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.

Kategória Z

Vzorové riešenia 2. kola zimnej časti

Milí riešitelia!
Tí najlepší z vás sa už určite tešia na sústredenie. A vy ostatní by ste sa mali o to pozornejšie začítať do týchto vzorových riešení, aby ste nabudúce boli múdrejší a mohli sa aj vy týždeň ulievať zo školy :-)

Quidquid latine dictum sit, altum viditur.[1]

KSPáci

1. Zememerači v Kiribati

opravoval Poko
(max. 15 bodov)

Milí riešitelia. Prajem vám všetko dobré do nového roku :-) Ale namiesto péefky vám radšej napíšem dajaký ten vzorák. Do rúk sa mi dostalo asi 40 riešení, z nich drvivá väčšina bola správna. To ma potešilo. Zasa sa však našli ľudkovia, čo opisovali (resp. si požičiavali zdrojáky), a to ma veru nepotešilo.

Celé riešenie môžeme rozdeliť do troch krokov. Zistíme, ktoré políčka sú more, ktoré sú ostrov, a nakoniec spočítame plochu a pobrežie. Prvé dve podúlohy sa riešia rovnako, ofarbovaním. Ukážeme si to na ofarbení mora. Na začiatku si ofarbíme to políčko, o ktorom určite vieme, že je more (napr. $[1, 1]$). A ďalej sa snažíme ofarbiť jeho susedov, susedov jeho susedov, atď. Tu sa vyskytli tri prístupy. Prvé riešenie je postupne prechádzať mapu, a keď nejaké políčko – voda susedí s už ofarbeným políčkom – morom, ofarbíme aj to a zapamätáme si, že sme v tomto prechode mapou niečo ofarbili. Je zrejmé, že pokiaľ v nejakom prechode mapou už nič neofarbíme, môžeme skončiť. Tento prístup potrebuje v najhoršom prípade až $O(M^2N^2)$ času.

Druhé riešenie využíva rekurzívny prístup. Z ofarbeného políčka sa pozrieme na jeho susedov. Ake je niektoré zo susedných políčok neofarbené, a malo by byť (t.j. je tam voda), tak ho ofarbíme, a rekurzívne sa na neho zavoláme. Toto riešenie potrebuje iba lineárne veľa času v závislosti od veľkosti mapy (t.j. $O(MN)$), ale nevýhodou je, že sa ťažko kontroluje pretečenie zásobníka a spotrebuje sa (síce iba konštantný násobok, ale predsa) viacej pamäte. Tento prístup zodpovedá prehľadávaniu do hĺbky, no dá sa upraviť tak, že si naprogramujeme svoj vlastný zásobník (potom by som proti tomu nič nenamietal).

Vzorové riešenie využíva prehľadávanie do šírky a dátovú štruktútu zvanú fronta (alebo rad). Fronta funguje presne tak isto ako rad u mäsiara. Človek si príde kúpiť bravčové na rezne, no v obchode sú nejakí iní ľudia, tak sa poslušne postaví do radu (fronty :-) a poslušne čaká na obslúženie. A na druhej strane, priblížime si ešte pohľad mäsiara. Keď obslúži jedného zákazníka, môže zo začiatku rady pristúpiť ďalší, a toho mäsiar potom obslúži. Na začiatku budeme mať vo fronte políčko so súradnicami $[1, 1]$, kde má byť určite more. Ofarbíme ho teda farbou „more“. Ďalej predpokladajme, že vo fronte budeme mať iba ofarbené políčka, u ktorých sme ešte nepozreli susedov. Zavoláme si políčko, ktoré je na začiatku fronty. Naše „obslúženie“ bude vyzerať tak, že sa pozrieme dookola na susedov, a ak niektorý z nich je

[1] Voľne preložené: Čokoľvek povedané po latinsky znie múdro.

(neofarbená) voda, tak ho prefarbíme na more a vložíme na koniec fronty, nech tam poslušne čaká na obslúženie (to spravíme s každým susedom – vodou). Celé to bude vyzerať naozaj tak, ako by sme očakávali. Na začiatku je vo fronte iba jedno políčko, následne sú tam len jeho susedia, po niekoľkých obslúženiach iba susedia susedov, atď. A keby sme si celý proces vykresľovali, videli by sme, že ofarbovanie postupuje ako vlna, keď hodíte do vody kamienok.

Ofarbenie ostrova prebieha rovnako, len ako prvé vložíme do fronty políčko so súradnicami $[P_x, P_y]$. A potom prefarbujeme pevninu. Posledná podúloha sa dá riešiť rôznymi spôsobmi. Najzrejmejší sa mi zdal nasledujúci: Prejdeme raz postupne od hora dolu a zľava doprava celú mapu. Vždy, keď narazíme na políčko ofarbené farbou „ostrov“ (vo význame „náš ostrov“), zväčšíme premennú, v ktorej si pamätáme plochu ostrova. A ešte sa (podľa definície v zadaní) pozrieme, či naše políčko nesusedí s nejakým políčkom ofarbeným farbou „more“. Ak áno, zväčšíme aj premennú, v ktorej si pamätáme dĺžku pobrežia, lebo toto políčko je pobrežie.

Pamätáme si celú mapu, ktorej veľkosť závisí od zadania a frontu, do ktorej uložíme každé políčko maximálne raz, preto jej veľkosť je približne rovnaká, ako veľkosť mapy. Okrem toho používame niekoľko málo jednoduchých premenných. Teda celková pamäťová zložitosť bude $O(MN)$. Ofarbenie mora aj skúmaného ostrova pobeží tiež v čase $O(MN)$. Totiž, každé políčko dáme maximálne raz do fronty (v najhoršom prípade tam dáme každé políčko), a keď je políčko vo fronte, práve raz ho „obslúžime“. Posledná časť (zisťovanie plochy ostrova a dĺžky pobrežia) je tiež v čase $O(MN)$, lebo prejdeme raz celú mapu, to znamená, že každé políčko práve raz. Preto časová zložitosť je tiež $O(MN)$.

