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 1. kola letnej časti

Milí kovboji,

Prišla jar. Preto vytiahnite svoje oceľové tátoše zo stajne a hor sa bicyklovať. „Kto sa nebicykluje, nech ani neje“, tak znie heslo moderných kovbojov. A komu sa lení alebo nechová doma žiadneho tátoša, nech si sadne doma na zadok a prečíta si vzoráky KSP. Nech sa potom ale nečuduje, že dostane z toľkého štúdia vlka či očibôľ. Toto dupľom platí o tohtoročných maturantoch a maturantkách, ktorí sa už nadrepeli doma dosť.

Maturita je ako plot - keď neprelezieš sám, prehodia ťa.

KSPáci

1. O Kleofášovych pozemkoch

opravoval WSX
(max. 15 bodov)

Medzi riešeniami sa nachádzalo veľa 15-bodových, a teda príklad bol asi ľahký. Takmer všetci ste si uvedomili, že treba odstrániť najväčšie úseky medzi náleziskami. Vaše riešenia sa pohybovali v polynomiálnych zložitostiach v závislosti od použitého algoritmu. Väčšinou ste používali QuickSort, niektorí hľadali medián, čo však nezlepšilo výslednú časovú zložitosť, lebo bolo treba najskôr zotriediť vstup.

Bodovanie:

- Časová zložitosť $O(N \log N)$ – **15 bodov**
- Časová zložitosť $O(NK)$ – **10 bodov**
- Časová zložitosť $O(N^2)$ – **9 bodov**
- Časová zložitosť $O(N!), O(2^N)$ – **7 bodov**
- Nefunkčné – **0-4 body**

Za nedostatočný popis bolo možné stratiť 2 body.

Vzorové riešenie: Uvážme najmenší interval $I =< a, b >$, v ktorom ležia všetky náleziská, t.j. a je pozícia najľavejšieho náleziska a b je pozícia najpravejšieho náleziska. Všetky Kleofášove parcely budú nejaké podintervaly intervalu I, pričom medzi týmito parcelami budú v intervale I ležať ešte voľné úseky. Zrejme koncové body parciel aj voľných úsekov budú pozície nálezísk.

Základ je uvedomiť si, že minimalizovať celkovú veľkosť parciel, je to isté, ako maximalizovať celkovú dĺžku úsekov medzi nimi. Z nášho intervalu I budeme postupne vylučovať od najväčšieho úseku, až kým nedosiahneme požadovaný počet parciel. Vždy, keď vylúčime nejaký úsek, zvýši sa nám počet intervalov o 1. Preto môžme vylúčiť nanajvýš $K-1$ úsekov.

Najprv utriedime náleziská podľa ich pozície zľava doprava. Teraz iba v cykle zistíme dĺžky úsekov medzi susednými náleziskami. Zotriedime dĺžky podľa veľkosti, čím ľahko získame prvých $K-1$ úsekov.

Teraz zoberieme prvých $K-1$ intervalov a utriedime ich podľa ich začiatku. V poli intervalov teraz máme na prvých $K-1$ pozíciách $K-1$ najväčších intervalov utriedených podľa

polohy. Už nám zostáva len prejsť týmto poľom a vypísať výsledok. Vždy si pamätáme koniec posledného intervalu (na začiatku nech je to prvé nálezisko) a pri každej iterácii cyklu vypíšeme súradnicu konca posledného doteraz vylúčeného intervalu, ktorá bude udávať začiatok požadovaného intervalu a začiatok ďalšieho intervalu bude zase udávať koniec požadovaného intervalu.

Triediť vieme v čase $O(N \log N)$, zvyšok algoritmu má lineárnu časovú zložistosť, preto výsledná časová zložitosť je $O(N \log N)$, pamäťová $O(N)$. Pri troche snahy by sa časová zložitosť dala zlepšiť na $O(N \log K)$, rozmyslite si, ako.

