Korešpondenčný seminár z programovania
XX. ročník, 2002/2003
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 1. kola zimnej časti

Milí riešitelia!

Prší, prší len sa leje... Jesenné sústredenie za nami, do Vianoc ďaleko... Hoci už bolo Martina, ešte veľa snehu nie je, zato je pľušť jak sviňa. Tak nechoďte radšej von, aby ste neprechladli. Pekne sa zababušte do deky, prečítajte si vzorové riešenia a čakajte na sneh a Vianoce ako húska na nôž.

V nádržke už voda šumí, mydielkom si ruky umy.

KSPáci

O ideálnom riešení

Na úvod by sme vám radi ešte raz pripomenuli, ako si predstavujeme ideálne riešenie.

Nechceme, aby ste sa zbytočne hrali so vstupom a výstupom. Samozrejme, zakazovať vám to nebudeme, ale bodíky navyše za to nedostanete a ešte môžete znechutiť opravovateľa, ktorý musí čítať množstvo kódu nesúvisiaceho s riešením samotnej úlohy. Vo svojom riešení sa nemusíte zaoberať kontrolovaním správnosti vstupných údajov.

Pod pojmom **popis algoritmu** *nerozumieme* preloženie programu z Pascalu, prípadne C-čka do slovenčiny. Popis by mal zhruba obsahovať: **Popis myšlienky**, na ktorej je založený algoritmus (bez detailov ako sú názvy premenných, procedúr...). **Popis dátových štruktúr**. Čo si vlastne chceme počas výpočtu pamätať a aké štruktúry sú na to vhodné. **Popis algoritmu**. Tu môžeme využívať už popísané dátové štruktúry a spomenúť dôležité procedúry a funkcie, ako aj najdôležitejšie premenné. **Zdôvodnenie správnosti**. Nemusí to byť formálne presný ani detailný dôkaz, avšak mal by obsahovať argumenty zdôvodňujúce každý dôležitý aspekt programu, ktorý nie je na prvý pohľad zrejmý. Sem zaradíme aj dôkaz konečnosti v tých prípadoch, keď nie je zrejmé, že sa program nezacyklí. Na koniec príde **odhad** časovej a pamäťovej zložitosti programu.

Takáto štruktúra popisu nie je univerzálna. Niektoré časti možno vynechať, niektoré zlúčiť, prípadne nejaké pridať. Poradie by však malo byť zhruba zachované. Popis by sa v žiadnom prípade nemal detailne venovať spracovávaniu vstupu, ani výpisu výstupu.

Niekoľko slov o odhade zložitosti. Pod odhadom časovej zložitosti rozumieme odhad času, ktorý bude program trvať, v závislosti od vstupných premenných. Tento čas je priamo úmerný počtu elementárnych operácií, ktoré program vykonáva – priradenie jednoduchej premennej, porovnanie jednoduchých premenných, aritmetické operácie, atď. Väčšinou sa zaujímame o to, ako dlho beží program v priemernom, alebo najhoršom prípade (čiže si všímame priemerný, alebo maximálny čas vykonávania programu). Analogicky odhad pamäťovej zložitosti je odhad veľkosti použitej pamäti v závislosti od vstupných premenných.

Nezaujímajú nás presné hodnoty (ktoré je hlavne u časovej zložitosti navyše aj problém určiť), iba ich rádová veľkosť. Za týmto účelom bola zavedená tzv. O-notácia. Hovoríme, že funkcia $f(n)$ je $O(g(n))$, ak funkciu $f(n)$ vieme zhora odhadnúť nejakým násobkom funkcie $g(n)$.[1] Túto notáciu budeme používať na jednoduchý zápis zložitosti.

[1] Keby sme chceli byť formálni: $f(n) = O(g(n)) \Leftrightarrow \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+; \forall n > n_0; f(n) \leq c \cdot g(n)$

Napríklad ak program pre vstup veľkosti n nebeží nikdy viac ako $T(n) = 0.5n^3+5n \log n+$ 83 mikrosekúnd, vieme čas jeho behu zhora odhadnúť vhodným násobkom funkcie n^3. V takomto prípade hovoríme, že časová zložitosť nášho programu je $O(n^3)$. (Tento zápis môžeme tiež čítať nasledovne: Čas behu nášho programu je **(v najhoršom prípade) rádovo** n^3.) Všimnite si, že ak je nejaká funkcia $O(n^2)$, tak je aj $O(n^3)$, atď. Vždy sa však snažíme nájsť čo najmenšie horné ohraničenie, t.j. funkciu, ktorá čo najpomalšie rastie.

Takto odhadnúť časovú a pamäťovú zložitosť programu nie je väčšinou nič ťažké. Ak napr. obsahuje program dva vnorené cykly, z ktorých sa každý vykonáva rádovo n-krát, bude jeho časová zložitosť $O(n^2)$. Zložitosti $O(n)$ zvykneme hovoriť lineárna, $O(n^2)$ kvadratická a pod. Konštantnú (časovú alebo pamäťovú) zložitosť môžeme značiť $O(1)$. Viacero príkladov odhadu časovej i pamäťovej zložitosti nájdete, ak si pozorne prečítate tieto vzorové riešenia.

A ešte jedno upozornenie pre **vás** – riešiteľov: uvedením nepostačujúceho popisu prípadne jeho vynechaním riskujete, že opravovateľ vaše riešenie nepochopí a dostanete napr. 0 (slovom **nula**) bodov.

Opravoval Goober
(max. 13 bodov)

1. O telocviku II

Tento príklad ma veľmi potešil, lebo takmer všetky riešenia čo prišli boli správne. Najčastejšie sa vyskytovali dva druhy – také, ktoré úlohu previedli do reči grafov a také, ktoré ju chápali ako problém permutácií. Vzorové riešenie bude patriť do druhej skupiny.

A ako vyzerá? Predstavme si, že by chlapci vedeli, aké má byť ich výsledné poradie. Pre zjednodušenie si očíslujme miesta aj chlapcov číslami 1 až N tak, že na konci má chlapec i stáť na mieste i. Každé usporiadanie čísel 1 až N (chlapcov) nazveme *permutáciou*. Počiatočnú permutáciu označíme p. Vezmime si ľubovoľného chlapca x a všimnime si postupnosť $x, p[x], p[p[x]], \ldots$ Keď sú v tejto postupnosti čísla x, y bezprostredne za sebou, znamená to, že y zavadzia x-ovi (t.j. y je na mieste, kde má byť x). Zjavne, táto postupnosť sa raz začne opakovať a to dokonca tak, že sa v nej zopakuje číslo x. (To preto, že na každé miesto chce ísť práve 1 chlapec. Keby sa ako prvý zopakoval niekto iný, všimnime si chlapcov, ktorí sú v tejto postupnosti tesne pred týmito opakovaniami. Sú rôzni, lebo sme vzali prvého, čo sa opakuje. Zároveň ale obaja chcú ísť na jeho miesto. To je ale spor.)

Takto sa nám chlapci rozdelia do niekoľkých *cyklov*, v ktorých si jednotliví chlapci zavadzajú dookola. Je asi jasné, že každá permutácia sa dá jednoznačne rozložiť na cykly (napr. vyššie popísaným spôsobom) a že každý chlapec bude patriť do práve jedného cyklu. Celkový počet cyklov permutácie, v ktorej sa nachádzajú na začiatku, označíme C.

