**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.*

# Vzorové riešenia 4. kola

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. O chemických zlúčeninách

opravoval Goober
(max. 15 bodov)

Súdiac podľa počtu vašich riešení som nadobudol pocit, že tento príklad patril medzi tie ťažšie. Veľmi ma však potešil fakt, že všetky riešenia boli správne. Preto sa udelené body pohybovali v rozmedzí 14–15, pričom ten jeden bod bolo možné stratiť za nepresnosti v dôkaze.

A ako sa to malo robiť? Budeme riešiť konkrétny prípad – budeme predpokladať, že máme $N$ atómov a každé dva sú spojené (ak nájdeme vhodné rozloženie atómov pre túto úlohu, môžeme ho použiť pre ľubovoľnú molekulu s $N$ atómami, bez ohľadu na ich pospájanie). Jedno vyhovujúce rozloženie atómov je také, že ležia na krivke $K = \{[t, t^2, t^3] \mid t \in R\}$.

Lema: Pre krivku $K$ platí, že žiadna rovina (resp. priamka) nemá s $K$ viac ako 3 (resp. 2) priesečníky.

Dôkaz: Rovina je v priestore popísateľná všeobecnou rovnicou tvaru $aX + bY + cZ + d = 0$. Potom priesečníky tejto roviny a $K$ spĺňajú rovnicu $at + bt^2 + ct^3 + d = 0$. Toto je kubická rovnica, a preto má nanajvýš 3 riešenia v obore reálnych čísel, čiže dostávame nanajvýš tri priesečníky s $K$. Aby sme dokázali druhú časť tvrdenia (tú o priamke), spravíme projekciu do roviny $XY$ (to znamená, že bod $[x, y, z]$ sa zobrazí na bod $[x, y, 0]$). Pri tejto projekcii platí, že dva rôzne body krivky $K$ sa zobrazia na dva rôzne body v rovine $XY$, čiže ak sa nejaká priamka pretína s krivkou $K$ v niekoľkých bodoch, bude sa jej obraz v projekcii (čo je priamka alebo bod) pretínať s obrazom $K$ v rovnakom počte bodov. Obrazom $K$ je však parabola, čiže krivka druhého stupňa, a tá má s ľubovoľnou priamkou (a teda nepochybne aj s bodom) nanajvýš dva priesečníky. Preto aj krivka $K$ má s ľubovoľnou priamkou nanajvýš dva spoločné body.

Nech $A$, $B$, $C$, $D$ sú štyri rôzne body $K$. Dokážeme, že ich spojnice $AB$ a $CD$ (dokonca aj keď o nich budeme uvažovať ako o priamkach) nemajú spoločný bod. Sú dve možnosti, ako by mohli priamky $AB$, $CD$ mať spoločný bod – buď sú rôznobežné, alebo totožné. Ak $AB$, $CD$ sú rôznobežné, ležia body $A$, $B$, $C$, $D$ v jednej rovine. To je však v spore s lemou, preto tento prípad nenastane. Druhá možnosť je, že $AB$, $CD$ sú totožné, čiže $A$, $B$, $C$, $D$ ležia na tej istej priamke (tu povolíme aj prípad, že sú niektoré dva body z $A$, $B$, $C$, $D$ totožné). Máme teda minimálne tri spoločné body krivky $K$ a priamky. Opäť máme spor s

lemou, takže ani tento prípad nenastane. Čiže, ak umiestnime atómy na krivku $K$, žiadne ich dve spojnice sa nebudú pretínať.

Implementácia: Celý program pozostáva z načítania $N$ a výpisu $N$ bodov ležiacich na $K$ so súradnicami $[i, i^2, i^3], i = 1..N$. Časová zložitosť je $O(N)$, pamäťová $O(1)$.

### Listing programu:

```
int main() {
  int n,i;
  scanf("%d",&n);
  for(i=1;i<=n;i++) printf("%d. (%d,%d,%d)\n",i,i,i*i,i*i*i);
}
```

## 2. O nepríjemnostiach

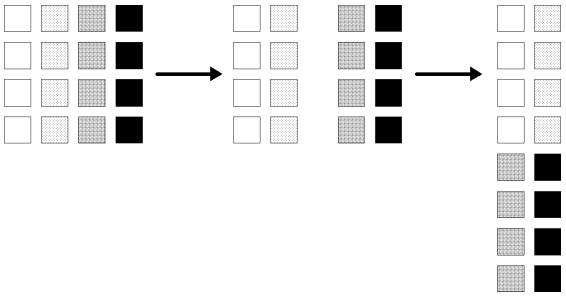
opravoval MišoF
(max. 15 bodov)

Na úvod pár viet o názvosloví. Pole zo vstupu budeme volať matica a jeho (jej) preklopenie podľa osi $y = x$ budeme volať transponovanie. Takže k vzorovému riešeniu:

Nie je problém transponovať jeden stĺpec – len prechádzame starým súborom, čítame čísla v stĺpci a píšeme ich do nového súboru do riadku. Takže nie je problém úlohu vyriešiť v čase $O(N^3)$ – postupne budeme transponovať stĺpce pôvodnej matice. Nešlo by to ale lepšie? Zjavne nám nezlepší zložitosť, keby sme transponovali dva či tri stĺpce naraz. Ale čo takto ich transponovať všetky naraz?