Bodovanie:

- Riešenia podobné vzorovému som hodnotil maximálne 15 bodmi.
- Riešenia bežiace v čase $O(MN)$, ale využívajúce rekurziu, dostali maximálne 14 bodov.
- Riešenia, ktoré potrebovali čas $O(M^2N^2)$, získali maximálne 9 bodov.
- Zlé riešenia dostali najviac 5 bodov, podľa toho, do akej miery boli dobré :-)
- Strhával som 1 bod za zlý alebo chýbajúci odhad zložitosti, 1-2 body za nedostatočný popis, a nejaké bodíky šli dolu aj za chybičky vo vaších programoch.

Listing programu:

```
program Zememeraci_v_Kiribaty; {KSP-Z 1.priklad} {by Poko}
const max = 50; more = 2; ostrov = 3;
var P: array[0..max+1,0..max+1] of integer; {mapa}
    N,M,Px,Py: integer;                     {dalsi vstup}
    Q: array[0..max*max-1] of record i,j: integer; end; {fronta}
    qb,qe: integer;                                     {zaciatok,koniec}

procedure Nacitaj;
  var i,j: integer;
begin
  assign(input,'z1.in'); reset(input);
  read(N,M,Px,Py);
  for j:=1 to M do for i:=1 to N do read(P[i,j]);
  for i:=0 to N+1 do begin P[i,0]:=more; P[i,M+1]:=more; end;
  for j:=0 to M+1 do begin P[0,j]:=more; P[N+1,j]:=more; end;
  close(input);
end;

procedure Init; {inicializuje frontu}
begin qb:=0; qe:=0; end;

procedure Vloz(i,j: integer); {vlozi suradnice policka na koniec fronty}
begin Q[qe].i:=i; Q[qe].j:=j; Inc(qe); end;
```

```
procedure Vyber(var i,j: integer); {vyberie suradnice zo zaciatku fronty}
begin i:=Q[qb].i; j:=Q[qb].j; Inc(qb); end;

procedure Ofarbi(i,j,farba: integer); {ofarbi suvislu plochu od miesta i,j z farby "co" na "farba"}
  var co,ii,jj,di,dj: integer;
begin
  Init;
  co:=P[i,j]; P[i,j]:=farba;
  Vloz(i,j);
  while qb<>qe do begin
    Vyber(i,j);                {vyberieme dalsi bod na spracovanie}
    di:=1; dj:=0;              {oznacuju "zmenu" suradnic}
    for ii:=1 to 4 do begin
      if P[i+di,j+dj] = co then begin
        P[i+di,j+dj]:=farba;   {ak bod patri do plochy, ofarbime ho}
        Vloz(i+di,j+dj);       {a vlozime do fronty}
      end;
      jj:=di; di:=-dj; dj:=jj; {pozrieme vsetkych susedov -- rozmysli si, ako to funguje!!!}
    end;
  end;
end;

procedure Vypis;
  var i,j,plocha,pobrezie: integer;
begin
  plocha:=0; pobrezie:=0;
  for j:=1 to M do
    for i:=1 to N do
      if P[i,j]=ostrov then begin {pocitame plochu ostrova}
        Inc(plocha);
        if (P[i-1,j] = more) or (P[i+1,j] = more) or
           (P[i,j-1] = more) or (P[i,j+1] = more) then Inc(pobrezie);
      end; {ak policko susedi s morom, je to pobrezie}
  writeln('Plocha ostrova je ',plocha,'.');
  writeln('Dlzka morskeho pobrezia je ',pobrezie,'.');
end;

begin
  Nacitaj; Ofarbi(1,1,more); Ofarbi(Px,Py,ostrov); Vypis;
end.
```

2. Z gaštanov II

opravovala Monika
(max. 13 bodov)

Začnime tým, že si uvedomíme nasledovnú skutočnosť: Aby figúrka z G gaštanov držala pokope, musí mať aspoň $G-1$ zápaliek. Totiž keď postupne spájame gaštany zápalkami, každou pridanou zápalkou spojíme (najviac) dva súvislé kusy do jedného. A teda na to, aby sme z G kusov=gaštanov vyrobili jednu figúrku, potrebujeme aspoň $G-1$ zápaliek.

Majme teraz nejakú figúrku z G gaštanov. Už vieme, že aspoň $G-1$ zápaliek na nej musíme nechať, aby sa nerozpadla. Ale dá sa to vždy? Nepotrebujeme niekedy nechať viac zápaliek?

Ukazuje sa, že $G-1$ zápaliek vždy stačí nechať. Indukciou od G. Pre 1 a 2 gaštany to funguje. Majme teraz figúrku z $G>2$ gaštanov. Každú figúrku vieme dostať tak, že začneme s jedným gaštanom a niekoľkokrát pridáme nový gaštan a spojíme ho zápalkami s niektorými skôr pridanými. Všimnime si teraz posledný pridávaný gaštan. Tým, že ho (a zápalky doň) odoberieme, sa figúrka zjavne nerozpadne. Tak ale dostaneme figúrku z $G-1$ gaštanov.

Podľa indukčného predpokladu v nej stačí nechať $G-2$ zápaliek. No a teraz pridáme späť odobratý vrchol a ľubovoľnú jednu z odobratých zápaliek a vyhrali sme.

