Korešpondenčný seminár z programovania
XXVI. ročník, 2008/09
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: MICROSTEP-MIS spol. s r.o.
Whitestein Technologies spol. s r.o.

Vzorové riešenia 2. kola zimnej časti

Milé naše riešiteľky, milí naši riešitelia, milé naše riešiteľatá,

zase raz sa dala mohutná mašinéria zvaná „banda lenivých vedúcich“ do pohybu a priniesla vám úplne nové vzoráky. Sú to dobré vzoráky... plné emócií, nádejí, správnych riešení, tragédií i splnených želaní. Veríme, že Vás potešia aspoň tak, ako nás ich úspešná kolaudácia.

Nečítajte túto vetu![1]

KSPáci

1. Zn. Kravina

Opravoval MMx
(max. 10 bodov)

Predstavme si zoznam náradia ako telefónny zoznam. Prelistovať celý zoznam kvôli každému menu, ktoré chceme nájsť, si trúfne len ozajstný optimista. O niečo lepší nápad je utriediť si zoznam mien, ktoré hľadáme, podľa abecedy. Potom nám stačí prelistovať celý zoznam len raz, pretože vieme, v akom poradí v ňom máme hľadané mená očakávať. Ale stále musíme v najhoršom prípade prečítať každé jedno meno zo zoznamu. Presnejšie povedané, zložitosť tohoto algoritmu je $O(N + K \log K)$. Ak je $N \gg K$ (čo zadanie naznačilo), potom zložitosť tohto algoritmu $O(N)$ je horšia ako zložitosť vzorového. Ak by K bolo rádovo rovnako veľké ako N, potom budú obidva algoritmy porovnateľne rýchle..

Optimálne riešenie funguje podobne, ako snáď každý, kto taký telefónny zoznam dostane do rúk: Otvorí ho na nejakom mieste, tým sa zoznam rozdelí na dve časti. Potom sa pozrie či hľadané meno je v abecede skôr alebo neskôr a podľa toho sa zaoberá už len prvou alebo druhou časťou. Teraz potrebuje riešiť rovnaký problém, len na menšej časti zoznamu, teda nič mu nebráni postup opakovať až kým sa nedopracuje k miestu, kde by hľadané meno patrilo.

Tento algoritmus binárneho vyhľadávania je známy tým, že ak si dopredu nepremyslíte detaily ako „v ktorej časti poľa ešte hľadám“ alebo „kedy sa mám zastaviť“, potom strávite dlhé zimné večery naháňaním kadejakých plus mínus jednotiek. Na pomoc prichádza invariant: to je tvrdenie, ktoré v istom bode programu vždy platí. Napríklad vo vzoráku platí vo vnútri while-cyklu, že a ukazuje na prvý prvok kde ešte môže byť výsledok a b ukazuje hneď za posledný takýto prvok. Preto napríklad keď zistím, že kravina[2] požiadavka je skôr ako testovaný prvok, môžem s istotou priradiť $b := x$, lebo viem že x je prvý prvok, v ktorom už výsledok nebude. Keď mi premenné a a b ukazujú už len na jeden prvok, nie je dôvod ďalej pokračovať, pretože jeden prvok ľahko otestujem, či to je výsledok.

Časová zložitosť jedného takéhoto hľadania závisí od toho, koľkokrát sa pole dĺžky N dá rozdeliť napoly, aby nám ostal aspoň jeden prvok. To nám približne vyjadruje funkcia $\log_2 N$, teda časová zložitosť je $O(K \log N)$. Ak do pamäťovej zložitosti započítame aj zoznam náradia, bude to $O(N)$, ináč bude konštantná.

[1] Kto sa pozrel aj sem je ale úplná lama. A kto sa sem pozrel aj minulý rok, je ešte väčšia lama.

[2] A toto nie je kravina!

Listing programu:

```
var i,n,kva     :longint;
    naradie     :array[0..100] of string;
    policka     :array[0..100] of longint;
    poziadavka :string;
    a,b,x       :longint;

begin
  readln(n);
  for i:=0 to n-1 do
  begin
    readln(naradie[i]);
    readln(policka[i]);
  end;
  readln(k);
  for i:=0 to k-1 do
  begin
    readln(poziadavka);
    a:=0; b:=n;
    while b-a>1 do
    begin
      x:=(a+b) div 2;
      if naradie[x]<=poziadavka then a:=x
      else b:=x;
    end;
    if (poziadavka<>naradie[a]) or (policka[a]=-47) then
    begin
      writeln('Nedokazeme krave vyhoviet.');
      halt;
    end;
  end;
  writeln('Dokazeme krave vyhoviet.');
end.
```

Opravoval Maty
(max. 10 bodov)

2. Zákerná hra

Prvé riešenie, ktoré človeka napadne, je vyskúšať všetky možné začiatky a konce a spočítať súčet kartičiek, čím dostávame zložitosť $O(N^3)$. Za túto zložitosť ste mohli získať 5 bodov. Jednoduchou úvahou to môžeme zlepšiť na N^2 tým, že si uvedomíme, že zo súčtu postupnosti od i po j vieme zrátať súčet od i po $j+1$ iba pričítaním kartičky na mieste $j+1$ namiesto znova sčítavania kartičiek od i po $j+1$. Za takéto riešenia sa dalo získať 7 bodov. A teraz poďme ku vzorovému lineárnemu riešeniu. Najprv ku každej kartičke pričítame 1 za dĺžku postupnosti. Teraz už hľadáme len podpostupnosti s najväčším súčtom.

Predstavme si situáciu, že začneme hrať od i-tej kartičky a nech tá je kladná. V hre sa nám oplatí ostať iba kým súčet našich kartičiek nie je záporný. Nech po pričítaní j-tej kartičky prvýkrát dostaneme záporný súčet, teda aj samotná j-ta kartička je záporná. Je zjavné že ak začneme v rozmedzí od $i+1$ až po j-tu kartičku, nedostaneme lepší súčet, ako keď začneme od i-tej kartičky. A teda najbližší vhodný začiatok na hľadanie optimálnej postupnosti je $(j+1)$-vá kartička. A rovnako pokračovať v hre od i-tej kartičky so záporným súčtom sa neoplatí oproti možnosti začať znovu od $j+1$-tej karty s nulovým súčtom. Takže náš algoritmus funguje nasledovne. Začnime hrať od prvej kartičky. Berieme kartičky, kým nemáme záporný

počet bodov.[3] V momente, keď máme záporný počet bodov, si vytvoríme nový začiatok a znova ideme hľadať najlepší koniec k tomuto začiatku. Takto lineárne prejdeme kartičkami a určite niekedy nájdeme optimálny začiatok a k nemu optimálny koniec, takže stačí stále porovnávať doterajší súčet s maximom.

Listing programu:

```
var sucet,aktzac,bestzac,bestkon,max,n,i: integer;
    a : array [1..200] of integer;
begin
  Readln(n);
  for i:=1 to n do begin
    readln(a[i]);
    inc(a[i]);
  end;
  aktzac:=1;{odkial hladam najlepsiu postupnost}
  bestzac:=1;
  bestkon:=1;
  sucet:=0;
  max:=-9999;
  for i:=1 to n do begin
    sucet:=sucet+a[i];
    if (sucet>max) then begin{zapametam si aktualne najlepsiu moznost}
      bestzac:=aktzac;
      bestkon:=i;
      max:=sucet;
    end;
    if (sucet<0) then begin
      aktzac:=i+1;{zmenim aktualny zaciatok a zacnem hrat odzonva}
      sucet:=0;
    end;
  end;
  for i:=bestzac to bestkon do write(a[i]-1,' ');
  writeln;
end.
```

3. Zimomravý obdĺžnik

Opravoval Mišo
(max. 10 bodov)

Najjednoduchšie bolo zistiť, že vyskúšaním všetkých možností nájdeme určite minimálne riešenie, lebo vždy aspoň jedno existuje – napríklad s rozmermi $1 \times N$. Týmto sa dala získať lineárna zložitosť. Jedno vylepšenie je také, že skúsime iba všetky možnosti menšej strany, ktorá bude mať dĺžku menšiu ako $\sqrt{N}$, v opačnom prípade by bola dlhšia ako tá druhá strana. A hľadáme taký rozmer, ktorý je čo najbližšie k štvorcu. Toto tvrdenie sa síce zdá zjavné, ale treba ho dokázať. Avšak nestrhával som body, keď chýbal. Predstavme si, že máme dva obdĺžniky s rozmermi $a \times \frac{N}{a}$ a $b \times \frac{N}{b}$, pričom platí $a \leq \frac{N}{a}$ (a je kratšia strana obdĺžnika) a $a > b > 0$ (a je bližšie k $\sqrt{N}$)

Určite platí, že $a^2 \leq N$, lebo $a \leq \sqrt{N}$. Keďže $a > b$, tak aj $a \cdot b < N$. Z toho dostaneme, že $N - ab > 0$. Vynásobíme to $(a - b)$ a dostaneme $(N - ab)(a - b) > 0$ (mohli sme to spraviť preto, lebo $a > b$ a teda $a - b > 0$). Po roznásobení dostaneme $Na - Nb - a^2b + ab^2 > 0$.

[3] Zaujimavá situácia je, keď dostaneme súčet 0. Vtedy, ak ho zahodíme a začneme znovu, nájdeme postupnosť s najkratšou dĺžkou a naopak, ak ju necháme, tak s najväčšou dĺžkou.

Vynásobíme to $\frac{2}{ab}$ a dostaneme $2N/b - 2N/a - 2a + ab > 0$. Členy obsahujúce a prehodíme na druhú stranu a dostaneme $2(N/b + b) > 2(N/a + a)$.

