Korešpondenčný seminár z programovania
XXIV. ročník, 2006/07
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: aSc – Applied Software Consultants spol. s r.o.
MICROSTEP-MIS spol. s r.o.
Gnome spol. s r.o.
Whitestein Technologies spol. s r.o.

Vzorové riešenia 2. kola zimnej časti

Mala by byť zima. Malo by snežiť. Ale nesneží. A nie je zima. Aspoň nie v Bratislave. To však nevadí, či sneží alebo nie, KSPéčko funguje, a preto sme vám opravili vaše riešenia a napísali tieto skvelé vzoráky. Tak sa pekne usaďte a vnorte sa do čítania tohto letáčiku, možno sa v ňom aj niečo nové dozviete `:-)`.

Trafená hus zašteká
KSPáci

1. Zákutia Matfyzu

opravoval Maják
(max. 15 bodov)

Tento príklad bol ľahký, o čom svedčí aj vysoká úspešnosť vašich riešení. Úloha bola priamočiara, jediné, čo sa dalo spraviť, bola obyčajná simulácia. Problém tohto príkladu sa však skrýval inde – v implementácií.

Viacerí z vás preberali veľké množstvo možností: do ktorého smeru je matfyzák natočený, či chodba pokračuje rovno, alebo sa musí zatočiť doprava alebo doľava, atď. Takýmto spôsobom sa riešenia statočne natiahli, niektoré aj na (úctyhodné) štyri strany.

Ukážeme si preto tri triky, vďaka ktorým sa dala úloha naprogramovať tak jednoducho, ako si môžete všimnúť v listingu. V programe si okrem mapy budeme pamätať, kde sa práve nachádza matfyzák (súradnice x a y), a ktorým smerom je otočený.

Trik #1: Pre názornosť zabudnime na matfyzákove ruky. Matfyzák bude otočený *čelom (!)* k stene. V každom kroku spravíme toto:

1. kým má matfyzák pred sebou stenu, točí sa doľava (t.j. pri pohľade zhora je to proti smeru hodinových ručičiek)
2. následne spraví krok dopredu
3. a otočí sa doprava (v smere hodinových ručičiek)

Nakreslite si to a presvedčte sa, že ak chodba pokračuje rovno, matfyzák sa otočí doľava, spraví krok a otočí sa späť doprava (opäť je čelom ku stene). Ak chodba pokračuje vľavo, matfyzák sa otočí dvakrát doľava, krok a doprava (opäť správne). Ak je to slepá ulička, postup opäť funguje. Na koniec, ak chodba zahýba doprava, matfyzák sa znovu poberie správnym smerom. A v čom je krása tohto postupu? Nepísali sme na každú zo 4 možností zvlášť riešenie, ale vymysleli sme univerzálny postup, ktorý vyrieši naraz všetky štyri. Zabili sme teda, ako sa hovorí, štyri muchy jednou ranou.

Trik #2: Ako sa bude matfyzák pohybovať? Ak je natočený smerom nadol, mali by sme zvýšiť y-ovú súradnicu, ak doprava, mali by sme zvýšiť x-ovú, ak doľava, mali by sme znížiť x-ovú súradnicu a ak nahor, mali by sme znížiť y-ovú súradnicu. Oveľa jednoduchší prístup je takýto: smery si očíslujeme: 0 je doprava, 1 nahor, 2 doľava a 3 nadol. Spravíme si konštantné pole dx a dy, kde si pre každý smer zaznačíme, o koľko treba zmeniť súradnice x a y. Napríklad, ak chceme ísť smerom doprava, treba x zvýšiť o 1 a y o 0, atď. Takto nám stačí napísať x := x+dx[smer]; y := y+dy[smer]; aby sme sa pohli daným smerom (pozri listing).

Ako sa otočíme doľava / doprava? Stačí smer zvýšiť / znížiť o 1 (všimnite si, že smery sú očíslované proti smeru hodinových ručičiek).

Trik #3: Posledný zdroj viacerých podmienok sú okraje. Ak vyjdeme mimo mapy, mali by sme skončiť. Ako ale zistíme, či sme mimo mapy? Musí byť $x < 1$ alebo $x > m$ alebo $y < 1$ alebo $y > n$. Uff. Jednoduchšie to je takto: spravíme si mapu v každom smere o 1 väčšiu, ako v skutočnosti a po celom jej obvode napíšeme písmeno `K`. Takto vieme, že sme mimo mapy jednoducho tak, že je tam `K`.

Zložitosť: Niekoľko z vás narazilo na problém pri odhadovaní časovej zložitosti. Napísali, že je lineárna od počtu krokov, ktoré musí matfyzák prejsť. To je síce pravda, ale to by ste potom mohli na ľubovoľný algoritmus napísať, že jeho časová zložitosť je lineárna od počtu vykonaných operácií (to je tautológia [1], ale nič nehovorí o rýchlosti algoritmu). Nás zaujíma, koľko operácií algoritmus rádovo vykoná *v závislosti od veľkosti vstupu*. V tomto prípade vieme o našom riešení povedať len toľko, že určite počet krokov nepresiahne veľkosť vstupu. T.j. časová zložitosť bude v najhoršom prípade $O(N \cdot M)$.

Keďže si musíme pamätať celú cestu matfyzáka, musíme si určite pamätať aj skoro celé bludisko, a preto aj pamäťová zložitosť bude $O(N \cdot M)$.

Bodovanie: A na záver ešte pár slov k bodovaniu – za riešenie v čase $O(N \cdot M)$ ste dostali 15 bodov, iné ani neprišli `:-)`. Z toho ste niekoľko bodov mohli stratiť za zlé odhady zložitostí $(1 + 1)$, chýbajúci zdroják (7), či iné drobné chyby.

Listing programu:

```
const max = 100;
  dx : array [0..3] of integer = (1, 0, -1, 0);
  dy : array [0..3] of integer = (0, -1, 0, 1);

var m, n, x, y, smer, i, j : integer;
    mapa : array [0..max+1,0..max+1] of char;

procedure vypis;
begin
  for i:=1 to n do begin
    for j:=1 to m do write(mapa[i,j]);
    writeln;
  end;
end;

begin
  readln (n, m);
  { mapu obalime K-ckami }
  for i:=0 to n+1 do for j:=0 to m+1 do mapa[i,j]:='K';
  { nacitame mapu }
  for i:=1 to n do begin
    for j:=1 to m do begin
      read(mapa[i,j]);
      if mapa[i,j]='Z' then begin y := i; x := j; end;
    end;
    readln;
  end;

  { pociatocny smer }
  smer:=3;
  if y=n then smer:=0;
  if x=m then smer:=1;
```

[1] čiže vždy pravdivé tvrdenie

```
  if y=1 then smer:=2;

  { bludime, kym neprideme na K - koniec }
  while mapa[y,x]<>'K' do begin
    mapa[y,x]:='*';
    while mapa[y+dy[smer], x+dx[smer]]='#' do smer := (smer+1) mod 4;
    x:=x+dx[smer];
    y:=y+dy[smer];
    smer := (smer-1 +4) mod 4;
  end;
  vypis;
end.
```

opravoval Peťo
(max. 15 bodov)

2. Zaslaná pošta

Predtým ako začneme riešiť si ešte raz a poriadne prečítame zadanie. Máme načítať *pole*, preusporiadať *to pole* a nepoužívať pomocné polia. Tá posledná požiadavka sa podarila splniť väčšine z vás, zato tie prvé dve neboli vždy úplne splnené. Vzorové riešenie splní všetky tri a ešte bude pracovať v neuveriteľne rýchlom čase.

Najprv si treba uvedomiť, že preusporiadať neznamená utriediť. Stačí nám, aby prvých l zásielok malo cenu menšiu ako K a zvyšných $N-l$ zásielok malo cenu väčšiu. Na to budeme potrebovať indexy i a j, pričom všetky prvky pred i sú menšie ako K a všetky prvky po j sú väčšie alebo rovné ako K. Teraz našou snahou bude, aby táto podmienka stále platila počas vykonávania cyklu (zvykne sa tomu hovoriť invariant). Na začiatku i nastavíme na 1 a j nastavíme na N (i bude ukazovať na prvý prvok pola, j na posledný). Zjavne nám invariant platí. Teraz budeme postupne i zväčšovať, až kým nenájdeme prvok väčší alebo rovný ako K (čize $A[i] \geq K$). Potom budeme j zmenšovať, až kým nenájdeme nejaký prvok menší prvok ako K ($A[j] < K$). Teraz môžu nastať dva prípady[2]:

- $i < j$: i ani j nemôžeme ďalej posúvať, lebo by nám invariant prestal platiť. Takže vymeníme prvky $A[i]$ a $A[j]$. Všimnite si, že invariant stále platí. Pokračujeme ďalším opakovaním cyklu – posúvaním indexov i a j.
- $i > j$: Naše pole je už usporiadané práve tak, ako chceme, takže môžeme skončiť. Toto náhodou vyplýva z nášho invariantu a podmienky $i > j$.

Ešte je dobré uvedomiť si, že tento postup vždy skončí.

Takýto algoritmus na preusporiadanie je aj súčasťou veľmi známeho triediaceho algoritmu *QuickSort*. Nepoužili sme žiadne pomocné pole, iba zopár premenných. Ale do pamäťovej zložitosti pole zarátame, takže sme akoby použili $O(N)$ premenných. Okrem načítania sme cez každé miesto v poli prešli práve raz. To znamená, že časová zložitosť je $O(N)$.