Tým chlapcom, ktorí sú na správnych miestach, zodpovedajú cykly s jediným prvkom, ostatným dlhšie. Všimnime si, čo spôsobí výmena dvoch chlapcov. Sú dve možnosti – buď vymeníme dvoch chlapcov z toho istého cyklu, čím sa tento cyklus rozbije na dva menšie, alebo vymeníme dvoch chlapcov z rôznych cyklov, čím sa počet cyklov naopak o 1 zníži.[2]

Každá výmena teda zvýši počet cyklov najviac o 1. Permutácia v ktorej začínajú má C cyklov, koncová ich má mať N. To ale znamená, že určite potrebujeme aspoň $N - C$ výmen. Zároveň platí, že toľkoto výmen nám vždy bude stačiť – stačí vždy vymeniť dvoch chlapcov, ktorí sú spolu v cykle.

Zostáva už len prísť na to, ako si chlapcov vlastne očíslovať. Na to potrebujeme zistiť ich výsledné poradie. Lenže to nie je nič iné ako staré známe triedenie. Stačí preto použiť ľubovoľný triediaci algoritmus (samozrejme, chceme čo najrýchlejší, preto použijeme QuickSort alebo HeapSort). Vzorový program používa HeapSort, lebo ten je $O(N \log N)$ v najhoršom prípade, zatiaľ čo štandardný QuickSort je $O(N \log N)$ „len“ v priemernom prípade.

Implementácia? Tá je asi jasná – najprv zistíme správne usporiadanie a potom pohľadáme prvého, ktorý nie je na správnom mieste. Pokiaľ taký existuje, prehodíme ho na jeho správne

[2] Premyslite si, prečo je to tak.

miesto (čím sa na jeho pôvodné miesto dostal niekto iný) a postup opakujeme. Keďže chlapcov prechádzame v poradí podľa čísel, nemusíme sa pri hľadaní nesprávne stojaceho vracať späť, a teda časová zložitosť druhej časti je $O(N)$, čo sa stratí popri zložitosti triedenia.

Iný (rovnako dobrý) prístup je rozdeliť si permutáciu na cykly a potom postupne povymieňať chlapcov v každom cykle. Náš program vlastne robí to isté, len naraz pre všetky cykly.

No a už len bodovanie: nesprávne riešenia dostali symbolické 3 body, správne riešenia v čase $O(N \log N)$ dostali plných 13 bodov, za kvadratické riešenie bolo 9 bodov. Pár bodov mohlo ísť dolu za nedostatočný popis, chýbajúce zdvôvodnenie správnosti alebo chýbajúce odhady časovej a pamäťovej náročnosti.

Listing programu:

```
const MAX=100;
var vyska:array[1..MAX] of integer;
    spravne:array[1..MAX] of integer;
    n,i:integer;

procedure vymen(i,j:integer);
 var t:integer;
 begin t:=spravne[i]; spravne[i]:=spravne[j]; spravne[j]:=t; end;

procedure Halduj(i,n:integer);
var j,t:integer;
begin
  j:=2*i;
  while j<=n do begin
    if j+1<=n then if vyska[spravne[j+1]]<vyska[spravne[j]] then inc(j);
    if vyska[spravne[j]]<vyska[spravne[i]] then vymen(i,j) else break;
    i:=j; j:=2*j;
  end;
end;

begin
  assign(input,'input.txt'); reset(input);
{ Nacitame }
  write('Pocet ziakov: '); readln(n);
  for i:=1 to n do
    begin write('Vyska ziaka #',i,': '); read(vyska[i]); spravne[i]:=i; end;
{ Utriedime }
  for i:=n div 2 downto 1 do Halduj(i,n);
  for i:=n downto 2 do begin vymen(1,i); halduj(1,i-1); end;
{ A uz len prejdeme }
  i:=1;
  while i<=n do if spravne[i]=i then inc(i) else
  begin
   writeln('Vysky ',vyska[i],' a ',vyska[spravne[i]],' vymente sa !');
   vymen(i,spravne[i]);
  end;
end.
```

2. O metre

Opravoval WSX
(max. 13 bodov)

Napriek tomu, že táto úloha bola pomerne ľahká, mnohí z vás zabudli na všelijaké okrajové prípady. Všetky vaše riešenia mali časovú zložitosť $O(N)$. Niektorým z vás stačila pamäť $O(1)$. Takéto riešenia, samozrejme pokiaľ boli funkčné, dostali 13 bodov. Algoritmy používajúce $\Theta(N)$ pamäte dostali o bod menej, čiže 12 bodov. Ale aké by to bolo hodnotenie,

keby všetci mali 12 alebo 13 bodov? Body som stŕhal za rôzne chyby podľa ich serióznosti. Napríklad ak váš program našiel všetky údaje potrebné k vypísaniu rovnice priamky, ale nezvládol ju vypočítať korektne, dostali ste o 1 bod menej. Podstatnejšia chyba je ak váš program nefungoval pre interval uhlov obsahujúci nulový uhol – za túto chybu som strhával až 6 bodov.

Vzorové riešenie: Predpokladajme, že už máme také a, b, c, že priamka s rovnicou $ax+by+c=0$ je riešením úlohy. Potom aj priamka $ax+by=0$ je jej riešením. Prečo? Táto nová priamka je rovnobežná s pôvodnou priamkou, teda poradie piät kolmíc na ňu sa nezmení. Preto ak bolo poradie pôvodných piät kolmíc v správnom poradí, bude to platiť aj pre túto priamku. Určite všetci viete, že priamka tvaru $ax+by=0$ prechádza cez bod $[0,0]$. Naviac, nič nám nepokazí, ak si túto priamku orientujeme, t.j. chápeme ju ako jej smerový vektor. (Smer nám vlastne hovorí, od ktorého konca začneme pozerať päty kolmíc.) Pod uhlom priamky budeme rozumieť orientovaný uhol, ktorý zviera tento uhol s osou x.

Zjavne jediné, čo nám stačí určiť, je práve uhol hľadanej priamky. Označme si súradnice staníc $S_1=[x_1,y_1], S_2=[x_2,y_2], \ldots, S_N=[x_N,y_N]$, kde N je ich počet. Päty kolmíc z bodov $S_1, S_2, \ldots, S_N$ majú na našej orientovanej priamke byť v tomto poradí (keďže sme našu priamku orientovali, budeme brať len poradie začínajúce pätou kolmice z S_1). Toto je ekvivalentné s požiadavkou, aby $\forall i$ bola päta kolmice z S_i pred pätou kolmice z S_{i+1}. Každá z týchto požiadaviek nám nejako obmedzí uhol našej priamky. Ako?

Majme dve stanice S_i a S_{i+1}. Keďže (ako sme si ukázali vyššie) môžeme našu priamku ľubovoľne posúvať, posuňme ju tak, aby prechádzala bodom S_i. Pätou kolmice z S_i bude teda priamo bod S_i. Kde bude päta kolmice z S_{i+1}?

Vektor $\overrightarrow{S_iS_{i+1}}$ označíme V_i, uhol, ktorý tento vektor zviera s našou orientovanou priamkou označíme α. Zjavne ak $\alpha=\pm\pi/2$, päta kolmice z S_{i+1} je práve S_i. Pre $\alpha\in(-\pi/2,\pi/2)$ budú päty kolmíc v správnom poradí, ináč v nesprávnom. Naša hľadaná orientovaná priamka musí teda s vektorom V_i zvierať uhol v absolútnej hodnote menší ako $\pi/2$.

