Korešpondenčný seminár z programovania
XXV. ročník, 2007/08
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 letnej časti

Milé naše riešiteľky, milí naši riešitelia,

práve sa vám dostali do rúk najnovšie a posledné vzoráky tohto ročníka KSP. Keďže ide leto, tak sme sa rozhodli spraviť nasledovnú súťaž o čokoládu. Pošlite nám fotku vás a našich vzorákov z čo najzaujímavejšieho miesta, ktoré v lete navštívite. Výťaza vyberie odborná porota skladajúca sa z KSPákov. Fotky zasielajte na adresu `mic@sutaz.ksp.sk`.

Vedeli ste, ze 47% štatistík je vymyslených?

KSPáci

1. Zľav, lebo to jeho topánka kúpi!

opravoval Maty
(max. 15 bodov)

Tak tento príklad je prvý a teda väčšina z vás ho mala dobre. Pár bodov ste mohli postrácať za chýbajúce odhady alebo zbytočne veľkú lineárnu pamäť.

A teraz ku vzoráku. Keďže zľavy, ktoré získavame sa nám sčítavajú a môžeme isť do záporu (obchod nám vráti peniaze, keď máme zľavu väčšiu ako 100%), môžeme o každej zľave uvažovať samostatne bez ohľadu na ostatné. Rozmyslite si, že tento postup nebude fungovať, keby sme mohli použiť zľavy maximálne do 100%, alebo keby sa namiesto sčítania aplikovali postupne. Teraz sa stačí zaoberať iba jednými konkrétnymi teniskami, ktorých cena je a_i a zľava ktorú za ne dostaneme je b_i. Tenisky sa nám oplatí kúpiť jedine vtedy, ak dostaneme väčšiu zľavu z vysnívaných tenisiek, ktoré stoja C, ako to čo sme museli zaplatiť. Teda kúpime ich, iba ak $a_i < \frac{C \cdot b_i}{100}$. V programe namiesto tejto nerovnosti používame ekvivalentnú $a_i \cdot 100 < C \cdot b_i$. Násobenie je o kúsok rýchlejšie ako delenie a navyše pri delení by sme museli použiť dátový typ pre desatinné čísla. Pri násobení môžeme zostať pri celočíselnom dátovom type.

Ak by sme používali celé čísla a delili, tak by sme v niektorých prípadoch mohli dôjsť k nesprávnemu výsledku. Nech $a_i = 100, C = 1001, b_i = 10$. Ak počítame v celých číslach, tak sa dostaneme k nerovnici $100 < \frac{10010}{100}$, čo sa v celých číslach vyhodnotí ako $100 < 100$ a tenisky nekúpime. Ak to vypočítame v reálnych číslach, tak dostaneme nerovnicu $100 < 100.1$ a tenisky kúpime. Preto je lepšie prehodiť stovku na druhú stranu nerovnice a nemusíme používať reálne čísla.

Program je už teraz zjavný. Postupne načítavame vstup po teniskách a pre každé zistíme, či sa nám ich oplatí kúpiť a vypíšeme o tom správu. Časová zložitosť bude $O(N)$. A keďže nám netreba pamätať si ceny a zľavy všetkých tenisiek, ale iba tých práve spracovávaných, pamäťová zložitosť je $O(1)$.

Listing programu:

```
var a,b,c,n,i : integer;
begin
  Readln(c,n);
```

```
  for i:=1 to n do begin
    readln(a,b);
    if a*100<c*b then writeln('Kup tenisky cislo ',i);
  end;
end.
```

opravoval Peťo
(max. 15 bodov)

2. Zberač Ivan

Prenesme si našu úlohu do teórie grafov. Graf je tvorený množinou vrcholov V, v našom prípade stanice, a množinou hrán E, v našom prípade trate. Kým je graf súvislý, tak je všetko v poriadku[1]. Okrem toho Ivan a jeho protivník odoberajú hrany (trate) tak, aby prehrali čo najneskôr.

V teórii grafov poznáme grafy, ktoré nazývame *stromy*. O nich vieme dokázať nasledovné tvrdenia:

1. Graf je strom vtedy a len vtedy, keď sú ľubovoľné dva vrcholy spojené práve jednou cestou.
2. Graf je strom vtedy a len vtedy, keď je minimálne súvislý, to znamená, že odobraním ľubovoľnej hrany prestane byť súvislý.
3. Graf je strom vtedy a len vtedy, keď je maximálne acyklický, to znamená, že nemá žiaden cyklus a pridaním ľubovoľnej hrany vznikne cyklus.
4. Súvislý graf s n vrcholmi je strom vtedy a len vtedy ak má $n-1$ hrán.

Navyše, ak graf nie je strom, tak určite existuje hrana, ktorej odobratím zostane graf stále suvislý. Vyplýva to napríklad z tretieho tvrdenia: každý súvislý graf, ktorý nie je strom, obsahuje cyklus. Odobratím hrany z cyklu nič nepokazíme.

Z týchto tvrdení nám vyplýva, že obaja hráči budú odoberať hrany z grafu dovtedy kým z neho nevznikne strom. Ten kto odoberie hranu zo stromu poruší jeho súvislosť (tvrdenie 2) a teda prehrá. No a keďže strom má vždy $n-1$ vrcholov, tak je jedno v akom poradí budú hrany odoberať a dokonca je jedno aké hrany v grafe sú na začiatku.

Dostávame sa k tomu, že ak m je počet hrán a n je počet vrcholov, tak v grafe existuje $m-(n-1)$ hrán, ktoré keď odoberieme, tak dostaneme strom a zároveň neexistuje $m-(n-1)-1$ takých hrán, že graf po och odobratí ostane súvislý. To znamená, že prehrá ten, kto odoberie $m-(n-1)-1$ hranu. Preto keď Ivan začína a obaja sa stále striedajú, tak Ivan vyhrá práve vtedy keď je $m-(n-1)$ nepárne, pretože zoberie prvú a rovnako aj poslednú z $m-(n-1)$ hrán a potom je na rade protivník, ktorý tým pádom prehrá. Jasne vidieť, že stačí načítať n a m a z týchto dvoch čísel vieme výsledok.

Tento krát sme pripravili naozaj ľahký príklad, čo sa ukázalo vysokým percentom správnych riešení. Preto som kládol trochu väčší dôraz na menej kvalitný popis a rovnako na zbytočné načítavanie všetkých hrán.

Listing programu:

```
var n,m:integer;
begin
   readln(n,m);
   if m - (n - 1) mod 2 = 1 then
      writeln("Ivan vyhra.")
   else
      writeln("Ivan prehra.");
end;
```

[1] Ľudia z Interpolu papajú šišky

3. Zbúrajme si mrakodrapy!

opravoval Pershing
(max. 15 bodov)

Páčila sa vám hra? Tak ju poďme vyriešiť a to rovno nie pre 5 ale pre n kôpok.

Najskôr si budeme musieť definovať, čo je to vyhrávajúca a prehrávajúca pozícia. Vyhrávajúca pozícia je taká, že ak som v nej, tak ak budem hrať optimálnou stratédiou, tak určite vyhrám. Prehrávajúca pozícia je taká, že ak súper bude hrať optimálnou stratégiou, tak prehrám nech by som hral akokoľvek. Inými slovami.

Základom nášho riešenia bude myšlienka, ktorá funguje na nejednu takúto hru a to konkrétne symetria. Symetria v zmysle, že "Ak som vo vyhrávajúcej pozícii, súper spraví ťah, tak viem spraviť podobný ťah aj ja a opäť vyhrávam".

Našou symetriou bude počet najvyšších mrakodrapov, a presnejšie parita tohoto počtu. Kedy sme vo vyhrávajúcej pozícii? No, keď je počet najvyšších mrakodrapov po našom ťahu párny (alebo sme všetko práve zbúrali). Spravme preto jemnú modifikáciu - ak máme nepárny počet všetkých mrakodrapov, pridajme jeden fiktívny mrakodrap výšky nula - odstránime tým nutnosť kontrolovať, či sme nezbúrali posledný mrakodrap - overenie cez paritu najvyšších mrakodrapov (všetky sú nulovej výšky a teda najvyššie) bude fungovať.

Teraz by sme si mali povedať, prečo sú vyhrávajúce pozície práve tie s párnym počtom najvyšších mrakodrapov. Predstavme si, že po našom ťahu ostane počet najvyšších mrakodrapov párny. Čo spraví súper? Zbúra vrch nejakého mrakodrapu. Teraz sme na rade my. Ak nám ostali aspoň 3 [2] najvyššie mrakodrapy jednoducho jeden zbúrame celý a opäť sa dostávame do vyhrávajúcej pozície. Jediný problém je, ak máme jediný najvyšši mrakodrap. V tejto pozícii si spočítame počet druhých najvyšších mrakodrapov - ak je ich nepárny počet, zrovnáme to na ich výšku. Ak je ich párny počet, zrovnáme to na ľubovoľnú menšiu. Tu je dobré sa zamyslieť nad tým, že menšia výška existuje. Totižto ak druhá maximálna výška je 0, tak tých mrakodrapov musí byť nepárne (ako sme si na začiatku povedali) ale to je v spore s predpokladom. Preto aj z možnosti že je práve jeden najvyšší mrakodrap sa vieme dostať do párneho počtu najvyšších mrakodrapov a teda vyhrávajúcej pozície.

Takto môžeme hrať ďalej, nakoniec aj tak vyhráme. Fajn, toto by bola stratégia, ak už vyhrávame. Naopak, ak naším ťahom nevieme zrovnať počet najvyšších mrakodrapov na párny počet (Rozmyslite si, že je to jedine ak je počet najvyšších mrakodrapov pred naším ťahom párny), v ďalšom ťahu súper aplikuje overenú taktiku a čuduj sa svete, **prehráme**.