Nejaké bodíky sa pridávali za správnu myšlienku a ducha bojovníka. Naopak nejaké body občas zmizli za drobné chyby, resp. nie úplné dodržanie zadania a chýbajúci popis či odhad zložitosti. Cenník:

- $O(N)$: 15
- $O(N \log N)$: 11
- $O(N^2)$: 8
- nedodržanie zadania: -3
- chýbajúci popis, odhad: $-2, -1$

[2] všimnite si, že tretí prípad, teda že $i = j$ nemôže nastať

Listing programu:

```
var A : array [1..1000] of integer;
    k, n, i, j, pom : integer;

begin
  readln(n, k);
  for i:=1 to n do read(A[i]);

  i:=1; {jeden kontrolny ukazovatel od zaciatku}
  j:=n; {jeden kontrolny ukazovatel od konca}
  while (i<j) do begin
    {od zaciatku hladame prilis velky prvok, ktory tam nepatri}
    while (A[i]<k) and (i<n) do inc(i);
    {od konca hladame prilis maly prvok, ktory tam nepatri}
    while (A[j]>=k) and (j>1) do dec(j);
    {vsimnite si, ze kontrolujeme, ci nesahame mimo pola}
    if (i<j) then begin
      pom := A[i]; A[i] := A[j]; A[j] := pom;
    end;
  end;

  for i:=1 to n do write(A[i], ' ');
  writeln;
end.
```

3. Zlé prstene

opravoval Mišo
(max. 15 bodov)

Po načítaní a neusporiadaní poľa čísel vo veršoch treba zistiť, či medzi najmenším a najväčším číslom sú všetky čísla, ktoré treba, a či je každé práve raz (okrem prípadu, keď sú všetky rovnaké). A ktoré sú čísla, ktoré „treba“? No vieme, že počet čísel je N, vieme si zistiť najmenšie a najväčšie (pri načítavaní máme v premenných max a min uložené priebežne najväčšie a najmenšie číslo). Nech d je vzdialenosť (odborne nazývaná diferencia) medzi susednými číslami (v aritmetickej postupnosti je tá vzdialenosť d rovnaká). Platí, že $d = (max - min)/(N-1)$. Každé číslo z našej aritmetickej postupnosti je tvaru $min + d \cdot (i-1)$, pričom i ide od 1 po N. Zoberme si nejaké číslo $p[i]$. Ak $(p[i] - min) \bmod d \neq 0$, tak dané číslo nemôže byť v aritmetickej postupnosti. Ešte by sa nám oplatilo vedieť, na koľkom mieste v aritmetickej postupnosti sa číslo $p[i]$ nachádza. Je to práve[3] na mieste $(p[i] - min)/d + 1$. Každé číslo sa v aritmetickej postupnosti nachádza práve raz (výnimku tvorí konštantná aritmetická postupnosť, ktorú ale ošetrujeme na začiatku). Tak aj to treba zistiť. Na to nám poslúži pole r, ktoré má N prvkov. Ako budeme postupovať? Pre každé číslo zistíme, či môže byť v aritmetickej postupnosti, teda či $(p[i] - min) \bmod d = 0$ a či sa tam také číslo nevyskytuje viackrát. V pomocnom poli r budú na začiatku len samé nuly. Postupne si do neho značíme, na ktoré miesta v postupnosti sme už niektoré čísla dali. Ak zistíme, že na nejakom mieste by mali byť aspoň dve čísla, dáme to patrične najavo.

Bodovanie: Za optimálne riešenie v čase $O(N)$ bolo 15 bodov, za triedenie v $O(N \log N)$ 12, za riešenie bez triedenia v $O(N^2)$ bolo 10 bodov, za triedenie v $O(N^2)$ bolo 8 bodov. Za nefunkčné riešenie bolo menej ako 7 bodov, väčšinou ste porovnali súčet so vzorcom $(max + min) \cdot n/2$. Vyskytlo sa ešte zopár neošetrených okrajových prípadov, napr. $N = 1$. Za každý som strhol bod. Za zlý alebo chýbajúci odhad zložitosti som strhol bod.

[3] +1 je tam preto, lebo pole indexujeme od 1

Listing programu:

```
const maxn = 1000;
      infty = 32767; {nekonecno}

var p : array [1..maxn] of integer;
    bolo : array [0..maxn-1] of boolean;
    i, d, n, min, max : integer;
    dobre : boolean;

begin
  readln (n); min := infty; max := -infty;
  for i := 2 to n do begin
    read (p[i]); bolo[i] := false;
    if p[i] < min then min := p[i];
    if p[i] > max then max := p[i];
  end;
  dobre := true;
  if n >= 3 then
    if (max-min) mod (n-1) > 0 then dobre := false
    else begin
      d := (max-min) div (n-1);
      if d=0 then begin {konštantná postupnosť}
        for i := 2 to n do
          if p[i] <> p[i-1] then begin dobre := false; break; end;
      end else begin
        for i:=1 to n do begin
          if ((p[i]-min) mod d <> 0) {nie je členom aritmetickej postupnosti}
          or bolo[(p[i]-min) div d] then begin {už bolo}
            dobre := false; break;
          end else bolo[(p[i]-min) div d]:=true; {zaznačíme si, že také už bolo}
        end;
      end;
    end;
  if dobre then writeln ('ano') else writeln ('nie');
end.
```

opravoval Miňo
(max. 15 bodov)

4. Zo školskej jedálne

Riešení tohto príkladu prišlo presne 40, pričom sa dali porozdeľovať do niekoľkých kategórií. Existujú nejaké nefunkčné riešenia, z funkčných riešení sa našlo zopár fungujúcich metódou hrubej sily so zložitosťou $O(N!)$. Polynomiálne riešenia mali zväčša zložitosť okolo $O(N^3)$ a $O(N(N+M))$.

Podľa zadania úlohy máme na vstupe usporiadané dvojice A a B, pričom z jedla A sa dá vyrobiť jedlo B. Takéto vzťahy sa dajú výborne reprezentovať pomocou grafu – v tomto prípade orientovaného. Vrcholy v tomto grafe budú jednotlivé jedlá a hranami budú „šípky", ktoré hovoria, že z jedného jedla vieme vyrobiť druhé. Vyberme si teraz nejaké jedlo A a nájdime zodpovedajúci vrchol v tomto grafe. Pre všetky jedlá, ktoré sa dali priamo pripraviť z A existujú hrany vedúce do ďalších vrcholov. Ďalej, pre každý z týchto vrcholov takisto existujú pre všetky ich premeny hrany smerujúce do ďalších vrcholov a tak ďalej. V konečnom dôsledku, jedlo A vieme premeniť na jedlo B práve vtedy, keď existuje cesta z vrcholu A do vrcholu B. Teda pre každý vrchol stačí zistiť, do akých všetkých vrcholov sa z neho vieme dostať.

Najjednoduchší prístup je vyskúšať všetky možné cesty z nejakého vrcholu v grafe a zapamätať si pre každý dosiahnuteľné vrcholy. Toto je pekná idea, má však malý háčik – bez úprav a v úplnom grafe (každé dva vrcholy sú prepojené) dostávame zložitosť $O(N!)$, čo aj počet všetkých ciest v takomto grafe.

Skúsme ísť na vec z iného konca. Ak existuje nejaká cesta medzi vrcholmi A a B, môžeme medzi tieto vrcholy pridať hranu bez toho, aby nám to niečo pokazilo. A ak niečo takéto spáchame pre všetky vrcholy dosiahnuteľné z A, stačí nám pozrieť sa na priamych susedov A a máme všetky dosiahnuteľné vrcholy. Graf s takto popridávanými hranami sa nazýva tranzitívny uzáver pôvodného grafu. Vyrobiť sa dá veľmi jednoducho použitím Floyd-Warshallovho (v tomto prípade presnejšie Roy-Warshallovho) algoritmu. Čas behu je $O(N^3)$. Ale toto stále nie je vzorové riešenie.

Na to, aby sme sa mohli pustiť do vzorového riešenia, popíšeme najskôr algoritmus, na ktorom sa zakladá – prehľadávanie do hĺbky. Jedná sa o rekurzívny algoritmus (teda taký, kde funkcia volá samu seba). Zavolá sa vždy s nejakým počiatočným vrcholom a potom sa funkcia sama zavolá na každý z vrcholov, do ktorého sa dá dostať. A čo teraz s tým? Funkcia sa zavolá na vrchol 1, z neho na vrchol 2, z neho opäť na vrchol 1, na 2, ... Takže samotný takýto algoritmus nám nepomôže. Skúsme ho ale upraviť – naviac si ešte niekde pamätajme, ktoré vrcholy sme už niekedy navštívili. Vrcholov máme konečne veľa a keďže si v každom kroku poznačíme nejaký vrchol ako navštívený a vchádzame iba do ešte nenavštívených, po istom čase už isto nenavštívený vrchol nenájdeme a vynoríme sa z rekurzie. Keď sa teraz pozrieme na zoznam navštívených vrcholov, dostávame vlastne všetky vrcholy dosiahnuteľné z pôvodného vrcholu.

Tento algoritmus nie je ťažké prepísať/pokaziť na backtrack. Rozdiel je v tom, ako sa správame ku grafu. Pri backtracku by sme sa iba snažili, aby sme sa nezacyklili, pričom si pamätáme aktívne vrcholy (vrcholy, ktoré sú teraz navštívené). Vrcholy, ktoré opustíme, budú opäť voľné. Takto vieme vyskúšať skutočne všetky cesty v grafe. Naopak, pri prehľadávaní do hĺbky už navštívené vrcholy nezabúdame a nevraciame sa do nich – už sme ich videli a preskúmali, nič nové nám nepovedia.

Skúsme veľmi jednoduchý prístup – pre každý vrchol spustíme prehľadávanie do hĺbky, pričom sa vyhýbame už navštíveným vrcholom. Jedno takéto prehľadanie trvá $O(N + M)$, pričom ho vykonáme pre každý vrchol – N-krát, čo nám dáva priemernú zložitosť $O(N(N + M))$. Ale pozor – ak si vezmeme úplný graf, platí $M = N^2$ a naša zložitosť razom vyskočí na $O(N^3)$. Toto nám ale nevadí – je to iba najhorší prípad. Zároveň je toto aj vzorové riešenie príkladu.

Bodovanie bolo teda zhruba nasledovné: 15 bodov dostalo riešenie v čase rovnakom ako vzorové, 12 bodov dostalo riešenie fungujúce priemerne v $O(N^3)$, 9 bodov bolo za backtrack. V niektorých prípadoch sa dalo pokaziť si pamäťovú zložitosť, za ňu sa strhával 1 bod. Ďalšie strhávanie bodov sa konalo pri chybajúcich zložitostiach (po 1 bode) a pri chýbajúcom alebo veľmi slabom popise (2 body). Ak by sa vyskytol nejaký iný dôvod, na riešení bude rozhodne popísaný a vysvetlený.

