A-III-1 Bizónia rezervácia

Na vstupe máme n priamok. Tie majú dokopy $n(n-1)/2$ priesečníkov. Hľadaná ohrada je zjavne konvexným obalom týchto n bodov. Stačí teda všetkých $\Theta(n^2)$ bodov zostrojiť a následne v čase $\Theta(n^2 \log n)$ nájsť ich konvexný obal.
Existuje však aj efektívnejšie riešenie:

- Začneme tým, že si všetky priamky usporiadame podľa uhlu, ktorý zvierajú s osou x. (Pri implementácii toto vieme spraviť aj v celých číslach pomocou vektorového súčinu. Ekvivalentne sa na to môžeme dívať tak, že priamky usporadúvame podľa ich smernice – pri tom by ale bolo treba špeciálne ošetriť zvislú priamku, ak takú máme.)
- Následne spočítame priesečníky, ale len pre dvojice priamok, ktoré v usporiadanom poradí spolu susedia, a to vrátane priesečníku prvej a poslednej priamky.
- Na záver už len nájdeme konvexný obal pre n priesečníkov, ktoré sme zostrojili v predchádzajúcom kroku.

Vyššie popísané riešenie má časovú zložitosť $\Theta(n \log n)$. Potrebujeme však dokázať, že naozaj funguje – teda že naozaj vždy všetky vrcholy hľadaného konvexného obalu sú priesečníkmi susedných priamok v našom usporiadanom poradí.

Jeden jednoduchý argument vyzerá nasledovne:
Na úvod si dokážeme, že najpravejší spomedzi všetkých priesečníkov je priesečníkom dvoch priamok, ktoré susedia v našom usporiadanom poradí. Predstavme si zvislú priamku p napravo od posledného priesečníka. Priesečníky našich priamok s priamkou p zodpovedajú nášmu usporiadanému poradiu – čím má priamka väčšiu smernicu, tým vyššie priamku p pretne. No a keď teraz budeme posúvať priamku p doľava, poradie, v ktorom ju naše priamky pretínajú, sa nebude meniť, až kým neprídeme k prvému (teda najpravejšiemu) priesečníku. A v ňom sa teda musia stretnúť niektoré dve priamky, ktoré v našom usporiadanom poradí susedili.

Práve sme si dokázali, že najpravejší spomedzi všetkých priesečníkov je priesečníkom dvoch priamok, ktoré susedia v našom usporiadanom poradí. Lenže smer „doprava“ nie je ničím špeciálny. Tento istý výsledok platí v úplne každom smere. Nech si vyberieme akýkoľvek smer, posledný priesečník v tom smere musí byť priesečníkom niektorých dvoch priamiek, ktoré v našom usporiadanom poradí susedia. (Dôkaz: Otočíme rovinu tak, aby nami zvolený smer padol na os x. Následne zopakujeme predchádzajúci argument.)

No a teraz celý dôkaz uzavrieme. Na to si už stačí len uvedomiť, že každý vrchol hľadaného konvexného obalu je nutne najvzdialenejším priesečníkom v nejakom smere.
Toto vyplýva z toho, že v každom vrchole má konvexný obal vnútorný uhol ostro menší ako 180°. Dotyčným vrcholom teda vieme preložiť priamku q tak, aby celý konvexný obal ležal na jednej strane priamky q. Potom ale je náš vrchol zjavne najvzdialenejším priesečníkom v smere kolmom na priamku q.
Všetky vrcholy hľadaného konvexného obalu teda naozaj ležia medzi priesečníkmi priamok, ktoré susedia v poradí usporiadanom podľa uhlu.

