Slovenská komisia Matematickej olympiády
FPEDaS ŽU, Veľký diel, 010 26 Žilina

52. ročník matematickej olympiády

Riešenia úloh II. kola kategórie P

P-II-1

Riešenie tejto úlohy môžeme rozdeliť na dve časti. Najskôr si spočítame, kde sa musí nachádzať stred symetrie, potom overíme, či sú naozaj hviezdy rozmiestnené stredovo súmerne okolo neho.

Potenciálny stred symetrie nájdeme ľahko. Všimnime si hviezdy $A[a_x, a_y, a_z]$ a $B[b_x, b_y, b_z]$, ktoré sú stredovo súmerné okolo bodu $S[s_x, s_y, s_z]$. Potom súradnice bodu S sú zjavne priemerom súradníc bodov A a B. Alebo inými slovami $A + B = [a_x + b_x, a_y + b_y, a_z + b_z] = 2S$. Takže ak sú všetky body rozmiestnené stredovo súmerne okolo bodu S, ich sčítaním dostaneme N-násobok bodu S, t.j. $[Ns_x, Ns_y, Ns_z]$. Odtiaľ už ľahko spočítame súradnice (jediného možného) bodu S. Kandidáta na bod S vieme teda nájsť v čase $O(N)$. *(Uvedomte si, že z fyzikálneho hľadiska tento bod musí ležať v ťažisku sústavy, no a súradnice ťažiska množiny hmotných bodov dostaneme práve ako priemer súradníc týchto bodov.)*

Zostáva overiť, či sú hviezdy naozaj rozmiestnené stredovo súmerne okolo bodu, ktorý sme práve našli. Aby sme to vedeli rýchlejšie overiť, hviezdy si utriedime lexikograficky podľa ich súradníc, t.j. tak, že $[a_x, a_y, a_z]$ je pred $[b_x, b_y, b_z]$ ak buď $a_x < b_x$, alebo $a_x = b_x \wedge a_y < b_y$, alebo $a_x = b_x \wedge a_y = b_y \wedge a_z < b_z$. Keď teraz vieme súradnice nejakej hviezdy a stredu súmernosti, vieme ľahko spočítať, na akých súradniciach má ležať hviezda s našou súmerná. Následne vieme binárnym vyhľadávaním v čase $O(\log N)$ zistiť, či takúto hviezdu máme. Ak áno, obe si označíme ako spracované a pokračujeme ďalšou nespracovanou hviezdou.