Ľavá strana nerovnosti je obvod obdĺžnika zo stranami $b, N/b$ a pravá strana nerovnosti je obvod obdĺžnika zo stranami $a, N/a$. A teda čím bližšie je strana obdĺžnika k odmocnine, tým menší má obvod.

Listing programu:

```
var N,a:integer;
begin
  readln(N);
  a:=floor(sqrt(N));
  while ((a>1) and (not ((N mod a)=0))) do
    dec(a);
  writeln('Rozmery su '+a+':'+(N div a));
end.
```

Opravoval Hermi
(max. 15 bodov)

4. Zúfalý Miško

Sedliacky rozum nám napovedá, že priamočiare riešenie, ktoré vygeneruje inkriminovaný reťazec (prípadne ho bude generovať priebežne) neprinesie autorovi kýžených 15 bodov. Ako teda takúto ošemetnú záležitosť obabrať? Zjavne budeme chcieť preskakovať väčšie úseky reťazca bez toho, aby sme vedeli aké presne sú. Ak začneme nad problémom dumať, ihneď nám udrie do očí zjavná komplikácia: tie prekliate mocniny majú rôznu dĺžku! A to sa, milí priatelia, stane našou zbraňou proti systému. A tak si človek povedal: čo tak skúsiť preskakovať po úsekoch, ktoré sú tvorené číslami s rovnakým počtom cifier?

Ak zoberieme úsek radu, ktorý pozostáva iba z členov, ktoré majú počet cifier K, vieme zistiť prvý i posledný člen tohto úseku. Nech A a B sú tieto čísla a nech $A < B$, potom počet členov tohto úseku vyrátame ako $\sqrt[3]{B} - \sqrt[3]{A} + 1$. Z toho ľahko vyrátame, že tento úsek v reťazci bude zaberať $(\sqrt[3]{B} - \sqrt[3]{A} + 1)K$ číslic. Ostáva nám zistiť hodnoty A a B a matematickú časť by sme mali za sebou. Čo vieme o hodnote A? Je to najmenšie číslo ktoré je väčšie ako 10^K a pritom platí, že jeho odmocnina je celé číslo. Matematicky to vieme presne vyjadriť nasledovne $A = \lceil \sqrt[3]{10^K} \rceil^3$. Podobnou úvahou dospejeme aj k vzťahu $B = \lfloor \sqrt[3]{10^{K+1} - 1} \rfloor^3$.

Algoritmus je teda zjavný. Budeme postupne rátať aký úsek reťazca zaberajú čísla s nejakým počtom cifier, až kým sa nedostaneme do úseku, v ktorom leží hľadaná cifra a potom si vyrátame o ktorý člen v poradí sa jedná a jednoducho vypíšeme. Akú bude mať takýto algoritmus zložitosť? Zamyslime sa nad úsekmi, ktoré preskakujeme. Ich dĺžka narastá exponenciálne (ako je ľahko vidieť z predchádzajúceho odstavca). Teda zložitosť bude $O(\log N)$. Pamätáme si pritom iba konštantne veľa premenných, čiže pamäťová zložitosť je $O(1)$.

Listing programu:

```
program Zufalec;

var
  n,a,b,k,moc,cifier,kolke,ktora: integer;
  prvy: boolean;
  s: string;

function odmocni(x: integer) : double;
begin
  odmocni:=exp(ln(x)/3);
```

```
end;

function umocni(x: integer) : integer;
begin
  umocni:=x*x*x;
end;

begin
  readln(N);
  moc:=1;
  k:=0;
  cifier:=0;
  prvy:=true;
  repeat
    inc(k);
    dec(n,cifier);
    if prvy then {nepanikárte, toto je len aby sme najprv robili}
      prvy:=false
    else          {mocniny jednociferné}
      moc:=moc*10;
    b:=trunc(odmocni(moc*10-1));
    {pascal nemá hornú celú časť, fintíme}
    if abs(odmocni(moc) - trunc(odmocni(moc))) < 1e-12 then
      a:=round(odmocni(moc))
    else
      a:=trunc(odmocni(moc)+1);
    cifier:=(b-a+1)*k;
  until cifier>=n;
  kolke:=(n-1) div k; {vyrátame v koľkej mocnine je}
  ktora:=umocni(a+kolke);
  kolke:=((n-1) mod k)+1; {koľkatá cifra tejto mocniny to je a vytiahneme ju}
  str(ktora,s);
  writeln(s[kolke]);
end.
```

5. Obchodné orákulum

Opravoval András
(max. 15 bodov)

Najskôr si musíme uvedomiť, že sa nám neoplatí peniaze deliť na menšie kôpky, z ktorých niektoré pôjdu do jedného produktu a iné pôjdu do iného, ktorý má v tom istom čase platnosť, ale vkladať a vyberať všetky peniaze naraz. Dôvod je jednoduchý. Ak je v nejakom produkte úrok u, tak ak do neho vložíme x peňazí, tak po jeho skončení budeme mať $x(1+u)$ peňazí. Ak teraz vložíme peniaze do ďalšieho produktu s úrokom v, tak budeme mať $x(1+u)(1+v)$ peňazí. Ak vložíme peniaze postupne do produktov s úrokmi $u_1, u_2, \ldots, u_k$, tak budeme mať $x(1+u_1)(1+u_2)\cdots(1+u_k)$ peňazí. Ak $p_1, p_2, \ldots, p_k$ je postupnosť časovo sa neprekrývajúcich bankových produktov s úrokmi $u_1, u_2, \ldots, u_k$, tak číslu $(1+u_1)(1+u_2)\cdots(1+u_k)$ budeme hovoriť výnos postupnosti bankových produktov. Zoberme takú postupnosť bankových produktov, že má najväčší výnos a označme ju P. Ak zoberieme ľubovoľnú inú postupnosť, tak ona má menší výnos. Ak peniaze rozdelíme a každý kus vložíme do nejakej postupností bankových produktov, tak sa výnos bude prinajlepšom taký veľký, ako výnos postupnosti P. Preto odteraz budeme hľadať postupnosť P.

Postupnosť bankových produktov je taká postupnosť postupnosť bankových produktov, ktoré sa časovo neprekrývajú a súčin ich úrokov[4] je najvyšší. Do nich vložíme všetky naše peniaze.

Na nájdenie takej postupnosti použijeme techniku, ktorá sa nazýva dynamické programovanie. Nech $P(x)$ je najväčší zisk, aký vieme mať v deň x. Ako vypočítame $P(x)$? Predstavme si, že pre každý deň, ktorý je pred x máme vypočítané, aký najväčší zisk vieme dosiahnuť, ak skončíme s burzou v ten deň (poznáme hodnoty $P(1), P(2), \ldots, P(x-1)$). Nech $d_1, \ldots, d_k$ sú tie produkty, ktoré končia v deň x. Nech $z_1, \ldots, z_k$ sú dni, kedy začínajú dané produkty a $u_1, \ldots, u_k$ sú prislúchajúce úroky. Potom $P(x)$ je maximum z nasledovných čísiel: $P(x-1), P(z_1) \cdot u_1, P(z_2) \cdot u_2, \cdots, P(z_k) \cdot u_k$. Inými slovami, pre každý produkt, ktorý končí v daný deň sa pozrieme, koľko vieme dostať, ak ho použijeme a vyberieme z nich maximum. Za $P(x)$ potom vyberieme buď $P(x-1)$ (v x nebudeme používať žiadny produkt), alebo naše nájdené maximum (podľa toho, ktoré je väčšie).

Potrebujeme však vypočítať $P(1), P(2), \cdots, P(x-1)$. Ako? To je jednoduché. Vieme, že $P(1) = 1$. $P(2)$ vypočítame podľa hore uvedeného postupu. $P(3)$ vypočítame tak isto. Takto vieme vypočítať $P(1), P(2), \cdots P(x-1)$. Keďže deň D je deň kedy chceme skončiť, tak keď vypočítame $P(D)$, tak máme výsledok najväčší výnos, aký vieme dosiahnuť.

Prečo to funguje? Dá sa to jednoducho dokázať matematickou[5] indukciou od počtu dní (premenná x). Ak $x = 1$, tak to funguje, lebo $P(1) = 1$. Predpokladajme teraz, že v $P(1), P(2), \ldots, P(x-1)$ máme uložené optimálne hodnoty. Ako to bude v $P(x)$? Môžu nastať dva prípady. Buď v daný deň sa nám neskončí produkt a vtedy $P(x) = P(x-1)$, alebo sa nám skončí nejaký produkt p zo začiatkom v dni z a úrokom u. Avšak my zo všetkých končiacich produktov zoberieme ten najvýhodnejší, a ten nutne musí byť produkt p.

Teraz sa vrhnime na implementáciu. Aby sme nemuseli každý deň otestovať všetky produkty, na začiatku programu priraďme každý produkt ku dňu v ktorom končí. Tým dosiahneme,že nemusíme vyhľadávať produkty končiace v daný deň, čím si výrazne zlepšíme časovú zložitosť (bez tohto kroku by bola zložitosť až $O(ND)$.

Časová zložitosť: $O(N + D)$ Pre každý deň prejdeme všetky produkty, ktoré v daný deň končia. Preto prejdeme všetky dni a všetky produkty raz, lebo môžeme predpokladať, že každý úrok skončí v našom intervale a ani jeden úrok sa nemôže skončiť dvakrát, takže každý prejdeme presne raz. Lepší algoritmus už neexistuje, lebo každý produkt a každý deň musíme aspoň raz prejsť, aby sme nestratili možnosti a my sme prešli každý produkt a každý deň raz, takže každú možnosť raz.