Listing programu (C++)

```
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <cassert>
#include <vector>
#include <cmath>
using namespace std;

const double EPSILON = 1e-7;

struct line { double a, b, c; };
struct point { double x, y; };

// operátor < na usporiadanie priamiek podľa smeru
bool operator< (const line &A, const line &B) {
    double ux = -A.b, uy = A.a, vx = -B.b, vy = B.a;
```

```
    if (uy < -EPSILON || (uy < EPSILON && ux < -EPSILON)) { ux = -ux; uy = -uy; }
    if (vy < -EPSILON || (vy < EPSILON && vx < -EPSILON)) { vx = -vx; vy = -vy; }
    return (ux*vy - vx*uy) > EPSILON;
};

// operátor < na usporiadanie bodov zľava doprava
bool operator< (const point &A, const point &B) {
    return A.x + EPSILON < B.x || ( abs(A.x-B.x) < EPSILON && A.y + EPSILON < B.y );
}

// vektorový súčin: vráti >0 / 0 / <0 podľa toho, či OAB zatáča proti smeru ručičiek / ide rovno / zatáča v smere
long long cross(const point &O, const point &A, const point &B) { return (A.x-O.x)*(B.y-O.y)-(A.y-O.y)*(B.x-O.x); }

// konvexný obal pomocou Grahamovho algoritmu
vector<point> convex_hull(vector<point> P) {
    int n = P.size(), k = 0;
    vector<point> H(2*n);
    for (int i = 0; i < n; i++) {
        while (k >= 2 && cross(H[k-2], H[k-1], P[i]) < EPSILON) k--;
        H[k++] = P[i];
    }
    for (int i = n-2, t = k+1; i >= 0; i--) {
        while (k >= t && cross(H[k-2], H[k-1], P[i]) < EPSILON) k--;
        H[k++] = P[i];
    }
    H.resize(k-1);
    return H;
}

// priesečník priamiek
point intersection(const line &A, const line &B) {
    point answer;
    answer.x = (A.b * B.c - B.b * A.c) / (B.b * A.a - A.b * B.a);
    answer.y = (A.a * B.c - B.a * A.c) / (A.b * B.a - A.a * B.b);
    return answer;
}

int main() {
    int N;
    cin >> N;
    vector<line> L(N);
    for (int n=0; n<N; ++n) cin >> L[n].a >> L[n].b >> L[n].c;
    sort(L.begin(), L.end());

    vector<point> P;
    for (int n=0; n<N; ++n) P.push_back( intersection( L[n], L[(n+1)%N] ) );
    sort(P.begin(), P.end());

    vector<point> H = convex_hull(P);
    for (const auto &pt : H) cout << pt.x << "␣" << pt.y << endl;
}
```

A-III-2 Kvietky

Začnime tým, že si vyriešime ľahšiu verziu súťažnej úlohy. Čo by sa stalo, keby žiaden kvet nekvitol cez polnoc? Všimnime si rastlinu, ktorá ako prvá zatvorí svoje kvety (teda má najmenšie b_i). Tento kvet musíme niekedy odfotiť. No a kedy to spraviť? Zjavne najlepším momentom na jeho odfotenie je práve čas b_i, kedy rastlina zatvára kvety – totiž keby sme fotili skôr, neodfotíme žiadne iné kvety ako v čase b_i, jedine sa nám môže stať, že niektoré z tých kvetov, ktoré sú v čase b_i otvorené, v skoršom čase ešte otvorené neboli.
Vieme teda, kedy spraviť prvú fotografiu. Čo teraz ďalej? Zistíme si, ktoré rastliny sme už odfotili, a tie už nebudeme brať do úvahy. Zostala nám teda nejaká nová, menšia množina ešte neodfotených rastlín. No a teraz je to už ľahké: jednoducho pre ne zopakujeme predchádzajúcu úvahu. Opäť teda nájdeme medzi neodfotenými rastlinami tú, ktorá najskôr zatvorí kvety, a v čase, kedy sa tak stane, všetky rastliny odfotíme.
Priamočiara implementácia tohto postupu by mala časovú zložitosť $\Theta(n^2)$, keďže sa môže stať, že budeme postupne robiť $\Theta(n)$ fotografií a kvôli každej potrebujeme prejsť a spracovať všetky ešte neodfotené rastliny.

Existuje aj šikovnejšia implementácia. Všetky okamihy, kedy nejaká rastlina otvára alebo zatvára kvety, nazveme *udalosti*. Všetky udalosti si usporiadame podľa času kedy nastanú (s tým, že ak sa viacero udeje v tom istom čase, tak budeme kvety najskôr otvárať a až potom iné zatvárať). Toto nám následne umožní prechádzať cez udalosti zľava doprava a udržiavať si množinu kvitnúcich ale ešte neodfotených kvetov. Vždy, keď sa v nejakom čase zatvorí ešte neodfotený kvet, zväčšíme počet fotiek a označíme všetky práve otvorené kvety ako odfotené.

