Korešpondenčný seminár z programovania
XXI. ročník, 2003/04
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 1. kola zimnej časti

Milí riešitelia!
Trvalo dlho, ale dočkali ste sa. Vytúžené vzorové riešenia sú už vo vašich rukách a nik vám ich nevezme. Tentokrát vás čakajú až dve riešenia, ktoré by si zaslúžili viac ako plný počet bodov, takže sa máte na čo tešiť. A samozejme na vás čaká výsledková listina po prvom kole. Ale najskôr prečítať naše riešenia!

Čítaj tieto odkazy, môžu sa ti hodiť na sústredení.

KSPáci

1. O slovných rovniciach

opravoval Dávidko
(max. aj 18 bodov)

Treba si uvedomiť, že dĺžka slovnej premennej môže byť rádovo až taká dlhá ako vstup. Napríklad rovnica tvaru $a \dots aX \dots X = X \dots X$, kde naľavo je k áčok a toľko isto iksiek, a napravo $k+1$ iksiek, má riešenie k áčok. Vstup aj riešenie sú teda veľkosti $O(k)$. Po dosadení za všetky X do pôvodnej dôjde ku kvadratickému nafúknutiu. Konkrétne budú mať dĺžky ľavej a pravej strany až $k(k+1) > k^2$ písmen. Na to väčšina ľudí prišla, ale napriek tomu naprogramovala kvadratické riešenie. Nad lineárnym riešením neuvažoval nikto.

Mimochodom kvadratický alebo dokonca kubický algoritmus sa dá dostať aj omnoho trápnejšie: Stačí zle používať stringy. Napríklad pascalovské zreťazenie reťazcov `u:=u+v` trvá až čas úmerný dĺžke u a v. Toto som ticho toleroval, ak v bolo krátke alebo sa to robilo len málo krát v programe,[1] lebo je možné to ľahko naprogramovať, tak aby to bolo úmerné len dĺžke v, ale treba ako vedieť fungujú stringy v pascale vo vnútri. Tým, ale nevyriešime to, ak v je dlhé. V jazyku C zase funkcia `strlen` vracajúca dĺžku reťazca pracuje lineárne od počtu jeho písmen (narozdiel od pascalovského `length`). Ak sa jedno alebo druhé použije blbo, dostaneme o rád horší algoritmus ako sme zamýšlali. V C++ funguje oboje rýchlo aj keď nie nutne v konštatnom čase. Problematika stringov je stará informatická disciplína, ale rozhodne nie ľahká. Viac sa o nej dozviete napríklad pri čítaní tohto vzoráku a vzoráku štvrtého príkladu.

Keďže vzorové lineárne riešenie nikto nemal, kvadratické riešenia s lineárnou pamäťou dostali plný počet 15 bodov. Za kvadratický čas aj pamäť som dával 12 bodov, za kubické a horšie maximálne 9 bodov. Za chýbajúci alebo nedostatočný popis, odhad časovej a pamäťovej zložitosti alebo chyby v programoch jeden až dva bodíky dole.

Vzorové riešenie

Slovnú premennú označme X a jeho i-te písmenko zľava X_i, teda $X = X_1X_2 \dots X_d$, kde d je jej (zatiaľ neznáma) dĺžka. Slovo na ľavej strane označme $\ell = \ell_1\ell_2 \dots \ell_n$, pričom ale X chápeme len ako jeden jeho znak. Podobne nech $r = r_1r_2 \dots r_m$ je slovo na pravej strane. Ich dĺžky sú n a m. Ďalej označme počet normálnych písmeniek (t.j. rôznych od X) na ľavej strane p_ℓ a na pravej p_r. Počet výskytov slovnej premennej na ľavej strane f_ℓ a na pravej f_r. Ak sa majú slová na ľavej a pravej strane po nahradení premennej za písmenká rovnať, tak

[1] Málo, krátke = konštatné. Veľa, dlhé = lineárne od veľkosti vstupu.

sa musia rovnať aj ich dĺžky, teda

$$p_\ell + f_\ell d = p_r + f_r d.$$

Odtiaľ vyrátame dĺžku premennej $d = (p_\ell - p_r)/(f_r - f_\ell)$. Ak po tomto výpočte nevýjde d celé nezáporné číslo, tak riešenie slovnej rovnice zjavne neexistuje a končíme. Tu za povšimnutie stojí, že $d \leq |p_\ell - p_r| \leq n + m$ a teda riešenie rovnice nebude väčšie ako vstup. (Nechutný prípad $d = 0$ sa ľahko ošetrí zvlášť. Odteraz ďalej predpokladajme, že d je kladné.)

Postupujme zľava doprava naraz po písmenkách oboch strán, teda uvážme písmenká ℓ_i a r_i postupne pre i od 1 po $\min(n, m)$. Potom sa určite nájde taká pozícia v slovách, že práve jedno z písmen ℓ_i a r_i bude X. Toto smelé tvrdenie vyplýva z toho, že $x_\ell \neq x_r$.[2] Nech teda k je prvá taká pozícia. Bez ujmy na všeobecnosti nech $l_i = X$ a $r_i \neq X$.

V tejto chvíli, už vieme vypočítať celé slovo X a to takto: V ľahšom prípade, ak žiadne z r_k až r_{k+d-1} nie je X, tak jednoducho $X = r_k r_{k+1} \ldots r_{k+d-1}$. Inak, nájdime najbližší výskyt X napravo do aktuálnej pozície k na pravej strane rovnice. Formálne, nech teda $r_j = X$, $j > k$ a j je najmenšie také možné. Nazvime časť pravej strany od r_k po r_{j-1} w, to bude *perióda* slova X. Potom X je len niekoľkrát za sebou napísané slovo w, pričom poslednýkrát tam nemusí byť w celé.[3]

Teraz to už láka len dosadiť a overiť rovnosť. Cha, to je ale to, čo by z dosiaľ lineárneho algoritmu spravilo kvadratický, takže treba na to ísť veľmi sofistikovane. Predstavme si, že máme tabuľku t_i pre $0 \leq i < d$, kde t_i hovorí, či X a X posunuté o i sa zhodujú v prekrývajúcich sa $d - i$ znakoch.

Vyzbrojení tabuľkou ľahko spravíme skúšku správnosti. Tvárime sa, že porovnávame slová po dosadení za X, ale v skutočnosti pracujeme s pôvodnými slovami l, r. Udržiavame si dva indexy i, j ukazujúce do pôvodných slov l, r. Ak písmenko pod indexom i na ľavej strane je X, tak si ešte pamätáme index i', na ktorej pozícii v rámci X sme. Rovnako j' na pravej strane.