Pamäťová zložitosť: $O(N + D)$

Listing programu:

```
program Project1;

uses
  SysUtils;

Type TUrok = record
  a,b,u : integer;
  end;

Type TDen = record
  uroky : array of TUrok;
  aktualne_peniaze : real;
  end;
```

[4] Presnejšie súčin 1+úrok.

[5] Pozor! Nie elektromagnetickou!

```
Var
  N,P,D : longint;
  i,o : integer;
  kalendar : array of TDen;
  urok1 : TUrok;

procedure vloz(u1 : TUrok);
begin
  setlength(kalendar[u1.b].uroky,length(kalendar[u1.b].uroky)+1);
  kalendar[u1.b].uroky[length(kalendar[u1.b].uroky)-1] := u1;
end;

begin
  read(N,P,D);
  { ini }
  setlength(kalendar,D+1);
  for i := 0 to D do
    begin
      kalendar[i].aktualne_peniaze := p;
      setlength(kalendar[i].uroky,0);
    end;

  for i := 0 to N-1 do
    begin
      read(urok1.a,urok1.b,urok1.u);
      vloz(urok1);
    end;

  for i := 1 to D do
    begin
      kalendar[i].aktualne_peniaze := kalendar[i-1].aktualne_peniaze;
      for o := 0 to length(kalendar[i].uroky)-1 do
        begin
          urok1 := kalendar[i].uroky[o];
          if kalendar[urok1.a].aktualne_peniaze*(100+urok1.u)/100 >
kalendar[i].aktualne_peniaze then
            kalendar[i].aktualne_peniaze :=
kalendar[urok1.a].aktualne_peniaze*(100+urok1.u)/100;
        end;
    end;


  writeln(kalendar[D].aktualne_peniaze:0:2);
  readln;
end.
```

6. On snáď nepozná žiadne zábrany!

Opravoval Ivan
(max. 20 bodov)

Keď sa v úlohe začne spomínať náhoda, niektorí prejdu na nasledovnú úlohu, zaradujú sa, že to bude ľahké, zamyslia nad otázkou náhody, vesmíru a všetkého, iných zase chytí migréna, no všetci títo ľudia majú spoločné to, že nik z nich poriadne nevie, čo je to náhoda.[6] Aby

[6] Je toto číslo náhodné: 7?

situácia nebola taká zúfalá, v tomto príklade sme si presne definovali, čo budeme považovať za náhodnú postupnosť a pridali si prostriedky na jej docielenie – funkciu $random(n)$.

Prvú vec, ktorú si treba uvedomiť, je, že rastúce postupnosti dĺžky k z n prvkov zodpovedajú výberom k prvkov z n prvkovej množiny. Teraz si už vieme ľahko aj vyjadriť ich počet $\binom{n}{k} = \frac{n\cdot(n-1)\cdot\ \cdots\ \cdot(n-k+1)}{1\cdot 2\cdot\ \cdots\ \cdot k} = \frac{n!}{k!\cdot(n-k)!}$. Z čoho už vidno, že algoritmus vyber si náhodné prvé číslo, potom druhé väčšie ako prvé ... asi nebude to čo chceme (nieže bol nedostatok odvážlivcov, ktorý sa túto cestu pokúsili zdolať). Totiž počet hľadaných postupností začínajúcich číslom i je $\binom{n-i}{k}$, z čoho keď si vyjadríme rozvrhnutie pravdepodobností jednotlivých začiatočných čísel, dostaneme zvrhlý rad podielov kombinačných čísel, ktorým sa radšej vyhneme. Podobná ale predsa trochu iná možnosť je spočítať si počet možných postupností, z nich si vybrať náhodnú a vypísať ktorá to je, ale je to tiež cesta obrastená veľkým kombinačným číslím.

Pozrime sa na problém z iného uhla. Ak je naša postupnosť výber k prvkov z n, prečo by sme proste tých k prvkov nevyberali po jednom? Ak každý prvok vyberáme s rovnakou pravdepodobnosťou, nezávisle na už vybraných prvkoch, aj každá kombinácia bude vybraná s rovnakou pravdepodobnosťou. Relatívne časté riešenie bolo vytvoriť si n prvkové pole čísel $0, 1, \ldots, n-1$ a k-krát vybrať náhodný prvok, na jeho miesto dať posledný a pokračovať s o 1 menším poľom. Zväčša boli prvky pamätané buď v boolovskom poli, cez ktoré sa pri výpise prešlo, čo na celkovú zložitosť $O(n)$ nevplývalo, alebo v nevyužitej časti poľa, kde sa následne museli zotriediť, zväčšujúc časovú zložitosť na $O(n+k\log k)$. Možno sa nezdá, ale z druhého prípadu vieme dostať lepšie riešenie, lebo sa môžeme zbaviť n. Zabijeme dve muchy jedným úderom, keď použijeme binárny prehľadávací strom, v ktorom si budeme pamatať vybrané prvky a k nim ešte ich náhradu, ak sú dosť malé, aby mohli byť vybrané znovu. Čísla vyberieme tak, že k-krát vyberieme náhodný prvok z n, pridáme ho do stromu (resp. jeho náhradu), pridelíme mu (novú) náhradu $n-1$ a pokračujeme s menším n. Teda zbavili sme sa poľa a prvky stačí vypísať v stromu prírodnom rastúcom poradí. Máme teda jedno z možných riešení za 20 bodov, ktoré má časovú zložitosť $O(k\log k)$ a pamäťovú $O(n)$.

Ešte iný pohľad na vec dostaneme, keď si všimneme, že sa vieme relatívne jednoducho rozhodnúť, či postupnosť začať s najmenším možným (0). Totiž hľadaných postupností, ktoré začínajú s 0 je $\binom{n-1}{k-1}$ čo z celkového množstva postupností je $\frac{\binom{n-1}{k-1}}{\binom{n}{k}} = \frac{k}{n}$. Potom už hneď vidíme, že sme si problém zredukovali na podobný, ale menší: z $n-1$ čísel vybrať k (resp. $k-1$). Teda algoritmus, ktorí odovzdalo hodne riešiteľov je: postupne pre čísla $0, 1, \ldots, n-1$ vypíš dané číslo s pravdepodobnosťou $\frac{k}{n}$, zmenši n, príp. aj k. Keďže takýmto spôsobom korektne rozdeľujeme všetky možné postupnosti na 2 disjunktné[7] množiny a s pravdepodobnosťou úmernou ich veľkostiam sa rozhodujeme, z ktorej si budeme ďalej vyberať, všetky postupnosti majú rovnakú pravdepodobnosť byť vybrané. Máme teda ešte jedno 20 bodové riešenie, s časovou zložitosťou $O(n)$ a pamäťovou $O(1)$. Vo vzorovom kóde sú implementované obe, pričom sa z nich vyberá to rýchlejšie.

Ešte poznámka k riešeniam väčšine účastníkov (hlavne tých odovzdávajúcich elektronicky). Funkcia, ktorá ako vstup dostane 1 z n rôznych hodnôt s rovnakou pravdepodobnosťou ($\frac{1}{n}$) a vráti 1 z m s pravdepodobnosťou $\frac{1}{m}$ na svete neexistuje a teda nemôže to byť ani $f(x) = x \bmod m$. Jedno možné riešenie je prerozdeliť toľko vstupných hodnôt, koľko sa len dá, a pre ostatné skúsiť znovu s iným vstupom (pozri kód). Iné je tým sa nebaviť a použiť funkciu random, ktorú podľa zadania máme k dispozícii. Pri hodnotení riešení som tento nedostatok nebral do ohľadu.

Listing programu:

```
#include <iostream>
```

[7] neobsahujúce spoločné prvky

```
#include <cstdlib>
#include <map>
#include <cmath>

using namespace std;

/* ak niekto ma potrebu implementovat funkciu random pomocou funkcie
 * generujucej "nahodne" cisla 0 .. RAND_MAX, toto je jeden sposob
 * ako to urobit */
int random(int n) {
    int r;
    do r = rand();
    while (r >= RAND_MAX - RAND_MAX % n);
    return r % n;
}

void out(int m) {
    cout << m << ' ';
}

void sol_n(int n, int k) {
    for (int i = 0; k > 0; ++i) {
        if (random(n) < k) {
            out(i);
            --k;
        }
        --n;
    }
}

int get(const map<int, int> & m, int p) {
    map<int,int>::const_iterator i = m.find(p);
    if (i != m.end())
        return i->second;
    return p;
}

// map<int, int> je binrny strom vybrany prvok + jeho nahrada
void sol_k_log_k(int n, int k) {
    map<int, int> m;
    for (int i = 0; i < k; ++i) {
        int r = random(n - i);
        m[get(m, r)] = 42;          // vyznacime vybrany prvok
        m[r] = get(m, n - i - 1);   // na jeho miesto dame posledny
    }
    for (map<int, int>::iterator i = m.begin(); i != m.end(); ++i)
        out(i->first);
}

int main() {
    int n, k;
    cin >> n >> k;
    if (n < k * log(k))     // obe riesenia su spravne,
        sol_n(n, k);        // vyberieme si asympototicky rychlejsie
    else
        sol_k_log_k(n, k);
    cout << endl;
    return 0;
}
```

7. O holubej pošte

Opravoval Dano
(max. 20 bodov)

Riešenia tohto príkladu ma veľmi nepotešili. Všetky totiž boli nesprávne. A to vrátane vedúcovských, čo sa prejavilo na rekordne malom priemernom počte bodov získaných za príklad. Spravil som si rýchlo prehľad príkladov za posledných približne päťnásť ročníkov a vyšlo mi, že priemer 0.75 bodu je ďaleko najmenší. Pre porovnanie, druhý najmenší priemer je 2.87, ktorý sa vysykytol v treťom kole 15. ročníka. Ďalší, a o mnoho dôležitejši následok, je, že jediné funkčné riešenie, ktoré poznáme je backtrack. Ponaučenie pre vás je, že vždy sa oplatí poslať aspoň obyčajný backtrack. V tomto prípade by ste zaň dokonca dostali aj plný počet bodov, ak by ste ho správne napísali.