Listing programu:

```
const  N=5;

var max_vyska,max_vyska2:integer;
    pocet_max,pocet_max2:integer;
    cislo_max:integer;
    q,vyska:integer;

begin
  max_vyska:=0;
  max_vyska2:=0;
  pocet_max2:=N mod 2;
  cislo_max:=0;

  for q:=1 to N do
   begin
```

[2] všimnime si, že 2 nemôžu byť kvôli parite

```
    read(vyska);

    if (vyska>=max_vyska) then
    begin
      if (vyska=max_vyska) then
        inc(pocet_max)
      else begin
        pocet_max2:=pocet_max;
        max_vyska2:=max_vyska;
        max_vyska:=vyska;
        pocet_max:=1;
        cislo_max:=q;
      end;
    end
    else if (vyska>=max_vyska2) then
    begin
      if (vyska=max_vyska2) then
        inc(pocet_max2)
      else
      begin
        max_vyska2:=vyska;
        pocet_max2:=1;
      end;
    end;
  end;

  if (pocet_max mod 2=0) then
    writeln('Prehram')
  else if (pocet_max2 mod 2=0) then
    writeln(cislo_max,' ', max_vyska) { buram cely }
  else
    writeln(cislo_max,' ', max_vyska-max_vyska2); {buram po nove maximum}
end.
```

4. Ostrov veľkého slizniaka

opravoval Mišo
(max. 15 bodov)

Na kvadratické riešenie vyskúšaním všetkých dvojíc by prišiel v podstate každý. Zaujímavé, ale náročnejšie bolo celkom v čase $O(N \cdot \log N)$, kde pre kždú sedačku sa nevyskúšajú všetky sedačky, ale trochu sa využije, že sú usporiadané, a nájsť sedačku, ktorá je najvzdialenejšia od konkrétnej sedačky X, sa dá upraveným binárnym vyhľadávaním. Začne sa intervalom pozostávajúcim zo všetkých sedačiek okrem X (na krajoch sú susedia X) a v každom kroku sa pozrie na prostredné dve sedačky. Ak je ich vzdialenosť od X rovnaká, výsledok je jedna z tých dvoch. Nech je vzdialenejšia tá pravá(keby ľavá, tak symetricky robíme to isté). Rekurzívne sa zavoláme na interval medzi tou pravou sedačkou a pôvodnym pravým okrajom, lebo vľavo sú už iba sedačky bližšie k X, teda najvzdialenejšia je vpravo.

Ešte zaujímavejšie aj ľahšie je riešenie lineárne, kde sa dá úplne využiť uporiadanosť sedačiek. Klasické riešenie kopy úloh forcyklom vo forcykle sa dá niekedy zlepšiť. Budú to tiež v podstate dva cykly akoby „v sebe“, ale nie tak celkom. Máme teda dva pointre, na začiatku vedľa seba. Nazvime si ich prvý, druhý, alebo i, j. Zvyšujeme j, až kým sa nesplní nejaká podmienka. Potom zvyšujeme i. Skončíme, keď prejdeme celé kolečko. To je len taký všobecný návod, ako sa dajú riešiť niektoré úlohy. Je zjavné, že je to lineárne, lebo i prejde maximálne jedno kolečko, j prinajhoršom dve. Ako to aplikovať na slizniaky? Takže po každom zvýšení zistíme vzdialenosť sedačiek i, j. Pamätáme si vždy poslednú vzdialenosť,

aby sme ju mohli porovnať s ďalšou. Ak sá vzdialenosť zvyšuje, ideme ďalej, o chvíľu možno nájdeme hľadanú dvojicu. Ak sa prvýkrát zníži, vrátime sa na predchádzajúcu (ďalej to už nemá zmysel, už sa budú vzdialenosti len zmenšovať). Teraz je j najvzdialenejšia sedačka od i. Zvýšime i. Druhým pointrom j teraz hľadáme len ďalej od momentálnej pozície j, lebo tie pozície, ktoré sme už prešli, sú určite bližšie k i, lebo sme sa j-čkom vzďaľovali. Tento postup opakujeme pre každé i.

Pozrime sa na časovú zložitosť. j-čko sa posunie maximálne $4n$ krát (nespraví viac ako 2 kolečká a pri každom posune i sa vráti o 1 pozíciu späť) Počítanie a porovnanie vzdialeností je konštantné, preto je celková časová zložitosť lineárna. Pamäť tiež treba lineárnu, pamätáme si pole, dva pointre a zopár konštánt. Bodovanie: Kvadratické riešenie 8 bodov, $O(N \cdot \log N)$ 11 bodov, lineárne 15 bodov. Za chýbajúci/zlý odhad bolo po -1 bode. Za chýbajúci popis/zdroják polovica bodov dole. Cenu Veľkého Slizniaka za najelegantnejšie riešenie získava Tomáš Kuzma!

Listing programu:

```
function vzdialenost(x1:double;y1:double;x2:double;y2:double):double;//vrati stvorec
vzdialenosti, lahsie sa rata, a na porovnavania staci
    begin
        vzdialenost:=(x2-x1)*(x2-x1)+(y2-y1)*(y2-y1);
    end;
var x,y:array[1..100] of double;
        N,i,j,maxi,maxj:longint;
        max,last:double;
begin
    last:=-1;max:=-1;//kazda vzdialenost je kladna, takze takto sa nestane, ze by sa
do max nic nepriradilo
    readln(N);
    for i:=1 to N do readln(x[i],y[1]);
    j:=2;
    for i:=1 to N do
        begin
            while vzdialenost(x[i],y[i],x[j],y[j])>last do
                begin
                    last:=vzdialenost(x[i],y[i],x[j],y[j]);
                    j:=j mod N +1;
                end;
            j:=(j+N-2)mod N+1;
            if last>max then
                begin
                    max:=last;
                    maxi:=i;
                    maxj:=j;
                end;
        end;
    writeln(maxi,' ',maxj);
end.
```

5. Okšaram a náhrdelník

Luki
(max. 15 bodov)

Na úvod pár slov k riešeniam, ktoré prišli. Môžno ich rozdeliť do 2 skupín, tie ktoré fungovali a potom tie zvyšné. Čudujme sa, ale tých funkčných bola menšina. Na druhej strane si uvedomujem, že nie každý pozná *Eulerovský cyklus* a bez nutnej a postačujúcej podmienky

pre Eulerovský cyklus bolo riešenie tohto príkladu komplikované. Optimálne riešenie ma časovú aj pamäťovú zložitosť $O(N + M)$.

Pre tých, ktorí o teórii grafov veľa nevedia: *Graf* je množina *vrcholov* (v našom prípade spojenia) a *hrán* (u nás kvietkov), kde hrana spája dva vrcholy. *Cesta* medzi vrcholmi u, v existuje, ak sa po hranách vieme dostať z jedného vrchola do druhého. *Komponent* je množina vrcholov, pre ktoré existuje v grafe cesta. *Stupeň vrchola* je počet hrán vychádzajúcich z vrchola.

Nutná a postačujúca podmienka pre Eulerovský cyklus: Graf má Eulerovský cyklus práve vtedy, keď je súvislý a stupeň každého vrchola je párny.

Po troške rozmýšľania zistíme, že v každom komponente máme párny počet vrcholov nepárneho stupňa (lebo hrana spája vždy dva vrcholy a teda súčet stupňov vrcholov v každom komponente je párny).

Uvedomme si, že z každého vrcholu s nepárnym stupňom musí vychádzať jedna hrana, lebo chceme, aby bol každý vrchol párneho stupňa. Tiež potrebujeme aby sme medzi sebou prepojili komponenty. Keď máme K komponentov, pridáme K hrán, aby sme z nich urobili jeden veľký komponent (spojíme ich do kruhu).

Ešte si musíme uvodomiť, že my potrebujeme len kružnicu prechádzajúcu všetkými hranami, ktorá však nemusí prechádzať vrcholmi (spojeniami), ktoré sa nevyskytovali v žiadnom kvietku na vstupe. Teda vynechame vrcholy z ktorých nejde žiadna hrana a nebudeme ich počítať medzi komponenty.

Inak povedané do každého komponentu pridáme 2 polhrany (hrany, ktorých zatiaľ poznáme len jeden vrchol). Pokiaľ má komponent vrcholy nepárneho stupňa, tak zoberieme dva z nich a k obom pridáme po jednej polhrane a tým sa z nich stanú vrchol párneho stupňa. Ak komponent nemá vrchol s nepárnym stupňom, musíme k jednemu vrcholu pridať 2 polhrany, aby bol opäť párneho stupňa. A teraz, keď z každého komponentu trčia práve 2 polhrany, pospájame tieto komponenty polhranami do kruhu.

Ale prečo nám nestačí použiť len $K - 1$ hrán? Lebo by sme vytvorili len jednu cestu medzi všetkými komponentami a keďže chceme aby existovala Eulerovská kružnica, tak potrebujeme prepojiť ešte začiatok s koncom, aby to bola skutočne kružnica.

Špecifický prípad je tiež keď všetky hrany sú v jednom komponente, potom len jednoducho pospájame dvojce vrcholov nepárneho stupňa a máme splnenú nutnú a postačujúcu podmienku pre existenciu Eulerovskej kružnice.

Prečo je toto *minimálny* počet hrán? Vieme, že ku každému vrcholu s nepárnym stupňom musíme pridať jednu polhranu, aby bol párneho stupňa. Týmto dokážeme pospájať medzi sebou všetky komponenty s vrcholmi nepárneho stupňa, ale ostali nám osamotené komponenty bez takýchto vrcholov. Pre každý z týchto komponentov však musíme pridať dve polhrany k lubovoľnému vrcholu (aby ostal párneho stupňa a mohol pripojiť komponent k ostatným). Nech p je počet vrcholov nepárneho stupňa a k je počet komponentov bez vrcholu s nepárnym stupňom. Pokiaľ je počet všetkých komponentov 1 tak potrebujeme minimálne p polhrán, teda $p/2$ hrán. Inak minimálne $(k + p)/2$ hrán. Ukázali sme si, ako týmto počtom hrán upraviť graf, na taký, ktorý obsahuje Eulerovskú kržnicu, teda je tento počet minimálny.

Takže prejdime k vzorovemu riešeniu: Načítame graf a zistíme stupne vrchola. Spočítame všetky vrcholy nepárneho stupňa. Zistíme koľko existuje komponentov a koľko z nich obsahuje vrcholy nepárneho stupňa, toto môžeme urobiť napr. prehľadávaním do hĺbky tak, že spustíme prehľadávanie z 1. vrchola a všetky navštívené vrcholy označíme jednotkou (číslo komponentu do ktorého patria). Potom zoberieme ešte nenavštívený vrchol a urobíme to isté len označíme vrcholy dvojkou,..., až pokiaľ neoznačíme všetky vrcholy. Nakoniec ak počet komponentov je viac ako jedna, tak výsledok je $(k + p)/2$, kde k je počet komponentov bez vrcholu s nepárnym stupňom a p počet vrcholov nepárneho stupňa.

