Korepondenčný seminár z programovania
XIX. ročník, 2001/2002
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje nadácia Open Society Fund Bratislava a IUVENTA – zariadenie pre vožný čas detí, mláde«e a dospelých MICROSTEP spol s.r.o.

Kategória Z

Vzorové riešenia 4. kola kategórie Z

Milé deti,

Tak, a už máte od KSP konečne pokoj! Tým, ktorým ešte nestačilo, nech si prečítajú tieto vzoráčiky. Tým, ktorým ešte ani to nestačilo, sme pripravili tradičný už 4. ročník **splavu Hronu**. Viac sa dozviete na **www.ksp.sk/ksp**. Ak toto ešte niekto vydrží, radi ho uvítame ako riešiteľa ďalšieho ročníka, prípadne sa s ním stretneme na jesennom sústredení...

Na boľavé nohy treba teplú vodu, na boľavú hlavu studenú a do brucha kofolu.

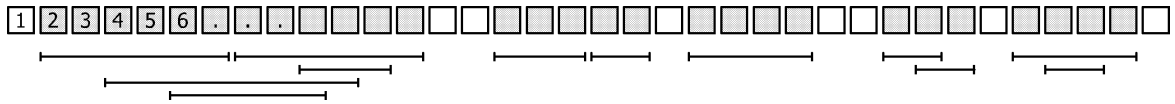
KSPáci

1. Z podzemného archívu

opravoval Dávidko
(max. 15 bodov)

Najprv k hodnoteniu. Riešenia bolo možné rozdeliť na dve veľké kopy. Jedna kopa boli riešenia, ktorých časová zložitosť závisela od počtu popísaných políčok pásky (napríklad keď jedno jediné dieťa popíše interval 1 až miliarda, tak takéto algoritmy budú bežať veľmi dlho). Za takéto riešenie ste mohli získať max. 5 bodov. Druhá kopa riešení boli riešenia založené na manipulácii s hranicami intervalov, a teda ich časová zložitosť závisí len od N. Riešenia tohto typu možno rozdeliť podľa časovej zložitosti na $O(N^3)$ – max. 10 bodov, $O(N^2)$ – max. 11 bodov, $O(N\sqrt{N})$ – max. 12 bodov, max. $O(N \log N)$ – max 15 bodov. Niektorí z vás predpokladali, že intervaly sú na vstupe zotriedené podľa ich začiatku. Za takýto predpoklad ste dostali bonus -5 bodov. Za chýbajúci alebo nedostatočný popis som strhal 1 bod. Podobne to bolo s odhadom (najmä časovej) zložitosti. Nefunkčné riešenia mohli získať max. 1 bod.

Vzorové riešenie: Najprv si nakreslíme pekný obrázok, aby sme vedeli o čo ide.



Predstavme si intervaly namaľované na reálnej osi. Pozor, interval `i j` zo zadania, zodpovedá uzavretému intervalu $\langle i-1, j\rangle$. To vyplýva z toho, že i-te políčko pásky má začiatok v bode $i-1$.

Našou úlohou je vlastne zrátať dĺžku ich zjednotenia. Všimnime si, že ich zjednotením je opäť niekoľko väčších intervalov, ktoré sa neprekrývajú. Tieto väčšie intervaly budeme volať *úseky*. Keby sme vedeli, ako budú také úseky vyzerať (kde končia a kde začínajú), ľahko by sme zrátali ich dĺžku. Určite však uveríme, že úsek začína niekde, kde začína nejaký interval a končí, kde nejaký (možno iný) interval končí.

Algoritmus bude fungovať nasledovne. Načítame do jedného poľa začiatky a konce intervalov. Pre každý bod si v poli pamätáme jeho pozíciu na reálnej osi a to, či je to začiatok alebo koniec. Body utriedime podľa ich pozície na reálnej osi (bez ohľadu na to, či ide o začiatok alebo koniec). Teraz poďme prstom zľava doprava po reálnej osi a potichu si počítajme nasledovnú hodnotu: Počet začiatkov intervalov naľavo od prsta mínus počet koncov intervalov tiež naľavo od prsta, alebo (inými slovami) počet intervalov, v ktorých práve náš prst je.

Fungovať to bude asi takto: Na začiatku si potichu povieme: „Nula!“. Ideme pomaly prstom po osi, až prídeme po prvý začiatok nejakého intervalu, kde si povieme: „Jedna!“. Podobne postupujeme až do omrzenia. Ak prídeme na ďalší začiatok, tak si čísielko zvýšime o jedna, ak prídeme na koniec nejakého intervalu, tak znížime o jedna. Ľakho uveríme[1], že úsek začína vtedy, keď sa náš rozdiel zvýšil z nula na jedna. Naopak úsek končí, keď sa rozdiel znížil z jedna na nula. Program to robí presne tak isto ako my prstom, s tým rozdielom, že rovno skáče po bodoch (začiatkoch a koncoch intervalov).