Vhodnou implementáciou predchádzajúcich myšlienok (s použitím triedenia bežiaceho v čase $O(N \log N)$, my sme použili HeapSort) dostávame algoritmus bežiaci v čase $O(N \log N)$. Toto riešenie sa ešte dá trochu zjednodušiť. Stačí si uvedomiť nasledovnú skutočnosť: Všimnime si utriedené poradie hviezd. Ak sú všetky hviezdy naozaj rozmiestnené stredovo súmerne okolo S, tak prvá hviezda musí nutne ležať oproti poslednej, druhá oproti predposlednej, atď. Takže aby sme overili, či je naozaj S stredom súmernosti, stačí nám raz prejsť utriedené pole. To vieme spraviť v čase $O(N)$. Najpomalšou časťou nášho algoritmu však zostane triedenie hviezd, ktoré nevieme spraviť v lepšom čase ako $O(N \log N)$. Taká je preto aj výsledná časová zložitosť nášho algoritmu.

```
program Hviezdy;
const MAXN = 100;
type Hviezda = record x, y, z : Integer; end;
     PoleHviezd = Array[1..MAXN] of Hviezda;
var N : Integer;                {Pocet hviezd}
    H : PoleHviezd;             {Jednotlive hviezdy}
    Stred : Hviezda;            {Spocitany stred symetrie}

{Nacita vstup}
procedure Nacitaj;
var i : Integer;
```

```
begin
  Write('Pocet hviezd: '); ReadLn(N);
  for i := 1 to N do begin Write('Hviezda: '); ReadLn(H[i].x, H[i].y, H[i].z); end;
end;

{Najde stred symetrie}
procedure NajdiStred(var Stred : Hviezda);
var i, xs, ys, zs : Integer;
begin
  xs := 0; ys := 0; zs := 0;
  for i := 1 to N do begin
    xs := xs + H[i].x; ys := ys + H[i].y; zs := zs + H[i].z;
  end;
  Stred.x := xs div N; Stred.y := ys div N; Stred.z := zs div N;
end;

{Porovna suradnice dvoch hviezd}
function PorovnajHviezdy(A, B : Hviezda) : Integer;
begin
  if A.x <> B.x then begin PorovnajHviezdy := A.x - B.x; Exit; end;
  if A.y <> B.y then begin PorovnajHviezdy := A.y - B.y; Exit; end;
  if A.z <> B.z then begin PorovnajHviezdy := A.z - B.z; Exit; end;
  PorovnajHviezdy := 0;
end;

{Prehodi dva prvky haldy}
procedure Prehod(var Halda : PoleHviezd; A, B : Integer);
var Tmp : Hviezda;
begin Tmp := Halda[A]; Halda[A] := Halda[B]; Halda[B] := Tmp; end;

{Vlozi prvok H do haldy}
procedure Vloz(var HT : Integer; var Halda : PoleHviezd; H : Hviezda);
var S : Integer;
begin
  Inc(HT); Halda[HT] := H; S := HT;
  while (S > 1) and (PorovnajHviezdy(Halda[S], Halda[S div 2]) < 0) do begin
    Prehod(Halda, S, S div 2); S := S div 2;
  end;
end;

{Vybere prvu hviezdu z haldy}
function Vyber(var HT : Integer; var Halda : PoleHviezd) : Hviezda;
var S, T : Integer;
begin
  Vyber := Halda[1]; Halda[1] := Halda[HT]; Dec(HT); S := 1;
  while 2 * S <= HT do begin
    T := 0;
    if PorovnajHviezdy(Halda[S], Halda[2 * S]) > 0 then T := 2 * S;
    if (2 * S + 1 <= HT) and (PorovnajHviezdy(Halda[S], Halda[2 * S + 1]) > 0) then
```

```
      if PorovnajHviezdy(Halda[2 * S], Halda[2 * S + 1]) > 0 then
        T := 2 * S + 1;
    if T > 0 then begin Prehod(Halda, S, T); S := T; end else S := HT;
  end;
end;

{Zotriedi hviezdy podla suradnic}
procedure Zotried;
var Halda : PoleHviezd; {Halda na triedenie}
    HT : Integer;       {Pocet prvkov v halde}
    i : Integer;
begin
  HT := 0;
  for i := 1 to N do Vloz(HT, Halda, H[i]);
  for i := 1 to N do H[i] := Vyber(HT, Halda);
end;

{Overi, ci su hviezdy symetricke podle daneho stredu}
function Over(Stred : Hviezda) : Boolean;
var i : Integer;
begin
  Zotried;      {Zotriedi hviezdy podla suradnic}
  for i := 1 to (N + 1) div 2 do
    if (Stred.x - H[i].x <> H[N - i + 1].x - Stred.x) or
       (Stred.y - H[i].y <> H[N - i + 1].y - Stred.y) or
       (Stred.z - H[i].z <> H[N - i + 1].z - Stred.z) then begin
         Over := False; Exit;
       end;
  Over := True;
end;

begin
  Nacitaj;
  NajdiStred(Stred);
  if Over(Stred) then
    WriteLn('Stred symetrie lezi na pozicii ',
      Stred.x, ', ', Stred.y, ', ', Stred.z, '.')
  else
    WriteLn('Hviezdy nie su symetricke podla ziadneho stredu.');
end.
```