Listing programu:

```
Program O_Kleofasovych_pozemkoch;

Const MAXN=100;
      Dlzka=1;
      Zaciatok=2;

Type Interval=Record
                A, B, L:Integer;
              End;

Var N,K,I,J:Integer;
    Nal:Array[1..MAXN] Of Integer;
    Int:Array[1..MAXN-1] Of Interval;

Function CmpFunc(A, B:Interval; I:Integer):Boolean;
Begin
  If I=Dlzka Then
    CmpFunc:=A.L>B.L Else
    CmpFunc:=A.A<B.A;
End;

Procedure QSortNal(L, R:Integer);
Var I,J:Integer;
    X,Y:Integer;
Begin
  I:=L; J:=R; X:=Nal[(L+R) Div 2];
  Repeat
    While(Nal[I]<X) Do Inc(I);
    While(X<Nal[J]) Do Dec(J);
    If I<=J Then
    Begin
      Y:=Nal[I]; Nal[I]:=Nal[J]; Nal[J]:=Y;
      Inc(I); Dec(J);
    End;
   Until I>J;
   If L<J Then QSortNal(L, J);
   If L<R Then QSortNal(I, R);
End;

Procedure QSortInt(L, R:Integer; Cmp:Integer);
Var I,J:Integer;
```

```
    X,Y:Interval;
Begin
  I:=L; J:=R; X:=Int[(L+R) Div 2];
  Repeat
    While(CmpFunc(Int[I], X, Cmp)) Do Inc(I);
    While(CmpFunc(X, Int[J], Cmp)) Do Dec(J);
    If I<=J Then
    Begin
      Y:=Int[I]; Int[I]:=Int[J]; Int[J]:=Y;
      Inc(I); Dec(J);
    End;
   Until I>J;
   If L<J Then QSortInt(L, J, Cmp);
   If L<R Then QSortInt(I, R, Cmp);
End;

Begin
  Read(N, K);
  If K>N Then K:=N;
  For I:=1 To N Do
    Read(Nal[I]);
  QSortNal(1, N);
  For I:=1 To N-1 Do
  Begin
    Int[I].A:=Nal[I];
    Int[I].B:=Nal[I+1];
    Int[I].L:=Int[I].B-Int[I].A;
  End;
  QSortInt(1, N-1, Dlzka);
  QSortInt(1, K-1, Zaciatok);
  J:=Nal[1];
  For I:=1 To K-1 Do
  If Int[I].A<>Int[I].B Then
  Begin
    WriteLn(J,' ',Int[I].A);
    J:=Int[I].B;
  End;
  WriteLn(J,' ',Nal[N]);
End.
```

2. O bájnom poklade

opravoval Tom
(max. 15 bodov)

Tento príklad bol asi dosť ľahký, keďže väčšina z vás ho vyriešila správne. Väčšinou ste si všimli, že jednotlivé písmená sa dali nahradiť toľkými „premennými“, koľko je ich dĺžka. Ďalej ste postupovali v zásade dvoma spôsobmi. Prvý z nich bol založený na tom, že ste sa snažili eliminovať jednotlivé premenné na základe porovnania daných zápisov tej postupnosti. Čiže ak sa našiel kameň, ktorého farba je určená číslom aj premennou tak ste za tú premennú dosadili to číslo, a ak bol určený rôznymi premennými tak ste dosadili jednu z nich za druhú. To najčastejšie viedlo k riešeniem s kvadratickou časovou zložitosťou, i keď sa to dalo optimalizovať na lineárnu. Druhý spôsob spočíval v tom, že ste si zostrojili graf, v ktorom vrcholom zodpovedali čísla a premenné, a hrany boli medzi tými vrcholmi, ktoré určovali

rovnaký kameň v rôznych zápisoch. Tieto riešenia mali zložitosť $O(L)$ až $O(L \log L)$, kde L je počet kameňov.

Bodovanie bolo takéto: Body sa dali získať za program (max. 10 bodov) a za popis (max. 5 bodov). Podľa zložitosti (a funkčnosti) bolo rozdelenie nasledovné:

- Časová zložitosť $O(L)$, pamäťová $O(L)$ – **10 bodov**
- Časová zložitosť $O(L \log L)$, pamäťová $O(L)$ – **9 bodov**
- Časová zložitosť $O(L^2)$, pamäťová $O(L)$ – **6 bodov**
- Časová zložitosť $O(L^2)$, pamäťová $O(L^2)$ – **5 bodov**
- Časová zložitosť $O(L^2. \log L)$, pamäťová $O(L)$ – **5 bodov**
- Časová zložitosť $O(L^3)$, pamäťová $O(L)$ – **4 body**
- Nefunkčné – **0-2 body** (podľa nálady)

Vzorové riešenie je založené na jednoduchom pozorovaní, že kamene, ktoré sú (hoci aj v rôznych zápisoch) určené rovnakou premennou majú rovnakú farbu. Môžeme teda zostrojiť graf, v ktorom vrcholom zodpovedajú jednotlivé kamene a hrany vedú medzi vrcholmi, ktoré zodpovedajú kameňom určeným rovnakou premennou.

Pre každý vrchol si ešte pamätáme farbu jemu zodpovedajúceho kameňa ak je určená jedným zo zápisov na vstupe, alebo si pamätáme, že ju ešte nepoznáme. Ak je farba nejakého kameňa určená rôznymi zápismi rôzne, je zrejmé, že neexistuje postupnosť, ktorá by vyhovovala obom zápisom. Takisto, ak medzi dvomi vrcholmi vedie nejaká cesta musia mať im zodpovedajúce kamene rovnakú farbu. A ak medzi nejakými vrcholmi žiadna cesta nevedie, môžu mať im zodpovedajúce kamene rôznu farbu.

