Korešpondenčný seminár z programovania
XXIII. ročník, 2005/06
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: aSc – Applied Software Consultants spol. s r.o.
MICROSTEP-MIS spol. s r.o.
Gnome spol. s r.o.

Vzorové riešenia 1. kola letnej časti

Milé naše riešiteľky, milí naši riešitelia.

Dlhý a ťažký bol pôrod týchto vzorových riešení. Hlavnou príčinou bolo veľké sťahovanie celého KSP z miestnosti i33 do miestnosti T2. No, čo si budeme rozprávať, môžete byť radi, že vaše riešenia nepadli sťahovaniu za obeť. Keď sa už prach usadil naspäť na skrine a my naspäť za počítače, spísali sme vzorové riešenia, zadali body, a tu to celé máte. Dobrú chuť!

Kto číta len výsledkovku, bude blbý ako tágo!

KSPáci

1. O lúpeži

opravoval MMx
(max. 15 bodov)

V tomto príklade sa dal naplno využiť postup nazývaný dynamické programovanie. Spočíva v tom, že problém, ktorý potrebujeme riešiť, si rozdelíme na niekoľko rovnakých podproblémov. Ich riešenie potom vyskladáme z riešení menších problémov, pričom niektoré budú natoľko triviálne, že ich vieme vyriešiť hneď.

Najprv si ukážeme riešenie zjednodušeného problému, pri ktorom je bankoviek každého typu ľubovoľne veľa. Budeme používať pole A veľkosti M. V ňom si budeme pre každú sumu hodnôt bankoviek pamätať, či ju zatiaľ vieme dosiahnuť. Na začiatku bude $A[0] = 1$ a celý zvyšok poľa bude nulový (teda na začiatku nevieme dosiahnuť žiadnu sumu). Teraz budeme postupne pridávať jednotlivé typy bankoviek. Typ bankoviek s hodnotou h pridáme nasledovne: Prejdeme celé pole, a ak nájdeme sumu, do ktorej sa nevieme dostať, ale do sumy o h menšej sa dostať vieme, tak sme práve našli spôsob, ako aj túto sumu dosiahnuť.

Teraz sa ešte potrebujeme vyhnúť tomu, aby sme použili viac bankoviek nejakého typu, ako je v trezore. Na to použijeme pomocné pole *pom*, v ktorom si budeme pre každú sumu pamätať, koľko najmenej bankoviek práve pridávaného typu potrebujeme použiť na jej vyrobenie. Ak nájdeme sumu, ktorú vieme dosiahnuť pomocou práve pridávaného typu, ale presiahli by sme tým počet takýchto bankoviek, tak ju jednoducho nepridáme. Toto pole budeme aktualizovať zakaždým, keď zistíme nejakú novú sumu, ktorú vieme dosiahnuť.

Časová zložitosť takéhoto riešenia je $O(NM)$, pretože pre každý z N typov bankoviek prejdeme celé pole veľkosti M. Pamäťová zložitosť je $O(M)$, pretože nám stačí pamätať si dve polia veľkosti M. Za takéto riešenie sa dalo získať 15 bodov. 12 bodov sa dalo získať za riešenie, ktoré prehľadávalo pole A pre každú bankovku zvlášť, teda malo zložitosť $O(NMp)$, kde $p = max\{p_i\}$. Za riešenie, ktoré skúšalo všetky kombinácie počtov bankoviek pre jednotlivé typy, sa dalo získať 9 bodov. Vyskytli sa ale aj riešenia, ktoré pre každú bankovku zvlášť určovali, či vo výslednej sume bude alebo nie. Toto je ale zbytočné, pretože ak sa rozhodnem zobrať n_i bankoviek s hodnotou h_i, potom je jedno, ktorú n_i-ticu vyberiem, všetky budú mať rovnakú hodnotu. Najviac bodov za takéto riešenie bolo 7. O body sa dalo prísť za chýbajúci alebo nedostatočný popis, zlý odhad zložitostí alebo chýbajúci program.

Listing programu:

```
#include <cstdio>
#include <strings.h>

int main(void) {
    int n, m, h, p, vysledok=0;
    scanf("%d %d", &n, &m);
    int a[m], pom[m];
    bzero(a, m*sizeof(int)); a[0]=1;

    for (int i=0; i<n; i++) {
        scanf("%d %d", &h, &p);
        bzero(pom, m*sizeof(int));
        for (int j=h; j<m; j++)
            if (a[j]==0 && a[j-h]==1 && pom[j-h]<p) {
                a[j]=1; pom[j]=pom[j-h]+1;
                if (vysledok<j) vysledok=j;
            }
    }
    printf("%d\n", vysledok);
    return 0;
}
```

2. O Zázvorovom pive III

opravoval KuKo
(max. 15 bodov)

Ahojte, pozdravuje vás Zachariáš, že ďakuje za programy, máte to uňho. Aj ja to mám uňho, za to, že som tie vaše riešenia pobodoval a že som to pobodoval takto:

- $O(N^2)$ (alebo $O(M \log N)$) bolo najlepšie – 15 bodov
- našli sa takí, čo mali $O(M \log N)$, ale nevedeli o tom – 14 bodov
- $O(ZN^2)$, kde Z je počet závozníkov – 11 bodov
- horšie – menej

Všetci (čo niečo poslali) sa dovtípili, že tu ide o najkratšie vzdialenosti, resp., že ak zistia najkratšie vzdialenosti pre mestá, tie cesty sa už nejak dobúšia (dokopú). A naozaj, majme mestá A a B, pričom do mesta A má najbližšie závozník z_A, do B zase z_B a vzdialenosti do týchto miest sú d_A a d_B. Potom časť cesty medzi A a B bude mať na starosti z_A, časť z_B (nevylučujeme ani, že $z_A = z_B$). Ešte dĺžku cesty medzi A a B označme w_{AB} a podiel, ktorý okupuje z_A nech je x ($0 \le x \le 1$). Na ceste medzi A a B (ak $z_A \neq z_B$) je miesto, ktoré je rovnako vzdialené od z_A ako od z_B a tu platí rovnosť $d_A + x \cdot w_{AB} = d_B + (1 - x) \cdot w_{AB}$. Odtiaľ $x = (d_B - d_A)/2w_{AB} + 1/2$.