Nesmieme však zabudnúť rátať, čo sme pôvodne chceli, t.j. celkovú dĺžku intervalov. Vieme, že úseky sa už neprekrývajú, preto to hravo zvládneme. Budeme si ešte navyše pamätať pozíciu začiatku posledného úseku. V okamihu, keď úsek končí, vieme jeho začiatok aj jeho koniec. Odčítaním týchto dvoch hodnôt dostaneme jeho dĺžku. Túto hodnotu potom stačí pripočítať k medzivýsledku.

Ešte sa vráťme späť k takej naoko jednoduchej veci ako je triedenie. Na triedenie existuje hŕba štandardných algoritmov, preto si neváhajte pozrieť nejaký v knižke alebo sa spýtajte svojho učiteľa. My sa budeme zaoberať algoritmom zvaným Quicksort. Funguje rekurzívne, preto sa nezľaknite. Volanie `Quicksort(l,r)` utriedi úsek poľa od l po r (vrátane), pričom zvyšok poľa nechá napokoji. Ak ide o úsek dĺžky 1, nie je čo robiť, je už utriedený. Inak vyberieme a zapamätáme si hodnotu ľubovoľného prvku x z úseku (v našej implementácii je to stredný prvok). On bude slúžiť ako merná lata. Našou úlohou bude poprehadzovať prvky v poli, tak aby naľavo od x boli prvky menšie alebo rovné x a napravo väčšie alebo rovné ako x. To sa spraví nasledovne. Postupujeme indexom i zľava a nájdeme prvý prvok väčší ako x. Podobne indexom j sprava nájdeme prvý menší ako x. Tieto dva vymeníme. Postup opakujeme od týchto pozícií ďalej doľava, resp. doprava, až kým sa indexy neprekrížia. V tom okamihu sú naľavo prvky menšie alebo rovné x a napravo väčšie. Potom sa rekurzívne zavoláme na obidve polovice, čím dostaneme utriedené pole.

Prácu algoritmu si možno rozdeliť podľa úrovní vnorenia do rekurzie. Na každej úrovni sa pracuje s celým poľom. Na úrovni jedna pracujeme s celým poľom naraz, na úrovni 2 s dvoma polovicami, na úrovni 3 štyrmi štrvtinami atď. Úrovní je v priemernom prípade $\log_2 N$ a časová zložitosť na každej z nich je $O(N)$. Spolu je teda časová zložitosť triedenia $O(N \log N)$.

Časová zložitosť zvyšného rátania je len $O(N)$, keďže na každý z $2N$ bodov (začiatkov alebo koncov) intervalov minieme len konštantne veľa času. Celková časová zložitosť je teda $O(N \log N) + O(N) = O(N \log N)$. Pamäťová zložitosť algoritmu je pochopiteľne $O(N)$, keďže si pre každý interval pamätáme len niekoľko málo čísel.

Listing programu:

```
Program Zlta_paska;

type bod = record
                kde:integer; { pozicia }
                co:boolean;  { zaciatok/koniec }
           end;

var a:array[1..1000] of bod;
    N,i,z,k:integer;
    P:integer; { pocet zafarbenych policok pasky }
    zaciatok:integer; { zaciatok useku }
    intervalov:integer; { pocet zacatych ale este neskoncenych intervalov }
```

[1] ak neuveríme, tak si to vyskúšame

```
{ triedenie }
Procedure QuickSort(l,r:integer);
var i,j,x:integer; p:bod;
begin
  if l>=r then exit;
  i:=l; j:=r; x:=a[(l+r) div 2].kde;
  repeat
    while a[i].kde<x do inc(i);
    while a[j].kde>x do dec(j);
    if i<=j then
    begin
      p:=a[i]; a[i]:=a[j]; a[j]:=p;
      inc(i); dec(j);
    end;
  until i>j;
  QuickSort(l,j);
  QuickSort(i,r);
end;

begin
  assign(input,'paska.in'); reset(input);
  readln(N);

  for i:=1 to N do
  begin
       readln(z,k);
       { zaciatocny bod }
       a[2*i-1].kde:=z-1; { pozor zaciatok intervalu je 1 mensi }
                          { ako jeho prve policko }
       a[2*i-1].co:=false;
       { koncovy bod }
       a[2*i].kde:=k;
       a[2*i].co:=true;
  end;

  Quicksort(1,2*N); {  body utriedime }

  { Mame 2N bodov. Co s nimi? }
  intervalov:=0; { pocet zacatych ale este neskoncenych intervalov }
  P:=0; { pocet popisanych policok }
  for i:=1 to 2*N do
  begin
     if a[i].co then { koniec / zaciatok ?}
     begin
         dec(intervalov); { znizime pocet intervalov }
         { ak skoncili usek, tak zapocitame dlzku }
         if intervalov=0 then P:=P + (a[i].kde-zaciatok);
     end else
     begin
         inc(intervalov); { zvysime pocet intervalov }
         { ak zacal novy usek, tak si zapamatame jeho zaciatok }
```

```
        if intervalov=1 then zaciatok:=a[i].kde;
    end;
  end;

  writeln('Deti popisali ',P,' policok zltej pasky.');
end.
```

opravovala Ňaňka
(max. 15 bodov)

2. Zbytočný Miško

Tento príklad rozhodne nebol najobľúbenejší z tejto série. Počtom riešení ho predbehla dokonca aj Zeofína. Pritom nešlo o ťažký príklad, o čom ma presvedčili aj vaše väčšinou správne riešenia. Preto som sa rozhodla, že ho budem bodovať netradične.