Na záver si ešte dovolím malú poznámku a tiež pochvalu. Niekoľko z vás poslalo riešenie, ktoré je rýchlejšie ako naše vzorové, za čo ste boli odmenení 16-timi bodmi. Zložitosť postupu ale presahuje rámec tohto textu.

Listing programu:

```
const maxn=10;
      maxm=100;

type
    TNavstivene = array[1..maxn] of boolean;

var
```

```
    M,N: integer;
    graf: array[1..maxn,1..maxm] of integer;
    pocty: array[1..maxn] of integer;
    navstivene: TNavstivene;
    i, j, a, b: integer;

procedure dfs (var visited: TNavstivene; vrchol: integer);
var kam:integer;
begin
    visited[vrchol]:=true;
    for kam:=1 to pocty[i] do
        if not visited[graf[vrchol][kam]] then dfs (visited,graf[vrchol][kam]);
end;

begin
    readln (N, M);

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

    for i:=1 to M do
    begin
        readln(a,b);
        inc(pocty[a]);
        graf[a][pocty[a]]:=b;
    end;

    for i:=1 to N do
    begin
        for j:=1 to N do navstivene[j]:=false;
        dfs(navstivene,i);
        write('',i,':');
        for j:=1 to N do
            if navstivene[j] then write(' ',j);
        writeln;
    end;
end.
```

5. Otestujte si monitor!

opravovala Julka
(max. 12 bodov)

Tento príklad patril k ľahším. Ak sa vám náhodou nepodarilo vymyslieť optimálne riešenie, tak sa dalo poslať aspoň jednoduché riešenie, ktoré prešlo všetky pixely z okna a zistilo, či sú zlé. Vďaka tomu mal tento príklad veľa riešiteľov. Hodnotila som ho nasledovne:

- Riešenie v čase $O(MN + K)$: 12 bodov
- Riešenie v čase $O(MNK)$: 8 bodov
- Nedostatočný alebo chýbajúci popis: strhla som najviac 4 body
- Chýbajúci alebo zlý odhad zložitostí: -1 bod

Našim cieľom vo vzorovom riešení bude vedieť rýchlo (v konštantnom čase) odpovedať na otázku, či je v danom okne chybný pixel. Dôležité je uvedomiť si, že počet chybných pixelov v okne $[x_1, y_1; x_2, y_2]$ sa dá vypočítať ako počet chybných pixelov v okne $[1, 1; x_2, y_2]$ mínus počet chybných pixelov v okne $[1, 1; x_1 - 1, y_2]$ mínus počet chybných pixelov v okne $[1, 1; x_2, y_1 - 1]$ plus počet chybných pixelov v okne $[1, 1; x_1 - 1, y_1 - 1]$. Toto sa nazýva tiež príncíp zapojenia a vypojenia. Keď odrátame (vypojíme) obdĺžniky $[1, 1; x_1 - 1, y_2]$ a

$[1,1;x_2,y_1-1]$, tak obdĺžnik $[1,1;x_1-1,y_1-1]$ sme odrátali dvakrát, preto ho treba ešte pripočítať, teda pripojiť.

Toto nám už napovedá, aby sme si najprv pre každú súradnicu $[x,y]$ vypočítali, koľko je chybných pixelov v obdĺžniku $[1,1;x,y]$. Toto vieme zrátať v čase $O(MN)$ a to tiež podobným spôsobom. Prejdeme všetky pixely a počet chybných pixelov v obdĺžniku $[1,1;x,y]$ je toľko, koľko je súčet v obdĺžnikoch $[1,1;x,y-1]$ a $[1,1;x-1,y]$ mínus $[1,1;x-1,y-1]$. K tomuto číslu je ešte treba prirátať 1, ak je na políčku $[x,y]$ chybný pixel.

Keď už máme toto zrátané, môžeme odpovedať na otázky. Teraz môžeme podľa vyššie popísaného postupu vypočítať počet chybných pixelov v okne. Ak je ich viac ako 0, tak tam chybný pixel je a ak nie, tak tam nie je ;).

Listing programu:

```
var pix, dyn: array [0..100,0..100] of integer;
    N, M, K, i, j, kk, r1, r2, s1, s2, lx, px, hy, dy: integer;

function min(a,b: integer): integer;
begin
  if (a<b) then min:=a else min:=b;
end;

function max(a,b: integer): integer;
begin
  if (a>b) then max:=a else max:=b;
end;

begin
  readln(N,M);

  { nacitame }
  for i:=1 to N do begin
    for j:=1 to M do read(pix[i,j]);
    readln;
  end;

  { dame nuly na kraj pola dyn, aby sme neskor nemuseli tolko ifovat }
  for i:=1 to N do dyn[i,0] := 0;
  for j:=1 to M do dyn[0,j] := 0;

  { vypracujeme pole dyn tak, aby na mieste [x,y] bol pocet chybnych pixlov
    v obdlzniku [1,1] [x,y] }
  for i:=1 to N do
    for j:=1 to M do
      dyn[i,j] := dyn[i-1,j] + dyn[i,j-1] - dyn[i-1,j-1] + pix[i,j];

  { odpovedanie na otazky }
  readln(K);
  for kk:=1 to K do begin
    readln(r1,s1,r2,s2);

    { nech vieme, ktora suradnica je lava horna, ... }
    lx:=min(s1,s2);
    px:=max(s1,s2);
    hy:=min(r1,r2);
    dy:=max(r1,r2);

    { odpovieme }
    if (dyn[px,dy]-dyn[px,hy]-dyn[lx,dy]+dyn[lx,hy]>0) then writeln('Je chybny')
      else writeln('Nie nie je chybny pixel');
```

```
  end;
end.
```

6. O hrdinskom Micovi

opravoval KuKo
(max. 15 bodov)

Začnime bodovaním, nech máte predstavu, ako som vás asi zaškatuľkoval:

1. škatuľa: riešenia v čase $O(n^2 \log n)$ užívajúce si len $O(n)$ pamäte; vzorák: 15 bodov.
2. škatuľa: rovnako rýchle ale s pamäťovou zložitosťou $O(n^2)$: 12 bodov.
3. škatuľa: riešenia v čase $O(n^3)$: exkluzívnych 7 bodov.
4. škatuľa: joj, ešte aj $O(n^2)$ pamäte: len 4 body.
5. potom bola ešte škatuľka „mimo“ – programy, ktoré uvažovali iba niektoré priamky (pod $k \cdot 45°$ uhlom) – 2 body.

Okrem týchto základných škatuliek sa dali získať všakovaké ujmy na bodoch za myšlienkové bugy, za šetrenie na popise alebo zdrojáku, za zlé odhady zložitosti a tak.

Zle. Čo sa týka tretej škatule, bolo to to najjednoduchšie a najpriamejšie riešenie. Riešenie, ktoré malo napadnúť každého, kto sa nad tým ani trochu nezamyslel. Jednoducho hrubá sila. Zrejme (pre $n \geq 2$) nemá zmysel brať priamku, ktorá neprechádza žiadnym alebo len jediným bodom. Zoberieme teda každú dvojicu bodov, preložíme nimi priamku a spočítame, koľko bodov na nej leží (sedí, stojí). Zo všetkých týchto vyberieme maximum. Dvojíc bodov je rádovo n^2 a zakaždým treba skontrolovať zdruba n bodov, či na priamke leží (alebo čo). A na svete je kubické riešenie.

Dobre. Ak sa nad tým niekto nebodaj zamyslí, malo by mu zísť na um triedenie. Prečo triedenie? Utriedené veci sú pekné, lebo sa v nich ľahko vyhľadáva. Okrem iného tiež zoskupí rovnaké veci tesne za sebou.

Majme úlohu U'': zistiť tzv. modus postupnosti (číslo, ktoré sa v nej najčastejšie opakuje). Triviálna možnosť je zobrať si každé číslo ako kandidáta na modus a spočítať, koľko ostatných je rovnakých – $O(n^2)$. Lepšie riešenie je postupnosť utriediť. Ak je napríklad v postupnosti viacero 42-ojok, v utriedenej postupnosti budú všetky 42-ojky za sebou. Preto nám stačí raz pole prejsť a počítať pri tom, ako dlhé sú úseky rovnakých čísel. Triediť vieme v čase $O(n \log n)$ a potom to už iba v lineárnom čase prejdeme – spolu je to teda $O(n \log n)$.

Riešme teraz úlohu U': nájdite priamku, ktorá prechádza bodom $(0, 0)$ a je na nej najviac bodov. Súc poučený riešením úlohy U'', lepšie ako v čase $O(n)$ skúšať $O(n)$ priamok (spolu $O(n^2)$) je body polárne utriediť. To znamená, pre každý bod vypočítame uhol, pod ktorým ho vidíme (x-ová os má $0°$). Tak sa nám body, ktoré vidíme pod tým istým uhlom (t.j. tie, čo ležia na jednej priamke) dostanú vedľa seba. Potom stačí raz prejsť utriedeným poľom a pozerať čo najdlhšie úseky bodov s rovnakým uhlom.

No a tak dostávame vzorové riešenie aj našej úlohy U_0 – stačí tento postup zopakovať n-krát – každý bod na chvíľu spravíme počiatkom súradnicovej sústavy (premiestnime ho do $(0, 0)$ a (spolu)patrične posunieme aj ostatné body) a budeme hľadať priamku, ktorá ide tadiaľ.

Ako. Alebo: Poznámky k implementácii. Všimnime si, že na body, ktoré majú x-ovú súradnicu menšiu ako 0 môžeme kašľať (a aj kašleme). Je to preto, že každá priamka má nejaký najľavejší bod a aj z neho budeme niekedy začínať, takže žiadne body takto nestratíme.