A ako nájdeme tie najkratšie cesty? Použijeme algoritmus uja Dijkstru. Dijkstra, Dijkstra, Dijkstra... napriek tomu, že je to notoricky známy algoritmus, meno uja „Dijkstru“ sa nám v riešeniach neustále komolí[1]. Nie je to ani Dijkster, ani Djikstra, ani Dixtra, ani len tá Dijkstra, je to Edsger Wybe Dijkstra. Takže vy, čo neviete napísať „Dijkstra“, si to teraz skúste 3× napísať. A vy, čo neviete napísať „Dijkstru“ (teda naprogramovať Dijkstrov algoritmus :)) si ho po prečítaní nasledujúceho odseku skúste tiež 3× napísať. Zvyšok nemusí nasledujúci odsek čítať.

Majme takúto úlohu: dané sú mestá a cesty medzi nimi. Tieto cesty majú všetky *kladnú* dĺžku (ináč by to nefungovalo). „Vypichnuté“ je jedno mesto (budeme ho volať „štart“), z

[1] to sa nám stať nemôže, pretože vzoráky si po sebe čítame :) ... na druhej strane, my si ich čítame aj navzájom, čo vy by ste nemali

ktorého chceme vedieť najkratšie cesty všade inam. Ako na to? Mestá budeme mať rozdelené do dvoch skupín: V skupine H (ako hotovo[2]) budú mestá, o ktorých vieme na betón povedať, ako sú ďaleko. Na začiatku to vieme povedať iba o štarte – ten je zo štartu vzdialený 0 (slovom nula). Ostatné mestá sú v druhej skupine. Pre ne poznáme dĺžku najkratšej cesty, ktorá ide iba cez hotové mestá (tie zo skupiny H). V každom kroku vyberieme z ešte nehotových miest to mesto m, ktoré má túto vzdialenosť najmenšiu – mesto, ktoré je od štartu najbližšie (ak budeme chodiť iba po hotových mestách). Vtip je v tom, že žiadna cesta, ktorá ide cez nehotové mestá nie je kratšia. Skúste si takú cestu predstaviť – akonáhle prídeme do prvého nehotového mesta, už sme prešli väčšiu vzdialenosť ako do m (m má túto vzdialenosť minimálnu) a všetky cesty majú kladnú dĺžku (tu to potrebujeme), takže dĺžka tejto cesty je už iba väčšia. To ale znamená, že najkratšia cesta do m nie je len najkratšia z tých, ktoré cupitajú po hotových mestách ale vôbec zo všetkých ciest do m. A preto je táto vzdialenosť „skutočná“ – s mestom m sme teda hotoví[3]. Teda m môžeme pridať medzi hotové mestá. Čo však ešte musíme urobiť je upraviť nehotovým mestám ich vzdialenosti. Povedali sme si, že pre nehotové mestá vieme dĺžku najkratšej cesty cez hotové mestá. Keď však do H pridáme aj m, môžu tieto cesty chodiť aj cez m. Jediné mestá, ktorým sa vzdialenosť mohla zmeniť sú však iba susedia mesta m. Pozrieme sa teda na všetkých susedov m a ak je cesta cez m kratšia ako doteraz nájdená, upravíme im vzdialenosti.

Uf, tak som to vopchal do jedného odstavca, tí čo skákali, chytajte sa, tu sme. Ako Dijkstrov algoritmus využijeme v našej pôvodnej úlohe? Nuž jeden horší spôsob je použiť ho Z-krát – od každého závozníka. Tak pre každé mesto zistíme najkratšiu vzdialenosť od každého závozníka a z nich vyberieme najmenšiu. Tak dostaneme riešenie s časovou zložitosťou $O(ZN^2)$.

Dá sa lepšie. Vráťme sa k popisu Dijkstrovho algoritmu: „Mestá budeme mať rozdelené do dvoch skupín: V skupine H budú mestá, o ktorých vieme na betón povedať, ako sú ďaleko.“ Keď sme chceli vedieť vzdialenosť od jedného mesta (štartu), zaradili sme do skupiny H iba štart – o ňom jedinom sme na začiatku vedeli povedať, ako je ďaleko od štartu, a to 0 (slovom nula). Naša úloha je podobná – akurát chceme vedieť vzdialenosť od nejakej skupiny miest. Algoritmus bude rovnaký, akurát na začiatku nebude v H jediný vrchol (štart), ale všetci závozníci – o nich vieme povedať, akú majú najkratšiu vzdialenosť, a to je 0 (slovom nula).

A to je všetko.

Listing programu:

```
#include <cstdio>
#define INFTY 10000000
#define REP(i,n) for(int i=0;i<(n);++i)
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)

int n, z, m;          // pocet miest, zavoznikov a ciest
int a[1000][1000]; // matica vzdialenosti
int dist[1000];     // dlzka najkratsej cesty do vrcholu
int zav[1000];       // najblizsi zavoznik
int done[1000];     // je vrchol uz spracovany?

int main() {
  scanf ("%d %d %d", &n, &z, &m);
  REP(i,n) REP(j,n) a[i][j] = INFTY;
  REP(i,n) { dist[i]=INFTY; done[i]=0; }
  REP(i,z) { dist[i]=0; zav[i]=i; }
  REP(i,m) {
    int x, y, w;
```

[2] pôvodne som chcel mať skupinu AB ako „a basta“, no ale...

