Slovenská komisia Matematickej olympiády
FPEDaS ŽU, Moyzesova 20, 010 26 Žilina

51. ročník matematickej olympiády

Riešenia úloh II. kola kategórie P

P-II-1

Prvé riešenie, ktoré môže človeka napadnúť, je vyskúšať všetky možné obdĺžniky. Obdĺžnik je určený dvojicou protiľahlých vrcholov, pre každý vrchol máme mn možností. Pre každý potenciálny obdĺžnik prejdeme po jeho obvode a overíme, či všetky jeho obvodové prvky sú jednotky. Tento algoritmus pracuje v čase $O((mn)^2(m+n))$.

Naše prvé vylepšenie bude v tom, že zrýchlime zisťovanie, či je daný obdĺžnik orámovaný. Nech $h[i,j]$ označuje dĺžku úseku jednotiek v stĺpci *nad* prvkom (i,j), vrátane tohto prvku. Presnejšie, $h[i,j]=d$ je také číslo že platí platí $A[i,j]=A[i-1,j]=\ldots=A[i-d+1,j]=1$ a zároveň buď $d=i$, alebo $A[i-d,j]=0$. Všimnite si, že ak $A[i,j]=0$, podľa tejto definície aj $h[i,j]=0$. Podobne, nech $l[i,j]$ označuje dĺžku úseku jednotiek v riadku *naľavo* od prvku (i,j) (vrátane prvku (i,j)). Hodnoty $h[i,j]$ a $l[i,j]$ si môžeme predpočítať dopredu pre všetky prvky matice. Pre každý prvok zvlášť vieme zistiť dĺžku úseku jednotiek naľavo a nahor od neho v čase $O(m+n)$. Spolu máme mn prvkov, takže predpočítanie polí h a l poľahky implementujeme v čase $O(mn(m+n))$. Môžeme ho však urýchliť. Ak totiž $A[i,j]=0$, vieme že $h[i,j]=0$. Ak $A[i,j]=1$, tak počet jednotiek v súvislom úseku nad prvkom (i,j) je o jedna väčší ako počet jednotiek v úseku nad prvkom $(i-1,j)$, t.j. $h[i,j]=h[i-1,j]+1$. Stačí nám teda počítať zhora dole a v jednom prechode v čase $O(mn)$ získame všetkých mn hodnôt $h[i,j]$.

Predpokladajme teraz, že chceme zistiť, či je obdĺžnik s ľavým horným rohom (i_1,j_1) a pravým dolným rohom (i_2,j_2) orámovaný. Obdĺžnik má dolnú hranu práve vtedy, ak úsek jednotiek naľavo od prvku (i_2,j_2) má dĺžku aspoň j_2-j_1+1. Podobne overíme, či úsek naľavo od prvku (i_1,j_2) (horná hrana) má dĺžku aspoň j_2-j_1+1 a či úseky nahor od prvkov i_2,j_1 a i_2,j_2 (ľavá a pravá hrana) majú dĺžku aspoň i_2-i_1+1. S pomocou polí l a h teda vieme v koštantnom čase zistiť, či je daný obdĺžnik orámovaný. Opäť vyskúšame všetky možnosti pre ľavý horný a pravý dolný roh, pre každý obdĺžnik zistíme, či je orámovaný a vyberieme ten s najväčšou plochou. Tento algoritmus potrebuje $O(mn)$ času na prípravu a $O((mn)^2)$ času na prejdenie všetkých obdĺžnikov. Celkový čas behu je teda $O((mn)^2)$.

Vzorové riešenie je ešte o čosi rýchlejšie a pracuje nasledovne. Pre každú dvojicu riadkov $i_1<i_2$ nájdeme najväčší orámovaný obdĺžnik, ktorého horná hrana je v riadku i_1 a dolná hrana v riadku i_2. Sústreďme sa na dvojicu pevne zvolených riadkov i_1 a i_2. Stĺpec j nazveme *okrajový*, ak v úseku medzi riadkami i_1 a i_2 (vrátane) obsahuje samé jednotky. Všimnite si, že stĺpec je okrajový práve vtedy, ak $h[i_2,j]\geq i_2-i_1+1$. Stĺpec j nazveme *jednotkový*, ak v i_1 aj i_2-tom riadku obsahuje jednotku. Stĺpec, ktorý obsahuje nulu v aspoň jednom z týchto dvoch riadkov nazveme *nulový*.