Rozdelíme si teda náš graf na komponenty. Tie môžeme rozdeliť do troch skupín: *zlé* – to sú také, v ktorých už máme kamene rôznych farieb; *pevné* – to sú také, ktoré nie sú zlé a obsahujú vrchol, ktorému zodpovedá kameň s určenou farbou (túto farbu zrejme musia mať všetky kamene zodponedajúce nejakému vrcholu z tohto komponentu) a *voľné* – to sú tie, ktoré nie sú zlé, ani pevné. Zrejme ak sa nájde nejaký zlý komponent, tak už neexistuje žiadna postupnosť, ktorá by vyhovovala obom zápisom. A ak žiadny zlý komponent nenájdeme, existuje aspoň jedna postupnosť, ktorá vyhovuje obom zápisom. Totiž, ak nejaký kameň zodpovedá vrcholu v pevnom komponente, je jeho farba podľa predchádzjúcich úvah už jednoznačne určená. Ďalej pre každý voľný komponent si môžeme ľubovoľne (nezávisle od farieb ostatných komponentov) zvoliť jeho farbu, ktorú budú mať všetky kamene zodpovedajúce vrcholom z tohto komponentu. Počet rôznych postupností, ktoré vyhovujú obom zápisom bude teda N^K, kde N je počet farieb a K je počet volných komponentov.

Implementačné detaily: Keďže len hľadáme komponenty v našom grafe, nemusíme spájať vrcholy zodpovedajúce kameňom určeným rovnakou premennou „každý s každým“, ale stačí ich napr. spojiť do cyklu v poradí, v akom sa vyskytli na vstupe. Takto budú z každého vrchlu viesť najviac 2 hrany. Hľadanie komponentov sa robí prehľadávaním do hĺbky.

Zložitosť: Pamäťová zložitosť je $O(L)$, kde L je dĺžka hľadanej postupnosti, lebo o každom kameni si pamätáme len konštantne veľa údajov (farba, 2 hrany, ktoré z neho vedú), podobne o každej premennej, a tých je najviac $2L$. Časová zložitosť je $O(L)$, lebo prehľadávanie do hĺbky má zložitosť $O(L + M)$, kde M je počet hrán v grafe, pričom $M \leq 2L$.

Listing programu:

```
const vela=100;
      malo=30;

type
  TKamen=record   {udaje o jednom kameni v postupnosti}
    f:integer;  {farba}
```

```
    v:Boolean;  {ci som ho uz videl (pri prehladavani)}
    n:array[0..1] of integer;  {vychadzajuce hrany}
  end;

  TPrem=record     {udaje o premennej}
    d,          {dlzka}
    f,          {prvy vyskyt}
    l:integer;  {zatial posledny vyskyt}
  end;

var Pos:array[1..vela] of TKamen;  {postupnost kamenov}
    Prem:array[1..malo] of TPrem;  {premenne}
    Nk,   {pocet kamenov v postupnosti}
    Np,   {pocet premennych}
    Nf:integer;  {pocet farieb}
    JR:Boolean;  {ci to urcuje daku postupnost}
    pk:integer;  {pocet komponentov s neurcenou farbou}

function Rd:integer;  {nacita element zo vstupu}
var c:char;           {navratova hodnota>=0 ak je to cislo}
    v:integer;        {                 < 0 ak je to premenna}
begin
  read(c);
  while(c=' ') do read(c);
  if(c>='0')and(c<='9') then begin
    v:=ord(c)-ord('0');
    read(c);
    while (c>='0')and(c<='9') do begin
      v:=10*v+ord(c)-ord('0');
      read(c);
    end;
  end else begin
    v:=-(ord(c)-ord('a'))-1;
  end;
  Rd:=v;
end;

procedure PH(i,j:integer);  {prida hranu}
begin
  if Pos[i].n[0]=-1 then Pos[i].n[0]:=j else
                         Pos[i].n[1]:=j;
end;

procedure PriHr(i,v:integer);  {prida vyskyt premennej}
var j:integer;
begin
  if Prem[v].f=-1 then begin
    Prem[v].f:=i;
    Prem[v].l:=i;
  end else begin
    for j:=1 to Prem[v].d do PH(Prem[v].l+j-1,i+j-1);
```

```
    Prem[v].l:=i;
  end;
end;

procedure RdLn(fr:Boolean);  {precita jeden zapis}
var t,i,j:integer;
begin
  i:=1;
  while not SeekEoLn do begin
    t:=Rd;
    if t>=0 then with Pos[i] do begin  {je to cislo ...}
      if fr then begin
        v:=False;
        f:=-1;
        n[0]:=-1;
        n[1]:=-1;
      end;
      if f<0 then f:=t else if f<>t then JR:=False;
      inc(i);
    end else begin                     {je to premenna}
      for j:=1 to Prem[-t].d do begin
        if fr then with Pos[i+j-1] do begin
          v:=False;
          f:=-1;
          n[0]:=-1;
          n[1]:=-1;
        end;
      end;
      PriHr(i,-t);
      i:=i+Prem[-t].d;
    end;
  end;
  ReadLn;
  Nk:=i-1;
end;

procedure UzPrem(v:integer); {este spoji posledny vyskyt premennej sprvym}
var i:integer;
begin
  for i:=1 to Prem[v].d do PH(Prem[v].l+i-1,Prem[v].f+i-1);
end;

procedure RdIn;  {cita vstup}
var i:integer;
begin
  Assign(input,'poklad.in');
  Reset(input);
  ReadLn(Nf);  {nacitame pocet farieb }
  ReadLn(Np);  {pocet premennych}
  for i:=1 to Np do with Prem[i] do begin  {velkosti premennych ...}
    Read(d);
```

```
    f:=-1;
    l:=-1;
  end;
  ReadLn;
  RdLn(True);  {jednotlive zapisy ...}
  RdLn(False);
  Close(input);
  for i:=1 to Np do UzPrem(i);
end;

procedure dfs(ka:integer;fa:integer);   {prehladavanie do hlbky ...}
var i:integer;
begin
  with Pos[ka] do begin
    if f=-1 then f:=fa else if f<>fa then JR:=False;
    v:=True;
    for i:=0 to 1 do begin
      if(n[i]>=0)and(not Pos[n[i]].v) then dfs(n[i],fa);
    end;
  end;
end;

procedure zpk;  {zisti pocet komponentov s neurcenou farbou}
var i:integer;
begin
  for i:=1 to Nk do if(Pos[i].f>=0)and(not Pos[i].v) then dfs(i,Pos[i].f);
  pk:=0;
  for i:=1 to Nk do if(Pos[i].f<0)and(not Pos[i].v) then begin
    inc(pk);
    dfs(i,0);
  end;
end;

function Na(z,e:integer):integer;  {umocnenie}
var r:integer;
begin
  r:=1;
  while e>0 do begin
    r:=r*z;
    dec(e);
  end;
  Na:=r;
end;

begin
  JR:=True;
  RdIn;
  zpk;
  Write('zapisy urcuju ');
  if not JR then Write(0) else Write(Na(Nf,pk));
  WriteLn(' moznosti');
```

```
end.
```

