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 2. kola letnej časti

Milí riešitelia!

Je tu leto a preto[1] si rýchlo prečítajte vzoráky, mrknite na výsledkovú listinu a choďte sa radšej kúpať. Prefíkanejší jedinci si môžu zobrať vzoráky k vode.

S neštvrtákmi, ktorí sa umiestnili na začiatku výsledkovej listiny sa stretneme na jesennom sústredení.

Krátke vlasy, dlhý rozum. Navyše aj skôr uschnú.

KSPáci

1. Ohromný hazard

opravoval Dávidko
(max. 15 bodov)

Väčšina z vás prišla na to, ako vyzerá optimálna postupnosť. Potom sa ale prístupy líšili. Podľa toho, či ste pole triedili alebo len hľadali medián, ste dospeli k riešeniam s časovou zložitosťou $O(N^2)$, $O(N \log N)$ alebo len $O(N)$. Za to, aký prístup ste zvolili, ste dostali buď **8**, **11** alebo **15 bodov**. Dajaké bodíky som tradične stŕhal za nedobré zdôvodnenie, prečo je postupnosť optimálna, za chýbajúci alebo zlý odhad zložitosti.

Vzorové riešenie: Fero zrejme dostane určite naspäť aspoň hodnotu žetónov. To, čo dostane navyše, nazvime *zisk*.

Majme žetóny $a_1 \leq a_2 \leq \cdots \leq a_N$. Ukážeme, že postupnosť $a_1, a_N, a_2, a_{N-1}, a_3, a_{N-2}, \ldots$ dosahuje maximálny zisk. Predtým však pár všeobecných úvah:

Ľubovoľná postupnosť vhodení žetónov sa dá rozdeliť na niekoľko za sebou idúcich neklesajúcich podpostupností. Napríklad postupnosť $(1, 2, 4, 3, 2, 3, 3, 4, 5)$ sá rozdeliť na tri neklesajúce podpostupnosti $(1, 2, 4)$, (3), $(2, 3, 3, 4, 5)$. Vezmime si teraz hocijakú neklesajúcu podpostupnosť $(a_1, a_2, \ldots, a_k)$, pričom $a_1 \leq a_2 \leq \cdots \leq a_k$. Zisk z tejto postupnosti je $(a_2 - a_1) + (a_3 - a_2) + \cdots + (a_k - a_{k-1}) = a_k - a_1$.

Teraz naspäť k dokazovanému tvrdeniu. Vezmime si hocijakú ich postupnosť s maximálnym ziskom. Ukážeme, ako ju pretransformovať na postupnosť $a_1, a_N, a_2, a_{N-1}, a_3, a_{N-2}, \ldots$ tak, aby bolo nad slnko jasnejšie, že zisk nám neklesol.

Vezmime najmenší prvok a_1 a najväčší a_N, a všimnime si, v akých neklesajúcich podpostupnostiach ležia. Nutne a_1 leží na začiatku nejakej podpostupnosti a a_N nutne na konci nejakej. Ak ležia obe v jednej podpostupnosti, tak neurobíme v prvom kroku nič. Inak si všimneme neklesajúce podpostupnosti $(a_1, p, \ldots, q)$ a $(r, \ldots, s, a_N)$ so ziskom $(q - a_1) + (a_N - r)$. Ak $q \leq r$, tak možno preusporiadať poradie takto: $(a_1, p, \ldots, q, a_N)$ a $(r, \ldots, s)$. Zisk bude $(a_N - a_1) + (s - r)$ a teda neklesne, kedže $q \leq r \leq s$. Naopak ak $q > r$, tak to preusporiadame takto: $r, (p, \ldots, q)$ a $(a_1, \ldots, s, a_N)$. Sú dve možnosti, buď $r \leq p$ alebo nie. V oboch však zisk bude $(q - r) + (a_1 - a_N)$ t.j. rovnaký.

[1] toto je vnútorný rým

Teda už máme a_1 a a_N v tej istej neklesajúcej podpostupnosti $(a_1, a_i, \ldots, a_j, a_N)$. Túto zmeníme na dve (a_1, a_N) a $(a_i, \ldots, a_j)$, pričom zisk evidentne neklesne. Na záver dáme podpostupnosť (a_1, a_N) na začiatok všetkých podpostupností, čím sa zisk nezmení. Podobne to potom spravíme s a_2 a a_{N-1}, a_3 a a_{N-2} atď. Takto teda dostaneme žetóny do sľúbeného poradia. Táto postupnosť má zisk

$$(a_N - a_1) + (a_{N-1} - a_2) + \cdots + (a_{\lceil N/2 \rceil + 1} - a_{\lfloor N/2 \rfloor}) =$$

$$= (a_N + a_{N-1} + \cdots + (a_{\lceil N/2 \rceil + 1}) - (a_1 + a_2 + \cdots + a_{\lfloor N/2 \rfloor}).$$

Takýto istý zisk má ale hocijaká postupnosť žetónov $x, y, x, y, x, y, \ldots$, kde písmenko x zastupuje ľubovoľný žetón z $\lfloor N/2 \rfloor$ najmenších a písmenko y zastupuje ľubovoľný žetón z $\lfloor N/2 \rfloor$ najväčších. Ak máme nepárny počet žetónov, tak na konci nám ostane prostredný žetón, ktorý sa zvykne volať *medián*.