Myšlienka vzorového riešenia bude nasledovná: Naukladáme všetky stĺpce v nejakom poradí pod seba. Tým dostaneme jeden stĺpec, ktorý vieme ľahko transponovať v čase $O(N^2)$. Dostaneme jeden riadok, v ktorom sú v takom istom poradí riadky transponovanej matice. No a potom akoby opačným postupom popresúvame riadky na správne miesta a vyhrali sme. Teda vyhrali sme, ak vieme spraviť tie presuny v čase lepšom ako $O(N^3)$.

Predpokladajme najskôr pre jednoduchosť, že $N = 2^M$ pre nejaké $M$. Predstavme si, že vezmeme našu maticu, podľa zvislej osi ju „rozstrihneme“ napoly a pravú polovicu presunieme pod ľavú. Tým dostaneme obdĺžnik, v ktorého prvom stĺpci budú pod sebou prvý a $(2^{M-1} + 1)$-vý stĺpec pôvodnej matice, atď. až v poslednom stĺpci budú $2^{M-1}$-tý a $2^M$-tý stĺpec.

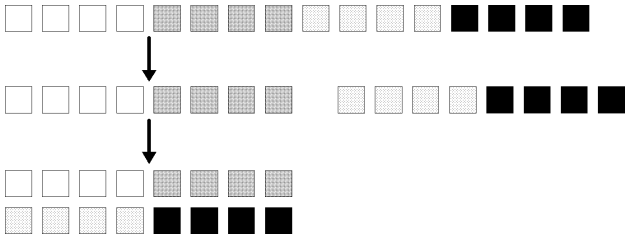


Tento proces urobíme $M$-krát (teda vždy zoberieme obdĺžnik, prestrihneme ho napoly po zvislej osi a pravú časť presunieme pod ľavú). Jedno vykonanie procesu vieme spraviť na dva prechody súborom – v prvom píšeme do nového súboru prvé polovice riadkov, v druhom ich druhé polovice pod ne. Nakoniec starý súbor nahradíme novým. Jeden proces teda trvá $O(N^2)$, čiže celá táto časť algoritmu beží v čase $O(MN^2) = O(N^2 \log N)$.

Tým sme dostali jeden stĺpec, v ktorom sú v nejakom poradí pod sebou stĺpce pôvodnej matice. Tento stĺpec ľahko transponujeme, čím dostaneme riadok, obsahujúci v tom istom poradí riadky transponovanej matice.

Ako ju z nich zostrojiť? Prekvapujúco jednoducho – opäť spravíme $M$-krát vyššie popísaný proces. Ľahko sa dá indukciou dokázať, že po $K$ krokoch procesu dostaneme obdĺžnik,

ktorý bude transpozíciou obdĺžnika, ktorý vznikol z pôvodnej matice $M - K$ krokmi. Teda po $M$ krokoch dostaneme transponovanú pôvodnú maticu, čo sme chceli dosiahnuť.



Celý tento algoritmus teda zjavne beží v čase $O(N^2 \log N)$. Ostáva už len doriešiť prípad, keď $N$ nie je mocnina dvoch, teda keď $2^{M-1} < N < 2^M$ pre nejaké $M$. V tomto prípade stačí použiť drobnú fintu: Doplníme našu maticu nulami na maticu veľkosti $2^M \times 2^M$ a pustíme na ňu vyššie uvedený algoritmus, aby sme sa nemuseli zaoberať okrajovými prípadmi, keď by bola strana obdĺžnika nepárna. No a keďže $2^M < 2N$, časovú zložitosť nám to nezhorší (budeme riešiť úlohu pre najviac dvakrát tak veľkú maticu). Takže sme dostali algoritmus, ktorý beží v čase $O(N^2 \log N)$, konštantnej pamäti, na disku používa miesto $O(N^2)$ a potrebuje konštantne veľa pomocných súborov.

Existujú aj iné prístupy k riešeniu tejto úlohy. Skúste si napríklad predstaviť, že by ste mali štvorcovú maticu a tú rozstrihli na štyri časti po vodorovnej a zvislej osi. Teraz už len stačí každú z nich transponovať a vymeniť ľavú dolnú s pravou hornou.

Tradične na záver, ako som vaše riešenia hodnotil?