3. O vláčikovej súprave

opravoval Poko
(max. 15 bodov)

Neprišlo príliš veľa riešení. No čo už, aspoň bolo menej roboty :-) Tých riešení bolo možno preto pomenej, že čas pokročil a mnohí štvrtáci nemyslia na nič iné, len na maturitu. Tým chcem popriať, nech im všetko vyjde podľa predstáv.

Vzorové riešenie: Úlohu budeme riešiť pomocou dynamického programovania. Vláčik si môžeme predstaviť ako jeden bod. Ako bude prechádzať stanicou, bude sa x-ová súradnica určujúca jeho polohu zväčšovať. Nech je teraz vláčik na súradnici x. Ďalej predpokladajme, že máme v poli pre každú koľaj vypočítané, aký najmenší počet výhybiek musíme prehodiť, aby sa vláčik na súradnici x mohol nachádzať na koľaji i. Toto číslo nazveme *hodnota koľaje*. Teraz zväčšujme x. Je zrejmé, že pokiaľ nenarazíme na začiatok alebo koniec nejakej výhybky, tak sa hodnoty koľají nebudú meniť. Predpokladajme, že na súradnici x sme zaznamenali začiatok nejakej výhybky (taká výhybka môže byť iba jediná) vedúcej z koľaje a na koľaj b. V tej chvíli sa určite nebude meniť hodnota ostatných koľají, a dokonca ani koľaje b. Na koľaji b sa totiž naša výhybka prejaví až na súradnici $x + 1$. Budeme teda meniť iba hodnotu koľaje a. V prípade, že bola výhybka vypnutá, pôvodna hodnota pre túto koľaj sa nezmení. V opačnom prípade sa nám hodnota zväčší o 1, lebo na to, aby vláčik mohol po tejto koľaji pokračovať, potrebujeme prehodiť našu výhybku.

Teraz nech nastane situácia, že na súradnici x na koľaji b nejaká výhybka prichádzajúca z koľaje a končí. Taká výhybka môže byť opäť iba jediná, a zmení sa nám iba hodnota koľaje b. Máme dve možnosti, ako sa vláčik dostane na súradnici x na koľaj b. Buď príde po koľaji b a prípadne prehodíme jednu výhybku (ak je naša výhybka v stave „zapnutá“), alebo na súradnici $x - 1$ odbočí z koľaje a (opäť musíme uvažovať prípadné prehodenie výhybky) a na koľaj b príde po výhybke. Novú hodnotu koľaje b dostaneme tak, že vezmeme minimum z hodnôt pre predošlé dva prípady. Lenže pozor! Na koľaji a sa medzi súradnicami $x - 1$ a x mohlo veľa zmeniť. Aby sme mohli s hodnotou pre koľaj a zo súradnice $x - 1$ pracovať, musíme si ju niekde pamätať, najlepšie priamo s príslušným koncom výhybky, až kým ho nespracujeme.