Ak sú pod oboma indexami i, j normálne písmená, tak ich skrátka porovnáme. Ak je jedno z nich X a druhé normálne písmenko, tak ho porovnávame s tým v rámci X. Najkomplikovanejší prípad nastáva, keď sú obe X. (Ľahko sa uvedomí, že aspoň jedno z i', j' je jedna.) Nech absolútna hodnota rozdielu i', j' je p. Ak je t_p pravdivé, tak príslušné kúsky X-ov sa zhodujú a môžeme sa ponunúť na aspôn jednej strane rovnice za písmeno X. Ak kdekoľvek narazíme na nezhodu končíme.

Ostáva porátať tabuľku t. Na to budeme potrebovať najprv porátať tabuľku p_i pre $1 \leq i \leq d$, ktorú budeme volať *prefixová funkcia slova* X.[4] Najprv pár označení: Slovo u voláme *prefixom* (resp. *sufixom*) nejakého slova w, ak existuje slovo v, také že $w = uv$ (resp. $w = vu$). Slovo w má $|w|+1$ prefixov/sufixov, kde $|w|$ značí dĺžku slova w. *Vlastným* prefixom/sufixom slova w voláme všetky prefixy/sufixy okrem w samotného. Slovo W_i ($0 \leq i \leq d$) nech je prefixom dĺžky i slova X, teda $W_i = X_1 X_2 \ldots X_i$. Prefixovú funkciu pre X definujeme takto:

$$p_i = \max\{k \mid 0 \leq k < i, W_k \text{ je sufix } W_i\}, \qquad 1 \leq i \leq d.$$

Teda W_{p_i} je najdlhším vlastným sufixom W_i spomedzi prefixov X (resp. W_i).[5] Ale ľahko sa uvedomí, že $W_{p_{p_i}}$[6] je druhým najdlhším vlastným sufixom W_i spomedzi prefixov X, atď. Tabuľka p_i sa ráta takto: p_1 je podľa definície 0. Majme vyrátanú tabuľku pre všetky $j < i$. Ak $X_{p_{i-1}+1} = X_{i+1}$, tak $p_i = p_{i-1} + 1$. Ak nie tak, možno $X_{p_{p_{i-1}}+1} = X_{i+1}$ a vtedy

[2] Tu je potrebné, aby $d > 0$.

[3] Nemusí to byť pochopiteľne najmenšia perióda.

[4] Pre zasvätených: p_i je tabuľka z Knuth-Morris-Prattovho algoritmu.

[5] Mimochodom, celé sa to robiť aj opačne: budeme rátať prefixy sufixov. Príslušná funkcia sa potom volá sufixová.

[6] $p_i < i$. Preto $i > p_i > p_{p_i} > p_{p_{p_i}} > \ldots$

$p_i = p_{p_{i-1}} + 1$ atď. Ak toto neplatí pre žiadny poschodový index, tak $p_i = 0$. Detaily a dôkaz linearity sa dočítate vo vzoráku k štvrtému príkladu.

V tabuľke t nám ide o prefixy X, ktoré sú zároveň jeho sufixami. Pravdivostná tabuľka t_i je ľahko vyrátateľná z tabuľky p:

$$t_i = W_{d-i} \text{ je sufix } X, \qquad 0 \leq i < d.$$

No ale prefixy slova X, ktoré sú zároveň sufixami vieme zaradom vymenovať. Sú to $X = W_d, W_{p_d}, W_{p_{p_d}}, \ldots$, až kým poschodový index nebude 0. Takže $t_0 = t_{d-d}, t_{d-p_d}, t_{d-p_{p_d}}, \ldots$ budú pravdivé. Ostatné t_i budú nepravda.

Celý algoritmus je časovo aj pamäťovo lineárny od n, m, d a keďže $d = O(n+m)$, tak aj od veľkosti vstupu.

Listing programu:

```
Program O_slovnych_rovniciach;

const MAX = 200;

var l,r,s:string[MAX];
    n,m,d,pl,pr,fl,fr,q:integer;
    i,j,k,ii,jj:integer;
    p:array[1..MAX] of integer; {prefixova funkcia}
    t:array[0..MAX-1] of boolean; {self-match table}

procedure die;
begin
     writeln('Rovnica nema riesenie.');
     halt;
end;

{ i-ty znak neznamej X }
function x(i:integer):char;
begin
     x:=r[k+(i-1) mod q];
end;

begin
     assign(input,'prikl1.in');
     reset(input);

     readln(l); readln(r);
     n:=length(l);  m:=length(r);
     pl:=0;          pr:=0;
     fl:=0;          fr:=0;
     for i:=1 to n do if l[i]='X' then inc(fl) else inc(pl);
     for i:=1 to m do if r[i]='X' then inc(fr) else inc(pr);

     { vyratame dlzku a overime ci je cele nezaporne cislo }
     d:=(pl-pr) div (fr-fl);
     if (d<0) or (pl+d*fl<>pr+d*fr) then die;

     { specialny pripad d=0 }
     if d=0 then
     begin
          i:=1; j:=1;
          while (i<=m) and (j<=n) do
          begin
               if l[i]='X' then inc(i)
```

```
            else
            if r[j]='X' then inc(j)
            else
            if l[i]<>r[j] then die;
            inc(i);
            inc(j);
        end;
        writeln('X = „prazdny retazec“');
        halt;
    end;

    { prva pozicia, kde je prave jedna premenna X }
    i:=1;
    repeat
         if (l[i]='X') xor (r[i]='X') then break;
         inc(i);
    until false;
    k:=i;

    { pre jednoduchost dalsieho programu vymenime l a r, tak aby l[k]=X }
    if (l[k]<>'X') then
    begin
        s:=l; l:=r; r:=s;
        j:=n; n:=m; m:=j;
    end;

    { najdeme najblizsie X v r od pozicie k }
    j:=k+1;
    while (j-k<d) and (r[j]<>'X') do inc(j);
    q:=j-k;

    { prefixova funkcia, rovnaka ako v KMP }
    p[1]:=0;
    j:=0;
    for i:=2 to d do
    begin
        while (j>0) and (x(j+1)<>x(i)) do j:=p[j];
        if x(j+1)=x(i) then inc(j);
        p[i]:=j;
    end;

    { tabulka t }
    for i:=0 to d-1 do t[i]:=false;

    j:=d;
    while j>0 do
    begin
        t[d-j]:=true;
        j:=p[j];
    end;

    { porovnanie lavej a pravej strany }
    i:=1;  j:=1;       { ukazovatele do l,r }
    ii:=1; jj:=1;      { ukazovatele do X }
    while (i<=n) and (j<=m) do
    begin
        { obe X }
        if (l[i]='X') and (r[j]='X') then
        begin
```

```
            if not t[abs(ii-jj)] then die;

            if ii=jj then
            begin
                ii:=1;
                jj:=1;
                inc(i);
                inc(j);
            end
            else if ii<jj then
            begin
                ii:=d+1+ii-jj;
                jj:=1;
                inc(j);
            end
            else
            begin
                jj:=d+1+jj-ii;
                ii:=1;
                inc(i);
            end
         end

        { jedno je X, druhe obycajne pismenko }
        else if (l[i]='X') and (r[j]<>'X') then
        begin
            if x(ii)<>r[j] then die;
            inc(ii); inc(j);
            if ii>d then
            begin
                ii:=1;
                inc(i);
            end;
        end

        { jedno je X, druhe obycajne pismenko }
        else if (l[i]<>'X') and (r[j]='X') then
        begin
            if l[i]<>x(jj) then die;
            inc(jj); inc(i);
            if jj>d then
            begin
                jj:=1;
                inc(j);
            end;
        end

        { obe su obycajne pismenka }
        else begin
            if l[i]<>r[j] then die;
            inc(i); inc(j);
        end;
    end;

    { vypiseme vysledok }
    write('X = ');
    for i:=1 to d do write(x(i));
    writeln;
end.
```