Listing programu:

```
uses crt;
var hrany:array[1..100,1..2]of integer;
    vrcholy: array[1..101] of integer;
    stupen:  array[1..100] of integer;
    vrcholy_index: array[1..101] of integer;
    hrany2: array[1..10000] of integer;
    komp: array[1..100] of integer;

    i,j,N,M, pom : integer;
    pocet_neparnych_vrcholov: integer;
    pocet_komp: integer;

procedure prehladaj(x, cislo_komp:integer);
var j:integer;
begin
 komp[x] := cislo_komp;
 if (stupen[x] mod 2) = 1 then inc(vrcholy_index[cislo_komp]);
 for j:= vrcholy[x] to vrcholy[x+1]-1 do
 begin
  if komp[hrany2[j]] = 0 then prehladaj(hrany2[j],cislo_komp);
 end;
end;


begin
 readln(N,M);
 for i:=1 to M do
 begin
  vrcholy[i] := 0;
  stupen[i] := 0;
  vrcholy_index[i] := 0;
 end;
 for i:=1 to N do
 begin
  readln(hrany[i,1], hrany[i,2]);
  inc(stupen[hrany[i,1]]);
  inc(stupen[hrany[i,2]]);
 end;

 pocet_neparnych_vrcholov := 0;
 j := 0;
 for i:=1 to M do
 begin
  if (stupen[i] mod 2) = 1 then inc(pocet_neparnych_vrcholov);
  vrcholy[i] := j+1;
  j := j + stupen[i];
 end;
 vrcholy[M+1] := j+1;

 for i:=1 to N do
 begin
  hrany2[vrcholy[hrany[i,1]]+vrcholy_index[hrany[i,1]]] := hrany[i,2];
  hrany2[vrcholy[hrany[i,2]]+vrcholy_index[hrany[i,2]]] := hrany[i,1];
  inc(vrcholy_index[hrany[i,1]]);
  inc(vrcholy_index[hrany[i,2]]);
 end;

 for i:=1 to M do
```

```
 begin
  vrcholy_index[i] := 0;
  komp[i] := 0;
 end;
 pocet_komp := 0;
 for i:=1 to N do
 begin
  if (komp[i] = 0) and (vrcholy[i] < vrcholy[i+1]) then
  begin
   inc(pocet_komp);
   prehladaj(i, pocet_komp);
  end;
 end;

 if pocet_komp = 1 then writeln(pocet_neparnych_vrcholov / 2)
 else
 begin
   pom := 0;
   for i:=1 to pocet_komp do
    if (komp[i] > 0) and (vrcholy_index[i] = 0) then begin  inc(pom);
writeln('vrchol ',i);end;
   writeln(pom+(pocet_neparnych_vrcholov / 2));
 end;
end.
```

6. O Krajine Strašných Pokladov

Opravoval MMx
(max. 15 bodov)

Tento príklad zrejme vyzeral na prvý pohľad prekvapivo ľahko, pretože väčšina z vás sa nechala prekvapiť a zo šiestich riešení prišlo len jedno správne. Takže ako na to?

Krajina na vstupe je samozrejme strom. Keď prvýkrát prídeme do nejakého vrcholu, musíme najprv prejsť cez všetkých jeho potomkov predtým, ako sa vrátime naspäť (ináč by sme sa neskôr nemali ako k vynechaným potomkom dostať). To znamená, že sme nútení prechádzať graf prehľadávaním do hĺbky, na nás teda ostáva voľba poradia podstromov.

Pre každý podstrom i si spočítame dve hodnoty. Potrebnú investíciu T_i, teda koľko najmenej peňazí treba pred vstupom do neho aby sme mali v celom podstrome z čoho platiť a zisk Z_i, teda o koľko sa naša hotovosť zmení prechodom cez tento podstrom. Zisk samozrejme môže byť aj záporný. Obidve hodnoty budú zahŕňať udalosti od platenie mýta pri vstupe do potomka i, vybratie pokladu u neho, prechod jeho potomkov až po platenie mýta pri návrate. Tieto hodnoty sa dajú ľahko spočítať pre ľubovoľný vrchol simuláciou prechodu, ak ich poznáme pre každý jeho podstrom a máme dané poradie podstromov. To znamená, že ich treba počítať od listov smerom ku koreňu. Teraz si ukážeme ako pomocou nich určiť poradie.

Intuitívne dáva zmysel prejsť najprv všetky ziskové podstromy ($Z_i > 0$), pretože z ich zisku sa nám môže podariť zaplatiť niečo z toho, čo stratíme v stratových podstromoch a o to menej si bude treba zobrať. Presnejšie po príchode do nejakého vrcholu potrebujeme T_1 peňazí na vstup do prvého podstromu, $T_2 - Z_1$ pre druhý, $T_3 - Z_2 - Z_1$ pre tretí, ... Ak by sme dopredu dali nejakú stratovú vetvu j, potom by sa všetky tieto hodnoty zvýšili o $-Z_j$, teda do tohoto vrcholu by sme si museli doniesť viac peňazí. Teraz to dokážeme formálne. Uvažujme najprv vrchol s dvomi synmi. Prvý nech je nestratový ($Z_1 \geq 0$), druhý stratový ($Z_2 < 0$). Po príchode do tohoto vrcholu a vybratí pokladu budeme mať nejaké peniaze, označme ich S. Teraz máme dve možnosti, buď navštívime najprv podstrom 1 alebo najprv podstrom 2. V prvom prípade budeme mať postupne toľkoto peňazí: $S - T_1$

(niekde v prvom podstrome), $S + Z_1$ (po návrate z neho), $S + Z_1 - T_2$ (niekde v druhom podstrome), $S+Z_1+Z_2$. Nás zaujíma, ktorá z týchto hodnôt bude najmenšia, pretože tá určí, koľko peňazí potrebujeme pred vstupom do tohoto vrcholu. Zjavne platí $S - T_1 > S + Z_1$ a $S+Z_1-T_2 \leq S+Z_1+Z_2$ (pretože nemôžeme vo vetve stratiť viac ako je potrebná investícia). Preto sa stačí zaoberať hodnotami $S - T_1$ a $S + Z_1 - T_2$. Analogicky ak by sme išli najprv do druhého podstromu, postupnosť naších hotovostí by vyzerala $S - T_2$, $S + Z_2 - T_1$. My si chceme vybrať tú, ktorá klesne čo najmenej (teda jej minimum bude vyššie). Z predpokladov o ziskoch podstromov priamo vyplýva $S-T_1 > S+Z_2-T_1$ a $S+Z_1-T_2 \geq S-T_2$, a teda nech je minimum prvej postupnosti ktorékoľvek, nemôže byť menšie ako minimum druhej. Z toho vyplýva, že prvá postupnosť je pre nás výhodnejšia a teda pri vrcholoch s dvomi potomkami treba uprednostniť nestratový pred stratovým. Toto zistenie sa dá priamo zovšeobecniť pre ľubovoľný počet potomkov. Ak uvažujeme v akom poradí navštíviť i- a $(i+1)$-teho potomka, potom zisk z prvých $i-1$ potomkov spolu s pokladom a prinesenými peniazmi si môžeme označiť S a predchádzajúca úvaha sa dá priamo použiť (všetky tvrdenia ostanú platné aj napriek tomu, že za $(i+1)$-tým potomkom môžu nasledovať ďalší, lebo investície T_i a T_{i+1} to nijak neovplyvní).

Vo výslednom usporiadaní teda budú najprv ziskové vrcholy. V akom poradí? Predstavme si trochu inú situáciu, vieme aká investícia stačí na prejdenie všetkých ziskových vetiev a my máme nájsť poradie v ktorom to určite pôjde. Ak sa nám niektorú vetvu podarí prejsť, určite tým nič nepokazíme, lebo po jej prejdení budeme mať viac peňazí a množina vrcholov, do ktorých môžeme ísť, sa nemôže zmenšiť (a do vetiev s príliš veľkou investíciou nás nepustí aktuálne množstvo peňazí). Presnejšie máme S peňazí a môžeme vojsť do tých vetiev, pre ktoré $S \geq T_i$. Určite sa teda dá vojsť do vrcholu s najmenším T_i (ináč máme spor s tým, že S peňazí stačí). Toto pôjde opakovať až kým nevyčerpáme všetky ziskové vrcholy. Teraz formálne. Rovnako ako v prvom prípade treba zistiť ktorá z postupností $S-T_1$, $S+Z_1-T_2$ a $S-T_2$, $S+Z_2-T_1$ má vyššie minimum. V tomto prípade sú predpoklady, že obidve zisky sú nezáporné a $T_1 < T_2$. Z nich vyplýva, že platia nasledujúce nerovnosti: $S-T_2 > S+Z_2-T_1$ (lebo $T_1 < T_2$ a Z_2 je kladné), $S-T_1 > S-T_2$ a $S+Z_1-T_2 > S-T_2$. Teda ľubovoľné číslo prvej postupnosti je väčšie ako ľubovoľné číslo druhej, čiže aj minimum. Preto je lepšie ísť najprv do podstromov s menším T (zovšeobecnenie funguje rovnako ako v prvom príklade).

Poradie stratových vetiev sa ukázalo byť menej zjavné. Z prvej postupnosti je $S+Z_1-T_2$ určite menšie ako $S - T_1$, pretože strata žiadnej vetvy nemôže byť väčšia ako investícia (rovnako pre druhú postupnosť). Z toho vyplýva, že rozhodnúť sa treba podľa toho, ktoré z čísiel $S+Z_1-T_2$ a $S+Z_2-T_1$ je väčšie. Drobnou úpravou dostaneme, že skôr musia ísť vetvy, ktoré majú väčšie $Z_1 + T_1$. Intuícia za tým je nasledovná: Vetvy s väčším T treba vybaviť skôr, aby sme čo najviac z príslušných T_i zaplatili z doteraz nazbieraných ziskov. Takisto vetvy s väčšími stratami si treba nechať na neskôr, pretože ak o peniaze prídeme na začiatku, nebude z čoho vyplácať investície do vetiev, z ktorých sa vráti väčšia časť. Podstrom je pre nás vlastne o toľko lepší, o koľko viac nám vráti z počiatočnej investície.