Implementácia bude vyzerať nasledovne: Aj keď to v zadaní nebolo napísané, budeme predpokladať, že výhybky sú na vstupe utriedené vzostupne podľa x-ovej súradnice začiatku. V opačnom prípade nie je veľký problém utriediť ich, zhorší nám to však časovú zložitosť. Hodnota koľaje p, po ktorej vlak prichádza, bude na začiatku 0, hodnota ostatných koľají bude „nekonečno“. Postupne načítavame výhybky. Nebudeme si však pamätať výhybku ako celok, ale zvlášť jej začiatok a koniec. Po načítaní j-tej výhybky najprv vytvoríme jej *koniec*, ktorý si uložíme do poľa na index j (každá výhybka má práve jeden koniec, preto koncov bude rovnako veľa ako výhybiek). Nakoľko výhybky boli utriedené, budú v poli utriedené aj ich konce. Potom zistíme, či sa pred začiatkom práve spracovávanej výhybky nenachádzajú nespracované konce. Ak také nájdeme, spracujeme ich podľa popisu uvedeného v predošlom odstavci. Potom nedávno vytvorený koniec spracovávanej výhybky doplníme o informáciu o hodnote koľaje, z ktorej výhybka vychádza, pre súradnicu, na ktorej začína. Po načítaní všetkých výhybiek ešte spracujeme zvyšné konce. Hľadaný výsledok bude hodnota koľaje q po spracovaní všetkých výhybiek (samozrejme aj ich koncov). Časová zložitosť je $O(N + K)$ (POZOR! Musíme inicializovať pole, v ktorom si uchovávame hodnoty jednotlivých koľají); ak výhybky nie sú utriedené, tak $O(N + K \log K)$. Pamäťová zložitosť je $O(N + K)$.

Ako som hodnotil:

- Riešenia podobné vzorovému, resp. riešenia pracujúce v zložitosti $O(N + K)$ dostali maximálne **15 bodov**.
- Riešenia s exponenciálnou časovou zložitosťou dostali maximálne **8 bodov**.

- Riešenia, ktoré na prvý pohľad vyzerali ako vzorové, ale zabúdali, že hodnota koľaje sa za úsek, na ktorom sa nachádza výhybka, môže zmeniť, prišli o polovicu bodov.
- Iným spôsobom zlé riešenia som ohodnotil maximálne **3 bodmi**.

Ďalej sa dali body stratiť za horšiu časovú alebo pamäťovú zložitosť spôsobenú nie najlepšou implementáciou algoritmu (dolu **3–5 bodov**), tiež za chýbajúci alebo nepostačujúci popis, za neuvedenú zložitosť vášho algoritmu, za chýbajúci listing programu a za nejaké drobné chyby. Taktiež sa vyskytli dvojice riešení, ktoré náš program CheatFinder 5.98 označil za opísané. Tieto boli ohodnotené polovicou pôvodne získaných bodov. Keďže program CheatFinder nie je šírený zadarmo, naopak, za jeho používanie platíme veľké prachy, boli by sme radšej, keby sa opisované riešenia jednoducho nevyskytovali, aby sme používanie programu mohli úplne vypustiť :-)

Listing programu:

```
Program O_vlacikovej_suprave; {by Poko}

Const   MaxN = 100;     {maximalny pocet kolaji}
        MaxK = 1000;    {maximalny pocet vyhybiek}

Type    TVyh = record
            x: real;    {suradnica, na ktorej je zaciatok/koniec vyhybky}
            k: integer; {cislo kolaje, na ktorej zacina/konci vyhybka}
            s: integer; {stav vyhybky}
            v: integer; {hodnota, ktoru si "koniec" so sebou nesie}
            end;

Var     N,K,p,q: integer;
        i,j,ptr: integer;   {ptr je index prave spracovavaneho konca}
        zac:     TVyh;      {zaciatok spracovavanej vyhybky}
        Kon:  array[1..MaxK] of TVyh;       {konce vyhybiek}
        Kolaj:   array[1..MaxN] of integer; {hodnoty kolaji}

procedure Spracuj;
begin
  {spracuje "koniec" vyhybky}
  if Kon[ptr].v < Kolaj[Kon[ptr].k]+Kon[ptr].s
    then Kolaj[Kon[ptr].k]:=Kon[ptr].v
    else Inc(Kolaj[Kon[ptr].k],Kon[ptr].s);
end;

begin
  for i:=1 to MaxN do Kolaj[i]:=MaxK+1;
  assign(input,'vyhybky.in'); reset(input);
  readln(N,K,p,q);
  Kolaj[p]:=0;
  ptr:=1;
  for i:=1 to K do begin
    readln(zac.x,zac.k,j,zac.s);        {nacitame vyhybku}
    with Kon[i] do begin     {vytvorime koniec vyhybky}
      x:=zac.x+1;            {koniec je o 1 posunuty vzhladom na zaciatok}
      k:=j;                  {koniec sa nachadza na kolaji, na ktoru vyhybka vedie}
```

```
      s:=zac.s;              {tento koniec funguje ako zarazka v dalsom cykle}
      end;
                             {spracujeme vsetky konce, ktore sa udeju skor}
                             {ako zacina posledne nacitana vyhybka}
    while Kon[ptr].x <= zac.x do begin Spracuj; Inc(ptr); end;
                             {teraz si zapamatame hodnotu z prvej kolaje}
    Kon[i].v:=Kolaj[zac.k]+1-Kon[i].s;
    Inc(Kolaj[zac.k],zac.s); {nakoniec spracujeme miesto, kde zacina vyhybka}
    end;
  while ptr<=K do begin Spracuj; Inc(ptr); end;
  writeln('Misko musi prehodit ',Kolaj[q],' vyhybiek.');
  close(input);
end.
```

4. O farebných trojuholníkoch

opravoval Dávidko
(max. 15 bodov)

Drvivá väčšina riešení načítala vstup do dvojrozmerného poľa, potom troma vnorenými cyklami prešla cez všetky trojice bodiek, zarátala každú trojicu tvoriacu jednofarebný trojuholník. Toto viedlo k algoritmu s kubickou zložitosťou t.j. N^3. Za takéto riešenia ste dostali po **8 bodov**. Našlo sa pár exotických riešení časovou zložitosťou N^6 až N^{11}. Takým som udelil po **4-5 bodov**.