2. O trpasličom probléme

opravoval Palo
(max. 15 bodov)

Najprv niekoľko slov k hodnoteniu. Riešenia s časovou zložitosťou $O(N)$ mohli získať maximálne 15 bodov, $O(N^2)$ maximálne 11 a $O(N^3)$ maximálne 8. Body ste stratili hlavne za nedostatočný popis a za vyššie pamäťové nároky. Teraz sa už rovno môžeme prejsť k vzorovému riešeniu.

Trpasličie domčeky a cesty medzi nimi tvoria graf. V tomto grafe je N vrcholov a $N-1$ hrán a graf je súvislý, teda medzi každými dvomi vrcholmi existuje práve jedna cesta a v grafe sa nenachádza cyklus. Takýto graf nazvime strom. Najjednoduchšie riešenie spočíva v tom skúsiť dať obchod do každého vrchola a zistiť, aké znečistenie by to spôsobilo. To môžeme urobiť jedným prehľadaním grafu do hĺbky z každého vrchola. Dokopy teda máme N prehľadaní grafu, každé so zložitosťou $O(N)$, takže celková časová zložitosť tohoto algoritmu je $O(N^2)$.

My by sme ale chceli lineárne riešenie. Prvý vrchol stromu budeme volať koreň. Nech a je sused prvého vrchola. Nazvime ho synom prvého vrchola a prvý vrchol nazvime jeho rodičom. Všetkých susedov a okrem prvého vrchola nazvime synmi vrchola a. Toto môžeme postupne urobiť pre všetky vrcholy. Teda každý vrchol okrem prvého má práve jedného rodiča a okrem toho môže mať každý vrchol niekoľko synov. Pod vetvou určenou vrcholom a budeme rozumieť také vrcholy v, z ktorých sa vieme do rodiča vrcholu a dostať iba cez a. Označme takúto vetvu V_a a počet trpaslíkov žijúcich v nej ako P_a. Označme C_a celkové znečistenie ovzdušia ak sa obchod nachádza vo vrchole a. Ďalej d_{ij} bude vzdialenosť vrcholov i a j, t_i bude počet trpaslíkov žijúcich v domčeku i a P bude počet všetkých trpaslíkov.

Skúmajme ako sa líšia celkové znečistenia ak sa obchod nachádza postupne v dvoch susedných vrcholoch a a b. Rátajme rozdiel $C_b - C_a$. Bez ujmy na všeobecnosti nech a je synom b. Ak by bol obchod najprv vo vrchole b a potom by sme ho premiestnili do vrchola a, tak všetkým trpaslíkom vo V_a sa celková cesta do obchodu zmenší o d_{ab} a všetkým ostatným sa naopak o d_{ab} zväčší. Teda $C_a = C_b - d_{ab}P_a + d_{ab}(P - P_a) = C_b + d_{ab}(P - 2P_a)$. Predstavme si, že by sme už vedeli všetky P_i a celkové znečistenie, ak by bol obchod v prvom vrchole C_1. Pomocou vyššie uvedeného vzorca sme schopní vypočítať celkové znečistenie všetkým synom prvého vrchola. Ale tak isto sme schopní potom vypočítať aj celkové znečistenie, ak by bol obchod v niektorom z ich synov atď. Takže by nám stačilo raz prehľadať graf do hĺbky z prvého vrchola a vždy pred vnorením sa hlbšie do rekurzie z nejakého vrchola i vypočítame C_j pre každého syna j vrchola i. Na konci prebehneme cez všetky C_i a vypíšeme minimum z nich.

Stačí nám teda najprv zistiť C_1 a všetky P_i. Označme D_i celkové znečistenie ovzdušia, ktoré spôsobia iba trpaslíci žijúci vo vetve V_i, ak by bol obchod vo vrchole i. Keďže V_1 je celý strom, tak $C_1 = D_1$. Zoberme si ľubovolný vrchol a. Označme $b_1, b_2 \ldots b_n$ synov vrchola a. Ľahko nahliadneme, že $D_a = D_{b_1} + P_{b_1} d_{b_1 a} + D_{b_2} + P_{b_2} d_{b_2 a} + \ldots + D_{b_n} + P_{b_n} d_{b_n a}$ a $P_a = t_a + P_{b_1} + P_{b_2} + \ldots + P_{b_n}$. Takže na to, aby sme zistili P_a a D_a sa najprv rekurzívne zavolajme na synov a a potom vyššie uvedeným vzorcom vypočítajme P_a a D_a. Teda prehľadajme graf do hĺbky z prvého vrchola a uvedeným spôsobom vieme nájsť P_a a D_a pre všetky vrcholy a, a teda vieme nájsť aj $C_1 = D_1$.

Takže na vyriešenie úlohy potrebujeme dvakrát prehľadať graf do hĺbky. Každé z nich vieme spraviť v čase $O(N + M)$, teda celková časová zložitosť programu je $O(N + M)$, kde N je počet vrcholov v grafe a M je počet hrán. Vieme, že $M = N - 1$, teda sa nám oplatí si graf pamätať tak, že pre každý vrchol si budeme pamätať zoznam jeho susedov. Takže výsledná časová aj pamäťová zložitosť je $O(N + M) = O(N + N - 1) = O(N)$.