- Vzorové riešenia – max. 15 bodov
- Riešenia s rovnakou časovou zložitosťou, ale zaberajúce viac miesta na disku/potrebujúce viac pomocných súborov – max. 13 bodov
- Riešenie potrebujúce čas $O(N^2)$, ale za cenu toho, že musí mať naraz otvorených $N$ súborov – 12 bodov
- Riešenie využívajúce $N^2$ súborov – 10 bodov
- Riešenia, ktoré celú úlohu odbili s tým, že `seek` má konštantnú časovú zložitosť. To nie je tak úplne pravda. Zložitosť seeku je úmerná veľkosti súboru, prípadne sektoru na disku a hlavne prudko závisí od operačného systému, pod ktorým svoj program púšťate. V úlohách takéhoto typu sa preto berie časová zložitosť seeku ako lineárna od veľkosti súboru, v lepšom prípade lineárna od vzdialenosti, o ktorú sa v súbore presunulo miesto, odkiaľ čítame. V KSP sa snažíme klásť dôraz na hľadanie algoritmov, teda postupov, ktorými sa úloha dá riešiť, a nie na využívanie konkrétnych fint konkrétnych programovacích jazykov v konkrétnych operačných systémoch. Preto takéto riešenia mohli získať najviac 8 bodov.
- Riešenia, ktoré potrebovali čas $O(N^3)$ a viac – max. 7 bodov
- Nesprávne riešenia – max. 2 body podľa toho, či sa myšlienka aspoň podobala na riešenie a mala hlavu a pätu.

Body sa dali stratiť za chýbajúci/zlý odhad časovej zložitosti algoritmu, za chýbajúci, resp. nezrozumiteľný popis a samozrejme za (nájdené) chyby v programe.

## Listing programu:

```
program O_neprijemnostiach;
var N,oldN,logN,i,width : integer;

procedure NextPower2(x : integer; var p2 : integer; var logp2 : integer);
begin
   p2:=1; logp2:=0;
```

```
   while p2<x do begin p2:=2*p2; inc(logp2); end;
end;

procedure UpravVstup;
var i,j,k : integer;
    f,g : text;
begin
   assign(f,'matica.in'); assign(g,'temp1.txt'); reset(f); rewrite(g);
   readln(f,oldN); NextPower2(oldN,N,logN);
   for i:=1 to oldN do begin
      for j:=1 to oldN do begin read(f,k); write(g,k,' '); end;
      for j:=oldN+1 to N do write(g,'0 ');
      writeln(g);
   end;
   for i:=oldN+1 to N do begin
      for j:=1 to N do write(g,'0 '); writeln(g);
   end;
   close(f); close(g);
end;

procedure Process(sirka : integer);
var f,g : text;
    i,j,k : integer;
begin
   assign(f,'temp1.txt'); assign(g,'temp2.txt'); rewrite(g);
   reset(f);
   while not eof(f) do begin
      for i:=1 to sirka div 2 do begin read(f,k); write(g,k,' '); end;
      for i:=1 to sirka div 2 do read(f,k);
      readln(f); writeln(g);
   end;
   close(f); reset(f);
   while not eof(f) do begin
      for i:=1 to sirka div 2 do read(f,k);
      for i:=1 to sirka div 2 do begin read(f,k); write(g,k,' '); end;
      readln(f); writeln(g);
   end;
   close(f); erase(f); close(g); rename(g,'temp1.txt');
end;

procedure TransposeColumn;
var f,g : text;
    k : integer;
begin
   assign(f,'temp1.txt'); assign(g,'temp2.txt');
   rewrite(g); reset(f);
   while not eof(f) do begin
      readln(f,k); write(g,k,' ');
   end;
   writeln(g);
   close(f); close(g);
```

```
   erase(f); rename(g,'temp1.txt');
end;

procedure UpravVystup;
var f,g : text;
    i,j,k : integer;
begin
   assign(f,'temp1.txt'); assign(g,'matica.out');
   reset(f); rewrite(g);
   for i:=1 to oldN do begin
      for j:=1 to oldN do begin read(f,k); write(g,k,' '); end;
      readln(f); writeln(g);
   end;
   close(f); close(g);
end;

begin
   UpravVstup; width:=N;
   for i:=1 to logN do begin
      Process(width); width:=width div 2;
   end;
   TransposeColumn;
   width:=N*N;
   for i:=1 to logN do begin
      Process(width); width:=width div 2;
   end;
   UpravVystup;
end.
```

opravovalo YoYo $\delta$
(max. 15 bodov)

## 3. O elixíroch

Podľa počtu riešení sa dá usudzovať, že miešané nápoje a tobôž elixíry nemáte v láske. A z toho mála bol iba jeden program správny. Bol odmenený 15 bodmi. Našlo sa ešte niekoľko popisov správneho lineárneho riešenia, ale bez programu. Tie si zaslúžili zopár bodíkov. No a ostali programy riešiace sústavy rovníc, bohužial ale neúspešne.

