Korešpondenčný seminár z programovania
XXVI. ročník, 2008/09
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: MICROSTEP-MIS spol. s r.o.
Whitestein Technologies spol. s r.o.

Vzorové riešenia 1. kola zimnej časti

Milé naše riešiteľky, milí naši riešitelia,
práve sa Vám dostávajú do rúk vzorové riešenia prvého kola nášho skvelého seminára na neuveriteľných ôsmych listoch. To však ale nevadí, však každý večer je čoraz dlhší a dlhší... Užite si teda naše vzoráky podľa Vášho najlepšieho vedomia a svedomia a nezabudnite poslať riešenia ďalšej série.

Komu sa nelení, tomu treba roboty naložiť.

KSPáci

1. Základy požierania

opravovala Dominika
(max. 10 bodov)

Tento príklad bol vcelku jednoduchý, a preto mal aj veľa riešiteľov. 97% z vás malo správnu, konštantnú časovú aj pamäťovú zložitosť $O(1)$. Nie všetci si však dali tú námahu a napísali aj poriadny dôkaz. Pre niektorých zdôrazňujem, že tvrdenie „pokusom som zistil...“ nie je dôveryhodné riešenie. Za chýbajúci dôkaz som strhávala 1–4 body. Za chýbajúcu zložitosť -1 bod.

Pustíme sa ale do vzorového riešenia. Zostaňme pritom pri názornosti s čokoládou: celú sálu vidíme ako čokoládu s rozmermi $m \cdot n$, pričom ju chceme rozlámať na jednotlivé kúsky. Na začiatku máme jeden veľký kus čokolády. Čo sa stane po jednom lome (reze)? Z pôvodnej 1 časti čokolády získame 2 časti. Potom, ak máme k častí čokolády, zoberieme jednu z nich a po reze budeme mať $k + 1$ častí. Keďže potrebujeme $n \cdot m$ kusov čokolády a začíname s 1 kusom, budeme potrebovať $n \cdot m - 1$ rezov.

Listing programu:

```
var n, m : longint;

begin
  readln (n, m);
  writeln (n*m - 1);
end.
```

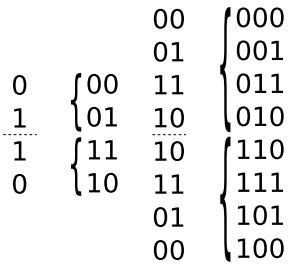
2. Zúrivé Godzillie nájazdy

opravovali Katka a Samo
(max. 10 bodov)

Poďme sa porozprávať o našich obľúbených ničivých Godzilách. Aby sme uplatnili správny, vedecký prístup, musíme si definovať, čo taká Godzilla je. Samozrejme je to obrovské a ničivé zvieratko, ktoré má N atribútov. Každý atribút má len dve možné hodnoty: môžeme si ich označiť ako 0 a 1. Tiež vieme, že pre každú možnú kombináciu atribútov existuje Godzilla. A preto môžeme uvažovať o každej Godzille ako o N-atribútovej postupnosti 0 a 1.

Už síce vieme, čo je Godzilla, ale nás zaujímať, v akom poradí ich môžeme vidieť v japonských filmoch. Dve po sebe idúce Godzilly sa líšia iba v hodnote jedného atribútu. Konkrétne na jednej pozícii N-atribútovej postupnosti má jedna z nich hodnotu 1 a druhá 0. Ale na všetkých ostatných pozíciach sa im hodnoty atribútov rovnajú. Takže zadanie by sme mohli preformulovať nasledovne: hľadáme také N-atribútové postupnosti 0 a 1, ktoré sa líšia práve na jednej pozícii.

Už zostáva pred nami iba otázka ako ich čo najefektívnejšie nájsť. Pozrime sa, či by sme vedeli zostrojiť z $(n-1)$ atribútovej postupnosti n-atribútovú. Predstavme si, že máme $(n-1)$-atribútovú postupnosť 0 a 1, ktorej členy sa líšia práve na jednej pozícii. Vypíšme túto postupnosť dvakrát za seba, ale druhýkrát ju napíšme odzadu. Postupnosť potom vyzerá, ako keby sme ju zobrazili v zrkadle, pozri obrázok. A teraz pridajme pred všetky členy pôvodnej postupnosti 0 a pred všetky členy opačnej postupnosti 1. Všimnime si, že sme dostali hľadanú N-atribútovú postupnosť. Na prechode medzi pôvodnou a opačnou postupnosťou sa Godzilly líšia len v prvom atribúte, lebo posledný prvok normálnej postupnosti bol totožný s prvým prvkom opačnej. Všade inde sa zrejme po sebe idúce prvky líšia práve v jednom atribúte, keďže to vyplýva z predpokladu. Takto tvorená postupnosť sa nazýva *zrkadlový binárny kód* alebo tiež *Grayov kód*.



Vyhovujúcu postupnosť sme už objavili, zostáva už len otázka jej optimálnej konštrukcie. Prvý spôsob, ktorý nám napadne, je napísať rekurzívne riešenie, ktoré presne kopíruje algoritmus, ktorým sme postupnosť definovali. Jednu možnú implementáciu v Pascale môžete vidieť

Listing programu:

```
const
    maxN = 16;     {maximalny pocet atributov}
    maxG = 66000; {maximalny pocet godzill}

var
    N          : integer;                              {pocet atributov}
    godzill    : integer;                             {pocet godzill}
    postupnost : array [1..maxG, 1..maxN] of integer; {postupnost godzill}

procedure kopiruj_godzillu(odkial, kam, dlzka : integer);
var
    i : integer;
begin
    for i := 1 to dlzka do
        postupnost[kam, i] := postupnost[odkial, i];
end;

function generuj(k : integer) : integer;
var
    i, j  : integer;
    pocet : integer;
begin
    {Pre k = 0 vygeneruje funkcia postupnost s jedinou bezatributovou godzillou}
    if k = 0 then
```

```
        generuj := 1
    else
    begin

        {vygenerujeme riesenie pre k - 1 a pocet jeho prvkov si zapamatame}
        pocet := generuj(k - 1);

        {
          Vlozime do pola zrkadlovo symetricku kopiu postupnosti. K prvej polovici
          pridame na koniec hodnotu atributu 0, k druhej polovici postupnosti
          pridame na koniec hodnotu atributu 1.
      }
        i := pocet;
        j := pocet + 1;
        while i > 0 do
        begin
            kopiruj_godzillu(i, j, k - 1);
            postupnost[i, k] := 0;
            postupnost[j, k] := 1;
            i := i - 1;
            j := j + 1;
        end;

        {vratime pocet vygenerovanych godzill}
        generuj := pocet * 2;
    end;
end;

procedure vypis(godzill, atributov : integer);
var
    i, j : integer;
begin
    for i := 1 to godzill do
    begin
        {vypisuje jednotlive cifry reprezentacie godzilly - vypisujeme ich
      zostupne, rozmyslite si preco}
        for j := atributov downto 1 do
            write(postupnost[i, j]);

        {prejde na novy riadok}
        writeln();
    end;
end;

begin
    readln(N);              {nacitanie vstupu}
    godzill := generuj(N); {vygenerovanie postupnosti}
    vypis(godzill, N);      {vypisanie vysledku}
end.
```

Toto riešenie má časovú aj pamäťovú zložitosť $O(N \cdot 2^N)$. Časová zložitosť sa už zrejme zlepšiť nedá, keďže program nemôže mať menšiu časovú zložitosť, ako je časová zložitosť vypísania výstupu. To, čo by sme chceli zlepšiť, je pamäťová zložitosť.

Na zlepšenie pamäťovej zložitosti sa dá použiť rekurzívny prístup, ale to to riešenie je princípom veľmi podobné predchádzajúcemu riešeniu a preto uvedieme riešenie z iného súdka.

Na ceste za zlepšením pamäťovej zložitosti sa budeme musieť oboznámiť s dvoma bitovými operáciami: `xor` a bitový posun. Xor dvoch čísel vypočítame tak, že čísla prevedieme do dvojkovej sústavy, napíšeme zodpovedajúce cifry pod seba a vo výsledku napíšeme na danú pozíciu nulu práve vtedy, keď boli cifry rovnaké a jednotku práve vtedy, keď boli cifry rôzny. Bitový posun zas funguje, ako už názov napovedá, tak, že posunieme dvojkový zápis čísla o niekoľko cifier vľavo, alebo vpravo. Bitový posun vľavo je ekvivalentný vynásobeniu čísla mocninou dvojky, bitový posun vpravo zodpovedá vydeleniu čísla mocninou dvojky. Pokúsime sa dokázať, že dvojkový zápis čísla, ktoré dostaneme xorom i s i binárne posunutým o jedna vpravo, je presne i-ty prvok Grayovej postupnosti.

Tvrdenie: Dvojkový zápis N-bitového čísla, ktoré dostaneme xorom i s i binárne posunutým o jedna vpravo, je presne i-ty prvok Grayovej postupnosti.

Dôkaz: Dokážeme to indukciou cez číslo N. Zrejme pre 1-bitové čísla tvrdenie platí[1]. Predpokladajme, že tvrdenie platí pre $(N-1)$-bitové čísla. Z toho ukážeme, že tvrdenie platí aj pre N-bitové čísla.

Napíšme si binárne všetky čísla od 0 po $2^N - 1$. Prvá z polovica z týchto čísiel začína nulou, druhá začína jednotkou a až na najvyšší bit sú všetky čísla zhodné. Pre prvú polovicu čísiel tvrdenie zjavne platí (lebo platí indukčný predpoklad[2]) a teda ak prvú polovicu N-bitovej Grayovej postupnosti získame správym shiftnutím a zoxorovaním. Ako je to s druhou polovicou?

Druhú polovicu Grayovej postupnosti sme získali (spôsobom z prvej časti vzoráku) tak, že sme napísali prvú polovicu opačne (zrkadlová postupnosť) a potom sme na začiatok pridali jednotku. Pozrime sa najskôr na zrkadlovú postupnosť (ešte pred pridaní toho jednotkového bitu na začiatok!). Ako sa líši od pôvodnej? Líši sa iba v prvom bite a to tak, že prvý bit je opačný (rozmyslite si prečo). To znamená, že i-ty prvok v prvej polovici Grayovej postupnosti sa líši od i-teho prvku v druhej polovici Grayovej postupnosti iba prvými dvoma bitmi. Ten v prvej polovici má prvý bit nula, druhý ľubovoľný a ten druhý má oba dané bity opačné. Teraz si ukážeme, že táto vlastnosť platí aj pre našu postupnosť.

Nech A je i-te číslo z prvej polovice našich čísiel (sú to binárne vypísané čísla od 0 po $2^N - 1$, nie Grayove čísla), a nech B je i-te číslo z druhej polovice. A a B sa líšia len v jednom (najvyššom) bite. Teda A vieme zapísať ako $0a_1a_2a_3\ldots a_{N-1}$ a B ako $1a_1a_2\ldots a_{N-1}$ kde $a_1, a_2, \ldots, a_{N-1}$ sú nejaké bity. Ako to vyzerá po shiftnutí a xorovaní?

$$A \text{ xor } (A \text{ shr } 1) = 0a_1a_2\ldots a_{N-1} \text{ xor } 00a_1\ldots a_{N-2} = 0a_1b_1\ldots b_{N-2}$$
$$B \text{ xor } (B \text{ shr } 1) = 1a_1a_2\ldots a_{N-1} \text{ xor } 01a_1\ldots a_{N-2} = 1a'_1b_1\ldots b_{N-2}$$

kde a'_1 je negácia bitu a_1 (je tam negácia, lebo 1 xor x je vždy negácia x) a $b_i = a_i$ xor a_{i+1} sú ostatné bity (pre nás sú ich hodnoty nepodstatné, lebo v oboch výsledkoch sú rovnaké). Všimnite si, že tu sa nám výsledok líši iba v prvých dvoch bitoch a to dokonca úplne rovnako ako v Grayovej postupnosti (prvé dva bity sú v B opačné). Čo sme tým dosiahli? Týmto sme ukázali, že ak tvrdenie platí pre $N-1$, tak platí pre N. Ale keďže tvrdenie platí pre $N = 1$, tak musí platiť pre ľubovoľné N.[3]

Nami nájdené riešenie bude teda spočívať z jedného cyklu v ktorom postupne vypíšeme prvých 2^N Grayových čísel. Pamäťová zložitosť bude $O(N)$, časová bude $O(N2^N)$. Jednoduchú implementáciu v C++ môžete vidieť tu:

[1] Vyskúšajte si všetky (slovom dve) možnosti.

[2] Teda že tvrdenie platí pre $N-1$

[3] Kto neverí, nech sa tu na mieste zastaví a poriadne si to rozmyslí.

Listing programu:

```
#include <iostream>

using namespace std;

void vypis_binarne(int godzilla, int atributov)
{
    for (int i = atributov - 1; i >= 0; i--)
        cout << ((godzilla >> i) & 1);
    cout << endl;

}

int main()
{
    int N;
    cin >> N;
    int godzill = 1 << N;
    for (int i = 0; i < godzill; i++)
        vypis_binarne(i ^ (i >> 1), N);
    return 0;
}
```

Poznámka: Prečo je pamäťová zložitosť $O(N)$? Veď používame iba zopár celočíselných premenných. V tom je ten fígeľ. Ak máme 32 bitovú premennú, môžeme si v nej zapamätať až 32 bitov[4] ak máme 32 bitový integer (pre pascalistov: integer v pascale má 16 bitov). Keby sme však chceli generovať viacbitový Grayov kód, tak by sme museli použiť pole, alebo aspoň také celočíselné premenné, ktoré majú aspoň N bitov a teda to znamená, že majú aspoň $N/8$ bajtov (lebo 1 bajt je 8 bitov).

3. Zdravotné problémy

opravoval András
(max. 10 bodov)

Vzorové riešenie je inšpirované riešením Jána Hozzu.