Listing programu:

```
const
    MAXN=100;
```

```
var
   a,b,c: integer;
   i,j: integer;
   n: integer;
   min: integer;
   dom: array[1..MAXN] of integer;
   vsobyv: integer;

   sus: array[1..MAXN, 1..MAXN] of integer;
   dlz: array[1..MAXN, 1..MAXN] of integer;
   pocsus: array[1..MAXN] of integer;

   bol: array[1..MAXN] of integer;
   znec: array[1..MAXN] of integer;
   obyv: array[1..MAXN] of integer;

   celznec: array[1..MAXN] of integer;

procedure zisti1(kde: integer);
var
   i, tmp1, tmp2: integer;
begin
   bol[kde] := 1;

   tmp1 := 0;
   tmp2 := 0;
   for i := 1 to pocsus[kde] do begin
         if bol[sus[kde][i]] = 0 then begin
         zisti1(sus[kde][i]);
         tmp1 := tmp1 + znec[sus[kde][i]] + obyv[sus[kde][i]] * dlz[kde][i];
         tmp2 := tmp2 + obyv[sus[kde][i]];
      end;
   end;

   znec[kde] := tmp1;
   obyv[kde] := tmp2+dom[kde];
end;

procedure ziscel(kde: integer);
var
   i: integer;
begin
   bol[kde] := 1;

   for i := 1 to pocsus[kde] do begin
     if (bol[sus[kde][i]] = 0) then begin
       celznec[sus[kde][i]] := celznec[kde]+(vsobyv-2*obyv[sus[kde][i]])*dlz[kde][i];
       ziscel(sus[kde][i]);
     end;
   end;
end;

begin
   read(n);
   vsobyv := 0;
   for i := 1 to n do begin
      read(dom[i]);
      pocsus[i] := 0;
      bol[i] := 0;
      znec[i] := 0;
```

```
        obyv[i] := 0;
        vsobyv := vsobyv + dom[i];
    end;

    for i := 1 to n-1 do begin
        read(a,b,c);
        pocsus[a]:=pocsus[a]+1;
        pocsus[b]:=pocsus[b]+1;
        sus[a][pocsus[a]]:=b;
        dlz[a][pocsus[a]]:=c;
        sus[b][pocsus[b]]:=a;
        dlz[b][pocsus[b]]:=c;
    end;

    zisti1(1);

    for i := 1 to n do begin
           bol[i] := 0;
    end;

    celznec[1] := znec[1];
    ziscel(1);

    min := 1;
    for i := 2 to n do begin
           if celznec[min] > celznec[i] then begin
           min := i;
        end;
    end;

    writeln('Obchod treba umiestnit do ', min, '. domceka.');
end.
```

3. Ohne druidov

opravoval Goober
(max. 15 bodov)

Ach jaj! Tak takýto príklad som ešte neopravoval... Nemalá množina ľudí neposlala ani náznak programu (za čo boli odmenení násobiacim koeficientom $\frac{1}{2}$), viacerí mali v riešeniach chybičky i veľké chyby, ... Hodnotenie teda vyzeralo veľmi prosto – za lineárne riešenie bolo 15 bodov, za $O(N \log N)$ bolo 12 bodov, za kvadratické 9 bodov a horšie dostali 7 bodov. Nefunkčným sa ušlo po 3 bodoch a ako vždy, nejaký ten bodík mohol pribudnúť/odbudnúť za popis/odhady/atď.

A ako sa to malo riešiť? Najprv si úlohu preložíme do trošku matematickejšej formy. Máme teda zadaných $2N$ bodov B_i (domčeky) a treba zistiť, či existuje taký bod O (druidské ohnisko), že všetky body B_i možno posúvaním po polpriamkach OB_i presunúť do vrcholov pravidelného $2N$-uholníka so stredom v O.

Keďže tento mnohouholník má párny počet vrcholov, ku každému bodu B_i existuje bod B_j taký, že polpriamky OB_i a OB_j tvoria dohromady priamku. Inak povedané, ak spojíme body B_iB_j, dostaneme priamku prechádzajúcu bodom O. Navyše, táto priamka bude mať tú vlastnosť, že v oboch ňou určených polrovinách bude rovnako veľa bodov B_i (konkrétne $N-1$). Keby sme teda k danému bodu B_i vedeli nejako nájsť zodpovedajúci bod B_j, mali by sme aj priamku, na ktorej leží bod O. A ak by sme to spravili pre dva rôzne body B_{i_1}, B_{i_2}, dostali by sme dve rôznobežné priamky, na ktorých musí ležať bod O. Inak povedané, bod O by bol ich priesečníkom. Ako však rýchlo nájsť zodpovedajúce B_j ?

Keby sme nejako vedeli určiť (bez znalosti j), na ktorej strane od hľadanej priamky B_iB_j budú jednotlivé body B_k ležať, stačilo by nájsť také B_j, aby ich na oboch stranách bolo rovnako veľa. Aby sa nám ľahšie počítalo, vyberme si bod B_i tak, že bude mať najmenšiu x-ovú súradnicu (t.j. žiadny bod nebude naľavo od neho) a usporiadajme si ostatné body B_k podľa uhla, ktorý zviera polpriamka B_iB_k s osou y. V takomto usporiadaní nám bude platiť, že ak je bod B_c medzi bodmi B_a a B_b, tak body B_a a B_b ležia na opačných stranách od priamky B_iB_c. Ak teda chceme také B_j, že na jednej i druhej strane od B_iB_j leží rovnako veľa bodov, stačí vybrať za B_j stredný prvok v našom usporiadaní. Takému prvku sa hovorí aj „medián“, a sú známe pomerne jednoduché algoritmy na jeho hľadanie.

Keď tento postup použijeme dvakrát, dostaneme dve priamky a z nich nejaký bod O, ktorý je jediným možným kandidátom na ohnisko. Treba ešte overiť, či to skutočne ohnisko byť mohlo. Na to stačí napríklad porátať, aké sú uhly medzi polpriamkami OB_i a polpriamkou OB_1. Tieto uhly musia byť násobkami $\frac{360^\circ}{2N}$ (keďže sú to vlastne uhly medzi dvojicami uhlopriečok v pravidelnom $2N$-uholníku) a všetky musia byť navzájom rôzne.

Implementácia Na hľadanie mediánu je použitý postup zvaný Median-Of-Five, ktorý má lineárnu časovú i pamäťovú náročnosť. Funguje podobne ako QuickSelect (to je taký skoro QuickSort, ktorý sa však volá rekurzívne iba na jednu z častí), akurát pivota vyberá prefíkanejšie – nájde mediánov maličkých úsekov (päťprvkových, odtiaľ ten názov) a pivota zvolí ako mediána z nich. No a ostatné časti sú očividne lineárne (časovo i pamäťovo).