Takéto riešenie má kvôli triedeniu časovú zložitosť $O(n \log n)$. Samotný prechod udalosťami a ich spracovanie vieme implementovať aj v lineárnom čase.

Vyššie sme si vyriešili úlohu, kde sa na každý kvet môžeme dívať ako na interval na číselnej osi. Vráťme sa teraz k pôvodnej úlohe. Na tú sa môžeme dívať podobne, ale tentokrát intervaly, počas ktorých majú jednotlivé rastliny otvorené kvety, ležia akoby na kruhu.
Ukážeme si najskôr riešenie s časovou zložitosťou $\Theta(n^2)$. Začneme tým, že si rovnako ako v predchádzajúcom riešení usporiadame všetky udalosti. (Toto stačí spraviť raz na začiatku, ich poradie sa nijak meniť nebude.)
Predstavme si teraz, že Šandyna prišla a v nejakom čase t_0 spravila fotografiu. Čo sa teraz stane? Keď prestaneme brať do úvahy rastliny, ktorých kvety práve odfotila, dostaneme vlastne úlohu „na priamke“. Čas t_0 hrá pre zostávajúce rastliny úlohu polnoci: žiadna z nich v tomto čase nemá otvorené kvety. Ak by sme teda poznali čas t_0, vieme nájsť optimálne ostatné časy fotenia v lineárnom čase. (Vyššie popísaným algoritmom postupne prejdeme všetky udalosti týkajúce sa ešte neodfotených kvetov, začneme ale v čase t_0, nie v čase 0.)
My síce čas t_0 nepoznáme, nič nám ale nebráni vyskúšať „všetky“ možnosti. Ľahko nahliadneme, že ako t_0 má zmysel skúšať len časy b_i v ktorých niektorá rastlina zatvára kvety. (Hocijaký iný čas fotenia vieme posunúť na neskôr bez toho aby nám zo záberu ubudol nejaký kvet.)
Dokopy teda máme n možností pre čas t_0 a každú vieme spracovať v lineárnom čase. Spomedzi takto zostrojených riešení už len vyberieme najlepšie a sme hotoví.

Ako vyššie popísané riešenie vylepšiť?
Zaveďme značenie $f(t)$ nasledovne: ak sme prvú fotku spravili v čase t, kedy je optimálne spraviť druhú? (Ak je viac možností, tak berieme tú najneskoršiu z nich, podobne ako vo všetkých doteraz popísaných riešeniach. Platí teda, že $f(t)$ je čas, kedy prvýkrát prestane mať otvorené kvety niektorá z rastlín, ktoré svoje kvety otvorili až po fotografii v čase t.)
Ak by sme poznali hodnoty $f(t)$, vedeli by sme ľahko implementovať vyššie popísané riešenie: postupne pre každé t_0 by sme postupne robili fotky v časoch $t_0, f(t_0), f(f(t_0)), \ldots$, až kým by sme „neobišli dokola celý kruh“, teda kým by nenastala situácia, že pri prechode z nejakého času x na nasledujúci čas $f(x)$ prídeme naspäť na t_0, prípadne dokonca prejdeme cez t_0. V takomto prípade už v čase $f(x)$ fotiť netreba, už máme určite odfotené všetko.
Do úplného riešenia nám ostávajú ešte dva kroky. V prvom rade potrebujeme ukázať, ako hodnoty $f(t)$ efektívne vypočítať. No a v druhom rade potrebujeme ukázať, ako pomocou nich efektívnejšie vyriešiť našu úlohu – vyššie popísané riešenie je totiž v najhoršom prípade stále ešte kvadratické od počtu rastlín.