Každý orámovaný obdĺžnik s vodorovnými hranami ležiacimi v riadkoch i_1 a i_2 zodpovedá úseku od stĺpca j_1 po stĺpec j_2 pre nejaké $j_1<j_2$ také, že j_1 aj j_2 sú okrajové stĺpce a stĺpce $j_1+1,\ldots,j_2-1$ sú jednotkové. Uvažujme maximálny úsek po sebe idúcich jednotkových stĺpcov (pojmom maximálny myslíme, že sa nedá rozšíriť, t.j. na oboch stranách susedí buď s nulovým stĺpcom alebo s okrajom matice). Ak tento úsek neobsahuje aspoň dva okrajové stĺpce, zjavne nemôže obsahovať ani žiadny orámovaný obdĺžnik. Ak obsahuje aspoň dva okrajové

stĺpce, potom najväčší orámovaný obdĺžnik v danom úseku je určený najľavejším a najpravejším okrajovým stĺpcom daného úseku. Na nájdenie najväčšieho obdĺžnika v páse medzi riadkami i_1 a i_2 nám teda stačí pre každý maximálny jednotkový úsek nájsť najľavejší a najpravejší okrajový stĺpec. Toto sa dá ľahko dosiahnuť v čase $O(n)$. Keďže musíme vyskúšať všetky dvojice riadkov, celková časová zložitosť je $O(m^2n)$.

```
program Obdlznik; {MO-P 51-II-1}

const MAX = 100;
var A : array[1..MAX, 1..MAX] of integer;
    h : array[0..MAX, 1..MAX] of integer;
    m, n, i, j : integer;
    naj_plocha, naj_vyska, naj_sirka, naj_i, naj_j : integer;
    i1, i2, j1, j2 : integer;
    f : text;

begin
  readln(m, n);
  for i := 1 to m do for j := 1 to n do
    read(A[i][j]);

  {výpočet stĺpcových súm}
  for j := 1 to n do h[0][j] := 0;
  for i := 1 to m do for j := 1 to n do
    if A[i][j] = 0 then h[i][j] := 0
    else h[i][j] := h[i-1][j] + 1;

  naj_plocha := 0;

      { pre každú dvojicu riadkov                    }
  for i1 := 1 to m do for i2 := i1 + 1 to m do begin
    j1 := 1; j2 := 1;
    while j2 <= n do begin
      { nájdeme potenciálnu ľavú hranu obdĺžnika     }
      while (j1 <= n) and (h[i2][j1] < i2 - i1 + 1) do j1 := j1 + 1;
      j2 := j1 + 1;
      { hľadáme potenciálnu pravú hranu, kým nenarazíme }
      { na nulu v niektorej z vodorovných hrán       }
      while (j2 <= n) and (A[i1][j2] = 1) and (A[i2][j2] = 1) do begin
        if (h[i2][j2] >= i2 - i1 + 1) and
           ((i2 - i1 + 1) * (j2 - j1 + 1) > naj_plocha) then begin
      { našli sme pravú zvislú hranu a nájdený obdĺžnik }
      { je zatiaľ najväčší }
          naj_plocha := (i2 - i1 + 1) * (j2 - j1 + 1);
          naj_vyska := i2 - i1 + 1;
          naj_sirka := j2 - j1 + 1;
          naj_i := i1; naj_j := j1;
        end;
        j2 := j2 + 1;
```

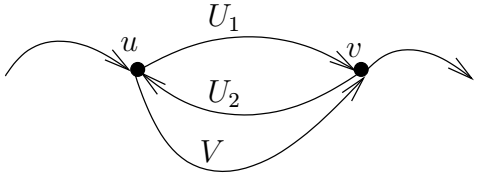
```
      end;
      j1 := j2 + 1;
    end;
  end;

  if naj_plocha = 0 then
    writeln('V matici sa nenachadza ziadny oramovany obdlznik')
  else
    writeln('Najvacsi oramovany obdlznik ma rozmery ',
            naj_vyska, ' x ', naj_sirka,
            ' a jeho lavy horny roh je v ',
            naj_i, ' riadku a ', naj_j, ' stlpci');
end.
```

P-II-2

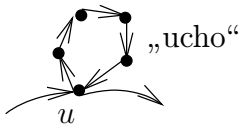
Preveďme našu úlohu do teórie grafov. Sieť rúr je jednoduchý orientovaný graf bez násobných hrán a slučiek. Uzly budú tvoriť vrcholy a rúry orientované hrany tohoto grafu. Počet hrán, vedúcich do vrcholu, budeme nazývať jeho vstupným stupňom a počet vychádzajúcich hrán jeho výstupným stupňom. Zjavne v našom grafe sa vstupný a výstupný stupeň každého vrcholu rovnajú. Toto číslo budeme tiež volať počtom prechodov cez vrchol.