[3] hotoví v zmysle „na tomto už nebudeme nič meniť, môžeme sa pohnúť ďalej“, nie v zmysle „unesení“

```
    scanf ("%d %d %d", &x, &y, &w); --x; --y;
    a[x][y] = a[y][x] = w;
  }

  int min, v;
  REP(k,n) {
    // najdeme najblizsi nespracovany vrchol v
    min=INFTY;
    REP(i,n) if (!done[i] && dist[i]<min) { min=dist[i]; v=i; }
    // spracujeme ho
    REP(i,n)
      if (dist[v]+a[v][i] < dist[i]) {
        dist[i] = dist[v]+a[v][i];
        zav[i] = zav[v];
      }
    done[v]=1;
  }

  REP(i,n)
    FOR(j,i+1, n-1)
      if (a[i][j] != INFTY) {
        printf ("cesta z %d do %d:\n", i+1, j+1);
        if (zav[i] == zav[j]) printf(" na tejto ceste je sefom %d\n", zav[i]+1);
        else {
          int x = (dist[j]-dist[i])*50/a[i][j]+50;
          printf (" zavoznik %d ma %d%% cesty\n", zav[i]+1, x);
          printf (" zavoznik %d ma %d%% cesty\n", zav[j]+1, 100-x);
        }
      }
  return 0;
}
```

3. Odpovedz, je tam hrana?

opravoval Lukáš
(max. 15 bodov)

Veľmi ste ma nepotešili,[4] lebo prišlo len jedno optimálne riešenie. Ale poďme pekne po poriadku. Najpomalšie riešenie pracovalo v čase $O(1)$ na predpočítanie a v čase $O(N)$ na jednu otázku. Na začiatku nič nespravíme a pri každej otázke prejdeme celý zoznam hrán, či sa tam daná hrana nachádza. Za takéto riešenie sa dalo získať 7 bodov.

Zrýchlenie na $O(N \log N)$ a $O(\log N)$ sa dalo dosiahnuť tak, že hrany si na začiatku utriedime. Hrany porovnávame najprv podľa prvého vrcholu. Ak majú rovnaký prvý vrchol, porovnáme ich podľa druhého vrcholu. Keď máme hrany utriedené, vieme v nich binárne vyhľadávať v čase $O(\log N)$. Za takéto riešenie som dával 10 bodov.

Objavili sa tiež dve riešenia využívajúce hašovanie. Prvé využívalo hašovanie zreťazením. Predstavme si, že máme K spájaných zoznamov. Keď pridávame hranu (u, v), najprv si pomocou hašovacej funkcie vyrátame pozíciu spájaného zoznamu, do ktorého máme pridať túto hranu. Hašovacia funkcia môže vyzerať napríklad takto: $h(u, v) = (47u + 42v + 63(u^2 + v^2)) \bmod K$. Chceme totiž, aby funkcia rozmiestňovala hrany približne rovnomerne. Vkladanie je v konštantnom čase a vyhľadávanie priemerne v čase $O(1 + M/K)$. Druhý prístup využíva tabuľku veľkosti K a pre hranu (u, v) pomocou hašovacej funkcie[5] nájde pozíciu v tabuľke, kam sa má táto hrana vložiť. Ak je už táto pozícia obsadená, hľadá najbližšiu voľnú

[4] Alebo žeby „nie veľmi ste ma potešili“?

[5] môže byť rovnaká ako v prvom prípade

za ňou. V tomto prípade je hľadanie aj vkladanie rovnako rýchle – v priemernom prípade[6] to je $O(1/(1-\frac{M}{K}))$. V obidvoch riešeniach je preto vhodné zvoliť $K > 2M$. Problémom hašovania je, že v najhoršom prípade sa nám môže časová zložitosť veľmi zhoršiť. Takéto riešenie bolo odmenené 13 bodmi.

Všetky doteraz spomenuté riešenia vôbec nevyužívali fakt, že graf je planárny. Vieme, že $M \leq 3N-6$. Súčet stupňov vrcholov je $2M$, z čoho máme $2M \leq 6N-12$. Ukážeme, že v planárnom grafe je aspoň jeden vrchol so stupňom najviac 5. Sporom, nech majú všetky vrcholy stupeň aspoň 6, potom je súčet stupňov aspoň $6N$, čo je viac ako $6N-12$. Teda takýto graf nemôže byť planárny, čo je spor.

Nech v je taký vrchol grafu, ktorého stupeň je najviac 5. Keď ho z grafu odoberieme, dostaneme zase planárny graf. V tomto menšom planárnom grafe musí znova existovať nejaký vrchol so stupňom najviac 5. Takto vieme pokračovať, až kým sa nám neminú všetky vrcholy. Nech $v_1, \ldots, v_n$ sú postupne vrcholy, tak ako ich odoberáme z grafu. Zaveďme funkciu $c(v_i) = i$. Všimnime si situáciu, keď odoberáme z grafu vrchol v_i. Do podgrafu s vrcholmi $v_{i+1}, \ldots, v_n$ z neho vedie najviac 5 hrán. Preto spravíme nasledovnú vec: vytvoríme nový orientovaný graf. Ak v pôvodnom grafe bola hrana medzi v_i a v_j pričom $i < j$, v novom grafe bude orientovaná hrana z v_i do v_j. Z každého vrcholu preto vychádza najviac 5 hrán. Potom pri otázke na hranu medzi vrcholmi v_i a v_j si pozrieme všetkých susedov vrchola v_i. Týchto susedov je však najviac 5, preto otázku vieme vybaviť v konštantnom čase.

Teraz sa podrobnejšie zamyslime nad implementáciou. V prvej fáze použijeme front („fronta“ je po česky).[7] Najprv do frontu vložíme všetky vrcholy, ktoré majú stupeň najviac 5. Keď vrchol vyberáme z frontu, znižujeme stupeň susedným vrcholom (odoberáme hrany z grafu). Ak sa pri tom stane, že niektorý vrchol má stupeň menší alebo rovný 5, pridáme ho na koniec frontu. Zároveň si pre každý vrchol značíme, kedy sme ho odobrali z grafu. Potom si vyrobíme nový orientovaný graf. Nech pre hranu medzi vrcholmi u a v platí $c(u) < c(v)$ (teda vrchol u sme odobrali skôr). Potom si do spájaného zoznamu vo vrchole u vložíme ako suseda vrchol v. Každý vrchol pridáme do frontu raz a každú hranu spracujeme najviac dvakrát, preto je časová zložitosť predspracovania $O(N+M) = O(N)$.