V prvom rade, samotný program – riešenie príkladu – maximálne 8 bodov:

- vzorové riešenie – 8 bodov
- mierne zhoršenia vzorového riešenia – 4–6 bodov
- nefunkčné riešenia – 1–2 body

Hor' sa na vzorák. Najprv teda hlavná časť programu. Predpokladajme, že by sme špiónov mali očíslovaných číslami $1..N$. Mnohí (myslím, že všetci) ste správne pochopili, že špióni budú vrcholy a podávanie správ orientované hrany grafu. Vzorové riešenie je známe pod názvom topologické triedenie. Myšlienka je veľmi jednoduchá. Z nejakej množiny špiónov nazvime *„najväčším šéfom“* tých, ktorí nemusia informovať žiadneho z nich. Ľahko nahliadneme, že pokiaľ máme množinu špiónov, v ktorej je nejaký najväčší šéf, jeho odstránením z nej nič nepokazíme. To preto, že ak sa dala táto množina usporiadať pred jeho odstránením, dá sa aj po ňom, lebo nijakému špiónovi nepribudol iný, ktorý ho informuje. Inými slovami jedno vyhovujúce usporiadanie zvyšku množiny je také, že zoberieme usporiadanie celej množiny a vynecháme z neho tohto šéfa. No a keď tohto šéfa pridáme na začiatok takto získaného poradia, nič tým nepokazíme, lebo on nikoho za sebou neinformoval. Takže z ľubovolnej množiny špiónov môžeme hociktorého najväčšieho šéfa vybrať a vypísať ho. Tým v nej môže vzniknúť niekoľko nových najväčších šéfov.

Jediný problém nastáva, ak sme v situácii, keď ešte máme nejakých špiónov, ale žiaden z nich nie je najväčší šéf. V takomto prípade ale každý niekoho informuje a teda nik nemôže byť prvý v poradí – nedajú sa usporiadať, no a z predchádzajúceho odseku vieme, že v takomto prípade sa nijako nedala usporiadať ani pôvodná množina.

Konkrétna implementácia tohto algoritmu (inšpirovaná Jánom Katreničom)

Pre každého špióna si budeme pamätať jeho meno (*meno*), počet špiónov, ktorých informuje (*pkomu*), počet špiónov, ktorí ho informujú (*podkoho*) a zoznam špiónov, ktorí ho informujú (Inf). Na začiatku si do fronty[2] (Q) zaradíme všetkých tých špiónov, ktorí nikoho neinformujú. Ľubovoľné ich poradie je správne, pretože nikto z nich nie je nadriadený ani podriadený inému z nich.

Postupne budeme ľudí vo fronte spracovávať nasledovným spôsobom: znížime *pkomu* všetkým jeho informátorom a vypíšeme ho; tým sme vlastne zrušili hrany vedúce do tohto vrcholu (a žiadne hrany z neho už nevedú, pretože práve preto sme ho zaradili do fronty), takže sme z nášho grafu odstránili jeho šéfa spolu so všetkými príslušnými hranami. Ak sa medzi jeho informátormi vyskytol šéf zvyšnej skupiny (teda po znížení jeho *pkomu* to bude 0), zaradíme ho do fronty. Máme zaručené, že každého špióna do fronty zaradíme iba raz, a to vtedy, keď zrušíme posledného jeho nadriadeného. Tiež je zrejmé, že ak špión A nosí správy špiónovi B, tak sa nemôže stať, že prvého vypíšeme špióna A, pretože u neho sa $pkomu = 0$ až vtedy, keď mu ho zníži aj odstránenie špióna B.

[2] pozri 1. kolo

Zložitosť tejto časti programu je $O(M + N)$, teda lineárna od veľkosti grafu, pretože každú hranu (tých je spolu M) spracúvame práve vtedy, keď rušíme špióna (ktorých je spolu N), ku ktorému vedie.

Pokračujme v udeľovaní bodov. Nemožno zabudnúť ani na popis – podľa kvality, maximálne 3 body.

A nakoniec, teda na začiatok – načítanie – maximálne 4 body. V tejto časti ide o to, akým spôsobom zisťujete, o ktorého špióna ide, či už bol načítaný a iné problémy s tým spojené:

- vzorové riešenie – hashovanie – 4 body
- stromy a iné riešenia v $O(N \log N)$ – 3 body
- tabuľka špiónov v $O(N^2)$ – 2 body
- rôzne zjednodušenia (napr. načítavanie čísel) – 1 bod

Idea obyčajnej tabuľky je jednoduchá a keďže už vieme, čo v nej chceme mať, implementáciu ponechávam plne na vás. Ak chceme časovú zložitosť programu aspoň v priemernom prípade zlepšiť, môžeme použiť hashovanie. V najjednoduchšom prípade ide o zakódovanie mena do nejakého čísla z intervalu $\langle 0, p-1 \rangle$, čo už priam navádza na zvyšok po delení číslom p. Konkrétna použitá funkcia je iba jedna z mnohých možných, ktoré sa dajú nájsť v múdrych knižkách alebo aj vymyslieť. Problém nastáva, keď dve rôzne mená majú rovnakú hodnotu. Tomuto budeme hovoriť kolízia. Jej ošetrenie môžeme opäť hľadať v knihách, ja využívam najjednoduchšie posúvanie indexu, kým sa nedostanem až na voľné miesto. Potom pri hľadaní pozerám dovtedy, kým nenájdem hľadané meno, alebo voľné miesto (v tom prípade tam hľadané meno nie je). Konkrétnu implementáciu si pozrite priamo v programe.