Čo by sa stalo, ak by niektorý vrchol v mal počet prechodov aspoň 3? Trasa na vstupe prechádza cez v aspoň trikrát. Nech X je úsek trasy medzi prvým a druhým príchodom do v a Y úsek medzi druhým a tretím príchodom doň. Zostrojme teraz novú trasu pre robota. Po prvý príchod do v ide podľa trasy na vstupe. Po príchode do v pôjde najskôr úsek Y (čím sa vráti do v), potom úsek X (čím sa opäť vráti do v) a dokončí svoju trasu rovnako ako v trase na vstupe. Inými slovami oproti trase na vstupe sme vymenili poradie úsekov X a Y. Tým sme ale zostrojili inú trasu. Preto aby žiadna iná trasa neexistovala, musí mať každý vrchol počet prechodov 1 alebo 2.

Majme teraz takýto graf a v ňom vyznačenú trasu, ktorá začína aj končí vo vrchole 1 a prechádza po každej hrane práve raz. Pokiaľ tento graf neobsahuje žiadne hrany, tak trasa je zjavne jediná (prázdna trasa). Čo ak náš graf nejaké hrany má? Poďme po tejto trase, kým sa nám prvýkrát nejaký vrchol u nezopakuje. To určite skôr či neskôr nastane. Všimnime si úsek cesty medzi prvým a druhým príchodom do u. Označme ho U. Čo ak má niektorý vrchol U (iný ako u) počet prechodov 2? To by znamenalo, že sa tento vrchol ešte niekedy na trase vyskytne. Označme v ten z takýchto vrcholov, ktorý sa v nej vyskytne najskôr. Tento vrchol rozdelí U na dve časti – časť od u po v označme U_1 a časť od v po u označme U_2. Časť od druhého príchodu do u po druhý príchod do v označme V. Naša trasa teda vyzerá nasledovne: Robot príde do u, prejde U_1, U_2, V a zvyšok trasy. Potom ale existuje aj iná trasa: Robot rovnako príde do u, prejde V, U_2, U_1 a zvyšok trasy prejde rovnako, ako v trase na vstupe.



Preto aby žiadna iná trasa neexistovala, musia mať všetky vrcholy U (okrem u) počet prechodov 1. (Teda úsek U bude tvoriť *ucho* nad vrcholom u.)



Uvedomme si teraz, že keď hrany tvoriace toto *ucho* z grafu odstránime, nezmeníme tým počet trás v grafe. (Každá trasa v grafe totiž vyzerá nasledovne: Robot nejako príde do u, prejde *ucho* a nejako prejde zvyšok grafu. Keď hrany tvoriace *ucho* odstránime, ku každej trase v pôvodnom grafe nájdeme zodpovedajúcu trasu v novom grafe tak, že z nej odstránime hrany *ucha*.) Týmto ale dostaneme graf s menej hranami, na ktorom môžeme tento postup zopakovať. Ak sa nám takto podarí postupne odstrániť všetky hrany z grafu, znamená to, že aj pôvodný graf mal len jednu trasu. Naopak, ak v niektorom kroku zistíme, že v práve spracovávanom grafe existuje viac trás, znamená to, že aj náš pôvodný graf obsahoval viac trás.

To už je aj návod, ako zostrojiť algoritmus riešiaci túto úlohu. Pre každý vrchol si budeme pamätať počet prechodov cezeň. Ak má niektorý vrchol počet prechodov aspoň 3, vyhlásime, že existuje iná trasa a skončíme. V opačnom prípade začneme postupne čítať zo vstupu trasu a budeme si pamätať, cez ktoré vrcholy sme už išli. Keď sa nám niektorý vrchol v zopakuje, pozrieme sa, či niektorý vrchol medzi prechodmi cez v nemá počet prechodov 2. Ak taký nájdeme, vyhlásime, že existuje iná trasa a skončíme. Ak nie, odstránime tieto vrcholy z trasy a pokračujeme. (Všimnite si, že si vôbec nepotrebujeme pamätať graf a meniť ho, keďže je jednoznačne popísaný trasou.) Keď skončíme (s prázdnou trasou), vyhlásime, že žiadna iná trasa neexistuje a skončíme.