Náš algoritmus bude fungovať jednoducho. Načítame hodnoty žetónov do poľa, a potom nájdeme ich medián.[2] Ako vedľajší efekt hľadania mediánu sa nám pole usporiada tak, že v prvej polovici budú hodnoty menšie ako medián, potom samotný medián a potom hodnoty väčšie. Na záver už len berieme hodnoty striedavo zo začiatku a konca poľa. Ak je N nepárne, vypíšeme samotný medián. Časová aj pamäťová zložitosť bude lineárna – $O(N)$.

Drobnosť pre zasvätených: Algoritmus, ktorý som implementoval, je tzv. *median of five partitioning*. Jeho časová zložitosť je lineárna **v najhoršom prípade**. Toto ale neplatí o algoritme, ktorý bol uvedený vo vzorákoch zimnej časti – jeho zložitosť je totiž lineárna, ale len **v priemernom prípade**. Popis algoritmu *median of five partitioning* a dôkazu jeho linearity by zabral dalšie dve strany, zrejme by mu nikto nerozumuel, a navyše ho ani nikto nečítal. Preto som ho neuviedol. Nejaké komentáre nájdete v samotnom programe. Pochybujem ale, že z toho budete múdri.

Listing programu:

```
Program Ohromny_hazard;

type Pole = array[1..100] of integer;

var a:Pole;       { hodnoty zetonov }
    N,i,vyhra:integer;

{ vymena dvoch cisel }
procedure swap(var x,y:integer);
var p:integer;
begin
     p:=x;
     x:=y;
     y:=p;
end;

{ Vstup je pole a s N prvkami.
  Hladame k-ty najmensi prvok tohto pola.
  Ale zaroven toto pole preusporiadame tak,
  aby na zaciatku boli prvky mensie ako k-ty prvok,
  potom jemu rovne a na koniec vacsie. }
function find(var a:Pole; N,k:integer):integer;
```

[2] ako sa to robí, nájdete vo vzorákoch zimnej časti

```
var i,j,l,m:integer;
    b,c,d:Pole;
    x,y,z:integer;
begin
     { ak ma pole menej ako pat prvkov, tak ho utriedime a vratime k-ty }
     if N<5 then
     begin
          for i:=1 to N do
              for j:=i+1 to N do
                  if a[i]>a[j] then swap(a[i],a[j]);

          find:=a[k];
          exit;
     end;

     { rozdelime pole na patice a tie striedime }
     for l:=0 to (N div 5)-1 do
     begin
          for i:=1 to 5 do
              for j:=i+1 to 5 do
                  if a[5*l+i]>a[5*l+j] then swap(a[5*l+i], a[5*l+j]);

          b[l+1]:=a[5*l + 3]; { prostredny prvok kazdej patice }
     end;

     { najdeme z nich median }
     m:=find(b, N div 5, (N div 5+1) div 2);

     { rozdelime prvky povodneho pola na tri kopky:
       b <- mensie ako median
       c <- rovne medianu
       d <- vacsie ako median
     }
     x:=0; y:=0; z:=0;
     for i:=1 to N do
         if a[i]<m then begin inc(x); b[x]:=a[i]; end else
         if a[i]=m then begin inc(y); c[y]:=a[i]; end else
                        begin inc(z); d[z]:=a[i]; end;

     { najdeme k-ty prvok rekurzivne }
     if k<=x then find:=find(b,x,k) else
     if k<=x+y then find:=m else
        find:=find(d,z,k-x-y);

     { na zaver pole usporiadame }
     for i:=1 to x do a[i]:=b[i];       { male prky na zaciatok }
     for i:=1 to y do a[x+i]:=c[i];     { prvky rovne medianu }
     for i:=1 to z do a[x+y+i]:=d[i];   { velke prvky na koniec }
end;

begin
```

```
    read(N);
    for i:=1 to N do read(a[i]);

    { preusporiada pole tak,
      aby boli v prvej polovici boli mensie prvky ako v druhej
      a pripadne v strede bol median
    }
    find(a,N, (N+1) div 2);

    vyhra:=0;

    { striedavo vypisujeme prvok z prvej polovice a z druhej polovice }
    for i:=1 to N div 2 do
    begin
        write(a[i],' ',a[N-i+1],' ');
        vyhra:=vyhra + 2*a[N-i+1];
    end;

    { ak je N neparne, vypiseme median }
    if N mod 2 = 1 then
    begin
       write(a[(N+1) div 2]);
       vyhra:=vyhra + a[(N+1) div 2];
    end;

    writeln;
    writeln('Vyhra: ',vyhra);
end.
```

2. O zaľúbenom princovi

opravoval Rišo
(max. 15 bodov)

V tomto príklade bolo treba nájsť najkratšiu cestu medzi dvoma vrcholmi v danom grafe. Tento graf však mohol obsahovať aj záporné hrany. To úlohu trochu komplikuje. Bolo sa treba zamyslieť, ktoré algoritmy známe z „obyčajných“ grafov sa dajú použiť.