Keď príde otázka na hranu medzi u a v, najprv zistíme, ktorý vrchol sme odobrali z grafu skôr (nech je to u). Potom sa pozrieme, či medzi susedmi vrchola u je vrchol v. Týchto susedov je najviac 5, takže táto fáza má časovú zložitosť $O(1)$.

Listing programu:

```
#include <iostream>
#include <vector>
#include <list>
#include <queue>
using namespace std;

int main() {
    int n, m; cin>>n>>m;
    list<int> h[100], vysl[100];
    for (int i=0; i<m; i++) {
        int a, b; cin>>a>>b;
        a--; b--;
        h[a].push_back(b); h[b].push_back(a);
    }

    vector<int> p(n);
    for (int i=0; i<n; i++) p[i]=h[i].size();
    queue<int> f;
```

[6] odhad zložitosti je nad rámec tohto príkladu

[7] Nie všetci vedúci KSP sa stotožňujú s týmto názorom. Vyberte si :-)

```
    int poc=0;
    vector<int> c(n, -1);
    //najprv dáme do frontu všetky vrcholy so stupňom menej ako 6
    for (int i=0; i<n; i++) if (p[i]<=5) {
        f.push(i);
        c[i]=poc++;
    }

    while (!f.empty()) {
        int u=f.front(); f.pop();
        //odoberieme z grafu všetky hranu idúce z tohto vrcholu
        for (list<int>::iterator it=h[u].begin(); it!=h[u].end(); it++)
            if (c[*it]==-1) if (--p[*it]<=5) {
                f.push(*it);
                c[*it]=poc++;
            }
    }

    //vyrobíme nový orientovaný graf
    for (int i=0; i<n; i++)
        for (list<int>::iterator it=h[i].begin(); it!=h[i].end(); it++)
            if (c[*it]>c[i]) vysl[i].push_back(*it);

    int k; cin>>k;
    for (int i=0; i<k; i++) {
        int a, b; cin>>a>>b;
        a--; b--;
        if (c[a]>c[b]) swap(a, b);

        bool ok=false;
        for (list<int>::iterator it=vysl[a].begin(); it!=vysl[a].end(); it++)
            if (*it==b) ok=true;
        if (ok) cout<<"Ano"<<endl; else cout<<"Nie"<<endl;
    }
}
```

opravoval Palo
(max. 15 bodov)

4. O skokanoch

Prišlo veľmi málo riešení tohoto príkladu, z toho len pár správnych. Bodovalo sa ľahko. Za korektné riešenie bežiace v čase $O(N \log X)$ sa dalo získať 15 bodov – N je počet skokov a X je najväčšia súradnica skokov na vstupe. Každé iné korektné riešenie mohlo dostať aspoň 10 bodov podľa svojej časovej a pamäťovej zložitosti. Nefunkčné riešenia dostali maximálne 6 bodov. Body boli strhnuté hlavne za chýbajúci program a zlý odhad zložitosti. Začnime priamo vzorovým riešením:

Pokúsime sa zjednodušiť popis skokanov. Ukážeme, že teritórium každého skokana vieme popísať jednoznačne pomocou najviac dvoch skokov. Potom stačí už iba porovnať, či obe teritóriá majú rovnaké popisy – t.j. či sú popísané rovnakým počtom skokov a či sú tieto skoky totožné. Konkrétne ukážeme, že teritórium každého skokana leží celé na priamke (popíšeme ho len jedným skokom) alebo ho celé vieme jednoznačne popísať dvomi skokmi, z ktorých jeden je dokonca rovnobežný s x-ovou osou. Kvôli ľahšiemu vyjadrovaniu ofarbime políčka patriace teritóriu na zeleno – budeme ich volať zelené políčka. Ostatné budú nezelené políčka. Rozoberme niekoľko prípadov podľa počtu skokov, ktorými je skokan určený:

- Skokan je určený len jedným skokom $v = [v_x, v_y]$. Potom jeho teritórium leží na jednej priamke a tvoria ho políčka tvaru $[kv_x, kv_y]$, pre $k \in Z$. Takéto teritórium vieme určiť niektorým z dvoch skokov: $v = [v_x, v_y]$ a $-v = [-v_x, -v_y]$. My si z nich vyberieme kvôli jednoznačnosti taký, ktorý skáče doprava (má kladnú x-ovú súradnicu). V prípade, že sú rovnobežné s y-ovou osou, si vyberieme taký, ktorý skáče hore (má kladnú y-ovou súradnicu).
- Skokan je určený dvomi skokmi $v = [v_x, v_y]$ a $s = [s_x, s_y]$, ktoré sú lineárne závislé – t.j. jeden je násobkom druhého. Aj v tomto prípade teritórium leží na priamke. Ktoré políčka do neho patria? Najprv predpokladajme, že všetky súradnice oboch skokov sú kladné. Pozrime sa na zelené políčko $p = [p_x, p_y]$ s najmenšou kladnou x-ovou súradnicou. Na políčko p vieme doskočiť, teda p vieme zapísať v tvare $p = av + bs$ t.j. $p_x = av_x + bs_x$, $p_y = av_y + bs_y$, kde $a, b \in Z$. Je jasné, že p_x je najmenšie kladné číslo s takou vlastnosťou. Takisto aj p_y je najmenšie číslo s takou vlastnosťou. Neskôr ukážeme, že $p_x = \text{nsd}(v_x, s_x)$ a $p_y = \text{nsd}(v_y, s_y)$, kde nsd značí najväčšieho spoločného deliteľa svojich argumentov.

  Takže sa vieme dostať na ľubovolné políčko tvaru $[kp_x, kp_y]$, kde $k \in Z$. Z políčka $[0, 0]$ sa pomocou nejakej sekvencie skokov dostaneme na políčko $[p_x, p_y]$, z neho tými istými skokmi na $[2p_x, 2p_y]$ a indukciou ďalej. Opačnými skokmi sa dostaneme na opačné políčka.