Body triedime podľa smernice, t.j. podielu $\Delta y/\Delta x \ldots$ to je to číslo a v rovnici priamky $y = ax+b$. Činíme tak preto, lebo sa nám nechce počítať uhol (i keď by sa dal pomocou arkus tangensu, netreba ho). Treba si však dať pozor, že zvislé priamky sa takto zapísať nedajú. Tu to riešime tak, že namiesto $y_1/x_1 < y_2/x_2$ porovnávame $y_1x_2 < y_2x_1$, čo je to isté. Tu nám pomohlo, že sme sa vykašľali na záporné ix-y. Pre tie by totiž nerovnosť nemusela platiť (pri

násobení záporným sa znamienko otáča). Výhoda je, že nám to teraz funguje aj pre zvislé priamky (nulové x_1, x_2).

Všimnite si tiež, že v zdrojáku som nepoužil reálne čísla – tie sa mi vo viacerých riešeniach nepáčili (hoci som za to bod nestŕhal). Body na vstupe mali celočíselné súradnice a dalo sa to spraviť aj bez reálnych čísel. Na vašom mieste by som sa tak trochu obával, aby sa počítač nesekol na nejakom „bezvýznamnom“ mieste a hop, zrazu sa dve čísla nerovnajú (i keď ich rozdiel je iba dajme tomu 10^{-7}).

Lepšie? Dá sa aj lepšie. Nerád to robím, ale asi sa nevyhnem klišé „nad rámec tohto textu“. Záujemci si môžu pozrieť skriptá od Davida Mounta[4], kapitoly 8, 19 a 20 alebo sa popýtať svojho obľúbeného vedúceho na sústredku.

Listing programu:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
#define REP(i,n) for (int i=0; i<(n); ++i)

struct bod { int x, y; };

int cmp (const void *a, const void *b) {
  bod A = *(bod*)a, B = *(bod*)b;
  return A.y*B.x - B.y*A.x;
}

int main() {
  int N, max=0;
  bod p[MAX], d[MAX];

  scanf ("%d", &N);
  REP(i, N) scanf ("%d %d", &p[i].x, &p[i].y);
  if (N <= 2) { printf ("%d", N); return 0; }

  REP(i, N) {
    // zoberieme i-teho topiča ako stred súr. súst.
    int M = 0;
    REP(j, N) {
      d[j].x = p[j].x - p[i].x;
      d[j].y = p[j].y - p[i].y;
      if (d[j].x > 0 || (d[j].x == 0 && d[j].y > 0)) ++M;
    }

    // zotriedime podľa smernice
    qsort (d, M, sizeof(bod), cmp);

    // nájdeme rovnakých
    int r=1;
    REP(j,M-1)
      if (!cmp(&d[j], &d[j+1])) ++r;
      else { if (r>max) max = r; r = 1; }
    if (r>max) max = r;
  }

  printf ("%d\n", max+1);
  return 0;
}
```

[4] http://www.cs.umd.edu/ mount/754/Lects/754lects.pdf

7. Ohromná zima...

opravoval Mic
(max. 15 bodov)

Riešení neprišlo veľa, ale aspoň väčšina bola správna a dokonca bolo aj zopár vzorových `:-)`. Bodoval som nasledovne:

- Za riešenia v čase $O(M \log M)$ som dával 15 bodov.
- Kvadratické riešenia dostali najviac 13 bodov.
- Za časovú zložitosť $O(M^2 \log M)$ som udeľoval 12 bodov.
- Exponenciálne časové zložitosti mohli dostať najviac 9 bodov.
- Nefunkčné riešenia zvyknú dostávať málo bodov.
- Strhával som taktiež za nedostatočný popis, zlý odhad časovej zložitosti, prípadne za chyby v programe.

Ako na to? Dostal som grafovú úlohu, tak si zostrojím graf. Vrcholom v tomto grafe bude zastávka autobusu v nejakom čase. Ako vrchol budem označovať usporiadanú dvojicu (z, t) kde z je zastávka a t je čas, v ktorom som na zastávke. Ak ide autobus zo zastávky a na zastávku b v čase t a trvá mu to d, tak vyrobím dva vrcholy (a, t) a $(b, t + d)$ a spojím ich orientovanou hranou z prvého vrcholu do druhého s dĺžkou 0. Takto vyrobím najviac $2 \cdot M$ vrcholov a práve M hrán. V takomto grafe sa ale nedá čakať na zastávke. Pre každú zastávku z zoberme vrcholy tvaru (z, t), kde t môže byť ľubovoľné. Utrieďme si tieto vrcholy podľa času, takže dostaneme vrcholy (z, t_i) pričom $t_i \leq t_{i+1}$. Každý vrchol (z, t_i) spojíme orientovanou hranou s vrcholom (z, t_{i+1}) s dĺžkou $t_{i+1} - t_i$. Ľudskou rečou povedané, v ľubovoľnom vrchole si môžeme povedať, že čakáme na najbližší príchod alebo odchod autobusu. Ak chceme čakať ešte na ďalší autobus, tak počkáme na najbližší, odignorujeme ho, a čakáme, a ignorujeme až kým nepríde náš vysnívaný autobus. Týmto spôsobom pridáme do grafu maximálne $2 \cdot M$ hrán. Aké vlastnosti bude mať náš graf? Bude orientovaný, acyklický a riedky (má $O(M)$ vrcholov a tiež len $O(M)$ hrán). Je ľahké nahliadnuť, že dĺžka cesty medzi dvomi vrcholmi v tomto grafe zodpovedá dĺžke čakania na zastávke. Keďže musíme triediť a triedenie je najzložitejšia operácia, tak časová zložitosť zostrojenia grafu je $O(M \log(M))$.

Do grafu pridáme ešte vrchol $(1, 0)$, čo je nástupná zastávka v čase 0 a hranu k prvému autobusu. Teraz nám už stačí nájsť najkratšiu cestu z tohto vrcholu do vrcholu, ktorý je na poslednej zastávke. Keďže graf je acyklický, vieme túto cestu nájsť v lineárnom čase. Ako? Už pri vytváraní grafu si všetky hrany obrátime. Spravíme si nasledovnú rekurzívnu funkciu. Ako parameter dostane vrchol v. Jej výstupom bude dĺžka najkratšej cesty z prvého vrchola do vrchola v. Popri tom si ešte pre vrchol v zaznačí, ktorý vrchol ho predchádza na najkratšej ceste. Ako bude táto funkcia fungovať? Najskôr sa pozrie, či náhodou už pre vrchol v nebola zavolaná. Ak áno, tak predsa výsledok sme už raz vypočítali a nepotrebujeme ho rátať znovu! Stačí vrátiť uložený výsledok. Ak nie, tak výsledok predsa len musíme vypočítať.

Lenže najkratšia cesta vedie určite cez nejakého môjho suseda (áno, cez suseda, lebo sme otočili hrany). Tak sa teda zavoláme na svojho suseda u a zistíme, aká je najkratšia cesta z vrcholu 1 doň. Keď k nej prirátame dĺžku hrany vedúcej z v do u, tak získame dĺžku najkratšej cesty z 1 do v (v pôvodnom grafe), ak predposledný vrchol na nej bol u. Keďže predposledný vrchol na najkratšej ceste môže byť ľubovoľný sused, musíme tieto dĺžky vypočítať pre každého z nich. Najkratšia cesta teda nutne musí byť taká dlhá, ako minimum z nich. Taktiež si nesmieme zabudnúť zaznačiť, cez ktorého suseda vedie najkratšia cesta (spravíme si šípku, kam ísť). Toto má však jeden háčik. Čo ak vrchol nemá žiadneho suseda (teda v reči autobusov, na danú zastávku do toho času žiadne autobusy nechodia)? To však môže znamenať, že buď je to nedosiahnuteľný vrchol, alebo prvý vrchol. Ak je to prvý, tak môžem bez ostychu vrátiť nulu, (prečo?). Ale keď nie, tak by som mal vrátiť ∞. Je ľahké

nahliadnuť, že toto bude trvať $O(M)$. Ale pozor, ak by som si nezapamätával už raz vyrátané hodnoty, tak by celková zložitosť tejto časti narástla až na $O(M!)$. Poriadne si rozmyslite, prečo!

Dĺžku najkratšej cesty už teraz vieme veľmi jednoducho zistiť. Stačí, keď túto funkciu zavoláme na každý vrchol, ktorý prislúcha cieľovej zastávke. Ale ako je to so samotnou najkratšou cestou? Zoberme vrchol, v ktorom končí najkratšia cesta. V ňom je zakreslená šípka. Prejdime po nej. Prídeme do ďalšieho vrcholu, kde je znovu šípka. Takto pôjdeme po šípkach, až kým neprídeme do počiatočného vrcholu. Takto si ľahko zrekonštruujeme výslednú cestu a potom ju už môžme vypísať.

Ako je to ale s časovou zložitosťou? Nájdenie najkratšej cesty je síce v čase $O(M)$, ale vytvorenie grafu nám potrvá až $O(M \log M)$, takže celková časová zložitosť bude $O(M \log M)$. Nemohlo by to ísť rýchlejšie? Nie, nemohlo. Prečo? Keby sme vedeli túto úlohu riešiť v čase $O(f(N))$, $f(N) \geq N$, vedeli by sme aj triediť v čase $O(f(N))$. Na domácu úlohu si to skúste dokázať[5]. Pamäťová náročnosť tohto algoritmu je $O(M + N)$.

Listing programu:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

#define ZACIATOK 1
#define KONIEC 0

#define MAX_INT 1000000

struct vrchol{
    int cas,typ;
    vector<pair<vrchol*,int> > hrany;
    int cakanie,kadial,povod;
};

struct cmp{
    bool operator()(vrchol* const &a,vrchol* const &b)const{
        if(a->cas==b->cas)return a->typ<b->typ;
        return a->cas<b->cas;
    }
};

vector<vector<vrchol*> > G;
int N,M;

int pocitaj(vrchol *v){
    if(v->cakanie>=0)return v->cakanie;
    int min=MAX_INT;
    int kadial=-1;
    for(unsigned int  i=0;i<v->hrany.size();i++){
        int m=pocitaj(v->hrany[i].first)+v->hrany[i].second;
        if(m<min) {
            min=m;
            kadial=i;
        }
    }
    if(v->povod==-1)min=0;
```