Vzorové riešenie: Najprv si treba uvedomiť, že ku každej fľaštičke nám stačí pamätať si iba dve čísla. Stačí, ak kažú zložku predelíme súčtom pôvodných 3 čísel. Z ľubovoľných dvoch potom vieme dopočítať tretiu (ich súčet musí byť 1). Teraz si tieto dvojice čísel predstavme ako body v rovine. Nech $A_i$ je bod prislúchajúci fľaštičke $i$. Aké elixíry vieme namiešať z fľaštičiek $i$ a $j$? Sú to elixíry prislúchajúce bodom ležiacim na úsečke $A_iA_j$. Pre tri body (fľaštičky) je to už celý trojuholník. Po chvíľke zamyslenia prídeme na to, že vieme namiešať ľubovoľný elixír, ktorého bod leží v nejakom trojuholníku $A_iA_jA_k$. Presnejšie teda, ak leží v konvexnom obale bodov $A_1$ až $A_n$. Ako ale v lineárnom čase zistiť, či v ňom daný bod zodpovedajúci nášmu elixíru (označme si ho $B$) leží?

Najprv vezmem ľubovoľné dva body $A_p$ a $A_q$, napríklad $p = 1$ a $q = 2$, a budeme sa zaujímať o uhol $\sphericalangle A_pBA_q$. Potupne budeme prechádzať body $A_3$ až $A_n$ a udržiavať $p$ a $q$ tak, aby všetky body $A_1$ až $A_i$ (kde $A_i$ je práve spracovávaný bod) patrili uhlu $\sphericalangle A_pBA_q$. Ak $A_i \in \sphericalangle A_pBA_q$, tak sa nič nemení. Ak nepatrí, tak posunieme $p$ alebo $q$ na $i$, podľa toho, ktorý z $\sphericalangle A_pBA_i$ alebo $\sphericalangle A_qBA_i$ je menší. Ak by nový uhol bol rovný 180°, tak sme na úsečke, na ktorej bod $B$ leží. Ak by mal byť tento uhol väčší, tak sme našli trojuholník

$A_pA_qA_i$, v ktorom bod $B$ leží. Ak spracujeme všetky body a $\sphericalangle A_pBA_q$ bude stále menší ako $180°$, tak bod $B$ leží mimo konvexného obalu a Zoja má smolu.

V prípade úspechu samozrejme nesmieme zabudnúť vypočítať ešte sústavu troch rovníc o troch neznámych, aby sme získali množstvá, koľko máme z každej fľaštičky odliať. A aby teda bola Zoja spokojná, nasleduje ešte program podľa jediného správneho riešenia Petra Bellu.

## Listing programu:

```
#include <stdio.h>
#include <math.h>
#define N 5
#define PI 3.1415926535897932

typedef struct {
  double z1,z2,z3;
  double x,y,uhol;
} flasa;

int n,c[3];
flasa p[N];

void nacitaj() {
  int i;
  scanf("%d", &n);
  for ( i=0; i<=n; i++ ) {
    scanf("%lf %lf %lf", &p[i].z1, &p[i].z2, &p[i].z3);
    p[i].x=p[i].z1 / (p[i].z1+p[i].z2+p[i].z3);
    p[i].y=p[i].z2 / (p[i].z1+p[i].z2+p[i].z3);
  }

  for ( i=0; i<n; i++ ) {
    p[i].uhol = atan2(p[i].y-p[n].y,p[i].x-p[n].x);
    if (p[i].uhol<0) p[i].uhol+=2*PI;
  }
}

int srot() {
  int i;
  double p1=p[0].uhol,p2=p[1].uhol,p3,p4;

  c[0]=0; c[1]=1;
  p3=p2-p1; if (p3<0) p3+=2*PI;
  if ( p3 == PI ) return 2; // lezia na priamke
  if (p3 > PI ) {p3=p1; p1=p2; p2=p3; c[1]=0; c[0]=1;}

  for ( i=2; i<n; i++ )
  {
    p3 = p[i].uhol-p1; if (p3<0) p3+=2*PI;
    p4 = p2-p[i].uhol; if (p4<0) p4+=2*PI;
    if (p3+p4<PI) continue; // lezi medzi p1 a p2
```

```
    /* posuvam p2 na p[i] */
    p4 = -1*p4; if (p4<0) p4+=2*PI;

    if ( (p4>0) && (p3<PI) ) {p2 = p[i].uhol; c[1]=i; continue;}
    if ( (p4>0) && (p3==PI) ) {c[1]=i; return 2;}

    /* posuvam p1 na p[i] */
    p3 = -1*p3; if (p3<0) p3+=2*PI;
    p4 = -1*p4; if (p4<0) p4+=2*PI;
    if ( (p3>0) && (p4<PI) ) {p1=p[i].uhol; c[0]=i; continue;}
    if ( (p3>0) && (p4==PI) ) {c[0]=i; return 2;}

    /* lezi v konvexnom obale */
    p4 = -1*p4; if (p4<0) p4+=2*PI;
    if ( p3+p4>PI ) {c[2]=i; return 3;}
  }
  return 0;
}

int pomer(int pocet) {
  double p0,p1,p2,p3;
  double a1,b1,c1,a2,b2,c2, d1,d2,d,x,y;
  if (pocet==0) {
    printf("Zoja ma smolu.\n");
    return 0;
  }

  if (pocet==2) {            //lezia na jednej priamke
    if (p[n].x==p[c[0]].x)
         {p0=fabs(p[n].y-p[c[1]].y); p1=fabs(p[n].y-p[c[0]].y);}
    else {p0=fabs(p[n].x-p[c[1]].x); p1=fabs(p[n].x-p[c[0]].x);}

    printf("%d. flaska: %lf dielov\n", c[0]+1,
                               p0/(p[c[0]].z1+p[c[0]].z2+p[c[0]].z3));
    printf("%d. flaska: %lf dielov\n", c[1]+1,
                               p0/(p[c[1]].z1+p[c[1]].z2+p[c[1]].z3));
    return 0;
  }

  if (pocet==3) {
    a1 = p[n].y - p[c[0]].y;
    b1 = p[c[0]].x - p[n].x;
    c1 = -1*(a1*p[n].x+b1*p[n].y);

    a2 = p[c[1]].y - p[c[2]].y;
    b2 = p[c[2]].x - p[c[1]].x;
    c2 = -1*(a2*p[c[1]].x+b2*p[c[1]].y);

    d=a1*b2 - a2*b1; d1=c2*b1 - c1*b2; d2=a2*c1 - c2*a1;
    x=d1/d; y=d2/d; // priesecnik priamok p[n]-p[c[0]] a p[c[1]]-p[c[2]]
```

```
    if (p[n].x==x) {p0 = fabs(p[n].y - y); p1 = fabs(p[n].y - p[c[0]].y);}
    else           {p0 = fabs(p[n].x - x); p1 = fabs(p[n].x - p[c[0]].x);}

    if (p[c[1]].x==x) {p2 = fabs(p[c[2]].y - y); p3 = fabs(p[c[1]].y - y);}
    else              {p2 = fabs(p[c[2]].x - x); p3 = fabs(p[c[1]].x - x);}

    printf("%d. flaska: %lf dielov\n", c[0]+1,
                    p0/(p[c[0]].z1+p[c[0]].z2+p[c[0]].z3) );
    printf("%d. flaska: %lf dielov\n", c[1]+1,
                    (p1*p2)/(p2+p3)/(p[c[1]].z1+p[c[1]].z2+p[c[1]].z3) );
    printf("%d. flaska: %lf dielov\n", c[2]+1,
                    (p1*p3)/(p2+p3)/(p[c[2]].z1+p[c[2]].z2+p[c[2]].z3) );
  }
  return 0;
}

int main() {
  nacitaj();
  pomer(srot());
  return 0;
}
```