  Na ostatné políčka na priamke nevieme skočiť, lebo to by znamenalo, že vieme preskákať z nejakého políčka na políčko k nemu bližšie ako je vzdialenosť p od $[0, 0]$. Teda by sme aj z políčka $[0, 0]$ tými istými skokmi preskákali na políčko bližšie k $[0, 0]$ ako je p. Teda všetky políčka teritória sa potom dajú zapísať v tvare $[kp_x, kp_y]$. Teda teritórium má rovnaký tvar ako v predchádzajúcom prípade a skok, ktorý ho určuje si vyberieme pomocou rovnakých kritérií.

  Ostatné prípady, keď nejaká súradnica v alebo s je záporná, prevedieme na horeuvedený prípad. Najprv zistíme $p' = [p'_x, p'_y] = [\text{nsd}(|v_x|, |s_x|), \text{nsd}(|v_y|, |s_y|)]$. Potom podľa orientácie priamky určenej skokmi v a s zistíme korektné znamienka pre p'_x resp. p'_y – stačí zistiť cez ktoré kvadranty priamka prechádza.
- Skoky $v = [v_x, v_y]$ a $s = [s_x, s_y]$ nie sú lineárne závislé, teda jeden nie je násobkom druhého, t.j. $v_x/s_x \neq v_y/s_y$, t.j. $v_x s_y - v_y s_x \neq 0$. V tomto prípade skoky tvoria rovnobežníkovú sieť – jedným skokom skáčeme po priamke, druhým túto priamku posúvame.

  Najprv trochu neformálne: Na našu rovnobežníkovú sieť trochu zaškúlime a zistíme, že je vlastne tvorená aj inými rovnobežníkmi než tými zo vstupu. Konkrétne takými, že jedna z ich strán je vodorovná (druhá nemusí byť). Teda celé teritórium vieme určiť dvomi skokmi: $c = [c_x, 0]$ a $d = [d_x, d_y]$. Formálnejšie:

  Pozrime sa na ľubovoľnú vodorovnú priamku. Buď na nej nie je žiadne zelené políčko, alebo je ich tam nekonečno. Totiž ak je tam nejaké políčko $b = [b_x, b_y]$, tak ak s_y krát skočíme skokom v a v_y krát skokom $-s$, tak prídeme na políčko $b' = [b_x + v_x s_y - v_y s_x, b_y] \neq b$, ktoré leží na tej istej vodorovnej priamke. Z neho vieme skákať ďalej na ďalšie políčka rovnakým spôsobom. Nech najbližšie zelené políčko napravo od políčka b na tej istej vodorovnej priamke je $c = [b_x + c_x, b_y]$. Potom z rovnakých príčin ako v predchádzajúcom prípade sa všetky zelené políčka dajú písať v tvare $[b_x + kc_x, b_y]$, kde $k \in Z$.

  Pozrime sa na x-ovú os a k nej najbližšiu vodorovnú priamku q s aspoň jedným zeleným políčkom s kladnou y-ovou súradnicou d_y. Premyslite si, že y-ové súradnice všetkých neprázdnych vodorovných priamok musia mať tvar kd_y, kde $k \in Z$. Zoberme si nejaké zelené políčko $d = [d_x, d_y]$ na priamke q. Potom na vodorovnej priamke s y-ovou súradnicou kd_y leží zelené políčko $[kd_x, kd_y]$ a teda všetky zelené políčka na nej majú

tvar $[kd_x + lc_x, kd_y]$, kde $k, l \in Z$. Teda na každé zelené políčko vieme doskákať iba skokmi c a d. Takže už len treba vypočítať c_x, d_x, d_y.

Najprv vypočítame c_x. Vieme, že $c = av + bs$, pre nejaké $a, b \in Z$. Teda $c_x = av_x + bs_x$, $0 = av_y + bs_y$. Vieme, že s_y a v_y nemôžu byť súčasne 0 (skoky by boli lineárne závislé). Bez ujmy na všeobecnosti $s_y \neq 0$. Vyjadríme si b z druhej rovnice a dosadíme do prvej. Po úprave dostaneme $c_x = a(v_x s_y - v_y s_x)/s_y = ah/s_y$, kde $h = v_x s_y - v_y s_x$. Hľadáme také a, že c_x je najmenšie celé číslo spĺňajúce danú rovnosť. Upravme zlomok h/s_y na jeho základný tvar $\frac{u}{v}$, t.j. vykráťme ho $g = \mathrm{nsd}(h, s_y)$. Dostaneme, že $c_x = au/v$. Keďže v nedelí u, musí deliť a. Najmenšie také nenulové a deliteľné v je samozrejme priamo v, takže $a = v = s_y/g$. Teda $c_x = b_y(v_x s_y - v_y s_x)/\mathrm{nsd}(b_y, v_x s_y - v_y s_x)$.

Ako vypočítať d? Vieme, že $d_x = av_x + bs_x$, $d_y = av_y + bs_y$, pre nejaké $a, b \in Z$. Chceme najmenšie kladné d_y s danou vlastnosťou. Už som sľúbil, že neskôr ukážem, že $d_y = \mathrm{nsd}(v_y, s_y)$. Ďalej neskôr ukážem, ako nájsť nielen d_y, ale aj koeficienty $a, b \in Z$ také, že $d_y = av_y + bs_y$. Keď vieme a, b, tak ľahko spočítame aj d_x.

Ešte sme zabudli na jednoznačnosť. c_x a d_y sú určené jednoznačne. Ak skokan vie doskákať na $d = [d_x, d_y]$, tak vie doskákať aj na políčka tvaru $[d_x - kc_x, d_y]$, kde $k \in Z$. Z takých si vyberieme ten s najmenšou kladnou x-ovou súradnicou, teda celé teritórium je jednoznačne určené skokmi $c = [c_x, 0]$ a $d' = [d_x \bmod c_x, d_y]$.