P-II-2

Zahľaďme sa na skriňu, ktorá je riešením úlohy. Zjavne môžeme medzi sebou povymieňať poličky tak, aby boli usporiadané podľa výšky (najvyššia bude hore). Teraz ak máme nejaké dve knihy na rôznych poličkách, pričom kniha na nižšej poličke je vyššia, môžeme túto dvojicu kníh vymeniť. (Vyššia kniha sa zmestila do svojej police, tým skôr sa zmestí do vyššie položenej police, ktorá je aspoň tak isto vysoká. Nižšia kniha sa do novej police zmestí, lebo sa do nej

predtým zmestila vyššia kniha.) Konečnou postupnosťou takýchto výmen dostaneme skriňu, ktorá je tiež riešením úlohy a zároveň platí, že na prvej poličke je niekoľko najvyšších kníh, na druhej niekoľko ďalších najvyšších, atď.

Teraz môžeme knihy na každej polici usporiadať podľa výšky. Opäť dostaneme rovnako širokú skriňu, ktorá bude naďalej spĺňať podmienky zo zadania. Nájdime si teraz najvyššiu policu, ktorá nie je plná, t.j. je na nej menej kníh ako je jej šírka. Ak sú ešte nižšie v skrini nejaké ďalšie knihy, môžeme zobrať najvyššiu z nich a presunúť ju na koniec tejto police. Tento postup budeme opäť opakovať, kým to pôjde. Ak nám pri jeho opakovaní niekedy vznikne prázdna polica, môžeme ju z našej skrine vyhodiť.

V predchádzajúcich odsekoch sme vlastne ukázali, že ak existuje vyhovujúca skriňa so šírkou s, tak existuje aj taká, v ktorej sú knihy zhora nadol usporiadané podľa výšky a navyše sú všetky poličky až na možno poslednú plné (je na nich po s kníh). Ak teda chceme zistiť, či existuje nejaká skriňa šírky s, stačí nám hľadať takú skriňu, ktorá bude mať nami spomenuté vlastnosti.

Základom nášho algoritmu bude funkcia, ktorá pre dané s rozhodne, či existuje skriňa šírky s, vyhovujúca zadaniu a našim podmienkam. Optimálnu šírku skrine potom nájdeme binárnym vyhľadávaním (šírka 0 nestačí, šírka N určite stačí). Samotná funkcia spočíta najmenšie možné výšky jednotlivých poličiek (výška i-tej poličky bude o 1 väčšia ako výška $(s(i-1)+1)$. najvyššej knihy) a zistí, či dokopy vôjdu do 250 cm.

Odhadnime teraz časové a pamäťové nároky nášho algoritmu. Na začiatku potrebujeme zotriediť výšky N kníh. To vieme spraviť v čase $O(N \log N)$, prípadne ak predpokladáme, že sú to celé čísla (navyše zhora ohraničené 250), vieme ich utriediť v čase $O(N)$. Keď zavoláme funkciu, ktorá zisťuje, či existuje skriňa šírky s, bude potrebovať čas $O(N/s)$, lebo musí sčítať $\lceil N/s \rceil$ čísel. Túto funkciu budeme volať $O(\log N)$-krát, preto časová zložitosť nášho algoritmu je $O(N \log N)$.