Listing programu:

```
program Zbytocny_Misko;

uses crt,strings;
const nmax=50;
      p=151;  {pre hashovanie - nejake prvocislo vacsie ako nmax, inak nmax}
type spion=record
            meno:string;                                              {meno}
            pkomu,podkoho:0..nmax;   {pocet spionov ktorych/ktori ho infor.}
            Inf:array[1..nmax] of 0..nmax;      {spioni, ktori ho informuju}
           end;
     tab=record
           s:string;
           i:0..nmax;
          end;
var A:array[1..nmax] of spion;
    Q:array[1..nmax] of 0..nmax;                    {fronta cakajucich sefov}
    T:array[0..p-1] of tab;                              {hashovacia tabulka}
    i,j,n,nq:integer;
    s1,s2:string;
    c:char;

function Getkey(s:string):integer;
var x:integer;
 begin
  x:=0;
```

```
  for i:=1 to length(s) do
   x:=(64*x + ord(s[i])) mod p-1;
  while (T[x mod p].s<>s) and (T[x mod p].s<>'') do inc(x);
  { osetrenie plnej tabulky zvladnete aj sami }
  if T[x mod p].s=s then Getkey:=(x mod p) else Getkey:=0;
 end;

procedure Zarad(s1,s2:string);
var n1,n2,n0:integer;
 begin
  n1:=Getkey(s1);                              {zistime ktory spion je s1 a s2}
  n2:=Getkey(s2);                 {... tu moze byt pouzita obycajna tabulka}
  if T[n1].i=0 then                                {ak sme takeho este nemali}
   begin                                            {tak ho nazveme a je n-ty}
     inc(n); T[n1].i:=n; T[n1].s:=s1; A[n].meno:=s1;
   end;
  n1:=T[n1].i;
  if T[n2].i=0 then
   begin inc(n); T[n2].i:=n; T[n2].s:=s2; A[n].meno:=s2; end;
  n2:=T[n2].i;

  inc(A[n1].pkomu);                    {zvacsime pocet komu hovori spravy n1}
  inc(A[n2].podkoho);                   {zvacsime pocet od koho ma spravy n2}
  A[n2].Inf[A[n2].podkoho]:=n1;     {n1 zaradime do zoznamu informatorov n2}
 end;

procedure Insert(i:integer);              {i-teho spiona pridavame do fronty}
 begin
  inc(nq); Q[nq]:=i;
 end;

procedure Delete(i:integer);              {i-teho spiona vypisujeme a rusime}
var k:integer;
 begin
  for j:=1 to A[i].podkoho do begin
   k:=A[i].Inf[j];
   dec(A[k].pkomu);                                        {zrusime jeho hrany}
   if A[k].pkomu=0 then Insert(k);  {a ak tym padom je niekto adept na sefa}
  end;
  writeln(A[i].meno);
 end;

begin
 n:=0;
 while not eof do begin
  s1:=''; read(c);
  while c<>' ' do begin s1:=s1+c; read(c); end;{mena spionov su jednoslovne}
  readln(s2);
  Zarad(s1,s2);
 end;
 nq:=0;                           {zaciname vytvarat frontu najvacsich sefov}
```

```
 for i:=1 to n do
  if A[i].pkomu=0 then Insert(i);
 i:=1;
 while i<=nq do
  begin Delete(Q[i]); inc(i); end;                {kym nie je prazdna fronta}
 if nq<>n then writeln('nema riesenie'); {skoncili sme a este neboli vsetci}
end.
```

opravovalo YoYo

3. Zvláštne poznámky

(max. 15 bodov)

Kleofášove poznámky sa predsa len podarí rozlúštiť a to hlavne vďaka vám. Prišlo nám niekoľko rýchlych riešení, niekoľko neefektívnejších, nuž a ako býva zvykom, aj niekoľko zlých. Po opravení a obodovaní si bývalí Kleofášovi asistenti môžu vybrať ten najlepší program[3]. Tých niekoľko rýchlych riešení, ktoré sčítavali priamo vo faktoriálovej sústave, si zaslúžilo 15 bodov, neefektívnejšie, ktoré previedli číslo do desiatkovej sústavy, sčítali a previedli naspäť mohli získať 12 bodov. Nesprávne, prípadné iné riešenia sa bodovali podľa zásluh a počasia. No a samozrejme ako vždy sa body strácali za chýbajúce popisy a iné.