Naspäť ale k príkladu. Na prvý pohľad sa zdá správna myšlienka, že musí existovať aj také optimálne riešenie, v ktorom mestá v jednej skupine tvoria súvislý úsek a žiadne skupiny sa neprekrývajú. Každý riešiteľ toto buď v tichosti predpokladal, alebo povedal, že to tak je, ale že sa mu to nepodarilo dokázať. To ale nie je žiadne prekvapenie, keďže toto tvrdenie neplatí. Zoberme si napríklad takúto situáciu. Máme 13 miest, ktoré chceme rozdeliť do dvoch skupín. Jedno mesto je na pozícií 0, tri mestá su na pozícií 5 a zvyšných deväť miest na pozícií 10. V tomto prípade je najlepšie mať tri mestá na pozícií 5 v jednej skupine a zvyšok v druhej skupine. Môžete síce namietať, že zo zadania viac-menej vyplýva, že mestá nemôžu byť na tom istom mieste, ale keďže vždy môžeme všetky mestá o malý kúsok poposúvať, toto naozaj problém nie je.

Čo sa týka bodovania príkladu, bolo nasledovné. Ak by ste poslali (skoro) hocičo fuknčné, dostali by ste aspoň 10 bodov. Za úplne obyčajný backtrack by bolo 15 a za o trochu lepší až 20. Tých, ktorí poslali riešenie založené na zlej domnienke, som bodoval nasledovne. Ak poslali riešenie využívajúce dynamické programovanie bez žiadných fínt, dostali 2 body, ak mali aj nejaké finty vylepšujúcu zložitosť, dostali po 3 bodoch. Tí, ktorí poslali hocičo backtrackujúce, dostali najviac jeden bod. A ostatné riešenia, ktoré využívali rôzne, nie vždy fungujúce, heuristiky nedostali bohužiaľ nič. Plus po bode som väčšinou strhával ak chýbal kód, odhad zložitosti alebo sa vyskytli dáke menšie chyby.

Teraz ale k riešeniu. Ako som už povedal, jedine nám známe riešenie je backtrack. Inými slovami, použijeme riešenie, ktoré vyskúša všetky možnosti a z nich vybere tú najlepšiu. Zjavne, každé mesto môže byť v jednej z K skupín a teda celkovo máme až K^N možných rozdelení. Hneď na úvod, jedno možné zlepšenie je, že nemusíme skúšať úplne všetkých K^N možností, ale, že veľa „rôznych“ rozdelení nám dá ten istý výsledok. Napríklad pre $N = K = 2$ je úplne jedno, či máme prvé mesto v prvej skupine a druhé mesto v druhej skupine, alebo máme prvé mesto v druhej skupine a druhé mesto v prvej skupine. Aby sme teda nemuseli uvažovať niektoré rozdelienia viackrát, použijeme nasledovnú fintu. Prvé mesto vždy priradíme do prvej skupiny a nikdy nepriradíme žiadne mesto do i-tej skupiny ak tá a $i-1$-vá sú obe prázdne. Týmto docielime, že každé rozdelenie vygenerujeme len, v istom zmysle slova, v základnom/minimálnom tvare.

Ďalej, uvažujme jednu množinu miest o veľkosti M. Úplne základným spôsobom, ak sčítame všetky vzdialenosti medzi všetkými dvojicami, na vyrátanie priemernej vzdialenosti v skupine potrebujeme až rádovo M^2 operácií. Teraz si ukážeme, ako si vieme postupne aktualizovať priemernú vzdialenosť v konštantnom čase, keď pridávame nové prvky. V podstate jediná vec, ktorú si potrebujeme aktualizovať, je súčet vzdialeností medzi všetkými dvojicami v skupine. Na začiatok si utriedime všetky mestá a potom ich budeme postupne spracovávať zľava doprava. Čo znamená, že vždy keď pridáme nové mesto do skupiny, všetky mestá v skupine sú už naľavo od pridaného mesta. Pre každú skupinu si budeme pamätať jej veľkosť V, súčet vzdialenosti medzi mestami S, pozíciu najpravejšieho mesta P a súčet vzdialenosti D medzi najpravejším mestom a zvyšnými mestami. Potom, po pridaní nového prvku na

pozícii x, vieme ľahko vypočítať novú hodnotu V a P. Taktiež, $D' = D + (x - P) \cdot M$ a $S' = S + D'$, kde D', S' sú nové hodnoty D a S, keďže $P \leq x$.

Takto si vieme postupne porátať súčet vzdialeností, z ktorých už ľahko dostaneme priemernú vzdialenosť. Po vygenerovaní celého rozdelenia nám už len stačí nájsť najväčšiu priemernú vzdialenosť, aktualizovať najlepšiu nájdenú hodnotu doteraz a vrátiť sa v rekurzii. Vybratie najväčšej priemernej vzdialenosti trvá $O(K)$, alebo, ak použijeme nejakú lepšiu dátovu štruktúru podporujúcu operácie insert, delete a get_min v logaritmickom čase, $O(\log K)$. Teda, celková zložitosť je $O(R \log K + N \log N)$, kde R je celkový počet rozdelení v základnom tvare. Potrebujeme si pamätať všetky pozície a, navyše, pre každú skupinu si potrebujeme pamätať konštantne veľa pamäte. Preto, pamäťová zložitosť je $O(N + K)$.

Listing programu:

```
#include <cstdio>
#include <set>
#include <algorithm>

using namespace std;

const int maxN=1000, maxK=1000;
int n,k;
int a[maxN], v[maxK], s[maxK], p[maxK], d[maxK];

double ans=999999999;
multiset<double> pv;

double getAD(int sum, int n){
  if(n<=1) return 0.0;
  return (2*sum)/(double)(n*(n-1));
}

void f(int i){
  if(i==n) ans=min(ans,*(--pv.end()));
  else {
    int x=a[i];
    for(int j=0; j<k; j++){
      int oldS=s[j];
      int oldP=p[j];
      int oldD=d[j];
      if(v[j]!=0) pv.erase(pv.find(getAD(oldS,v[j])));
      p[j]=x;
      d[j]+=(x-oldP)*v[j];
      s[j]+=d[j];
      v[j]++;
      pv.insert(getAD(s[j],v[j]));
      f(i+1);
      pv.erase(pv.find(getAD(s[j],v[j])));
      v[j]--;
      s[j]=oldS;
      d[j]=oldD;
      p[j]=oldP;
      if(v[j]!=0) pv.insert(getAD(oldS,v[j]));
      if(v[j]==0) break;
    }
  }
}

int main(){
  scanf("%d%d",&n,&k);
```

```
  for(int i=0;i<n;i++)scanf("%d",&a[i]);
  sort(a,a+n);
  for(int i=0;i<k;i++)v[i]=s[i]=p[i]=d[i]=0;
  v[0]=1; p[0]=a[0]; pv.insert(0.0);
  f(1);
  printf("%lf\n",ans);
  return 0;
}
```

Opravoval Zemčo
(max. 25 bodov)

8. Ohrozené druhy

Tento príklad nebol taký jednoduchý, napriek tomu prišlo celkom dosť riešení. Niektoré boli úspešné, niektoré neúspešné. V živote je to tak, že história si zapamätá len tie úspešné riešenia. V tomto vzoráku sa, podobne ako v živote, tiež budeme točiť len okolo úspešných riešení. Na druhú stranu, výsledková listina si zapamätá všetkých.

Kto sa dočítal až sem, sa teraz dozvie, že za úspešné riešenia budeme považovať tie, ktoré bežia v linérnom čase s lineárnou pamäťou od veľkosti grafu. Tieto riešenia dostali plný počet bodov. Prišli aj nápadité riešenia s časovou zložitosťou $O(N \log N)$, ktoré získali 21 bodov. Za kvadratický čas sa udeľovalo 18 bodov a za kubický 15. Za absenciu kódu sa strhala polovica bodov a za chyby v programoch niečo medzi 1 bodom a polovicou.

Ako sa mal teda problém Hermiho a jeho hotela úspešne riešiť? V prípade, že je graf nesúvislý, je odpoveď nekonečno, pretože existuje vrchol, z ktorého neexistuje cesta k hotelu a preto naň môžeme umiestniť ľubovoľný počet zvieratiek. Najjednoduchší spôsob, ako toto overiť, je skontrolovať hodnotu K. Predpokladáme, že graf je acyklický, a preto môže obsahovať najviac $N-1$ hrán. Ak ich obsahuje ostro menej ako $N-1$, potom je nutne nesúvislý[8]. No a ten ostatný prípad, kde je graf súvislý, si rozoberieme v nasledujúcich odstavcoch.

Úvodom si uveďme niekoľko pozorovaní, na ktorých urobíme niekoľko všimnutí. Úplne prvé bude, že zvieratká sa neoplatí umiestňovať na vrcholy, ktoré nie sú listy. Ak vrchol nie je list, potom má aspoň dvoch susedov. Práve pre jedného z nich platí, že ak sa vyberieme smerom k nemu, tak prídeme bližšie k hotelu. Toto je pravda, pretože v strome je medzi každými dvoma vrcholmi práve jedna cesta. Preto, ak pôjdeme hociktorou inou vetvou, tak na ostrove vedľa môžeme umiestniť dvojnásobný počet zvieratiek ako v pôvodnom vrchole s rovnakým efektom: zvieratká sa požerú a vyberú sa smerom k hotelu. Jediný prípad, kedy nemáme kam zvieratká posunúť je list: má len jedného suseda, ktorým sa k hotelu priblíži (samozrejme to neplatí v prípade, keď sme na hotelovom ostrove).