- Skokan je určený aspoň tromi skokmi. Nech je skokan určený práve tromi skokmi a, b, c. Ak sú niektoré dva z nich lineárne závislé, tak z tých dvoch vyrobíme jeden skok určujúci rovnaké teritórium ako pôvodné dva. Dostaneme dva skoky, ktoré upravíme vyššie spomenutým spôsobom. Ak žiadne dva nie sú lineárne závisle, tak zo skokov a, b vyrobíme skoky t_1, t_2, kde t_1 je vodorovný skok. Potom zo skokov t_2, c vytvoríme skoky t_3, t_4, kde t_3 je vodorovný. t_3 a t_1 sú obidva vodorovné, vytvoríme z nich jeden vodorovný skok t_5. Teda skoky $t_5 = [t_{5x}, 0]$ a $t_4 = [t_{4x}, t_{4y}]$ určujú rovnaké teritórium ako skoky a, b, c. Kvôli jednoznačnosti ešte namiesto skoku t_4 zoberieme skok $t'_4 = [t_{4x} \bmod t_{5x}, t_{4y}]$. Čo ak sú na vstupe viac ako tri skoky? Načítame prvé tri a spracujeme ich – t.j. dostaneme k nim najviac dva ekvivalentné. Načítame ďalší skok, spracujeme a znovu máme najviac dva k nim ekvivalentné, atď. Takto pre každé teritórium dostaneme jednoznačne najviac dva skoky, ktoré ho celé určujú a pre porovnanie teritórií stačí porovnať tieto skoky.

Ešte treba ukázať, ako pre dané celé nezáporné čísla x, y nájsť najmenšie také s, že s sa dá písať v tvare $s = ax + by$, $a, b \in Z$. Ukážeme, že $s = \mathrm{nsd}(x, y)$.

Najprv si povedzme niečo o tom, ako spočítať najväčší spoločný deliteľ x a y. Bez ujmy na všeobecnosti nech $x \geq y$. Ak $y = 0$, tak zjavne $\mathrm{nsd}(x, 0) = x$. Inak x vieme písať v tvare $x = ty + r$, kde $r < y$ (r je zvyšok x po delení y). Na domácu úlohu dokážte, že v tomto prípade $\mathrm{nsd}(x, y) = \mathrm{nsd}(y, r)$. Príklad práce algoritmu: $\mathrm{nsd}(360, 294) = \mathrm{nsd}(294, 360 \bmod 294 = 66) = \mathrm{nsd}(66, 30) = \mathrm{nsd}(30, 6) = \mathrm{nsd}(6, 0) = 6$. Algoritmus vždy skončí, lebo sa volá vždy s menšími parametrami. Tento algoritmus je jedným z najstarších známych algoritmov a volá sa po svojom objaviteľovi Euklidov algoritmus. Všimnime si, že vždy sa nám aspoň jeden z parametrov zmenší aspoň 2-krát, teda celková časová zložitosť Euklidovho algoritmu je $O(\log x)$. Pamäťová zložitosť je tiež kvôli rekurzii $O(\log x)$ – pre každé volanie si minimálne musíme na zásobníku pamätať jeho parametre.

Teraz ukážeme, že $\mathrm{nsd}(x, y)$ sa dá písať v tvare $\mathrm{nsd}(x, y) = ax + by$. Dokážeme to indukciou od počtu krokov, ktoré potrebuje Euklidov algoritmus na spočítanie $nsd(x, y)$. Rozšírme Euklidov algoritmus tak, že nám vráti nielen najväčší spoločný deliteľ, ale aj hodnoty a, b tak, že $\mathrm{nsd}(x, y) = ax + by$. Ak $y = 0$, tak $\mathrm{nsd}(x, 0) = x = 1x + 0y$, teda $a = 1$ a $b = 0$. Ak $y \neq 0$, teda $x = ty + r$, tak sa rekurzívne zavoláme na $\mathrm{nsd}(y, r)$ a dostaneme hodnoty c, d tak, že $\mathrm{nsd}(y, r) = cy + dr$. Potom ale $\mathrm{nsd}(x, y) = \mathrm{nsd}(y, r) = cy + dr = cy + d(x - ty) = dx + (c - dt)y$, teda $a = d$ a $b = c - dt$; $t = x \operatorname{div} y$, kde div je celočíselné delenie.

Ešte musíme ukázať, že nsd(x, y) je najmenšie také kladné s, že $s = ax + by$. Ľubovolný spoločný deliteľ a a b delí aj výraz $ax + by$, teda špeciálne aj nsd(x, y) delí $s = ax + by$. Teda všetky prípustné čísla s sú násobkami nsd(x, y) a teda nsd(x, y) musí byť z nich najmenšie kladné.

Celková časová zložitosť je teda $O(N \log X)$, kde N je počet skokov na vstupe a X je najväčšie číslo na vstupe (pre každý skok voláme konštantne veľa krát funkciu *nsd*). Pamäťová zložitosť je potom $O(\log X)$.

Listing programu:

```
#include <cstdio>
#include <cmath>
#include <vector>
#include <utility>
#include <algorithm>
#define PII pair<int,int>
#define X first
#define Y second

using namespace std;

int modulo(int a, int b) { if (a%b < 0) return b+a%b; return a%b; }

int gcd(int a, int b, int &k, int &l) {
    if (b==0) { k=1; l=0; return a; }
    if (a<b) return gcd(b,a,l,k);
    int ret=gcd(b,a%b,k,l);
    int tmp=k;
    k=l;    l=tmp-(a/b)*l;
    return ret;
}

int transform(PII &v1, PII &v2) {
    int tmp, tmp2;
    if (v1.X*v2.Y==v1.Y*v2.X) { //su linearne zavisle ?
        PII ret=PII(gcd(abs(v1.X), abs(v2.X),tmp,tmp2),
                    gcd(abs(v1.Y), abs(v2.Y), tmp, tmp2));
        if (v1.X*v1.Y<0) ret.Y=-ret.Y;
        v1=ret;
        return 1;
    } else {
        if (v2.Y==0) swap(v1,v2);
        int cx=abs(v1.X*v2.Y-v1.Y*v2.X)
               /gcd(abs(v1.X*v2.Y-v1.Y*v2.X),abs(v2.Y),tmp,tmp2);
        int dy=gcd(abs(v1.Y), abs(v2.Y), tmp, tmp2);
        int dx=modulo((tmp*v1.X+tmp2*v2.X),cx);
        v1=PII(cx,0);
        v2=PII(dx,dy);
        return 2;
    }
}

int spracuj(PII &v1, PII &v2) {
    PII v3;
    int n,poc;

    scanf("%d %d %d ", &n, &v1.X, &v1.Y);
    if (v1.X<0) { v1.X=-v1.X; v1.Y=-v1.Y; }
    poc=1;
```

```
    for (int i=1;i<n;i++) {
        scanf("%d %d", &v3.X, &v3.Y);
        if (poc==1) {
            poc=transform(v1,v3);
            v2=v3;
        } else {
            transform(v3,v2); transform(v1,v3);
            v2.X=modulo(v2.X,v1.X);
        }
    }
    return poc;
}

int main() {
    int poc1,poc2;
    PII v1,v2,s1,s2;
    poc1=spracuj(v1,v2); poc2=spracuj(s1,s2);
    if (poc1==poc2 && v1==s1 && (poc1==1 || v2==s2)) { printf("ANO\n"); return 0; }
    printf("NIE\n");
    return 0;
}
```