Listing programu:

```
const MAX=100;

type tsmer=record p:integer; h:real; end;
var smer:array[1..2*MAX] of tsmer;
    test:array[0..2*MAX] of boolean;
    domcek:array[1..2*MAX] of record x,y:real; end;
    n,i:integer;
    a1,b1,c1,a2,b2,c2,sx,sy:real;

procedure vymen(i,j:integer);
var t:tsmer; begin t:=smer[i]; smer[i]:=smer[j]; smer[j]:=t; end;

function cmp(i,j:integer):integer;
 begin
  if smer[i].h>smer[j].h then cmp:=1
  else if smer[i].h<smer[j].h then cmp:=-1
  else cmp:=0;
 end;

procedure rozdel(l,r:integer;var m1,m2:integer);
 var k:integer;
 begin
  vymen(l,l+m1);
  m1:=l; m2:=l;
  for k:=l+1 to r do
   case cmp(k,m1) of
    -1: begin inc(m2); vymen(k,m1); vymen(k,m2); inc(m1); end;
    0: begin inc(m2); vymen(k,m2); end;
   end;
 end;

procedure utried(l,c:integer);
 var i,j,r:integer;
 begin
  r:=l+c-1;
```

```
  for i:=l to r-1 do for j:=i+1 to r do if cmp(i,j)>0 then vymen(i,j);
 end;

procedure median(l,r,p:integer);
 var i,j,m,v:integer;
 begin
  if (r-l<=10) then utried(l,r-l+1)
  else
   begin
    i:=l; j:=l;
    while (i+5)<=r do begin utried(i,5); vymen(i+2,j); inc(i,5); inc(j); end;
    if i<=r then begin utried(i,r-i+1); vymen((i+r) div 2,j); inc(j); end;

    m:=(j-l) div 2;
    median(l,j,m);
    rozdel(l,r,m,v);

    if p<=m-l then median(l,m,p)
    else if p>v-l then median(v,r,p-(v-l));
   end;
 end;


function atan2(dx,dy:real):real;
 begin
  if (dx>0) then
    if (dy>=0) then atan2:=180*arctan(dy/dx)/pi
    else atan2:=270+atan2(-dy,dx)
  else
    if (dy>=0) then atan2:=90+atan2(dy,-dx)
    else atan2:=180+atan2(-dx,-dy);
 end;

procedure najdipriamku(var a,b,c:real);
 var i,j,k1,k2:integer;
 begin
  k1:=1;
  while test[k1] do inc(k1);
  for i:=k1+1 to 2*n do
   if (not test[i]) and (domcek[i].y<domcek[k1].y) then k1:=i;
  test[k1]:=TRUE;

  j:=1;
  for i:=1 to 2*n do if not test[i] then with smer[j] do
   begin
    p:=i;
    h:=atan2(domcek[i].x-domcek[k1].x,domcek[i].y-domcek[k1].y);
    inc(j);
   end;
  median(1,j-1,j div 2);
  k2:=smer[j div 2].p;
  test[k2]:=TRUE;

  a:=domcek[k1].y-domcek[k2].y;
  b:=domcek[k2].x-domcek[k1].x;
  c:=domcek[k2].x*domcek[k1].y-domcek[k1].x*domcek[k2].y;
end;

function kontrola(a1,b1,c1,a2,b2,c2:real;var sx,sy:real):boolean;
 var
```

```
 d,q:real;
 i,j:integer;
begin
 kontrola:=false;
 d:=a1*b2 - b1*a2;
 if d=0 then exit;
 sx:=(c1*b2-b1*c2)/d;
 sy:=(a1*c2-c1*a2)/d;

 for i:=1 to 2*n do test[i]:=FALSE;
 with domcek[1] do d:=atan2(x-sx,y-sy);
 for i:=1 to 2*n do with domcek[i] do
  begin
   q:=(atan2(x-sx,y-sy)-d)*n/180+2*n;
   j:=trunc(q) mod (2*n);
   if (abs(q-trunc(q))>0.0001) or (test[j]) then exit;
   test[j]:=TRUE;
  end;
 kontrola:=true;
end;

begin
 read(n);
 for i:=1 to 2*n do with domcek[i] do read(x,y);
 for i:=1 to 2*n do test[i]:=FALSE;
 najdipriamku(a1,b1,c1);
 najdipriamku(a2,b2,c2);
 if kontrola(a1,b1,c1,a2,b2,c2,sx,sy) then
  writeln('Mame ohnisko na suradniciach (',sx:5:2,', ',sy:5:2,')')
 else writeln('Nenasli sme druidov...');
end.
```

Opravoval Braňo
(max. 15 bodov)

4. O kanárikovi

Majme slovo w s dĺžkou n. Priamočiare riešenie skúša, či v slove w existuje perióda dĺžky 1, 2, ... n. Aby sme zistili, či v slove existuje perióda konkrétnej dĺžky l, skontrolujeme (presne podľa definície v zadaní) pre každé $i = 1 \ldots (n - l)$ rovnosť písmen i a $i + l$. Máme riešenie v čase $O(n^2)$.

Ako sa to dá vylepšiť? Pozrime sa ešte raz na definíciu periódy. Ak má slovo w periódu l, tak je na pozíciach i, $l + i$, $2l + i$, ... to isté písmeno. To ale znamená, že môžme zobrať slovo w, posunúť ho o l písmen doprava a nadpísať nad pôvodné slovo w. Písmená v stĺpcoch budú rovnaké. Platí to aj opačne. Ak slovo w posunieme o l písmen doprava, nadpíšeme nad pôvodné w a písmená v stĺpcoch sú rovnaké, písmená i a $(i + l)$ sú tiež rovnaké.

K čomu je to dobré? Predstavme si, že na zemi je v políčkach $0 \ldots n$ napísané slovo w a že máme po ruke kopec malých kakapotvůrek.[7] Každá taká kakapotvůrka má na začiatku v rukách stĺpik písmen. Na ňom sú zaradom všetky písmená slova w. Na vrchu je prvé písmeno, potom druhé, ... Taktiež si každá kakapotvůrka pamätá počet písmen, ktoré už znehodnotila (na začiatku je to 0). Keď sa kakapotvůrka chce pohnúť, tak skontroluje, či písmená pod jej nohami a na vrchu jej kôpky sú rovnaké. Ak áno, znehodnotí (použije?) písmeno na vrchu svojho stĺpiku, spraví na políčko pod nohami kôpku o veľkosti počtu svojich znehodnotených

[7] Vy nepoznáte malé kakapotvůrky? Tak usilovne riešte, aby ste sa dostali na sústredenie. Na poslednom sústredení sme ich niekoľko videli.

písmen a pohne sa na ďalšie políčko. Ak sa písmená nezhodujú, kakapotvůrka ujde niekde do rohu a už sa nikdy neukáže. Všimnime si, že keď pustíme kakapotvůrku z pozície i a ona dôjde až na koniec slova, vieme že slovo má periódu i. Stačí teda zaradom púšťať z pozície $i = 2 \dots n$ kakapotvůrku. Ak dôjde do konca, vypíšeme, že najmenšia perióda je i. Ak nedôjde, pustíme ďalšiu z nasledujúcej pozície. Ak žiadna kapotvúrka nedôjde, perióda je n. Toto riešenie má však stále časovú zložitosť $O(n^2)$.