Alebo ešte raz pre tých z vás, ktorí už čo-to pochytili z teórie grafov. Na gaštany a zápalky sa môžeme pozerať ako na graf – gaštany sú vrcholy, zápalky sú hrany. Máme figúrku = súvislý graf s N vrcholmi, chceme nájsť jej súvislý podgraf s čo najmenej hranami. Zjavne musí mať aspoň $N-1$ hrán. Graf, ktorý má práve N vrcholov, $N-1$ hrán a je súvislý, voláme strom. Podgraf grafu, ktorý obsahuje všetky jeho vrcholy a je strom, voláme kostra. A ako to funguje u ľudí, tak je tomu aj u grafov – každý súvislý graf má kostru. Jednu kostru nájdeme napr. tak, že pustíme z ľubovoľného vrcholu ľubovoľné prehľadávanie a zoberieme každú hranu, ktorou prídeme do nového vrcholu. Takýchto hrán bude zjavne $N-1$, dokopy budú tvoriť strom (prečo?), a teda kostru.

Zhrnutie: Každý súvislý graf má kostru. Kostra je najmenší (čo do počtu hrán) podgraf grafu, ktorý obsahuje všetky jeho vrcholy a je súvislý. Kostra súvislého grafu (s N vrcholmi a M hranami) má $N-1$ hrán, preto z grafu môžeme odstrániť najviac $M-N+1$ hrán bez toho, aby sa rozpadol.

Teraz zostáva už iba otázka, ako ovplyvní množstvo figúrok počet potrebných zápaliek. Majme K figúrok. Počty gaštanov, ktoré obsahujú figúrky označíme ako $G_1, G_2, \ldots, G_K$ (G_i je počet gaštanov v i-tej figúrke). Podľa úvodnej úvahy najmenší počet zápaliek, ktoré musíme nechať vo figúrke i je $G_i - 1$, preto najmenší počet zápaliek, ktoré dokopy musíme nechať, je $(G_1-1)+(G_2-1)+\ldots+(G_K-1) = G_1+G_2+\ldots+G_K-K$. Každý gaštan sa bude nachádzať práve v jednej figúrke, preto súčet $G_1+G_2+\ldots+G_K = N$. Z toho vidíme, že najmenší počet zápaliek, ktoré musíme nechať, je $N-K$. Keby sme vedeli počet všetkých zápaliek M, ľahko spočítame, že najviac môžeme odstrániť $M-(N-K)$ zápaliek.

Ostáva už len jediná otázka. Ako zistíme počet figúrok? Odpoveď na túto otázku nám dáva predchádzajúci príklad „Z gaštanov“, kde sme práve tento problém riešili. Ako sme už aj vo vzorovom riešení tohto príkladu písali na hľadanie počtu figúrok existujú štandardné metódy zvané prehľadávanie do šírky a prehľadávanie do hĺbky, ktoré obe bežia v rovnakej (optimálnej) časovej aj pamäťovej zložitosti[2] $O(N+M)$. V minulom riešení sme podrobne vysvetľovali algoritmus prehľadávania do šírky, v ktorom sa vyskytla chybička, za ktorú sa ospravedlňujem. *(V riadku pred repeatom namiesto* `fronta[kon]:=1` *má byť* `fronta[zac]:=i`*.)* Ale som rada, že väčšina pozorných čitateľov vzorových riešení si ju všimla a hravo opravila a opravené vzorové riešenie využila pre tento príklad. Pre učenia chtivých sa v tomto vzorovom riešení podrobne vysvetľuje algoritmus prehľadávania do hĺbky.

Budeme využívat rekurziu. Rekurzia je volanie procedúry (funkcie) vo svojom vlastnom tele. Funguje nasledovne: Zavolá sa procedúra (funkcia), niečo sa v nej vykoná a sama seba zavolá znovu (toto sa môže opakovať niekoľkokrát). Pri tomto volaní sa všetky lokálne premenné využívané v procedúre (funkcii) so svojími hodnotami odzálohujú a v novom volaní procedúry (funkcie) sa pracuje s novými. Keď sa skončí táto procedúra (funkcia), vráti sa program na miesto, odkiaľ sa volala a všetky lokálne premenné nadobudnú svoje pôvodné hodnoty. Teda nič nové, všetko sa správa presne ako keby volala nejakú inú funkciu.[3]

Prehľadávanie do hĺbky funguje nasledovne: Budeme postupne farbiť gaštany. Pre každý gaštan si pamätáme, či už bol zafarbený alebo ešte nie. Na začiatku nie je zafarbený žiaden. Kým ešte máme nejaké neofarbené gaštany, opakujeme nasledovný postup: Vyberieme si ľubovolný neofarbený gaštan (v ktorom začneme ofarbovať jeho figúrku), zafarbíme ho a postupne pre každý gaštan, ktorý je s ním spojený a nie je ofarbený sa znova zavoláme a opakujeme tento postup až kým nebudú všetky gaštany zafarbené.

(Prehľadávanie do šírky sa dá opísať metaforou „začnem do vrcholu liať farbu, tá sa postupne rozteká a ofarbuje okolité vrcholy, až kým neofarbí všetko, kam sa dá dostať“. Prehľadávanie do hĺbky zodpovedá postupu: „blúdim po neznámom meste, značím si farbou,

[2] kde N je počet vrcholov a počet hrán prehľadávaného grafu

[3] Najklasickejším príkladom rekurzie je definícia faktoriálu: $fact(1)=1$, pre $n>1$ je $fact(n)=n.fact(n-1)$

kade som už išiel, keď prídem niekam, kde som už bol, vrátim sa ulicou, ktorou som prišiel a skúsim ísť inokade".)

Takéto prehľadávanie si vieme napísat ako procedúru, ktorá dostane ako vstupný parameter gaštan, označí ho a potom sa postupne pre každý gaštan, ktorý je s ním spojený pozrie, či je oznacený a ak nie, tak sa zavolá s týmto gaštanom ako vstupným parametrom.