Dokonca ak si uvedomíme, že pri i-tom volaní našej funkcie bude šírka skrine aspoň $N/2^i$, a teda i-te volanie bude trvať najviac $O(2^i)$, nahliadneme, že celkový čas, ktorý zaberú všetky volania našej funkcie, bude dokonca len $O(N)$. Pamäťové nároky sú zjavne lineárne, t.j. $O(N)$.

```
program kniznica;
const MAXN = 100;
      VYSKA_MIESTNOSTI = 250;
var vyska : array[1..MAXN] of word;      { vysky knih }
    n : word;    { pocet knih }

procedure utried_vysky(i1, i2 : word); { quicksort }
var pivot, w, j1, j2 : word;
begin
  if i1 >= i2 then exit;
  pivot := vyska[(i1 + i2) div 2];
  j1 := i1; j2 := i2;
  while (j1 < j2) do begin
    while (vyska[j1] > pivot) do inc(j1);
    while (vyska[j2] < pivot) do dec(j2);
    w := vyska[j1]; vyska[j1] := vyska[j2]; vyska[j2] := w;
    inc(j1); dec(j2);
  end;
  utried_vysky(i1, j2); utried_vysky(j1, i2);
end;
```

```
function existuje(s : word) : boolean;
var v, i : word;
begin
  v := 1; i := 1;
  repeat
    v := v + vyska[i] + 1;
    i := i + s;
  until i > n;
  existuje := v <= VYSKA_MIESTNOSTI
end;

var i, s1, s2, v : word;
begin
  readln(n); for i := 1 to n do read(vyska[i]);
  utried_vysky(1, n);
  if vyska[1] > VYSKA_MIESTNOSTI - 2 then begin
    writeln('Pre zadane rozmery knih skrina neexistuje!');
    halt;
  end;
  s1 := 1; s2 := n;
  while s1 < s2 do
    if existuje((s1 + s2) div 2) then s2 := (s1 + s2) div 2
      else   s1 := (s1 + s2) div 2 + 1;
  writeln('Optimalna sirka skrine je ', s1, ' cm.');
  writeln('Pocet policiek v skrini: ', (n + s1 - 1) div s1);
  i := 1; v := 1;
  while (i <= n) do begin
    v := v + vyska[i] + 1;
    writeln('Vyska policky: ', vyska[i], ' cm');
    write('Vysky knih na policke:');
    repeat
      if (i > n) then break;
      write(' ', vyska[i], ' cm');
      inc(i);
    until (i mod s1) = 1;
    writeln;
  end;
  writeln('Vyska skrine: ', v, ' cm');
end.
```

P-II-3

Použijeme podobnú myšlienku ako pri riešení úlohy v domácom kole. (Vieme posledný stĺpec tabuľky, jeho utriedením dostaneme prvý stĺpec, v hľadanom reťazci za posledným písmenom ľub. riadku tabuľky nasleduje prvé písmeno z toho istého jej riadku.) Jediný rozdiel je ten, že keď sa môžu písmená opakovať, nemusíme vedieť na prvý pohľad určiť, ako po sebe písmená nasledujú. Napr. pre tabuľku zo zadania nevieme hneď povedať, po ktorom 'a' má ísť 'b', po ktorom 'k' a po ktorom 'a'.

Spravme teraz nasledujúcu úvahu: Keďže vieme, v ktorom riadku tabuľky leží naše slovo, vieme jeho prvé písmeno c_1. Toto písmeno sa môže v prvom stĺpci vyskytovať viackrát, nech sú príslušné riadky (zhora nadol) $c_1v_1, c_1v_2, \ldots, c_1v_k$. Potom ale $v_1 < v_2 < \ldots < v_k$, a teda aj $v_1c_1 < v_2c_1 < \ldots < v_kc_1$. To sú ale práve všetky riadky, ktoré *končia* znakom c_1. Nech je náš riadok i-ty zhora spomedzi riadkov začínajúcich na c_1. Práve sme ukázali, že presunutím c_1 na koniec dostaneme i-ty riadok zhora spomedzi tých, ktoré končia na c_1. Prvé písmeno tohto riadku je zároveň druhým písmenom slova, ktoré hľadáme. Označíme ho c_2 a túto úvahu opakujeme, kým takto nezostrojíme celé hľadané slovo.

Implementácia je pomerne jednoduchá. Písmená v poslednom stĺpci očíslujeme od 1 po N. Použijeme priehradkové triedenie (viď program), aby sme zistili prvý stĺpec tabuľky v lineárnom čase. Písmená zotriedime podľa abecedy, v prípade rovnosti podľa čísla, ktoré dostali. Následne hľadané slovo zostrojíme priamočiarym prechodom riadkami podľa vyššie uvedeného algoritmu. Náš program spočíta len indexy písmen v prvom stĺpci, samotné písmená z nich ľahko vieme určiť. Časová aj pamäťová zložitosť nášho riešenia sú lineárne od počtu písmen slova.