Časová zložitosť je $O(n \log n)$ z nasledujúceho dôvodu. Prechod grafom beží v čase $O(n)$, rozhodujúci bude teda čas triedenia polí podstromov. Každý vrchol sa nachádza v zozname následovníkov nejakého iného vrcholu práve raz (okrem koreňa, ktorý nie je v žiadnom zozname). To znamená, že triedime n polí, označme si dĺžku i-teho s_i. Dokážeme že platí $s_i \log s_i + s_j \log s_j < (s_i + s_j) \log(s_i + s_j)$. Bez ujmy na všeobecnosti nech $s_i \geq s_j$. Zo základných vlastností logaritmu platí

$$s_i \log s_i + s_j \log s_j = \log s_i^{s_i} + \log s_j^{s_j} = \log(s_i^{s_i} s_j^{s_j}) < \log s_i^{s_i+s_j}$$

$$(s_i + s_j) \log(s_i + s_j) = \log(s_i + s_j)^{s_i+s_j} > logs_i^{s_i+s_j}$$

Z toho vyplýva, že čas potrebný na utriedenie dvoch postupností je menší ako čas na utriedenie postupnosti, ktorá vznikne ich spojením. Indukciou sa dá dokázať, že to platí pre

ľubovoľný počet častí. Teda triediť zoznamy nasledovníkov po častiach bude rýchlejšie ako utriediť pole dĺžky n.

Pre každý vrchol aj hranu si pamätáme iba konštantné množstvo informácii a maximálna hĺbka rekurzie je n, teda pamäťová zložitosť je $O(n)$.

Listing programu:

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

int n;
vector<int> poklad;
vector<vector<pair<int,int> > > mesto;
vector<int> zisk, invest;

bool cmp(const int& a, const int& b) {
  if (zisk[a]>=0) {
    if (zisk[b]<0) return true;
    return invest[a] < invest[b];
  }
  if (zisk[b]>=0) return false;
  return zisk[a]+invest[a] > zisk[b]+invest[b];
}

void DFS(int vrchol, int predok, int myto) {
  // vyrob zoznam podstromov, pre kazdy rekurzivne vypocitaj zisk a invest
  vector<int> podstrom;
  for (int i=0; i<mesto[vrchol].size(); i++) {
    int v = mesto[vrchol][i].first, m = mesto[vrchol][i].second;
    if (v != predok) {
      DFS(v, vrchol, m);
      podstrom.push_back(v);
    }
  }

  sort(podstrom.begin(), podstrom.end(), cmp);

  // vstup do vrchola a zobratie pokladu
  invest[vrchol] = myto; zisk[vrchol] = poklad[vrchol]-myto;
  // prechod podstromov
  for (int i=0; i<podstrom.size(); i++) {
// ak tento podstrom potrebuje vacsiu investiciu ako hociktory pred nim, zapamataj si ju
    invest[vrchol] = max(invest[vrchol], invest[podstrom[i]]-zisk[vrchol]);
    zisk[vrchol] += zisk[podstrom[i]];
  }
  // myto pri navrate
  invest[vrchol] = max(invest[vrchol], myto-zisk[vrchol]);
  zisk[vrchol] -= myto;
}

int main() {
  scanf("%d", &n);
  poklad.resize(n); mesto.resize(n); zisk.resize(n); invest.resize(n);
  for (int i=0; i<n; i++) scanf("%d", &poklad[i]);
  for (int i=0; i<n-1; i++) {
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
```

```
    a--; b--;
    mesto[a].push_back(make_pair(b,c));
    mesto[b].push_back(make_pair(a,c));
  }

  DFS(0,0,0);

  printf("%d\n", invest[0]);

  return 0;
}
```

7. Obrovská spotreba veteránov

opravoval Zemčo
(max. 15 bodov)

Počet riešení tejto úlohy sa dal zrátať na prstoch dvoch štandardných päťprstých rúk, a aj to bola väčšina prstov druhej ruky nevyužitá. Teda, dala sa použiť na porátanie riešení osmičky alebo desiny. Veru, tento počet nepotešil.

Úplne na úvod treba poznamenať jednu vec. V zadaní bolo napísané, že nás zaujíma najkratšia cesta, avšak vaše programy hľadali sled. Pripomeniem, že zatiaľ čo v ceste sa nemôže zopakovať vrchol, v slede sa môže zopakovať aj vrchol aj hrana. Takže ak by sme vás brali za slovíčka, máte všetci nula bodov, pretože vaše riešenia sú nekorektné. Zachraňuje vás ale to, že zadanie bolo napísané neporiadne a táto úloha bola od začiatku myslená na hľadanie sledov a zachraňuje vás aj to, že pri tejto úlohe je naozaj troška prirodzenejšie uvažovať o sledoch, keďže ľahko vyrobíte graf, kde najkratšia cesta do cieľa neexistuje, lebo neexistuje žiadna, ale existuje sled, ktorý vás do cieľa úspešne dovedie – stačí dať veľa vrcholov spojených do hada a niekde v strede odbočku k pumpe. Ale môžete sa zamyslieť, ako by sa zmenilo riešenie, kebyže požadujeme najkratšiu cestu. Toľko úvodné hranie sa so slovíčkami, v ďalšom sa teda vyberieme za hľadaním najkratšieho sledu.

Vzorák má pomerne veľkú časovú obtiažnosť. Za rýchlejšie, ale nesprávne riešenie som dal na svoje pomery veľa bodov – až 4. To preto, lebo v ňom mohlo byť celkom dosť správnych myšlienok. Pre jednoduchosť zatiaľ uvažujme, že je našou úlohou nájsť len dĺžku najkratšieho sledu a nezaujíma nás postupnosť vrcholov. Na úvod dáme slovo pánovi Dijkstrovi.

Dijkstrov algoritmus

Algoritmus od Dijsktru[3] rieši úlohu hľadania najkratšej cesty z jedného vrcholu v do všetkých ostatných vrcholov grafu, ktorý ma nezáporne ohodnotené hrany. Jeho výstup je pole A, kde na $A[i]$ je dĺžka najkratšej cesty medzi vrcholom v a i.

Vrcholy sú rozdelené do dvoch skupín H (hotové) a O (otvorené). H sú vrcholy, o ktorých už vieme najkratšiu vzdialenosť z vrcholu v (označíme $d[i]$ pre vrchol i), v O sú tie ostatné, ku ktorým možno vieme nejakú cestu, ale ešte nevieme, že je najkratšia zo všetkých.

Na začiatku je vrchol v v množine H, pretože už na začiatku vieme, že najkratšia cesta z v do v je dlhá 0. Zároveň všetky ostatné vrcholy sú v O, pretože o nich ešte nevieme nič. Ku každému vrcholu z O si pamätáme aj dĺžku najkratšej známej cesty z v, čo je na začiatku dĺžka prípadnej hrany z v alebo ∞, ak táto hrana neexistuje[4].

Algoritmus postupuje nasledovne: v každom kroku vyberie vrchol u z O, ktorý má najkratšiu vzdialenosť od v, pridá ho do H a jeho vzdialenosť prehlási za konečnú dĺžku najkratšej cesty z v do u. Následne prezrie susedov u a pre všetkých, ktorí sú ešte v O upraví najbližšie vzdialenosti, ak vzdialenosť do vrcholu u spolu s hranou medzi u a týmto susedom je kratšia

[3] Česi mu hovoria *Jarníkuv algoritmus*, pretože podľa nich ho Jarník vymyslel skôr ako Dijkstra. A vraj je to aj pravda.

[4] Prakticky väčšinou nejaká dostatočne veľká hodnota reprezentujúca nekonečno.

ako doteraz najbližšia známa vzdialenosť do tohto suseda. Všimnite si, že ak sme do tohto suseda zatiaľ nepoznali žiadnu cestu, tak máme v jeho vzdialenosti nekonečno a preto nová cesta bude určite kratšia.

Prečo je to korektné a môžeme prehlásiť vzdialenosť najbližšieho vrcholu v O, ktorý označíme w za finálnu? Ak by pri zaraďovaní w do hotových nemal ešte zistenú optimálnu cestu, potom by musela existovať nejaká lepšia. Táto cesta by nutne musela ísť cez nejaký iný vrchol v O. Ak by totiž nešla, potom by prechádzala len cez hotové vrcholy z H. Ale potom, keďže je kratšia, by sme o nej už vedeli a poznali jej hodnotu. Objavili by sme ju totiž v momente, keby bol do H pridávaný posledný vrchol na tejto ceste pred w. V tomto momente by sme zistili, že existuje kratšia cesta, ktorá prechádza cez tento vrchol a nastavili by sme podľa nej hodnotu vzdialenosti w. Preto by teda musela kratšia cesta prechádzať cez nejaký vrchol v O. Lenže tu sa dostávama do sporu, keďže v momente, ako by prišla do tohto vrcholu v O by nemohla byť kratšia, keďže všetky vrcholy v O sú aspoň také vzdialené ako w.

Existujú rôzne implementácie, ktoré sa líšia hlavne spôsobom pamätania si množiny O. Tá najzákladnejšia využíva jednoduché polia, v ktorých má uloženú pre každý vrchol najbližšiu vzdialenosť a to, či už je vrchol hotový. Potom vždy prejde celé pole, vyberie najbližší z nehotových a prehlási ho za hotový. Následne prejde všetkých jeho susedov a prípadne im skráti vzdialenosť. Jedno takéto pripájanie trvá $O(N)$ a robí sa $N-1$ krát, keďže zakaždým pribudne do množiny hotových 1 vrchol. Preto je celkový čas tejto implementácie $O(N^2)$. Keďže máme graf zapamätaný ako maticu susednosti, táto implementácia je nezávislá od počtu hrán, čo je jej výhoda v prípade hustého grafy. Iné implementácie využívajú haldu alebo fibonacciho haldu.

Ako sa to robilo zle?

Dôležitým faktorom v tomto príklade je nádrž auta o objeme K a pumpy v mestách. Ak by bola pumpa v každom meste, nebol by problém jednoducho eliminovať z grafu hrany dlhšie ako K a spustiť z vrcholu 1 Dijkstrov algoritmus. Avšak podobne ako v reálnom živote, pumpy v každom meste nie sú. Pri realizácii Dijkstrovho algoritmu nás teda mohlo napadnúť pamätať si stav nádrže. Tú by sme si pri príchode do mesta s pumpou doplnili a ak by sa nám na nejakej ceste mohla vyprázdniť nádrž, tak by sme po nej nešli. Také ľahké to však nebude. Môže sa nám totiž stať, že prídeme do nejakého mesta bez pumpy po najkratšej ceste, avšak s pomerne prázdou nádržou, s ktorou sa nedostaneme ďalej. Keby sme však nevolili pri presúvaní do tohto mesta nakratšiu cestu, mohli by sme ísť cez nejakú pupmu a prísť síce v horšom čase, ale zato s plnšou nádržou a väčšou nádejou, že sa dostaneme ďalej. Ale zase, ak budeme klášť na prvé miesto stav nádrže, môže sa nám stať, že nenájdeme najkratšiu cestu, pretože nám naozaj môže stačiť prádznejšia nádrž, veď čo ak je hneď za rohom pumpa? Takže sa dostávame do situácie, v ktorej máme dve neporovnateľné kritéria - dĺžka cesty a stav nádrže. Neporovnateľné preto, pretože vo všeobecnosti nevieme, ktorá je tá najkratšia cesta spomedzi tých, pri ktorých nám palivo na pokračovanie ešte vystačí. Kto implementoval Dijkstru s uprednostňovaním plnej nádrže alebo uprednostňovaním najkratšej cesty mal zlé riešenie a kto implementoval riešenie, ktoré odhaľovalo mestá podľa najkratšej cesty, ale dovolilo vojsť do mesta aj viackrát, ak to bolo s plnšou nádržou, mal riešenie korektné, ale žalostne pomalé. Do jedného mesta tak totiž môžeme vojsť až rádovo K krát a keďže potom vojdeme znova aj do všetkých nasledujúcich, dostaneme exponenciálnu časovú zložitosť.

A ako sa to malo robiť?

Korektné a pomerne rýchle riešenie dostaneme nasledovnou úvahou: najkratší sled od začiatočného mesta do ciela pôjde cez niekoľko[5] púmp, pričom medzi každými dvoma pumpami pôjde najkratšou možnou cestou. Dôležité bude, že každú pumpu navštívime určite

[5]Možno aj nula

najviac raz, pretože viac krát sa nám to neoplatí. Pracovne si dáme pumpu do začiatočného a konečného vrcholu a potom nejakým spôsobom zistíme vzdialenosti najkratších ciest medzi pumpami. Následne si treba spraviť nový graf, v ktorom budú vrcholy len pôvodné pumpy a hrany budú vyrátané najkratšie cesty medzi pumpami. V tomto novom grafe stačí nájsť najkratšiu cestu z 1 do N a to je zároveň náš výsledok. Všimnite si, že táto najkratšia cesta v novom grafe nemusí byť cestou v pôvodnom, ale môže to byť ten v úvode spomínaný sled. To preto, lebo ak mám najkratšiu cestu z pumpy i do pumpy j a potom najkratšiu cestu z pumpy j do pumpy k, kľudne sa môže stať, že tieto dve cesty prechádzajú spoločnou hranou. Skúste vymyslieť príklad! Môžete sa inšpirovať druhým odstavcom tohto vzoráku.

Ostáva odpovedať na otázku, aký čas bude mať naše riešenie. Skôr ale treba odpovedať na otázku, akým spôsobom budeme hľadať dĺžky najkratších ciest medzi každou dvojicou púmp. Môžeme na to použiť Floyd-Warshallov algoritmus s časom $O(N^3)$, alebo môžeme spustiť z každej pumpy Dijkstru, čím dostaneme $O(PN^2)$, ak P je počet púmp. Keďže zvyšok postupu stíhame v $O(N^2)$, stáva sa táto fáza najpomalšou a rozhodujúcou. Za prvú možnosť som dával 14 bodov a za druhú možnosť 15.

Posledný detail je vypísanie cesty. V prípade Dijkstrovho algoritmu stačí len troška upraviť postup, aby si pre každé mesto pamätal svojho predchodcu na ceste. Takto získame nielen postupnosť púmp, ale aj postupnosť miest medzi každými dvoma pumpami.

Listing programu:

```
#include <cstdio>
#define MAXN 100
#define INF 100000

//graf je povodny graf, graf2 bude prerobeny graf
int
graf[MAXN][MAXN],graf2[MAXN][MAXN],predchodca[MAXN][MAXN],pumpa[MAXN],N,M,K;

int min(int a,int b){return a < b ? a : b;}

//dijkstrov algortitmus. nastavuje polia res a predchodca
void dijkstra(int graf[][MAXN],int res[], int predchodca[], int start){
    //tu si pamatame pre vrcholy, ze ci uz hotove
    int hotove[N];
    for (int i=0;i<N;i++){hotove[i]=0;}
    for (int i=0;i<N;i++){res[i]=INF;}
    res[start]=0; hotove[start]=1; predchodca[start]=-1;
    //nastavime pdla hran zaciatocneho vrcholu, ze pozname vzdialenosti
    for (int i=0;i<N;i++){if (graf[start][i]!=INF){res[i]=graf[start][i];
predchodca[i]=start;}}
    for (int i=0;i<N-1;i++){
        int minv=-1;
        for (int j=0;j<N;j++)if (hotove[j]==0 &&
res[j]!=INF){if((minv==-1)||(res[j]<res[minv])){minv=j;}}
        //nenasli sme najblizsi vrchol (nesuvisly graf?)
        if (minv==-1){break;}
        //nastavime si, ze tento vrchol je hotovy a upravime vzdialenosti do susedov
        for (int j=0;j<N;j++)if (res[j] > res[minv]+graf[minv][j]){
            res[j] = res[minv]+graf[minv][j];
            predchodca[j]=minv;
        }
        hotove[minv]=1;
    }
    return;
}
```

```
int main(){
    scanf("%d %d %d ",&N,&M,&K);
    for (int i=0;i<N;i++) for(int j=0;j<N;j++){graf[i][j]=graf2[i][j]=INF;}
    for (int i=0;i<N;i++){graf[i][i]=0;}
    for (int i=0;i<M;i++){
        int x,y,z;
        scanf("%d %d %d ",&x,&y,&z);
        //vrcholy indexujeme od 0
        x--; y--;
        graf[x][y]=graf[y][x]=z;
    }
    for (int i=0;i<N;i++){scanf("%d ",&pumpa[i]);}
    //na start a do ciela postavime pumpu
    pumpa[0]=1; pumpa[N-1]=1;
    //z kazdej pumpy spustime dijkstru
    for (int i=0;i<N;i++)if (pumpa[i]==1){
        int A[MAXN];
        dijkstra(graf,A,predchodca[i],i);
        //a upravime, ak je vzdialenost taka velka, ze nam nestaci nadrz
        for (int j=0;j<N;j++)if (pumpa[j]==1 && A[j] <= K){graf2[i][j] = A[j];}
    }
    int res[MAXN],pred[MAXN];
    //spustime dijkstru na upravenom grafe
    dijkstra(graf2,res,pred,0);
    if (res[N-1]==INF){
        printf("Do cieloveho mesta sa neda dostat.\n");
        return 0;
    }
    //a ak nasiel cestu, ideme ju vypisovat
    printf("Dlzka najkratsej cesty: %d\n",res[N-1]);
    int cesta[MAXN], cesta2[MAXN], where=N-1, p=0, pc=0;
    while(where!=0){
        cesta[p]=where;
        where=pred[where];
        p++;
    }
    cesta[p]=0;
    p++;
    for (int i=p-1;i>0;i--){
        int where2=cesta[i];
        while (where2!=cesta[i-1]){
            cesta2[pc]=where2;
            where2=predchodca[cesta[i-1]][where2];
            pc++;
        }
    }
    printf("Cesta:");
    for (int i=0;i<pc;i++){printf(" %d",cesta2[i]+1);}
    printf(" %d\n",N);
    return 0;
}
```

8. Ospalý Lukáš

opravoval Lukáš
(max. 15 bodov)

Úlohu vyriešime dynamickým programovaním. Trochu nám robí problém to, že prvá časť dňa nasleduje po poslednej, teda v istom zmysle sú časti dňa v kruhu. V takomto prípade

nemôžeme len tak bez rozmyslu použiť dynamické programovanie. Nemáme totiž žiadne triviálne hodnoty, od ktorých by sme mohli začať počítať.

Najprv si popíšeme pomalšie riešenie. Ak $N = K$, Lukáš môže spať celý deň. V zadaní však medzi riadkami bolo napísané, že $N > K$, takže prípadom $N = K$ sa nebudeme zaoberať[6]. Z $N > K$ vyplýva, že aspoň jednu časť dňa Lukáš neprespí. Na tomto mieste by sme mohli náš kruh (deň) rozstrihnúť a ďalej pokračovať dynamickým programovaním. Časti dňa si prečíslujeme tak, ako keby sa deň začínal na mieste, kde sme ho prestrihli.

V algoritme budeme rátať hodnotu $p(i, j, k)$ pre $0 \leq i \leq N, 0 \leq j \leq K, 0 \leq k \leq 1$, čo znamená, že vieme získať $p(i, j, k)$ energie v prvých i častiach dňa tak, že prespíme dokopy j častí a ak $k = 0$, tak i-tu časť sme neprespali; ak $k = 1$, tak i-tu časť sme prespali. Nech hodnoty energie pre jednolivé časti dňa sú $e_1, e_2, \ldots, e_N$, potom

$$p(i, j, 0) = \max\{p(i-1, j, 0), \quad p(i-1, j, 1)\}$$

$$p(i, j, 1) = \max\{p(i-1, j-1, 0), \quad p(i-1, j-1, 1) + e_i\}$$

V prvom prípade sa v i-tej časti dňa nespí, takže nepribúda žiadna energia. V druhom prípade pribúda energia e_i ak sme počas predchádzajúcej časti dňa spali. Ešte doplníme, že pre počiatočnú hodnotu $p(1, 0, 0)$ platí $p(1, 0, 0) = 0$ (lebo sme povedali, že kruh prestrihneme v časti dňa, počas ktorej nebude Lukáš spať).

Predpokladajme, že tabuľku p sme už úspešne vyrátali. Lukáš prespí z N častí dňa práve K častí, preto výsledkom algoritmu bude maximum z hodnôt $p(N, K, 0), p(N, K, 1)$ pre všetky rozstrihnutia kruhu. Vyrátanie jedného políčka tabuľky trvá konštantne dlho, preto vyrátanie celej tabuľky trvá $O(NK)$. Tabuľku musíme vyrátať pre každé miesto, kde sa dá kruh rozstrihnúť, preto je celková zložitosť $O(N^2K)$.

Ako vidíte, vzorák sa ešte nekončí. Zložitosť vieme zlepšiť. Teraz nebudeme strihať deň na všetkých N miestach, ale iba medzi prvou a poslednou časťou dňa. Prestrihneme ho aj v prípade, ak počas poslednej alebo prvej časti Lukáš spí.

Použijeme podobný prístup ako predtým. Tabuľku p však zväčšíme dvakrát: pridáme ešte jednu súradnicu, teda políčko bude mať súradnice $p(i, j, k, l)$, pričom $0 \leq l \leq 1$. Hodnota $p(i, j, k, 0)$ znamená, že počas úplne prvej časti dňa Lukáš nespí, naopak $p(i, j, k, 1)$ znamená, že počas prvej časti dňa Lukáš spí. Na začiatku platí $p(1, 0, 0, 0) = p(1, 1, 1, 1) = 0$, teda počas prvej časti nezíska Lukáš žiadnu energiu. Zvyšné hodnoty vyrátame podobne, ako v pomalšom algoritme:

$$p(i, j, 0, l) = \max\{p(i-1, j, 0, l), \quad p(i-1, j, 1, l)\}$$

$$p(i, j, 1, l) = \max\{p(i-1, j-1, 0, l), \quad p(i-1, j-1, 1, l) + e_i\}$$

Teraz ešte musíme nájsť hodnotu, ktorú náš algoritmus prehlási za výsledok. Kandidátmi sú hodnoty $p(N, K, 0, 0), p(N, K, 0, 1), p(N, K, 1, 0)$ a špeciálne $p(N, K, 1, 1) + e_1$. Z týchto hodnôt vyberieme maximum. Prvé tri hodnoty sú také, že buď v prvej alebo poslednej časti dňa Lukáš nespal. Preto kruh môžeme smelo rozstrihnúť medzi prvou a poslednou časťou dňa. Kvôli strihaniu musíme hodnotu $p(N, K, 1, 1)$ zvýšiť o e_1. Počas prvej aj poslednej časti dňa Lukáš spí, teda v prvej časti dňa získa e_1 energie. V algoritme sme však rátali s tým, že počas prvej časti dňa nezíska žiadnu energiu.

Časová zložitosť algoritmu je $O(NK)$, keďže každé políčko tabuľky vieme vyrátať v konštantnom čase. V jednom momente si stačí pamätať iba posledné dva riadky tabuľky, teda pamäťová zložitosť je $O(K)$.

[6] Veď kto by chcel prespať celý svoj život?

Listing programu:

```
#include <algorithm>
#include <cstdio>
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)

int main() {
    int e[1000];
    int p[2][1000][2][2];
    int n, b; scanf("%d %d", &n, &b);
    REP(i,n) scanf("%d", &e[i]);

    REP(i,2) REP(k,b+1) REP(z,2) REP(kon,2)
        p[i][k][z][kon] = -123456789;
    p[0][1][1][1] = 0;
    p[0][0][0][0] = 0;
    FOR(i,1,n)
    {
        int y = i % 2, x = (i + 1) % 2;
        REP(z,2) REP(j,i+1)
        {
            p[y][j][z][0] = max(p[x][j][z][0], p[x][j][z][1]);
            p[y][j + 1][z][1] = max(p[x][j][z][1] + e[i], p[x][j][z][0]);
        }
    }
    p[(n + 1)%2][b][1][1] += e[0];

    int ma = 0, y = (n + 1) % 2;
    REP(z,2) REP(kon,2) ma = max(p[y][b][z][kon], ma);
    printf("%d\n", ma);
}
```

Opravoval Mic
(max. 15 bodov)

9. Tuniakové a iné bagety

Tento príklad ma opäť nepotešil, riešení bolo málo:-(Bodovanie bolo jednoduché. Váš program sa spustil na 13 vstupoch, za každý správne vyriešený ste mohli dostať 15/13 bodu. Body sa zaokrúhľovali nahor. Spravodlivé. Nie?

Teraz sa môžeme nehorázne vrhnúť na vzorové riešenie. Nedočkavcom prezradím, že vzorové riešenie má časovú zložitosť $O(N \log N)$ a pamäťovú zložitosť $O(N)$. Ako na to? Keďže môže existovať viac riešení tejto úlohy, my budeme hľadať také, v ktorom sa bagety jedia čo najneskôr, ako sa len dá (aby Mirko neostal hladný, keď pôjde domov). Ako ste mohli zo zadania vytušiť, Mirko je pažravý a teda použijeme greedy[7] prístup.

Najskôr si všetky úlohy utriedime podľa konca (úlohy, ktoré končia skôr, budú skôr v našom poradí). Nech a_i je začiatok i-tej úlohy, b_i je jej koniec a c_i je počet bagiet, ktoré musí Mirko zjesť počas vykonávanie danej úlohy. Predstavme si, že máme riešenie pre $N-1$ úloh (teda pre úlohy $1, 2, \cdots, N-1$) také, že bagety sa jedia čo najneskôr, ako sa len dajú. Ak $b_{N-1} < a_N$, tak potom bude riešenie také, že všetky potrebné bagety pre posledný interval dáme čo najviac na koniec posledného intervalu. V prípade, že $b_{N-1} \geq a_N$, tak sa pozrieme na počet bagiet, ktoré sú v intervale $[a_N, b_{N-1})$ riešení pre $N-1$ úloh. Zo všetkých riešení

[7] greedy = pažravý

pre $N-1$ intervalov je v tom intervale bagiet najviac (lebo bagety sa jedia čo najneskôr, ako sa dajú). Nech ich je B. Potom do nášho posledného intervalu musíme vložiť $c_N - B$ bagiet tak, aby boli čo najneskôr, ako sa len dá. Takto sme našli optimálne riešenie pre N intervalov.

Riešenie sa nám už začína celkom dobre črtať. Po utriedení zostrojíme riešenie pre 1 úlohu, z neho riešenie pre dve úlohy, ..., až kým nebudeme mať riešenie pre N úloh. Ako ale zostrojiť nové riešenie efektívne (v $O(\log N)$)? Budeme na to potrebovať rýchlo vedieť robiť dve veci. Prvou z nich je zistenie, koľko bagiet už bolo pridaných niekedy v minulosti v čase úlohy, ktorú pridávame a druhou je zistiť, kam dať nové bagety. V našom riešení[8] sú bagety za sebou súvislo, maximálne v N intervaloch. Majme intervaly utriedené a majme pre každý interval vypočítané, koľko bagiet je v ňom a vo všetkých intervaloch pred ním. Binárnym vyhľadávaním nájdeme interval, ktorý obsahuje a_N (alebo interval, ktorý je za a_N a medzi ním a a_N nie je žiadny ďalší interval bagiet). Keďže si pamätáme čiastočné súčty, tak vieme, v konštantnom čase zistiť počet bagiet, ktoré ešte treba dodať. Potrebné bagety rozložíme nasledovne. Zoberieme b_N a pred neho dáme čo najviac bagiet tak, aby sa neprekrývali s predchádzajúcim riešením. Ak sa prekryjú, tak posledný interval bagiet zrušíme, pridáme ho k nášmu vytváranému a pokračujeme ďalej až kým nemáme náš interval plne pokrytý bagetami a neprekrýva sa s predchádzajúcimi intervalmi. Toto by v najhoršom prípade mohlo trvať aj $O(N)$ času, ale každý interval bagiet tam maximálne raz pridáme, a maximálne raz uberieme, takže celková zložitosť tohto bude $O(N)$ (pre každú robotu bude maximálne jeden interval).

Binárne vyhľadávanie nám zaberie $O(\log N)$ času, robíme ho N krát, triedime raz, čo je v $O(N \log N)$ a prerábanie intervalov nám zaberie $O(N)$ času. Preto časová zložitosť bude $O(N \log N)$.

Listing programu:

```
#include <vector>
#include <algorithm>
#include <cstdio>
#include <cstdlib>
using namespace std;

struct Bageta{
    int a,b,c;
    Bageta(int a,int b,int c){
        this->a=a;this->b=b;this->c=c;
    }

};

bool cmp(const Bageta &a,const Bageta &b){
    return a.b<b.b;
}

int main(){
    int N;
    vector<Bageta> B;
    vector<pair<int,int> > intervaly;
    vector<int> sucty;
    scanf(" %d",&N);
    for(int i=0;i<N;i++){
        Bageta b(0,0,0);
        scanf(" %d %d %d",&b.a,&b.b,&b.c);
```

[8] Bagety čo najneskôr, ako sa len dajú