Takto dostávame $N-1$ intervalov (pre každé dve po sebe idúce stanice jeden), v ktorých má ležať uhol našej priamky. Zjavne to ide práve vtedy, keď je ich prienik neprázdny. Tento prienik budeme počítať postupne. Vždy si budeme pamätať uhly pre ľavú a pravú hranicu intervalu – v programe premenné L, P. Vypočítame si nový interval uhlov do premenných LL a PP. Zoberieme tú z hodnôt L, LL, ktorá je zľava bližšie k P a uložíme ju do L. Analogicky z hodnôt P, PP vyberieme tú, ktorá je sprava najbližšie k L a tým dostaneme nový prienik. Tu si treba dať pozor, či nám počas vytvárania nových intervalov nevznikli intervaly väčšie ako π – keďže pôvodné intervaly boli široké nanajvýš π, znamenalo by to, že sme dostali záporný interval, teda prienik neexistuje. Ak takéto niečo nastane, ukončíme program s hláškou, že metro sa nedá postaviť.

Ak nakoniec dostaneme neprázdny interval, vyberieme z neho ľubovoľný uhol (v našom prípade priemer koncov intervalu). Nech je tento uhol α. Potom rovnicu priamky môžeme parametricky zapísať nasledovne: $p=\{[x,y] \mid x=t\cos\alpha,\ y=t\sin\alpha;\ t\in\mathbb{R}\}$ Odtiaľ dostaneme, že pre všetky body p platí $x\sin\alpha=y\cos\alpha$, a teda hľadaná priamka má všeobecnú rovnicu $x\sin\alpha-y\cos\alpha=0$.

Ešte sa oplatí spomenúť, ako vypočítať uhol, ktorý zviera vektor V_i s kladnou poloosou osi x. (Keď vieme rátať takéto uhly, vieme následne ľahko porátať orientovaný uhol ľubovoľných dvoch vektorov.) Na toto sa dá s úspechom využiť funkcia arkus tangens. Správna verzia, ktorá ráta presne to, čo chceme, sa vo FreePascale volá $ArcTan2(y,x)$, v C-čku $atan2(y,x)$ (obe sú v knižniciach math príslušných jazykov). Táto funkcia dostane ako parametre súradnice koncového bodu vektora umiestneného do počiatku sústavy súradníc a vráti orentovaný uhol z intervalu $\langle-\pi,\pi\rangle$, ktorý zviera vektor $[x,y]$ s kladnou poloosou osi x.

Algoritmus má časovú zložitosť $O(N)$. Pamäťová zložitosť je $O(1)$, lebo si pamätáme len informácie o intervale a práve spracovávanom bode.

Listing programu:

```
Uses Math;
Const Eps=0.000001;
Var
  I, N          :Integer;
  L, P          :Real; { lavy a pravy koniec hladaneho intervalu uhlov }
  LL, PP  :Real; { pomocny interval uhlov }
  X, Y    :Real; { novy bod }
  OX, OY  :Real; { predchadzajuci bod }
  Zle    :Boolean;

{ znormalizuj uhol -- vrati uhol v intervale <0,2*PI) }
Function NormUhol(U:Real):Real;
Begin While U<=0 Do U:=U+2*PI; While U>2*PI Do U:=U-2*PI; NormUhol:=U; End;

Begin
  ReadLn(N);
  If N<2 Then Begin WriteLn('x + y = 0'); Halt; End; { mame 0/1 stanic }

  ReadLn(OX, OY); ReadLn(X, Y);
  L:=NormUhol(ArcTan2(Y-OY, X-OX)+PI/2);
  P:=NormUhol(ArcTan2(Y-OY, X-OX)-PI/2);
  Zle:=False;

  For I:=3 To N Do Begin
    WriteLn('L=',L:10:10); WriteLn('P=',P:10:10);

    OX:=X; OY:=Y;
    ReadLn(X, Y);
    LL:=NormUhol(ArcTan2(Y-OY, X-OX)+PI/2);
    PP:=NormUhol(ArcTan2(Y-OY, X-OX)-PI/2);

    If NormUhol(LL-P)<NormUhol(L-P) Then L:=LL;
    If (NormUhol(L-P)>PI+Eps) Or (Abs(L-P)<Eps) Then Zle:=True;
    If (NormUhol(L-PP)>PI+Eps) Or (Abs(L-PP)<Eps) Then Zle:=True;
    If NormUhol(L-PP)<NormUhol(L-P) Then P:=PP;

    If Zle Then Begin
      WriteLn('Metro sa neda postavit.');
      WriteLn('L=',L:10:10); WriteLn('P=',P:10:10);
      Halt;
    End;
  End;
  WriteLn('L=',L:10:10); WriteLn('P=',P:10:10);
  WriteLn('(',Sin((L+P)/2):5:5,')x + (',-Cos((L+P)/2):5:5,')y = 0');
End.
```

Opravoval Dávidko
(max. 16 bodov)

3. O telegrafnej sieti

Koľko ľudí, toľko prístupov. Asi šlo o najťažší príklad kola, takže nikto nemal plný počet bodov. Bodovanie bolo nasledovné:

- Vzorové riešenie s časovou zložitosťou $\Theta(N^2)$ – 16 bodov.[3]
- $\Theta(N^3)$ alebo $\Theta(NM)$ riešenia – 14 bodov.

[3]To grécke písmenko sa volá theta. Značenie $g(n) = \Theta(f(n))$ znamená, že funkcie f a g sa líšia len o konštantu. V našom prípade teda ide o riešenia s **práve** kvadratickou časovou zložitosťou. Keby tam namiesto $\Theta(N^2)$ bolo $O(N^2)$, išlo by o riešenia s **najviac** kvadratickou čas. zlož.

- $\Theta(N^4)$ alebo $\Theta(N^2M)$ riešenia – 12 bodov.
- Exponenciálne a horšie riešenia – 8 bodov.
- Nefunkčné riešenia – 3 body.

Za zlý popis som stŕhal po dvoch bodoch. V polynomiálnych riešeniach, bol priam nutný dôkaz vety 2. Ten ale nemal nikto (dobre). Po zrelej úvahe som teda naň nebral ohľad. Pár ľudí nenapísalo program, tí dostali max. 4 body.

Vzorové riešenie: Najprv si úlohu preformuluje do reči teórie grafov. Telegrafné stanice budeme volať *vrcholy*, spojenia *hrany* a celú sieť budeme volať *graf*. *Cenou hrany* voláme jej dĺžku, *cenou* nejakej množiny hrán voláme súčet cien jej hrán. Pojmy *cesta* a *cyklus* sú, dúfam aspoň intuitívne, každému jasné. Graf voláme *súvislý*, ak medzi ľubovoľnými dvoma vrcholmi vedie cesta. *Stromom* voláme taký graf, ktorý je súvislý a nemá cyklus. Poľahky sa predsvečíme, že strom s N vrcholmi má práve $N-1$ hrán. Medzi ľubovoľnými dvoma vrcholmi stromu existuje práve jedna cesta. Ak odoberieme hranu zo stromu, prestane byť súvislým. A naopak, pridaním hrany do stromu vznikne jediný cyklus.

Zo zadania plynie[4], že náš graf (volajme ho G) je *súvislý*. Potom ale obsahuje strom s N vrcholmi a $N-1$ hranami. Takýto strom voláme *kostrou* nášho grafu. Malo by byť nad svetlo jasnejšie, že našou úlohou je teda nájsť druhú najlacnejšiu kostru grafu. (A vyhodiť všetky ostatné hrany.)