## 4. O novom probléme v Chile

opravoval Dávidko
(max. 15 bodov)

Drvivú väčšinu riešení bolo možné rozdeliť do dvoch kategórií. Jedna s časovou zložitosťou $O(N^3)$ – 11 bodov, druhá s časovou zložitosťou $O(N^2)$ a lineárnou pamäťou – 15 bodov. Pár riešení nezapadlo do žiadnej z týchto kategórií. Išlo o jeden backtrack a jedno riešenie bez programu.

Vzorové riešenie: (podľa Pavla Juhosa a Jána Katreniča)
je založené na dynamickom programovaní. Pre každé mesto $i$ ($1 \leq i \leq N$) vyrátame $m_i$ – za koľko najmenej peňazí sa tam vieme dostať a aký lístok ukážeme revízorovi ako posledný (tento lístok bude na cestu z mesta $a_i$ do mesta $b_i$).

Niekoľko drobných pozorovaní: Uvažujme nejakú cestu z mesta 1 do mesta $i$. Predstavme si platnosti nami vybraných lístkov ako prekryvajúce sa intervaly na reálnej osi. Zrejme nemá význam, aby nejaký interval bol podmnožinou iného, taký lístok by sme nemuseli kúpiť a dosiahli by sme lacnejšie riešenie. Potom možno použité lístky prirodzeným spôsobom usporiadať podľa začiatku ich platnosti.[1]

Samotný algoritmus: Na začiatku položíme $m_1 = 0$ a $m_i = \infty$ ($1 < i \leq N$). Potom pracujeme v cykle. V $i$-tom kroku načítame cesty z mesta $i$ do všetkých ostatných miest $j$ ($i < j \leq N$). Pre každé mesto $j$ ($j > i$) zrátame cenu $c$ najlacnejšieho lístka idúceho z mesta $i$ za alebo do mesta $j$. Ak je cesta z $i$ do $j$ v kombinácii s cestou do $i$ lacnejšia ako doteraz najlepšia nájdená $m_j$, tak si ju zapamätáme (samozrejme si zapamätáme aj $a_j = i$ a $b_j = k \geq j$).

Po skončení sa stačí pozrieť na hodnotu $m_N$.