Predstavte si, že medzi mrakodrapmi nie sú žiadne medzery a teda Godzilla môže zobrať veľký kýbeľ vody a vyliať ho na mrakodrap, kde stojí. Táto voda sa začne roztekať (rýchlosťou jeden mrakodrap za časovú jednotku) a bude sa roztekať len na nižšie mrakodrapy[5]. Ak si zoberieme čas, kedy sa dostane voda na cieľový mrakodrap, tak práve toľko to bude trvať Godzille na cieľové políčko. Túto vodu si trošku upravíme, aby nechodila na mokré políčka a aby pre každý mrakodrap napísala na neho, koľko jej to trvalo, kým sa dostala na daný mrakodrap. Všimnite si, že na každé políčko sa dostane voda po nejakej najkratšej ceste. Teraz túto vodu treba naprogramovať.

Ako? Zoberme si nejaký mrakodrap, napríklad ten na pozícii[6] $[x, y]$. Nech na neho príde voda v čase i. Nech $[x', y']$ je susedný nižší mrakodrap, na ktorom v čase i ešte nebola voda. Potom na mrakodrap $[x', y']$ príde voda v čase $i+1$ (Napríklad sa prileje z políčka $[x, y]$). Ak teda pozrieme všetkých susedov $[x, y]$, tak vieme zistiť, na ktoré políčka sa z neho dostane voda v čase $i + 1$.

[4] Toto platí ak je premenná bez znamienka, vo vzoráku je zo znamienkom, a teda si môžeme zapamätať iba 31 bitov.

[5] Voda väčšinou tečie smerom dole

[6] Odteraz budem zamieňať pozíciu mrakodrapu a mrakodrap, aby sa lepšie vyjadrovalo

Budeme brať mrakodrapy postupne podľa toho, kedy sa dostane na ne voda. Teda najskôr tie, na ktoré sa dostane voda v čase 1, potom v čase 2, ... Vždy, keď zoberieme mrakodrapy, na ktoré sa dostane voda v čase i, tak z nich vieme vypočítať mrakodrapy, na ktoré sa dostane voda v čase $i+1$. Keď narazíme na cieľový vrchol (v čase K), tak vieme, že cieľový vrchol je vzdialený K skokov. Ostáva nám toto iba implementovať.

Použijeme dátovú štruktúru s názvom front, ktorý sa často nesprávne nazýva frontou, ako keby bol ženského rodu. Front funguje ako rada v jedálni. Má dve operácie. Jedna je vlož, a tá spôsobí, že front si zapamätá daný prvok. Alebo môžeme odtiaľ vybrať prvok (front vyhodí zo seba prvok, ktorý tam prišiel najskôr). Implementujeme to v obyčajnom poli. Budeme si pamätať dve čísla, jedno bude označovať, kde v poli začína front, a druhé kde končí. Ak príde nový prvok, tak zvýšime koniec o jedna a uložíme prvok na miesto v poli, kam ukazuje koniec. Ak chceme vybrať z frontu, tak zoberieme prvok na ktorý ukazuje začiatok a začiatok si zvýšime o jedna (a teda odteraz začiatok ukazuje na ďalší prvok). Ak sa stane, že začiatok frontu, alebo koniec frontu príde na koniec poľa (dosiahnu najväčší možný index), tak sa nastavia na začiatok poľa (najnižší index poľa). Ak zvolíme naše pole dostatočne veľké, tak sa nám nestane, že začiatok a koniec sa stretnú.

Ako vyzerá celé riešenie? Zoberieme počiatočný mrakodrap a vložíme ho do frontu, zaznačíme si, že už je vo fronte a že jeho vzdialenosť je nula. Potom vždy zoberieme prvý mrakodrap z frontu (nech má vzdialenosť i), zoberieme jeho nižších susedov, a ak nie sú a neboli vo fronte, tak ich vložíme do frontu, zaznačíme že sú vo tam a nastavíme ich vzdialenosť na $i+1$ (je to určite vzdialenosť najkratšej cesty od začiatku k nim). Ak raz vyberieme z frontu koncový mrakodrap, sme šťastní, lebo pre neho máme poznačenú minimálnu vzdialenosť. Tomuto algoritmu sa hovorí **prehľadávanie do šírky**. Na zdôvodnenie názvu si je dobré predstaviť, ako vlastne tento algoritmus funguje: najprv prehľadá všetky mrakodrapy, ktoré sú vzdialené 1, potom prehľadá všetky, ktoré sú vzdialené 2, ..., a takto postupne rozširuje plochu prehľadaných mrakodrapov.

Časová aj pamäťová zložitosť algoritmu je $O(NM)$ (pre každý mrakodrap spravíme konštantný počet operácii).

Listing programu:

```
program uloha3;
const maxN=100;
      maxM=100;
      smery:array[1..4,1..2] of integer=((+1,0),(-1,0),(0,+1),(0,-1));
type item=record
       p,q:integer;
     end;
var
  n,m,sx,sy,cx,cy,x,y : integer;
  i,j,nx,ny:integer; {pomocne premenne}
  fb,fe:integer; {zaciatok a koniec fronty}
  front : array[0..maxN*MaxM] of item;
  bol : array[1..maxM,1..MaxN] of integer;
  vyska : array[1..maxM,1..maxN] of integer;

procedure put(p,q:integer);
begin
 front[fe].p:=p;
 front[fe].q:=q;
 fe:=(fe+1)mod (maxM*maxN);
end;

procedure get(var p,q:integer);
begin
  p:=front[fb].p;
```

```
  q:=front[fb].q;
  fb:=(fb+1) mod (maxM*maxN);
end;

begin
  readln(n,m);
  readln(sy,sx);
  readln(cy,cx);
  for i := 1 to n do
    begin
      for j := 1 to m do begin
        read(vyska[j,i]);
        bol[j,i]:=-1;
      end;
      readln;
    end;
  fb := 0; {nastavime zaciatok a koniec fronty na 0}
  fe := 0;
  put(sx,sy);
  bol[sx,sy]:=0;
  repeat
    get(x,y);
    if (x=cx) and (y=cy) then {ak z frontu vyberiem ciel, mozem skoncit}
      break;
    for i:=1 to 4 do begin  {prejdeme vsetyk smery, ktorymi sa da potencialne ist}
      nx:=x+smery[i][1];
      ny:=y+smery[i][2];
      if (nx>0) and (nx<=m) and (ny>0) and (ny<=n) then
        if (vyska[x,y]>vyska[nx,ny]) and (bol[nx,ny]=-1)then begin
          bol[nx,ny]:=bol[x,y]+1;
          put(x,y);
        end;
    end;
    inc(fb);
  until (fb = fe); {ak mam prazdny front, mozem skoncit}
  if bol[cx,cy] > -1 then
    writeln(bol[cx,cy]) else
    writeln('Neexistuje cesta');
  readln;
end.
```

4. Olivy vs. Godzilla

opravoval Maják
(max. 15 bodov)

Tento príklad mal jednu príjemnú vlastnosť. Stačilo sa trochu pohrať s olivami, ani postup ste písať nemuseli, a už ste mali istých pár bodov. V čom bol najväčší problém (pre niekoho možno najväčšia výhoda) tohto príkladu? Všetkých možností rozmiestnenia olív je $2^{16 \cdot 16}$ Ak by sme ich chceli všetky vyskúšať hrubou silou, tak aj keby sme zrátanie výnosu jednej možnosti vedeli spraviť v jednej inštrukcii procesora a spustili náš program na všetkých počítačoch sveta, tak by to trvalo oveľa oveľa viac ako je odhadovaný vek vesmíru.[7] A to sme do kalkulácií vôbec nezarátali rôzne možnosti zašliapávania záhradky...

Tadiaľto teda cesta nevedie. Problém je, že ak chceme nájsť určite najlepšie riešenie, tak ona zatiaľ nevedie nikadiaľ. Keby sa niekomu z vás podarilo nájsť niečo efektívnejšie, tak už dávno nerieši KSP, ale si užíva nekonečnú slávu v Karibiku.

[7] $2^{16 \cdot 16} = 115792089237316195423570985008687907853269984665640564039457584007913129639936$

Druhá možnosť je, že sa uspokojíme aj s riešením, o ktorom síce nebudeme vedieť, že je najlepšie, ale je dostatočne dobré. Ale ako také riešenie nájsť?

Jedna z alternatív je použiť rôzne heuristiky. Heuristika, to je vlastne len honosný názov pre intuíciu. Keď si na papieri začneme kresliť rôzne olivové políčka a bude sa nám zdať, že vysoký počet zašľapnutí máme vždy pri nejakom vzore[8], tak si naprogramujeme taký program, ktorý bude generovať olivové záhrady obsahujúce ten vzor. Toto je príklad heuristiky – nič nám nezaručuje že nájdeme najlepšie riešenie, ale „vyzerá to tak“, že bude patriť medzi tie lepšie.

Jednou z možností je odskúšať náhodných milión možností a vybrať z nich tú najlepšiu. Toto jednoduché riešenie sa dá ešte viac zlepšiť, a to metódou zvanou horolezcov algoritmus[9]. Predstavme si, že sme horolezec, ktorý sa rozhodol vyliezť na Gerlachovský Štít. Máme ale tú smolu, že sa akurát spustila hmla, a mi nevidíme ani na koniec vlastného nosa. Preto postupujeme nasledovým spôsobom. Spravíme krok na sever, a zistíme, či sme šli do kopca. Potom sa vrátime krok späť, a zopakujeme to na všetky ostatné svetové strany. Náš cieľ bude pravdepodobne[10] tým smerom, pri ktorom sme zaznamenali najväčšie stúpanie.[11] Preto spravíme krok tam, a postup opakujeme: ideme vždy v smere najväčšieho stúpania, kým sa z našej pozície dá výjsť vyššie. Ak nie, sme na vrchole. Problém je, že kvôli hmle nevidíme, či sme náhodou nezablúdli na Veľkú Svišťovku, ktorá je nižšia ako Gerlach.[12] Čo s tým? Zostúpime niekam náhodne dolu, a celý tento výstup niekoľkokrát zopakujeme z rôznych štartovacích miest. Dúfame, že niektorým pokusom sa dostaneme na úplné maximum (hoci si nikdy nebudeme úplne istí). Všimnite si, že pri tomto postupe hrá veľkú úlohu vyššie spomínaná intuícia.

Na náš problém sa dá tiež pozerať ako na hľadanie Gerlachu: Každé riešenie – olivovú záhradu – si môžeme predstaviť ako nejaké miesto a jeho hodnotou – počet zadupaných olív – si môžeme predstaviť ako jeho nadmorskú výšku. Takto vlastne hľadáme najvyššie miesto alebo vrchol. Ak do záhrady pridáme alebo uberieme pár olív, akoby sme sa pohli v nejakom smere. Vyberieme si tú možnosť, ktorá je najlepšia. Takto stúpame, kým už olivovú záhradu nevieme nijak zlepšiť. Ak je to najlepšia záhrada, akú sme kedy našli, zapamätáme si ju a začneme z iného náhodného miesta (s novou, náhodnou záhradou).

Najlepšie riešenia boli tie, ktoré dokázali minimálnym počtom olív vyrábať maximálny počet nových olív, ktoré hneď zašľapli. Ako najvýhodnejšie sa ukázalo zoskupovať tri olivy vedľa seba do tvaru písmena L. Problém bol vysporiadať sa so situáciou pri okraji záhrady. To najlepšie zvládol Martin Baláž, pozrime si jeho riešenie:

[8] angl. pattern

[9] jazykové okienko pokračuje, tentokrát si zapamätajte slovo hillclimbing

[10] hoci nie nutne

[11] lepšie ako ísť dolu kopcom...

[12] máme lokálne maximum, ale nevieme, či je aj globálne

```
.o..o..o..o..o..
oo.oo.oo.oo.oo..
..............o.
.o..o..o..o..o..
oo.oo.oo.oo.oo..
..............o.
.o..o..o..o..o..
oo.oo.oo.oo.oo..
..............o.
.o..o..o..o..o..
oo.oo.oo.oo.oo..
.............o..
.o..o..o..o...o.
oo.oo.oo.ooo.o..
..o..o..o...o.oo
...............o
```

Na na záver veľmi dôležitá rada – vždy si poriadne prečítajte *celé* zadanie, aby ste vedeli, akým spôsobom máte odovzdať riešenie. V opačnom prípade môžete ľahko prísť o body, čo sa žiaľ toto kolo niektorým z vás podarilo.

5. O japonskom dadaizme

opravoval Adam
(max. 15 bodov)

Tento príklad nebol z tých najľahších a veľa optimálnych riešení neprišlo. Väčšina riešení sa uspokojila s jednoduchým spôsobom ako vypočítať výsledok v lineárnom čase. Optimálne riešenie je prirodzene prefíkanejšie a bol som veľmi rád, že sa našli riešitelia, ktorí ho vymysleli. Poďme teda na to, čo nás všetkých zaujíma, na samotný vzorák.

Postup bude nasledovný: Odvodíme nejakú jednoduchú rekurenciu pre počet slov dĺžky N, dôjdeme k podozreniu, že by mohlo ísť o chronicky známu postupnosť, odbočíme, dokážeme pre túto postupnosť nejaké veci, našu rekurenciu naštelujeme tak, aby sme využili to, čo sme práve dokázali pre známu postupnosť, ukážeme si fintu ako túto postupnosť rýchlo spočítať a bude po príklade a všetci budeme zase o kus múdrejší. Tak čo, ideme na to? :-)

Pre poriadok, najprv si označme veci s ktorými budeme chvíľu pracovať. Mechagodzillu budeme skrátene označovať znakom A, Godzillu ako B a bodku ako C. Označenie P_i bude zodpovedať počtu reťazcov dĺžky i, ktoré v sebe neobsahujú žiadny podreťazec AB (podľa podmienok úlohy).

Tvrdím, že ak poznáme hodnotu P_{i-1} a P_{i-2}, tak dokážeme spočítať aj hodnotu P_i. Jednoducho spočítame, koľko reťazcov získame, ak zreťazíme každý vyhovujúci reťazec dĺžky $i-1$ so znakom z množiny $\{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ a samozrejme odčítame tie reťazce, ktoré predtým končili písmenom A, a my sme ich práve zreťazili s B (t.j. také zreťazenia, ktoré by boli neplatné). Rečou rovnice, $P_i = 3P_{i-1} - P_{i-2}$ (zamyslite sa, prečo sme odčítali práve P_{i-2} reťazcov). Zjavne $P_0 = 1$, $P_1 = 3$ a $P_2 = 8$. Potiaľto sa vás v riešení dostalo vcelku dosť a v tejto fáze už nie je ťažké vymyslieť algoritmus ktorý s časovou zložitosťou $O(N)$ spočíta výsledok.