Myšlienka algoritmu v kocke je nasledovná: Nájdeme najlacnejšiu kostru T grafu G. (Na to existuje kopa štandardných a efektívnych algoritmov). Pridaním mimokostrovej hrany e (t.j. nepatriacej do T) a odobratím vhodnej kostrovej hrany f, dostaneme druhú najlacnejšiu kostru T', teda $T' = T \cup e - f$. (Toto budeme volať zámena.) Nám teda stačí jednoducho vyskúšať všetky mimokostrové hrany. Teraz podrobnejšie.

Veta 1: *Ak sú ceny hrán navzájom rôzne, existuje jediná najlacnejšia kostra*
Dôkaz: Sporom. Nech existujú dve rôzne najlacnejšie kostry S a T. Nech e je hrana, ktorá je v S, ale nie je v T. Odstránením tejto hrany z S sa nám S rozpadne na dve časti S_1 a S_2. Keďže S je najlacnejšia kostra, je e najlacnejšia hrana spájajúca S_1 a S_2 (inak by sme ju vymenili).

Pridajme teraz do T hranu e. Takto nám v T vznikne cyklus. Keďže jeden koniec e je v S_1 a druhý v S_2, musí v tomto cykle byť ešte aspoň jedna hrana e' vedúca medzi S_1 a S_2. Jej vyhodením dostaneme kostru T'. Ako sme však ukázali, práve vyhodená hrana bola drahšia ako e. To ale znamená, že kostra T' je lacnejšia ako T, čo je spor.

Poznámka: Keby mohli mať hrany rovnaké ceny, mohlo by sa nám stať, že hrany e a e' majú rovnakú cenu, a teda žiadny spor nenastane. Skutočne, v takomto prípade môže mať graf viac rôznych najlacnejších kostier. (Koľko ich má napríklad kompletný graf s jednotkovými hranami?)

Ak náš graf má len $N-1$ hrán (teda je stromom) tak existuje len jediná jeho kostra, ktorou je on sám. Táto kostra je zároveň najlacnejšou; vtedy teda druhá najlacnejšia neexistuje. Ak má graf aspoň N hrán, tak má viacero kostier, z nich je jediná najlacnejšia a preto niektorá z ostaných je druhá najlacnejšia. Tvrdíme, že platí nasledovná veta:

Veta 2: *Ak sú ceny hrán navzájom rôzne, tak všetky druhé najlacnejšie kostry*[5] *sa od (jedinej) najlacnejšej kostry líšia jedinou zámenou hrany.*
Dôkaz: Majme teda ľubovoľnú druhú najlacnejšiu kostru S a nech T je (jediná) najlacnejšia kostra. Zoberme hranu e, ktorá je v T a nie je v S. Podobne ako v prvej vete sa odstránením tejto hrany T rozpadne na T_1 a T_2, pričom keďže T je najlacnejšia kostra, e je najlacnejšia hrana medzi T_1 a T_2.

[4] aj keď nie veľmi

[5] Môže existovať aj sto rôznych druhých najlacnejších kostier.

Aj ďalej postupujme rovnako – pridajme e do S. Tam vznikne cyklus, ktorý určite obsahuje aspoň jednu ďalšiu hranu e', vedúcu medzi T_1 a T_2. Vyhodením tejto hrany z S dostávame novú kostru S', ktorá je lacnejšia ako S. Ak $S' = T$, dokázali sme naše tvrdenie. Ale ak $S' \neq T$, znamená to, že sme našli kostru, ktorá je síce drahšia ako T, ale lacnejšia ako S. To je ale spor s tým, že S je druhá najlacnejšia kostra.

Algoritmus bude modifikáciou (resp. rozšírením) Primovho algoritmu na hľadanie najlacnejšej kostry. Originálny Primov algorimus funguje takto: Postupne budujeme stromy $T_1, T_2, \ldots, T_N$, pričom každý ďalší vznikne pridaním jednej hrany a jedného vrcholu, až nakoniec T_N bude najlacnejšia kostra. T_1 je len izolovaný vrchol číslo 1. Ďalej postupujeme induktívne. Ak máme strom T_k s množinou vrcholov V_k, tak ako ďalšiu hranu pridáme, najlacnejšiu spomedzi hrán vedúcich medzi vrcholmi V_k a ostanými vrcholmi nášho grafu G. Toto vieme efektívne robiť napr. tak, že pre každý ešte nespracovaný vrchol si pamätáme najkratšiu hranu, ktorá doň vedie z nejakého už spracovaného vrcholu. Potom vždy len prejdeme dĺžky týchto hrán, vyberieme najkratšiu, príslušný vrchol označíme za spracovaný a pozrieme sa, či sa do niektorého nespracovaného nevieme dostať kratšou hranou ako doteraz.

Dôkaz správnosti je založený na tomto tvrdení: Ak ľubovoľne rozdelíme vrcholy grafu do dvoch množín, potom existuje najlacnejšia kostra, ktorá obsahuje najlacnejšiu hranu vedúcu medzi týmito množinami. V našom prípade vždy ako jednu množinu volíme už spracované vrcholy (t.j. V_k), za druhú ostatné. Dôkaz máte za domácu úlohu (podobá sa na dôkaz Vety 1).

Popri tejto stromotvorbe rátame hodnoty $max[u, v]$, čo je cena najdrahšej hrany na (jedinej) ceste v najlacnejšej kostre medzi vrcholmi u a v. (Hodnoty $max[u, u]$ položíme definitoricky rovné $-\infty$.) Po $k - 1$ krokoch bude platiť, že hodnoty $max[u, v]$ máme vyrátané pre všetky dvojice vrcholov u, v patriacich do V_k. Na začiatku máme len $max[1, 1] = -\infty$ a ďalej postujeme induktívne. Rozoberme prechod od T_k ku T_{k+1}, pričom T_{k+1} vzniklo pripojením nového vrcholu z novou hranou e ku starému vrcholu w ($w \in V_k$). My teraz potrebujeme určiť $max[u, z]$, pre všetky $u \in V_k$. (Hodnoty $max[x, y]$ ($x, y \in V_k$) rátať nemusíme, lebo sa nezmenila ani cesta medzi nimi vedúca.) Hodnoty $max[u, z]$ vyrátame teda, ako $max[u, z] := \text{maximum}(max[u, w], c(e))$, kde $c(e)$ je cena hrany e.

Máme vyrátanú najlacnejšiu kostru T s cenou $c(T)$ a hodnoty $max[u, v]$, čo s nimi? Je to jednoduché: Skúšame všetky mimokostrové hrany $e = (u, v)$. Pridaním hrany e dostávame cyklus, pričom e je zjavne najdrahšia hrana v tomto cykle. Aby sme dostali čo najlacnejšiu kostru, vyhodíme z tohoto cyklu druhú najdrahšiu hranu f (zjavne ak vyhodíme lacnejšiu hranu ako f, druhú najlacnejšiu kostru nedostaneme). No ale aká je cena f? Šikovnejší si už domysleli, že $c(f) = max[i, j]$, takže cena takejto kostričky bude $c(T') = c(T) + c(e) - c(f) = c(T) + c(e) - max[u, v]$. Z takto vytvorených kostier vyberieme najlacnejšiu, tá bude hľadanou druhou najlacnejšou kostrou daného grafu.