[5] **Návod:** nájdite spôsob, ako prerobiť postupnosť čísel na autobusy v lineárnom čase tak, že keď na to pustíte spomínaný algoritmus, tak z výstupu budete vedieť vyrobiť utriedenú postupnosť opäť v lineárnom čase.

```
    v->cakanie=min;
    v->kadial=kadial;
    return min;
}

int vypis_cestu(vrchol *v){
    if(v->kadial==-1)return 0;
    vypis_cestu(v->hrany[v->kadial].first);
    if((v->hrany[v->kadial].second==0) &&
       (v->cas-v->hrany[v->kadial].first->cas!=0))
        cout << v->povod << endl;
    return 0;
}

int main(){
    cin >> N >> M;
    G.resize(N);
    vrchol prvy;
    prvy.cas=0;
    prvy.typ=KONIEC;
    prvy.cakanie=prvy.kadial=prvy.povod=-1;
    G[0].push_back(&prvy);
    for(int i=0;i<M;i++){
        int a,b,t,d;
        vrchol *v1=new vrchol(),*v2=new vrchol();
        cin >> a >> b >> t >> d;
        v1->cas=t;
        v2->cas=t+d;
        v1->typ=ZACIATOK;
        v2->typ=KONIEC;
        v1->cakanie=v2->cakanie=v1->kadial=v2->kadial=-1;
        v1->povod=v2->povod=i;
        v2->hrany.push_back(make_pair(v1,0));
        G[a].push_back(v1);
        G[b].push_back(v2);
    }
    for(int i=0;i<N;i++){
        sort(G[i].begin(),G[i].end(),cmp());
        for(unsigned int j=1;j<G[i].size();j++){
            G[i][j]->hrany.push_back(
                make_pair(G[i][j-1],G[i][j]->cas-G[i][j-1]->cas));
        }
    }
    int index=0;
    int min=pocitaj(G[N-1][0]);
    for(unsigned int i=1;i<G[N-1].size();i++){
        int m=pocitaj(G[N-1][i]);
        if(m<min){
            m=min;
            index=i;
        }
    }
    cout << min << endl;
    vypis_cestu(G[N-1][index]);
}
```

8. Otravná chémia

opravoval Lukáš
(max. 15 bodov)

Vaše riešenia by sa dali rozdeliť na veľmi pomalé a celkom rýchle. Veľmi pomalé získali najviac 7 bodov a rýchlejšie 10 a viac. Prvé riešenie prechádza všetky možnosti rozdelenia fliaš do dní. Počet týchto možností môžeme zhora odhadnúť číslom $2^{N-1+M-1}$. Totiž v poličke dĺžky k máme $k-1$ možností, kde sa môže končiť nejaký súvislý úsek fliaš, ktorý sme zobrali počas dňa – sú to medzery medzi fľašami. No a pre každé miesto máme dve možnosti, teda možných rozdelení fliaš do dní je 2^{k-1}. Keď to vynásobíme pre obidve poličky, dostaneme $2^{N-1+M-1}$. Tento odhad by sa dal ešte zlepšiť, ale nie veľmi výrazne.

Poďme však ku vzorovému riešeniu. Najprv si povedzme dôležité tvrdenie, ktoré si asi polovica z vás všimla a ktoré nám pomôže znížiť časovú zložitosť. Predpokladajme, že by sme v nejakom dni zobrali z prvej poličky $a+b$ a z druhej $c+d$ chemikálií (týmto zápisom chcem povedať, že obidva úseky sa dajú rozdeliť na dve menšie časti veľkostí a, b resp. c, d). Ak ich zmiešame, dostaneme $(a+b)(c+d) = ac+bd+ad+bc$ dymu. Keby sme tieto množstvá miešali v dvoch dňoch, dostaneme $ac+bd$ dymu. No a keďže $ad+bc > 0$, dostávame $(a+b)(c+d) = ac+bd+ad+bc > ac+bd$. Z toho vyplýva, že každú dlhú postupnosť fliaš môžeme deliť na menšie časti. Ďalej ľahko vidíme, že v optimálnom riešení berieme v každom dni z niektorej poličky iba jednu fľašu.

Vaše rýchlejšie riešenia väčšinou využívali predchádzajúce tvrdenie a dynamické programovanie. Vyskytlo sa však aj jedno riešenie využívajúce Dijkstrov algoritmus. Pre zaujímavosť si skúste rozmyslieť, čo by boli vrcholy a hrany. Vzorové riešenie bude vypĺňať tabuľku veľkosti NM. Hodnota $x(i,j)$ vyjadruje, koľko najmenej dymu vyrobíme tak, že zoberieme i fľašiek z prvej poličky a j z druhej (v obdivoch poličkách berieme tieto fľaše zaradom od začiatku). Nech $a_1, \ldots, a_N$ sú fľaše na prvej poličke a $b_1, \ldots, b_M$ fľaše na druhej poličke. Potom platí nasledovný vzorec:

$$(\forall i,j \geq 1)\ \ x(i,j) = \min\{x(i-1,j), x(i,j-1), x(i-1,j-1)\} + a_i b_j$$

$$x(0,0) = 0$$

$$\text{inak}:\ x(i,j) = \infty$$

Tento vzorec dvaja z vás použili, ale bez dôkazu. A ako uvidíme ďalej, toto tvrdenie nie je triviálne.

Ak sme už použili i fliaš z prvej poličky a j z druhej, určite sme museli zmiešať spolu posledné fľaše z prvej a druhej poličky. Preto je vo vzorci použitý výraz $a_i b_j$. Môžu nastať tri možnosti:

- Ak fľašami a_i a b_j začíname nový deň, potom použijeme hodnotu $x(i-1,j-1)+a_i b_j$. Toto je zrejmé.
- Ak optimálna hodnota $x(i-1,j)$ vznikla tak, že j-ta fľaša bola v poslednom dni zmiešaná s niekoľkými fľašami z prvej poličky, teda $x(i-1,j) = x(k-1,j-1) + (a_k + \ldots + a_{i-1})b_j$, potom môžeme použiť hodnotu $x(i-1,j) + a_i b_j$. Totiž určite $x(i,j) \leq x(k-1,j-1) + (a_k + \ldots + a_{i-1})b_j + a_i b_j$.
- Môže sa však stať, že optimálna hodnota $x(i-1,j)$ vznikne tak, že zmiešame fľaše a_{i-1} s $b_l, \ldots, b_j$, pričom $l < j$ (teda berieme aspoň dve fľaše z druhej poličky). Hodnota $x(i-1,j) + a_i b_j$ by potom vyjadrovala situáciu, keby sme v jeden deň zmiešali fľaše a_{i-1} s $b_k, \ldots, b_j$ a neskôr aj a_i s b_j. A to je nekorektné vzhľadom na zadanie, lebo do výsledku zarátavame $a_{i-1}(b_k + \ldots + b_j) + a_i b_j$ namiesto $(a_{i-1}+a_i)(b_k+\ldots+b_j)$. Avšak toto nebude vadiť. Všimnime si, že $x(i-1,j) = x(i-2,l-1) + a_{i-1}(b_l + b_{l+1} \ldots + b_j)$. Potom však $x(i-1,j-1) \leq x(i-2,l-1) + a_{i-1}(b_l + b_{l+1} \ldots + b_{j-1}) \leq x(i-1,j)$ a $x(i-1,j-1) \leq x(i-1,j)$. Nekorektná hodnota $x(i-1,j) + a_i b_j$ teda nebude lepšia ako iné korektné a nebudeme ju brať do úvahy.

Situáciu pre $x(i, j-1)$ nebudeme rozoberať, lebo je rovnaká ako pre $x(i-1, j)$.

Vďaka jednoduchému vzorcu je náš program krátky a zjavne má časovú zložitosť $O(NM)$. Pri rátaní hodnôt $x(i, j)$ si nemusíme pamätať celú tabuľku, ale stačia len posledné dva riadky. Pamäťová zložitosť je preto $O(\min\{N, M\})$ – vyberieme si kratšiu z dvoch poličiek, ktorú použijeme ako riadok.

Listing programu:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    vector<int> a(n), b(m);
    for (int i=0; i<n; i++)
        scanf("%d", &a[i]);
    for (int i=0; i<m; i++)
        scanf("%d", &b[i]);
    if (n<m) {
        swap(a, b);
        swap(n, m);
    }
    vector<int> x[2]={vector<int>(m), vector<int>(m)};

    x[0][0]=a[0]*b[0];
    for (int i=1; i<m; i++)
        x[0][i]=a[0]*b[i]+x[0][i-1];