Správnosť tohoto algoritmu vyplýva z vyššie uvedeného popisu. Aká je jeho časová zložitosť? Nech má náš graf N vrcholov a M hrán. Ak z niektorého vrcholu vychádzajú aspoň 3 hrany, akonáhle prvý takýto vrchol nájdeme, môžeme skončiť. V tomto prípade je časová zložitosť nášho algoritmu $O(N)$. (Pokiaľ by sme dočítali zo vstupu celý graf a až potom kontrolovali počty prechodov cez vrcholy, zhorší nám to zložitosť na $O(M + N)$.) V opačnom prípade z každého vrcholu vychádzajú najviac dve hrany, preto $M = O(N)$. (Čiže náš graf má len lineárne veľa hrán v závislosti od počtu vrcholov.) A teda aj počet hrán trasy je $O(N)$, lebo v trase je každá hrana práve raz. Každý vrchol na trase najviac raz načítame, najviac raz sa počas behu algoritmu pozrieme na počet prechodov cez neho a najviac raz ho z trasy vyhodíme. To znamená, že (pri vhodnej implementácii) bude aj v tomto prípade časová zložitosť nášho algoritmu $O(N)$.

```
program Rury_II;

const MAXN = 100;

var f : text;
    prechody : array[1..MAXN] of byte; {počet prechodov cez vrchol}
    n, m : integer; {počet vrcholov a hrán}

procedure Nacitaj_Graf;
var i, x, y : integer;
begin
  readln(f, n, m);
  fillchar(prechody, n, 0); {vyplň pole nulami}
  for i := 1 to m do begin
    readln(f, x, y);
    inc(prechody[x]);
    if (prechody[x] = 3) then begin
```

```
            writeln('Existuje ina trasa.');
            close(f);
            halt;
        end;
    end;
end;

procedure Spracuj_Trasu;

var bol : array[1..MAXN] of boolean;  {či už trasa šla cez daný vrchol}
    trasa : array[1..MAXN+1] of integer;
    ltr : integer;    {dĺžka už načítanej trasy}
    i, kde : integer;

begin
    ltr := 0;
    fillchar(bol, sizeof(bol), 0);
    for i := 1 to m do begin
        inc(ltr);
        read(f, trasa[ltr]);
        if (bol[trasa[ltr]]) then begin
            {ideme kontrolovať, či máme „ucho“}
            kde := ltr - 1;
            while (trasa[kde] <> trasa[ltr]) do begin
                if (prechody[trasa[kde]] = 2) then begin
                    writeln('Existuje ina trasa.');
                    close(f);
                    halt;
                end;
                bol[trasa[kde]] := false;
                dec(kde);
            end;
            dec(prechody[trasa[ltr]]);  {odstránili sme ucho}
            ltr := kde;
        end else begin
            bol[trasa[ltr]] := true;
        end;
    end;
end;

begin
    assign(f, 'rury.in'); reset(f);
    Nacitaj_Graf;
    Spracuj_Trasu;
    close(f);
    writeln('Neexistuje ina trasa.');
end.
```

P-II-3

Ak sa učiteľ otáča zo svojho základného smeru k nejakému žiakovi proti smeru hodinových ručičiek, budeme hovoriť, že tento žiak je naľavo. Ak sa otáča v smere hodinových ručičiek, žiak je napravo.

Najprv dokážeme, že vždy existuje riešenie, v ktorom je učiteľ otočený smerom k nejakému žiakovi. Predstavme si priamku cez bod $[0,0]$ určujúcu základný smer. Otočme teraz túto priamku doľava o malý uhol α tak, aby žiaden žiak, ktorý bol naľavo neprešiel na pravú stranu a naopak. Po takomto otočení uhly otočenia všetkých žiakov naľavo klesnú o α a uhly otočenia všetkých žiakov napravo stúpnu o α. Teda ak naľavo je viacej žiakov, priemerný uhol otočenia sa zmenšil, ak naľavo je menej žiakov, priemerný uhol sa zväčšil a ak je na oboch stranách rovnako žiakov, priemerný uhol zostáva rovnaký. Ak priamka určujúca najlepší základný smer neprechádza cez žiaden bod, tak v aspoň jednom smere ju môžeme kúsok otočiť bez toho, aby priemerný uhol vzrástol. V otáčaní prestaneme, len čo priamka narazí na prvý bod.