Teraz uvažujme ľubovoľný list vo vzdialenosti x od miesta, kde má Hermi hotel. Zjavne, ak umiestníme do tohto vrcholu 2^x zvieratiek, tak nám jedno z nich bude schopné prejsť až k hotelu. Takže na tomto ostrove musí byť bezpodmienečne najviac $2^x - 1$ zvieratiek. Ak by sa tieto zvieratká začali navzájom žrať a presúvať sa k hotelu, potom platí, že sa k nemu nikdy nedostanú, pretože ich je málo. V prípade, že sa k nim ale cestou nejaké zvieratko pridá, tak potom bude mať Hermi problém. V tomto vzoráku budeme idealisticky predpokladať, že nechceme, aby mal Hermi problém.

Uvažujme situáciu, keď sme na ostrove a jedným smerom sa môžeme priblížiť k hotelu. Pri vzďaľovaní sa od hotela máme dve možnosti, pričom každá z nich sa sa už ďalej nevetví. Jedna pokračuje smerom k listu ešte v mostami, druhá u mostami a nech $u \leq v$. Navyše, nech môže na ostrov, na ktorom stojíme, prísť najviac k zvieratiek (toto obmedzenie vyplýva zo schopnosti $k+1$ zvieratiek prísť až k hotelu z danej križovatky). Kebyže príde všetkých k zvieratiek z vetvy dlhej v, tak na konci tejto vetvy môže byť najviac $2^v \cdot k + 2^v - 1$. Z každého

[8] Pre poriadok v terminológii: súvislý acyklický graf sa nazýva strom, nesúvislý sa nazýva les.

2^v zvieratka v tomto liste príde totiž jedno na našu križovatku, druhá časť výrazu sú tie, ktoré tam môžu byť navyše a na križovatku sa nedostanú. To, že z druhej vetvy má prísť nula neznamená, že na jej konci má byť nula zvieratiek – môžeme tam nejaké dať, ale najviac $2^u - 1$. Pre celkový počet vypustených zvierat dostávame $2^v \cdot k + 2^v + 2^u - 2$. V prípade, že $i, 0 < i \leq k$ zvierat príde z vetvy, ktorá pokračuje ešte u vetvami, potom dokážeme s odbodbným odôvodnením umiestniť do celej uvažovanej sústavy $(k-i) \cdot 2^v + 2^v + 2^u + i \cdot 2^u - 2$. Keď si dáme uvedené počty do nerovnosti a odškrtneme rovnaké veci, dostávame: $2^v \cdot k \geq (k-i) \cdot 2^v + i \cdot 2^u$. Nerovnosť platí, keďže $u \leq v$. Rovnosť nastáva pre každé i, ak $u = v$. Takže vieme, že najlepšie spravíme, ak všetky zvieratká prídu z hlbšej vetvy. Dozvedeli sme sa ale aj, že ak dve vetvy pokračujú rovnako ďaleko, tak nám nezáleží, z ktorej prídu zvieratká – budeme ich môcť rozmiestniť rovnako veľa.

Uvedený argument sa dá zovšeobecniť aj na prípad, že sa nám súostrovie vetví na viac ako dve vetvy. Pre tých, ktorí si potrebujú niečo dokazovať, odkazujem, že si to môžete (podobným spôsobom) formálne dokázať na domácu úlohu. My už budeme vedieť, že najvýhodnejšie je, aby nám zvieratká prišli z najdlhšej vzdialenosti. Aby sme dokončilu teoretickú prípravu, potrebujeme ešte ukázať, čo robiť vo všeobecnej situácii. Uvažujme ostrov, na ktorý môže prísť k zvieratiek a vzdialiť sa od hotela môžeme ľubovoľným počtom vetiev a tie sa ďalej môžu všelijako vetviť. Opäť sa dá ukázať, že sa najviac oplatí, ak všetky zvieratká prídu z najhlbšej vetvy – z najvzdialenejšieho lista od tohto ostrova. Po ceste sa k nim nemôžu nijaké pridať, pretože sa to neoplatí. Keďže uvažujeme najhlbší list, tak potom by zvieratko, ktoré by sa pridalo k putujúcim niekam z boku znamenalo menej zvieratiek v nejakom plytšom liste, ako keby toto zvieratko prišlo z toho najhlbšieho.

Táto teoretická príprava nám teraz poslúži ako základ algoritmu. Ten bude založený na prehľadávaní do hĺbky a odkazovať sa na hlbšie ostrovy (vzdialenejšie od hotela). Vo všeobecnosti sa pri stromoch veľmi často používajú upravené prehľadávania do hĺbky. Strom si zakoreníme na ostrove, kde je hotel. Pre každý ostrov platí, že jednou vetvou sa dá dostať k hotelu a tými ostatnými niekam hlbšie.

Na úvod sa zíde poznať pre každý ostrov, ktorá je to tá najhlbšia vetva. Toto spravíme jednoduchým prehľadávaním, ktoré sa bude volať po ostrovoch a vracať, ako hlboko jednotlivými cestami môžeme ešte ísť. V prípade listu je to jednoduché – vrátime nulu, pretože z listu sa už nikam hlbšie nedostaneme. Ak máme na ostrove ešte nejaké iné mosty okrem toho, ktorým sme prišli od hotela, tak sa zavoláme na ne a zistíme si, ako hlboko môžeme ďalej tadeto putovať. Keď dostaneme informácie o každej vetve, tak len vrátime maximum týchto hĺbok zväčšené o jedna.

Následne skonštruujeme rekurzívnu funkciu, ktorá nám bude hovoriť pri zavolaní sa na ostrov, koľko zvieratiek môžeme umiestniť do podstromu, ktorého koreň je daný ostrov. Jej parameter bude číslo k, ktoré značí, koľko najviac zvieratiek sa môže dostať na daný ostrov. V prípade, že sa nachádzame na liste, je to jednoduché – vrátime hodnotu k. V ostatnom prípade sa zavoláme na najhlbšiu vetvu s parametrom $2k + 1$, pretože ak z tejto vetvy prídu všetky zvieratká, tak toto je maximum, ktoré sa smie na najbližší vrchol v tejto vetve dostať. Pre všetky ostatné vetvy sa zavoláme s parametrom 1, pretože vieme, že z tohto ostrova nesmú prísť žiadne zvieratká, ktoré by sa *zrazili* s tými z najhlbšej vetvy. Výsledok nie je nič iné ako hodnota funkcie zavolanej na ostrove s hotelom pre $k = 0$, pretože na tomto ostrove nechceme žiadne zvieratko. Fajnšmekri prišli na to, že postup sa dá implementovať aj pomocou jedného prehľadávania. My sme ale dali prednosť dvom jednoduchším a zrozumiteľnejším.

Listing programu:

```
program kiribati;
const MAXN = 100000;
var G: array[1..MAXN]of array of longint;
    najhlbsia,deg,vstup1,vstup2: array[1..MAXN]of longint;
```

```
    N,i,K: longint;

{prehladavanie do hlbky na zistenie najhlbsich vetiev}
function najhlbsie_vetvy(v, odkial: longint):longint;
var aktualna,max,i: longint;
begin
    max := -1; najhlbsia[v] := -1;
    for i:=1 to deg[v] do if (odkial <> G[v][i])then
    begin
       {zavolame sa nizsie a zistime, ako hlboko sa vieme tade dostat}
       aktualna := najhlbsie_vetvy(G[v][i],v);
       if (aktualna > max) then begin max := aktualna; najhlbsia[v] := i; end;
    end;
    najhlbsie_vetvy := max+1;
end;

{prehladavanie do hlbky, ktore vrati, kolko zvierat vieme umiestnit
do podstromu tak, aby sa do vrcholu 'v' dokazalo dostat najviac 'kolko' zvierat}
function pocet_zvierat(v, odkial, kolko: longint):longint;
var i: longint;
begin
    {sme v liste. umiestnieme sem 'kolko' zvierat a sme hotovi}
    if (deg[v] = 1) then pocet_zvierat := kolko
    else pocet_zvierat := 0;
    for i:=1 to deg[v] do if (G[v][i] <> odkial) then
    begin
        {zavolame sa hlbsie. do najhlbsiej vetvy s inymi parametrami ako do ostatnych}
if ( najhlbsia[v] <> i )then pocet_zvierat := pocet_zvierat+pocet_zvierat(G[v][i],v,1)
else pocet_zvierat := pocet_zvierat+pocet_zvierat(G[v][i],v,kolko*2+1);
    end;
end;

begin
    readln(N,K);
    for i:=1 to N do deg[i] := 0;
    {cely cirkus okolo vstupu robime preto, lebo chceme zavolat metodu resize len raz
pre
   kazdy vrchol. ak by sme ju volali pri kazdom rozsireni pola o jedno cislo, tak by sme
mali
   kvadraticky cas. preto najprv zistime stupne a potom si do pola G nacitame graf.}
    for i:=1 to K do
    begin
       readln(vstup1[i],vstup2[i]);
       inc(deg[vstup1[i]]); inc(deg[vstup2[i]]);
    end;
    {je graf suvisly?}
    if (K<N-1)then
    begin
       writeln('Hermi moze nasadit nekonecno zvierat!');
       halt;
    end;
    for i:=1 to N do
    begin
      setlength(G[i],deg[i]+1);
      deg[i] := 0;
    end;
    {nacitame si do G graf}
    for i:=1 to K do
    begin
inc(deg[vstup1[i]]); inc(deg[vstup2[i]]);
```

```
G[vstup1[i]][deg[vstup1[i]]]:=vstup2[i];
G[vstup2[i]][deg[vstup2[i]]]:=vstup1[i];
    end;
    najhlbsie_vetvy(1,-1);
    writeln('Hermi moze nasadit ',pocet_zvierat(1,-1,0),' zvierat.');
end.
```

Opravoval Mic
(max. 25 bodov)

9. Túžba po rovnováhe

Tento príklad vôbec nebol ťažký a preto ma nepotešil počet riešení. Bolo ich iba 5 :-(. Ako sa mal teda tento príklad riešiť? Najskôr si ho preformulujeme do reči grafov. Máme graf G a našou úlohou je ohodnotiť hrany grafu číslami $-1, 1$ tak, aby súčet hrán pri každom vrchole bol 0 (čo je práve vtedy, keď je počet hrán s číslom -1 rovnaký ako počet hrán s číslom 1).