Podľa tohto algoritmu sa dá napísať program, ktorého časová aj pamäťová zložitosť bude priamo úmerná počtu gaštanov a zápaliek dokopy. Pre každý gaštan si budeme pamätat, koľko zápaliek z neho vychádza a do ktorých gaštanov vedú (takto vieme rýchlo prezrieť všetky susedné gaštany). Pamäťová zložitost tohto programu je teda zjavne $O(N + M)$. Aj časová zložitosť nášho algoritmu je $O(N + M)$, lebo každý gaštan práve raz zafarbíme a zavoláme pre neho našu procedúru a po každej zápalke ideme najviac dvakrát – tam a späť.

Hodnotenie:

- funkčné riešenie podľa zložitosti 6-10 bodov
- nefunkčné riešenie alebo s chybami 0-5 bodov
- nejaká myšlienka, vysvetlenie, popis, ... max. 3 body

Listing programu:

```
Program Z_gastanov_II;
Uses Crt;
Const max=100;
Var oznac : Array [1..max] of boolean;
    vstup : Array [1..max,1..max] of integer;
    deg   : Array [1..max] of integer;
    utvary,N,M,i,a,b: integer;

{prehladavacia procedura}
Procedure Hladaj(q:integer);
var k:integer;
begin
  oznac[q]:=true;   {oznaci gastan v ktorom je}
  for k:=1 to deg[q] do   {pozrie vsetky ostatne gastany s ktorymi je spojeny}
    if oznac[vstup[q,k]]=false then
      Hladaj(vstup[q,k]);   {ak s nejakom nebol, tak tam ide}
end;

begin
  utvary:=0;
  {vstup}
  readln(N,M);
  for i:=1 to m do begin
    readln(a,b);
    Inc(deg[a]); vstup[a,deg[a]]:=b;
    Inc(deg[b]); vstup[b,deg[b]]:=a;
  end;
  {pozera vsetky gastany a ak nejaky nie je oznaceny, tak pusti z neho prehladavanie}
  for i:=1 to N do
    if oznac[i]=false then begin
      Inc(utvary); Hladaj(i);
    end;
  writeln('Misko moze odobrat najviac ',M-(N-utvary),' ks zapaliek.');
end.
```

3. Zvianočnieva sa II

opravoval Paľo
(max. 15 bodov)

Najprv niekoľko slov k hodnoteniu. Nech N je počet vetiev. Za riešenia v čase $O(N \log N)$ sa dalo získať maximálne 15 bodov, za riešenia v čase $O(N^2)$ maximálne 10 bodov, za riešenia, ktorých časová zložitosť závisela od veľkosti súradníc jednotlivých vetiev, ste mohli dostať maximálne 5 bodov. Body ste mohli stratiť za závažné chyby v programoch a za nedostatočný alebo chýbajúci popis programu.

Preformulujme si trochu našu úlohu. Vetvy budeme reprezentovať bodmi v rovine a my máme nájsť takú priamku prechádzajúcu začiatkom súradnicovej sústavy, ktorá rozdelí rovinu na dve polroviny tak, že v jednej z nich ostane najväčší možný počet bodov. Takú priamku nazvime „najlepšia“ priamka. Skúmajme teda polohu „najlepšej“ priamky. Pokiaľ neprechádza žiadnym zo zadaných bodov, tak ju môžeme o kúsok otočiť a polohy jednotlivých bodov voči nej sa nezmenia. Je zrejmé, že ju môžeme takto otáčať dovtedy, pokiaľ nebude prechádzať jedným z bodov a teda stačí skúmať len také priamky, ktoré nejakým z bodov prechádzajú. Teda veďme priamku každým z bodov a zistime koľko bodov je od nej naľavo resp. napravo a zapamätajme si tú „najlepšiu“ z nich. Toto sa dá pomerne ľahko implementovať v čase $O(N^2)$.

Skúsme to ale ešte o niečo zlepšiť. Utrieďme si body polárne, teda podľa uhlu ktorý zvierajú s kladnou x-ovou poloosou proti smeru hodinových ručičiek. Ten uhol môže nadobúdať hodnoty od 0 po 2π.[4] Problém je ako ten uhol vyrátať. V programovacích jazykoch máme k dispozícií funkciou arkustangens, ktorá má ako jediný parameter pomer y-ovej súradnice ku x-ovej súradnici. Tá nám dá ale správny uhol len pre bod v prvom kvadrante, takže ho musíme ešte upraviť podľa kvadrantu, v ktorom sa bod nachádza. Špeciálnym prípadom sú body na y-ovej ose, pretože tie majú nulovú x-ovú súradnicu, takže ich ošetríme zvlášť. V C-čku máme k dispozícii funkciu `atan2`, ktorá to všetko spraví za nás. (Takisto FreePascal už má funkciu `arctan2`, ktorá robí presne to isté.)

Postavme sa teraz do začiatku súradnicovej sústavy a otočme sa tak, aby sme sa pozerali na prvý bod (teda na bod, ktorý zviera s kladnou x-ovou poloosou najmenší uhol). Skúmajme postupne body proti smeru hodinových ručičiek. Tie sa spočiatku budú nachádzať naľavo od nás (teda ich uhol bude väčší od uhlu prvého bodu maximálne o π) a nájdime prvý z nich, ktorý sa nachádza napravo. Označme si ho X_1. Tým sme teda rozdelili body do dvoch polrovín určených priamkou prechádzajúcou prvým bodom a začiatkom súradnicovej sústavy. Otočme sa kúsok doľava tak, aby sme sa pozerali na druhý bod. Všimnime si, že netreba znovu skúmať polohu všetkých bodov, ale len polohu bodov ďalej od bodu X_1, lebo body medzi X_1 a druhým bodom sú určite aj teraz naľavo od nás. Nájdime prvý bod, ktorý je teraz od nás napravo prechádzaním bodov proti smeru hodinových ručičiek od bodu X_1. Tento označme X_2. Analogicky môžeme postupovať ďalej. Teda takto môžeme pre všetky priamky prechádzajúce nejakým zadaným bodom zistiť, koľko bodov je naľavo resp. napravo od nej. „Najlepšiu“ z nich si zapamätáme a na konci vypíšeme. Ešte sa dá urobiť jedno zjednodušenie. A to, že si stačí pamätať len počet bodov naľavo od priamky cez i-ty bod, pretože počet bodov napravo od nej je určite menší alebo rovný počtu bodov naľavo od priamky cez zodpovedajúci bod X_i, a teda stačí kontrolovať počty bodov naľavo od skúmaných priamok.