    for (int i=1; i<n; i++) {
        int u=(i+1)%2;
        x[i%2][0]=x[u][0]+a[i]*b[0];
        for (int j=1; j<m; j++)
            x[i%2][j]=min(x[u][j-1], min(x[u][j], x[i][j-1]))+a[i]*b[j];
    }
    printf("%d\n", x[(n+1)%2][m-1]);
}
```

9. Tajuplný Egypt

opravoval Mic
(max. 15 bodov)

Ach jaj... Veľmi ste ma nepotešili. Zo všetkých riešení boli len 2 vzorové a ešte jedno malo tú česť dostať nadpolovičný počet bodov. Zase je pravda, že príklad nebol až taký jednoduchý. Tak sa poďme pozrieť na to, ako som bodoval:

- Za riešenia v čase $O(N \log(N))$ som dával 15 bodov.
- Backtrack mohol dostať 9 bodov.
- Nefunkčné riešenia dostali najviac 4 body.
- Ďalej sa dali body stratiť za chýbajúci alebo nedostatočný popis, chýbajúci kód, zlý odhad časovej zložitosti.

Tiež by som rád upozornil na to, že treba čítať upozornenia[6]. Konkrétne upozornenie bolo, že tento príklad nie je až taký jednoduchý, ako sa môže na prvý pohľad zdať. Takže ak

[6] aj toto:-)

to vymyslím veľmi rýchlo, tak by sa ešte oplatilo chvíľku zamyslieť, či som ešte nezabudol na nejaký detail, za ktorý môžem stratiť body...

Ale poďme späť k vzoráku. Čo bolo treba spraviť? Tak sa na to pozrime. Najskôr si budeme musieť čísla utriediť, aby sme vedeli, na aké miesto má ktoré ísť. Zoberme si ľubovoľný prvok. Nájdime si taký prvok, ktorý chce ísť na jeho miesto a vymeňme ich. Tým sme dostali jeden prvok na miesto, kde chce byť a jeden prvok sa dostal na nové miesto. Teraz spravme znovu to isté. Nájdime prvok, ktorý chce ísť na miesto, kde je náš terajší prvok a vymeňme ich. Toto budeme robiť dovtedy, kým sa náš prvok nedostane na svoje miesto. Takto sme dostali na miesto niekoľko prvkov. Tieto prvky patria spolu do jedného cyklu. Všimnite si, že každý prvok patrí do nejakého cyklu[7]. Koľko nás stojí usporiadanie cyklu hore napísaným spôsobom? Nech S je súčet prvkov v cykle, a je cena nášho vybratého prvku a k je počet prvkov cyklu. Potom cena uporiadania je $C = S + a \cdot (k-2)$. Ako dosiahnuť, aby táto cena bola čo najmenšia? S zmenšiť nevieme, lebo každý prvok musíme presunúť aspoň raz. k je pevne dané, takže musíme minimalizovať a. Zoberme teda za a minimum z nášho cyklu.

Takto dostávame náhľad na optimálne riešenie. Skrátka týmto spôsobom usporiadame všetky cykly a máme vyhraté. Ale POZOR! Toto ešte stále nie je úplne správne. Čo sa nám môže stať? Predstavme si dva cykly, oba dĺžky 10, jeden ale s minimálnym prvkom 1 a druhý s minimálnym prvkom 100. Prvý cyklus nech má súčet $S_1 \geq 10$, druhý nech má súčet $S_2 \geq 1000$. Potom cena za usporiadanie naším spôsobom bude $S_1 + S_2 + 8 \cdot (1+100)$. Čo keby sme prvý cyklus usporiadali normálne ale druhý cyklus usporiadali tak, že minimálny prvok z prvého cyklu vymeníme za minimálny prvok z druhého cyklu, usporiadame druhý cyklus (s tým že tá jednotka pôjde na miesto kam chce ísť 100). A po usporiadaní ich zase vymeníme naspäť. Cena tohto usporiadania je $S_1 + (S_2 - 100 + 1) + 8 \cdot (1+1) + 2 + 200$. Keď ich porovnáme, tak toto druhé prefíkané riešenie bude výhodnejšie. Teraz sa na to pozrime poriadnejšie. Nech minimálny prvok z celej postupnosti je a. Nech súčet prvkov v cykle je S, dĺžka cyklu je k a minimálny prvok v cykle je b. Potom obyčajné usporiadanie cyklu nás stojí $S + b \cdot (k-2)$. Prefíkané usporiadanie cyklu nás stojí $S - b + a + a \cdot (k-2) + 2 \cdot (a+b) = S + a \cdot (k+1) + b$. Ale prefíkane sa nám nie vždy oplatí, preto sa vždy musíme rozhodnúť kedy ho použijeme. Použijeme ho práve vtedy, keď $S + b \cdot (k-2) > S + a \cdot (k+1) + b$, z čoho po upravení dostaneme $b \cdot (k-3) > a \cdot (k+1)$. Nie je ťažké nahliadnuť, že toto je práve optimálne preusporiadanie ľubovoľného cyklu[8].

Ako teda bude fungovať náš algoritmus? Usporiadame si prvky, pričom pri každom prvku si budeme pamätať aj jeho pôvodné umiestnenie. Po preusporiadaní číslo v pôvodnom usporiadaní nám povie, ktorý prvok chce ísť na naše miesto. Pozrieme sa teda na prvok na tom mieste a na jeho pôvodnú pozíciu. Toto robíme, až kým sa nevrátime k nášmu prvému prvku. Takto nájdeme cyklus a počas hľadania cyklu nájdeme jeho súčet, minimálny prvok a jeho dĺžku. Z týchto hodnôt a z hodnoty minimálneho prvku spomedzi všetkých prvkov vieme zistiť, ako to vieme čo najlepšie utriediť. Po nájdení všetkých cyklov[9] nám stačí sčítať ceny usporiadaní každého cyklu a máme vyhrané. Teda takmer. Ešte sa nám môže stať, že máme na vstupe niekoľko rovnakých prvkov a aspoň jeden z nich je na správnej pozíci. Vtedy sa môže stať, že nejaký opakovaný prvok sa chce presunúť na iné miesto a pritom už je na dobrom mieste. Preto pri hľadaní cyklu treba myslieť na to, že do súčtu (a dĺžky) sa nesmú zarátavať prvky, ktoré sú na mieste, kde bol taký istý prvok pred usporiadaním.

Ako je to s časovou zložitosťou? Triedenie nám zaberie $O(N \log(N))$ času. Ostatné časti riešenia sú lineárne(všimnite si, ze každý prvok prehľadávame práve raz). Takže celková časová zložitosť je $O(N \log(N))$ a pamäťová zložitosť je $O(N)$.

[7] ak je prvok už od začiatku na správnom mieste, tak je v cykle dĺžky 1

[8] všimnite si, že nemusíme kontrolovať, či náhodou prvok b nie je ten istý ako a, lebo vtedy daná nerovnosť nie je splnená

[9] pre každý prvok si pamätáme, či sme v ňom boli, aby sme nerobili tú istú robotu viac krát

Listing programu:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main(){
    int N;
    vector<pair<int,int> > cisla;
    vector<int> povodne;
    vector<bool> was;
    cin >> N;
    was.resize(N,false);
    for(int i=0;i<N;i++){
        int a;
        cin >>a;
        cisla.push_back(make_pair(a,i));
        povodne.push_back(a);
    }
    sort(cisla.begin(),cisla.end());
    int minimum=cisla[0].first;
    for(int i=1;i<N;i++)minimum=min(minimum,cisla[i].first);
    int praca=0;
    for(int i=0;i<N;i++){
        if(!was[i]&&cisla[i].second!=i){
            int m=cisla[i].first;
            int sum=0;
            int len=0;
            int point=i;
            do{
                was[point]=true;
                if(povodne[point]!=cisla[point].first){
                    sum+=povodne[point];
                    len++;
                    m=min(m,povodne[point]);
                }
                point=cisla[point].second;
            }while(i!=point);
            if(m*(len-3)>minimum*(len+1)){
                praca+=sum+minimum*(len+1)+m;
            }else{
                praca+=sum+m*(len-2);
            }
        }
    }
    cout << praca << endl;
    return 0;
}
```

10. Tvoja matrioška, či moja?

Opravoval Lukáš
(max. 15 bodov)

V riešeniach tohto príkladu sa vyskytli dva prístupy bežiace v polynomiálnom čase. Prvý využíva dynamické programovanie. Najprv matriošky zotriedime vzostupne podľa šírky. Potom pre matriošky s indexami $i, j, i < j$ platí, že j-tu nemôžeme vložiť do i-tej. Hodnota $a(k, l, m)$ hovorí, či existuje také rozdelenie prvých k matriošiek do dvoch postupností, že

jedna postupnosť sa končí matrioškou l a má dĺžku m. Navyše platí $l < k$. A keď sa lepšie zamyslíme, tak druhá postupnosť má dĺžku $k - m$ a musí sa končiť matrioškou k, lebo tá je z prvých k matriošiek najširšia. No a na zistenie, či je $a(k, l, m)$ pravda alebo nie, nám stačí pozrieť sa na políčko $a(k-1, l, m)$. Ak $l = k - 1$, tak musíme ešte skontrolovať všetky $a(k-1, 0, k-m), \ldots, a(k-1, k-2, k-m)$. Pri kontrole políčka zistíme, či môžeme na koniec postupnosti doplniť k-tu matriošku.

Detaily implementácie domyslite na domácu úlohu. Toto riešenie má časovú aj pamäťovú zložitosť $O(N^3)$. Možno sa na prvý pohľad zdá, že časová zložitosť je $O(N^4)$, ale nie je to pravda. My to však vieme lepšie a preto sa týmto riešením nebudeme zaoberať. Okrem toho ak by ste ho chceli naozaj písať, mohli by ste sa pomýliť pri plusoch a mínusoch, ktoré sa tam vyskytujú. Teraz sa pustime do vzorového riešenia, ktorý beží v čase $O(N^2)$.

Preveďme si zadanie do teórie grafov. Každá matrioška predstavuje jeden vrchol a matriošky sú spojené hranou práve vtedy, ak ich nemôžeme do seba vložiť. Zo zadania vieme, že matriošky sa dajú rozdeliť do dvoch postupností. Predstavme si, že máme nejaké riešenie a vrcholy zafarbíme dvomi farbami podľa toho, v ktorej postupnosti sa nachádzajú. Všimnime si, že na koncoch jednej hrany musia byť rôzne farby. Grafy, ktoré vieme takto zafarbiť dvomi farbami voláme bipartitné a časti s rovnakou farbou voláme partície.

Prvá fáza nášho riešenia bude spočívať v tom, že nájdeme všetky komponenty nášho grafu a každý z týchto komponentov nejak ofarbíme dvomi farbami. Najľahšie sa nám to bude robiť prehľadávaním do hĺbky. Ľubovoľný ešte nepreskúmaný vrchol v ofarbíme farbou 1. Potom prehľadáme komponent v ktorom je vrchol v, pričom ak má nejaký vrchol farbu f, tak všetci jeho susedia majú farbu $3 - f$. Na neofarbených susedov sa rekurzívne zavoláme.