Teraz si spravíme malú odbočku a povieme si, čo je to *Eulerovský cyklus*. Nech G je graf a $S = u_1u_2 \dots u_k$ je postupnosť vrcholov grafu G (vrcholy sa v nej môžu opakovať). S je *Eulerovský cyklus* práve vtedy, keď každý vrchol z G je v S, každá hrana z G je práve raz v S a $u_1 = u_k$. Inými slovami, graf G obsahuje *Eulerovský cyklus* práve vtedy, keď sa dá nakresliť jedným ťahom tak, že každú hranu nakreslím len raz a skončím v tom vrchole, v ktorom som začal. Dané nakreslenie mi tvorí *Eulerov cyklus*.

Dôležité: Od teraz až do odvolania budeme predpokladať, že graf G je súvislý (z každého vrchola existuje nejaká cesta do každého vrchola).

Ak graf G má nepárny počet hrán, alebo nejaký vrchol má nepárny *stupeň* (stupeň vrchola je počet susedov vrchola), tak vyvážené vzťahy nedostaneme. Preto budeme predpokladať, že v grafe G má každý vrchol párny stupeň. Ak v takomto grafe existuje *Eulerovský cyklys*, tak môžeme ísť po obvode cyklu a budeme ohodnocovať hrany na striedačku -1 a 1, tak každý vrchol bude mať súčet ohodnotení svojich hrán 0. Preto, ak existuje v grafe *Eulerovský cyklus*, tak ho stačí nájsť a potom sme za vodou.

Čo ak sa náhodou taký cyklus v grafe nenachádza? To by mohol byť problém. Ale niet sa čoho báť, lebo platí tvrdenie, že v súvislom grafe existuje *Eulerovský cyklus* práve vtedy, keď má v ňom každý vrchol párny stupeň (Všimnite si, že tento cyklus môže mať aj nepárny počet hrán). A teda *Eulerovský cyklus* v našom grafe existuje vždy[9]. Teraz si ideme dokázať toto tvrdenie a pritom aj vymyslíme algoritmus, ktorým daný cyklus nájdeme.

Presnejšie, ukážeme si tvrdenie, že ak má každý vrchol grafu párny stupeň, tak graf obsahuje *Eulerovský cyklus*. Opačné tvrdenie je oveľa jednoduchšie a skúste si ho sami dokázať.

Začnime prehľadávať graf G do hĺbky tak, že povolíme prechádzať cez každý vrchol viac krát, ale cez každú hranu povolíme prejsť iba raz. Vždy, keď pri prehľadávaní prejdeme cez nejakú hranu, tak tú hranu z grafu odstránime (alebo si poznačíme, že je odstránená). Vždy, keď prídeme do nejakého vrchola, tak sa mu zníži stupeň o 1 a vždy keď z neho opäť vyjdeme, tak sa mu zase zníži stupeň o 1. To znamená, že za každý prechod vrcholom sa mu zníži stupeň o 2. Časom dorazíme do vrchola, z ktorého sa už nebude viesť žiadna cesta. Bude to práve ten vrchol, z ktorého sme začali prehľadávať. Prečo? Každý vrchol mal na začiatku párny stupeň. Z prvého vrchola sme vyšli a teda sme mu znížili stupeň o 1. Potom každému vrcholu cez ktorý sme šli sme znížili stupeň o 2 (alebo o väčšie párne číslo, ak sme šli cez neho viac krát). Preto ak by sme neskončili v počiatočnom vrchole, tak by sa stalo to, že by stupeň vrchola, v ktorom práve sme, bol nepárny a teda by sme ešte niekam vedeli ísť.

Keď už nevieme prehľadávať ďalej, tak sa budeme vracať, až kým sa nevrátime do vrchola, z ktorého vieme pokračovať v prehľadávaní (má ešte nejaké nenavštívené hrany). Pri vracaní

[9] Nezabudnite, že teraz predpokladáme súvislé grafy s párnymi stupňami vrcholov

si budeme zapisovať hrany, po ktorých vraciame. Čo sa stane, ak nájdeme vrchol, z ktorého vieme pokračovať ďalej? Nech je to vrchol u. Budeme z vrchola u ďalej prehľadávať, až kým sa zase nezasekneme. Ale to musí byť opäť vo vrchole u (lebo každý stupeň vrchola je párny). Začneme sa vracať po prejdených hranách a zapisujeme si vrcholy, cez ktoré sa vraciame. Takto si zapíšeme vrcholy na ceste z u do u. Teda stále máme zapísané vrcholy, ktoré sú na nejakej ceste. Ak budeme takto pokračovať ďalej, tak po skončení budeme mať zapísaný celý *Eulerovský cyklus*.

Dôležité: Odteraz už nepredpokladáme, že graf G je súvislý, ani to, že každý vrchol má párny stupeň!

Ako bude vyzerať výsledný algoritmus? Načítame graf a skontrolujeme, či má každý vrchol párny stupeň. Ak áno, tak pre každý komponent súvislosti grafu nájdeme *Eulerovský cyklus* po jeho obvode nastavíme hrany na striedačku na 1 a -1. Zároveň pre každý komponent súvislosti zistíme, či má párny počet hrán (lebo ak nie, tak rovnováhu nevieme dostať). Pri správnej implementácii bude mať výsledný algoritmus časovú aj pamäťovú zložitosť $O(N + M)$.

Ako to vhodne implementovať? Pre každú hranu si pamätáme, či sme v nej boli a pre každý vrchol si pamätáme zoznam susedných hrán a index do zoznamu, aby sme vedeli, ktoré hrany sme už prechádzali (V jednom vrchole môžeme byť viackrát a nechceme vždy prejsť celý zoznam hrán).

Ako som bodoval? Za optimálne riešenie som dal 25 bodov, za kvadratické som dal najviac 18 bodov a za nefunkčné riešenie som dal najviac 3 body.

Poznámky ku kódu: Pole E je zoznam hrán, v $G[v]$ je zoznam dvojíc (u, e), pričom u susedí s v a sú spojené hranou e. V poli I sú indexy do zoznamov hrán, aby sme neprechádzali viackrát tie isté hrany. Do poľa *path* sa ukladá výsledný *Eulerovský cyklus*. Graf nerozkladáme na komponenty, ale zavoláme prehľadávanie na každý vrchol raz, lebo ak sa zavoláme druhýkrát na ten istý komponent, tak sa už nič nevykoná a teda nám to časovú zložitosť nezhorší. Teraz si už s môžete prečítať zdrojový kód.

Listing programu:

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

vector<pair<int,int> > E;
vector<bool> was;
vector<vector<pair<int,int> > > G;
vector<unsigned int> I;
vector<int> path;


void dfs(int v){
    unsigned int &i=I[v];
    for(;i<G[v].size();i++){
        int ii=i;
        if(!was[G[v][ii].second]){
            was[G[v][ii].second]=true;
            dfs(G[v][ii].first);
            path.push_back(G[v][ii].second);
        }
    }
}