Dôkaz správnosti algoritmu založíme na jednoduchom pozorovaní. Po $i-1$ krokoch cyklu máme vypočítané správne všetky hodnoty $1 \ldots m_i$, $1 \ldots a_i$, $1 \ldots b_i$. Dôkaz spravíme úplnou indukciou vzhľadom na $i$. Pre $i = 1$ je trvdenie hravo dokázané, pretože zrejme $m_1 = 0$, čo je aj pravda. Nech teda dokazované tvrdenie platí pre všetky $j < i$, dokážeme ho aj pre $i$. Jediné, čo vlastne treba dokázať, je, že $m_i$ bude cena najlacnejšej cesty do mesta $i$. Vráťme

[1] alebo podľa konca; dostaneme to isté usporiadanie.

sa preto k našej predstave intervalov. Nech by sme cestovali akokoľvek lacno, museli sme mať kúpený akýsi posledný lístok. Tento posledný lístok sme si museli kdesi kúpiť, nazvime to mesto $j$ ($j < i$). Najlacnejšia cesta sa skladá z najlacnejšej cesty do mesta $j$ a cesty z neho za pomoci spomínaného lístka. Tento lístok je tiež najlacnejší z mesta $j$ do mesta $i$ alebo zaň. Ale my sme v $j$-tom kroku algoritmu túto možnosť uvážili. Navyše v tom čase sme podľa indukčného predpokladu už mali nájdenú najmenšiu cenu $m_j$, za ktorú sa vieme dostať z 1 do $j$.

Implementačné detaily: Každý krok algoritmu spravíme v lineárnom čase. Načítať menej ako $N$ cien lístkov asi nebude problém. Potom ideme odzadu a v premennej $c$ si pamätáme cenu najlacnejšieho lístka idúceho cez alebo do mesta $j$. Potom aktualizujeme $m_j := \min(m_j, m_i + c)$.

Ako si pamätať a vypísať lístky, ktoré treba použiť? Jednoducho pri každej akualizácii ceny $m_j$ si zapíšeme aj $a_j$ a $b_j$. Po skočnení hlavnej časti algoritmu ideme odzadu a vypisujeme lístky.

Časová zložitosť bude zjavne $O(N^2)$ a keďže si pamätáme len jeden riadok vstupu, pamäťová zložitosť bude len $O(N)$.

## Listing programu:

```
Program Chile;
var N,i,j,c,kam:integer;

{ a[i]-b[i] = posledny listok, ktory si na ceste do mesta i kupime
  m[i] = najmensia cena, za ktoru sa da dostat z 1 do i }

  a,b,m:array[1..100] of integer;
  cena:array[1..100] of integer; { to, co je na vstupe }

Begin
  readln(N); { nacitame N }

  for i:=2 to N do m[i]:=maxint; { nekonecno }
  m[1]:=0; { do mesta 1 sa ide samozrejme zadarmo }

  for i:=1 to N-1 do
  begin
    for j:=i+1 to N do read(cena[j]);    { nacitame ceny listkov z mesta i }

    c:=maxint; { nekonecno }
    for j:=N downto i+1 do
    begin                              { najlacenjsi spoj i-kam, pricom kam>=j}
      if cena[j]<c then
      begin
        c:=cena[j];
        kam:=j;
      end;

      if m[i]+c<m[j] then   { ak je to do mesta i + spojom i-kam lacnejsie }
      begin
        m[j]:=m[i]+c;
        a[j]:=i;
        b[j]:=kam;
```

```
      end;
    end;
  end;

  writeln('Najlacnejsie sa da dostat za ',m[N],' korun.');
  writeln('Kupi si listky: ');
  i:=N;
  while (i>1) do  { vypisovanie listkov }
  begin
    writeln(a[i],'-',b[i]);
    i:=a[i];
  end;
End.
```

## 5. O Santolande IV

opravoval Braňo
(max. 15 bodov)

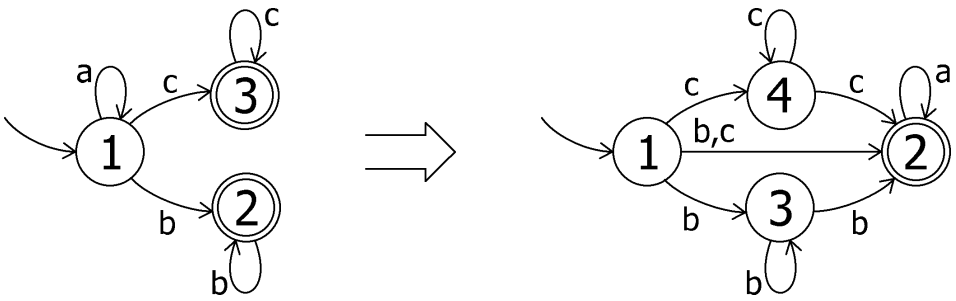
O tento príklad ste neprejavili veľký záujem. Je otázne, či bol až taký ťažký.

Dostaneme na vstupe popis rumovej jaskyne. Chvíľku predpokladajme, že má len jednu rumovú komoru. Vieme, že v každej komore máme iba jednu možnosť, kam sa vydať s konkrétnym písmenkom. Ako sa dá popísať správanie zlatokopa v tejto jaskyni, keď nájde rum? Zlatokop dostane papierik s postupnosťou znakov (slovom). Začne v komore 1. Prvé písmenko slova mu jednoznačne povie, do ktorej komory sa vyberie. Tam sa rozhodne podľa druhého písmenka, atď. Keďže nájde rum, musí nakoniec skončiť v jedinej rumovej komore.