A ako sa teda sčituje vo faktoriálovej sústave? No predsa skoro úplne rovnako ako v ľubovoľnej inej sústave. Len jednu konštantu nahradíme premennou[4]. Ale pekne po poriadku. Číslo N si budeme pamatať v poli od 0 po `MAXN`. `A[0]` bude dĺžka čísla (teda miesto, od ktorého sú už vyššie rády nulové). No a `A[i]` $= a_i$, $0 < i \leq m$, pričom $N = \sum_1^m a_i \cdot i! = a_1 \cdot 1! + a_2 \cdot 2! + \cdots + a_m \cdot m!$. Sčítavať budeme pekne po jednotlivých rádoch, začínajúc najnižším (teda 1!). P bude prenos do vyššieho rádu, na začiatku teda 0. Ak $a_i + b_i + P \leq i$, tak je všetko v poriadku, položme $c_i = a_i + b_i + P$. Ak $a_i + b_i + P > i$, tak nastáva prenos. Existujú teda k a l také, že $a_i + b_i + P = (i+1)k + l$, $l \leq i$ [5]. Koľko prispievajú do súčtu i-te cifry aj s prenosom? Je to:

$$(a_i + b_i + P)i! = k \cdot (i+1)i! + l \cdot i! = k \cdot (i+1)! + l \cdot i!$$

Čo znamená, že k bude nový prenos a l bude i-ta cifra súčtu (teda $c_i = l$). A ako rozumne zistiť k a l? No predsa $k = (a_i + b_i + P)$ div $(i+1)$ a $l = (a_i + b_i + P)$ mod $(i+1)$.

Tááák, to by bolo, už vieme, ako sa sčituje vo faktoriálovej sústave a možeme napísať pekný, rýchly program. Alebo si pozrieť vzorák...

Listing programu:

```
program Faktorialova_sustava;
const MAXN=100;
type tcislo = array[0..MAXN] of word;
var A,B,C:tcislo;
    S1,S2,S:string;

function max(a,b:integer):integer;                              {no comment}
begin if a<b then max:=b else max:=a; end;

procedure scitaj(var A:tcislo; var B:tcislo; var C:tcislo);
var i,P:integer;
```

[3] čo je samozrejme náš vzorový

[4] dobre, uznávam, nie je to premenná, ale premenná zvýšená o 1

[5] v našom prípade, keďže $a_i \leq i$, $b_i \leq i$ a $P \in \{0,1\}$, tak $a_i + b_i + P \leq 2i + 1$ a teda $k = 1$ (ide vlastne o delenie so zvyškom)

```
begin
  P:=0;                                          {na zaciatku prenos nie je}
  for i:=1 to max(A[0],B[0]) do begin
    C[i]:=(P+A[i]+B[i]) mod (i+1);                          {scitame cifry}
    P:=(P+A[i]+B[i]) div (i+1);        {kazdych i+1 navyse znamena 1 prenos}
  end;
  if P<>0 then begin
    C[max(A[0],B[0])+1]:=P;        {nezabudnut poslednu cifru ak bol prenos}
    C[0]:=max(A[0],B[0])+1;          {a spravne nastavit dlzku noveho cisla}
  end else
    C[0]:=max(A[0],B[0]);
end;

procedure StrToCis(var S:string; var C:tcislo);
var i:integer;                                  {konvertuje cislo zo stringu}
begin                                                            {do tcislo}
  FillChar(C,SizeOf(C),0);
  C[0]:=Length(S);
  for i:=C[0] downto 1 do
    case S[i] of
      '0'..'9': C[C[0]-i+1]:=Ord(S[i]) - Ord('0');
      'A'..'Z': C[C[0]-i+1]:=Ord(S[i]) - Ord('A')+10;
   end;
end;

procedure  CisToStr(var C:tcislo; var S:string);
var i:integer;                               {konvertuje z tcislo do stringu}
begin
  S:='';
  for i:=C[0] downto 1 do
    if C[i]<10 then S:=S+Chr(Ord('0')+C[i])
               else S:=S+Chr(Ord('A')+C[i]-10);
end;

begin
  Write('Prve cislo: ');
  Readln(S1);
  Write('Druhe cislo: ');
  Readln(S2);
  StrToCis(S1,A);
  StrToCis(S2,B);
  Scitaj(A,B,C);
  CisToStr(C,S);
  Writeln('Sucet je: ',S);
end.
```

4. Zabudnutá pekáreň

opravoval Rišo
(max. 15 bodov)

Tento príklad patril medzi tie jednoduchšie (aj keď niektorí z vás si možno myslia niečo iné `:-)`). Svedčí o tom aj skutočnosť, že takmer všetci z vás, ktorí tento príklad riešili, ho vyriešili v zásade správne.

A teraz k bodovaniu: Za úplne správny program s časovou zložitosťou $O(n^2)$ ste mohli získať max. 15 bodov, za program s časovou zložitosťou $O(n^3)$ ste mohli získať max. 10 bodov a za nefunkčný program max. 5 bodov.

Vašou najčastejšou chybou bolo, že ste si neuvedomili, že riešenie nemusí vždy existovať. Napríklad, ak je počet chlebov rovný 2 a tieto chleby sú v opačnom poradí, ako majú byť. Podobná situácia môže nastať aj pre počet chlebov rovný 3. Až pre 4 a viac chlebov riešenie vždy existuje. Za neošetrenie situácie, keď riešenie neexistuje ste mohli stratiť 3 body.