opravoval MišoF.
(max. 15 bodov)

5. Optimálna cestná sieť

Prvou možnosťou, ako sa popasovať s touto úlohou, bolo zjednodušiť si ju. Nebudeme stavať žiadne križovatky, len pospájame niektoré dvojice miest tak, aby vznikla súvislá cestná sieť. Takto sme dostali problém, ktorému sa v odbornej literatúre nadáva *najlacnejšia Euklidovská kostra*.

Existujú veľmi efektívne algoritmy, ktoré túto úlohu riešia. My však môžeme ísť ešte o krok ďalej – zabudneme na to, že mestá sú body v rovine. Situáciu si popíšeme jednoducho grafom. Vrcholy budú mestá, medzi každými dvoma vrcholmi dáme hranu, ktorej dĺžka je rovná priamej vzdialenosti zodpovedajúcich miest.

V tomto grafe chceme nájsť *najlacnejšiu kostru*, teda takú množinu hrán, po ktorých sa dá prejsť medzi ľubovoľnými dvoma vrcholmi a ktorá má spomedzi všetkých takýchto množín najmenší súčet dĺžok hrán.

Nie je ťažké prísť na to, že najlacnejšia kostra bude určite *strom*, teda nebude obsahovať žiadne kružnice. Z kružnice by sme totiž mohli ľubovoľnú jednu hranu bez porušenia súvislosti vyhodiť.

V stručnosti popíšeme algoritmus pánov Jarníka a Prima, ktorý hovorí, ako nájsť v grafe s N vrcholmi najlacnejšiu kostru v čase $O(N^2)$.

Vrcholy budeme mať rozdelené na dve množiny: už spojené vrcholy S a ostatné, ešte nespojené vrcholy O. Pre každý vrchol $v \in O$ si budeme pamätať k nemu najbližší vrchol z S (označíme ho $n(v)$) a hodnotu $d(v)$, čo je vzdialenosť z v do $n(v)$.

Na začiatku dáme do S jeden ľubovoľný začiatočný vrchol z. Pre každý iný vrchol v bude $n(v) = z$ a $d(v)$ vzdialenosť z v do z.

V každom kroku algoritmu vyberieme z O ten vrchol x, ktorý má najmenšiu vzdialenosť $d(x)$, a zodpovedajúcou najkratšou hranou ho pripojíme k S. Čo sa teraz stalo? Niektorým vrcholom $v \in O$ sa mohol zmeniť najbližší vrchol z S (teda $n(v)$) na x. Prejdeme preto všetky vrcholy z O a pre každý $v \in O$ pozrieme, či je vzdialenosť z x do v menšia ako $d(v)$, a ak áno, príslušný záznam opravíme.

Každý krok teda vieme spraviť v čase $O(N)$. A keďže v každom kroku pripojíme jeden vrchol, krokov bude $N-1$. Preto celková časová zložitosť je $O(N^2)$.

Správnosť algoritmu vyplýva z nasledujúceho tvrdenia: *Rozdeľme vrcholy grafu na dve množiny X a Y. Potom existuje najlacnejšia kostra, ktorá obsahuje najlacnejšiu hranu medzi X a Y.*

Dôkaz: Majme najlacnejšiu kostru, ktorá našu hranu h neobsahuje. Pridajme do nej túto hranu, tým v nej vznikne (práve jedna) kružnica. Teraz sa po tej kružnici prejdeme, pričom začneme prejdením po práve pridanej hrane z X do Y. V tomto okamihu sme v množine Y, niekedy sa ale musíme vrátiť do X. Naša kružnica teda určite obsahuje aspoň jednu ďalšiu hranu h' vedúcu medzi X a Y. Ak teraz hranu h' vyhodíme, dostaneme opäť strom. Navyše vieme, že h' je aspoň tak dlhá ako h (predpokladali sme, že h je najkratšia). Preto nový strom je aspoň taký dobrý ako pôvodný. Ale už pôvodný bol optimálny, a preto jediná možnosť je tá, že nový je tiež optimálny a h a h' sú rovnako dlhé.

(Ukázali sme teda nielen to, že vieme nájsť najlacnejšiu kostru, ktorá hranu h obsahuje, ale navyše aj to, že ak je h jediná najlacnejšia hrana medzi X a Y, tak v najlacnejšej kostre byť musí.)

Domáca úloha. Krajinu popíšeme grafom, v grafe nájdeme najlacnejšiu kostru a podľa nej postavíme v krajine priame cesty zodpovedajúce hranám kostry. Môžu sa niektoré dve takto postavené cesty križovať?

Ako už ale ukazoval príklad s rovnostranným trojuholníkom v zadaní, najlacnejšia kostra nemusí byť najlepším riešením tejto úlohy. Úloha zo zadania sa volá *Euklidovský Steinerovský strom*[8] a na rozdiel od najlacnejšej kostry je sakramentsky ťažká.