Ešte stále ale zostáva prípad, že priamka síce prechádza cez niektorý bod, ale učiteľ sa pozerá opačným smerom, t.j. je chrbtom k tomuto žiakovi. To ale nikdy nie je optimálne riešenie. Predpokladajme, že napravo je aspoň toľko žiakov ako naľavo. Vtedy, ak otočíme priamku kúsok doprava, všetkým napravo a žiakovi za chbtom klesne uhol otočenia a teda celkovo sa uhol otočenia zlepší. Ak naopak je naľavo viac žiakov, otočenie doľava zlepší uhol.

Je teda potrebné preskúmať iba n základných smerov, v ktorých je učiteľ otočený smerom k niektorému žiakovi. Pre každý je možné spočítať priemerný uhol otočenia a vybrať najlepší. Ak budeme počítať priemerný uhol otočenia pre každý smer osobitne, dostaneme algoritmus s časovou zložitosťou $O(n^2)$. My však ukážeme algoritmus s časovou zložitosťou $O(n \log n)$.

Nech α_i je uhol, ktorý zviera polpriamka idúca cez žiaka i a bod $[0,0]$ s osou x (t.j. uhol, ktorý vypočítame funkciou `uhol`(x_i, y_i)). Uvedomme si, aký je vlastne uhol otočenia medzi učiteľom natočeným k žiakovi u a medzi žiakom i. Musíme uvažovať 4 prípady:

- Žiak i je naľavo a $\alpha_i > \alpha_U$. Uhol otočenia je $\alpha_i - \alpha_U$.
- Žiak i je naľavo a $\alpha_i < \alpha_U$. Uhol otočenia je $\alpha_i - \alpha_U + 360°$.
- Žiak i je napravo a $\alpha_i < \alpha_U$. Uhol otočenia je $\alpha_U - \alpha_i$.
- Žiak i je napravo a $\alpha_i > \alpha_U$. Uhol otočenia je $\alpha_U - \alpha_i + 360°$.

Aby sme našli priemerný uhol otočenia, potrebujeme sčítať uhly otočenia pre všetkých žiakov. Sčítaním dostaneme nasledujúci výraz:

$$\sum_{\substack{\text{žiak } i \\ \text{je naľavo}}} \alpha_i - \sum_{\substack{\text{žiak } i \\ \text{je napravo}}} \alpha_i - L \cdot \alpha_U + P \cdot \alpha_U + X \cdot 360°$$

kde L je počet žiakov naľavo, P je počet žiakov napravo a X je počet žiakov, ktorí sú naľavo s uhlom menším ako α_U alebo napravo s uhlom väčším ako α_U.

Teda súčet uhlov otočenia vieme spočítať v konštantnom čase, ak poznáme súčet hodnôt α_i pre žiakov naľavo a napravo a ak poznáme počty L, P, X.

Náš algoritmus si najprv utriedi body podľa uhla α_i (t.j. proti smeru hodinových ručičiek). Potom si pre každé i spočíta súčet uhlov α_j pre prvých i bodov v utriedenom poradí a tieto čísla uloží do poľa β (t.j. $\beta_i = \alpha_1 + \alpha_2 + \cdots + \alpha_i$). Všimnite si, že $\beta_i = \alpha_i + \beta_{i-1}$, a teda idúc zľava doprava vieme spočítať všetky hodnoty β_i v lineárnom čase. Ak teraz chceme spočítať súčet uhlov v úseku od i-teho po j-teho žiaka, stačí spočítať $\beta_j - \beta_{i-1}$.

Po tejto inicializácii budeme postupne skúmať možné základné smery v poradí, v akom sú v striedenom poli. Nech U je žiak, ktorý určuje základný smer. Pre každé U si nájdeme

posledného žiaka l, ktorý je naľavo. Všetci žiaci medzi U a l (vrátane l) sú naľavo, ostatní žiaci sú napravo. Niekedy máme $l < U$, a vtedy sú naľavo žiaci $U+1, \ldots, n$ a $1, \ldots, l$. Podobne žiaci napravo buď tvoria jeden alebo dva súvislé úseky v utriedenom poli. Ak teda vieme U a l, môžeme použiť pole β na to, aby sme v konštantnom čase zistili súčet uhlov α_i pre žiakov naľavo a napravo, lebo sú to súčty jedného alebo dvoch súvislých úsekov v poli α. Počty žiakov naľavo a napravo (L a P) tiež ľahko spočítame. Hodnota X je trošku ťažšia, ale žiak i má α_i menšie ako α_U len vtedy, ak je i menšie ako U (lebo pole je striedené podľa α). Teda, ak poznáme indexy U a l, vieme spočítať súčet (a priemer) uhlov otočenia v konštantnom čase.