A ako bude vyzerať nová jaskyňa, kde sa bude postupovať podľa papierika s otočenou postupnosťou znakov? Otočme všetky chodby v jaskyni opačným smerom. Rumovú komoru prehlásme za začiatočnú a začiatočnú za rumovú. Nemalo by byť ťažké nahliadnuť, že ak je papierik rumový (t.j. viedie k rumu) v pôvodnej jaskyni, bude po otočení rumový v novej jaskyni. V zadaní príkladu sa požaduje ekvivalencia, preto treba aj opačnú implikáciu. Tá znie: Ak je reverz papierika rumový v novej jaskyni, bol pôvodný papierik rumový v pôvodnej jaskyni. Toto by malo byť tiež uveriteľné. Pre záujemcov bude formálnejší dôkaz tvrdení na konci. Za povšimnutie stojí, že táto operácia môže pokaziť jednoznačnosť trasy jaskyňou.

Skvelé! Špeciálny prípad, keď je rumová jaskyňa iba jedna, máme poriešený. Čo však všeobecný? Jediný rozdiel je, že teraz mohol zlatokop skončiť v ľubovolnej z rumových komôr. Všetko by bolo v pohode, keby sme mohli mať v novej jaskyni viac začiatočných komôr (každá rumová v pôvodnej jaskyni) a nechať si zlatokopovi tipnúť, v ktorej má začať. Keď si zvolí tú správnu, pokračuje ako v predchádzajúcom odseku (premyslieť!). Lenže počiatočná komora môže byť z definície rumovej jaskyne iba jedna. Preto použijeme fintu $\check{F}^2$. Vyrobíme úplne novú komoru, ktorá bude začiatočná. Z nej bude viesť tunel do komory $K$ označený písmenom $a$ práve vtedy, keď existoval tunel v pôvodnej jaskyni z komory $K$ do niektorej z rumových komôr a mal značku $a$. Zvyšok jaskyne pomeníme ako v predchádzajúcom prípade. Z takto zostrojenej komory sa zjavne na každé písmeno dá dostať práve do tých komôr, do ktorých sa na toto písmeno dalo dostať v pôvodnej jaskyni z niektorej rumovej komory proti smeru tunelu. Teda prvý krok z tejto komory zodpovedá prvému kroku z ľubovoľnej komory, ktorá bola pôvodne rumová. No a nedeterminizmus sa už za nás rozhodne, ktorý tunel použiť. Treba si uvedomiť, že do nového začiatočného vrcholu žiadny tunel nevedie, preto neovplyvní žiadny ďalší dej v jaskyni, iba prvý krok.

[2] čítaj *fň*




Implementácia nie je zložitá. Potrebujeme pre každú komoru na vstupe vedieť, či je rumová (nejaké to pole booleanov). Ďalej už len načítavame tunely a hneď ich aj vypisujeme otočené na výstup. Ak sme načítali tunel, ktorý vedie z komory $K$ do rumovej komory, tak vypíšeme navyše aj tunel z novej počiatočnej komory do komory $K$. Popri tom samozrejme prečísluvávame komory, lebo vznikla nová s číslom 1.

Zložitosť: Pamäťová $O(n)$ a časová $O(n+m)$, kde $n$ je počet komôr a $m$ je počet tunelov.

Bodovanie: Mali sme tu deň detí, preto mierne.

Dôkaz, ktorý som sľúbil a na ktorý sa všetci určite už tešíte. Ak je papierik rumový v pôvodnej jaskyni $A$, bude po reverze rumový v novej jaskyni $B$.

V jaskyni $A$ existuje cesta s daným papierikom z komory 1 do niektorej rumovej komory $K$. Tá prechádza postupne cez komory $K_1K_2\ldots K_vK_{v+1}$, kde $K_1 = 1$, $K_{v+1} = K$ a $v$ je počet písmen na papieriku. Z konštrukcie jaskyne $B$ je zrejmé, že v nej musí existovať cesta $L_{v+1}L_v\ldots L_2L_1$, kde $L_{v+1} = 1$, a všetky zvyšné $L_i = K_i + 1$. Tým je táto implikácia dokázaná.

Opačná implikácia sa prevedie analogicky a šikovného čitateľa ňou nebudeme zbytočne zatažovať.