V príklade zo zadania zotriedením $k_1a_2r_3a_4a_5b_6$ dostaneme $a_2a_4a_5b_6k_1r_3$. Slovo leží v 2. riadku, teda prvý znak je a_4. Riadok končiaci a_4 začína b_6, to je teda druhý znak. Riadok končiaci b_6 začína r_3, atď.

```
program transformacie;
const MAX = 10000;
var prvy_stlpec : array[1 .. MAX] of integer;
    posledny_stlpec : string;
    riadok, dlzka, i, l : integer;
    buckets : array[char] of integer;
    ch : char;
begin
  {nacitanie a ocislovanie}
  readln (posledny_stlpec); readln (riadok);
  dlzka := length (posledny_stlpec);
  for ch := #0 to #255 do buckets[ch] := 0;
  for i := 1 to dlzka do inc (buckets[posledny_stlpec[i]]);

  {zotriedenie}
  l := 1;
  for ch := #0 to #255 do begin
    i := l;
    inc (l, buckets[ch]);
    buckets[ch] := i;
  end;
  for i := 1 to dlzka do begin
    ch := posledny_stlpec[i];
    l := buckets[ch];
    inc (buckets[ch]);
    prvy_stlpec[l] := i;
  end;

  {vypis}
  for i := 1 to dlzka do begin
    write (posledny_stlpec[prvy_stlpec[riadok]]);
```

```
    riadok := prvy_stlpec[riadok];
  end;
  writeln;
end.
```

P-II-4

Podobnosť úlohy s hľadaním dĺžky najkratšej cesty v orientovanom grafe nie je čisto náhodná. Preto sa aj my budeme držať tejto analógie. Miestnosti v inštitúcii budú vrcholmi nášho grafu, potrubia orientovanými hranami. Potom A nie je nič iné ako matica susednosti tohto grafu. Štandardným algoritmom na riešenie danej úlohy je prehľadávanie do šírky, len ho musíme upraviť, aby bolo reverzibilné.

Vrcholy si rozdelíme do *vrstiev*, pričom do vrstvy W_i budú patriť tie vrcholy, do ktorých je z vrcholu 1 vzdialenosť i (t.j. použitím žiadnych $i-1$ rúr tam správu nevieme dostať, použitím vhodných i rúr ju tam dostať vieme). Neprázdnych vrstiev je zjavne najviac N a vieme ich ľahko zostrojiť indukciou – $W_0 = \{1\}$, do W_i patria tie vrcholy, ktoré doteraz nie sú v žiadnej vrstve a vedie do nich hrana z nejakého vrcholu v W_{i-1}. (Vďaka tejto hrane doň existuje cesta dĺžky i, keby existovala aj kratšia, patril by tento vrchol do niektorej už spočítanej vrstvy.)

Práve opísaný postup zostrojenia vrstiev je zjavne reverzibilný – pri konštrukcii novej vrstvy nijako nemeníme už spočítané vrstvy. Nakoniec nájdeme číslo vrstvy, v ktorej leží vrchol N, pripočítame jej číslo k výstupnej premennej a všetky informácie o vrstvách opäť odpočítame. (Na to je vhodná konštrukcia *wrap ... on ...*) Priamočiara implementácia tohoto riešenia má časovú zložitosť $O(N^3)$ a pamäťovú $O(N^2)$.

Pamäťovú zložitosť vieme ešte zlepšiť (a dokonca sa tým náš program aj zefektívni) nasledovne: Vrstvy si vieme pamätať efektívnejšie, lebo vo všetkých vrstvách dokopy je najviac N vrcholov. Tie budeme postupne ukladať do poľa V a budeme mať pomocné pole S, ktoré nám bude hovoriť, kde v poli V ktorá vrstva začína. Vrcholy vrstvy W_i teda budú uložené v prvkoch $V[S[i]]$ až $V[S[i+1]-1]$.