Zostáva vyriešiť problém, ako efektívne nájsť hodnotu l, t.j. index posledného prvku vľavo. Pre $U = 1$ jednoducho začneme s $l = 1$ a budeme zvyšovať l, až kým nenájdeme posledný prvok, ktorý je naľavo. Pre každú ďalšiu hodnotu U využijeme fakt, že l z predchádzajúceho kroku je teraz určite naľavo a teda začneme z predchádzajúcej hodnoty l a budeme l zvyšovať, až kým nenájdeme prvý bod vpravo (ak prídeme na koniec poľa, pokračujeme od jednotky). Index l takýmto spôsobom počas celého algoritmu obíde celé pole iba dvakrát, teda celková zložitosť hľadania posledného prvku vľavo je $O(n)$.

Celková časová zložitosť je $O(n \log n)$, lebo triedenie pracuje v čase $O(n \log n)$, pole β vieme vypočítať v čase $O(n)$, celkový čas otáčania indexu l je $O(n)$ a výpočet súčtu uhlov otočenia je konštantný pre jeden smer, t.j. $O(n)$ pre všetky smery.

V uvedenom programe sme kvôli úspore miesta nahradili $O(n \log n)$ triedenie kvadratickým. Taktiež sme použili pole dĺžky $2n$, v ktorom sa každý bod nachádza dvakrát, čo zjednodušuje výpočty (netreba uvažovať, že posledný bod vľavo bude mať index menší ako U – ak treba jednoducho pokračujeme s hľadaním indexu l ďalej za n, využívajúc druhé kópie bodov v poli). Pri implementácii treba dbať na to, aby program správne ošetril rôzne okrajové prípady, napríklad, že všetky body sú naľavo alebo napravo od U.

```
program P_II_3;
const MAX = 1000;                { maximálny počet bodov }
const priamo = 1; vlavo = 2; vpravo = 3; { smery }
type bod = record                { údaje pre jedného žiaka }
             x, y : real;        { súradnice }
             uhol : real;        { uhol vypočítaný funkciou „uhol“ }
             sucet : real;       { súčet uhlov prvých niekoľko žiakov }
           end;
var a : array [0..2*MAX] of bod; { pole bodov, každý je tam dvakrát }
  n : integer;                   { počet bodov (žiakov) }

function uhol(x, y : real) : real;
var vysl : real;
begin { implementácia funkcie „uhol“ definovanej v zadaní }
  if x = 0 then begin            { špeciálny prípad x=0 }
    if y > 0 then uhol := 90
    else uhol := 270;
  end
  else begin                     { x<>0, môžeme použiť arctan }
    vysl := arctan(y/x) / pi * 180;
    if x < 0 then vysl := vysl - 180;  { zmeň výsledok podľa kvadrantu }
    if vysl < 0 then vysl := vysl + 360;
    uhol := vysl;
  end;
```

```
end; { uhol }

procedure nacitaj;
var i : integer;
begin { načítaj číslo n a súradnice n bodov }
   read(n);
   for i := 1 to n do read(a[i].x, a[i].y);
end; { nacitaj }

procedure tried(zac, kon : integer);
var pom : bod;
var i, j, min : integer;
begin { tried pole a od pozície „zac“ po pozíciu „kon“ podľa uhla }
   for i := zac to kon − 1 do begin
      min := i;    { nájdi minimum v neutriedenej časti pola }
      for j := i + 1 to kon do begin
         if a[j].uhol < a[min].uhol then min := j;
      end;
      { ulož minimum do a[i] }
      pom := a[i]; a[i] := a[min]; a[min] := pom;
   end;
end; { tried }

procedure priprav_pole;
var i : integer;
begin { sprav predvýpočty na pripravenie poľa a }
   for i := 1 to n do    { každému bodu spočítaj uhol }
      a[i].uhol := uhol(a[i].x, a[i].y);
   tried(1, n);          { utried pole podľa uhlov }
   for i := 1 to n do    { urob kópiu každého bodu }
      a[i + n] := a[i];
   a[0].sucet := 0;      { spočítaj čiastočné súčty uhlov }
   for i := 1 to 2 ∗ n do
      a[i].sucet := a[i − 1].sucet + a[i].uhol;
end; { priprav_pole }

function poloha(kto, odkoho : integer) : integer;
begin { zisti, v akej polohe je bod „kto“ vzhľadom na bod „odkoho“ }
   if a[kto].uhol = a[odkoho].uhol then poloha := priamo
   else if a[odkoho].uhol < a[kto].uhol then begin
      if a[kto].uhol < a[odkoho].uhol + 180 then poloha := vlavo
      else poloha := vpravo;
   end
   else begin
      if a[kto].uhol < a[odkoho].uhol − 180 then poloha := vlavo
      else poloha := vpravo;
   end;
end;
```