Také jednoduché to však nebude, a preto ešte nasleduje druhá fáza. Graf sme celý nejak ofarbili (resp. rozdelili na dve partície), ale nemusí platiť, že počet vrcholov obidvoch farieb je N. Môžeme si však pomôcť. Nech počet vrcholov farby 1 v celom grafe je p_1, počet vrcholov farby 2 je p_2. Nech v nejakom komponente grafu je a vrcholov farby 1 a b vrcholov farby 2. Keby sme v tomto komponente farby vymenili, dostali by sme zafarbenie celého grafu také, že počty dvoch farieb sú $p_1 + b - a$ a $p_2 + a - b$.

Zamyslime sa lepšie, čo vlastne chceme dosiahnuť. Nech počet komponentov grafu je k. Ďalej vieme, že v i-tom komponente je veľkosť dvoch partícií $z(1, i)$ a $z(2, i)$. Chceme vhodne prefarbiť partície, aby sme dostali želané ofarbenie s rovnakým počtom vrcholov jednej aj druhej farby. Povedané formálnejšie: chceme nájsť takú postupnosť $r_1, \ldots, r_n$, že $z(r_1, 1) + z(r_2, 2) + \ldots + z(r_k, k) = N$.

Ako to už pri takýchto problémoch býva, využijeme dynamické programovanie. Nech $a(i, s)$ je pravda, ak vieme prvých i komponentov prefarbiť tak, že dostaneme s vrcholov farby 1. Tieto hodnoty získame ľahko. Pre $i = 1$ sú pravdivé iba hodnoty $a(1, z(1, 1))$ a $a(1, z(2, 1))$, ostatné sú nepravdivé. Nech hodnota $a(i, s)$ je pravda. Potom aj hodnoty $a(i+1, s+z(1, i+1))$ a $a(i + 1, s + z(2, i + 1))$ sú pravdivé. Keďže chceme na konci ešte zrekonštruovať farby v komponentoch, musíme si v nejakom poli navyše pamätať, či sa $a(i, s)$ stala pravdivou vďaka $z(1, i)$ alebo $z(2, i)$.

Riešenie máme, poďme sa už len presvedčiť, že máme naozaj takú časovú zložitosť, akú sme chceli. Zostrojenie grafu a prehľadávanie do hĺbky s nájdením partícií nám zaberie čas $O(N^2)$, keďže vrcholov je $2N$ a hrán najviac $4N^2$. V druhej fáze vypĺňame tabuľku veľkosti najviac $2N \times 2N$ – pretože komponentov grafu je najviac $2N$. Každé políčko spracujeme v konštantom čase, teda tu máme opäť časovú zložitosť $O(N^2)$. No a nakoniec už len musíme zrekonštruovať riešenie z vyplnenej tabuľky a utriediť obe výsledné postupnosti. Táto fáza má zložitosť $O(N \log N)$ a teda dokopy je časová aj pamäťová zložitosť $O(N^2)$.

Listing programu:

```
#include <vector>
#include <algorithm>
#include <utility>
```

```
#include <cstdio>
#include <cstring>
using namespace std;

#define REP(i,n) for(__typeof(n) i=0; i<(n); ++i)
#define FOR(i,a,b) for(__typeof(b) i=a; i<(b); ++i)

typedef pair<int, int> PI;
typedef pair<int, PI> TRI;
int h[200], d[200], w[200], n, f[200];
bool da(int a, int b) {
    //dajú sa do seba vložiť bábiky číslo a, b?
    if (h[a]<h[b] || (h[a]==h[b] && d[a]<d[b])) swap(a, b);
    return h[a]>=h[b] && d[a]-2*w[a]>=d[b];
}
bool g[200][200], bol[200];
vector<vector<int> > z[2];
void prehl(int j) {
    //ofarbujeme graf dvomi farbami
    bol[j]=true;
    z[f[j]][z[0].size()-1].push_back(j);
    REP(i,2*n) if (g[j][i] && !bol[i]) {
        f[i]=!f[j];
        prehl(i);
    }
}
void vypis(vector<int> &z) {
    vector<TRI> t;
    REP(i,z.size())
        t.push_back(TRI(h[z[i]] , PI(d[z[i]], w[z[i]]) ));
    sort(t.begin(), t.end());
    for (int i=t.size()-1; i>=0; i--)
        printf("%d %d %d\n", t[i].first, t[i].second.first, t[i].second.second);
}
int main() {
    scanf("%d", &n);
    REP(i,2*n) scanf("%d %d %d", &h[i], &d[i], &w[i]);
    fill(bol, bol+2*n, false);
    REP(i,2*n) REP(j,2*n)
        g[i][j]=!da(i, j);
    REP(i,2) z[i].clear();

    REP(i,2*n) if (!bol[i]) {
        //našli sme vrchol v neofarbenom komponente
        //tak ho ofarbíme
        REP(j,2) z[j].push_back(vector<int>());
        f[i]=0;
        prehl(i);
    }

    //v poli z máme uložené všetky partície
    //podľa veľkostí particií teraz nájdeme riešenie
    vector<vector<bool> > a(z[0].size()+1, vector<bool>(n+1, false) ), kto=a;
    a[0][0]=true;
    REP(i,z[0].size())
        REP(j,n+1) if (a[i][j])
            REP(k,2) {
                int u=z[k][i].size()+j;
                if (u<=n) {
                    a[i+1][u]=true;
```

```
                kto[i+1][u]=k;
            }
        }

    vector<int> r(z[0].size(), 1); int u=n;
    for (int i=z[0].size(); i>0; i--) {
        //zrekonštruujeme riešenie
        r[i-1]=kto[i][u];
        u-=z[r[i-1]][i-1].size();
    }
    vector<int> q[2];
    REP(i,r.size()) {
        q[r[i]].insert(q[r[i]].end(), z[0][i].begin(), z[0][i].end());
        q[!r[i]].insert(q[!r[i]].end(), z[1][i].begin(), z[1][i].end());
    }
    vypis(q[0]);
    printf("-\n");
    vypis(q[1]);
}
```

Proboj Online

Máš pocit, že na sústredení KSP nie je dosť času na proboj? Chýba ti niečo také aj doma? Prípadne sa ti ešte nepodarilo dostať na sústredenie a rád by si zistil o čo ide? Tak préve preto sme pre teba spravili **Proboj Online**. Celý ten zázrak nájdeš na adrese

`http://ksp.sk/~proboj`

Dúfame, že sa zapojíte v hojnom počte.

Programátorská liaheň

Zdajú sa ti úlohy KSP príliš ťažké? Rád by si sa dozvedel, čo je to tá zložitosť, teória grafov či dynamické programovanie, no nemáš odkiaľ? Máš chuť riešiť príklady nielen teoreticky, ale aj „naostro“?

Tak práve pre teba je tu naša skvelá **Programátorská liaheň**, ktorá ti pomôže „vyliahnuť sa“ – dočítaš sa o rôznych oblastiach súvisiacich s algoritmami a všetko si budeš môcť vyskúšať pri riešení príkladov, podobných ako v KSP. V Liahni však jediné, o čo ide pri riešení príkladu, je napísať funkčný program, ktorý ti automaticky otestujeme.

Všetci ste vítaní, budeme len radi, ak sa vás do riešenia úloh a čítania textov v našej Liahni zapojí čo najviac.

Všetko potrebné nájdete tu:
`http://ksp.sk/~liahen/`

Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
1	Kostolányi Peter	Gym. Jura Hronca BA	3	49	14	15	14	12	12	116
1	Ondruška Peter	SPŠ Dubnica nad Váhom	3	57	13	12	15	12	7	116
1	Varga András	Gym. H. Selyeho Komárno	3	57	14	15	8	14	8	116
4	Herencsár Albert	Gym. maď. Galanta	2	44	13	15	12	12	12	108
5	Fecko Stanislav	Gym. Pankúchova Bratislava	4	35	15	14	7	11	8	90
5	Hagara Michal	Gym. Jura Hronca BA	1	38	13	5	14	13	7	90
7	Súkeník Ján	Gym. Lettricha Martin	3	37	15	8	9	10	8	87
8	Hozza Michal	Gym. Jura Hronca BA	2	23	15	15	12	12	8	85
9	Simanová Lucia	Gym. Grösslingová BA	3	38	14	13	15		3	83
10	Štefanišin Peter	Gym. Svidník	4	37		15	4	15	8	79
11	Fekiač Jozef	Gym. Grösslingová BA	2	25	15	15	15		8	78
12	Kudláč Matúš	Gym. Bilíkova BA	3	26		15	15	12	8	76
13	Kučin Alexander	Gym. Lipany	3	22	14	15	9	5	8	73
14	Polák Lukáš	Gym. Jura Hronca BA	1	23	13	5	14	5	11	71
15	Boško Lukáš	Gymnázium Považská Bystrica	0	26	8	15	8	5	8	70
15	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	26	13	12	2	10	7	70
17	Piter Miroslav	Gym. Svidník	0	31		15	4	11	8	69
18	Hojčka Michal	Gym. Partizánske	2	14	13	15	4	10	8	64
18	Tureková Katarína	Gym. Tajovského B. Bystrica	3	13	15	15	9		12	64
20	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	25		15	10		8	58
20	Saleh Adam	Gym. Jura Hronca BA	3	34				13	11	58
22	Molčan Dávid	Gym. Slovenská Bardejov	4	30		7	5	7	7	56
23	Magdolen Matej	Gym. Golianova Nitra	1	0	15	14	8	10	8	55
24	Matula Peter	Gym. Školská Turč. Teplice	3	17	13	6	10	1	7	54
25	Bene Daniel	Gym. Haličská Lučenec	3	30		15	8			53
25	Kvasnička Igor	Gym. Jura Hronca BA	3	16	15	11			11	53
27	Liška Igor	Gym. Jura Hronca BA	2	21	13	11	7			52
28	Bachratý Martin	Gym. Veľká Okružná Žilina	2	27		14	9			50
28	Halabuk Milan	Gymnázium Levice	4	24	6	6	6		8	50
28	Maršík Ladislav	Gym. Grösslingová BA	4	0	13	15	13	9		50
31	Končok Jakub	Gym. Haličská Lučenec	3	37		12				49
32	Balogh Tomáš	SPŠE Nové Zámky	2	10	13	10	5	7	3	48
33	Chrzan Martin	Gym. Ľ. Štúra Zvolen	2	28		8	7	3		46
34	Košdy Martin	Gym. Jura Hronca BA	2	0		15	15		11	41
35	Matejovičová Lenka	Gym. Grösslingová BA	3	37						37
36	Šulák Viktor	SPŠE Nové Zámky	2	16		9	3		7	35
37	Kohút Matej	Gym. Jura Hronca BA	2	5		12	8		8	33
38	Truben Martin	Gym. Jura Hronca BA	3	0	15	13	4			32
39	Chudý Lukáš	Gym. Liptovský Hrádok	2	28						28
39	Szabadosová Emília	Škola pre mim. nadané deti BA	3	0	8	12	8			28
41	Dittrich Andrej	Gym. Jura Hronca BA	3	0	15				12	27
41	Kompan Martin	Gym. Partizánske	4	27						27
43	Buchovecká Simona	Gym. Medzilaborce	4	5		10	9			24
44	Knotek Tomáš	SPŠ Nitra	4	23						23
45	Kuchyňár Martin	Gym. Jura Hronca BA	1	0		15	7			22
46	Kedrovič Ondrej	Gym. Jura Hronca BA	0	9		6	2		3	20
47	Babej Tomáš	Gymnázium	0	16						16
47	Greň Peter	Gym. L. Stöckela Bardejov	1	16						16
47	Peťura Oto	SPŠE Michalovce	1	16						16
50	Duchoň Gabriel	Gym. maď. Želiezovce	4	11						11