Aká je časová zložitosť tohoto algoritmu? Všimnime si, že vlastne len posúvame 2 indexy po jednotlivých bodoch – index bodu, ktorý nám určuje rozdelenie roviny a index takého bodu, ktorý je v pravej polrovine a má z nich najmenší uhol. Každý z indexov sa postupne posúva po bodoch proti smeru hodinových ručičiek a nakoniec sa nanajvýš vráti do bodu, z ktorého sa začal posúvať. Žiadny z nich sa ale nikdy nepohne opačným smerom, teda každý z nich zvýšime o 1 maximálne N-krát a pri každom zvýšení urobíme len konštantný počet

[4] Teda 2π radiánov, čo je to isté ako 360 stupňov.

operácií, teda zložitosť tejto časti programu je $O(N)$. Triediť ale vieme najrýchlejšie v čase $O(N \log N)$ (napr. quicksortom), a teda výsledná časová zložitosť je $O(N \log N)$. Pamäťová zložitosť programu je zjavne $O(N)$, pretože si pamätáme pre každý bod len konštantné množstvo údajov.

Listing programu:

```
const MAXKON = 1000;

type VETVA = record x,y,u: real; {suradnice a uhol vetvy} end;

var n: integer; {pocet vetiev}
    vet: array[1..MAXKON] of VETVA; {jednotlive vetvy}
    PI: real;

{funkcia triedi vetvy podla uhla quicksortom}
procedure qsort(zac, kon: integer);
var i, j: integer;
    str: real;
    tmp: VETVA;
begin
   if zac >= kon then exit;
   str := vet[kon].u;   {urci pivota}
   {rozdel pole na dve casti tak, ze v jednej su prvky < pivot, v druhej >= }
   while true do begin
      i := zac; while (vet[i].u < str) do inc(i);
      j := kon; while (j > zac) and (vet[j].u >= str) do dec(j);
      if (j <= i) then break;
      tmp := vet[i]; vet[i] := vet[j]; vet[j] := tmp;
   end;
   tmp := vet[i]; vet[i] := vet[kon]; vet[kon] := tmp;
   {zavolaj sa rekurzivne na vytvorene casti}
   qsort(zac, j); qsort(j + 1, kon);
end;

{funkcia vrati uhol bodu [x,y] s kladnou poloosou proti smeru hodinovych ruciciek}
function atan2(x,y:real): real;
var tmp: real;
begin
   {body na y-ovej osi osetri zvlast}
   if (x = 0) then begin
      if (y > 0) then atan2 := PI / 2 else atan2 := 3 * PI / 2;
   end else begin
      {vypocitaj uhol v prvom kvadrante}
      tmp := arctan(abs(y/x));
      if (y >= 0) and (x < 0) then begin {druhy kvadrant}
         tmp := PI - tmp;
      end else if (y < 0) and (x < 0) then begin {treti kvadrant}
         tmp := PI + tmp;
      end else if (y < 0) and (x > 0) then begin {stvrty kvadrant}
         tmp := 2 * PI - tmp;
      end;
      atan2 := tmp;
   end;
end;

{Funkcia vrati uhol medzi i-tou vetvou a predchadzajucou}
function uhol(i: integer): real;
begin
```

```
    if (i = 1) then begin
      uhol := vet[1].u + 2 * PI - vet[n].u;
    end else begin
      uhol := vet[i].u - vet[i - 1].u;
    end;
end;

var i,j: integer;
    uh: real; {uhol, ktory zviera j-ta vetva s i-tou vetvou}
    max,st: integer; {najvacsi najdeny pocet vetiev a odpovedajuca stena}
    poc: integer;  {pocet vetiev medzi j-tou a i-tou vetvou}

begin
    {vypocitanie PI} PI := arctan(1)*4;

    {nacitanie}
    read(n);
    for i := 1 to n do read(vet[i].x, vet[i].y);

    {pokial je na strome menej ako 2 vetvy, je jedno, kadial vedie stena}
    if (n <= 1) then begin
      writeln('Stena prechadza bodom [0, 1].');
        writeln('Na strome ostane ', n, ' vetiev.');
        exit;
    end;

    {vypocitame uhly, ktore zvieraju vetvy s kladnou x-ovou poloosou}
    for i := 1 to n do vet[i].u := atan2(vet[i].x,vet[i].y);
    {utriedime vetvy podla uhla}
    qsort(1,n);

    j := 2;
    uh := uhol(2);
    max := 0;
    poc := 1;

    {pre kazdy vetvu zistime, kolko vetiev je od nej nalavo resp. napravo}
    for i := 1 to n do begin
      {pokial su vetvy nalavo od i-tej, zapamataj si ich pocet a prvu vetvu napravo}
      while (uh <= PI) do begin
            inc(j);
            if (j = n + 1) then j := 1;
            uh := uh + uhol(j);
            poc := poc + 1;
        end;
        {pokial ich pocet je vacsi ako doteraz najdene maximum, tak si zapamataj stenu}
        if poc > max then begin max := poc; st := i; end;
        {posun sa o jednu vetvu dolava}
        if (i < n) then uh := uh - uhol(i + 1);
        poc := poc - 1;
    end;
    writeln('Stena prechadza bodom [', vet[st].x:0:3, ', ', vet[st].y:0:3, '].');
    writeln('Na strome ostane ', max, ' vetiev.');
end.
```