Príklad bol bodovaný nasledovne (n je počet vrcholov grafu):

- riešenia s časovou zložitosťou $O(n^3)$ a pamäťovou zložitosťou $O(n^2)$ – **max. 15 bodov**
- riešenia s horšou, ale polynomiálnou časovou a pamäťovou zložitosťou – **max. 13 bodov**
- riešenia s exponenciálnou časovou zložitosťou (rôzne backtracky) – **max. 8–10 bodov**, podľa použitia drobných optimalizácií
- nefunkčné riešenia – **max. 6 bodov**

Ak zvyčajne, po jednom bode ste mohli stratiť za chýbajúci (resp. zlý) odhad časovej a pamäťovej zložitosti programu. Jeden až tri body ste mohli stratiť za chyby vo vašom programe.

Medzi najčastejšie chyby patrila snaha použiť Dijkstrov algoritmus. Tento algoritmus pre grafy so zápornými hranami nefunguje. Opiera sa totiž o fakt, že ak určitý vrchol prehlásime za definitívny, už do neho nemôžme nájsť kratšiu cestu. Ak však existujú v grafe záporné hrany, toto nemusí byť pravda. Napr. pre graf s 3 vrcholmi a hranami $(1, 2, 5)$, $(1, 3, 6)$, $(3, 2, -2)$ nájde Dijkstrov algoritmus najkratšiu cestu z 1 do 2 s dĺžkou 5 (priama cesta), ale cesta 1-3-2 má dĺžku 4, je teda kratšia.

Dijkstrov algoritmus sa ani nedá nijako priamočiaro upraviť tak, aby fungoval pre grafy so zápornými hranami. Vaše snahy o úpravu dopadli v lepšom prípade vytvorením riešenia s exponenciálnou časovou zložitosťou, v horšom prípade vytvorením nefunkčného algoritmu.

Vzorové riešenie: Vzorovým riešením bolo použitie Floyd-Warshallovho algoritmu na nájdenie najkratších ciest medzi každými dvoma vrcholmi. Tento algoritmus pracuje správne na všetkých grafoch, ktoré neobsahujú záporný cyklus[3] a jeho časová zložitosť je $O(n^3)$, kde n je počet vrcholov grafu.

Algoritmus pracuje nasledovne: Označme $T[i, j, k]$ dĺžku najkratšej cesty z vrchola i do vrchola j, ktorá okrem týchto dvoch vrcholov prechádza iba vrcholmi s číslom menším alebo rovným k (vrcholy číslujeme 1 až n). V prípade, že takáto cesta neexistuje, túto hodnotu považujeme za ∞. Potom zrejme $T[i, j, 0]$ je rovné dĺžke priamej cesty z vrchola i do vrchola j.

Ak máme vypočítané všetky hodnoty $T[i, j, k-1]$ pre nejaké $k-1$, môžme pomocou nich vypočítať hodnoty $T[i, j, k]$. Princova cesta z vrchola i do vrchola j po vrcholoch s číslom menším alebo rovným ako k musí vyzerať nasledovne: buď princ nikdy neprejde vrcholom s číslom k, alebo ním prejde len raz. Ak by totiž vrcholom prešiel viackrát, príslušnú časť cesty je lepšie vynechať (lebo jej dĺžka nemôže byť záporná). Ak princ neprejde vrcholom k, dĺžka jeho cesty bude presne $T[i, j, k-1]$. Ak týmto vrcholom prejde, najlepšie, čo môže urobiť, je prejsť z i do k najkratšou cestou a aj z k do j. Potom dĺžka jeho cesty bude $T[i, k, k-1] + T[k, j, k-1]$. Teda hodnota $T[i, j, k]$ sa dá vypočítať ako minimum hodnôt $T[i, j, k-1]$ a $T[i, k, k-1] + T[k, j, k-1]$.

Poznámky k implementácii: Pri výpočte hodnôt T si netreba pamätať všetkých n^3 hodnôt. Stačí si pamätať iba $T[i, j, k]$ pre práve počítané k, čo je n^2 hodnôt. Floyd-Warshallov algoritmus potom pozostáva iba z troch vnorených cyklov[4] a jednej podmienky (viď. listing vzorového riešenia). To, že prepisujeme hodnoty $T[i, j, k-1]$ hodnotami $T[i, j, k]$ nám pri výpočte nemôže uškodiť. Jediné, čo sa nám môže stať je, že namiesto hodnoty $T[i, k, k-1]$ použijeme hodnotu $T[i, k, k]$ (resp. namiesto $T[k, j, k-1]$ hodnotu $T[k, j, k]$). Tieto hodnoty sa však zjavne rovnajú (najkratšia cesta z i do k cez vrcholy s číslom menším alebo rovným k do vrcholu k vojde len raz, a to na konci).