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
51	Zelenaj Roman	Gym. Golianova Nitra	2	10						10
52	Migaš Jozef	Gym. Jura Hronca BA	0	9						9
53	Mamojko Štefan	Gym. Jura Hronca BA	0	8						8
54	Végh Ladislav	Gym. Tornaľa	2	7						7
55	Milko Matej	Gym. Jura Hronca BA	3	0		6				6

Výsledková listina po 2. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda		4	5	6	7	8	Σ
1	Okruhlica Adam	Gym. Jura Hronca BA	4	71	16	12	15	15	14	143
2	Danilák Michal	Gym. Hubeného BA	4	68	13	12	15	14	13	135
3	Rampášek Ladislav	Gym. Jura Hronca BA	4	50	16	12	15	13	12	118
4	Brezáni Samuel	Gym. Rajec	4	57	12	12	15	11	10	117
5	Ondruška Peter	SPŠ Dubnica nad Váhom	3	52	12	7	15	15	10	111
6	Hlavatý Peter	Gym. Jura Hronca BA	3	50	8	12	12	11	13	106
7	Boža Vladimír	Gym. Tatarku Poprad	3	37	14	12	15	15	12	105
8	Hapák Samuel	Gym. Grösslingová BA	3	34	16	12	12	13	14	101
9	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	3	47	12	12	7	8	5	91
10	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	38	14	12	11	8		83
11	Kostolányi Peter	Gym. Jura Hronca BA	3	32	12	12	7	13		76
12	Varga András	Gym. H. Selyeho Komárno	3	33	14	8	7	5	6	73
13	Vojtko Jakub	Gym. Jura Hronca BA	2	32	14	11	15			72
14	Fedáková Dominika	Gym. Stará Ľubovňa	3	37	13	8	2	8	2	70
15	Nagy Kristián	Gym. Jura Hronca BA	3	32	12	12	4	8		68
16	Korcsok Peter	Gym. Mládežnícka Šahy	3	36	12	11			7	66
17	Košinárová Alena	Gym. Grösslingová BA	3	20	14	11	12		7	64
18	Kočiský Tomáš	Gym. Grösslingová BA	3	45	7	11				63
19	Herman Peter	Gym. Jura Hronca BA	4	61						61
20	Saleh Adam	Gym. Jura Hronca BA	3	28	13	11	3			55
20	Tomcsányi György	Gym. H. Selyeho Komárno	4	24	15	8		8		55
22	Petrucha Michal	Gym. Metodova BA	2	20	11	12	8			51
23	Herencsár Albert	Gym. maď. Galanta	2	20	12	12				44
24	Nagy Anton	Gym. maďarské BA	4	42						42
25	Súkeník Ján	Gym. Lettricha Martin	3	20	10	8	2			40
26	Mészáros Roman	SPŠ Levice	4	19	12	8				39
27	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	20	10	7				37
28	Kudláč Matúš	Gym. Bilíkova BA	3	15	12	8				35
28	Štefanišin Peter	Gym. Svidník	4	12	15	8				35
28	Zámečník Dušan	Gym. Trebišovská Košice	4	35						35
31	Hagara Michal	Gym. Jura Hronca BA	1	13	13	7				33
32	Mikuláš Ondrej	Gym. Haličská Lučenec	4	32						32
33	Fecko Stanislav	Gym. Pankúchova Bratislava	4	12	11	8				31
33	Magyar Vladimír	SPŠE Nové Zámky	3	22	1	8				31
33	Piter Miroslav	Gym. Svidník	0	12	11	8				31
36	Hozza Michal	Gym. Jura Hronca BA	2	10	12	8				30
37	Sucha Martin	Gym. Jura Hronca BA	3	17		12				29
38	Čevorová Kristína	Škola pre mim. nadané deti BA	4	0	8	12	7			27
39	Boško Lukáš	Gymnázium Považská Bystrica	0	12	5	8				25
39	Hojčka Michal	Gym. Partizánske	2	7	10	8				25
41	Polák Lukáš	Gym. Jura Hronca BA	1	8	5	11				24
42	Liška Igor	Gym. Jura Hronca BA	2	23						23
43	Kvasnička Igor	Gym. Jura Hronca BA	3	11		11				22
44	Chudý Lukáš	Gym. Liptovský Hrádok	2	21						21
45	Fekiač Jozef	Gym. Grösslingová BA	2	12		8				20
45	Halabuk Milan	Gymnázium Levice	4	12		8				20
45	Hanes Filip	Gym. Tajovského B. Bystrica	4	20						20
45	Molčan Dávid	Gym. Slovenská Bardejov	4	6	7	7				20
49	Kompan Martin	Gym. Partizánske	4	18						18
49	Magdolen Matej	Gym. Golianova Nitra	1	0	10	8				18

	Meno a priezvisko	Škola	Trieda		4	5	6	7	8	Σ
51	Chrzan Martin	Gym. Ľ. Štúra Zvolen	2	14	3					17
51	Knotek Tomáš	SPŠ Nitra	4	17						17
51	Šulák Viktor	SPŠE Nové Zámky	2	10		7				17
54	Balogh Tomáš	SPŠE Nové Zámky	2	6	7	3				16
54	Simanová Lucia	Gym. Grösslingová BA	3	13		3				16
56	Bene Daniel	Gym. Haličská Lučenec	3	12						12
56	Dittrich Andrej	Gym. Jura Hronca BA	3	0		12				12
56	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	12						12
56	Kedrovič Ondrej	Gym. Jura Hronca BA	0	9		3				12
56	Končok Jakub	Gym. Haličská Lučenec	3	12						12
56	Kučin Alexander	Gym. Lipany	3	12						12
56	Matejovičová Lenka	Gym. Grösslingová BA	3	12						12
56	Tureková Katarína	Gym. Tajovského B. Bystrica	3	0		12				12
56	Vlček Tomáš	Gym. Šk. Bratov BA	4	12						12
65	Bachratý Martin	Gym. Veľká Okružná Žilina	2	11						11
65	Košdy Martin	Gym. Jura Hronca BA	2	0		11				11
67	Maršík Ladislav	Gym. Grösslingová BA	4	0	9					9
68	Kohút Matej	Gym. Jura Hronca BA	2	0		8				8
69	Duchoň Gabriel	Gym. maď. Želiezovce	4	6						6
69	Fico Tomáš	SPŠ Nitra	4	6						6
69	Peťura Oto	SPŠE Michalovce	1	6						6
72	Buchovecká Simona	Gym. Medzilaborce	4	4						4
72	Greň Peter	Gym. L. Stöckela Bardejov	1	4						4
74	Matula Peter	Gym. Školská Turč. Teplice	3	3						3
74	Végh Ladislav	Gym. Tornaľa	2	3						3

Výsledková listina po 2. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda		8	9	10	Σ
1	Danilák Michal	Gym. Hubeného BA	4	38	13	15	15	81
2	Okruhlica Adam	Gym. Jura Hronca BA	4	37	14	15	6	72
3	Rampášek Ladislav	Gym. Jura Hronca BA	4	18	12	8	15	53
4	Ondruška Peter	SPŠ Dubnica nad Váhom	3	16	10	3	9	38
5	Boža Vladimír	Gym. Tatarku Poprad	3	0	12	4	15	31
6	Korcsok Peter	Gym. Mládežnícka Šahy	3	17	7			24
7	Brezáni Samuel	Gym. Rajec	4	13	10			23
7	Hlavatý Peter	Gym. Jura Hronca BA	3	7	13	3		23
9	Hapák Samuel	Gym. Grösslingová BA	3	0	14			14
9	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	3	9	5			14
11	Fedáková Dominika	Gym. Stará Ľubovňa	3	9	2			11
12	Varga András	Gym. H. Selyeho Komárno	3	4	6			10
13	Zámečník Dušan	Gym. Trebišovská Košice	4	9				9
14	Herman Peter	Gym. Jura Hronca BA	4	8				8
14	Liška Igor	Gym. Jura Hronca BA	2	6		2		8
16	Kompan Martin	Gym. Partizánske	4	7				7
16	Košinárová Alena	Gym. Grösslingová BA	3	0	7			7
16	Mikuláš Ondrej	Gym. Haličská Lučenec	4	7				7
16	Nagy Anton	Gym. maďarské BA	4	7				7
20	Kočiský Tomáš	Gym. Grösslingová BA	3	4				4
21	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	3				3
21	Kedrovič Ondrej	Gym. Jura Hronca BA	0	3				3
23	Magyar Vladimír	SPŠE Nové Zámky	3	0		2		2