Ďalej ste mohli stratiť 1 bod za chýbajúci, resp. nesprávny odhad časovej a pamäťovej zložitosti, 2 body za chýbajúci popis, a 1–3 body za chyby v programe a neelegantné konštrukcie (napríklad časti programu, ktoré sa nikdy nemohli vykonať, používanie premennej cyklu po jeho skončení v pascale, neprehľadné používanie globálnych premenných v procedúrach a pod.).

Myšlienka vzorového riešenia je jednoduchá. Najskôr nájdeme chlieb s najmenšou chrumkavosťou a niekoľkými operáciami ho presunieme na koniec radu, teda na miesto, na ktorom sa má nachádzať. Potom na svoje miesto presunieme chlieb s druhou najmenšou chrumkavosťou, atď. Tento postup opakujeme až po chlieb s treťou najväčšou chrumkavosťou.

Po aplikovaní tohoto postupu sú všetky chleby na svojom mieste, dva chleby s najväčšou chrumkavosťou však môžu byť vymenené. V prípade, že nastane takáto situácia, prvé dva chleby môžeme vymeniť postupnosťou operácií `(2,3)`, `(2,3)`, `(1,4)`. Túto postupnosť však môžme vykonať len vtedy, ak sú v rade aspoň 4 chleby. V opačnom prípade úloha zjavne nemá riešenie, lebo cyklickým posúvaním troch chlebov dva z nich nikdy nevymeníme.

Teraz sa zamyslime, ako presunúť daný chlieb na svoje miesto. Pretože v zadaní sa nepožaduje, aby bol počet vykonaných operácií minimálny, nemusíme si lámať hlavu nad tým, ako to urobiť čo najrýchlejšie. Namiesto toho je lepšie napísať čo najjednoduchší program (je menšia pravdepodobnosť, že niekde spravíme chybu). Keď presúvame na svoje miesto chlieb s chrumkavosťou i, všetky chleby s menšou chrumkavosťou už sú na svojom mieste na pozíciách $n-i+2, \ldots, n$. Chlieb i potrebujeme presunúť na pozíciu $n-i+1$. Budeme ho teda posúvať iba doprava. Pri tomto posúvaní nám nezáleží, čo sa stane s ostatnými chlebmi na pozíciách $1, \ldots n-i+1$. Dôležité je len to, aby sme nezmenili polohu chlebov s menšou chrumkavosťou ako i.

Nech j je poloha chleba s chrumkavosťou i. Dokiaľ $j < n-i+1$, budeme opakovať takýto postup: Ak $j > 1$, vykonáme operáciu `(`$j-1$`,3)`. Tým posunieme chlieb s chrumkavosťou i o 1 pozíciu doprava. Naše predpoklady zabezpečujú, že táto operácia je korektná. Ak $j = 1$, vykonáme operáciu `(1,3)`. Tým tiež posunieme chlieb s chrumkavosťou i o 1 pozíciu doprava. Pretože i je nanajvýš $n-2$, vykonaná operácia je korektná.

Takýto algoritmus nájde korektné riešenie vždy, keď existuje. V opačnom prípade zistí, že neexistuje. Okrem konštantného počtu pomocných premenných používame jedno pole pre chrumkavosti chlebov dĺžky N, teda pamäťová zložitosť je $O(N)$, kde N je počet chlebov. Na svoje miesto musíme dopraviť $N-2$ chlebov. Jeden chlieb dopravíme na jeho miesto pomocou $O(N)$ operácií. Nakoniec na prípadnú výmenu prvých dvoch chlebov potrebujeme už len konštantný počet operácií. Celková časová zložitosť algoritmu je teda $O(N^2)$.

Listing programu:

```
program Pekaren;
var A:array[1..100] of integer;     { Pole chrumkavosti chlebikov }
    n,i,j:integer;

procedure Posun(z,l:integer);       { Vykonanie jednej operacie }
var i,p:integer;
begin
  writeln('(',z,',',l,')');
  p:=A[z+l-1];
  for i:=z+l-2 downto z do A[i+1]:=A[i];
  A[z]:=p;
end;

begin
  read(n);                          { Nacitanie vstupu }
  for i:=1 to n do read(A[i]);

  for i:=1 to n-2 do                { Pre vsetky chrumkavosti }
    begin                           { okrem najvacsich dvoch }
      j:=1;
      {Najdi, kde sa chleba s touto chrumkavostou nachadza}
      while A[j] <> i do inc(j);
      while j < n-i+1 do           {Dokial sa nenachadza na spravnom mieste}
        begin
          if j = 1 then Posun(j,3)    {Posun chleba o jednu poziciu doprava}
                   else Posun(j-1,3);
          inc(j);
        end;
    end;
  if A[1] < A[2] then  {Ak su dva chleby s najvacsou chrumkavostou vymenene}
    if n < 4 then writeln('Riesenie neexistuje!')
             else begin                 {Ak existuje, vymen prve dva chleby}
                    Posun(2,3);
                    Posun(2,3);
                    Posun(1,4);
                  end;
end.
```

5. Zblúdilá Zeofína

opravovala Janka
(max. 15 bodov)

Takmer všetky riešenia mali myšlienku podobnú vzorovému riešeniu, za ktorú sa dalo získať max. 15 bodov. Pár bodov sa dalo stratiť za neošetrenie špeciálnych prípadov, nesprávne použitie arcus tangensu, a samozrejme za chýbajúci či nezrozumiteľný popis riešenia.

