoločný vrchol. Úloha zrejme nemá riešenie, ak je hrán nepárny počet. V ostatných prípadoch ukážeme, ako nájdeme hľadané rozdelenie.

Budeme graf prehľadávať do hĺbky a vždy pred návratom z každého vrcholu spravíme nasledujúce: Vezmeme všetky doposiaľ nespárované hrany susediace s aktuálnym vrcholom. Ak je hrán nepárny počet, vynecháme hranu vedúcu k otcovi (tá musí byť doposiaľ nespárovaná). Teraz máme párny počet hrán a môžeme ich teda ľubovoľne spárovať. Po spárovaní hrán môžeme ukončiť spracovanie aktuálneho vrcholu a vrátiť sa k otcovi. Týmto spôsobom sa nám muselo podariť spárovať všetky hrany, pretože pri každom vrchole sme nechali nespárovanú nanajvýš jednu hranu – ale tá viedla k otcovi a bola nutne spárovana pri spracovávaní otca.

Uvedomte si, že ani pri poslednom spracovávanom vrchole problém nastať nemohol. Teoreticky by mohla nastať situácia, v ktorej by neexistovala hrana, ktorú by sme mohli vynechať, ak by nám v tomto vrchole zostal nepárny počet nespárovaných hrán. Všetkých hrán však bol párny počet a aj po spárovaní niektorých hrán je počet nespárovaných hrán naďalej párny. Preto nám v poslednom vrchole musel zostať párny počet hrán, ktoré ľahko spárujeme. Algoritmus má časovú aj pamäťovú zložitosť $O(M + N)$, kde N je počet vrcholov grafu G a M je počet jeho hrán.

Listing programu:

```
program asfalter;
const
  INP = 'asfalt.in';    OUT = 'asfalt.out';
  MAXN = 10000;    MAXM = 40000;
```

```
type
  edge = record
    a, b : Integer;
    pair : Longint; {číslo hrany, s ktorou je v páre (-1, ak so žiadnou)}
  end;
  vertex = record
    start : Longint; {index, kde začínajú hrany z neho v zozname hrán}
    deg, visited : Integer;
  end;

var
  et : Array[1..MAXM] of edge; { Pole so všetkými hranami }
  ep : Array[1..2*MAXM] of Longint;
    { Čísla hrán zoradené podľa ich koncových vrcholov }
  v : Array[1..MAXN] of vertex; { Pole s vrcholmi }
  vn, en : Longint; { Počet vrcholov a hrán }

procedure read_input; { Načítaj vstup, vytvor graf }
var
  i : Longint;
  FI, FO : Text;
begin
  Assign(FI, INP); Reset(FI);
  Read(FI, vn, en);
  { Otestujeme, či je úloha riešiteľná }
  if en mod 2 = 1 then begin
    Assign(FO, OUT); Rewrite(FO);
    WriteLn(FO, 'Cesty sa nedaju vyasfaltovat.');
    Close(FO); Close(FI); Halt;
  end;

  for i := 1 to vn do begin v[i].deg := 0;  v[i].visited := 0;  end;
  for i := 1 to en do begin
    Read(FI, et[i].a, et[i].b);
    et[i].pair := -1;
    Inc(v[et[i].a].deg);   Inc(v[et[i].b].deg);
  end;
  Close(FI);

  { Utriedime pole s hranami }
  v[1].start := 1;
  for i := 2 to vn do begin
    v[i].start := v[i-1].start+v[i-1].deg;
    v[i-1].deg := 0;
  end;
```

```
  v[vn].deg := 0;
  for i := 1 to en do begin
    ep[v[et[i].a].start+v[et[i].a].deg] := i;    Inc(v[et[i].a].deg);
    ep[v[et[i].b].start+v[et[i].b].deg] := i;    Inc(v[et[i].b].deg);
  end;
end;

{ Vráti druhý koniec hrany }
function otherend(v : Longint; edgenum : Longint) : Longint;
begin
  if et[edgenum].a <> v then otherend := et[edgenum].a
    else otherend := et[edgenum].b;
end;

procedure print_out; { Vypíše výsledok }
var
  i : Longint;
  O : Text;
  v : Array[1..4] of Integer;
  tmp : Integer;
begin
  Assign(O, OUT);
  Rewrite(O);
  for i := 1 to en do begin
    if et[i].pair > i then begin
      v[1] := et[i].a;                v[2] := et[i].b;
      v[3] := et[et[i].pair].a;       v[4] := et[et[i].pair].b;

      if (v[1] = v[3]) or (v[1] = v[4]) then begin
        tmp := v[1];   v[1] := v[2];   v[2] := tmp;
      end;
      if (v[4] = v[1]) or (v[4] = v[2]) then begin
        tmp := v[3];   v[3] := v[4];   v[4] := tmp;
      end;
      WriteLn(O, v[1],' ', v[2],' ', v[4]);
    end;
  end;
  Close(O);
end;

procedure DFS(act, parent : Integer); { Prehľadávanie, párovánie hrán }
var
  pair, i : Longint;
begin
  pair := -1;   v[act].visited := 1;
```

```
  { Postupne sa voláme na susedné doteraz nenavštívené vrcholy }
  for i := v[act].start to v[act].start+v[act].deg-1 do
    if v[otherend(act, ep[i])].visited = 0 then
      DFS(otherend(act, ep[i]), act);

  { Popárujeme zvyšné hrany}
  for i := v[act].start to v[act].start+v[act].deg-1 do
    if (et[ep[i]].pair = -1) and (otherend(act, ep[i]) <> parent) then
    begin
      if pair = -1 then  pair := ep[i]  else begin
        et[ep[i]].pair := pair;
        et[pair].pair := ep[i];
        pair := -1;
      end;
    end;

  if pair <> -1 then begin
    { Zostala hrana? Spárujeme ju s hranou k rodičovi }
    i := v[act].start;
    while otherend(act, ep[i]) <> parent do Inc(i);
    et[ep[i]].pair := pair;
    et[pair].pair := ep[i];
  end;
end;

begin
  read_input;
  DFS(1, 1);
  print_out;
end.
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

54. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Riešenia 3. kola kategórie P

2. súťažný deň

Vydala IUVENTA pre vnútornú potrebu Ministerstva školstva SR

Zodpovedný redaktor: M. Forišek

Sadzba programom LaTeX