int main(){
    int N,M;
```

```
    cin>>N>>M;
    G.resize(N);
    for(int i=0;i<M;i++){
        int a,b,ind;
        cin>>a>>b;
        a--;b--;
        ind=E.size();
        E.push_back(make_pair(a,b));
        G[a].push_back(make_pair(b,ind));
        G[b].push_back(make_pair(a,ind));
    }
    was.resize(E.size(),false);
    for(int i=0;i<N;i++)
        if(G[i].size()%2==1){
            cout<<"Neda sa to:-("<<endl;
            return 0;
        }
    I.resize(N,0);
    int prevsize=path.size();
    for(int i=0;i<N;i++){
        prevsize=path.size();
        dfs(i);
        if((path.size()-prevsize%2)!=0){
            cout<<"Neda sa to:-("<<endl; // Komponent ma neparny pocet hran
            return 0;
        }
    }
    for(unsigned int i=0;i<path.size();i++){
        cout<<E[path[i]].first+1<<" "<<E[path[i]].second+1<<" - ";
        if(i%2)
            cout<<"pozitivny"<<endl;
        else
            cout<<"negativny"<<endl;
    }
}
```

Opravoval Lukáš
(max. 25 bodov)

10. Trasa okolo rovníka

Vyrobme si zo vstupu graf. Vrcholy budú Hermiho kamaráti a hrany budú medzi kamarátmi, ktorých spája letecká linka. Zároveň si budeme pre hranu pamätať, o aký uhol sa vďaka nej posunieme. Ak letíme zo západu na východ, bude číslo kladné, inak záporné. Ak cesta obletí Zem, tak súčet uhlov letov zo západu na východ sa nemôže rovnať súčtu uhlov letov z východu na západ. V našom grafe to bude znamenať, že súčet uhlov na hranách cesty sa nerovná nule. Ak ďalej v texte budeme spomínať dĺžku cesty, máme na mysli počet leteckých liniek, teda hrán na ceste. Súčet uhlov na ceste budeme nazývať uhlový súčet cesty.

Pre každý vrchol si budeme pamätať dve cesty z vrcholu 1 – jedna z nich bude najkratšia (ak je ich viac, zapamätáme si ľubovoľnú z nich). Druhú zvolíme tak, aby bola najkratšia spomedzi tých, ktoré majú inú uhlovú dĺžku ako najkratšia cesta. Znova, ak je takýchto ciest viac, stačí nám ľubovoľná z nich. Odteraz ak v texte spomenieme najkratšie cesty do vrcholu, budeme ich chápať v takomto význame.

Ak budeme pre každý vrchol a špeciálne pre vrchol 1 poznať dve najkratšie cesty, ľahko vyriešime úlohu. Najkratšia cesta do vrcholu 1 má dĺžku nula a uhlový súčet tiež nula. Druhá cesta, ktorú si zapamätáme vo vrchole 1, bude najkratšia spomedzi tých, ktoré majú

iný uhlový súčet ako 0. A to je presne to, čo hľadáme – najkratšia okružná cesta. Zostáva nám však vyriešiť, ako tieto dĺžky ciest zistiť.

Vzorové riešenie bude trochu pozmenené prehľadávanie do šírky. V prehľadávaní do šírky budeme každý vrchol spracúvať najviac dvakrát. Vždy raz po tom, čo nájdeme doňho jednu z dvoch najkratších ciest. Pri spracúvaní vrcholu v a jednej z jeho najkratších ciest $v_1, v_2, \ldots, v_k, v$ pozrieme všetky hrany vychádzajúce z v a pokúsime sa aktualizovať zoznam najkratších ciest pre susedné vrcholy. To znamená, že ak susedný vrchol u ešte nemá nájdené 2 najkratšie cesty, tak mu do zoznamu pridáme cestu $v_1, v_2, \ldots, v_k, v, u$. Jediný prípad, kedy sa nám to nemusí podariť, je, ak u už má nájdenú najkratšiu cestu a tá má rovnakú uhlovú dĺžku ako cesta $v_1, v_2, \ldots, v_k, v, u$.

Máme teda načrtnutý algoritmus, poďme sa zamyslieť nad zložitosťou. Vo vrcholoch si nemusíme pamätať celé najkratšie cesty, ale iba ich dĺžky a uhlové súčty. Keďže si musíme pamätať aj hrany, tak pamäťová zložitosť je $O(N + M)$. Každý vrchol a hranu spracúvame v prehľadávaní do šírky najviac dvakrát, preto je časová zložitosť $O(N + M)$.

Listing programu:

```
#include <iostream>
#include <vector>
#include <queue>
#include <utility>
using namespace std;

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

vector<int> poz;
typedef pair<int, int> PI;
int vzd(int a, int b)
{
    int y = poz[b] - poz[a];
    while (y > 180) y -= 360;
    while (y < -180) y += 360;
    return y;
}
int main() {
    int n, m; scanf("%d %d", &n, &m);
    poz.assign(n, 0);
    REP(i,n) scanf("%d", &poz[i]);

    vector<vector<PI> > h(n);
    REP(i,m)
    {
        int a, b;
        scanf("%d %d", &a, &b);
        a--; b--;
        h[a].push_back(PI(b, vzd(a, b)));
        h[b].push_back(PI(a, vzd(b, a)));
    }

    vector<vector<int> > vz(n);
    vector<vector<int> > poc(n);
    vz[0] = vector<int>(1, 0);
    poc[0] = vector<int>(1, 0);
    int kol = -1;
    queue<int> f;

    for (f.push(0); !f.empty(); f.pop())
    {
```

```
        int i = f.front();
        int q;
        REP(j,h[i].size())
        {
            q = h[i][j].first;
            REP(k,vz[i].size())
            {
                int vzdialenost = vz[i][k] + h[i][j].second;
                if (vz[q].size() == 0 || (vz[q].size() == 1 && vz[q][0] !=
vzdialenost))
                {
                    vz[q].push_back(vzdialenost);
                    poc[q].push_back(poc[i][k] + 1);
                    f.push(q);
                }
            }
        }

        if (vz[0].size() > 1)
        {
            kol = poc[0][1];
            break;
        }
    }