Predpočítanie hodnôt $f(t)$ bude veľmi podobné úplne prvému algoritmu. Presnejšie, namiesto hodnôt $f(t)$ v podobe, v akej sme si ich definovali vyššie, si pre každú rastlinu i predpočítame číslo rastliny $g(i)$ takej, že $f(b_i) = b_{g(i)}$. Inými slovami, ak sme rastliny odfotili v čase, kedy rastlina i zatvárala kvety, najbližšie ich budeme fotiť v čase, kedy kvety zatvára rastlina $g(i)$.
Ako na to? Použijeme dva prsty. Ľavý prst začne na najmenšom zo všetkých b_i. Pravý prst tiež. Následne budeme pravý prst posúvať doprava po udalostiach (t.j. dopredu v čase), až kým nenájdeme prvú rastlinu, ktorá po čase b_i otvorila aj zavrela kvety. V tejto chvíli sme našli hodnotu $g(i)$ a máme (podobne ako v úplne prvom algoritme) zapamätanú množinu všetkých kvetov, ktoré sa otvorili od času b_i.
Následne budeme dokola opakovať nasledovné: najskôr posúvame ľavý prst doprava po udalostiach, až kým neprídeme na ďalšie zavretie kvetu. Ak sme medzi tým stretli nejaké otvorenia kvetov, tieto rastliny vyhodíme z aktuálne pamätanej množiny. No a keď už ľavý prst ukazuje na nejaký čas zatvorenia kvetu, posúvame doprava zase pravý prst, a to dovtedy, kým zase nenájdeme najbližší nepokrytý koniec kvitnutia.
Počas tohto algoritmu ľavý prst postupne raz prejde po všetkých udalostiach a pravý prst prejde po každej udalosti nanajvýš dvakrát (raz na začiatku a raz počas prechádzania dokola). Dokopy má teda toto predpočítanie lineárnu časovú zložitosť.

Teraz už teda pre každý čas fotenia poznáme optimálny nasledujúci čas fotenia. Ako pomocou nich šikovne nájsť optimálne riešenie, teda minimálny počet fotení?
Prechod na optimálnu nasledujúcu fotku nazvime *krok*. Pre každý možný začiatok sme si predpočítali, kam sa dostaneme na jeden krok a koľko času tým preskočíme.

Z týchto údajov teraz vieme pre každý začiatok spočítať, kam sa dostaneme na dva kroky.
Keď budeme mať spočítané to, budeme vedieť pre každý začiatok spočítať, kam sa dostaneme na **štyri** kroky. Stačí sa vždy dvakrát pozrieť do predchádzajúcej tabuľky. Keď napr. chceme zistiť, kam sa dostaneme z i na 4 kroky, zistíme si, kam sa dostaneme na 2, a následne sa pozrieme, kam sa odtiaľ dostaneme na ďalšie 2.
Rovnako si spočítame túto informáciu pre 8, 16, 32, ... krokov. Prestať môžeme pri n, keďže viac ako n krokov určite nespravíme. Takto si teda predpočítame $\Theta(n \log n)$ pomocných údajov, a to v čase $\Theta(n \log n)$.
No a pomocou takto predpočítaných hodnôt vieme teraz pre ľubovoľný začiatok v čase $O(\log n)$ zistiť minimálny počet krokov potrebný na to, aby sme pokryli všetky rastliny. To spravíme tak, že postupne skúšame pridať ..., 32, 16, 8, 4, 2 a 1 krokov, ale spravíme to vždy len vtedy, ak celkový čas neprekročí jeden celý deň.
Takto teda dostávame riešenie s časovou aj pamäťovou zložitosťou $\Theta(n \log n)$.
Na záver ešte pre zaujímavosť podotkneme, že jediným skutočne úzkym hrdlom je triedenie. Ak by sme mali vstup, pre ktorý vieme udalosti usporiadať v lineárnom čase, vedeli by sme následne v lineárnom čase predpočítať hodnoty $f(i)$ a následne existuje iný spôsob, ako z týchto hodnôt zistiť optimálnu odpoveď v lineárnom čase. Takto by sme teda dostali riešenie bežiace v čase $\Theta(n)$. Detaily tohto riešenia už nebudeme uvádzať.

Z priestorových dôvodov bude program zverejnený až v elektronickej verzii týchto riešení.

A-III-3 Stromochod

Postupne si ukážeme riešenia všetkých troch súťažných úloh.

Súťažná úloha, podúloha A (1 bod)

Na to, aby sme overili, či je daný strom cesta doľava, stačí ho raz celý prejsť (napríklad postupom uvedeným v študijnom texte) a priamo v pamäti stromochodu si pamätať v pomocnej boolovskej premennej, či sme niekedy navštívili vrchol, ktorý by mal pravého syna. Program môžeme (ale nemusíme) vylepšiť tým, že aj ak nejaký vrchol má pravého syna, nepôjdeme doň.