4. Žužu žuje žuvačku

opravoval Žužlo
(max. 15 bodov)

Žužu opravoval riešenia KSP. Keďže chcel všetkým body zadeliť spravodlivo, bol si kúpiť cukríky. Došiel do bufetu. Na poličke uvidel veľký výber cukríkov. Mali tam cukríky za všemožné, ale vždy celé kladné, ceny. Rozhodol sa, že investuje svoje úspory a nakúpi ich. Žužu, to prozreteľný finančník, nosí všetky svoje úspory vždy v ľavom vrecku nohavíc. Tu siahol do vrecka, spočítal všetky mince, gombíky a iné dôležité veci a určil jedno číslo – počet. To číslo určilo osud všetkých riešení. Čo nevidieť mal napísané všetky možnosti a začal overovať programy účastníkov, tam jeden padal na pamäťový, tam druhý na časový limit, a ten tretí sa nepodarilo ani skompilovať. No Žužu sa nevzdával a ďalej testoval, až vonku zašlo slnko. To musel riešiť iný problém; potreboval nájsť zásuvku, hmatal, hmatal po dlážke, až narazil na stenu, obhmatávajúc všetky steny našiel zdroj energie, do ktorého vložil svoju príručnú lampu, ktorú si nechával vždy poruke. A takto pri tlejúcom svetle, doopravoval zostávajúce riešenia.

A ako si Žužu vlastne poradil s týmto príkladom? To ani my presne nevieme. No zachovalo sa nám aspoň trochu toho, čo sa mu podarilo. Išiel nato sedliacky. Potreboval rozdeliť svoje peniaze na menšie časti, za každú z nich si potom kúpil jeden cukrík. Prvé, čo mu zo zadania vrazilo do nosa, bola podmienka, že nemôže vypísať jedno rozdelenie viac ráz. Určil teda, že ľubovoľné rozdelenie bude vytvárať usporiadané (v nerastúcom poradí), a preto sa mu nikdy nestane, že by vypísal jedno rozdelenie viac krát, len inak usporiadané.

Žužu teraz hlboko vo svojej mysli vedel, že vždy, keď si z regálu vyberie ďalší cukrík, tak musí stáť najviac toľko, ako stál ten predchádzajúci (ak vyberá prvý cukrík, je mu jedno aký bude drahý). Vybral si teda prvý cukrík. Systematicky pre všetky výbery prvého cukríka musel, prirodzene, skúsiť vybrať druhý cukrík. Rovnako pre všetky možné výbery prvého a druhého cukríka musel vyskúšať vybrať tretí cukrík. Takto vyberal cukríky až kým nevyčerpal všetky svoje peniaze. Uvedomil si, že nikdy si nevyberie viac cukríkov ako má peňazí (vždy musí vybrať cukrík hodnoty aspoň 1).

Teraz potreboval tieto myšlienky preniesť do programu. Vytvoril si rekurzívnu funkciu, ktorá mu pre každý platný (cukríky boli vyberané cenovo nerastúco a zatiaľ počet peňazí v jeho vrecku prevyšuje ich sumárnu cenu) výber vyskúša vybrať všetky cukríky, ktoré si ešte môže kúpiť. Takýmto jednoduchým spôsobom dostal náš Žužu všetky potrebné rozdelenia.

Zostávala naňho neľahká úloha spočítať náročnosť tohto prístupu. Priestorovú zložitosť odhadol dobre, pretože dobre vedel, že si nemôže vybrať viac ako N cukríkov. Stačí mu $O(N)$ pamäte. A čo s časovou náročnosťou? Vedel, že každé rozdelenie vypíše práve raz, pričom ich vytváral priamo, bez preberania nejakých zbytočností popri tom. Časová náročnosť je teda úmerná počtu vypísaných rozdelení. Začal si ich teda pomaly počítať všetky možnosti. Po chvíli sa mu z nich zatočila hlava a zaspal. Keď sa nasledujúce ráno zobudil, zistil, čo všetko povystrájal pri opravovaní a zhrozil sa natoľko, že to nechal tak. Rozdelení je naozaj veľa: pre $N = 50$ ich je už 204226.

Listing programu:

```
var cukriky : array[1..100] of integer;
    n : integer;

procedure rozdel(kolky,naposledy,este:integer);
var i:integer;
begin
  i:=este; if i>naposledy then i:=naposledy; { i=minimum(naposledy,este) }
  if i=0 then begin
    write(cukriky[1]); for i:=2 to kolky-1 do write(' ', cukriky[i]); writeln;
  end else while i>0 do begin
    cukriky[kolky]:=i; rozdel(kolky+1,i,este-i); dec(i);
```

```
  end;
end;

begin
  readln(n); rozdel(1,n,n);
end.
```

5. Zárm Oded

opravovalo YoYo δ
(max. 15 bodov)

Tento príklad patril k „neštandardným“, a preto aj bodovanie nebolo úplne štandardné. Aj keď základné delenie vašich programov bolo istým spôsobom z algoritmického hľadiska, veľkú úlohu zohrával estetický dojem z vašich stromčekov.