Tí z vás, ktorí si vygenerovali pár hodnôt,

$$1, 3, 8, 21, 55, 144, \ldots$$

mohli pojať (oprávnené) podozrenie, že naša rekurencia je iba kamuflovaná sestra Fibonacciho postupnosti (postupnosť, ktorá začína $F_0 = 0$, $F_1 = 1$ a každé ďalšie číslo je vždy súčtom dvoch predošlých, teda $F_n = F_{n-1} + F_{n-2}$ (pre $n \geq 2$). Prvých zopár členov Fibonacciho postupnosti je

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

Je teda vážne podozrenie, že $P_n = F_{2n+2}$. Dokáže sa to jednoducho: Pozrime sa napríklad na také F_{2i}. Také $F_{2i} = F_{2i-1} + F_{2i-2} = (F_{2i-2} + F_{2i-3}) + (F_{2i-3} + F_{2i-4}) = F_{2i-2} + 2(F_{2i-2} - F_{2i-4}) + F_{2i-4} = 3F_{2(i-1)} - F_{2(i-2)}$. To sú ale výborné správy, naša rekurencia pre číslo N zodpovedá $(2N + 2)$-tému Fibonacciho číslu.[13]

Tu mnohí z vás skončili a uspokojili sa s počítaním Fibonacciho čísel v lineárnom čase. Tí z vás, ktorí riešili KSP minulý rok, alebo aspoň trochu hľadali na internete mohli dospieť k poznaniu, že to ide aj v logaritmickom čase. Povieme si o dvoch známych spôsoboch ako toto dosiahnuť.

Prvý vychádza z postupu počítajúceho zaokrúhlením[14]. Platí totiž, že $F_n = \lfloor \varphi^n/\sqrt{5} + 1/2 \rfloor$. Jediné, čo teda potrebujeme spraviť je, že nejak šikovne umocníme $\varphi = 1.61803399$ na N (čoskoro si povieme ako).

Druhý spôsob je trochu fintový. Tí z vás čo vedia čo je matica a ako sa násobí nech tento odsek pokojne preskočia. Pre tých ostatných vysvetlenie - matica je také matematické „zvieratko“, ktoré má vo všeobecnosti m riadkov, n stĺpcov a v každom políčku má nejaký rozumný matematický objekt (napríklad prirodzené číslo je dosť rozumné). S takýmito maticami vieme robiť všelijaké veci a napríklad ich aj násobiť. To sa robí tak, že sa vezmú dve matice A a B, a na index i, j v matici C sa dá (v našom prípade) číslo ktoré vznikne tak, že vezmeme prvý prvok z i-teho riadku matice A, vynásobíme ho s prvým prvkom j-teho stĺpca v B a výsledok sčítame s číslom ktoré získame ako súčin druhého prvku i-teho riadku v A s druhým prvkom v j-tom stĺpci v B a to zas sčítame s... atď kým sa nedôjdeme na koniec riadku v matici A, resp. na koniec stĺpca v matici B. Výsledkom je matica C, ktorá má toľko riadkov, koľko mala matica A a stĺpcov toľko, koľko mala matica B. Na koniec si len všimnite, že ak sa počet stĺpcov matice A nesedí s počtom riadkov matice B, násobenie nie je definované, lebo nám dôjdu prvky skorej, ako ich stihneme všetky navzájom ponásobiť.

O niečo viac formálnejšie, ak matica A má m riadkov a n stĺpcov, označíme si jej prvky A_{ij} pre $i = 1 \dots m, j = 1 \dots n$. Podobne, ak matica B má rozmery n krát p, potom jej prvky označíme B_{ij}, $i = 1 \dots n, j = 1 \dots p$. Potom výsledná matica $C = AB$ má rozmery m krát p a prvok $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$. Toto násobenie má pár vlastnosti, aké by človek od násobenia očakával. Napríklad, $(AB)C = A(BC)$. Ale napríklad $AB = BA$ bohužiaľ nie vždy platí. Často, aká to náhoda, že zrovna v tomto príklade, sa vám môže pod ruky dostať rekurencia, typu: na vypočítanie súčasnej hodnoty potrebujem vedieť len pár predošlých hodnôt, vynásobiť ich pár konštantami a sčítať. Napríklad, na vyrátanie n-teho Fibonacciho čísla potrebujeme len sčítať predošlé dve. No a presne na toto viem pekne využiť matice. Povedzme, že chceme vyrátať n-tý člen rekurencie b_n, ktorá záleží iba na k predošlých členoch nejako takto: $b_n = \sum_{i=1}^{k} a_l b_{n-i}$ kde $a_1, \dots, a_k$ sú len nejaké konštanty. Takto sme ešte postupnosť jednoznačne neurčili a pre odpichnutie sa si ešte musíme určiť úvodné hodnoty. Teda, musíme si pevne určiť $b_1, \dots, b_k$. A teraz už môžme pokne počítať. Z prvých k hodnôt viem vypočítať hodnotu $k + 1$, potom hodnotu $k + 2$, $k + 3$ a tak ďalej. A keď nás to omrzí, tak si všimneme, že v podstate len umocňujeme dáke matice. A to presne takého: nech matice B má k riadkov a jeden stĺpec. A $B_{11} = b_1, B_{21} = b_2, \dots, B_{k1} = b_k$. Inými slovami, matica B obsahuje počiatočné hodnoty rekurencie. A teraz do matice A rozmerov k krát k zakódujeme informácie o samotnej rekurencii takto. V prvých $k - 1$ riadkoch máme skoro samé nuly okrem pozície $A_{i,i+1}$ kde máme 1. A v poslednom riadku máme $A_{ki} = a_{k-i+1}$. Čo sa teraz stane ak vynásobíme AB? Čuduj sa svete, ale dostane maticu znovu o k riadkoch a jednom stĺpci, ktorá teraz obsahuje prvky $b_2, \dots, b_{k+1}$. Ak túto maticu znova prednásobíme Áčkom, dostane zase prvky $b_3, \dots, b_{k+2}$ a tak ďalej. Pozorní čitateľia si už určite všimli, že ak chceme vypočítať hodnotu b_{n+k}, stačí nám umocniť maticu A na n-tú, prenásobiť Béčkom a zobrať prvok v poslednom riadku. A najlepšie na tomto, a v podstate čaro celého triku

[13] Dokončenie dôkazu by bolo indukciou: $P_0 = F_2$, $P_1 = F_4$ a ak predpokladáme, že $P_{i-1} = F_{2(i-1)}$ a $P_{i-2} = F_{2(i-2)}$, tak $P_i = P_{i-1} + P_{i-2} = F_{2(i-1)} + F_{2(i-2)} = F_{2i}$.

[14] `http://en.wikipedia.org/wiki/Fibonacci_number#Computation_by_rounding`

je v tom, ako je vysvetlené neskôr, že ľubovoľnú danú štvorcovú maticu vieme umocniť na N-tú v čase $O(\log N)$. Napríklad, v našom konkrétnom prípade s Fibonacciho číslami, máme $A = ((0,1),(1,1))$ a úvodná hodnota B je $((0),(1))$.[15]

Ostal som vám dlžný ešte jednu vec – ako sa umocňuje v logaritmickom čase. Nuže, majme objekt A a chceme zistiť hodnotu A^N. Postupujme týmto spôsobom:

a) ak $N = 2k$, tak vypočítame A^k a výsledok umocníme na druhú.

b) ak $N = 2k+1$, tak vypočítame A^{N-1} podla postupu v a) a výsledok vynásobíme A.

Tento postup očividne redukuje náš problém vždy na problém polovičnej veľkosti. Napríklad pre $N = 9$, by sme postupovali takto:

$$A^9 = (A^4)^2 \quad A^4 = (A^2)^2 \quad A^2 = (A)^2$$

No a teraz je už len na nás, či použijeme tento postup na φ, alebo na maticu – v oboch prípadoch sa dopracujeme k algoritmu s časovou zložitosťou $O(\log N)$ a pamäťovou zložitosťou $O(1)$, čo je vzorové riešenie. Výhodou maticového riešenia je, že nepoužívame reálne čísla a nehrozia zaokrúhľovacie chyby; číslo ϕ je iracionálne a niekde ho musíme useknúť.

Teraz poďme k tomu, ako som vaše (miestami vtipné) riešenia hodnotil. Vzhľadom k tomu, že úloha sa dala riešiť aj „Odokinovou“ metódou (tzv. pozriem-vidím), hodnotil som ju trocha prísnejšie. Samotné riešenia boli hodnotené 0–14 bodmi a bodík som pridával, ak bol k riešeniu pridaný bezchybný popis v ktorom sa objavil aspoň náznak dôkazu používaných tvrdení (obzvlášť ak ste výsledok počítali pomocou Fibonacciho čísel). Za riešenie v logaritmickom čase sa udeľovalo 14 bodov, v lineárnom 10 a v horšom 8 bodov. V prípade lineárnych a lepších riešení som za logaritmickú pamäťovú zložitosť strhával bod, za lineárnu dva, za horšiu až tri body. Nefunkčné riešenia mohli dostať nanajvýš 6 bodov. Pokiaľ vášmu riešeniu úplne chýbal popis alebo kód, mínus štyri body boli vašim trestom. A aby toho nebolo dosť, za chýbajúce alebo nesprávne zložitosti ste mohli stratiť po bode.

Listing programu:

```
#include <iostream>
using namespace std;
struct Matica{ long f1,f2,s1,s2;};

Matica vynasob(Matica A,Matica B){
    Matica C;
    C.f1 = A.f1*B.f1 + A.f2*B.s1;
    C.f2 = C.s1 = A.f1*B.f2 + A.f2*B.s2;
    C.s2 = C.f1 - C.f2;
    return C;
}

double Fibonacci(int N){
    Matica X = {1,1,1,0};
    Matica A = {1,1,1,0};

    if(N==0) return 0; else N--;

    while(N>0){
        if(N%2==1) A = vynasob(A,X);
        X = vynasob(X,X);
```

[15] Pre fajnšmekrov. Dá sa ukázať, že ak matica A spĺňa istú celkom príjemnú obskúrnu vlastnosť, tak A sa dá napísať ako $A = UDU^{-1}$ kde D má všetky mimo (hlavno-)diagonálové prvky nulové. A potom $A^n = UD^nU^{-1}$ kde D^n sa do dostať z D len umocnením každého diagonáloveho prvku na n-tú. A presne takto sa dá dokázať vyššie spomínaná rovnosť s φ.

```
        N /= 2;
    }
    return A.f2;
}

int main(){
 int N,vysledok;
 cin >> N;
 cout << Fibonacci(2*(N+1));
 return 0;
}
```

Opravoval Zemčo
(max. 20 bodov)

6. Osaka pod Godzillou

Tento príklad bol pekný na rozmýšľanie, pričom na dobré riešenie nebolo treba poznať nijaký ťažký algoritmus. Prišli ale riešenia všelijaké a niektoré boli vskutku nápadité a originálne. Kam vy na to chodíte?

Vyskytlo sa riešenie, ktorého čas behu závisel od výšok mrakodrapov. Toto môže viesť k strašnej neefektivite, keďže výšky mrakodrapov neboli špecifikované a teoreticky mohli byť veľmi vysoké. Vo všeobecnosti sa podobným veciam snažte vyhnúť.

Jedna z možností, ako riešiť úlohu, je skúšať budovy spracovávať postupne a pre každú vypočítať, koľko najviac skokov môžeme spraviť, ak chceme skončiť na nej. Ak tento postup realizujeme od prvej budovy smerom doprava, tak túto hodnotu môžeme zistiť s pomocou predtým vypočítaných hodnôt – takéto myšlienky nazývame *dynamické programovanie*. Pre prvú budovu je to ľahké – je to vždy 0. Pre každú ďalšiu budovu to nie je nič iné ako maximum spomedzi všetkých budov, z ktorých môžeme na práve uvažovanú budovu skočiť. Samozrejme, ešte plus jedna za skok na danú budovu. Prejsť všetky už spracované budovy a určiť toto maximum zvládneme jedným for cyklom smerom späť k počiatočnej. Z nejakej budovy môžeme na aktuálne uvažovanú skočiť vtedy, ak je aspoň taká vysoká a žiadna vyššia skoku nebráni a tieto podmienky pri cykle overíme ľahko. Takéto riešenie bude pracovať v čase $O(N^2)$, pretože pre N budov sa budeme vždy vracať na začiatok pri hľadaní maxima.

Ešte sa dá poznamenať, že toto riešenie sa dá realizovať aj od konca smerom na začiatok: pre danú spracovávanú budovu si zistíme, koľko najviac skokov a označení môžeme spraviť, ak by sme začínali na tejto budove. Využijúc tieto hodnoty vypočítané pre budovy smerom ku koncu ľahko nájdeme maximum spomedzi budov, na ktoré môžeme skočiť.

Vzorové riešenie trochu pripomenie kvadratické, avšak využijeme ešte nejaké pozorovania navyše.

Uvažujme ľubovoľnú budovu b niekde v strede postupnosti zo vstupu. Možno sa to na prvý pohľad nezdá, ale ak máme záujem dosiahnuť najdlhšiu postupnosť skokov, tak budova, z ktorej skočíme na budovu b, je jednoznačne určená: Ak by sme mohli skočiť na budovu b z budov a_1 aj a_2 (nech a_1 je skôr), potom lepšia možnosť je skočiť z a_1 na a_2 a potom z a_2 na budovu b. Premyslite si, prečo sú obidva skoky uskutočniteľné. Vo všeobecnosti, ak sa dá skočiť na b z budov $a_1, a_2, \ldots, a_k$, tak sa dá postupne skákať z a_1 na a_2, z a_2 na a_3, atď., až z a_{k-1} na a_k a z a_k na b. Takže predchodca budovy b pri hľadaní najkratšej postupnosti je jednoznačne určený: Je to prvá aspoň tak vysoká budova smerom k začiatku postupnosti[16].