Všimnite si, že reverzibilita programu sa príliš nekamaráti so značkovaním vrcholov (na aké sme zvyknutí pri „klasickom“ prehľadávaní do šírky). Povedzme že by sme si pre každý vrchol pamätali, či sme v ňom už boli. Značkovanie bude vyzerať približne takto:

```
if UzSomTamBol[i]=0 then begin {objavil som novy vrchol}
  UzSomTamBol[i] += 1; ...;
end;
```

Tu sa ale dostávame do sporu s reverzibilitou – po vykonaní príkazu if totiž nevieme povedať, či bola podmienka splnená, alebo nie, lebo $UzSomTamBol[i]$ bude vždy 1. Ale presne toto náš jazyk zakazuje!

Zachráni nás jednoduchý trik: Ako značku budeme používať číslo vrstvy, ktorej bol vrchol objavený (alebo *inf* – dosť veľké číslo, ak sme tam ešte neboli). Značkovanie teda bude vyzerať nasledovne:

```
if Znacka[i] >= TatoVrstva then begin {objavil som novy vrchol}
  Znacka[i] -= inf - TatoVrstva; ...;
end;
```

To už je korektné, lebo vykonaním príkazu if sa platnosť podmienky nezmení. Zároveň to aj funguje – pri spracovaní ďalších vrstiev správne spoznáme už označkované vrcholy.

Zostáva už len dodať, že časová zložitosť výsledného algoritmu je kvadratická a pamäťová lineárna.

Poznámka: Ak by sme sa vzdali polynomiálnej časovej zložitosti, existovali by aj pamäťovo efektívnejšie riešenia. Jedno je založené na nasledovnej myšlienke: Hľadáme cestu dĺžky najviac l z x do y. Ak $l \leq 1$, je to triviálne. V opačnom prípade vyskúšame postupne všetky možnosti, kde sme sa mohli nachádzať v približnej polovici cesty. Pre každý vrchol v rekurzívne overíme, či sa vieme na najviac $\lfloor l/2 \rfloor$ krokov dostať z x do v a či sa vieme na najviac $\lceil l/2 \rceil$ krokov dostať z v do y. Ak takýto vrchol nájdeme, vlastne sme zistili, že existuje cesta z x do y dĺžky najviac l, naopak ak takýto vrchol v nenájdeme, zjavne žiadna takáto cesta neexistuje. Postupne napr. binárnym vyhľadávaním nájdeme najmenšie l, pre ktoré existuje cesta tejto dĺžky. Takto dostaneme pamäťovú zložitosť $O(\log N)$, ale časová sa nám zhorší na $O(N^{\log N})$.

```
procedure Skumaj(var n : word; var A : array [1..n] of array [1..n] of bit; var d : word);
var inf, cnt : word;
var L, V, S : array [0..n] of word;
begin
  wrap begin
    inf += n + 47;                          { „nekonecna vzdialenost“ }
    for var i = 1 to n do L[i] += inf;
    V[0] += 1;                              { nulta vrstva: vrchol 1 ... }
    L[x] -= inf;                            { ... vo vzdialenosti 0 ... }
    S[1] += 1;                              { ... a ziadny dalsi }
    for var i = 1 to n - 1 do begin         { hladame dalsie vrstvy }
      S[i + 1] += S[i];                     { zatial prazdna }
      for var w = 1 to n do
        if L[w] >= i then                   { nezaradeny vrchol }
          wrap
            for var j = S[i - 1] to S[i] - 1 do   { vedie do neho hrana z vrstvy i-1? }
              if A[V[j]][w] = 1 then
                cnt += 1
          on if cnt > 0 then begin          { ano => pridat do i-tej vrstvy }
              V[S[i + 1]] += w;
              S[i + 1] += 1;
              L[w] -= inf - i               { L[w] >= i stale platí }
              end
      end
    end
  on if L[N] > 0 then d += L[N]             { vratime vysledok }
end;
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY
52. ROČNÍK MATEMATICKEJ OLYMPIÁDY
Vzorové riešenia II. kola kategórie P
Vydala IUVENTA
pre vnútornú potrebu Ministerstva školstva SR
Autori príkladov P. Töpfer, D. Kráľ, M. Mareš
Zodpovedný redaktor M. Forišek
Sadzba programom LaTeX