Budeme postupne podľa príkazov (`DOPREDU`, `DOPRAVA`, `DOLAVA`) počítať súradnice Zeofíny a uhol, v ktorom je natočená. Na začiatku Zeofínka sedí na súradniciach $x = 0$, $y = 0$ a uhol natočenia je nulový. Ak prečítame príkaz `DOPRAVA` alebo `DOLAVA`, zmeníme Zeofíne uhol natočenia o tento uhol. Ak prečítame príkaz `DOPREDU h`, vypočítame nové súradnice podľa vzťahov:

$$x_{nove} = x_{stare} + h \cdot \cos(uhol)$$
$$y_{nove} = y_{stare} + h \cdot \sin(uhol)$$

Uhol, ktorý zviera polpriamka idúca z bodu $[0,0]$ do bodu $[x,y]$ s kladnou poloosou osi x (α) vypočítame zo vzťahu:

$$\alpha = \text{arctg}\,(y/x)$$

Na základe neho vypočítame uhol, o ktorý sa má Zeofína otočiť. Pri programovaní je to trochu zložitejšie, treba si dať pozor na rôzne detaily, ako napríklad špeciálne ošetriť prípad, keď $x = 0$ (vtedy by sme delili nulou) a takisto sa nie vždy dá veriť funkciám rátajúcim arctan, že robia to, čo majú. V tomto smere to mali jednoduchšie riešitelia, píšuci v C-čku. Tam existuje funkcia `atan2`, ktorá funguje pre ľubovoľné y a ľub. $x \neq 0$ a vráti uhol v radiánoch z intervalu $(-\pi, \pi\rangle$. V Pascale funkcia `arctan` nie je natoľko spoľahlivá. Najistejšie riešenie je zistiť si pomocou nej uhol β, ktorý zviera s kladnou poloosou osi x polpriamka z $[0,0]$ do $[|x|,|y|]$ a potom rozlíšiť 4 prípady podľa toho, v ktorom kvadrante pôvodný bod $[x,y]$ ležal. (Ak v prvom, $\alpha = \beta$, ak v druhom, $\alpha = \pi - \beta$ atď.) Takto sme dostali uhol v radiánoch, ten ešte treba premeniť na stupne prenásobením $180/\pi$.

Nakoniec podľa Pytagorovej vety vypočítame vzdialenosť Zeofínky od domu (d):

$$d = \sqrt{x^2 + y^2}$$

Listing programu:

```
program Zbludila_zeofina;
var
 x,y,uhol,otoc,alfa,h:real;
 ch:char;
 s:string;
 f:Text;

Begin
 x:=0;
 y:=0;
 uhol:=0;
 alfa:=0;
 otoc:=0;
 assign(f,'zeofina.in');
 reset(f);
 while not eof(f) do
 begin
   s:='';
   read(f,ch);
   while ch<>' ' do begin s:=s+ch; read(f,ch); end;
   readln(f,h);
   if s ='DOPREDU' then
      begin
        x:=x+h*cos(pi*uhol/180);
        y:=y+h*sin(pi*uhol/180);
      end;
   if s ='DOPRAVA' then
      begin
```

```
        uhol:=uhol-h;
        while uhol<0 do uhol:=uhol+360;
        while uhol>=360 do uhol:=uhol-360;
      end;
   if s ='DOLAVA'  then
      begin
        uhol:=uhol+h;
        while uhol<0 do uhol:=uhol+360;
        while uhol>=360 do uhol:=uhol-360;
      end;
 end;

 if x=0 then
 begin
   if y>0 then otoc:=uhol+90;
   if y<0 then otoc:=uhol-90;
   if y=0 then otoc:=0;        {Zeofina je uz doma}
 end
 else
 begin
   alfa:=arctan(abs(y/x));
   alfa:=alfa*180/pi;
   if (x>0) and (y>=0) then otoc:=uhol+180-alfa;
   if (x<0) and (y>=0) then otoc:=uhol+alfa;
   if (x<0) and (y<0)  then otoc:=uhol-alfa;
   if (x>0) and (y<0)  then otoc:=uhol+alfa+180;
 end;
 if otoc>=360 then otoc:=otoc-360;
 if otoc<0 then otoc:=otoc+360;
 writeln('DOPRAVA ',otoc:3:2);
 writeln('DOPREDU ',sqrt(x*x+y*y):3:2);
 close(f);
End.
```