Listing programu (Pascal)

```
var videl_praveho : boolean;

type vrchol = record
   ok : boolean;
   stav : 0..3;
end;

begin
   videl_praveho := false;
   while true do begin
      V.stav := V.stav + 1;
      case V.stav of
         1: if ex_l then krok_l;
         2: if ex_p then videl_praveho := true;
         3: if not ex_o then begin
            V.ok := not videl_praveho;
            halt;
         end else krok_o;
      end;
   end;
end.
```

Súťažná úloha, podúloha B (2 bod)

Stred cesty doľava vieme nájsť napríklad nasledovne: Budeme dokola prechádzať celý strom. (Opäť môžeme použiť postup zo študijného textu, prípadne ho môžeme zjednodušiť, keďže vieme, že žiaden vrchol nemá pravého syna.) Pri každom prechode stromom ofarbíme dva vrcholy: najvyšší, ktorý nie je červený, ofarbíme na červeno, a najnižší, ktorý nie je modrý, ofarbíme na modro. Celé to skončí tým, že v niektorej iterácii ofarbíme ten istý vrchol na červeno aj na modro. Tento vrchol je potom zjavne stredom cesty, označíme ho teda a skončíme.

Listing programu (Pascal)

```
var dal_cervenu, dal_modru, idem_dodola : boolean;

type vrchol = record
   cerveny, modry, stred : boolean; { na začiatku je všade false }
   stav : 0..3;
end;

begin
   while true do begin
      { začíname novú iteráciu, inicializujeme si naše lokálne premenné }
      dal_cervenu := false;
      dal_modru := false;
      idem_dodola := true;
      { prechádzame stromom }
      while true do begin
          { ofarbíme aktuálny vrchol, ak treba }
          if idem_dodola then begin
             if (not dal_cervenu) and (not V.cerveny) then begin
                V.cerveny := true;
                dal_cervenu := true;
             end;
          end else begin
             if (not dal_modru) and (not V.modry) then begin
                V.modry := true;
                dal_modru := true;
             end;
             if V.cerveny and V.modry then begin
                { našli sme stred }
                V.stred := true;
                halt;
             end;
          end;
          { a posunieme sa do ďalšieho vrcholu }
          if idem_dodola then begin
             if ex_l then krok_l else idem_dodola := false;
          end else begin
             if ex_o then krok_o else break;
          end;
      end;
   end;
end.
```

Rýchlejšie riešenie podúlohy B

Toto riešenie nebolo potrebné, ale jeho myšlienka nám bude užitočná neskôr. Ukážeme si, ako nájsť stred v čase $\Theta(n \log n)$, teda na rádovo $\log n$ prechodov cestou.

Na začiatku ofarbíme všetky vrcholy na červeno. Pripomíname, že ich je nepárny počet. Následne budeme dokola opakovať nasledovný postup: cestou zdola hore si spočítame, či je aktuálny počet červených vrcholov tvaru $4k + 1$ alebo tvaru $4k + 3$. Ak je tvaru $4k + 1$, cestou zhora dole vždy jeden červený vrchol necháme a jeden prefarbíme na bielo. Ak je tvaru $4k + 3$, robíme to isté ale s opačnou paritou – teda prvý červený vrchol prefarbíme na bielo, druhý necháme, atď.

Ľahko nahliadneme, že počet červených vrcholov sa vždy zmenší na zhruba polovicu a že stred cesty ostane po každom prechode červený. Preto po rádovo $\log n$ prechodoch budeme už mať len jeden červený vrchol – samotný stred cesty. (A to, že nám ostal už len jeden červený vrchol, tiež vieme zistiť počas prechodu ňou.)

Súťažná úloha, podúloha C (7 bodov)

Strom je dokonale vyvážený, ak v každom vrchole platí, že sa veľkosti jeho ľavého a pravého podstromu líšia nanajvýš o 1. V tejto podúlohe chceme o danom strome zistiť, či má túto vlastnosť. Základná myšlienka oboch riešení, ktoré si ukážeme, bude jednoduchá: postupne skontrolujeme každý vrchol.