Hŕstka riešiteľov pochopila, že ten príklad asi taký jednoduchý nebude, a stvorila riešenie s kvadratickou časovou zložitosťou - $O(N^2)$ a lineárnou pamäťovou zložitosťou - $O(N)$, podľa ktorého je napísané aj vzorové riešenie. Dostali po **15 bodov**.

Vzorové riešenie:

Nebudeme počítať, koľko je na obrázku jednofarebných trojuholníkov, ale naopak, koľko je na obrázku pestrých trojuholníkov t.j. takých trojuholníkov, ktorých strany nie sú rovnakej farby. Ak si označíme počet pestrých trojuholníkov K, tak potom jednofarebných trojuholníkov bude $\binom{N}{3} - K = \frac{N(N-1)(N-2)}{6} - K$.

Ako ale vypočítať K? Uvážme jednu bodku na obrázku s číslom i. Z nej vychádzajú nejaké modré úsečky a nejaké červené úsečky.

Vezmime nejakú modrú úsečku a nejakú červenú úsečku, obe vychádzajúce z našej bodky. Evidentne naša bodka spolu s koncovými bodmi modrej a červenej úsečky tvoria pestrý trojuholník.

Označme c_i počet červených úsečiek vychádzajúcich z našej bodky, potom modrých úsečiek vychádzajúcich z našej bodky bude $N-1-c_i$. Keďže máme na výber z c_i červených a nezávisle na tom z $N-1-c_i$ modrých úsečiek, takýchto pestrých trojuholníkov tam potom bude $c_i(N-1-c_i)$.

Spočítajme takéto súčiny pre všetky bodky: $P = c_1(N-1-c_1) + c_2(N-1-c_2) + \ldots + c_N(N-1-c_N)$. Koľkokrát je v súčte P zarátaný nejaký pestrý trojuholník?

Práve dvakrát! Stačí si uvedomiť, ako vyzerá pestrý trojuholník: Má dve strany jednej farby a jednu stranu druhej farby. A teda má jeden vrchol, kde sa stretávajú strany rovnakej farby, a dva vrcholy, kde sa stretáva červená s modrou stranou.[1] V práve týchto dvoch vrcholoch bude zarátaný pestrý trojuholník do súčtu P, a preto $K = \frac{P}{2}$

Napísať program je už jednoduché. Pre každú bodku si budeme pamätať v jednorozmernom poli hodnoty c_i. Na začiatku sú hodnoty c_i nulové. Ak pri načítavaní stretneme červenú úsečku medzi bodkami x,y, tak zvýšime hodnoty c_x a c_y o jedna. Nakoniec vyrátame súčet P. Jednofarebných trojuholníkov bude $\frac{N(N-1)(N-2)}{6} - \frac{P}{2}$.

[1] Odporúčam nakresliť si to.

Časová zložitosť je $O(N^2)$ t.j. kvadratická od počtu bodiek na obrázku. Pamäťová zložitosť je lineárna - $O(N)$.

Listing programu:

```
Program O_farebnych_trojuholnikoch;

var N,P,i,j,x,y:longint;
    { c[i] je pocet cervenych useciek vychadzajucich z bodky cislo i}
    c:array[1..10000] of longint;
    farba:string;

begin
     readln(N);

     for i:=1 to N do c[i]:=0;

     for i:=1 to N do
     for j:=i+1 to N do
     begin
          readln(x,y,farba);
          if (farba = 'cervena') then
          begin
               inc(c[x]);
               inc(c[y]);
          end;
     end;

     P:=0;
     for i:=1 to N do P:=P + c[i]*(N-1-c[i]);

     writeln( (N*(N-1)*(N-2) div 6) - (P div 2) );
end.
```

5. O čiernych krabičkách III

opravoval Braňo
(max. 15 bodov)

Nebolo tažké previesť úlohu do teórie grafov. Každému človeku, ktorého Janka skúma, prislúcha vrchol. Hrana medzi vrcholmi a a b vedie vtedy, keď sa ľudia a a b priatelia. Na tomto grafe chceme čo najrýchlejšie robiť zadané operácie. Aký je spodný odhad na zložitosť týchto operácií?

- *Init(n)*– $O(1)$
- *AddFriends(x,y)*– $O(1)$
- *DeleteFriends(x,y)*– $O(1)$
- *AreFriends(x,y)*– $O(1)$
- *EnumFriends(x)*– $O(k)$, kde k je počet susedov vrchola x

Náš cieľ je teda jasný. Avšak pekne postupne. Treba si rozmyslieť, v akej dátovej štruktúre budeme graf reprezentovať v pamäti. Od toho sa budú odvíjať aj časové zložitosti.