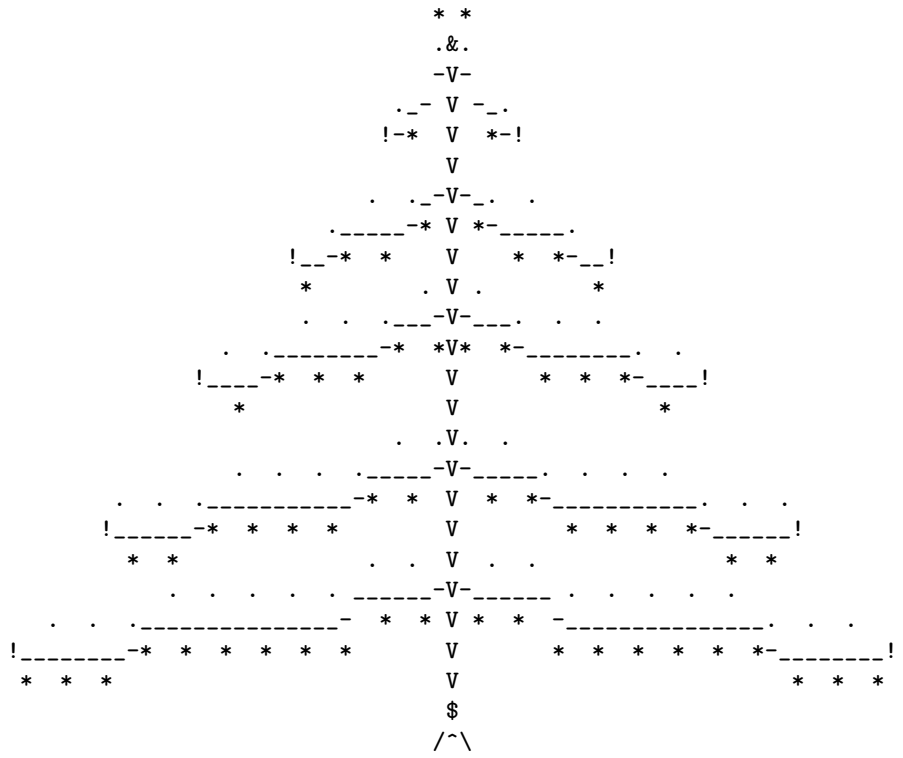
Veľkú skupinu Vašich riešení tvorili stromčeky s vodorovnými vetvami. Tie boli ohodnotené 10 až 13 bodmi. Našli sa aj nejaké, ktoré mali vetvy rozmiestnené **veľmi** nepravidelne. Tie získali aj menej ako 10 bodov.

Druhú skupinu tvorili tí, čo sa nezľakli a pustili sa do vykresľovania šikmých vetiev. Za takýto stromček sa dalo získať 13 až 15 bodov. Opäť ale platilo, že veľmi nepekné stromčeky si zaslúžili aj menej.

No a potom tu bolo ešte niekoľko špeciálnych stromčekov. Tie vykresľovali vetvy v nejakých zvláštnejších tvaroch. Sem patrili preddefinované tvary vetiev, z ktorých ste jednoducho vždy vykreslili potrebnú dĺžku. To ale väčšinou končilo veľmi nešťastnými kreáciami a bodový zisk bol tesne pod 10-timi bodmi. Našli sa ale aj riešenia, ktoré generovali rôznymi spôsobmi vetvy „neštandardných“ tvarov a ktoré vyzerali veľmi pekne (sem patrí aj víťazný stromček).

Estetický dojem závisel hlavne od zladenia dĺžky a sklonu vetiev, ale taktiež od správnej **výzdoby**. Niekoľko stromčekov ale bolo priam prezdobených.

No a tu je víťazný stromček (bohužiaľ čiernobiely, originál bol aj pekne farebný), autorom je Miroslav Cicko:



Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
1	Perešíni Peter	Gym. Tajovského B. Bystrica	1	68	15	12	14	15	13	137
1	Simančík František	Gym. Grösslingová BA	2	67	15	12	15	15	13	137
3	Cicko Miroslav	Gym. Tajovského B. Bystrica	2	63	14	13	15	15	15	135
4	Petruľák Matúš	Gym. Grösslingová BA	2	60	15	13	15	15	12	130
5	Porubský Juraj	Gym. Cyrila a Metoda Nitra	3	62	13	13	13	15	12	128
6	Labuda Tomáš	Gym. Grösslingová BA	3	66	15	13	14	8	10	126
7	Nánási Michal	Gym. Jura Hronca BA	2	61	14	12	9	13	14	123
8	Imriška Jakub	Gym. Jura Hronca BA	1	51	11	13	10	15	12	112
9	Blénessy Tibor	Gym. Poštová Košice	3	42	13	9	14	15	13	106
10	Krchniak Matej	Gym. Jura Hronca BA	1	49	14	12	1	15	12	103
10	Sedlák Milan	Gym. Liptovský Hrádok	3	47	15	13	10	6	12	103
10	Szórád Pavol	Gym. Cyrila a Metoda Nitra	3	57	14	2	10	6	14	103
13	Baník Dušan	Gym. Popradské nábr. Poprad	3	32	14	13	15	15	10	99
14	Hasaj Martin	Gym. Trebišovská Košice	4	41	14	5	10	15	13	98
15	Beleš Lukáš	Gym. Čadca	2	48	13	5	2	15	14	97
15	Mikulík Andrej	Gym. Grösslingová BA	3	46	14	11		15	11	97
17	Gottweis Martin	Gym. Jura Hronca BA	1	50	5	4	8	15	11	93
18	Petráš Daniel	Gym. Šk. Bratov BA	2	55	13	9		15		92
19	Plžík Milan	Gym. Ľ. Štúra Zvolen	2	42	14	11	9	15		91
20	Bundala Daniel	Gym. Jura Hronca BA	1	39	14	3	15	15	4	90
21	Ľudvík Martin	Gym. Školská Spiš. Nová Ves	2	42	8	11	10	14	4	89
22	Legény Jozef	Gymnázium	2	38	13	13	9		12	85
23	Zaťko Tomáš	Gym. 17. novembra Topoľčany	3	36	13	3	4	15	13	84
24	Buštor Stano	Gym. Jura Hronca BA	2	35	14	12	10	1	10	82
24	Zeman Marek	Gym. Jura Hronca BA	2	38	14	11		15	4	82
26	Ilavský Milan	SPŠE Liptovský Hrádok	3	34	5	9	5	15	13	81
27	Ambrošová Lucia	Gym. Grösslingová BA	1	26	15	13	10	15		79
28	Budáč Ondrej	Gym. Haličská Lučenec	1	36	14	13		15		78
29	Kuna Matej	Gym. Golianova Nitra	2	40	9	4		6	12	71
30	Poláčik Michal	GLN Tomášikova BA	3	14	15	13	13	15		70
31	Filo Michal	Gym. Golianova Nitra	4	24		13	4	15	12	68
32	Andruška Igor	SPŠ Levice	2	24	14	13		15		66
32	Nadanyi Peter	Gym. Tvrdošín	4	32		10	11		13	66
34	Šuška Martin	Gymnázium Levice	4	19	8		10	15	12	64
35	Varga Ľubomír	SPŠ Nitra	3	34	10		8	1	10	63
36	Morihladko Peter	Gym. Školská Spiš. Nová Ves	2	33	7	3		6	13	62
37	Schlosáriková Eva	Gym. SNP Piešťany	2	38		1	2	8	11	60
38	Vitásek Matej	Gym. Grösslingová BA	2	33	13				12	58
39	Batmendijnová Zuzana	Gym. Stará Ľubovňa	3	24	14	9	9			56
40	Klátik Peter	Gym. Grösslingová BA	3	54						54
41	Kováčik Martin	Gym. Čadca	4	15	8	10	4	15		52
42	Zelinka Ľuboš	Gym. Golianova Nitra	3	33		6	10			49
43	Mikuláš Ján	Gym. Haličská Lučenec	1	15	8	10	3		11	47
43	Tomori Alexander	Gym. Humenné	3	30	13		3	1		47
45	Beňo Michal	Gym. Kremnica	3	46						46
45	Ďuďák Juraj	Gym. Golianova Nitra	1	19	8			15	4	46
47	Rjabinin Igor	Gym. Cyrila a Metoda Nitra	3	29				6	10	45
48	Dzetkulič Michal	Gym. P. Horova Michalovce	2	41						41
49	Goga Peter	Gym. 17. novembra Topoľčany	3	15		5	1	2	15	38
50	Balga Jozef	Gym. - maďarské Šahy	4	23		12	1	1		37