Neexistenciu hrany je najjednoduchšie reprezentovať hranou s veľmi veľkou dĺžkou. Dĺžka musí byť väčšia ako súčet dĺžok všetkých hrán, ale musí byť menšia ako polovica maximálnej hodnoty, ktorú môžme použiť ako číslo (aby nám pri sčítavaní a porovnávaní nenastalo pretečenie).

Časová zložitosť riešenia je $O(n^3)$ a pamäťová zložitosť je $O(n^2)$. Pre tých zvedavejších však poviem, že existuje aj lepšie riešenie úlohy. Hľadanie najkratších ciest medzi každými dvoma vrcholmi v grafe s maticou susednosti A je ekvivalentné hľadaniu tranzitívneho uzáveru matice $I + A$ (teda matice $(I + A)^* = (I + A)^n$). S pomocou algoritmu, ktorý násobí matice v nejakom čase $T(n)$, sa dá táto úloha riešiť v čase $O(T(n) \lg n)$, prefíkanejším algoritmom dokonca v čase $O(T(n))$. Na násobenie matíc existuje veľa prefíkaných algoritmov, medzi tie známejšie patrí napr. Strassenov algoritmus pracujúci v čase $O(n^{\lg 7})$. Viac si o tom môžte prečítať v knižkách Dexter C. Kozen: *Design and Analysis of Algorithms* a T. H. Cormen, C. E. Leiserson, R. L. Rivest: *Introduction to Algorithms*.[5]

Listing programu:

```
program O_zalubenom_princovi;
```

[3] bez tejto podmienky (presnejšie po jej nahradení požiadavkou, aby sa žiaden vrchol na ceste neopakoval) je úloha oveľa ťažšia – nie je známe riešenie s lepšou ako exponenciálnou časovou zložitosťou (problém je NP-úplný)

[4] ale pozor, poradie cyklov je dôležité!

[5] alebo žiadajte u svojich vedúcich na najbližšom sústredení. :-)

```
const INFTY=MAXINT div 2;    { Reprezentácia neexistencie hrany }
      MAXN=100;

var A:array[1..MAXN,1..MAXN] of integer; { Matica susednosti daného grafu }
    n,m,i,j,k:integer;

begin
  read(n,m);               { Načítanie vstupu }
  for i:=1 to n do
    for j:=1 to n do
      A[i,j]:=INFTY;
  for i:=1 to m do begin
    read(j,k);
    read(A[j,k]);
  end;

  for k:=1 to n do         { Počítame hodnoty T[i,j,k] pre všetky k; }
    for i:=1 to n do       { pre jedno k pre všetky i a j }
      for j:=1 to n do
        if A[i,k]+A[k,j]<A[i,j] then A[i,j]:=A[i,k]+A[k,j];

  if A[1,2]=INFTY then writeln('Princ sa k princeznej nedostane.')
    else writeln('Princ bude mať ',20+A[1,2],' rokov.');
end.
```

3. O pobodkovanej kružnici

opravoval Tom
(max. 15 bodov)

Tento príklad bol dosť ľahký. Nasvedčuje tomu aj fakt, že takmer všetky vaše riešenia boli správne. Body za tento príklad boli udelované zvlášť za popis, za myšlienku – **max. 7** a zvlášť za implementáciu – **max. 4** za každú časť (4 za lineárnu zložitosť, 2 za kvadratickú alebo kubickú, 0 za nefunkčné).

Vzorové riešenie: Vezmime si najprv pravouhlý trojuholník. Ako si väčšina z vás všimla, podľa Thalesovej vety treba a stačí, aby sa na tej kružnici našli 2 body, ktorých spojnica tvorí priemer danej kružnice (a samozrejme ešte ďalší bod, lebo trojuholník má mať 3 vrcholy, ale to je už zaručené zadaním).

Také dva body nájdem takto: Pre každý bod A si určím uhol $\alpha(A)$, ktorý zviera polpriamka OA s kladnou časťou osi x. Zrejme hľadám také dva body, pre ktoré je rozdiel týchto uhlov rovný π. Budem mať premenné i a j, v ktorých budú čísla bodov také, že $i < j$ a j-ty bod je prvý taký, že $\alpha(A_j) - \alpha(A_i) \geq \pi$. Premenná i bude prebiehať od 1 po N. Zrejme keď sa i zvýši o jedna, bude treba zvyšovať aj j (rozhodne nie však znižovať), tak aby bola splnená nerovnosť $\alpha(A_j) - \alpha(A_i) \geq \pi$. Ak niekedy nastane v nerovnosti rovnosť $\alpha(A_j) - \alpha(A_i) = \pi$, tak sme našli pravouhlý trojuholník. Časová zložitosť tohto algoritmu je $O(N)$, lebo i aj j prejde cez každý bod najviac raz.