Všetko toto ide urobiť v čase a pamati $O(N^2)$. (Viem si predstaviť aj algoritmus v čase $O(N \log N + M)$ a pamäti $O(N + M)$ s použitím Fibonacciho haldy, ale to by zabralo asi 20 strán vysvetľovania a 300 riadkov kódu.)

Listing programu:

```
Program O_telegrafnej_sieti;

const MaxN = 100;
      inf = 9999; { nekonecno }

var N,M,i,j,x,y,l,min,min_i,best_i, best_j:integer;

    { matica susednosti t.j. g[i,j] je dlzka drotu medzi i,j }
    g:array[1..MaxN, 1..MaxN] of integer;
```

```
    { max[i,j] je dlzka najdlhsej hrany v najlacnejsej kostre
    na (jedinej) ceste medzi i,j }
    max:array[1..MaxN, 1..MaxN] of integer;

    d:array[1..MaxN] of integer;
    p:array[1..MaxN] of integer; { ku komu je vrchol i v kostre pripojeny }
    b:array[1..MaxN] of boolean; { je vrchol uz pripojeny do kostry ? }

begin
     readln(N,M);

     if (M<N)then
     begin
          writeln('Druha najlacnejsia kostra neexistuje.');
          halt;
     end;

     { vynulujeme }
     for i:=1 to N do
     for j:=1 to N do
          g[i,j]:=inf; { nekonecno }

     { nacitame dlzky drotov }
     for i:=1 to M do
     begin
          readln(x,y,l);
          g[x,y]:=l;
          g[y,x]:=l;
     end;

     for i:=2 to N do
     begin
          d[i]:=g[1,i]; { vzdialenosti k 1 }
          p[i]:=1;      { hrana k 1 }
          b[i]:=false;  { vsetci su mimo kostry }
     end;

     b[1]:=true; { 1 je uz v kostre }
     max[1,1]:=-inf; { minus nekonecno }

     { Primov algoritmus }
     for j:=1 to N-1 do
     begin
          min:=inf;

          { vyberieme najblizsieho ku kostre ...}
          for i:=1 to N do
          if (not b[i]) and (d[i]<min) then
          begin
               min_i:=i;
               min:=d[i];
          end;

          max[min_i, min_i]:=-inf;

          { poratame max[min_i, *] }
          for i:=1 to N do
              if b[i] then
              begin
                   max[i,min_i]:=max[p[min_i],i];
```

```
                if d[min_i]>max[i,min_i] then max[i,min_i]:=d[min_i];
                max[min_i,i]:=max[i,min_i];
            end;

        b[min_i]:=true; { ... pridame ho k nej }

        { prepocitame najkratsie hrany k mimokostrovym vrcholom }
        for i:=1 to N do
        if (not b[i]) and (g[min_i,i]<d[i]) then
        begin
             d[i]:=g[min_i,i];
             p[i]:=min_i;
        end;

    end;

    { takze TU mame hotovu najlacnejsiu kostru }

    min:=inf;

    { preberam mimokostrove hrany
    a kukame, ktoru zobrat do druhej najlacnejsej }
    for i:=1 to N do
    for j:=i+1 to N do
    if (g[i,j]-max[i,j]<min) and (g[i,j]-max[i,j]>0) then
    begin
         min:=g[i,j]-max[i,j];
         best_i:=i;
         best_j:=j;
    end;

    { vymazeme ju }
    g[best_i,best_j]:=inf;
    g[best_j,best_i]:=inf;

    { vymazeme vsetky hrany z kostry okrem tej za ktoru menime }
    for i:=2 to N do
        if g[i,p[i]]<>max[best_i,best_j] then
        begin
             g[i,p[i]]:=inf;
             g[p[i],i]:=inf;
        end;

    for i:=1 to N do
    for j:=i+1 to N do
        if g[i,j]<>inf then writeln(i,' ',j);
end.
```

Opravoval MišoF
(max. 15+ bodov)

4. O diamantoch

Najjednoduchší (a ako to už býva, zároveň najpomalší) spôsob, ako riešiť zadanú úlohu je prezrieť všetky možné diamanty a vybrať najväčší, ktorý obsahuje najviac jednu zlatú dlaždicu. Možných stredov diamantu je $O(MN)$, pre konkrétny stred budeme postupne zväčšovať diamant, kým obsahuje najviac jednu zlatú dlaždicu a neprekročí hranice podlahy. Toto vieme spraviť pre jeden stred v čase $O(MN)$ – každú dlaždicu pridáme do diamantu najviac raz. Preto celková časová zložitosť tohoto algoritmu je $O(M^2N^2)$. Za takéto riešenie sa dalo získať 8 bodov.

Poďme vymyslieť prefíkanejšie riešenie. Ak je zlatá dlaždica najviac jedna, riešenie ľahko nájdeme – je to najväčší diamant, ktorý vôjde na podlahu. Preto odteraz sa už budeme zaoberať len prípadom, keď sú zlaté dlaždice aspoň dve.

Vzdialenosťou dlaždíc $[x_1, y_1]$ a $[x_2, y_2]$ budeme volať číslo $|x_2 - x_1| + |y_2 - y_1|$. Takémuto meraniu vzdialenosti sa niekedy hovorí Manhattanovská vzdialenosť (alebo tiež metrika[6]). Manhattan je totiž (zjednodušene povedané) tvorený niekoľkými severojužnými ulicami a niekoľkými východozápadnými, a teda dĺžka najkratšej cesty medzi ľubovoľnými dvoma miestami nie je ich Euklidovská, ale práve Manhattanovská vzdialenosť. V našej úlohe je diamant veľkosti k tvorený práve tými dlaždicami, ktoré majú od stredu vzdialenosť menšiu ako k.

Zamyslime sa teraz nad tým, aký veľký diamant môže mať stred na dlaždici $[x, y]$. Môže obsahovať najviac jednu zlatú dlaždicu. To ale znamená, že jeho veľkosť môže byť najviac taká istá, ako (nezabúdajte že Manhattanovská) vzdialenosť druhej najbližšej zlatej dlaždice. Na druhej strane, takýto veľký diamant určite obsahuje najviac 1 zlatú dlaždicu. Ak by takto veľký diamant prečnieval mimo podlahy, budeme ho musieť ešte príslušne zmenšiť.

Takže jediné, čo potrebujeme na vymyslenie lepšieho riešenia, je vedieť pre každú dlaždicu určiť vzdialenosť druhej najbližšej zlatej. Najjednoduchší prístup je pre každý možný stred diamantu prejsť všetky zlaté dlaždice a nájsť druhú najbližšiu z nich. Takto dostávame riešenie s časovou zložitosťou $O(MNZ)$, kde Z je počet zlatých dlaždíc. Za takéto riešenie ste mohli dostať až 11 bodov.

Teraz si ukážeme dve riešenia, ktoré budú bežať v čase $O(MN)$, teda lineárnom od veľkosti podlahy. Za každé z nich ste mohli dostať 15 bodov. Prvé z nich bude vylepšením predchádzajúceho riešenia. Chceme pre každú dlaždicu určiť dve najbližšie zlaté. Použijeme upravené prehľadávanie do šírky – postupne budeme ofarbovať dlaždice, ktoré majú od zlatých vzdialenosť 0, 1, 2, atď. Akonáhle na dlaždicu prídeme od druhej zlatej, vieme, že toto je preň druhá najbližšia, takže si zapamätáme jej vzdialenosť od neho. Túto dlaždicu prehlásime za hotovú a už s ňou nič nerobíme.