```
        B.push_back(b);
    }
    sort(B.begin(),B.end(),cmp);
    intervaly.push_back(make_pair(B[0].b-B[0].c+1,B[0].b));
    sucty.push_back(B[0].c);
    for(int i=1;i<N;i++){
        int z=0,k=intervaly.size();
        while(z<k){
            int s=(z+k)/2;
            if(intervaly[s].second<B[i].a)z=s+1;else
                if(intervaly[s].first>B[i].a)k=s;else
                    z=k=s;
        }
        if(z>=intervaly.size()){
            intervaly.push_back(make_pair(B[i].b-B[i].c+1,B[i].b));
            sucty.push_back(sucty[sucty.size()-1]+B[i].c);
        }else{
            int suc=sucty[sucty.size()-1];
            if(intervaly[z].first<B[i].a)
                suc-=B[i].a-intervaly[z].first;
            if(z>0)suc-=sucty[z-1];
            int treba=B[i].c-suc;
            if(treba>0){
                pair<int,int> I(B[i].b-treba+1,B[i].b);
                while(intervaly.size()>0&&I.first<=intervaly[intervaly.size()-1].second){
                    int presiaha=intervaly[intervaly.size()-1].second-I.first+1;
                    I.first=intervaly[intervaly.size()-1].first-presiaha;
                    intervaly.resize(intervaly.size()-1);
                    sucty.resize(sucty.size()-1);
                }
                intervaly.push_back(I);
                sucty.push_back(I.second-I.first+1);
                if(sucty.size()>1)sucty[sucty.size()-1]+=sucty[sucty.size()-2];
            }
        }
    }
    printf("%d\n",sucty[sucty.size()-1]);
    return 0;
}
```

10. Obchodný cestujúci má hlad

opravoval Lukáš
(max. 15 bodov)

Úlohu vyriešime trikovo. Graf zo vstupu trochu poupravíme a na ňom nájdeme maximálny tok s najmenšou cenou. Najprv si však povieme, čo je to tok. Graf si predstavíme ako sieť potrubí. Nech s je vrchol, z ktorého do grafu vteká voda a t je vrchol, z ktorého voda vyteká. Vrchol s voláme zdroj, t je odtok. Pre všetky ostatné vrcholy okrem týchto dvoch platí, že množstvo vody, ktoré do vrchola pritečie, sa rovná množstvu vody, ktoré z vrchola odtečie. Navyše každá hrana v grafe má určenú kapacitu – najväčší prietok vody, aký môže hranou tiecť.

Zavedieme ešte aj cenu pre každú hranu. Cenu toku po jednej hrane získame tak, že množstvo pretekajúcej vody vynásobime cenou hrany. Teda cena toku na jednej hrane je priamo úmerná možstvu vody, ktoré ňou preteká. Cenu toku v celom grafe získame tak, že sčítame ceny tokov na jednotlivých hranách. Budeme hľadať maximálny tok s minimálnou

cenou. Táto formulácia znamená, že prvoradé kritérium je pre nás veľkosť toku a až potom cena za tento tok.

Ako upravíme vstupný graf? Chceme získať najkratší cyklus prechádzajúci vrcholmi v_1 a v_N. Takýto cyklus sa skladá z dvoch ciest z v_1 do v_N, ktoré sa nekrižujú. Zároveň dĺžka týchto dvoch ciest je v súčte najmenšia možná. Kapacita hrán bude taká, aby tok z v_1 do v_N bol práve dva (jeden pre každú cestu). Graf, ktorý vyrobíme, bude orientovaný. Navyše bude platiť, že ak existuje hrana z u do w, tak neexistuje opačná hrana z w do u.

Cesty sa nemôžu križovať, preto z vrcholu w v pôvodnom grafe rôzneho od v_1 a v_N spravíme dva vrcholy w^+ a w^-, pričom do vrcholu w^+ budú hrany vchádzať a z w^- budú vychádzať. Medzi w^+ a w^- pôjde jedna hrana s kapacitou 1. Týmto zabezpečíme, že cez každý vrchol bude tiecť tok najviac 1, teda cez každý vrchol pôjde najviac jedna cesta.

Vďaka vhodným kapacitám hrán nájdeme maximálny tok veľkosti 2 taký, že predstavuje dve nepretínajúce sa cesty. Chceme minimalizovať dĺžku týchto dvoch ciest v súčte, preto hranám priradíme také ceny, aké sme dostali na vstupe. Presnejšie, ak je na vstupe hrana z u do v s cenou c, vyrobíme hranu z u^- do v^+ a z v^- do u^+ s cenou c. Kapacitu môžeme zvoliť ľubovoľnú väčšiu ako 1. Hrany vrámci jedného vrcholu (teda medzi v^+ a v^-) budú mať nulovú cenu, keďže sme ich pridali umelo do grafu len preto, aby sa nekrižovali cesty.

Aby celý algoritmus fungoval, musíme ešte hrane z v_1^+ do v_1^- dať kapacitu 2 a nulovú cenu. Na takomto novom grafe spustíme algoritmus, ktorý hľadá maximálny tok s minimálnou cenou z vrcholu v_1^+ do v_N^+. Algoritmus nám vráti takýto tok a z neho už ľahko zrekonštruujeme najkratší cyklus prechádzajúci vrcholmi v_1 a v_N.

Ostáva nám už len doriešiť, ako budeme hľadať maximálny tok s minimálnou cenou. Podobne ako pri probléme maximálneho toku zavedieme pojem reziduálneho grafu. Predstavme si, že v grafe G už máme nejaký tok T. Nech po hrane z u do v s kapacitou c_{uv} tečie t_{uv} vody a cena hrany je p_{uv}. Potom v reziduálnom grafe G_T bude mať táto hrana kapacitu $c_{uv} - t_{uv}$ a cenu p_{uv}. $c_{uv} - t_{uv}$ je maximálneho množstvo vody, aké môžeme ešte po hrane poslať a cenu toku zvýšime o p_{uv} za každú jednotku toku. Pre opačný smer z v do u bude kapacita t_{uv}, lebo hrany sú orientované, takže opačným smerom môžeme tok znižovať iba po nulu. Všimnime si, že ak posielame vodu opačným smerom, tok na hrane znižujeme a tým znižujeme aj jeho cenu. Teda cena tejto hrany v reziduálnom grafe bude $-p_{uv}$. Pre jednoduchosť v reziduálnom grafe nebudeme uvažovať hrany s nulovou kapacitou, keďže po nich aj tak nemôžeme nič poslať.

Platí nasledovné tvrdenie o reziduálnom grafe: Nech T je maximálny tok v grafe G. Potom T je maximálny tok s minimálnou cenou práve vtedy, keď reziduálny graf G_T neobsahuje záporný cyklus. Tvrdenie nebudeme dokazovať. Záujemci si dôkaz nájdu v múdrych knižkách alebo vymyslia sami.

Vďaka tomuto tvrdeniu funguje nasledovný algoritmus: Začneme s nulovým tokom. Budeme postupovať v niekoľkých iteráciách, pričom v každej nájdeme najkratšiu cestu zo zdroja s do t v reziduálnom grafe a po tejto najkratšej ceste zvýšime tok.

Počas každej iterácie algoritmu platí, že graf neobsahuje záporný cyklus. Tvrdenie dokážeme matematickou indukciou. Pre nulový tok reziduálny graf neobsahuje záporný cyklus, lebo ceny všetkých hrán s kladnými kapacitami v reziduálnom grafe sú nezáporné. Sporom predpokladajme, že v i-tej iterácií sme mali reziduálny graf bez záporného cyklu, ale po zvýšení toku na najkratšej ceste z s do t (označme ju P) nám v grafe vznikol záporný cyklus. Určite existuje aspoň jedna hrana (u, v) z P taká, že (v, u) leží na novovzniknutom zápornom cykle. Nech x je cena záporného cyklu bez hrany (v, u). Platí $x + p_{vu} = x - p_{uv} < 0$, z čoho $x < p_{uv}$. Teda keby sme hranu (u, v) v najkratšej ceste P nahradili časťou záporného cyklu bez (v, u), dostali by sme kratšiu cestu. A to je spor, lebo P je najkratšia cesta v reziduálnom grafe.

Po každej iterácii nášho algoritmu teda budeme mať reziduálny graf bez záporného cyklu. Keď už nebude existovať cesta z s do t v reziduálnom grafe, našli sme maximálny tok. A

vďaka vyššie uvedenému tvrdeniu bude tento tok aj maximálnym tokom s minimálnou cenou.

Poďme sa teraz podrobnejšie pozrieť na detaily algoritmu. V našom prípade bude mať maximálny tok veľkosť 2, teda stačia nám dve iterácie. V každej iterácií potrebujeme nájsť najkratšiu cestu v reziduálnom grafe. Tu si treba dať pozor na to, že graf obsahuje aj záporné hrany, takže nemôžeme použiť Dijkstrov algoritmus, ale napríklad Bellman-Fordov. Časová zložitosť tohto algoritmu je $O(MN)$, kde M je počet hrán a N je počet vrcholov. Takéto bolo aj jediné riešenie, čo prišlo. Preto si zaslúžilo 15 bodov.

Listing programu:

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

int main() {
    int n, m;
    cin >> n >> m;
    int cena[100][100];
    int tok[100][100], kap[100][200];
    memset(tok, 0x00, sizeof(tok));
    memset(kap, 0x00, sizeof(kap));
    memset(cena, 0x00, sizeof(cena));

    for (int i=0; i<m; i++)
    {
        int a, b, c;
        cin >> a >> b >> c;
        a--; b--;
        kap[2*a+1][2*b] = kap[2*b+1][2*a] = 1;
        cena[2*a+1][2*b] = cena[2*b+1][2*a] = c;
        cena[2*b][2*a+1] = cena[2*a][2*b+1] = -c;
    }

    kap[0][1] = 2;
    for (int i=1; i<n; i++)
        kap[2*i][2*i+1] = 1;

    n *= 2;
    vector<bool> zm(n), zm2(n);
    vector<int> res;
    for (int r=0; r<2; r++)
    {
        vector<double> vz(n, 123456789);
        vector<int> p(n, -1);

        vz[0] = 0;
        fill(zm.begin(), zm.end(), false);
        zm[0] = true;

        for (int j=0; j<n; j++)
        {
            fill(zm2.begin(), zm2.end(), false);
            for (int i=0; i<n; i++) if (zm[i]) for (int k=0; k<n; k++)
                if (tok[i][k] < kap[i][k] && vz[i] + cena[i][k] < vz[k])
                {
                    vz[k] = vz[i] + cena[i][k];
                    p[k] = i;
                    zm2[k] = true;
```

```
                }
            if (count(zm2.begin(), zm2.end(), true) == 0)
                break;
            swap(zm, zm2);
        }

        int q = n-2;
        if (p[q] == -1) break;
        while (p[q] != -1)
        {
            tok[p[q]][q]++;
            tok[q][p[q]]--;
            q = p[q];
            if (q % 2 == 0)
                res.push_back(q / 2);
        }
        reverse(res.begin(), res.end());
        if (r == 0) res.push_back((n-2) / 2);
    }

    if (tok[0][1] != 2)
        cout << "Okruzna cesta neexistuje" << endl;
    else
    {
        for (unsigned i=0; i<res.size(); i++)
            cout << res[i] + 1 << " ";
        cout << endl;
    }
}
```

Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
1	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	3	46	15	15	15	15	14	120
2	Kekely Michal	Gym. Varšavská Žilina - Vlčince	1	52	15	15	8	15	12	117
3	Hozza Ján	Gym. Jura Hronca BA	1	63	15	15	15	8		116
4	Csiba Peter	Škola pre mim. nadané deti BA	3	43	15	15	13	15	3	104
5	Kuzma Tomáš	Gym. Alejová Košice	3	58	15	15		15		103
6	Proksa Ondrej	Gym. Párovská Nitra	3	49	15	15	7	11	3	100
7	Bačo Ladislav	Gym. Poštová Košice	2	56	15	15		8		94
7	Špánik Marián	Gym. sv. Františka Žilina	3	47	15	14	8	7	3	94
9	Macháč Juraj	Gym. Jura Hronca BA	1	45	15	15	8	7		90
9	Rohár Pavol	Gym. Alejová Košice	2	52	15	12		11		90
9	Vavrík Boris	Gym. Jura Hronca BA	1	44	15	15	8	8		90
9	Špano Marek	Gym. Jura Hronca BA	1	30	12	15	15	15	3	90
13	Phuong Mariana	Gym. Jura Hronca BA	1	38	15	15	15			83
14	Kikta Michal	Gym. Tatarku Poprad	1	32	15	15	8	8	3	81
15	Jursa Jakub	Gym. Alejová Košice	3	27	15	15	7	15		79
16	Korbaš Rafael	Gym. Hronská BA	1	33	15	12	8	8		76
17	Ziman Michal	Gym. Haličská Lučenec	2	27	15	14	10	7		73
18	Páleník Martin	Gymnázium Levice	3	34	7	13	8	2	3	67
19	Majdiš Mojmír	Gym. Dolný Kubín	2	30	15	13		8		66
19	Pinter Martin	Gym. Jura Hronca BA	1	30	15	14	7			66
21	Šrámek Martin	Gym. Alejová Košice	3	0	15	15	15	15		60
22	Miklovič Tomáš	Gym. Nové Zámky	2	18	15	13	7	6		59
23	Sayedová Monika	Gym. Grösslingová BA	4	25	15		10	7		57
24	Popovič Viktor	Gymnázium J.A. Raymana	2	22		15	11	8		56
25	Macháč Matej	Gym. P. de Coubertina Piešťany	2	23	14	14		2		53
25	Ort Miroslav	Gym. Pankúchova Bratislava	3	31		14		8		53
25	Stančiaková Katarína	Gym. Jura Hronca BA	2	26	15		6	6		53
28	Habovštiak Martin	Gym. Tvrdošín	2	32	12					44
28	Hozzánková Hana	Gym. Jura Hronca BA	2	18		14	7	5		44
30	Polák Lukáš	Gym. Jura Hronca BA	2	24			8	7	3	42
31	Hašík Juraj	Gym. Grösslingová BA	2	39						39
31	Kudláč Jakub	Gym. Bilíkova BA	3	26	13					39
33	Mudrík Richard	Gym. J. M. Hurbana Čadca	4	23	13					36
34	Kučera Martin	Gym. Golianova Nitra	1	34						34
35	Repková Lucia	Gym. Párovská Nitra	2	0	15	10		8		33
36	Morvay Tomáš	Gym. Golianova Nitra	1	32						32
36	Čerman Ondrej	Gym. Partizánske	0	32						32
38	Cudrák Miloš	Gymnázium	4	13	15	2				30
38	Novella Tomas	Gym. Alejová Košice	2	0	12	3	9	6		30
40	Baxová Katarína	Gym. Ľudovíta Štúra Trenčín	3	28						28
41	Holas Juraj	Gym. Jura Hronca BA	1	27						27
42	Iring Andrej	Gym. Jura Hronca BA	1	26						26
43	Piovarči Michal	Gym. Hronská BA	3	25						25
43	Toman Viktor	Gym. Golianova Nitra	1	0	12		6	7		25
45	Sabo Michal	Gym. Jura Hronca BA	4	15						15
46	Chrást Lukáš	Gym. Golianova Nitra	3	13						13
46	Dávid Tóth	Gym. Daxnera V.n. Topľou	?	13						13
46	Kentoš Michal	Gymnázium	3	13						13
46	Masár Juraj	Gym. Bilíkova BA	1	13						13
50	Michalek Martin	Gym. P. de Coubertina Piešťany	3	12						12

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
50	Táňa Tóthová	Gym. Jura Hronca BA	2	12						12
52	Haffner Oto	Gym. Hlohovec	4	10						10
52	Paulech Matej	Gym. P. de Coubertina Piešťany	3	10						10
52	Tomkovič Viktor	Gym. Jura Hronca BA	3	10						10
52	Žák Samuel	Gym. Rajec	2	10						10
56	Bubnár Michal	Gym. sv. Františka z Assisi V.n. Topľou	3	9						9
57	Kovaľ Anton	Gym. L. Stöckela Bardejov	2	8						8
58	Kučera Erik	Gym. Hlohovec	4	5						5
59	Sebechlebský Ján	Gym. Žiar nad Hronom	1	1						1
60	Kudláč Matúš	Gym. Bilíkova BA	4	0						0
60	Pásztor Bálint	Gym. maďarské Šahy	3	0						0

Výsledková listina po 2. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda		8	9	10	Σ
1	Ondrúška Peter	SPŠ Dubnica nad Váhom	4	44	15	15	15	89
2	Fulla Peter	SPŠ Spišská Nová Ves	3	21	15	9		45
3	Hudec Tobiáš	Gym. Partizánske	2	15	15			30
4	Hozza Michal	Gym. Jura Hronca BA	3	15				15
5	Kukan Matúš	Gym. Grösslingová BA	4	0		13		13
6	Belan Tomáš	Škola pre mim. nadané deti BA	2	0	5			5
7	Proksa Ondrej	Gym. Párovská Nitra	3	1		3		4
8	Szabó Peter	SPŠE Nové Zámky	3	2				2
9	Ort Miroslav	Gym. Pankúchova Bratislava	3	0				0

Výsledková listina po 2. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda		4	5	6	7	8	Σ
1	Fulla Peter	SPŠ Spišská Nová Ves	3	75	15	15	15	14	15	149
2	Hudec Tobiáš	Gym. Partizánske	2	70	15	10	6	8	15	124
3	Belan Tomáš	Škola pre mim. nadané deti BA	2	47	15	13	6	15	5	101
4	Hozza Michal	Gym. Jura Hronca BA	3	63	15	3				81
5	Fekiač Jozef	Gym. Grösslingová BA	3	44	15	3				62
6	Proksa Ondrej	Gym. Párovská Nitra	3	32	11	3	1	8		55
7	Kekely Michal	Gym. Varšavská Žilina - Vlčince	1	20	15	12				47
8	Hojčka Michal	Gym. Partizánske	3	16	15	10		5		46
8	Hozza Ján	Gym. Jura Hronca BA	1	38	8					46
10	Csiba Peter	Škola pre mim. nadané deti BA	3	23	15	3	4			45
10	Kuzma Tomáš	Gym. Alejová Košice	3	26	15		4			45
12	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	3	11	15	14				40
13	Petrucha Michal	Gym. Metodova BA	3	17	15	3		3		38
14	Ondrúška Peter	SPŠ Dubnica nad Váhom	4	15					15	30
15	Kučera Martin	Gym. Golianova Nitra	1	18	8					26
15	Macháč Juraj	Gym. Jura Hronca BA	1	19	7					26
17	Špánik Marián	Gym. sv. Františka Žilina	3	13	7	3				23
18	Ort Miroslav	Gym. Pankúchova Bratislava	3	12	8					20
19	Bačo Ladislav	Gym. Poštová Košice	2	11	8					19
19	Rohár Pavol	Gym. Alejová Košice	2	8	11					19
21	Špano Marek	Gym. Jura Hronca BA	1	0	15	3				18
22	Vavrík Boris	Gym. Jura Hronca BA	1	9	8					17
23	Jursa Jakub	Gym. Alejová Košice	3	0	15					15
23	Šrámek Martin	Gym. Alejová Košice	3	0	15					15
25	Korbaš Rafael	Gym. Hronská BA	1	4	8					12
26	Kikta Michal	Gym. Tatarku Poprad	1	0	8	3				11
27	Hašík Juraj	Gym. Grösslingová BA	2	10						10
27	Polák Lukáš	Gym. Jura Hronca BA	2	0	7	3				10
27	Čerman Ondrej	Gym. Partizánske	0	10						10
30	Majdiš Mojmír	Gym. Dolný Kubín	2	0	8					8
30	Popovič Viktor	Gymnázium J.A. Raymana	2	0	8					8
30	Repková Lucia	Gym. Párovská Nitra	2	0	8					8
33	Macháč Matej	Gym. P. de Coubertina Piešťany	2	5	2					7
33	Sayedová Monika	Gym. Grösslingová BA	4	0	7					7
33	Toman Viktor	Gym. Golianova Nitra	1	0	7					7
33	Ziman Michal	Gym. Haličská Lučenec	2	0	7					7
37	Miklovič Tomáš	Gym. Nové Zámky	2	0	6					6
37	Novella Tomas	Gym. Alejová Košice	2	0	6					6
37	Pinter Martin	Gym. Jura Hronca BA	1	6						6
37	Stančiaková Katarína	Gym. Jura Hronca BA	2	0	6					6
41	Hozzánková Hana	Gym. Jura Hronca BA	2	0	5					5
41	Páleník Martin	Gymnázium Levice	3	0	2	3				5
43	Baxová Katarína	Gym. Ľudovíta Štúra Trenčín	3	2						2
43	Morvay Tomáš	Gym. Golianova Nitra	1	2						2