To, čo je na úlohe ťažké, je povedať, kde budú križovatky, teda presnejšie, s ktorými vrcholmi budú spojené. Keď už si toto vycucáme z prsta, zvyšok sa už nejako dá doraziť. Jedno možné riešenie, ktoré sa ľahko programuje, je jednoduché iteračné. Postupne každú križovatku skúsime posunúť o malý krôčik niekoľkými smermi (napr. hore, dole, doprava a doľava) a zakaždým sa pozrieme, ako sa nám zmenil súčet dĺžok hrán, ktoré do nej vedú. Ak sa zmenšil, zmenu necháme, ak nie, vrátime križovatku späť. Toto robíme dokola, až kým sa nám všetky križovatky neprestanú hýbať.

(Tento krok sa dá spraviť aj efektívnejšie. Dá sa napríklad dokázať, že v optimálnom riešení sa v každej nami pridanej križovatke budú stretať práve tri cesty, a uhol medzi každými dvoma bude 120 stupňov.)

Ako ale povedať, kam dať križovatky? Tu bolo opäť možných viacero prístupov. Nami odporúčaný je vyskúšať si daný vstup na obrazovke zobraziť a križovatky „od oka“ ručne naklikať `:-)`. Prípadne sa dá vyskúšať nejaké naprogramovateľné heuristiky, ako napríklad:

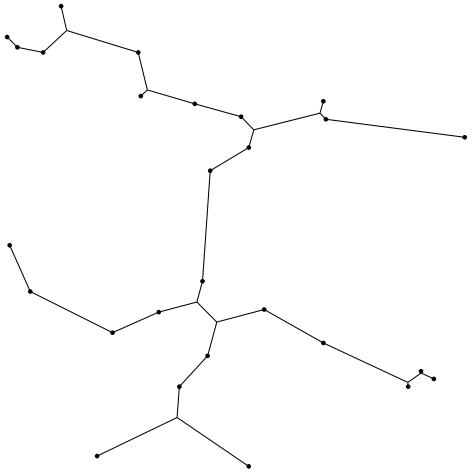
- Usporiadame body podľa vzdialenosti od ťažiska.
- Vložíme križovatku, spojíme ju s prvými tromi bodmi a nájdeme jej optimálnu polohu. Takto dostaneme *aktuálny strom*.
- Postupne spracúvame ostatné body. Spracovanie jedného bodu a vyzerá nasledovne: Postupne každú hranu bc v aktuálnom strome skúsime nahradiť križovatkou spojenou s bodmi a, b a c, v každom z takto získaných stromov nájdeme optimálne polohy križovatiek a vyberieme z nich najlepší. To bude náš nový, o vrchol väčší aktuálny strom.

Existujú ešte oveľa prefíkanejšie spôsoby, ako hľadať Steinerovské stromy. Záujemcov o ne odkážeme na stránku `http://www.diku.dk/geosteiner/`, a obzvlášť na články uvedené na nej úplne dole.

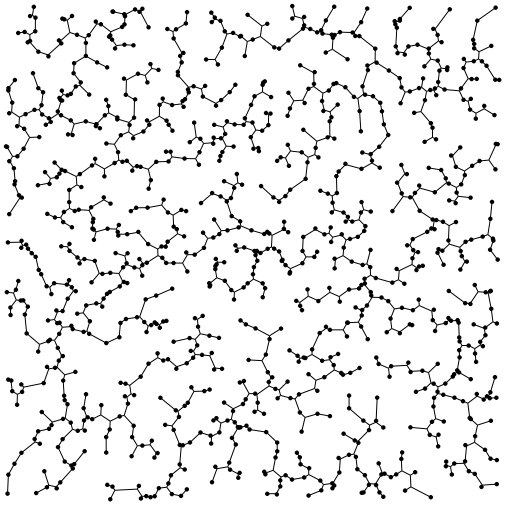
[8] Prvé prídavné meno podľa uja Euklida hovorí, že sa hráme v normálnej rovine, druhé je podľa uja Jacoba Steinera, ktorý túto konkrétnu úlohu medzi prvými riešil.

Nuž, a keďže jeden obrázok je lepší ako tisíc slov, ukončíme toto vzorové riešenie pohľadom na najlepšie nájdené riešenia pre niektoré zadané vstupy. Tuto sú:

vstup 4



vstup 7



A ešte jedno P.S.: Prekvapivo sa ukazuje, že Steinerovské stromy súvisia s mydlovými bublinami. Pozrite napríklad fotku `http://www.scottaaronson.com/soapbubble.jpg`.

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Perešíni Peter	Gym. Tajovského B. Bystrica	4	15	15	10	11	15	66
2	Herman Peter	Gym. Jura Hronca BA	3	7	15	10	5	10	47
3	Brezáni Samuel	Gym. Rajec	3	9	14	11	9	3	46
4	Okruhlica Adam	Gym. Jura Hronca BA	3	9	11	13	1	7	41
5	Tříska Martin	Gym. P. de Coubertina Piešťany	3	15	15	10			40
6	Rampášek Ladislav	Gym. Jura Hronca BA	3		15	11		12	38
7	Mikuláš Ondrej	Gym. Haličská Lučenec	3	11		11		9	31
8	Imriška Jakub	Gym. Jura Hronca BA	4		15	15			30
9	Nagy Anton	Gym. maďarské BA	3	3	10	7	4	3	27
10	Petrucha Michal	Gym. Metodova BA	1		14	10			24
11	Jerguš Ján	Gym. Alejová Košice	3	7		10			17
12	Sucha Martin	Gym. Jura Hronca BA	2		6	10			16
13	Tomcsányi György	Gym. H. Selyeho Komárno	3	7		7			14
14	Bundala Daniel	Gym. Jura Hronca BA	4					13	13
14	Pančík Andrej	Gym. Tajovského B. Bystrica	3			13			13
16	Danko Juraj	Gym. Jura Hronca BA	3			9			9
17	Bílka Ondřej	Gym. Zlín	4			7			7
18	Svonava Daniel	Gym. Jura Hronca BA	4					5	5
19	Korcsok Peter	Gym. Mládežnícka Šahy	2	2					2