A teraz trochu podrobnejšie. Budeme používať frontu, každý záznam v nej bude obsahovať súradnice práve spracúvanej dlaždice a poradové číslo zlatej dlaždice, odkiaľ ideme. (Keďže nás zaujímajú len najkratšie cesty zo zlatých dlaždíc, dĺžka cesty doteraz je práve vzdialenosť týchto dvoch dlaždíc a teda si ju nemusíme pamätať.) Na začiatku dáme do fronty pre každú zlatú dlaždicu $[x_i, y_i]$ záznam: spracúvame $[x_i, y_i]$, vyšli sme z zlatej dlaždice i.

Ako vyzerá spracovanie jedného záznamu? Ak sme prišli na dlaždicu, pre ktorú už vieme dve najbližšie zlaté, nič sa nedeje. Takisto sa nič nedeje, ak sme z tej istej zlatej dlaždice na túto už prišli aj skôr. (Napríklad na $[2, 2]$ sa z $[1, 1]$ vieme dostať cez $[1, 2]$ aj cez $[2, 1]$.) Ak sme na túto dlaždicu ešte doteraz neprišli, dozvedeli sme sa práve zlatú dlaždicu, ktorá je k nemu najbližšie. V opačnom prípade sme sa dozvedeli jeho druhú najbližšiu zlatú dlaždicu.

Keďže na každú dlaždicu prídeme len konečne veľa krát (koľko najviac? prečo?), bude časová zložitosť tohoto riešenia naozaj $O(MN)$. Implementáciu tohoto algoritmu nájdete na konci tohoto vzorového riešenia.

Iný možný prístup využíva metódu dynamického programovania. Budeme postupne prechádzať podlahu po riadkoch a pre každú dlaždicu $[x, y]$ si budeme počítať nasledovné hodnoty: $D_0[x, y]$ a $D_1[x, y]$ budú veľkosti najväčšieho diamantu, ktorý obsahuje najviac 0, resp. najviac 1 zlatú dlaždicu a má spodný roh na dlaždici $[x, y]$. Aby sme ich vedeli efektívne spočítať, budeme potrebovať ešte nasledujúce hodnoty: $L_0[x, y]$ a $L_1[x, y]$ budú dĺžky najdlhšieho úseku dlaždíc, ktorý končí na dlaždici $[x, y]$, vedie doľava dohora (t.j. cez dlaždice $[x - 1, y - 1]$, $[x - 2, y - 2]$, atď.) a obsahuje najviac 0, resp. najviac 1 zlatú dlaždicu. Podobne budeme mať $R_0[x, y]$ a $R_1[x, y]$ pre úseky vedúce doprava dohora.

Z priestorových dôvodov toto riešenie nebudeme rozpisovať. Rozmyslite si, že keď vieme tieto hodnoty pre všetky dlaždice s y-ovou súradnicou menšou ako dlaždica, ktorú spracú-

[6] metrika = spôsob merania vzdialenosti

vame, vieme pre ňu tieto hodnoty určiť v konštantnom čase. (L_i a R_i sú zrejmé. Diamant so spodným rohom na nejakej dlaždici sa dá rozdeliť na diamant so spodným rohom o dve dlaždice vyššie a na niekoľko dohora idúcich úsekov, jeho najväčšiu veľkosť vieme teda určiť z ich veľkostí. Teda D_0 spočítame z D_0 dlaždice o dva riadky vyššie a z L_0 a R_0 pre niekoľko okolitých dlaždíc. D_1 podobne, len prezrieme všetky možnosti (je ich len konečne veľa), v ktorej časti diamantu je zlatá dlaždica a vždy namiesto X_0 použijeme príslušnú hodnotu X_1.) Riešením úlohy je potom maximum z hodnôt $D_1\,[x, y]$.

A čo riešenie, za ktoré sa dalo získať viac ako 15 bodov? Nebudem ho tu rozoberať príliš detailne, podrobné riešenie a dôkaz toho, že naozaj funguje by bolo na samostatnú knižku. Chceme, aby časová zložitosť závisela len od počtu zlatých dlaždíc. Vyjdeme z myšlienky trochu podobnej prvému 15-bodovému riešeniu.

Budeme hovoriť, že každá dlaždica *patrí* k tej zlatej, ktorá je k nej najbližšie (ak je ich viac, patrí ku všetkým). Všimnime si, čo by sa stalo, keby sme si na každú dlaždicu podlahy poznačili, ku ktorej zlatej dlaždici patrí. Dlaždice by sa nám rozdelili na niekoľko (súvislých a „konvexných“) oblastí, pričom dlaždice v každej z oblastí by patrili k tej istej zlatej dlaždici. Takémuto rozdeleniu na oblasti sa hovorí Voronoiov diagram.[7]

Kľúčom k riešeniu bude nasledovná úvaha: Každú dlaždicu budeme reprezentovať bodom v jej strede. Hľadajme najskôr najväčší diamant, ktorý nebude obsahovať žiadnu zlatú dlaždicu.[8] Tvrdíme, že ako jeho stred sa oplatí skúšať len body v okolí miest, kde sa stretajú aspoň 3 rôzne oblasti. (A ešte ošetriť špeciálne prípady, keď sa diamant dotýka okraja podlahy.) Prečo je to tak?

Hovoríme, že bod reprezentujúci dlaždicu *leží pri hranici*, ak buď patrí k dvom zlatým dlaždiciam, alebo ak existuje bod vo vzdialenosti 1 od neho, ktorý patrí k inej zlatej dlaždici ako on. Majme diamant so stredom v bode, ktorý neleží pri žiadnej hranici. Potom stred tohoto diamantu môžeme o 1 posunúť smerom od najbližšej zlatej dlaždice. Táto zlatá dlaždica bude stále najbližšia (lebo sme neboli pri hranici), ale sme od nej ďalej, a teda vieme spraviť väčší diamant. Takto sa dostaneme do bodu, ktorý leží pri aspoň jednej hranici. Ak leží pri práve jednej, môžeme ho popri nej niektorým smerom posúvať tak, aby sa vzďaľoval od oboch najbližších zlatých dlaždíc. Toto robíme, až kým neskončíme v bode, ktorý leží pri aspoň dvoch hraniciach, teda pri mieste, kde sa stretajú aspoň 3 oblasti.

Teraz sme teda ukázali, že ku každému diamantu bez zlatých dlaždíc existuje aspoň taký istý veľký, ktorý má stred v bode ležiacom pri aspoň dvoch hraniciach. Takže ak hľadáme stred najväčšieho diamantu bez zlatých dlaždíc, stačí skúšať body v okolí miest, kde sa stretajú hranice oblastí. Pritom pre takýto bod vieme v lineárnom čase povedať veľkosť diamantu – prejdeme všetky zlaté dlaždice a nájdeme najbližšiu.

Nájsť takéto body vieme v čase $O(Z^3)$, keď si uvedomíme, že keďže každý z nich leží pri hranici dvoch susediacich oblastí, leží pri hraniciach Z_1Z_2 a Z_2Z_3 pre vhodne zvolené dlaždice Z_1, Z_2, Z_3. V našom riešení teda vyskúšame všetky trojice zlatých dlaždíc Z_1, Z_2, Z_3, pre každú (v konšt. čase) zistíme, či prienik hraníc Z_1Z_2 a Z_2Z_3 obsahuje len konšt. veľa bodov (inak nás táto trojica dlaždíc nezaujíma) a ak áno, tak každý z nich skontrolujeme ako potenciálny stred diamantu.