Aby sme zistili, či tam je rovnostranný trojuholník, stačí si uvedomiť, že uhol medzi spojnicami rôznych vrcholov a stredu kružnice musí byť $2\pi/3$. Teda stačí mi podobným spôsobom nájsť body A_i,A_j,A_k, $i < j < k$ také, že $\alpha(A_k) - \alpha(A_j) = 2\pi/3$ a $\alpha(A_j) - \alpha(A_i) = 2\pi/3$. Podobne ako predtým i prebieha od 1 po N, k nemu hľadám najmenšie j také, že $\alpha(A_k) - \alpha(A_j) \geq 2\pi/3$ a k nemu hľadám najmenšie k také, že $\alpha(A_k) - \alpha(A_j) \geq 2\pi/3$. Ak v oboch nerovnostiach nastane rovnosť, našli sme rovnostranný trojuholník.

Časová zložitosť algoritmu je opäť lineárna, lebo i, j aj k prejdú cez každý bod najviac raz. Pamäťová zložitosť oboch algoritmov je lineárna – pamätám si iba uhly.

Poznámka: Ako ste si iste všimli, treba robiť porovnávanie reálnych čísel, ktoré sme nejako vypočítali; to sa robí tak, že zisťujeme, či sa „dosť málo“ líšia, napríklad menej než o 0,0001.

Listing programu:

```
const
  VELA=1000;
  Fi=2*Pi/3;
  Eps=0.0001;

var
  P:array[1..VELA] of record  {body na kruznici}
    x,y,     {suradnice}
    a:real;  {uhol}
  end;
  N:integer;   {pocet bodov}

function Eq(x,y:Real):Boolean;
begin
  Eq:=abs(x-y)<=Eps;
end;

function absgn(var x:real):integer;
begin
  if x>=0 then absgn:=0 else begin
    x:=-x;
    absgn:=1;
  end;
end;

function at2(x,y:real):real;    {zrata uhol}
var s,t,u:integer;
begin
  s:=absgn(x);t:=absgn(y);u:=1-(s xor t)*2;
  if x=0 then at2:=Pi/2+t*Pi else at2:=u*arctan(y/x)+t*Pi+(s xor t)*Pi;
end;

function Pra:Boolean;        {pravouhly 3uholnik}
var i,j:integer;
begin
  j:=1;
  for i:=1 to N do begin
    while P[j].a<P[i].a+Pi-Eps do inc(j);
    if Eq(P[j].a,P[i].a+Pi) then begin
      Pra:=True;
      exit;
    end;
  end;
  Pra:=False;
```

```
end;

function Rov:Boolean;    {rovnostranny 3uholnik}
var i,j,k:integer;
begin
  j:=1;k:=1;
  for i:=1 to N do begin
    while P[j].a<P[i].a+Fi-Eps do inc(j);
    while P[k].a<P[j].a+Fi-Eps do inc(k);
    if Eq(P[j].a,P[i].a+Fi)and Eq(P[k].a,P[j].a+Fi) then begin
      Rov:=True;
      exit;
    end;
  end;
  Rov:=False;
end;

procedure Rd;
var i:integer;
begin
  ReadLn(N);
  for i:=1 to N do begin
    ReadLn(P[i].x,P[i].y);
    P[i].a:=at2(P[i].x,P[i].y);
  end;
  P[N+1].a:=3*Pi;
end;

begin
  Rd;
  if not Pra then write('nie ');writeLn('je tam pravouhly trojuholnik');
  if not Rov then write('nie ');writeln('je tam rovnostranny trojuholnik');
end.
```

opravoval Goober
(max. 15 bodov)

4. O výskumníčkach

Súdiac podľa počtu vašich riešení, tento príklad patril medzi tie ťažšie. Navyše aj z tých neveľa riešení bola polovička nesprávna. Bodovanie vyzeralo dosť jednoducho – riešenia podobné vzorovému **15 bodov**, časovo náročnejšie, ale funkčné riešenia – **8 bodov** a napokon nesprávne riešenia boli ocenené jednotne – **3 body**. Nejaké tie body sa odoberali za nedostatočné popisy, dôkazy správnosti alebo odhady zložitosti.

Najprv zavedieme niekoľko termínov – vstupné reťazce pozostávajúce z A,C,T,G,*,U[6] nazveme *sekvenciami*. *Prefix* reťazca S je taký reťazec P, ku ktorému keď pripíšeme na koniec niekoľko (aj nula) znakov, dostaneme S (napr. 'aho' je prefixom 'ahoj'). No a teraz sa už môžeme vrhnúť na riešenie.

Ako sa to robiť nemalo: Niektorí z vás (nesprávne) predpokladali, že hľadanie najkratšej DNA možno spraviť pomocou greedy metódy asi takto: Prechádzame súčasne obe sekvencie a vždy keď narazíme na hviezdičku, tak sa pozrieme do druhej sekvencie, čo o tomto úseku

[6] múdri biológovia vedia, že uracil (U) sa v DNA nevyskytuje (je len v RNA); medveď Richard to však nevedel, a tak jeho DNA obsahuje aj U

hovorí. Túto informáciu skúsime čo najviac prekryť s informáciou zistenou doteraz. Problém je však v tom, že nie vždy je výhodné najväčšie možné prekrytie (napríklad dvojica sekvencií `ATT*TAT` a `*TTAT*` má ako najlepšie riešenie `ATTTAT`, ale pri najväčšom prekrývaní by sme dostali `ATTATAT`).