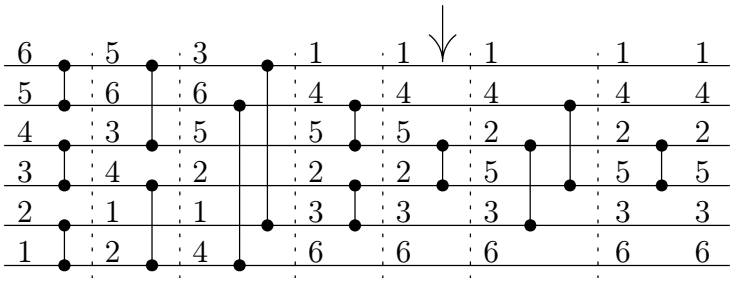
```
function sucet_uhlov(smer, posl_vlavo : integer) : real;
var sucet : real;
begin { spočítaj súčet uhol, ak je učiteľ otočený k žiakovi „smer“,
        „posl_vlavo“ je posledný žiak, ktorý je naľavo od učiteľa }
   { započítaj žiakov naľavo od učiteľa (od smer+1 po posl_vlavo) }
   sucet := (a[posl_vlavo].sucet - a[smer].sucet) -
            (posl_vlavo - smer) * a[smer].uhol;
   { niektorí žiaci naľavo majú menší uhol }
   if posl_vlavo > n then
      sucet := sucet + (posl_vlavo - n) * 360;
   { započítaj žiakov napravo od učiteľa (od posl_vlavo+1 po smer+n-1) }
   sucet := sucet + ((smer + n - 1) - posl_vlavo) * a[smer].uhol -
            (a[smer + n - 1].sucet - a[posl_vlavo].sucet);
   { niektorí žiaci napravo majú väčší uhol }
   if posl_vlavo + 1 <= n then
      sucet := sucet + (n - posl_vlavo) * 360;

   sucet_uhlov := sucet;
end;

function najdi_posl_vlavo(smer, zac : integer) : integer;
var posl_vlavo : integer;
begin { nájdi posledného žiaka naľavo od smeru „smer“,
        začni hľadať v bode „zac“ }
   posl_vlavo := zac;
   while (poloha(posl_vlavo, smer) <> vpravo) and
      (posl_vlavo < smer + n) do begin
         posl_vlavo := posl_vlavo + 1;
      end;
   najdi_posl_vlavo := posl_vlavo - 1;
end;

var smer, posl_vlavo, min_smer : integer;
var sucet, min : real;
begin { **** hlavný program **** }
   nacitaj;            { načítaj vstup }
   priprav_pole;       { trieď, predpočítaj uhly a ich súčty}

   smer := 1;          { spočítaj otočenie smerom k žiakovi 1 }
   posl_vlavo := najdi_posl_vlavo(smer, 1);
   min := sucet_uhlov(smer, posl_vlavo);
   min_smer := 1;

   while smer < n do begin { kým sa neprezrú všetky body }
      smer := smer + 1;    { otoč sa k ďalšiemu žiakovi }
      posl_vlavo := najdi_posl_vlavo(smer, posl_vlavo);
      sucet := sucet_uhlov(smer, posl_vlavo);
      if sucet < min then begin { porovnaj súčet uhlov s doterajším minimom }
         min := sucet;
```

```
        min_smer := smer;
      end;
    end;
    { vypíš výsledok }
    writeln('Otocit sa v smere [', a[min_smer].x : 0 : 1, ',',
            a[min_smer].y : 0 : 1, ']');
end.
```