Poďme teraz skúsiť vymyslieť rýchly algoritmus. Predstavme si, že spracovávame budovy od začiatku. Pre nejakú spracovávanú budovu chceme určiť jej predchodcu (vďaka čomu budeme schopní ľahko a rýchlo určit maximum skokov, ktoré môžeme spraviť, ak chceme skončiť

[16] Ak taká neexistuje, potom b musí byť vyššia ako počiatočná budova, a preto ona, ani žiadna za ňou nemôže byť súčasťou najdlhšej postupnosti.

na tejto budove). V každom momente budeme mať zoznam kandidátov na predchodcov (zoznam budov, ktoré môžu byť predchádzajúce k spracovávanej). Navyše, budeme mať tento zoznam utriedený podľa výšok od najväčšej po najmenšiu. Pouvažujte, prečo bude tým pádom zároveň utriedený aj od najskoršej po najneskoršiu. Na začiatku bude v tomto zozname len prvá budova. Ako máme postupovať, keď sme niekde v strede postupnosti a chceme spracovať ďalšiu budovu b? Našou úlohou je určiť maximum skokov, ktoré dokážeme urobiť, ak chceme skončiť na tejto budove a rovnako si potrebujeme aktualizovať zoznam. Poďme nájsť predchodcu. Začneme od najnižších kandidátov. Je najnižšia budova vyššia alebo rovnaká ako b?

- Ak áno, potom sme hotoví: našli sme práve jednoznačne určenú predchádzajúcu budovu[17]. Zaradíme budovu b na koniec zoznamu a nastavíme jej hľadanú hodnotu na hodnotu jej predchodcu +1. Takto sme dostali aktualizovaný zoznam, pretože budova b môže byť predchodcom nejakej ďalšej budove. V mojej implementácii sa vyhýbam dvom budovám v zozname s rovnakou výškou, preto v prípade rovnosti výšok pôvodného predchodcu vyhodíme. Keď sa nad tým zamyslíme, tak už pre nás nie je zaujímavý, pretože každá postupnosť z neho k budovám ďalej bude pokračovať cez budovu b.
- Ak nie, potom je jasné, že najnižší kandidát predchodca nebude. Ba čo viac, jasné je aj to, že z toho najnižšieho kandidáta už žiadna dlhšia postupnosť ako jeho maximum pokračovať nebude: skok cez budovu b totiž nie je možný. Preto tento najnižší kandidát prestáva byť kandidátom a môžeme ho zo zoznamu vyhodiť. Postupovať budeme ďalej tak, že sa skúsime pozrieť na druhého najnižšieho kandidáta a ak je nižší ako budova b, tak ho zo zoznamu vyhodíme. Takto budeme kandidátov vyhadzovať, až kým sa nám nepodarí nájsť kandidáta, ktorý je vyšší alebo rovnako vysoký ako naša budova b. Ako sme si ukázali, toto je jednoznačne určený predchodca. Všetci ďalší kandidáti ostávajú byť kandidátmi, pretože vždy môže v nasledujúcom kroku prísť budova, ktorej jednoznačne určený predchodca bude práve niekto z nich. A rovnako sa stáva kandidátom na pokračovanie aj budova b, ktorú zaradíme na koniec zoznamu kandidátov, pretože je v ňom určite najnižšia.

Ako uvedený zoznam naprogramovať? Dá sa to aj v obyčajnom poli, kde si budeme uchovávať ukazovateľ na posledný prvok. To nám bohato postačuje, pretože vždy pracujeme len s ním. Alebo ho mažeme zo zoznamu, alebo na koniec zoznamu pridávame budovu. Isto väčšina z vás vie, že toto sa volá zásobník – dátová štruktúra, z ktorej môžeme vybrať najneskôr vložený prvok, alebo vložiť prvok na vrch. Zásobník pripomína zaneprázdneného úradníka, ktorý ma na stole hromadu papierov, na vrch ktorej hádže nové papiere a keď si náhodou nájde čas, tak berie papiere z vrchu a rieši ich.

Takto počítané maximá jednotlivých budov sú určite dĺžky korektných postupností skokov, pretože algoritmus zaručí, že každý skok bude uskutočniteľný. Rovnako to budú najdlhšie postupnosti, pretože sme si dokázali, že optimálne vyberá predchodcov. Nezaškodí ošetriť prípad, keď načítame budovu, ktorá je vyššia ako počiatočná, aby sa nám maximá rátali správne.

Ostáva zodpovedať otázku zložitosti nášho riešenia. Možno to nie je zjavné, ale je lineárna. Zoznam kandidátov môže byť až taký veľký, ako je počet budov. A v jednom kroku ho možno celý môžeme prejsť (ak nám príde vysoká budova). Nie je to potom kvadratické? Nuž, nie je, a to preto, že keď raz zoznam prechádzame, tak odtiaľ budovy aj vyhadzujeme. A budova, ktorú vyhodíme, sa už nevráti, lebo nemá nožičky. Dá sa dokonca povedať, že každú budovu do zoznamu najviac raz vložíme a najviac raz vyberieme. Takže, z iného pohľadu: každá budova nás stojí konštantný počet krokov. A preto bude celkový počet krokov lineárny

[17] Pamätajte, že je to prvá aspoň tak vysoká budova ako b, a tú sme našli, keďže máme zoznam utriedený podľa výšky a zároveň podľa toho, ako sú od nás ďaleko

a časová zložitosť programu bude $O(N)$. Ak totiž nejaký krok trvá dlho, potom to znamená, že veľa iných krokov trvalo veľmi krátko. Pamäťová zložitosť bude tiež $O(N)$.

Podobne, ako pri pomalšom riešení, je možné realizovať tento postup aj z konca. Prechádzame budovy odzadu a zoznam udržujeme obrátene. Hodnota, ktorú pre každú budovu zisťujeme, nie je najväčší počet skokov, ktoré môžeme spraviť z počiatočnej budovy na túto budovu, ale je to počet skokov, ktoré môžeme spraviť, keby sme začínali na tejto budove a skákali smerom do konca.

Záverom ešte vetička o hodnotení: plný počet bol 20 bodov a tentokrát sa udeľoval celkom často. Za riešenie v čase $O(N \log N)$ som dával 16 bodov a za kvadratické riešenie 12 bodov. Pomalšie riešenia sa dočkali najviac 8 bodov. Ku klasickému hodnoteniu patrí aj stŕhanie bodov za nedostatočné (chýbajúce) odhady alebo popisy. Väčšinou to bolo v poriadku, ale opäť sa našli takí, ktorých program som pochopil, až keď som si simuloval jeho činnosť na nejakom vstupe a to nie je žiadúci stav. Rovnako poprosím, aby ste nad riešeniami nejedli a neposielali nám ich zamastené `:-)`.

Listing programu:

```
program osaka;
const MAXN = 1000000;
var budova, maxima, zasobnik : array[1..MAXN] of longint;
    maximum, pointer, i, N : longint;

begin
   readln (N);
   for i:=1 to N do maxima[i] := -1;
   { pointer je ukazovatel na posledne vlozeny prvok v zasobniku }
   pointer := 1;
   read (budova[1]);
   { 'vlozime' prvu budovu na zasobnik }
   zasobnik[1] := 1;
   maxima[1] := 0;
   for i := 2 to N do begin
      read(budova[i]);
      { hladame budovu v zasobniku, ktora je aspon taka vysoka ako nacitana.
       ostatne vyhadzujeme }
      while (pointer > 0) and (budova[i] > budova[zasobnik[pointer]]) do
dec(pointer);
      { minuli sa budovy v zasobniku?
       toto nastava, ak sme nacitali vyssiu budovu ako prvu }
      if (pointer = 0) then break
      else begin
         { mastavime maximum ako hodnotu predchodcu +1 }
         maxima[i] := maxima[zasobnik[pointer]]+1;
         { musime osetrit, aby neboli v zasobniku dve rovnake budovy }
         if (budova[i] = budova[zasobnik[pointer]])then dec(pointer);
         { a vlozime do zasobnika spracovanu budovu }
         inc (pointer);
         zasobnik[pointer] := i;
      end;
   end;
   {najdeme maximum}
   maximum := maxima[1];
   for i := 2 to N do if maximum < maxima[i] then maximum := maxima[i];
   writeln (maximum);
end.
```

7. O Godzillinom šmahu

opravoval Lukáš
(max. 20 bodov)

Najpomalšie riešenie, ktoré sa vyskytlo, bolo v čase $O(K^2)$, za ktoré ste mohli získať 8 bodov. Ide v podstate o skúšanie všetkých možností – všetkých úsekov je približne $O(K^2)$ a pre každý z nich si zrátame škodu, akú vieme spôsobiť. Tomuto riešeniu sa však nebudeme do detailov venovať, keďže ho skoro všetci vymysleli.

Vzorové riešenie pracuje v lineárnom čase a pamäti. Označme N počet bielych tankov a M počet čiernych tankov. Vytvoríme si novú postupnosť $x_1, \ldots, x_K$. Ak na i-tom mieste je biely tank, v novej postupnosti na i-tom mieste bude číslo $1/N$, teda $x_i = 1/N$. Ak je tam naopak čierny tank, v novej postupnosti budeme mať $x_i = -1/M$. Predstavme si, že chceme vyrátať výšku škody pre nejaký úsek, v ktorom je A bielych a B čiernych tankov. Výška škody je podľa definície $A/N - B/M$. Vieme to však spraviť aj pomocou novej postupnosti x. Keďže v danom úseku sa nachádza A bielych tankov, v novej postupnosti sa v zodpovedajúcom úseku bude nachádzať A-krát číslo $1/N$. Podobne sa v novej postupnosti bude nachádzať B-krát číslo $-1/M$. Spolu bude teda súčet v novej postupnosti pre daný úsek $A/N - B/M$.

Pomocou novej postupnosti teda vieme ľahko zistiť škodu pre nejaký úsek – stačí sčítať čísla na danom úseku. Našim cieľom je nájsť úsek s najväčšou škodou, teda v novej postupnosti úsek s najväčším súčtom. Pre jednoduchosť dôkazu si zaveďme novú postupnosť:

$$y_i = \sum_{j=1}^{i} x_j$$

y_i je súčet prvých i prvkov postupnosti x. Takýmto číslam sa múdro hovorí prefixové sumy. Vďaka nim vieme veľmi ľahko spočítať súčet nejakého úseku: súčet úseku od a po b je $\sum_{i=a}^{b} x_i = \sum_{i=1}^{b} x_i - \sum_{i=1}^{a-1} x_i = y_b - y_{a-1}$. Predstavme si, že pre daný pravý koniec chceme nájsť ľavý koniec tak, aby bol súčet úseku čo najväčší. Takže má platiť, že $y_b - y_{a-1}$ má byť čo najväčšie. Hodnota y_b je fixná, lebo pravým koncom teraz nevieme hýbať. Preto aby bol tento súčet maximálny, musí byť hodnota y_{a-1} minimálna.

Opäť si zrekapitulujme, k čomu sme sa dostali. Pre každé b chceme nájsť a také, že a je naľavo od b ($a \leq b$) a hodnota y_{a-1} je najmenšia možná. Označme $f(b)$ takéto a. Zamyslime sa, ako spolu súvisia minimálne hodnoty pre b a $b+1$. $b+1$ má v podstate rovnakých kandidátov na minimum ako b, ibaže má o jedného viac: $a+1$. Teda pre $b+1$ je $f(b+1)$ buď $f(b)$ (minimálna hodnota pre b), alebo $a+1$.

Všetky hodnoty $f(b)$ vieme ľahko vyrátať v lineárnom čase. Začneme s hodnotou $f(0) = 0$ a postupne dorátame hodnoty pre ďalšie čísla. Keď už poznáme hodnoty $f(b)$ pre všetky b, ľahko sa dopracujeme k výsledku celého algoritmu. Výsledkom bude najväčšia hodnota $y_b - y_{f(b)-1}$.

Ešte si dovolím pár slov k implementácii. Počas behu programu si nemusíme pamätať celé postupnosti $x_1, \ldots, x_K; y_0, \ldots, x_K$ a hodnoty $f(0), \ldots, f(K)$. Stačí si pamätať pre aktuálne b hodnoty $y_b, y_{f(b)}$ a $f(b)$. S konštantnou pamäťou si však nevystačíme, lebo pri prvom prechode vstupu musíme spočítať počet bielych a čiernych tankov.

Listing programu:

```
#include <iostream>
using namespace std;

int main()
{
    int k, n = 0, m = 0;
    cin >> k;
    int p[k];
    for (int i = 0; i < k; i++)
```

```
    {
        cin >> p[i];
        if (p[i] == 1)
            n++;
        else
            m++;
    }

    double naj = 0, akt = 0;
    int najind = 0;
    int zac = 0, kon = 0;
    double vys = 0;
    for (int i = 0; i < k; i++)
    {
        if (p[i] == 1)
            akt += 1.0 / n;
        else
            akt -= 1.0 / m;
        if (akt < naj)
        {
            naj = akt;
            najind = i + 1;
        }

        if (akt - naj > vys)
        {
            zac = najind;
            kon = i;
            vys = akt - naj;
        }
    }

    cout << "Godzilla znici usek od " << zac + 1 << " do " << kon + 1 << "." << endl;
}
```

8. Osakský Gusto

opravovala Monika
(max. 25 bodov)

V tejto úlohe nás zaujíma sila jednotlivých Godzíll v čase. Sila každej Godzilly je charakterizovaná dvojicou (d, f), ktorá nám udáva zmenu sily za jednu sekundu a počiatočnú silu Godzilly (pozor, v zadaní sú tieto hodnoty vymenené). Sila Godzilly v čase t sa teda dá vyjadriť ako lineárna funkcia $g(t) = d \cdot t + f$. Preto túto funkciu budeme v neskoršom texte volať *silová funkcia*, hodnotu d budeme v nasledujúcom riešení nazývať aj smernica a hodnoty f posun silovej funkcie g. Všimnite si, že nasledujúce tvrdenia platia aj pokiaľ niektoré smernice sú záporné. V zadaní sa však uvádzalo, že Godzilly zvyšujú svoju silu a teda väčšina vašich kódov neuvažovala túto možnosť.