Takto dostávame riešenie, ktoré nájde najväčší prázdny diamant v čase $O(Z^4)$. Ľahko ho vieme upraviť, aby našlo najväčší diamant s najviac 1 zlatou dlaždicou, aj keď na úkor časovej zložitosti: Postupne vyskúšame každú zlatú dlaždicu vyhodiť a nájsť najväčší diamant neobsahujúci zlatú dlaždicu. Toto riešenie má časovú zložitosť $O(Z^5)$.

---

[7] Presnejšie keby sme takto rozdelili všetky body roviny, dostaneme Voronoiov diagram v Manhattanovskej metrike. Skúste si nakresliť, ako by vyzeral Voronoiov diagram pre N bodov v rovine a Euklidovskú metriku.

[8] Diamant je vlastne kruh v Manhattanovskej metrike – je to množina všetkých bodov, ktoré majú od stredu vzdialenosť najviac r. Podobná úvaha ako táto bude fungovať pre ľubovoľnú inú metriku, špeciálne teda aj pre kruhy v rovine.

Existujú ešte omnoho efektívnejšie riešenia, tým sa však už vyhneme obrovským oblúkom. Lepšie riešenia sú založené na myšlienke, že si zostrojíme už spomínaný Voronoiov diagram, t.j. nájdeme príslušné rozdelenie roviny na oblasti patriace jednotlivým zlatým dlaždiciam. Časti hranice, tvoriace tento diagram, si pamätáme ako rovinný graf, pre každú jeho stenu vieme, ktorej zlatej dlaždici patrí. Kandidáti na stred najväčšieho diamantu bez zlatej dlaždice ležia v okolí vrcholov tohoto grafu, ktorých je len lineárne veľa. Navyše každého z nich vieme spracovať v konštantnom čase – stačí pozrieť jeho vzdialenosť od tej zlatej dlaždice, do ktorej oblasti patrí.

Ďalšie vylepšenie, ktoré sa dá spraviť, je také, že nebudeme postupne vyhadzovať zlaté dlaždice a volať tento algoritmus. Nájdeme najskôr rozdelenie roviny pre všetky zlaté dlaždice. Uvedomme si, že keď teraz jednu vyhodíme, toto rozdelenie sa zmení len nepatrne – oblasť patriaca odobranej dlaždici sa prerozdelí medzi okolité. Toto prerozdelenie sa dá spočítať v konštantnom čase. Takže na začiatku nájdeme rozdelenie roviny medzi všetky zlaté dlaždice. Potom pre každú zlatú dlaždicu toto rozdelenie upravíme na rozdelenie bez nej a nájdeme najväčší prázdny diamant. Najlepšie riešenie, využívajúce všetky tieto myšlienky, má časovú zložitosť $O(Z \log Z)$, najlepšie reálne napísateľné za menej ako pol roka má časovú zložitosť $O(Z^2)$. (Kritické časti algoritmu sú zostrojenie rozdelenia roviny a jeho úprava po odstránení dlaždice, zvyšok vieme pomerne ľahko spraviť v $O(Z)$.)

Na záver už snáď len toľko, že myšlienka tohoto riešenia je ďaleko nad rámec stredoškolského (hm... aj veľkej časti vysokoškolského) štúdia. Navyše skutočná implementácia tohoto riešenia so všetkými okrajovými prípadmi je *naozaj veľmi náročná*. Preto nemusíte byť smutní, ak ste takéto riešenie nedokázali vymyslieť. Už to, že ste sa dočítali až sem, je úspech. (Zaujímalo by ma, koľko ľudí sa sem vôbec dočíta...) Nečakali sme, že niekto takéto riešenie pošle, ale vedeli sme, že takéto riešenie existuje a vždy tu bola šanca, že ho niekto vymyslí. A iste uznáte, že za takéto riešenie by si body navyše právom zaslúžil.

Listing programu:

```
program O_Diamantoch;
{ (c) MisoF, cas aj pamat O(MN) }

const MAXD = 20;

type QRec = record x,y,odkial : integer; end;

var   { fronta }
    Q           : array[0..MAXD*MAXD*4] of QRec;
    qs,qf       : integer;
      { dve najblizsie zlate dlazdice k policku }
    Bol         : array[1..MAXD,1..MAXD,1..2] of integer;
      { rozmery etc. }
    M,N,Z,i,j,k : integer;
      { suradnice zlatych dlazdic }
    Zlate       : array[1..MAXD*MAXD,1..2] of integer;
      { pomocne premenne }
    kde         : QRec;
    max,mx,my   : integer;

function Vzdialenost(x1,y1,x2,y2 : integer) : integer;
begin Vzdialenost:=abs(x2-x1) + abs(y2-y1); end;

procedure Nacitaj;
begin
   readln(M,N,Z);
   for i:=1 to M do for j:=1 to N do begin
      Bol[i,j,1]:=-1; Bol[i,j,2]:=-1;
```

```
   end;
   for i:=1 to Z do begin
      readln(Zlate[i,1],Zlate[i,2]);
      Q[i-1].x:=Zlate[i,1]; Q[i-1].y:=Zlate[i,2];
      Q[i-1].odkial:=i; { naplnime frontu }
   end;
   qs:=0; qf:=Z;
end;

begin
   Nacitaj;

   { osetrime specialny pripad }
   if (Z<2) then begin
      mx:=(M+1) div 2; my:=(N+1) div 2;
      max:=mx; if (my<max) then max:=my;
      write('Najvacsi diamant je na [',mx,',',my,'] ');
      writeln('a ma velkost ',max,'.');
      halt;
   end;

   while (qs<>qf) do begin { kym je nieco vo fronte }
      kde:=Q[qs]; inc(qs);

      { skontrolujeme, ci sme sa dozvedeli nieco uzitocne }
      if ((kde.x<1) or (kde.x>M) or (kde.y<1) or (kde.y>N))
         then continue;
      if (Bol[kde.x,kde.y,2]<>-1) then continue;
      if (Bol[kde.x,kde.y,1]=kde.odkial) then continue;
      if (Bol[kde.x,kde.y,2]=kde.odkial) then continue;

      { poznacime si, odkial sme sem prisli }
      i:=2; if (Bol[kde.x,kde.y,1]=-1) then i:=1;
      Bol[kde.x,kde.y,i]:=kde.odkial;

      { rozlejeme sa na okolite 4 policka }
      for i:=-1 to 1 do for j:=-1 to 1 do
         if (abs(i)+abs(j)=1) then begin
            Q[qf].x:=kde.x+i; Q[qf].y:=kde.y+j;
            Q[qf].odkial:=kde.odkial; inc(qf);
         end;
   end;

   { najdeme maximum }
   max:=0;
   for i:=1 to M do for j:=1 to N do begin
      k:=Bol[i,j,2];
      k:=Vzdialenost(i,j,Zlate[k,1],Zlate[k,2]);

      { upravime aby vosiel na dlazku }
      if (k>i) then k:=i;
      if (k>j) then k:=j;
      if (k>M-i+1) then k:=M-i+1;
      if (k>N-j+1) then k:=N-j+1;

      if (k>max) then begin
         max:=k; mx:=i; my:=j;
      end;
   end;
```

```
   { vypiseme }
   write('Najvacsi diamant je na [',mx,',',my,'] ');
   writeln('a ma velkost ',max,'.');
end.
```

Opravoval YoYo δ
(max. 10 bodov)

5. O celulárnych automatoch I

Ak napriek tomu, že Life je velmi známy, neprišlo až tak veľa riešení tohoto príkladu. Človek by si až myslel, že sa ostatní riešitelia dali zastrašit názvom príkladu. Čo je ale ešte horšie, z toho mála bolo ešte menej takých ktoré si zaslúžili plný počet bodov. Niektorí z vás si asi povedali, že 4 body za a) a b) nestoja za to vymýšľat ich[9]. Ďalší si to chceli uľahčit a prehlásili riešenie b) aj za riešenie a). To ale nie je celkom pravda. Rovnaká konfigurácia znamená, že každá bunka je v tom istom stave, ako bola predtým. Tomu ale posunutý obrázok nevyhovuje.