Teraz si predstavme, že niekto pred name presne podľa predchádzajúceho postupu popúšťal kakapotvůrky. Nám však ostali len kôpky, ktoré kakapotvůrky ponechávali. Vieme teraz určiť najkratšiu periódu slova w? Zjavne prvá kakapotvůrka, ktorá prejde do konca nechá na poslednom políčko najväčšiu kôpku. Preto stačí pozreť na veľkosť c najväčšej kôpky na poslednom políčku a vypísať $n - c$.

Ale keďže je simulovanie kakapotvůrek také pomalé, nedajú sa rýchlejšie zistiť aspoň veľkosti ich kôpok? To nám predsa stačí. Prvé dôležité pozorovanie bude, že aj keby sme mali na každom políčku poznačenú len najväčšiu kôpku, veľkosti ostatných kôpok vieme ľahko zistiť. Označme si c_1 veľkosť najväčšej kôpky na danom políčku i. To že najväčšia kôpka má veľkosť c_1 ale znamená, že políčka $(i - c_1 + 1) \dots i$ su rovnaké, ako políčka $1 \dots c_1$. Preto druhá najväčšia kôpka bude rovnaká, ako je prvá najväčšia kôpka na políčku c_1. Označme jej veľkosť c_2. Veľkosť tretej najväčšej kôpky na políčku i je uložená na políčku c_2. Opakovaním tohto postupu dostaneme veľkosti všetkých kôpok na ľubovolnom políčku.

Kôpka na pozícii 1 má veľkosť 0, lebo tade nikdy žiadna kakapotvůrka neprejde. A čo s najväčšou kôpkou na ostatných políčkach? Majme zistené veľkosti najväčších kôpok pre všetky predchádzajúce políčka. Najväčšiu kôpku na políčku i môžu zanechať iba kakapotvůrky, ktoré prišli z predchádzajúceho políčka, alebo kakapotvůrka, ktorá tu začínala. Budeme preberať kakapotvůrky z predchádzajúceho políčka od najvzdialenejšej po najbližšiu a nakoniec tu začínajúcu kakapotvůrku. Ak sa kakapotvůrkine písmeno na vrchu jej stĺpika zhoduje s písmenom na políčku i, kakapotvůrka sem úspešne prešla a nechala svoju kôpku. Keď nájdeme prvú úspešnú kakapotvůrku, môžme skončiť, lebo sme našli najväčšiu kôpku na políčku i. Takto zaradom zistíme najväčšie kôpky pre všetky políčka $2 \dots n$.

Aká bude časová zložitosť? Pri spracovávaní políčka najskôr nájdeme niekoľko neúspešných kakapotvůrok (pri tom sa veľkosť aktuálnej najväčšej kôpky zmenšuje) a jednu úspešnú kakapotvůrku (pri tom sa veľkosť najväčšej kôpky zväčší o 1). Keďže zväčšení je maximálne n, nebude viac ani zmenšení. Preto časová aj pamäťová zložitosť je $O(n)$.

Implementácia je priamočiara, oplatí sa však indexovať slovo w od 0. Nemusíme potom pri hľadaní najväčšej kôpky na danej pozícii špeciálne ošetrovať práve začínajúcu kakapotvůrku.

Bodovanie: Za riešenia v čase $O(n)$ ste mohli získať 15 bodov. Za $O(n^2)$ 8 a za $O(n^2 \log n)$ 7. Ak som usúdil, že opis je nedostatočný, strhával som max. 2 body. Ak bolo riešenie nefunkčné, alebo som ho z opisu a zdrojového textu v rozumnom čase nepochopil, dostali ste max. 3 body.

Listing programu:

```
program O_kanarikovi;
const MAX = 1000000;
var w : array[0..MAX] of char;      {vstupne slovo}
    A : array[0..MAX] of integer;   {velkost najvacsich kopok}
    i, c, n : integer;
begin
  n:=0;
  while (not eoln(input)) do begin
    read( w[n] ); inc(n);
  end;

  A[0]:=-1;
```

```
  for i:=1 to n-1 do begin
    c:=A[i-1];            {najvacsia kopka z predch. policka}
    while true do begin
      if (w[i]=w[c+1]) then begin
        inc(c); break;  {ak sa pismena zhoduju, kopka rastie}
      end;
      if (c=-1) then break;
      c:=A[c];          {prechod na dalsiu mensiu kopku}
    end;
    A[i]:=c;
    writeln(A[i]);
  end;
  writeln('Najkratsia perioda je ',n-A[n-1]+1);
end.
```

5. O mravcoch I

opravoval MišoF.
(max. 15 bodov)

Ako si väčšina riešiteľov uvedomila, mravčie počítače vlastne predstavujú akýsi veľký počítač, ktorý je schopný robiť paralelne (t.j. naraz) viacero vecí, ktoré si navzájom príliš neprekážajú. Celý trik bude v tom, ako šikovne rozdeliť prácu medzi jednotlivé malé počítače.

Riešenie v lineárnom čase je jednoduché. Spustíme N počítačov, každý bude mať na starosti spočítanie jedného políčka. To spravia ľahko: i-ty počítač $i-1$ taktov počká a následne (ak $i > 0$) k `Mem[i]` priráta `Mem[i-1]`. Lineárny čas, lineárna pamäť, lineárny počet procesorov. Maximálne 5 bodov.

Maximálne 5 bodov... nie je to málo? Ani nie. Totiž: Spustime len jeden počítač. Ten prejde celé pole zľava doprava a (opäť v lineárnom čase) spočíta hľadané hodnoty. Predchádzajúce riešenie bolo teda ešte horšie ako klasické sekvenčné[8] – na dosiahnutie rovnakého času výpočtu potrebovalo viac počítačov. Nuž, ako sa hovorí, priveľa kuchárov pokazí polievku.