Najjednoduchšia otázka, ktorú si asi každý pri riešení tejto úlohy položil je nasledovná: „Majme dve Godzilly (d_1, f_1) a (d_2, f_2). V akom čase je sila týchto Godzíll rovnaká?“ Odpoveď asi každý ľahko spočíta. Kedže sily v nejakom čase t sa majú rovnať, tak potom sa aj príslušné silové funkcie Godzíll musia rovnať $(d_1 t + f_1 = d_2 t + f_2)$. Pokiaľ $d_1 \neq d_2$ tak riešenie je jasné: $t = (f_1 - f_2)/(d_2 - d_1)$. Pokiaľ ale $d_1 = d_2$, tak podľa zadania $f_1 \neq f_2$ a teda neexistuje čas, kedy by tieto Godzilly boli rovnako silné. Godzilla s väčším posunom bude vždy silnejšia a druhá Godzilla bude vždy Godzi(te)liatko. Preto Godzilly, ktoré majú rovnaké smernice vieme nájsť, uvažovať len tú s najväčším posunom a zvyšné rovno označiť za Godzi(te)liatka.

Poďme teraz na riešenie úlohy. Predpokladajme, že všetky silové funkcie majú rôzne smernice. Budeme si ich postupne podľa rastúcej smernice kresliť. Vždy si zakreslíme len úsek silovej funkcie, kde je zo všetkých už nakreslených maximálna. Bude nám vznikať taká lomená čiara, ktorá je konvexná (pozrite si obrázky) a nekonečná (posledná silová funkcia keď sa stane maximálnou, už ňou aj zostane). Na začiatku máme prvú silovú funkciu, tá je zatiaľ maximálna na celom intervale $\langle 0, \infty)$ a teda je to polpriamka.

Predpokladajme, že už máme nakreslené úseky, kde sú silové funkcie so smernicou a posunom $(d_1, f_1), (d_2, f_2), \ldots, (d_{k-1}, f_{k-1})$ maximálne (na obrázku 1 je táto situácia zachytená pre 6 funkcii: a, b, c, d, e, f). Ideme pridať silovú funkciu pre (d_k, f_k). Silová funkcia pre (d_k, f_k) má väčšiu smernicu ako už nakreslené funkcie a preto ich niekde pretne. Pozerajme postupne sprava doľava už nakreslené úseky silových funkcii. Ak silová funkcia pre (d_k, f_k) pretne funkciu pre (d_{k-1}, f_{k-1}), tak sme skončili: Silová funkcia pre (d_{k-1}, f_{k-1}) je maximálna do času $\frac{f_{k-1}-f_k}{d_k-d_{k-1}}$, od tohto času sa stane maximálnou nová silová funkcia pre (d_k, f_k) (obrázok 2 znázorňuje túto situáciu pri pridávaní funkcie g_1).

Ak však silová funkcia pre (d_k, f_k) nepretne úsek funkciu pre (d_{k-1}, f_{k-1}) kde je maximálna, tak to znamená, že funkcia pre (d_k, f_k) bude na celom úseku nad funkciou pre (d_{k-1}, f_{k-1}) (lebo $d_{k-1} < d_k$). To ale znamená, že silová funkcia pre (d_{k-1}, f_{k-1}) nebude nikdy maximálna a teda ju môžeme zmazať z obrázku (pre Godzillu to znamená, že je Godzi(te)liatko). Teraz budeme v podobnej situácii ako sme boli na začiatku. Pokiaľ silová funkcia pretne úsek pre funkciu (d_{k-2}, f_{k-2}), tak sme skončili. Ak nie, tak potom znovu: silová funkcia pre (d_k, f_k) je nad silovou funkciou pre (d_{k-2}, f_{k-2}) na celom úseku, kde by táto funkcia mala byť maximálna. Preto môžeme úsek pre (d_{k-2}, f_{k-2}) zmazať a prehlásiť, že Godzilla (d_{k-2}, f_{k-2}) je Godzi(te)liatko. Takto pokračujeme ďalej, až zmažeme všetky nakreslené úseky, alebo nájdeme nejaký priesečník medzi nakreslenými úsekmi a novou silovou funkciou (d_k, f_k) a skončíme spracovávanie (na obrázku 3 môžete vidieť ako výsledok tohto postupu vyzerá po pridaní funkcie g_2).

Na záver, keď sú už všetky silové funkcie spracované, si prejdeme nakreslené úseky a určíme časy, kde sa maximalita silových funkcií mení (v konvexnej lomenej čiare tomu zodpovedá zlom). Ešte treba snáď ukázať špeciálny prípad, ktorý počas spracovávania môže nastať: Na obrázku 4 je príklad, keď pridávaná silová funkcia g_3 sa pretne s funkciou e v jej krajnom bode. V tomto prípade nebolo presne povedané, ako má odpoveď vyzerať a preto obe možnosti sú v poriadku: buď je odpoveď interval, ktorý začína aj končí v tom istom bode, alebo je príslušná Godzilla godzi(te)liatko.

Pre úplnosť je ešte na obrázku 5 znázornený prípad ak by mali dve silové funkcie (f a g_4) rovnakú smernicu. Z obrázku je jasné, že funkciu s menším posunom netreba uvažovať a príslušná Godzilla je vždy godzi(te)liatko.

Implementácia tohto riešenia je ľahšia ako sa môže zdať. Najskôr si Godzilly utriedime podľa nasledovného usporiadania: $(d_1, f_1) < (d_2, f_2)$ práve vtedy ak $d_1 < d_2$ alebo $d_1 = d_2$ a súčasne $f_1 < f_2$. Spracúvavanie silových funkcií potom budeme riešiť cez pole, ktoré simuluje zásobník. V ňom si stačí pamätať postupne spracúvavané silové funkcie (v našom prípade index do utriedeného poľa so smernicou a posunom pre Godzílly). Vždy keď začneme spracúvavať nasledovnú, tak porovnávame časy prieniku dvoch posledných silových funkcii na zásobníku a čas prieniku poslednej silovej funkcie na zásobníku a spracúvanej silovej funkcie. Podľa porovnania týchto časov vieme rozhodnúť, že či silová funkcia navrchu zásobníku je niekedy maximálna (nechať ju v zásobníku) alebo nie je (zmazať ju zo zásobníku). Na záver spočítame krajné body intervalov ako časy priesečníkov susedných silových funkcií, ktoré zostali v zásobníku.

Toto riešenie potrebuje čas $O(N \log N)$ na triedenie dvojíc smernice a posunu a potom čas $O(N)$ na prácu so zásobníkom. Toto by malo byť jasné z faktu, že každá silová funkcia je raz spracúvavaná, nanajvýš raz pridaná do zásobníka a nanajvýš raz zmazaná. Pamäťová zložitosť je lineárna od počtu Godzíll.

Ak ste sa dočítali až sem a bonus k riešeniu sa vám už nechce čítať, tak nasledujúce odstavce preskočte a choďte na posledný, kde je bodovanie. No a pre ostatných si povieme ešte niečo ku *konvexnému obalu*[18]. To, že náš algoritmus počíta dolnú hranicu polrovín určenú priamkami reprezentujúcimi sily Godzílla by už z obrázkov a prechádzajúceho opisu malo byť jasné. Hornú hranicu vieme počítať rovnakým spôsobom, akurát že najskôr musíme funkcie preklopiť podľa x-ovej osi. Teraz si ale asi položíte otázku, že to je síce pekné ale čo to má s konečným obalom. No a na odpoveď si najskôr spomenieme zopár pekných vlastností, ktoré sa zvyknú nazývať ako dláta bodov a priamok.

Budeme mať dve roviny. Primárnu rovinu a k nej duálnu rovinu. Priamka $y = ax + b$ budeme reprezentovať bodom $(a, -b)$ v duálnej rovine a bod (p_x, p_y) budeme reprezentovať priamkou $y = p_x x - p_y$ v duálnej rovine. Všimnite si, že toto platí aj opačne a teda vieme body a priamky zobrazovať z duálnej roviny do primárnej. K priamke l v primárnej rovine budeme označovať ako $l\cdot$ bod, ktorý k nej prislúcha v duálnej rovine. Rovnako k bodu p v primárnej rovine budeme označovať ako $p\cdot$ priamku, ktorá k nemu prislúcha v duálnej rovine. Potom platia nasledujúce vlastnosti[19]:

- $(p\cdot)\cdot = p$ a $(l\cdot)\cdot = l$
- v primárnej rovine bod p leží nad/na/pod priamkou l práve vtedy ak v duálnej rovine priamka $p\cdot$ prechádza pod/cez/nad bodom $l\cdot$
- priamky l_1 a l_2 sa pretínajú v bode p práve vtedy keď v duálnej rovine priamka $p\cdot$ prechádza cez body $l_1\cdot$ a $l_2\cdot$
- tri body sú kolineárne (ležia na tej istej priamke) vtedy a len vtedy keď ich duálne priamky sa (v duálnej rovine) pretínajú v tom istom bode

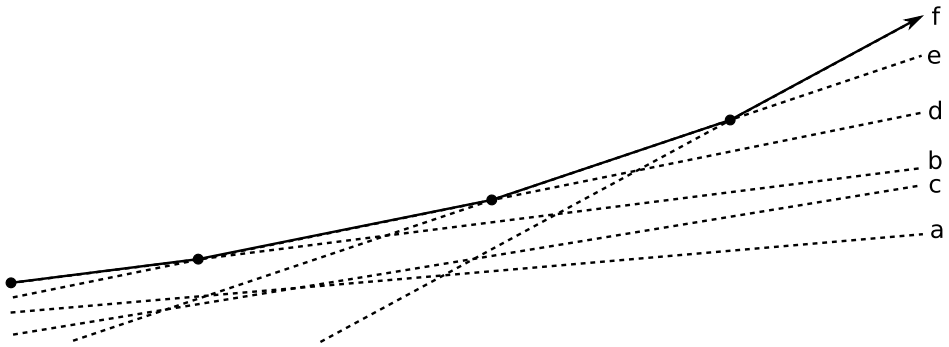
No a na záver vyslovím najdôležitejšie tvrdenie[20]. Nech máme dané priamky určené smernicou a posunom $(d_1, f_1), (d_2, f_2), \ldots (d_N, f_N)$. Potom dolná hranica polrovín určená týmito priamkami (zľava doprava) je rovnaká ako horný konvexný obal určený duálnymi bodmi $(d_1, -f_1), (d_2, -f_2), \ldots (d_N, -f_N)$ (proti smeru hodinových ručičiek). Toto je teda priamy návod na to, že ako použiť naše riešenie na výpočet konvexného obalu. Stačí vziať dané body. Rozdeliť si ich na hornú a dolnú časť (podla priamky určenej bodmi s maximálnou a manuálnou x-ovou súradnicou), z každej časti vyrobiť duálne priamky, vypočítať ich hranicu a potom výsledok previesť spať. Na úplný záver ešte poznamenám, že táto dualita platí medzi bodmi a priamkami, ktoré nie sú rovnobežné s y-ovou osou (pre rovnobežne priamky neexistuje smernicový tvar).

Bodovala som nasledovne. Za správne riešenie v časovej zložitosti $O(N \log N)$ ste mohli získať 23 až 25 bodov. Kvadratická časová zložitosť vášho programu bola hodnotená so 14-timi až 17-timi bodmi. Riešenie, ktoré zvyšovalo čas po jednej sekunde a kontrolovalo, či nejaká Godzilla sa nestane silnejšou mohlo získať niečo medzi 3-mi a 8-mimi bodmi. Body ste mohli stratiť za drobné bugy, myšlienkové chyby, chýbajúce alebo slabé popisy a chýbajúce odhady. V prípade opisovania sa celkový počet získaných bodov delil medzi opisovačov a zaokrúhľoval dole.

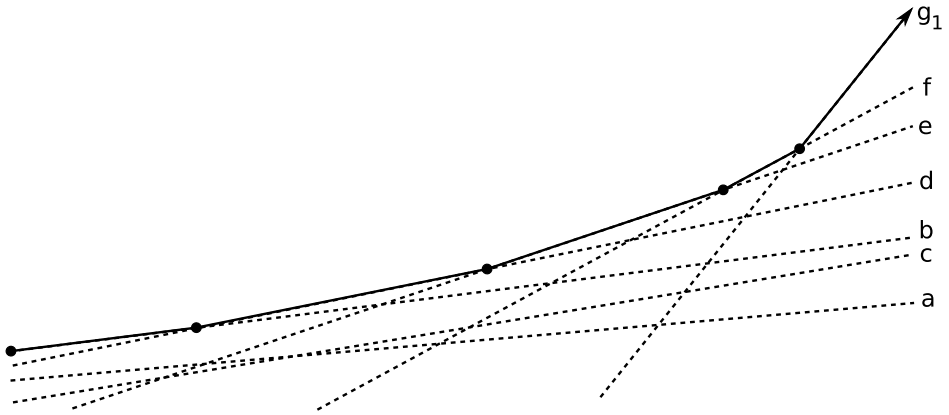
[18] kto ho nepozná, tak nech si pozrite wikipediu: `http://en.wikipedia.org/wiki/Convex_hull`

[19] Dokázať si ich viete rozpísaním daných vlastností a s trochou počítania.