Ako sa to teda dalo a malo robiť: Vzorová myšlienka tkvie v použití *dynamického programovania* – budeme postupne riešiť daný problém pre prefixy oboch sekvencií. Pri rátaní výsledku pre nejakú dvojicu reťazcov sa totiž dajú využiť riešenia pre prefixy týchto reťazcov. Aby sme si tieto údaje mali kde pamätať, budeme mať tabuľku A rozmerov $(M+1)\times(N+1)$, kde M je dĺžka prvého a N druhého reťazca. Na mieste $A[i,j]$ bude dĺžka najkratšieho riešenia pre i-znakový prefix prvej a j-znakový prefix druhej sekvencie (prefixy môžu byť aj prázdne reťazce; ak pre dané dva prefixy neexistuje riešenie, budeme tam mať „nekonečno“). Túto tabuľku budeme vypĺňovať postupne po riadkoch (od $i=0$ až po M) a v rámci riadku po stĺpcoch (od $j=0$ až po N). Predpokladajme, že práve hľadáme hodnotu $A[i,j]$, čiže riešenie pre prefixy dĺžky i a j. Rozoberieme niekoľko prípadov – buď oba tieto prefixy končia písmenkami, ktoré sú rôzne – vtedy iste neexistuje spoločná DNA, a teda môžeme do $A[i,j]$ dať nekonečno. Ak končia rovnakým písmenom (nie hviezdičkou), tak riešenie musí končiť práve týmto písmenom a na začiatku mať najkratšie riešenie pre prefixy o 1 kratšie. Inými slovami, $A[i,j]=A[i-1,j-1]+1$. Ak prvý prefix končí hviezdičkou (druhý môže končiť čím chce), tak máme dve možnosti – buď za túto hviezdičku dosadíme prázdny reťazec (inými slovami, budeme to riešiť tak, ako keby tam hviezdička ani nebola, a teda $A[i,j]=A[i-1,j]$). Druhá možnosť je, že za túto hviezdičku dosadíme nejaký znak (konkrétne posledný znak druhého prefixu, ak to nie je hviezdička). Vtedy platí $A[i,j]=A[i,j-1]+1$. Samozrejme, z týchto dvoch možností vyberieme tú lepšiu. Analogický je prípad, keď druhý prefix končí hviezdičkou. Zaujímavé je, že prípadom, keď oba prefixy končia hviezdičkami, sa netreba zaoberať, lebo je zahrnutý v predchádzajúcich (buď za jednu, alebo za druhú hviezdičku nemá zmysel nič dosadzovať).

Všetko je to zatiaľ krásne, ale zabudli sme celé vypĺňovanie „naštartovať“ – okraje tabuľky sa musia vypĺňovať odlišne, lebo by sme vybehli z tabuľky. Lenže pre prázdne prefixy (t.j. okraje tabuľky) sa výsledky dajú vyrátať priamo – prázdny reťazec sa totiž dá skombinovať len s prázdnym prefixom, alebo s prefixom pozostávajúcim zo samých hviezdičiek (a vtedy bude mať výsledok dĺžku 0, inak nekonečno).

Keď máme celú tabuľku vyplnenú, stačí sa pozrieť na políčko $A[M,N]$, ktoré hovorí, či pre dané dve sekvencie existuje riešenie. Keď to už vieme, mali by sme hľadať cestu, akou sme toto políčko vyplnili. Takto môžeme spätne postupovať až po prvé políčko $A[0,0]$. Aby sme nemuseli túto cestu zložito hľadať, budeme si pri každom zapisovaní na políčko písať, že ako táto zmena vznikla (t.j., z ktorého políčka sme sa na dané dostali). Potom stačí ísť od konca po týchto „šípkach“ (od konca) a máme riešenie.

Odhad zložitosti: Očividne používame pole rozmerov $(M+1)\times(N+1)$, čiže pamäťová náročnosť je $O(MN)$. Časová zložitosť je rovnaká, lebo každé políčko čítame konštantný počet krát. Použitím rafinovaných metód by sa dala pamäťová náročnosť zredukovať na $O(M)$ (resp. keby nám stačilo vedieť dĺžku DNA, nie samotnú DNA), ale z pedagogických príčin toto riešenie neuvádzame.