Predstavme si napríklad, že chceme začať od koreňa. Ako skontrolujeme, či v ňom platí naša podmienka? Ak by sme mali kopec pamäte, stačilo by si spočítať vrcholy naľavo a napravo a tieto dva počty porovnať. My však máme pamäte len konečne veľa a toto si nemôžeme dovoliť. Máme však aj iné možnosti ako porovnať dva počty. Asi najjednoduchšou možnosťou je postupne „škrtať“ vrcholy striedavo z ľavého a pravého podstromu. Myšlienka je podobná ako v riešení podúlohy B: dokola prechádzame celým stromom a v každom prechode označíme jeden neoznačený vrchol v ľavom a jeden v pravom podstrome. Ak sa nám oba podstromy minú „skoro naraz“, je v koreni podmienka splnená, inak nie.

Ako spoznať, že sa oba podstromy „skoro naraz“ minuli: počas prechodu stromom si o každom podstrome stačí napríklad pamätať, či ešte obsahuje 0, 1, alebo viac ako 1 neoznačený vrchol.
Celé riešenie teraz bude vyzerať nasledovne: V cykle prechádzame celý strom. Vždy, keď narazíme na vrchol, ktorý ešte nebol skontrolovaný, označíme ho ako práve kontrolovaný a spustíme v jeho podstrome vyššie popísaný podprogram.
Ako dlho to celé bude trvať? Kontrola konkrétneho vrcholu, pod ktorým je dokopy k vrcholov, môže mať až $k/2$ kôl, pričom v každom kole prejdeme všetkých k vrcholov. Dokopy teda takáto kontrola spraví približne $k^2/2$ krokov.
Pozrime sa najskôr na situáciu, kedy naozaj kontrolujeme dokonale vyvážený strom. Pri kontrole koreňa spravíme zhruba $n^2/2$ krokov. Pri kontrole každého z jeho dvoch synov spravíme $(n/2)^2/2$ krokov. V hĺbke 2 máme 4 vrcholy, pod každým z nich sa nachádza zhruba $n/4$ vrcholov, a teda pri kontrole každého z nich spravíme zhruba $(n/4)^2/2$ krokov. A tak ďalej. No a ľahko nahliadneme, že ide o geometrickú postupnosť: celkový počet krokov spravených počas kontroly vrcholov v hĺbke h je zhruba $n^2/2^{h+1}$, a teda úplne všetkých krokov dokopy je zhruba n^2.
No a v situácii, kedy strom, ktorý kontrolujeme, dokonale vyvážený nie je, spravíme dokopy tiež nanajvýš n^2 krokov. To preto, že kým kontrolujeme dokonale vyvážené vrcholy, robíme to isté ako vyššie, a akonáhle narazíme na prvý zlý, tak pri jeho kontrole ešte možno spravíme nejaké kroky, ale tých bude nanajvýš $n^2/2$ a hneď po nich celú kontrolu ukončíme.

Rýchlejšie riešenie podúlohy C

Ako šikovnejšie skontrolovať, či sú oba podstromy vrcholu v zhruba rovnako veľké? Upravíme vyššie popísané rýchlejšie riešenie podúlohy B. Najskôr si skontrolujeme paritu celkového počtu vrcholov v podstrome s koreňom v (vrátane v). Ak je vrcholov nepárny počet, musí byť v presne v strede in-order zápisu tohto stromu. Toto overíme práve algoritmom z časti B, len namiesto cesty ho robíme na strome, po ktorom dokola prechádzame prehľadávaním do hĺbky.
Ak je vrcholov párny počet, urobíme to isté dvakrát, pričom raz odignorujeme jeden vrchol z ľavého a raz jeden z pravého podstromu. Ak hociktorá kontrola uspeje, vrchol v je v poriadku, ak ani jedna neuspeje, strom je zlý. Kontrola vrcholu, pod ktorým je dokopy k vrcholov, nám teda trvá $\Theta(k \log k)$. Dá sa ukázať, že dokopy za celý strom sa nám to nasčíta na $\Theta(n \log^2 n)$ krokov.

Existuje ešte rýchlejšie riešenie? Túto otázku ponecháme otvorenú.

TRIDSIATY DRUHÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Eduard Batmendijn, Michal Forišek, Jaroslav Petrucha
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2017