P-II-4

Časť a)

Správnou odpoveďou je napríklad vstup $6, 5, 4, 3, 2, 1$ (ale aj mnoho iných vstupov). Pre vstup $6, 5, 4, 3, 2, 1$ výpočet siete prebieha nasledovne:



Časť b)

Treba odstrániť komparátor vyznačený šípkou na predchádzajúcom obrázku (jediné správne riešenie). Dokážme teraz, že po odstránení tohto komparátora sieť triedi. V prvých troch vrstách sa minimum dostane na prvý vodič a maximum na posledný vodič. Môžeme to dokázať nasledujúcim spôsobom. Po prvej vrstve je minimum na niektorom z vodičov 1, 3, alebo 5. Po druhej vrstve je na vodiči 1 alebo 5 a po tretej vrstve je určite na vodiči 1. Podobne maximum je po prvej vrstve na vodiči 2, 4 alebo 6, po druhej vrstve na vodiči 2 alebo 6 a po tretej musí byť na vodiči 6.

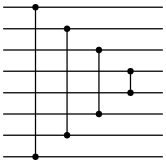
Prvý a posledný vodič teda po tretej vrstve obsahujú správne hodnoty. Ďalšie tri vrstvy striedia prostredné štyri vodiče. Štvrtá vrstva umiestni minimum z druhého a tretieho vodiča na druhý vodič. Podobne piaty vodič obsahuje maximum zo štvrtého a piateho vodiča. Piata vrstva pôvodnej siete sa vynecháva. V šiestej vrstve porovnáme medzi sebou maximá a minimá z predchádzajúceho kroku, t.j. po tejto vrstve už druhý vodič obsahuje minimum a piaty vodič maximum zo zvyšných štyroch prvkov. Zostáva dotriediť prostredné dva vodiče, čo spraví komparátor v poslednej vrstve.

Časť c)

Označme x_i číslo na vstupe i. Dokážme najprv nasledujúce pozorovanie: pre každé $i \leq n$ jedno z čísel x_i a x_{2n-i+1} patrí medzi n najmenších čísel a druhé medzi n najväčších. Predpokladajme, že obe čísla x_i a x_{2n-i+1} patria medzi n najmenších čísel. Potom čísla $x_1, x_2, \ldots, x_{i-1}$ patria tiež medzi n najmenších čísel, lebo sú menšie ako x_i. Podobne aj čísla $x_{n+1}, x_{n+2}, \ldots, x_{2n-i}$ patria medzi n najmenších čísel, lebo sú menšie ako x_{2n-i+1}. To znamená, že sme spolu identifikovali $i + (2n - i + 1 - n) = n + 1$ čísel, ktoré patria medzi n najmenších, čo je spor. Aspoň jedno z čísel x_i a x_{2n-i+1} musí teda patriť medzi n najväčších čísel.

Predpokladajme teraz, že obe čísla x_i a x_{2n-i+1} patria medzi n najväčších čísel. Potom ale aj čísla $x_{i+1}, x_{i+2}, \ldots, x_n$ a $x_{2n-i+2}, x_{2n-i+3}, \ldots, x_{2n}$ patria medzi n najväčších čísel, lebo sú väčšie ako x_i alebo x_{2n-i+1}. Spolu sme teda identifikovali $(n-i+1)+(2n-2n+i) = n+1$ čísel, ktoré patria medzi n najväčších, čo je spor. Teda nemôžu obe čísla patriť ani medzi n najväčších.

Dokázali sme teda, že jedno z čísel x_i a x_{2n-i+1} patrí do hornej polovice výstupov a druhé do dolnej. Do hornej patrí to z nich, ktoré je menšie. Stačí ich teda porovnať jedným komparátorom a každé sa dostane do správnej polovice výstupov. Toto spravíme pre každú dvojicu x_i a x_{2n-i+1} pre $i = 1, 2, \ldots, n$. Naša sieť teda bude mať $n/2$ komparátorov v jednej vrstve. Príklad takejto siete pre 8 vstupov ($n = 4$) uvádzame na obrázku.



SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

51. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Vzorové riešenia II. kola kategórie P

Vydala IUVENTA
pre vnútornú potrebu Ministerstva školstva SR
Autori príkladov B. Brejová, M. Forišek, M. Pál a T. Vinař
Zodpovedný redaktor M. Forišek
Sadzba programom LaTeX