Ak si samozrejme tí kuchári prácu rozumne nerozdelia. Pozrime sa teraz na jednu možnosť, ako hľadané súčty spočítať v čase $O(\log N)$. Budeme mať opäť N počítačov a úlohou i-teho z nich bude spočítať výslednú hodnotu políčka `Mem[i]`. Keďže hodnoty `Mem[i]` sa budú počas behu algoritmu meniť, budeme pri vysvetľovaní pôvodné hodnoty označovať `A[i]`.

Pre jednoduchosť vysvetľovania zatiaľ predpokladajme, že pred vstupom máme v poli ešte veľmi veľa núl. (Keď k niečomu pripočítame nulu, výsledok sa nám nezmení.)

Všetky počítače budú robiť presne to isté. V prvej fáze sa i-ty počítač pozrie na políčko `Mem[i-1]` a pripočíta hodnotu z neho k `Mem[i]`. Čo sme takto dostali? Na každom políčku poľa `Mem[]` je teraz súčet dvoch hodnôt zo vstupu.

Pokračujme ďalej. V druhej fáze sa i-ty počítač pozrie na políčko `Mem[i-2]` a pripočíta hodnotu z neho k `Mem[i]`. Čo sme dostali teraz? V `Mem[i-2]` bola spočítaná hodnota `A[i-3]+A[i-2]`, v `Mem[i]` bola spočítaná hodnota `A[i-1]+A[i]`. Preto na každom políčku poľa `Mem[]` sme dostali súčet štyroch hodnôt zo vstupu.

Už je zjavné, ako budeme pokračovať ďalej. Po k fázach máme v `Mem[i]` súčet hodnôt `A[i-`2^k`+1]+...+A[i]`. Keď k `Mem[i]` pripočítame `Mem[i-`2^k`]`, dostaneme v `Mem[i]` súčet dvakrát dlhšieho úseku vstupu.

Zjavne nám stačí spraviť $f = \lceil \log_2 N \rceil$ fáz. Na konci poslednej fázy bude v každom políčku `Mem[i]` spočítaný súčet `A[i-`2^f`+1]+...+A[i] = 0+...+0+A[1]+...+A[i]`, teda presne to, čo sme chceli.

Pri implementácií potrebujeme ošetriť, aby nám žiaden počítač nechcel pristupovať do poľa `Mem[]` mimo úseku, kde sú naše hodnoty. To je ale triviálne. Keď sme v k-tej fáze,

[8] Sekvenčný alg. je algoritmus bežiaci iba na jednom počítači. Opakom sekvenčného je paralelný algoritmus.

počítače s *cpuid* $\leq 2^k$ už sú hotové a môžu skončiť (už by prirátávali len nuly). Ostatné počítače nerušene rátajú ďalej.

Náš algoritmus teda potrebuje N počítačov. Časová zložitosť je $O(\log N)$ – máme rádovo $\log N$ fáz, každá zbehne v konštantnom čase. Pamäť je zjavne $O(N)$. Korektné riešenie s takouto časovou a pamäťovou zložitosťou mohlo získať plný počet bodov. (Samozrejme ak obsahovalo dostatočný popis, zdôvodnenie správnosti, odhady zložitosti a všetko ostatné čo má riešenie obsahovať.)

```
  var dva_na_k, pom : integer;

  if (cpuid > Mem[0]) then halt;
  dva_na_k := 1;
1: if (cpuid <= dva_na_k) then halt;
  pom := Mem[cpuid - dva_na_k];
  Mem[cpuid] := Mem[cpuid] + pom;
  dva_na_k := 2 * dva_na_k;
  goto 1;
```

Ako to, že toto vzorové riešenie ešte pokračuje? Veď sme si už ukázali riešenie, ktoré by dostalo plný počet bodov... tak čo nás ešte čaká?

Načo sú vlastne dobré paralelné algoritmy, keď v živote málokto stretne počítač s viac ako 20 procesormi? Jednoducho na to, že na našich 20 procesoroch budeme postupne simulovať kroky tých N počítačov z algoritmu. V jednom kroku náš paralelný algoritmus spravil N navzájom nezávisiacich krokov. Pri simulácií každý z 20 reálnych procesorov spraví $N/20$ z týchto krokov.

Definujme si *prácu* paralelného algoritmu ako súčet časov behu jednotlivých počítačov.[9] V ideálnom prípade sa nám podarí *prácu* rozdeliť medzi našich 20 procesorov rovnomerne a čas behu programu bude (*práca*/20). Keby sme mali 40 procesorov, čas bude prinajlepšom (*práca*/40). Ale aj keby sme mali miliardu procesorov, čas nikdy nebude lepší ako $O(\log N)$,[10] lebo napr. kroky N-tého počítača treba simulovať postupne.

Optimálny by bol algoritmus, ktorý by mal čo najlepšiu časovú zložitosť a pritom prácu rovnakú, ako sekvenčný algoritmus pre danú úlohu. Práca sekvenčného algoritmu, ktorý rieši túto úlohu, je zjavne lineárna. Predchádzajúce riešenie (a rovnako aj všetky riešenia, ktoré naozaj dostali 15 bodov) však majú prácu $O(N \log N)$ – každý z N počítačov spraví rádovo $\log N$ krokov.

Ukážeme si teraz fintu, ako pri nezhoršenej časovej zložitosti zlepšiť prácu nášho riešenia na lineárnu. Táto finta je prekvapivo všeobecná a dá sa použiť pri veľkom množstve paralelných algoritmov.

Označme si $k = \lfloor \log N \rfloor$. Budeme používať $P = \lceil N/k \rceil$ počítačov. Každému pridelíme úsek vstupu dĺžky k. Začneme tým, že v $O(k)$ krokoch si každý počítač sekvenčne spočíta čiastočné súčty na svojom úseku. Táto časť má časovú zložitosť $O(k)$ a vykoná sa práca $O(Pk) = O(N)$.

Teraz každý počítač nakopíruje súčet svojho úseku niekam ďalej do globálnej pamäte. Takto dostaneme P hodnôt. Pre tieto hodnoty spočítame čiastočné súčty vyššie popísaným algoritmom. Máme P hodnôt, P procesorov. Časová zložitosť tejto časti bude $O(\log P) = O(\log(N/\log N)) = O(\log N - \log\log N) = O(\log N)$. Vykonaná práca bude $O(P \log P) = O(P \log N) = O(N)$.

Teraz už ale každý počítač vie súčet hodnôt v predchádzajúcich úsekoch a v k krokoch ho pripočíta k hodnotám v svojom úseku a sme hotoví. Elegantné, nie?

[9] Zjavne *práca* sekvenčného algoritmu je rovná jeho časovej zložitosti.