Výsledková listina po 4. sérii kategórie KSP-Z

	Meno a priezvisko	kola	Trieda		41	42	43	44	45	Σ
1.	Pavol Mravec	3	GModra	67	15	14	15	15	15	141
2.	Tomáš Vrábel	3	GTótMT	67	15	13	14	12	15	136
3.	Ján Katrenič	3	GŠkSNV	64	15	13	15	12	12	131
4.	Jakub Kováč	1	GJH	56	11	13	15	10	15	120
	Oto Vozár	2	GPalBA	60	15	12	12	7	14	120
6.	Pavol Trenkler	1	GZbrKE	51	10	12	15	11	15	114
7.	Peter Libič	2	GTNĽŠ	52	4	11	15	14	15	111
8.	Štefan Konečný	2	GJH	56	14	11	15		10	106
9.	Lucia Tiererová	3	GJH	54	1	11	14	11	8	99
10.	Ján Palenčár	1	GTótMT	43	1	11	14	9	13	91
11.	Anton Štefanek	1	GJH	53	11		15	10		89
	Rasťo Šrámek	3	GTilBA	40	4	12	12	7	14	89
13.	Filip Legény			49	14	11			11	85
14.	Milan Burda	1	GGolNT	44	5	4	14	4	12	83
15.	Ján Veselý	1	GGolNT	46	4	3	12		11	76
16.	Miroslav Baláž	1	GJH	0	15	14	15	14	15	73
	Peter Čunderlík	1	GJH	37	13	8			15	73
	László Marák	2	GSmKom	39	5	5	12		12	73
19.	Michal Chudý	3	GLevic	40	4		12		12	68
20.	Anna Hanulová	1	GJH	29	14	12		12		67
21.	Martin Choma	2	GStĽub	65						65
22.	Juliana Lipková	2	GJH	26	1	4	15		15	61
23.	Pavol Cvik	2	GLevic	38	5		3		13	59
24.	Matej Max	3	GAlKE	51						51
	Juraj Sloboda			51						51
26.	Monika Steinová	3	GEinBA	49						49
27.	Tomáš Mikuš	2	GJH	25	9				13	47
28.	Róbert Kanász	2	GDneKE	46						46
29.	Marek Dovjak	3	GKežm	44						44
	Pavol Brilla	1	GŠroKE	44						44
31.	Filip Karas	1	GGolNT	43						43
32.	Tibor Blénessy	1	GPošKE	42						42
33.	Tomáš Kováč	1	GJH	0	12		14		15	41
34.	Peter Ambroz	1	GJH	38						38
35.	Michal Kavka	3	GAlKE	35						35
36.	Vesall Nourani	1	GJH	0	4		11	8	10	33
37.	Ján Dinga		GJH	21	4		6			31
38.	Jana Macsadiová		GJH	10	10	10				30
39.	Vlado Tomeček	2	GJH	0	5	10			14	29
	Rastislav Gregor	2	GVOZA	25	4					29
	Oliver Janík	1	GLSBar	12	5		12			29
42.	Jakub Tekeľ	1	GJH	0			15		13	28
43.	Eva Szarková	4	GJH	27						27
44.	Richard Štefanec	1	GJH	25						25
	Gabriel Karády	1	GŠroKE	25						25
46.	Maroš Vranec	2	GAlKE	24						24
	Ivan Trejbal	1	GŠroKE	24						24
48.	Marián Schmotzer	1	GHorBA	23						23
	Jana Vašíčková		GJH	14	9					23
50.	Peter Kálnai	3	GLevic	21						21

	Meno a priezvisko	kola	Trieda		41	42	43	44	45	Σ
51.	Michal Kramarič	1	GJH	20						20
	Matúš Dekánek	1	GJH	8	4				8	20
53.	Barbora Trubeňová	1	GJH	0	10			8		18
54.	Rastislav Halamíček	1	GJH	13	4					17
55.	Ivor Kollár	2	GJH	16						16
	Martin Demko		GAlKE	16						16
57.	Sana Nourani		GJH	15						15
	Pavol Kocsis	3	GAlKE	15						15
59.	Ľubica Chriašteľová	1	GJH	0			14			14
	Dana Smažáková	1	GJH	0	4			10		14
61.	Dominika Bartová	1	GJH	0			13			13
	Slavomír Sihelník	3	GAlKE	9	4					13
63.	Peter Harvan	1	GJH	12						12
	Ján Dojčár		GTurTe	12						12
65.	Radovan Čulák	1	GGolNT	11						11
	Tibor Óbert	3	GLevic	11						11
	Milan Pavlík		SPŠDBA	11						11
68.	Kamil Kuboň	1	GJH	0	10					10
	Michal Širáň		GJH	0	4		6			10
	Jana Lajdová		GVarZA	10						10
	Karol Takáč		GAlKE	10						10
	Pavol Király	3	GAlKE	10						10
	Peter Dučai	3	GAlKE	10						10
74.	Tomáš Pásztor	4	GŠahy	9						9
	Ladislav Hodermarský	3	SSÁ	9						9
76.	Filip Gschwandtner	1	GJH	0	4				2	6
77.	Ondrej Vavro	1	GJH	0	5					5
78.	Martin David	2	GJH	0	4					4
	Štefan Olejník	1	GJH	0	4					4
	Martin Vaško	2	GJH	0	4					4
	Kristian Danev	2	GJH	0	4					4
82.	Martin Vataha	4	GSnina	0						0
	Edo Drobný	2	GJH	0	0					0