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
50	Krištofič Milutín	Gym. Grösslingová BA	3	37						37
52	Hirjak Miroslav	Gym. Humenné	3	19	13		3	1		36
52	Polovková Darina	Gym. Školská Spiš. Nová Ves	3	8	15	13				36
54	Kažimír Pavol	Gym. Tornaľa	3	33						33
55	Fiala Martin	Gym. Grösslingová BA	3	31						31
56	Elman Michal	Gym. Golianova Nitra	3	29						29
57	Cyprich Peter	Gym. Čadca	4	13		3	3		9	28
57	Červenec Peter	Gym. Čadca	4	8	4	3	3	0	10	28
59	Králik Michal	GLN Tomášikova BA	3	0	14	4	9	0		27
59	Salaj Martin	Gym. Čadca	4	8	4	2	3		10	27
61	Antonič Michal	Gym. Grösslingová BA	1	13	13					26
62	Hennel Matúš	Neznama skola	3	25						25
62	Prievalský Juraj	Gym. Nedožerského Prievidza	3	25						25
62	Zápotocký Radoslav	Gym. Alejová Košice	3	25						25
65	Bielik Peter	Gym. SNP Piešťany	2	24						24
65	Borsuk Andrej	Gym. Grösslingová BA	1	15			9			24
65	Krištofík Štefan	SPŠ Levice	4	24						24
65	Sedlák Štefan	Gym. Daxnera V.n. Topľou	2	24						24
69	Budáčová Hana	Gym. Haličská Lučenec	3	23						23
69	Strobl Radoslav	Gym. Okružná Zvolen	3	23						23
71	Folkman Lukáš	Gym. Lettricha Martin	3	21						21
71	Potisková Lucia	Gym. Cyrila a Metoda Nitra	3	21						21
73	Trnovcová Zuzana	Gym. Jura Hronca BA	2	0	7	13				20
74	Smetana Róbert	Gym. Grösslingová BA	2	19						19
75	Doršic Matej	Gym. Grösslingová BA	2	18						18
76	Šimko Jakub	Gym. Jura Hronca BA	2	17						17
77	Kezeš Július	Gym. Golianova Nitra	3	16						16
78	Lebedová Barbora	Gym. Jura Hronca BA	1	15						15
78	Marek Pavol	Gym. A. Merici Trnava	4	15						15
78	Vranec Maroš	Gym. Alejová Košice	4	15						15
81	Golier Richard	Gym. Alejová Košice	3	14						14
81	Lacková Veronika	Gym. Jura Hronca BA	2	14						14
83	Hlavatá Ivana	Gym. Jura Hronca BA	2	0	13					13
83	Kadák Michal	Gym. Ľudovíta Štúra Trenčín	3	0	5	4		0	4	13
83	Minár Tomáš	Gym. Golianova Nitra	3	13						13
83	Ruska Štefan	Gym. Alejová Košice	2007	13						13
83	Tvrdý Lukáš	Gym. Čadca	2	8		5				13
88	Fzeši Štefan	Gym. - maďarské Šahy	2007	12						12
89	Husár Radoslav	Bilingválne gym. Sučany	2	10						10
90	Rusinko Rastislav	Gym. Alejová Košice	2007	8						8
91	Černík Peter	Gym. Trebišovská Košice	2	7						7
92	Petrezsél Tibor	Gym. - maďarské Šahy	3	6						6
92	Strama Ján	Gym. Alejová Košice	4	6						6
92	anonym	Neznama skola	2007	0		2	2	2		6
95	Hrabčák Ján	Gym. Alejová Košice	3	5						5
96	Polák Matúš	Gymnázium	2	3				1		4
96	Sakáč Pavol	Gym. Alejová Košice	3	4						4