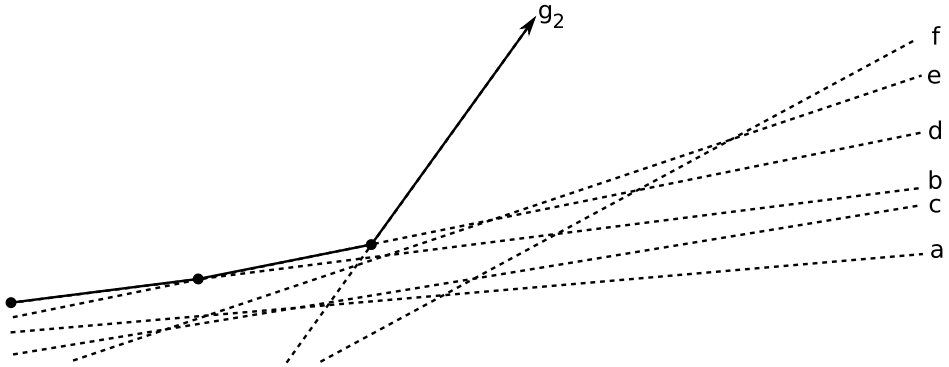
[20] Dôkaz ponecháme na usilovného čitateľa.;-)



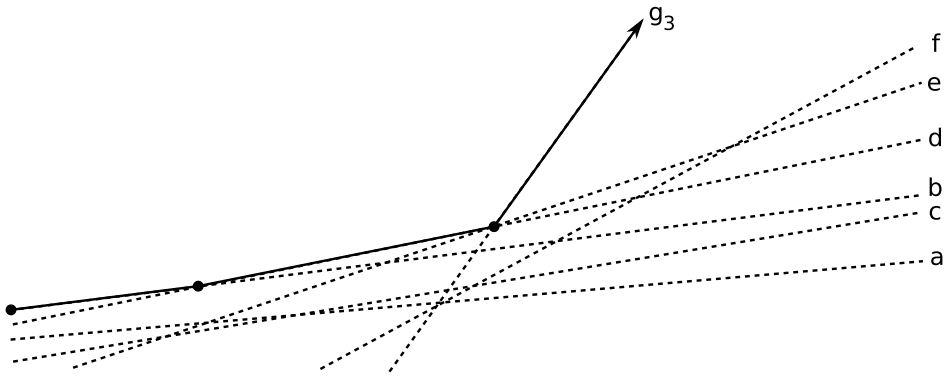
Obrázok 1



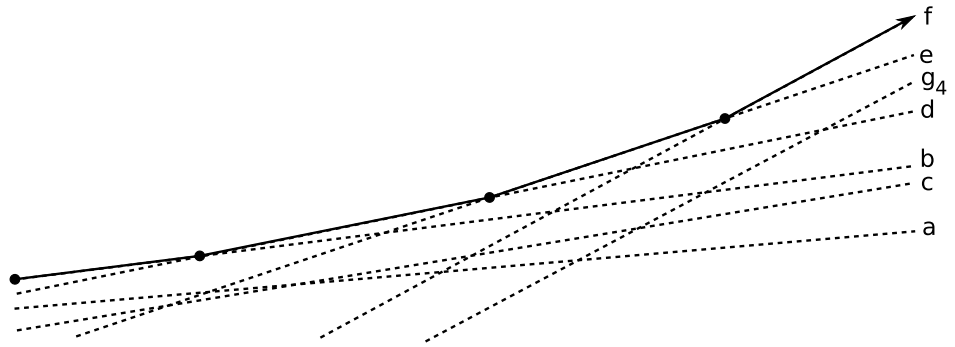
Obrázok 2



Obrázok 3



Obrázok 4



Obrázok 5

Listing programu:

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 10000                    // maximalny pocet Godzill
#define INF 100000                   // velke cislo

int N;

struct GODZILLA {
    double d;
    double f;
    int id;                                          // poradie Godzilly pri
nacitani
} G[MAX];

int S[MAX];
int X[MAX];
int last=0;
double from[MAX];
double to[MAX];

int cmp(const void *pp1, const void *pp2) {
    const struct GODZILLA *p1 = (const struct GODZILLA *)pp1;
    const struct GODZILLA *p2 = (const struct GODZILLA *)pp2;
    if (p1->d < p2->d || (p1->d == p2->d && p1->f < p2->f)) return -1;
    if (p1->d > p2->d || (p1->d == p2->d && p1->f > p2->f)) return 1;
    return 0;
}

double get_time(int a, int b) {              // zisti cas kedy sila Godzilly G[a] a G[b]
je rovnaka
    return ((G[a].f-G[b].f)/(G[b].d-G[a].d));
}

int main(void) {
    int i,n;
    double t1,t2;
    scanf("%d",&N);
    for(i=0;i<N;++i) {
        scanf("%lf %lf",&G[i].f,&G[i].d);
        G[i].id = i;
    }
    qsort(G,N,sizeof(struct GODZILLA),cmp);

    n=0;                                     // ostranenie godzill s rovnakym d
```

```
    for(i=0;i<N-1;++i)
        if (G[i].d != G[i+1].d)
            G[n++] = G[i];
    G[n++] = G[N-1];

    for(i=0;i<n;++i)                          // spracovavanie Godzill
        do {
            if (last > 1) {
                t1 = get_time(S[last-2],S[last-1]);
                t2 = get_time(S[last-1],i);
            }
            else if (last == 1) {
                t1 = 0;
                t2 = get_time(S[0],i);
            }
            else {
                t1 = 0;
                t2 = 0;
            }
            if (t1 <= t2) {
                S[last] = i;
                ++last;
            }
            else --last;
        } while(t1 > t2);

    for(i=0;i<N;++i) {                        // vytvaranie vystupnych poli
        from[i] = -1;
        to[i] = -1;
        X[i] = -1;
    }

    for(i=0;i<last;++i) {                     // vypocitanie casov, kedy su sily rovnake
        if (i==0) from[i] = 0;
        else from[i] = get_time(S[i-1],S[i]);
        if (i==last-1) to[i]=INF;
        else to[i] = get_time(S[i],S[i+1]);
        X[G[S[i]].id] = i;                     // vypocitanie si kde v poliach from/to
najdeme zaznam pre i-tu Godzillu
    }

    for(i=0;i<N;++i) {                        // vypis
        if (X[i] >= 0) {
            printf("%lf - ",from[X[i]]);
            if (to[X[i]]==INF) printf("*\n");
            else printf("%lf\n",to[X[i]]);
        }
        else
            printf("Toto je len godzi(te)lliatko.\n");
    }
    return 0;
}
```

opravoval Mic

9. Tradičná Godzilloballová liga

(max. 25 bodov)

Tento príklad nebol zložitý, ale napriek tomu prišlo veľmi málo riešení a k tomu iba zopár bolo správnych. Poďme si povedať, ako vyzerá také správne riešenie.

Najskôr si zavedieme označenia. Nech c_i je dĺžka chytania i-tej veľryby a p_i je dĺžka praženia i-tej veľryby. Nech

1. $s_i = 0$ ak $c_i = p_i$
2. $s_i = -1$ ak $c_i < p_i$
3. $s_i = 1$ ak $c_i > p_i$

Predstavme si, že sme si všetky veľryby usporiadali do nejakého (nie nutne do optimálneho) poradia. Nech k je taký index, že $s_k \geq 0$ a $s_{k+1} \leq 0$. To znamená, že k-ta veľryba sa dlhšie (alebo rovnako) chytá, ako praží (a teda platí, že $c_k = p_k + d_k$ pre $d_k \geq 0$) a $k+1$-vá veľryba sa dlhšie (alebo rovnako) praží ako chytá (a teda $p_{k+1} = c_{k+1} + d_{k+1}$ a $d_{k+1} \geq 0$).

Nech R je čas, ktorý treba počkať, kým sa upečie $k-1$ -vá veľryba (ako dlho sa bude „čakať" na praženie). Čo by sa stalo, ak by sme vymenili veľryby k a $k+1$? Ak chceme pražiť veľryby v poradí k, $k+1$, tak ich praženie skončí o $max(max(c_k, R)+p_k, c_k+c_{k+1})+p_{k+1} = max(max(p_k + d_k, R) + p_k, p_k + d_k + c_{k+1}) + c_{k+1} + d_{k+1} = max(max(p_k + d_k, R), d_k + c_{k+1}) + p_k + c_{k+1} + d_{k+1}$. Ak by sme veľryby pražili v poradí $k+1$, k, tak praženie skončí o $max(max(c_{k+1}, R) + p_{k+1}, c_{k+1} + c_k) + p_k) = max(max(c_{k+1}, R) + c_{k+1} + d_{k+1}, c_{k+1} + p_k + d_k) + p_k = max(max(c_{k+1}, R) + d_{k+1}, p_k + d_k) + c_{k+1} + p_k$. Chytanie veľrýb skončí v rovnakom čase (rozmyslite si prečo) a preto nás iba zaujíma ako dlho bude trvať, kým sa upraží posledná z tých dvoch veľrýb. Skôr je lepšie, lebo to môže viesť k celkovému lepšiemu času. Aby sme zistili, čo je lepšie, tak spravíme rozdiel:

$$max(max(p_k + d_k, R), d_k + c_{k+1}) + d_{k+1} - max(max(c_{k+1}, R) + d_{k+1}, p_k + d_k)$$

Poďme si rozobrať jednotlivé prípady:

- Nech $c_{k+1} \geq R$ a teda chceme vedieť aké je veľké $max(max(p_k + d_k, R), d_k + c_{k+1}) + d_{k+1} - max(c_{k+1} + d_{k+1}, p_k + d_k)$. Zoberme si posledný člen. Ak je rovný $c_{k+1} + d_{k+1}$, tak celý výraz je nezáporný, lebo prvý člen je aspoň c_{k+1}. Ak je posledný člen rovný $p_k + d_k$, tak je výraz opäť nezáporný, lebo prvý člen výrazu je aspoň $p_k + d_k$.
- Nach $c_{k+1} < R$. Chceme vedieť, aké je veľké $max(max(p_k + d_k, R), d_k + c_{k+1}) + d_{k+1} - max(R + d_{k+1}, p_k + d_k)$. Rovnakými úvahami ako v predchádzajúcom prípade dostaneme, že celý výraz je nezáporný.

Teraz sme si dokázali, že ak máme vedľa seba dve veľryby, pričom prvá má väčšie s_i, tak ak ich vymeníme, tak dostaneme lepšie poradie (alebo rovnako dobré). Preto ak zoradíme veľryby podľa s_i tak dostaneme lepšie poradie ako predtým.

Zoberme si teraz dve veľryby (k, $k+1$), pričom $s_k = s_{k+1}$. Nech $s_k = 0$ alebo $s_k = -1$. Potom je výhodnejšie usporiadať veľryby podľa času chytania v neklesajúcom poradí. Ak $s_k = 1$ tak treba veľryby usporiadať podľa času praženia v nerastúcom poradí. Dôkazy týchto tvrdení nechám na usilovného čitateľa (dôkaz je podobný, ako ten vyššie).

Teraz už máme určené jednoznačné poradie, ako treba veľryby usporiadať (najskôr sa porovnáva podľa s_i a potom podľa dĺžok chytania a pečenia). Že neexistuje lepšie poradie sa dá ukázať jednoducho, sporom. Nech čas, kedy skončíme, ak si usporiadame veľryby je A. Nech existuje lepšie poradie s časom $B < A$. Ak je usporiadané, tak musí byť rovnako dobré ako to naše, čo je spor. Ak nie je usporiadané, tak usporiadaním získame poradie s časom $C < B$. Ale dve usporiadané poradia majú rovnaký čas a teda $C = A$. Ale dostávame tam spor s tým, že $B < A$.

Zopakujme si celé riešenie. Najskôr si usporiadame veľryby tak, že najskôr pôjdu tie, ktoré sa rýchlejšie chytajú, ako pražia (a v rámci nich idú skôr tie, ktoré sa rýchlejšie chytajú), potom idú tie s totožným časom chytania a praženia (opäť usporiadané podľa času chytania)

a nakoniec idú tie, ktoré sa dlhšie pražia ako chytajú usporiadané zostupne podľa času praženia. Potom už jednoduchou simuláciou vypočítame, koľko to bude celé trvať.

Za správne riešenie aj s dôkazom dom dával 25 bodov, za nesprávne riešenie som dával najviac dva body. Tiež by som chcel upozorniť na to, že opisovanie nemáme príliš v láske a za opisovanie som patrične strhával body.

Celková časová zložitosť riešenia je $O(N \log N)$ (kvôli triedeniu) a pamäťová náročnosť je $O(N)$.

Listing programu:

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

int signum(int a){
    if(a<0)return -1;
    if(a>0)return 1;
    return 0;
}

struct cmp{
    bool operator()(const pair<int,int> &a, const pair<int,int> &b)const{
        int sa=signum(a.first-a.second),sb=signum(b.first-b.second);
        if(sa!=sb)return sa<sb;
        if(sa==-1)return a.first<b.first;
        if(sa==1)return a.second>b.second;
        return false;
    }
};

int main(){
   int N;
   cin>>N;
   vector<pair<int,int> > R(N);
   for(unsigned int i=0;i<R.size();i++){
       cin>>R[i].first>>R[i].second;
   }
   sort(R.begin(),R.end(),cmp());
   int priprava=0,pecenie=0;
   for(unsigned int i=0;i<R.size();i++){
        priprava+=R[i].first;
        pecenie=max(pecenie,priprava)+R[i].second;
   }
   cout<<pecenie<<endl;
}
```

10. Tradičná Godzillobalová liga

opravoval Dano
(max. 25 bodov)

K tomuto príkladu prišli len štyri riešenia, z ktorých jedine funkčné poslal Peťo Fulla. Pozrime sa ako také riešenie mohlo vyzerať.

Najprv si príklad trochu zjednodušme. Všimnite si, že polomer sfér vplyvu je prirodzené číslo nie väčšie ako R, a tých je len konečne veľa. Takže jedno možné riešenie je postupne vyskúšať všetky polomery, zrátať plochu a zobrať maximum. Takže, uvažujme jednoduchší problém, pri ktorom máme pevne daný polomer r sfér vplyvu. Keďže, podľa zadania, sa

prekryvy pri počítaní celkovej plochy neberú do úvahy, nám stačí len zistiť koľko najviac Godzílll vieme postaviť pri tomto polomere, vynásobiť πr^2 a máme odpoveď.

Predstavme si nasledujúci graf. Do dvoch stĺpcov si pekne pod seba nakreslíme vrcholy, kde každý jeden vrchol v pravom stĺpci prislúcha práve jednému rozšľapovačovi a každý jeden vrchol v ľavom stĺpci prislúcha práve jednému drtičovi. A ak sú dve Godzilly rôzneho druhu nebezpečne blízko pri sebe, nakreslíme hranu medzi príslušnými vrcholmi. Všimnite si, že hrany vedú iba medzi dvoma stĺpcami a žiadna hrana nezačína a nekončí v tom istom stĺpci. Takémuto grafu, ktorý sa dá rozdeliť na také dve časti, že všetky hrany vedú iba medzi týmito dvoma časťami, sa múdro hovorí *bipartitný* a jednotlivé časti sa nazývajú *partície*.