Čo sa týka prvej časti, tu sa našlo vo vašich riešeniach aj niekoľko zlých konfigurácií. Išlo vždy o to, že sa konfigurácia zopakovala po 2 sekundách (a nie viac ako 2, ako vyžadovalo zadanie). No a tu su aspoň dve správne konfigurácie:

2cm

„Cestujúcich“ konfigurácií je tiež veľa. Každý kto nejakú našiel, ju mal aj dobre. Najčastejšie (a asi aj najmenšie) konfigurácie sú:

2cm

Takže toto bola malá rozcvička, no a hor sa na zanikajúce hexaminá. Spôsobov, akými ste ich hľadali, bolo veľa a samozrejme sa neodzrkadlia na hodnotení `;-)`. Snáď najčastejší bol ručne si vyrobiť všetky hexaminá a potom spustiť nejaký simulátor, ktorý zistil, či zaniknú. Často sa ale stávalo, že ste na niektoré hexaminá pozabudli (je ich až 35). A boli aj takí, čo mali v riešení aj nezanikajúce hexaminá. Našlo sa aj niekoľko pokusov o program, ktorý by povytváral všetky hexaminá, ale jediné dostatočne úspešné boli založené na myšlienke: zoberme všetky pentaminá (samozrejme vyvtvorené ručne) popridávajme jednu kocku a vyhoďme rovnaké.

Tak, či onak, počet bodov bol závislý jedine od počtu správnych a nesprávnych konfigurácií a to takto:

$$b = \min\left(\frac{d}{2} - 2z - 1, 0\right)$$

kde d je počet správnych a z počet nesprávnych konfigurácií. A aby ste nepovedali, tak tu sú všetky zanikajúce hexaminá a jednoduchý simulátor Game of Life:

2cm

Listing programu:

```
uses crt;
const dx:array[1..8] of integer=(-1,-1,0,1,1,1,0,-1);
      dy:array[1..8] of integer=(0,-1,-1,-1,0,1,1,1);

var P,P1:Array[0..80,0..100] of integer;
    f:text;
    x,y,k,maxY:integer;
```

[9] alebo sa im buď nechcelo, alebo nevedeli

```
    s:string[120];

function min(a,b:integer):integer;
begin if a<b then min:=a else min:=b; end;

{ funkcia, ktorá ráta počet živých susedov}
function kolko(i,j:integer):integer;
var k,s:integer;
begin s:=0; for k:=1 to 8 do s:=s+P[ i+dx[k] , j+dy[k] ]; kolko:=s; end;

begin
  assign(f,'life.dat'); reset(f);
  y:=1;
  clrscr;
  while not eof(f) and (y<80) do begin
    readln(f,s);
    for x:=1 to min(length(s),79) do           {berieme iba prvých 79 stlpcov}
      if s[x]=' ' then begin                     {aby sa to dobre zobrazovalo}
        P[x,y]:=0; write(' ');
      end else begin
        P[x,y]:=1; write('#');
      end;
    writeln;
    inc(y);
  end;  { while }
  maxY:=y-1;                          {zobrazovať budeme toľko riadkov na výšku}
  while not(readkey=#27) do begin                          {koľko bolo v súbore}
    clrscr;
    for x:=1 to 79 do
      for y:=1 to 79 do begin
        k:=kolko(x,y);
        if (P[x,y]=1) and ( k in [2,3] ) then P1[x,y]:=1      {živá a prežije}
        else if (P[x,y]=0) and (k = 3) then P1[x,y]:=1       {tu vznikne nová}
        else P1[x,y]:=0;
      end;
    P:=P1;
    for y:=1 to maxY do begin                                    {a vykreslíme}
      for x:=1 to 79 do if P[x,y]=1 then write('#') else write(' ');
      writeln;
    end;
  end;  { while }
end.
```

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Závodný Jakub	Gym. Grösslingová BA	3	13	13	4	14	10	54
2	Tekeľ Jakub	Gym. Jura Hronca BA	3	13	7	14	11	7	52
2	Šatka Milan	Gym. Liptovský Hrádok	4	13	7	14	11	7	52
4	Jančuška Marek	Gym. Párovská Nitra	3	13	13	5	10	10	51
5	Klíma Jaroslav	Gym. Jura Hronca BA	4	13	13	3	10	10	49
6	Burger Michal	Gym. Grösslingová BA	3	13	12	3	10	10	48
7	Šomlo Ivan	Gym. Mládežnícka Šahy	3	13	13	8	10		44
8	Lenhardt Rastislav	GMMH Liptovský Mikuláš	3	13	8		8	7	36
8	Štefanek Anton	Gym. Jura Hronca BA	3	13	7		12	4	36
10	Lipková Juliana	Gym. Jura Hronca BA	4	13	4	4	10	4	35
10	Poláček Lukáš	Gym. K. Štúra Modra	3	13	7		10	5	35
12	Choma Martin	Gym. Stará Ľubovňa	4	13	4	14			31
13	Smažáková Dana	Gym. Jura Hronca BA	3			14	8	6	28
14	Burda Milan	Gym. Golianova Nitra	3	2	12	3	8		25
14	Palenčár Ján	Gym. Malá Hora Martin	3	13	6	4	2		25
14	Rejda Martin	Gym. Grösslingová BA	3	2	3	6	4	10	25
17	Kevický Michal	Gym. Grösslingová BA	3		13			10	23
18	Konečný Štefan	Gym. Jura Hronca BA	4	13	6	3			22
18	Kováč Jakub	Gym. Jura Hronca BA	3	13				9	22
18	Kuchynárová Mariana	Gym. Jura Hronca BA	3	11			5	6	22
18	Šufliarsky Peter	Gym. Nové Zámky	3	13				9	22
22	Repovský Michal	Gym. Komenského Trebišov	3	13	7				20
23	Šmitala František	Gym. Cyrila a Metoda Nitra	3	8		3	8		19
24	Štolc Miroslav	Gym. Párovská Nitra	3	8		3		6	17
25	Trejbal Ivan	Gym. Jura Hronca BA	3			8		7	15
26	Baláž Miroslav	Gym. Jura Hronca BA	3	12					12
26	Ružička Peter	Gym. Jura Hronca BA	2	5		4	3		12
28	Kollár Juraj	Gym. Jura Hronca BA	2	5		3	3		11
29	Janík Oliver	Gym. L. Stöckela Bardejov	3	8					8
30	Beňo Michal	Gym. Kremnica	3					7	7