    printf("%d\n", kol);
}
```

Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
1	Baláž Martin	Gym. Alejová Košice	4	49	10	10	10	15	15	109
2	Kekely Michal	Gym. Varšavská Žilina - Vlčince	2	45	10	7	10	15	9	96
3	Sládek Filip	Gym. Námestovo	3	39	10	10	10	14	11	94
4	Balog Matej	Gym. Grösslingová BA	2	50	6	10	7	9	11	93
5	Hozza Ján	Gym. Jura Hronca BA	2	56	10	10	10			86
6	Hruška Eugen	Gym. Hlohovec	3	42	5	10	7	14	6	84
6	Korbaš Rafael	Gym. Hronská BA	2	39	10	10	10	10	5	84
8	Varga Mátyás	Gym. H. Selyeho Komárno	2	33	8	10	10	10	11	82
9	Phuong Mariana	Gym. Jura Hronca BA	2	27	9	10	10	10	12	78
10	Bubnár Michal	Gym. sv. Františka z Assisi V.n. Topľou	4	35	5	7	10	10	10	77
11	Sebechlebský Ján	Gym. Žiar nad Hronom	2	35	10	7	10	10	3	75
12	Kieferová Mária	Gym. sv. Františka Žilina	4	37	9	8	10		10	74
13	Jankovič Radovan	Gymnázium Levice	2	21	7	9	9	9	4	59
14	Kunec Sebastián	Gym. Konštantínova Prešov	4	33	5	4	6	10		58
15	Cuc Bruno	Gym. Grösslingová BA	3	24	8	7	9	9		57
16	Jurových Jakub	Gym. Okružná Zvolen	2	31	5	6	4	9		55
16	Kučera Martin	Gym. Golianova Nitra	2	20	8	4	10	10	3	55
18	Dresslerova Anna	Gym. Jura Hronca BA	2	28		7	10	8		53
18	Kuzma Tomáš	Gym. Alejová Košice	4	38				15		53
18	Majdiš Mojmír	Gym. Dolný Kubín	3	24	9	10	10			53
21	Mariš Andrej	Gym. Piaristická Nitra	1	16	10	6	9	10	1	52
22	Páleník Martin	Gymnázium Levice	4	16	10	4	9	10	1	50
23	Plank Martin	Gym. A. Merici Trnava	4	16	10	5	9	8		48
24	Macháč Matej	Gym. P. de Coubertina Piešťany	3	14	8	7	7	10		46
25	Anderle Michal	Gym. Haličská Lučenec	2	18	4	7	7	8		44
26	Bosák Radomír	Gym. Grösslingová BA	4	41						41
27	Vavrík Boris	Gym. Jura Hronca BA	2	39						39
27	Vecerik Matej	Gymnázium	2	0	8	7	10	10	4	39
29	Kuchyňár Martin	Gym. Jura Hronca BA	3	36						36
30	Mrocková Mária	Gym. Jura Hronca BA	2	0	9	6	10	10		35
31	Machac Juraj	Gym. Jura Hronca BA	2	25	3	6				34
32	Šmihla Ján	ZŠ Saratovská	0	33						33
33	Anderko Maroš	Gym. Konštantínova Prešov	3	4	4	6	8	7		29
34	Pokorný Fridolín	Gym. Haličská Lučenec	4	0	3	6	4	7	7	27
34	Sabo Matus	Gym. Jura Hronca BA	3	0	4	6	4	7	6	27
36	Fedáková Kristína	Gym. Stará Ľubovňa	1	0	9		9	8		26
36	Mudrik Richard	Gym. J. M. Hurbana Čadca	4	7	5		6	8		26
38	Toman Viktor	Gym. Golianova Nitra	2	0	5	4	6	10		25
39	Špano Marek	Gym. Jura Hronca BA	2	24						24
40	Szabó Peter	SPŠE Nové Zámky	4	14				9		23
41	Táňa Tóthová	Gym. Jura Hronca BA	3	22						22
42	Hozzánková Hana	Gym. Jura Hronca BA	3	20						20
42	Macko Lukáš	Gym. Š. Moyzesa B. Bystrica	3	20						20
44	Iring Andrej	Gym. Jura Hronca BA	2	9			10			19
44	Stančiaková Katarína	Gym. Jura Hronca BA	3	19						19
46	Šimko Stanislav	Gym. Jura Hronca BA	3	18						18
47	Orenič Jozef	Gym. Slovenská Bardejov	3	17						17
47	Porubsky Martin	Gym. sv. Cyrila a Metoda Nitra	2	6	10		1			17
47	Štajer Andrej	Gym. Dolný Kubín	3	10			7			17
50	Bögi Tamás	SPŠ Komárno	2	16						16

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
50	Pinter Martin	Gym. Jura Hronca BA	2	16						16
52	Tulala Peter	Gym. A. Merici Trnava	3	14						14
52	Šrank Marek	Gym. A. Merici Trnava	4	0			6	8		14
54	Cudrák Miloš	Gym. J. M. Hurbana Čadca	4	5	4	2	2			13
55	Plavák Dušan	Gym. Trstená	2	11						11
56	Chomjak Maros	Gym. Alejová Košice	3	10						10
56	Novella Tomáš	Gym. Alejová Košice	3	10						10
56	Piovarči Michal	Gym. Hronská BA	4	10						10
59	Dzurilla Bohus	Gym. Alejová Košice	3	9						9
59	Kentoš Michal	Gym. sv. Františka z Assisi V.n. Topľou	4	9						9
59	Kovaľ Anton	Gym. L. Stöckela Bardejov	3	9						9
59	Pivovarči Michal	Gym. Hronská BA	4	9						9
59	Toth Robert	Gym. Alejová Košice	3	9						9
59	Uchnár Matúš	Gym. Alejová Košice	3	9						9
59	Zatrochová Zuzana	Gym. Alejová Košice	3	9						9
59	Ďurikovičová Lucia	Gym. sv. Uršule BA	2	0			9			9
67	Holas Juraj	Gym. Jura Hronca BA	2	8						8
67	Nagy Štefan	SPŠ Komárno	3	8						8
69	Kmec Peter	Gym. Konštantínova Prešov	2	5						5
70	Aštary Matej	Gym. Konštantínova Prešov	3	4						4
70	Fotta Michal	Gym. Konštantínova Prešov	3	4						4
70	Nguyen Tien Phong	Gym. Konštantínova Prešov	3	4						4
70	Trebichavský Richard	Gym. Jura Hronca BA	1	4						4
74	— Anonym	Gym. Jura Hronca BA	?	0						0
74	Bogárová Zuzana	Gym. Ľudovíta Štúra Trenčín	2	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 strojnícka Spišská Nová Ves	4	87	15	15	20	3	25	165
2	Šrámek Martin	Gym. Alejová Košice	4	88	15	12	17		25	157
3	Petrucha Michal	Gym. Metodova BA	4	76	15	15	19		25	150
4	Herencsár Albert	Gym. maď. Galanta	4	80	15	9	17	1	21	143
5	Hudec Tobiáš	Gym. Partizánske	3	79	12	15	6	1	25	138
6	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	4	77	10	14	15	0	15	131
7	Baláž Martin	Gym. Alejová Košice	4	87	15	15			12	129
8	Csiba Peter	Škola pre mim. nadané deti BA	4	61	15	15	0	0	21	112
9	Špánik Marián	Gym. sv. Františka Žilina	4	56	10	3	15		25	109
10	Rohár Pavol	Gym. M. R. Štefánika, Košice	3	40	15	12	12		25	104
11	Kuzma Tomáš	Gym. Alejová Košice	4	83	15					98
12	Ziman Michal	Gym. Haličská Lučenec	3	31	10	8	12		23	84
13	Hruška Eugen	Gym. Hlohovec	3	52	14	6	11	0		83
14	Sládek Filip	Gym. Námestovo	3	33	14	11	17	1	4	80
15	Hojčka Michal	Gym. Partizánske	4	33	15	1	12		3	64
16	Belan Tomáš	Škola pre mim. nadané deti BA	3	58						58
17	Bubnár Michal	Gym. sv. Františka z Assisi V.n. Topľou	4	21	10	10	16			57
17	Proksa Ondrej	Gym. Párovská Nitra	4	57						57
19	Jankovič Radovan	Gymnázium Levice	2	22	9	4	2	0	17	54
20	Kunec Sebastián	Gym. Konštantínova Prešov	4	21	10		2		17	50
21	Miklovič Tomáš	Gym. Nové Zámky	3	24	10				15	49
22	Kekely Michal	Gym. Varšavská Žilina - Vlčince	2	23	15	9				47
23	Jursa Jakub	Gym. Alejová Košice	4	25	10		9			44
23	Phuong Mariana	Gym. Jura Hronca BA	2	12	10	12	10			44
25	Habovštiak Martin	Gym. Tvrdošín	3	0	10		8		25	43
26	Balog Matej	Gym. Grösslingová BA	2	22	9	11				42
27	Hozza Ján	Gym. Jura Hronca BA	2	26			15			41
28	Korbaš Rafael	Gym. Hronská BA	2	23	10	5				38
29	Špano Marek	Gym. Jura Hronca BA	2	34						34
30	Kieferová Mária	Gym. sv. Františka Žilina	4	22		10				32
30	Varga Mátyás	Gym. H. Selyeho Komárno	2	11	10	11				32
32	Mariš Andrej	Gym. Piaristická Nitra	1	4	10	1	14			29
33	Kučera Martin	Gym. Golianova Nitra	2	0	10	3	14			27
34	Rigdová Emília	Gym. Popradské nábr. Poprad	3	0	10		16			26
34	Sebechlebský Ján	Gym. Žiar nad Hronom	2	13	10	3				26
36	Hagara Michal	Gym. Jura Hronca BA	3	25						25
37	Machac Juraj	Gym. Jura Hronca BA	2	24						24
37	Macháč Matej	Gym. P. de Coubertina Piešťany	3	14	10					24
39	Iring Andrej	Gym. Jura Hronca BA	2	21						21
39	Kikta Michal	Gym. Tatarku Poprad	4	21						21
39	Vavrík Boris	Gym. Jura Hronca BA	2	21						21
42	Anderle Michal	Gym. Haličská Lučenec	2	9	8		3			20
42	Dresslerova Anna	Gym. Jura Hronca BA	2	12	8					20
42	Hozzánková Hana	Gym. Jura Hronca BA	3	20						20
42	Jurových Jakub	Gym. Okružná Zvolen	2	11	9					20
46	Cuc Bruno	Gym. Grösslingová BA	3	7	9					16
46	Szabó Peter	SPŠE Nové Zámky	4	7	9					16
48	Stančiaková Katarína	Gym. Jura Hronca BA	3	15						15
49	Pokorný Fridolín	Gym. Haličská Lučenec	4	0	7	7				14
49	Vecerik Matej	Gymnázium	2	0	10	4				14

	Meno a priezvisko	Škola	Trieda		4	5	6	7	8	Σ
51	Sabo Matus	Gym. Jura Hronca BA	3	0	7	6				13
52	Piovarči Michal	Gym. Hronská BA	4	12						12
52	Táňa Tóthová	Gym. Jura Hronca BA	3	12						12
54	Anderko Maroš	Gym. Konštantínova Prešov	3	4	7					11
54	Bosák Radomír	Gym. Grösslingová BA	4	11						11
54	Páleník Martin	Gymnázium Levice	4	0	10	1				11
54	Šmihla Ján	ZŠ Saratovská	0	11						11
58	Kuchyňár Martin	Gym. Jura Hronca BA	3	10						10
58	Mrocková Mária	Gym. Jura Hronca BA	2	0	10					10
58	Toman Viktor	Gym. Golianova Nitra	2	0	10					10
61	Šimko Stanislav	Gym. Jura Hronca BA	3	9						9
62	Fedáková Kristína	Gym. Stará Ľubovňa	1	0	8					8
62	Mudrik Richard	Gym. J. M. Hurbana Čadca	4	0	8					8
62	Plank Martin	Gym. A. Merici Trnava	4	0	8					8
62	Šrank Marek	Gym. A. Merici Trnava	4	0	8					8
66	Tulala Peter	Gym. A. Merici Trnava	3	6						6
67	Orenič Jozef	Gym. Slovenská Bardejov	3	5						5
68	Aštary Matej	Gym. Konštantínova Prešov	3	4						4
68	Fotta Michal	Gym. Konštantínova Prešov	3	4						4
68	Nguyen Tien Phong	Gym. Konštantínova Prešov	3	4						4
71	Plavák Dušan	Gym. Trstená	2	2						2

Výsledková listina po 2. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda		8	9	10	Σ
1	Fulla Peter	SPŠ strojnícka Spišská Nová Ves	4	70	25	25	25	145
2	Hudec Tobiáš	Gym. Partizánske	3	52	25			77
3	Jankovič Radovan	Gymnázium Levice	2	12	17	15	15	59
4	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	4	17	15	18	3	53
5	Petrucha Michal	Gym. Metodova BA	4	25	25			50
5	Šrámek Martin	Gym. Alejová Košice	4	25	25			50
7	Špánik Marián	Gym. sv. Františka Žilina	4	16	25			41
8	Herencsár Albert	Gym. maď. Galanta	4	17	21			38
9	Csiba Peter	Škola pre mim. nadané deti BA	4	16	21			37
10	Baláž Martin	Gym. Alejová Košice	4	23	12			35
11	Rohár Pavol	Gym. M. R. Štefánika, Košice	3	5	25			30
12	Miklovič Tomáš	Gym. Nové Zámky	3	14	15			29
13	Habovštiak Martin	Gym. Tvrdošín	3	0	25			25
14	Kuzma Tomáš	Gym. Alejová Košice	4	24				24
15	Ziman Michal	Gym. Haličská Lučenec	3	0	23			23
16	Kunec Sebastián	Gym. Konštantínova Prešov	4	0	17			17
16	Proksa Ondrej	Gym. Párovská Nitra	4	17				17
18	Belan Tomáš	Škola pre mim. nadané deti BA	3	16				16
19	Iring Andrej	Gym. Jura Hronca BA	2	14				14
19	Sládek Filip	Gym. Námestovo	3	9	4	1		14
21	Hojčka Michal	Gym. Partizánske	4	7	3			10
22	Hruška Eugen	Gym. Hlohovec	3	8			0	8
23	Machac Juraj	Gym. Jura Hronca BA	2	6				6
23	Táňa Tóthová	Gym. Jura Hronca BA	3	6				6
25	Piovarči Michal	Gym. Hronská BA	4	4				4
25	Stančiaková Katarína	Gym. Jura Hronca BA	3	4				4
27	Anderle Michal	Gym. Haličská Lučenec	2	0		3		3
27	Hozzánková Hana	Gym. Jura Hronca BA	3	3				3
29	Anderko Maroš	Gym. Konštantínova Prešov	3	0				0
29	Aštary Matej	Gym. Konštantínova Prešov	3	0				0
29	Fotta Michal	Gym. Konštantínova Prešov	3	0				0
29	Kmec Peter	Gym. Konštantínova Prešov	2	0				0
29	Nguyen Tien Phong	Gym. Konštantínova Prešov	3	0				0