Takže, preložením do reči grafov, máme bipartitný graf a v ňom chceme nájsť množinu vrcholov takú, že žiadna hrana nevedie medzi dvoma vrcholmi v tejto množine. Odborne, takejto množine sa hovorí *nezávislá množina*. A spomedzi všetkých nezávislých množín, chceme nájsť tú najväčšiu. Zoberme si teraz ľubovoľnú nezávislú množinu S a všimnite si, že nikdy sa nemôže stať, že by oba konca jednej hrany patrili do S, lebo potom by množina nebola nezávislá. Inými slovami, aspoň jeden koniec každej hrany neleží v S. Opačne, ak máme množinu vrcholov T takú, že aspoň jeden koniec každej hrany patrí do T tak potom vrcholy ktoré nepatria do T musia tvoriť nezávislú množinu, lebo inak by existovala hrana, ktorá by nemala ani jeden koniec v T. Práve sme si ukázali, že doplnok nezávislej množiny v grafe je množina vrcholov, ktorej sa fundovane hovorí *vrcholové pokrytie*. Ak si toto všetko zhrnieme, jediné čo potrebujeme nájsť je minimálne vrcholové pokrytie v bipartitnom grafe, keďže, ako sme práve ukázali, jeho doplnok je najväčšia nezávislá množina.

A na čo je nám vlastne všetko toto dobré? No v podstate naničˇ, keby neprišiel ujo Kőnig a nepovedal nám svoju múdru vetu, že veľkosť minimálneho vrcholového pokrytia v bipartitnom grafe sa rovná veľkosti maximálneho párenia. Tí, čo vedia čo je a ako sa hľadá maximálne párenie v grafe môžu nasledujúcu sekciu s pokojom preskočiť. A tí, ktorí poznajú aj Kőnigovu vetu, môžu kľudne skočiť ešte ďalej.

Párenie[21] *Párenie* v (bipartitnom) grafe je taká množina hrán, že žiadne dve hrany v tejto množine nemajú spoločný vrchol. Inými slovami, snažíme sa popáriť vrcholy tak, aby žiaden vrchol nebol spárený s viac ako jedným vrcholom. Samozrejme, môže sa ľahko stať, napríklad ak partície nie sú rovnako veľké, že nie všetky vrcholy sú spárené. A maximálne párenie je párenie s najväčším počtom hrán. Tieto definície sú všeobecné a platia vo všeobecných grafov, my ale ďalej budeme uvažovať len grafy bipartitné[22].

Uvažujme, že máme nejaké, nie nutne optimálne párenie v bipartitnom grafe a ofarbíme si hrany v párení na modro a zvyšné na červeno. *Zlepšujúca cesta* je taká cesta v grafe, ktorá začína v nespárenom vrchole v ľavej partícii a postupne pokračuje striedavo po červených a modrých hranách až kým neskončí v nejakom nespárenom vrchole v pravej partícii. Všimnite si, že keďže zlepšujúca cesta začína vľavo a končí vpravo, počet prechodov doprava a následne červených hrán musí byť o jedna viac ako počet modrých hrán ležiacich na tejto zlepšujúcej ceste. Taktiež, prefarbením modrých hrán na červené a červených na modré na zlepšujúcej ceste dostaneme znovu párenie. A teda, ak vieme v párení nájsť zlepšujúcu cestu, vieme, iba prefrabením hrán na nej, nájsť párenie o jedna väčšie. Takže, ak párenie obsahuje zlepšujúcu cestu, nemôže byť maximálne. Ale čo o opaku tohto tvrdenia? Je pravda, že ak párenie nie je maximálne tak potom nutne obsahuje zlepšujúcu cestu?

Odpoveď je áno. Označme naše párenie M a uvažujme, že existuje aj väčšie párenie N. Teda $|N| > |M|$. A nech P je množina tých hrán, ktoré patria do M alebo N, ale nie do oboch. Múdrymi slovami na oblbovanie kamarátov, nech P je symetrická diferencia N a M. Keďže aj M aj N sú párenia, každý vrchol susedí s najviac jednou hranou z M a jednou z N a teda, s najviac dvoma hranami z P. Ako vyzerá taký graf, kde stupeň každého vrchola je najviac

[21] Bontóm mi nedovolí nespomenúť, že niektorí ľudia mu hovoria aj párovanie.

[22] Ambicióznym riešiteľom je odporúčané vygúgliť si dačo o všeobecnom párení.

2? Nutne, a tu sa treba trochu zamyslieť, takýto graf sa skladá z izolavaných vrcholov, cyklov a jednoduchých ciest bez cyklov. Na každom cykle a každej ceste, sa musia striedať hrany z M a hrany z N, lebo inak by niektorý vrchol susedil s dvomi hranami patriacim do toho istého párenia. Takže, všetky cykly v P musia byť párnej dĺžky a obsahovať rovnaký počet hrán z oboch párení. Odmyslime si teda cykly a všimnime si, že keďže my stále uvažujeme, že N je väčšie ako M, musí existovať aspoň jedna cesta C, ktorá má o jednu hranu z N viac ako z M. A keďže my vieme, že hrany z N a M sa striedajú, prvá aj posledná hrana na C musí byť z N. A tu už je ľahké vidieť, že C je zlepšujúca cesta pre M, lebo jeden jej koniec musí ležať vľavo, druhý vpravo, oba sú nespárené, lebo inak by konce neboli koncami, a taktiež striedajú sa na nej hrany z M a nie z M.

Toto pozorovanie nás vedie, k štandardnému algoritmu na hľadanie maximálneho párenia v grafoch a to, začnime s prázdnym párením a postupne hľadajme zlepšujúcu cestu prehľadávaním do šírky alebo hĺbky, prehoďme hrany na tejto ceste a ak už žiadnu zlepšujúcu cestu nevieme nájsť, vieme, že sme našli párenie maximálne a môžme skončiť. Neskôr v tomto príklade budeme potrebovať nájsť zlepšujúcu cestu len na jedno prehľadávanie, takže namiesto lenivého postupu spustiť prehľadávanie do hĺbky z každého jedného nespáreného vrchola v ľavej partícií, spustíme prehľadávanie akoby naraz zo všetkých nespárených vrcholov.

Kőnigova veta Chceme ukázať, že veľkosť maximálneho párenia M sa rovná veľkosti najmenšieho vrcholového pokrytia. Najprv si všimnite, že vrcholové pokrytie musí pokrývať všetky hrany v M. A teda, keďže M je párenie, veľkosť ľubovoľného pokrytia musí byť aspoň $|M|$ Takže nám stačí ukázať, že vieme zostrojiť pokrytie o veľkosti $|M|$. Od teraz, ceste v bipartitnom grafe budeme hovoriť *striedavá*, ak začína v nejakom nespárenom vrchole v ľavej partícii a všetky hrany z ľava do prava nie sú v M a všetky hrany z pravej partície do ľavej sú v M. Inými slovami, striedavá cesta je „zlepšujúca cesta“, ktorá nemusí končiť v spárenom vrchole.

Pre každú hranu v M, zoberme vrchol v pravej partícií, ak nejaká striedavá cesta končí v tom vrchole, a vrchol v ľavej partícií v ostatných prípadoch. Označme množinu týchto vrcholov S. Zjavne, $|S| = |M|$. Teraz si ukážeme, že S je vrcholové pokrytie. Zoberme ľubovoľnú hranu H v našom grafe. Ak H patrí do M, tak z konštrukcie S plynie, že (práve) jeden jej koniec je v S. Ak H nie je v M, tak označme jej ľavý vrchol a a pravý vrchol b. Teraz, M musí nutne obsahovať hranu (c, d), kde c je v ľavej a d v pravej partícii, takú, že $a = c$ alebo $b = d$, lebo inak $a \to b$ by bola zlepšujúca cesta a M by nebolo maximálne. Môže nám nastať pár situácií, ak $b = d$, potom $a \to d = b$ je striedavá cesta, keďže $H \notin M \Rightarrow a$ je nespárený vrchol, a teda $b = d \in S$. Na druhú stranu, ak $a = c$ a c patrí do S, potom, zjavne, a patrí do S. A teda, posledný prípad, ktorý musíme rozobrať je $a = c$ a c nie je v S. Potom d, opačný koniec páriacej hrany, musí patriť do S a teda, musí existovať striedavá cesta C končiaca v d. Ak a neleží na C, potom $C \to a \to b$ je striedavá cesta končiaca v b. Ak a leží na C, tak potom cesta, ktorá začína ako C, ale v a sa odpojí a pokračuje do b je striedavá cesta končiaca v b. B musí byť spárené, lebo inak striedavá cesta, ktorú sme práve zostrojili, by bola v skutočnosti zlepšujúca, ale my vieme, že to sa nemôže stať keďže M je maximálne párenie. Takže b je spárený vrchol taký, že nejaká striedavá cesta v ňom končí. A teda, z definície S, b patrí do S.

A na záver k riešeniu Majúc po ruke všetku ťažkú mašinériu z posledných pár odstavcov, riešenie je už celkom jednoduché. Pre každý možný polomer r sféry vplyvu si zostrojíme prislúchajúci bipartitný graf. V grafe nájdeme maximálne párenie P a z Kőnigovej vety vieme, že veľkosť maximálnej nezávislej množiny a teda maximálny počet Godzíll, ktoré môžme postaviť, je $M + N - |P|$. Tento počet vynásobíme πr^2 a máme maximálnu plochu pri danom polomere. Nakoniec, už len stačí zobrať maximum z plôch a sme hotoví.

Užitočná finta číslo jedna je si všimnúť, že postupným zväčšovaním polomeru sfér, hrany do grafu iba pribúdajú. Preto, vždy ako zväčšíme polomer, nemusíme hľadať párenie od začiatku, ale môžme pokračovať už vo vypočítanom párení. Užitočná finta číslo dva je si

všimnúť, že polomery, pri ktorých pridávame hrany do grafu a teda sa nám aj môže zmeniť párenie, sú polomery také, pri ktorých sa nejaké dve Godzilly rôznych typov dostanú nebezpečne blízko. A týchto je najviac len NM. Takže nám v skutočnosti stačí skúšať len NM rôznych polomerov zodpovedajúcim Godzillám nebezpečne blízko. Dokonca, všimnite si, že toto riešenie by fungovalo aj ak by veľkosť polomerov mohlo byť ľubovoľné nezáporné reálne číslo. Práve že, v tomto prípade je diskrétne riešenie o čosi zložitejšie, lebo musíme správne ošetriť prípady ako polomer$= 9.0, 9.4, 10.0$. Užitočná finta číslo tri je si všimnúť, že po pridaní jednej hrany do grafu sa môže maximálne párenie zväčšiť najviac o jedna.

Zhrnúc naše riešenie, na porátanie a utriedenie vzdialeností medzi Godzillami potrebujeme $O(NM \log NM)$ času. Následne, postupne po jednej v neklesajúcom poradí pridáme do grafu všetky hrany kratšie ako $R/2$. Týchto je najviac NM a po pridaní každej hrany spustíme jedno prehľadávanie z nespárených vrcholov. V najhoršom prípade prehľadáme celý graf, ktorý môže mať až NM hrán. Takže jedno prehľadanie je v $O(N+M+NM)$. A teda celková časová zložitosť je $O((NM)^2)$ a keďže si musíme pamätať celý graf, pamäťová zložitosť je $O(NM)$.

Listing programu:

```
#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>
#include <vector>
#include <utility>
#include <queue>

using namespace std;

int n, m;

int na_druhu(int x){return x*x;}

struct Hrana{
  int dlzka, vlavo, vpravo;
};

bool porovnaj(const Hrana &a, const Hrana &b){
  return a.dlzka < b.dlzka;
}

vector<vector<int> > graf;
vector<int> sparovany_vlavo, sparovany_vpravo;
vector<int> predosly;

bool zlepsujuca_cesta(){
  for(int i=0; i<m; i++) predosly[i] = -1;
  vector<int> bol_vlavo(n,false);
  queue<int> fronta;
  for(int i=0; i<n; i++)
    if (sparovany_vlavo[i] == -1){
      bol_vlavo[i] = true;
      fronta.push(i);
    }

  int nasiel = -1;
  while (!fronta.empty() && nasiel == -1){
    int vrchol = fronta.front(); fronta.pop();
    for(int i=0; i<(int)graf[vrchol].size(); i++){
```

```
      int kam = graf[vrchol][i];
      if(predosly[kam]!=-1) continue;
      predosly[kam] = vrchol;
      if (sparovany_vpravo[kam]==-1){
        nasiel = kam;
        break;
      }
      if(!bol_vlavo[sparovany_vpravo[kam]]){
        bol_vlavo[sparovany_vpravo[kam]] = true;
        fronta.push(sparovany_vpravo[kam]);
      }
    }
  }
  if(nasiel == -1) return false;

  int vrchol = nasiel;
  while(vrchol != -1){
    int kam = predosly[vrchol];
    sparovany_vpravo[vrchol] = kam;
    int dalsi = sparovany_vlavo[kam];
    sparovany_vlavo[kam] = vrchol;
    vrchol = dalsi;
  }

  return true;
}


int main(){
  int r;
  scanf("%d", &n);
  vector<int> drtic_x(n),drtic_y(n);
  for(int i=0; i<n; i++)
    scanf("%d %d", &drtic_x[i], &drtic_y[i]);
  scanf("%d", &m);
  vector<int> rozslapovac_x(n), rozslapovac_y(n);
  for(int i=0; i<m; i++)
    scanf("%d %d", &rozslapovac_x[i], &rozslapovac_y[i]);
  scanf("%d", &r);

  graf.resize(n);
  sparovany_vlavo.resize(n, -1);
  sparovany_vpravo.resize(m, -1);
  predosly.resize(m);

  vector<Hrana> hrany;
  for(int i=0; i<n; i++)
    for(int j=0; j<m; j++){
      int dlzka = na_druhu(drtic_x[i] - rozslapovac_x[j])+na_druhu(drtic_y[i] -
rozslapovac_y[j]);
      if (dlzka <= r * r * 4){
        Hrana hrana;
        hrana.dlzka = (int)(sqrt(dlzka)/2.0);
        hrana.vlavo = i;
        hrana.vpravo = j;
        hrany.push_back(hrana);
      }
    }

  sort(hrany.begin(), hrany.end(), porovnaj);
```

```
  int index = 0, parovanie = 0, najvacsia_plocha = 0;
  while(index < (int)hrany.size()){
    int povodna_dlzka = hrany[index].dlzka;
    int plocha = (n + m - parovanie) * na_druhu(povodna_dlzka);

    najvacsia_plocha = max(najvacsia_plocha, plocha);
    while(index < (int)hrany.size() && hrany[index].dlzka == povodna_dlzka){
      graf[hrany[index].vlavo].push_back(hrany[index].vpravo);
      index++;
      if(zlepsujuca_cesta()) parovanie++;
    }
  }

  najvacsia_plocha = max(najvacsia_plocha, (n + m - parovanie) * na_druhu(r));
  printf("%.9lf\n",3.1415926535897 * najvacsia_plocha);
  return 0;
}
```