Listing programu:

```
type
 SMERY=(Nedef,Posun1,Posun2,Posun12,Koniec);

const
 MAX=100;
 NEKONECNO=2*MAX+47;
```

```
var
 s,s1,s2:string;
 tab:array[0..MAX,0..MAX] of integer;
 smer:array[0..MAX,0..MAX] of SMERY;
 n1,n2,i1,i2:integer;
 v:integer;

procedure Update(i1,i2:integer;var n1,n2:integer;s:SMERY);
 begin
  n1:=i1; n2:=i2;
  case s of
   Posun1: dec(n1);
   Posun2: dec(n2);
   Posun12: begin dec(n1); dec(n2); end;
   end;
 end;

procedure Nastav(i1,i2,delta:integer;s:SMERY;force:boolean);
 var
  k1,k2:integer;
  value:integer;
 begin
  Update(i1,i2,k1,k2,s);
  value:=tab[k1,k2]+delta;
  if value>NEKONECNO then value:=NEKONECNO;
  if force or (value<tab[i1,i2]) then
   begin tab[i1,i2]:=value; smer[i1,i2]:=s; end;
 end;

 begin
  readln(s1); readln(s2);

{ Nastavime okraj tabulky }
  Nastav(0,0,0,Koniec,TRUE);
  v:=0; for i1:=1 to length(s1) do
   begin if s1[i1]<>'*' then v:=NEKONECNO; Nastav(i1,0,v,Posun1,TRUE); end;
  v:=0; for i2:=1 to length(s2) do
   begin if s2[i2]<>'*' then v:=NEKONECNO; Nastav(0,i2,v,Posun2,TRUE); end;

{ A (rafinovane) ratame... }
  for i1:=1 to length(s1) do for i2:=1 to length(s2) do
   begin
    if s1[i1]<>s2[i2] then Nastav(i1,i2,NEKONECNO,Nedef,TRUE)
    else Nastav(i1,i2,1,Posun12,TRUE);
    if (s1[i1]='*') or (s2[i2]='*') then
     begin
      Nastav(i1,i2,integer(s1[i1]<>'*'),Posun1,FALSE);
      Nastav(i1,i2,integer(s2[i2]<>'*'),Posun2,FALSE);
     end;
   end;
```

```
{ No a nakoniec to vsetko vypiseme... }
  s:=''; i1:=length(s1); i2:=length(s2);
  if tab[i1,i2]=NEKONECNO then write('Neda sa')
  else
   while smer[i1,i2]<>Koniec do
    begin
     Update(i1,i2,n1,n2,smer[i1,i2]);
     if tab[i1,i2]>tab[n1,n2] then
      case smer[i1,i2] of
       Posun1: s:=s1[i1]+s;
       Posun2: s:=s2[i2]+s;
       Posun12: s:=s1[i1]+s;
       end;
     i1:=n1; i2:=n2;
    end;
  writeln(s);
 end.
```

opravoval MišoF
(max. 15 bodov)

5. O čiernych krabičkách IV

Najskôr ukážeme, ako vieme pomocou našej čiernej krabičky K zistiť, či v grafe G existuje cesta, prechádzajúca cez každý vrchol práve raz. Nech má graf G N vrcholov. Zostrojíme nový graf G', ktorý bude mať $2N$ vrcholov tak, že do G pridáme N nových izolovaných vrcholov a na tento graf sa krabičky K opýtame. Zjavne v tomto grafe existuje cesta cez $2N/2 = N$ vrcholov práve vtedy, keď existuje cesta cez N vrcholov v pôvodnom grafe.

Takto sme vlastne zostrojili novú čiernu krabičku L, ktorá vie o ľubovoľnom grafe rozhodnúť, či v ňom existuje cesta cez všetky vrcholy. Ale ak nám krabička povie áno, stále nevieme rozhodnúť, medzi ktorými vrcholmi cesta vedie. Ako ju donútime, aby nám zistila, či existuje v grafe G cesta (cez všetky vrcholy) medzi dvoma konkrétnymi vrcholmi u, v? Zostrojíme nový graf G' tak, že pridáme do G nové vrcholy u', v' a hrany uu' a vv'. Na graf G' sa potom opýtame čiernej krabičky L.

Zjavne každá cesta v G' prechádzajúca cez u' v ňom musí začínať alebo končiť, to isté platí aj pre v'. Preto v G' každá cesta cez všetky vrcholy musí mať konce u' a v'. Keď z nej vyhodíme hrany uu' a vv', dostaneme cestu, ktorá má konce u, v a prechádza cez všetky vrcholy v G. A naopak, keď zoberieme cestu cez všetky vrcholy G, ktorá má konce u, v a pridáme k nej hrany uu' a vv', dostaneme cestu cez všetky vrcholy G'.

Preto v G existuje cesta cez všetky vrcholy s koncami u a v práve vtedy, keď v G' existuje cesta prechádzajúca cez všetky vrcholy. Takto sme vlastne zostrojili ďalšiu čiernu krabičku M, ktorá vie pre ľubovoľný graf G a jeho vrcholy u, v povedať, či v G existuje cesta cez všetky vrcholy, začínajúca v u a končiaca vo v.

Ako teraz o grafe G rozhodnúť, či v ňom existuje kružnica, prechádzajúca cez všetky vrcholy? (Takejto kružnici sa hovorí hamiltonovská.)