## Listing programu:

```
#include<stdio.h>
#define MAX 1000
int rum[MAX+10],n;

int main(void){
  int r,x,i;
  printf("Pocet rumovych jaskyn : ");
  scanf("%d",&r);
  printf("Zoznam rumovych jaskyn :\n");
  for (i=0; i<r; i++){
    scanf("%d",&x);
    rum[x]=1;
  }

  printf("Zoznam chodieb ukonceny 0 0 a:\n");
  while (1) {
    int k1, k2;
    char c;
    scanf("%d %d %c",&k1,&k2,&c);
    if (k1==0 || k2==0)
      break;

    printf("%d %d %c\n",k2+1, k1+1, c);
```

```
    if (rum[k2])            //hrana zo zaciatocnej komory
      printf("%d %d %c\n", 1, k1+1, c);
  }
  printf("0 0 a\n");
  return 0;
}
```

# Výsledková listina po 4. sérii kategórie KSP

| | Meno a priezvisko | kola | Trieda | | 41 | 42 | 43 | 44 | 45 | Σ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1. | Peter Bella | 3 | Gym. Jura Hronca BA | 67 | 14 | 14 | 15 | 15 | 15 | 140 |
| 2. | Pavol Juhos | 3 | Gym. Grösslingová BA | 49 | | 13 | 5 | 15 | 15 | 97 |
| 3. | Radovan Bauer | | Gym. Poštová Košice | 41 | 15 | 7 | 4 | 14 | | 81 |
| 4. | Peter Glaus | 2 | Gym. Jura Hronca BA | 37 | 14 | 6 | 4 | 11 | | 72 |
| 5. | Ján Katrenič | 3 | Gym. Školská Spiš. Nová Ves | 48 | | 8 | | 15 | | 71 |
| 6. | Jozef Veselý | 2 | Gym. Golianova Nitra | 46 | | 6 | .. | 11 | 5 | 68 |
| 7. | Marek Tesař | 5 | Gym. Haličská Lučenec | 57 | | | | 10 | | 67 |
| 8. | Tomáš Záthurecký | 4 | Gym. Malá Hora Martin | 64 | | | | | | 64 |
| 9. | Ján Mazák | 2 | Gym. Poštová Košice | 27 | 15 | 12 | 4 | 5 | | 63 |
| 10. | Peter Dzurjanin | 3 | Gym. Grösslingová BA | 35 | | 10 | 2 | 15 | | 62 |
| 11. | Michal Pokorný | 4 | Gym. Grösslingová BA | 59 | | | | | | 59 |
| 12. | Michal Mikuš | 3 | Gym. Jura Hronca BA | 39 | | 7 | | 10 | | 56 |
| | Tomáš Dzetkulič | 3 | Gym. Masarykova Michalovce | 38 | | 4 | 0 | 14 | | 56 |
| 14. | Peter Gaži | 4 | Gym. Grösslingová BA | 54 | | | | | | 54 |
| 15. | Jozef Tvarožek | 3 | Gym. Jura Hronca BA | 50 | | | | | | 50 |
| 16. | Radovan Čulák | 1 | Gym. Golianova Nitra | 33 | | 6 | .. | 10 | | 49 |
| 17. | Marek Ludha | 1 | Gym. Tajovského B. Bystrica | 24 | | 8 | | 15 | | 47 |
| 18. | Tomáš Sághy | 3 | SPŠ el. Nové Zámky | 21 | | 10 | | 15 | | 46 |
| 19. | Tomáš Záhorec | 3 | Gym. Jura Hronca BA | 41 | | | | | | 41 |
| 20. | Michal Glaus | 4 | Gym. Jura Hronca BA | 40 | | | | | | 40 |
| 21. | Ľuboš Steskal | 4 | Gym. Grösslingová BA | 35 | | | | | | 35 |
| 22. | Peter Libič | 2 | | 0 | 14 | 8 | | 8 | | 30 |
| 23. | Peter Miklian | 4 | Gym. Haličská Lučenec | 29 | | | | | | 29 |
| 24. | Martin Mrázik | 5 | Gym. Jura Hronca BA | 21 | | | | | | 21 |
| 25. | Ladislav Kucharík | | SPŠ el. SNP Piešťany | 20 | | | | | | 20 |
| 26. | Štefan Konečný | 2 | Gym. Jura Hronca BA | 17 | | | | | | 17 |
| 27. | Kamil Paulíny | 5 | Gym. Alejová Košice | 16 | | | | | | 16 |
| 28. | Radovan Berta | 2 | Gym. Daxnera V.n. Topľou | 11 | | | | | | 11 |
| | Marián Dvorský | 4 | Gym. Šrobárova Košice | 11 | | | | | | 11 |
| | Juraj Plavčan | 4 | Gym. Šk. bratov Čach. 14 BA | 11 | | | | | | 11 |
| 31. | Ondrej Danko | 3 | Gym. Jura Hronca BA | 10 | | | | | | 10 |
| 32. | Ľuboš Bednárik | 3 | Gym. Ľudovíta Štúra Trenčín | 9 | | | | | | 9 |
| 33. | Pavol Rybár | | Gym. Šk. bratov Čach. 14 BA | 7 | | | | | | 7 |
| 34. | Miroslav Rudišin | 4 | Gym. Šrobárova Košice | 5 | | | | | | 5 |
| 35. | Michal Rjaško | 2 | Gym. Daxnera V.n. Topľou | 3 | | | | | | 3 |
| 36. | Martin Ochránek | 1 | Gym. Jura Hronca BA | 1 | | | | | | 1 |