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
1	Hozza Ján	Gym. Jura Hronca BA	2	10	10	10	11	15	56
2	Balog Matej	Gym. Grösslingová BA	2	10	8	10	11	11	50
3	Kekely Michal	Gym. Varšavská Žilina - Vlčince	2	10	9	3	12	11	45
4	Baláž Martin	Gym. Alejová Košice	4		9	9	15	11	44
5	Hruška Eugen	Gym. Hlohovec	3	6	8	5	12	11	42
6	Bosák Radomír	Gym. Grösslingová BA	4	10	10	10		11	41
7	Korbaš Rafael	Gym. Hronská BA	2	10	6		14	9	39
7	Sládek Filip	Gym. Námestovo	3	10	8	10		11	39
7	Vavrík Boris	Gym. Jura Hronca BA	2	10	8		11	10	39
10	Kuzma Tomáš	Gym. Alejová Košice	4	5	4	10	8	11	38
11	Kieferová Mária	Gym. sv. Františka Žilina	4	10		5	11	11	37
12	Kuchyňár Martin	Gym. Jura Hronca BA	3	10	8	8		10	36
13	Bubnár Michal	Gym. sv. Františka z Assisi V.n. Topľou	4	10	4	8	3	10	35
13	Sebechlebský Ján	Gym. Žiar nad Hronom	2	10	8	4	6	7	35
15	Kunec Sebastián	Gym. Konštantínova Prešov	4	10	3		10	10	33
15	Varga Mátyás	Gym. H. Selyeho Komárno	2	10	8	4		11	33
15	Šmihla Ján	ZŠ Saratovská	0	8	8	6	0	11	33
18	Jurových Jakub	Gym. Okružná Zvolen	2	9	5	6	11		31
19	Dresslerova Anna	Gym. Jura Hronca BA	2	7	9		12		28
20	Phuong Mariana	Gym. Jura Hronca BA	2	10	9	8			27
21	Macháč Juraj	Gym. Jura Hronca BA	2	9	6		10		25
22	Cuc Bruno	Gym. Grösslingová BA	3	10	0	7	0	7	24
22	Majdiš Mojmír	Gym. Dolný Kubín	3	10	9	5			24
22	Špano Marek	Gym. Jura Hronca BA	2	10				14	24
25	Táňa Tóthová	Gym. Jura Hronca BA	3	9	5			8	22
26	Jankovič Radovan	Gymnázium Levice	2	9	8	4	0		21
27	Hozzánková Hana	Gym. Jura Hronca BA	3	10				10	20
27	Kučera Martin	Gym. Golianova Nitra	2	10	6	4			20
27	Macko Lukáš	Gym. Š. Moyzesa B. Bystrica	3	9	7	4			20
30	Stančiaková Katarína	Gym. Jura Hronca BA	3	9	7			3	19
31	Anderle Michal	Gym. Haličská Lučenec	?	6	3			9	18
31	Šimko Stanislav	Gym. Jura Hronca BA	3	5	5	4		4	18
33	Orenič Jozef	Gym. Slovenská Bardejov	3	7	3	2		5	17
34	Bögi Tamás	SPŠ Komárno	2	8	8				16
34	Mariš Andrej	Gym. Piaristická Nitra	1	8	3	1	2	2	16
34	Pinter Martin	Gym. Jura Hronca BA	2	10		6			16
34	Plank Martin	Gym. A. Merici Trnava	4	9	7				16
34	Páleník Martin	Gymnázium Levice	4	10	6				16
39	Macháč Matej	Gym. P. de Coubertina Piešťany	3	6	0			8	14
39	Szabó Peter	SPŠE Nové Zámky	4	7				7	14
39	Tulala Peter	Gym. A. Merici Trnava	3	9	5				14
42	Plavák Dušan	Gym. Trstená	2	9			2		11
43	Chomjak Maros	Gym. Alejová Košice	3	10					10
43	Novella Tomáš	Gym. Alejová Košice	3	10					10
43	Piovarči Michal	Gym. Hronská BA	4		6	4			10
43	Štajer Andrej	Gym. Dolný Kubín	3	10					10
47	Dzurilla Bohus	Gym. Alejová Košice	3	9					9
47	Iring Andrej	Gym. Jura Hronca BA	2	9					9
47	Kentoš Michal	Gym. sv. Františka z Assisi V.n. Topľou	4	9					9
47	Kovaľ Anton	Gym. L. Stöckela Bardejov	3	9					9

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
47	Pivovarči Michal	Gym. Hronská BA	4	9					9
47	Toth Robert	Gym. Alejová Košice	3	9					9
47	Uchnár Matúš	Gym. Alejová Košice	3	9					9
47	Zatrochová Zuzana	Gym. Alejová Košice	3	9					9
55	Holas Juraj	Gym. Jura Hronca BA	2	8					8
55	Nagy Štefan	SPŠ Komárno	3	5	3				8
57	Mudrik Richard	Gym. J. M. Hurbana Čadca	4	7					7
58	Porubsky Martin	Gym. sv. Cyrila a Metoda Nitra	3	6					6
59	Cudrák Miloš	Gym. J. M. Hurbana Čadca	4	5					5
59	Kmec Peter	Gym. Konštantínova Prešov	2	5					5
61	Anderko Maroš	Gym. Konštantínova Prešov	3	1				3	4
61	Aštary Matej	Gym. Konštantínova Prešov	3	1				3	4
61	Fotta Michal	Gymnázium Považská Bystrica	3	1				3	4
61	Nguyen Tien Phong	Gym. Konštantínova Prešov	3	1				3	4
61	Trebichavský Richard	Gym. Jura Hronca BA	1		4				4
66	— Anonym	Gym. Jura Hronca BA	?						0
66	Bogárová Zuzana	Gym. Ľudovíta Štúra Trenčín	2	:)					0

Výsledková listina po 1. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
1	Šrámek Martin	Gym. Alejová Košice	4	12	11	20	20	25	88
2	Baláž Martin	Gym. Alejová Košice	4	15	11	18	20	23	87
3	Kuzma Tomáš	Gym. Alejová Košice	4	8	11	20	20	24	83
4	Herencsár Albert	Gym. maď. Galanta	4	12	11	20	20	17	80
5	Hudec Tobiáš	Gym. Partizánske	3	13	11	20	10	25	79
6	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	4	12	11	20	19	15	77
7	Petrucha Michal	Gym. Metodova BA	4		11	20	20	25	76
8	Fulla Peter	SPŠ strojnícka Spišská Nová Ves	4	12	15		20	25	72
9	Csiba Peter	Škola pre mim. nadané deti BA	4		11	16	18	16	61
10	Belan Tomáš	Škola pre mim. nadané deti BA	3	12	10	12	8	16	58
11	Proksa Ondrej	Gym. Párovská Nitra	4	10	11	11	8	17	57
12	Špánik Marián	Gym. sv. Františka Žilina	4	11	11		6	16	44
13	Rohár Pavol	Gym. M. R. Štefánika, Košice	3	4	14	9	8	5	40
14	Špano Marek	Gym. Jura Hronca BA	2		14	20			34
15	Hojčka Michal	Gym. Partizánske	4		11	7	8	7	33
15	Hruska Eugen	Gym. Hlohovec	3	12	11		6	4	33
15	Sládek Filip	Gym. Námestovo	3		11	13		9	33
18	Ziman Michal	Gym. Haličská Lučenec	3		11	20			31
19	Hozza Ján	Gym. Jura Hronca BA	2	11	15				26
20	Hagara Michal	Gym. Jura Hronca BA	3	12	13				25
20	Jursa Jakub	Gym. Alejová Košice	4	14	11				25
22	Macháč Juraj	Gym. Jura Hronca BA	2	10			8	6	24
22	Miklovič Tomáš	Gym. Nové Zámky	3	5	5			14	24
24	Kekely Michal	Gym. Varšavská Žilina - Vlčince	2	12	11				23
24	Korbaš Rafael	Gym. Hronská BA	2	14	9				23
26	Balog Matej	Gym. Grösslingová BA	2	11	11				22
26	Jankovič Radovan	Gymnázium Levice	2	0		7	7	8	22
26	Kieferová Mária	Gym. sv. Františka Žilina	4	11	11				22
29	Bubnár Michal	Gym. sv. Františka z Assisi V.n. Topľou	4	3	10		8		21
29	Iring Andrej	Gym. Jura Hronca BA	2			7		14	21
29	Kikta Michal	Gym. Tatarku Poprad	4	14			7		21
29	Kunec Sebastián	Gym. Konštantínova Prešov	4	10	10	0	1	0	21
29	Vavrík Boris	Gym. Jura Hronca BA	2	11	10				21
34	Hozzánková Hana	Gym. Jura Hronca BA	3		10		7	3	20
35	Stančiaková Katarína	Gym. Jura Hronca BA	3		3		8	4	15
36	Macháč Matej	Gym. P. de Coubertina Piešťany	3		8		6		14
37	Sebechlebský Ján	Gym. Žiar nad Hronom	2	6	7				13
38	Dresslerova Anna	Gym. Jura Hronca BA	2	12					12
38	Phuong Mariana	Gym. Jura Hronca BA	2			12			12
38	Piovarči Michal	Gym. Hronská BA	4				8	4	12
38	Táňa Tóthová	Gym. Jura Hronca BA	3		8			4	12
42	Bosák Radomír	Gym. Grösslingová BA	4		11				11
42	Jurových Jakub	Gym. Okružná Zvolen	2	11					11
42	Varga Mátyás	Gym. H. Selyeho Komárno	2		11				11
42	Šmihla Ján	ZŠ Saratovská	0	0	11				11
46	Kuchyňár Martin	Gym. Jura Hronca BA	3		10				10
47	Anderle Michal	Gym. Haličská Lučenec	?		9				9
47	Šimko Stanislav	Gym. Jura Hronca BA	3		4		5		9
49	Cuc Bruno	Gym. Grösslingová BA	3	0	7				7
49	Szabó Peter	SPŠE Nové Zámky	4		7				7

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
51	Tulala Peter	Gym. A. Merici Trnava	3				6		6
52	Orenič Jozef	Gym. Slovenská Bardejov	3		5				5
53	Anderko Maroš	Gym. Konštantínova Prešov	3		3		1	0	4
53	Aštary Matej	Gym. Konštantínova Prešov	3		3		1		4
53	Fotta Michal	Gymnázium Považská Bystrica	3		3		1	0	4
53	Mariš Andrej	Gym. Piaristická Nitra	1	2	2				4
53	Nguyen Tien Phong	Gym. Konštantínova Prešov	3		3		1	0	4
58	Plavák Dušan	Gym. Trstená	2	2					2

Výsledková listina po 1. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda	8	9	10	Σ
1	Fulla Peter	SPŠ strojnícka Spišská Nová Ves	4	25	25	20	70
2	Hudec Tobiáš	Gym. Partizánske	3	25	25	2	52
3	Petrucha Michal	Gym. Metodova BA	4	25			25
3	Šrámek Martin	Gym. Alejová Košice	4	25			25
5	Kuzma Tomáš	Gym. Alejová Košice	4	24			24
6	Baláž Martin	Gym. Alejová Košice	4	23			23
7	Herencsár Albert	Gym. maď. Galanta	4	17			17
7	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	4	15	2		17
7	Proksa Ondrej	Gym. Párovská Nitra	4	17			17
10	Belan Tomáš	Škola pre mim. nadané deti BA	3	16			16
10	Csiba Peter	Škola pre mim. nadané deti BA	4	16			16
10	Špánik Marián	Gym. sv. Františka Žilina	4	16			16
13	Iring Andrej	Gym. Jura Hronca BA	2	14			14
13	Miklovič Tomáš	Gym. Nové Zámky	3	14			14
15	Jankovič Radovan	Gymnázium Levice	2	8	2	2	12
16	Sládek Filip	Gym. Námestovo	3	9			9
17	Hruska Eugen	Gym. Hlohovec	3	4	2	2	8
18	Hojčka Michal	Gym. Partizánske	4	7			7
19	Macháč Juraj	Gym. Jura Hronca BA	2	6			6
19	Táňa Tóthová	Gym. Jura Hronca BA	3	4	2		6
21	Rohár Pavol	Gym. M. R. Štefánika, Košice	3	5			5
22	Piovarči Michal	Gym. Hronská BA	4	4			4
22	Stančiaková Katarína	Gym. Jura Hronca BA	3	4			4
24	Hozzánková Hana	Gym. Jura Hronca BA	3	3			3
25	Anderko Maroš	Gym. Konštantínova Prešov	3	0	0		0
25	Aštary Matej	Gym. Konštantínova Prešov	3		0		0
25	Fotta Michal	Gymnázium Považská Bystrica	3	0	0		0
25	Kmec Peter	Gym. Konštantínova Prešov	2		0		0
25	Kunec Sebastián	Gym. Konštantínova Prešov	4	0	0		0
25	Nguyen Tien Phong	Gym. Konštantínova Prešov	3	0	0		0