Pridáme do G nový vrchol $1'$ a spojíme ho hranami práve s tými vrcholmi, s ktorými susedí vrchol 1. Zjavne v G existuje kružnica prechádzajúca cez všetky vrcholy práve vtedy, keď v novom grafe existuje cesta cez všetky vrcholy z 1 do $1'$. (Hrana, ktorou prichádzame do $1'$ zodpovedá hrane uzatvárajúcej kružnicu, t.j. tej, ktorou sa po prejdetí všetkých vrcholov vraciame späť do 1. Pozor, **nemusí fungovať pre** $N = 2$, rozmyslite si, prečo.)

Takže pre $N < 3$ povieme rovno, že kružnica neexistuje, pre $N \geq 3$ pridáme vrchol $1'$ a opýtame sa krabičky M, či v takomto grafe existuje cesta cez všetky vrcholy z 1 do $1'$.

Zatiaľ nie je známy algoritmus, ktorý by v polynomiálnom čase (bez pomoci čiernej krabičky) zistil, či v grafe existuje hamiltonovská kružnica. Týmto sme vlastne ukázali, že zistiť, či v grafe existuje cesta cez polovicu vrcholov je aspoň taká ťažká úloha – keby sme ju vedeli riešiť v polynomiálnom čase, vedeli by sme v polynomiálnom čase zistiť, či v grafe existuje hamiltonovská kružnica.

Výsledková listina po 2. sérii kategórie KSP

	Meno a priezvisko	kola	Trieda		21	22	23	24	25	Σ
1.	Jakub Kováč	Gym. Jura Hronca BA	2	68	15	15	15	15	14	142
2.	Jakub Tekeľ	Gym. Jura Hronca BA	2	64	15	13	14	14	15	135
3.	Michal Burger	Gym. Grösslingová BA	2	64	15	8	15		14	116
	Vladimír Repiský	Gym. Žiar nad Hronom	4	67	15	9	15	3	7	116
5.	Peter Glaus	Gym. Jura Hronca BA	3	53	11	13	12		13	102
6.	Peter Perešíni	ZS Radvanská B. Bystrica	9	44	11	14	15	8	8	100
7.	Marek Jančuška	Gym. Párovská Nitra	2	46	11	10	13	3	6	89
8.	Marek Tesař	Gym. Haličská Lučenec	4	43	15	14	14			86
9.	Marek Ludha	Gym. Tajovského B. Bystrica	2	45	15	5	15			80
10.	Peter Bella	Gym. Jura Hronca BA	4	72						72
11.	Michal Ďuriš	Gym. Grösslingová BA	1	31	15	12	10			68
12.	Peter Macko	Gym. Liptovský Hrádok	1	34	8	7	11	3	3	66
13.	Juliana Lipková	Gym. Jura Hronca BA	3	31	11	3	13	3		61
14.	Peter Šufliarsky	Gym. Nové Zámky	2	28	11	6	14			59
15.	Pavol Müller	Gym. Grösslingová BA	4	57						57
16.	Štefan Konečný	Gym. Jura Hronca BA	3	24	15	5	10			54
17.	Milan Šatka	Gym. Liptovský Hrádok	3	23	15	6	7			51
	Anton Štefanek	Gym. Jura Hronca BA	2	0	15		9	13	14	51
19.	Anna Hanulová	Gym. Jura Hronca BA	2	15	14	5	15			49
20.	Michal Malý	Gym. Žiar nad Hronom	4	48						48
21.	Mariana Kuchynárová	Gym. Jura Hronca BA	2	18	10	4	9	3		44
	Dana Smažáková	Gym. Jura Hronca BA	2	17	15	9	3			44
23.	Juraj Porubský	Gym. Farská Nitra	2	12	11	7	10			40
24.	Michal Rjaško	Gym. Daxnera V.n. Topľou	3	30						30
25.	Martin Choma	Gym. Stará Ľubovňa	3	0	14		15			29
26.	Rastislav Lenhardt	GMMH Liptovský Mikuláš	2	27						27
27.	Milan Burda	Gym. Golianova Nitra	2	25						25
	František Šmitala	Gym. Farská Nitra	2	25						25
29.	Ivor Kollár	Gym. Jura Hronca BA	3	22						22
30.	Lukáš Poláček	Gym. K. Štúra Modra	2	0	11		10			21
31.	Filip Karas	Gym. Golianova Nitra	2	20						20
32.	Martin Šuška	Gymnázium Levice	3	19						19
33.	Oliver Janík	Gym. L. Stöckela Bardejov	2	10	8					18
	Marián Schmotzer	Gym. Horvátha BA	2	18						18
35.	Ľuboš Bednárik	Gym. Ľudovíta Štúra Trenčín	4	15						15
	Radovan Čulák	Gym. Golianova Nitra	2	15						15
37.	Peter Libič	Gym. Ľudovíta Štúra Trenčín	3	0			13			13
38.	Ján Dojčár	Gym. Školská Turč. Teplice	3	10						10
39.	Radovan Berta	Gym. Daxnera V.n. Topľou	3	7						7

1