Prvá núkajúca sa možnosť je incidenčná matica. To je také dvojrozmerné pole A, v ktorom sa na pozíciu $A[i,j]$ značí, či hrana ij v grafe je, alebo nie je. V takejto matici sa

dajú rýchlo implementovať operácie *AddFriends(x,y)*, *DeleteFriends(x,y)*, *AreFriends(x,y)*. Naopak, operácia *EnumFriends(x)* vyžaduje čas $O(n)$.

Druhá možnosť je pre každý vrchol pamätať si zoznam susedov. V tomto prípade je rýchla operácia *EnumFriends(x)*. Naopak, operácie *AddFriends(x,y)*, *DeleteFriends(x,y)* a *AreFriends(x,y)* si vyžadujú čas $O(k)$.

Obidve spomínané možnosti majú svoje silné a svoje slabé stránky. Nedali by sa skĺbiť dobré vlastnosti obidvoch riešení?[2] Čo keby sme si pamätali zároveň aj incidenčnú maticu aj zoznamy susedov? Ako vkladať/odoberať hrany do/z obidvoch dátových štruktúr súčasne a rýchlo?

V incidenčnej matici na pozícii $A[i,j]$ si nebudeme pamätať len, či sa hrana ij v grafe nachádza, ale aj kde je záznam pre hranu ij v zozname susedov. Keď budeme pridávať novú hranu ij do grafu, pridáme nový záznam do zoznamu susedov a do $A[i,j]$ zapíšeme „šípku“ na nový záznam v zozname susedov. Keď hranu mažeme, zrušíme záznam v zozname susedov (ukazuje na neho šípka v $A[i,j]$), dieru vyplníme posledným záznamom, presunieme „šípku“ ukazujúcu na presúvaný záznam a nakoniec vynulujeme $A[i,j]$.

Doteraz sme taktne zamlčovali operáciu *Init(n)*. Dá sa urobiť v čase $O(1)$? Zdanlivo potrebujeme vynulovať incidenčnú maticu a vyprázdniť zoznam susedov všetkých vrcholov. Ale čo ak za sebou nasleduje veľa operácií *Init(n)*? Vtedy sa neoplatí zakaždým nulovať celé dátové štruktúry. Stačí predsa vynulovať len tie miesta, kde nie sú nuly. A čo keby sme v *Init(n)* nenulovali nič a nulovali neskôr? Ako ale neskôr spoznáme, či treba nejaký záznam vynulovať, alebo je tam niečo rozumné?

Zavedieme si aktuálny čas. Každý záznam v dátových štruktúrach bude mať pri sebe poznačený čas posledného prístupu. Ak sa tento čas zhoduje s aktuálnym časom, tak je záznam platný. Ak sa čas nezhoduje, tak záznam mal byť vynulovaný a vynulujeme ho teraz spolu so zmenou času posledného prístupu. Keď dôjde príkaz *Init(n)* nespravíme nič, len zväčšíme aktuálny čas.

Ale to už máme celé vzorové riešenie. Jeho časová zložitosť je optimálna, pamäťová $O(n^2)$. Ako som bodoval?

- Správne optimálne riešenia – **13 bodov**
- Správne optimálne riešenia až na *Init(n)* – **11 bodov**
- Správne optimálne riešenia až na *Init(n)* a *DeleteFriends(x,y)* – **8 bodov**
- Správne neoptimálne riešenia – **6 bodov**
- Nefunkčné riešenia – do **2 bodov**
- Popis a prehľadnosť zdojáku – **+2 body**

Listing programu:

```
program O_ciernych_krabickach_III;
const max_n=1000;
type tint = record        {integer, ktory ma pri sebe aj time stamp}
              i,time : integer            {i=data, time=posledny pristup}
            end;

{Ticho prepokladame, ze kompiler za nas tieto globalne premenne vynuluje}
var A:array[1..max_n,1..max_n] of tint;   {matica: A[i][j] je index do B[i]}
    B:array[1..max_n,1..max_n] of integer;{B[i] = zoznamy susedov vrcholu i}
    poc:array[1..max_n] of tint;          {poc[i]= pocet zaznamov v B[i]}
    time:integer;         {platny cas}
```

[2] Čo by to bola za básnická otázka, keby na ňu nemal autor v zápätí odpoveď.

```
procedure ensure_good_time_stamps(x,y:integer);
begin                       {ak su udaje stare, tak ich vynulujeme}
  if y>=0 then
  if A[x][y].time<time then begin         {je platny zaznam A[x][y]?}
    A[x][y].i:=-1;  A[x][y].time:=time;
  end;
  if poc[x].time<time then begin          {je platny zoznam B[i]?}
    poc[x].i:=0;    poc[x].time:=time;
  end;
end;

procedure init(n:integer);                {Hodnotu n nikdy nepouzijeme ;-) }
begin
  inc(time);              {novy platny cas. vsetky udaje su odteraz neplatne}
end;

procedure add_friend(x,y:integer);        {Prida orientovanu hranu x->y}
begin
  ensure_good_time_stamps(x,y);           {su udaje platne?}

  if A[x][y].i<>-1 then   {Ak je hrana uz v grafe, nechaj na pokoji.}
    exit;

  inc(poc[x].i);          {Pridame hranu na koniec zoznamu}
  B[x][poc[x].i]:=y;

  A[x][y].i:=poc[x].i;    {Pridame hranu do matice}
end;

procedure add_friends(x,y:integer);       {Prida neorientovanu hranu xy}
begin
  add_friend(x,y);
  add_friend(y,x);
end;

procedure delete_friend(x,y:integer);     {Zmaze orientovanu hranu x->y}
begin
  ensure_good_time_stamps(x,y);           {su udaje platne?}

  if A[x][y].i=-1 then    {Ak tato hrana v grafe nie je, nemame co robit.}
    exit;

  B[x][ A[x][y].i ]:= B[x][poc[x].i];     {posledny ide na volne miesto}
  A[x][ B[x][poc[x].i] ].i:= A[x][y].i;   {a ukazovatel na neho sa musi pohnut tiez}
  dec(poc[x].i);                          {v zozname je uz o jednu hranu menej}

  A[x][y].i:=-1;
end;

procedure delete_friends(x,y:integer);    {Zmaze neorientovanu hranu xy}
```

```
begin
  delete_friend(x,y);
  delete_friend(y,x);
end;

function are_friends(x,y:integer):boolean;
begin
  are_friends:= A[x][y].i<>-1;
end;

procedure enum_friends(x:integer);
var i:integer;
begin
  ensure_good_time_stamps(x,-1);          {su udaje platne?}

  if poc[x].i=0 then
    write('Nikto')  else
    for i:=1 to poc[x].i do               {prejdeme zoznam susedov}
      write(B[x][i],' ');
  writeln;
end;

begin end.
```

Výsledková listina po 3. sérii kategórie KSP

	Meno a priezvisko	kola	Trieda		31	32	33	34	35	Σ
1.	Peter Bella	Gym. Jura Hronca BA	4	15	14	15	15	13	72	
2.	Jakub Kováč	Gym. Jura Hronca BA	2	15	15	15	8	15	68	
3.	Vladimír Repiský	Gym. Žiar nad Hronom	4	15	15	15	8	14	67	
4.	Jakub Tekeľ	Gym. Jura Hronca BA	2	15	11	15	8	15	64	
5.	Pavol Müller	Gym. Grösslingová BA	4	15	11	10	8	13	57	
6.	Michal Burger	Gym. Grösslingová BA	2	15	13	15	8	4	55	
7.	Peter Glaus	Gym. Jura Hronca BA	3	15	10	7	8	13	53	
8.	Michal Malý	Gym. Žiar nad Hronom	4	15	12		8	13	48	
9.	Marek Jančuška	Gym. Párovská Nitra	2	9	11	8	8	10	46	
10.	Marek Ludha	Gym. Tajovského B. Bystrica	2	15	12		8	10	45	
11.	Peter Perešíni	ZS Radvanská B. Bystrica	0	10	8	8	8	10	44	
12.	Marek Tesař	Gym. Haličská Lučenec	4	15	6	5	4	13	43	
13.	Peter Macko	Gym. Liptovský Hrádok	1	10	4	7	5	8	34	
14.	Michal Ďuriš	Gym. Grösslingová BA	1	15			8	8	31	
14.	Juliana Lipková	Gym. Jura Hronca BA	3	15			8	8	31	
16.	Michal Rjaško	Gym. Daxnera V.n. Topľou	3	9			8	13	30	
17.	Peter Šufliarsky	Gym. Nové Zámky	2	15				13	28	
18.	Rastislav Lenhardt	GMMH Liptovský Mikuláš	2	9			8	10	27	
19.	Milan Burda	Gym. Golianova Nitra	2	7		2	8	8	25	
19.	František Šmitala	Gym. Farská Nitra	2	9			8	8	25	
21.	Štefan Konečný	Gym. Jura Hronca BA	3	4	2	7	8	3	24	
22.	Milan Šatka	Gym. Liptovský Hrádok	3	15			8		23	
23.	Ivor Kollár	Gym. Jura Hronca BA	3	10			4	8	22	
24.	Filip Karas	Gym. Golianova Nitra	2	2		2	8	8	20	
25.	Martin Šuška	Gymnázium Levice	3	10			5	4	19	
26.	Mariana Kuchynárová	Gym. Jura Hronca BA	2			3	15		18	
26.	Marián Schmotzer	Gym. Horvátha BA	2	10				8	18	
28.	Dana Smažáková	Gym. Jura Hronca BA	2			2	15		17	
29.	Ľuboš Bednárik	Gym. Ľudovíta Štúra Trenčín	4	15					15	
29.	Radovan Čulák	Gym. Golianova Nitra	2	15					15	
31.	Juraj Porubský	Gym. Farská Nitra	2	4			8		12	
32.	Ján Dojčár	Gym. Školská Turč. Teplice	3	10					10	
32.	Oliver Janík	Gym. L. Stöckela Bardejov	2	10					10	
34.	Radovan Berta	Gym. Daxnera V.n. Topľou	3	7					7	

0