[10] Resp. vo všeobecnosti: nebude nikdy lepší ako časová zložitosť príslušného paralelného algoritmu.

BONUS

V týchto vzorových riešeniach nám ešte ostal kopec voľného miesta, preto ho využijeme aspoň trochu užitočne. Budeme pokračovať na tému piateho príkladu z prvej série KSP-Z a ukážeme si niekoľko zaujímavých programov, väčšinou v jazyku C. Začneme zľahka. Čo robí tento pochybný program?

```
#include <stdio.h>
int x,y;
int main(void){scanf("%d %d",&x,&y); x^=y^=x^=y; printf("%d %d\n",x,y); return 0;}
```

A tento?

```
const a='begin writeln(const a=,#39,a,#39,#59);writeln(copy(a,1,14),#39,copy(a,15,8),#39,copy(a,23,79));end.';
begin writeln('const a=',#39,a,#39,#59); writeln(copy(a,1,14),#39,copy(a,15,8),#39,copy(a,23,79)); end.
```

No dobre. Doteraz to bolo ľahké. Ale čo preboha má byť toto? Tvar mnohé napovedá, aj názvy premenných pomôžu...[11]

```
char*M,A,Z,E=40,J[40],T[40];main(C){for(*J=A=scanf(M="%d",&C);
--            E;            J[            E]            =T
[E   ]=  E)   printf("._");  for(;(A-=Z=!Z)  ||  (printf("\n|"
)    ,   A    =             39            ,C            --
)    ;   Z    ||    printf   (M   ))M[Z]=Z[A-(E   =A[J-Z])&&!C
&    A   ==             T[                                  A]
|6<<27<rand()||!C&!Z?J[T[E]=T[A]]=E,J[T[A]=A-Z]=A,"_.":" |"];}
```

Jemne zákerná otázka: V akom jazyku je písaný nasledujúci program?

```
(*haluz/**)begin writeln('pas');end.(**/);int main(){printf("c\n");return 0;}//*)
```

A čo ešte tento?

```
(*haluz/* ) (PostScript)
/Times-Roman findfont 40 scalefont setfont
newpath 100 400 moveto show showpage % */ ); /*
<~ */
#include <stdio.h>
int main() /*~>( */ { /* )<~*/ printf("C\n"); return 0;
/* ~> ( *) {
% } begin writeln('pascal'); (*
} % *) end. {*/}
```

Na záver ešte exkurzia do krás jazyka C. Napovieme, že `'-'-'-'` je nula – je to (znak mínus) mínus (znak mínus). So zvyškom sa môžete pohrať sami.

```
int i;main(){for(;i["]<i;++i){--i;}"];read('-'-'-',i+++"hell\
o, world!\n",'/'/'/'));}read(j,i,p){write(j/p+p,i---j,i/i);}
```

Ak sa vám tento bonus páčil, pošlite pozitívny feedback, možno bude nabudúce pokračovanie...

[11] Technický detail: gcc potrebuje parameter `-fwritable-strings`, aby tento program skompilovalo správne.

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Ročník	11	12	13	14	15	Σ
1	Jančuška Marek	Gym. Párovská Nitra	4	14	15	12	15	15	71
2	Perešíni Peter	Gym. Tajovského B. Bystrica	2	14	15	15	8	15	67
3	Glaus Peter	Gym. Jura Hronca BA	4	14	15	6	15	15	65
4	Lenhardt Rastislav	Gym. Jura Hronca BA	4	15	11	15	8	15	64
5	Cicko Miroslav	Gym. Tajovského B. Bystrica	3	14	15	15	8	11	63
6	Bernát Marek	Gym. K. Štúra Modra	4	14	15	15	3	15	62
6	Simančík František	Gym. Grösslingová BA	3	12	15	12	8	15	62
8	Imriška Jakub	Gym. Jura Hronca BA	2	12	15	12	7	15	61
8	Nánási Michal	Gym. Jura Hronca BA	3	14	11	14	8	14	61
10	Poláček Lukáš	Gym. K. Štúra Modra	4	12	15	15	3	15	60
11	Smažáková Dana	Gym. Jura Hronca BA	4	12	10	14	8	15	59
12	Kuchynárová Mariana	Gym. Jura Hronca BA	4	15	9	15	3	14	56
13	Plžík Milan	Gym. Ľ. Štúra Zvolen	3	14	15	9	8	5	51
14	Kevický Michal	Gym. Grösslingová BA	4	14	15	4	2	15	50
15	Baláž Miroslav	Gym. Jura Hronca BA	4		13	14	8	13	48
16	Kováč Jakub	Gym. Jura Hronca BA	4	14	15		3	15	47
16	Tekeľ Jakub	Gym. Jura Hronca BA	4	12	11		9	15	47
18	Buštor Stano	Gym. Jura Hronca BA	3	12		12	8	14	46
18	Hrinčár Peter	ZŠ Jarná Poprad	0	12	10	3	8	13	46
20	Repovský Michal	Gym. Komenského Trebišov	4	11	15		3	14	43
21	Rejda Martin	Gym. Grösslingová BA	4	12	12	2	14	2	42
22	Trejbal Ivan	Gym. Jura Hronca BA	4	11	11	9	8		39
22	Zeman Marek	Gym. Jura Hronca BA	3	11	7		8	13	39
24	Trenkler Pavol	Gym. Zbrojničná Košice	4	12		7	3	15	37
25	Kuštárová Tamara	Bilingválne gym. Sučany	3	14			8	14	36
26	Bundala Daniel	Gym. Jura Hronca BA	2		11	2	8	14	35
27	Dzetkulič Michal	Gym. P. Horova Michalovce	3	5	11		3	15	34
27	Petruľák Matúš	Gym. Grösslingová BA	3	12		3	8	11	34
27	Šomlo Ivan	Gym. Mládežnícka Šahy	4	12	10	3	8	1	34
30	Ludha Marek	Gym. Tajovského B. Bystrica	4	12	1	5	8	6	32
31	Schlosáriková Eva	Gym. SNP Piešťany	3	12	8	2	8		30
32	Kundrák Ľubomír	Ev. lýceum BA	1	12	9		8		29
33	Beleš Lukáš	Gym. Čadca	3	15	7		3	3	28
34	Hanulová Anna	Gym. Jura Hronca BA	4	14		2	8		24
34	Štolc Miroslav	Gym. Párovská Nitra	4	14		3	3	4	24
36	Gottweis Martin	Gym. Jura Hronca BA	2	5	5	2	3	8	23
37	Bodnár Jozef	Gym. Fiľakovo	3	13			7		20
37	Mindek Peter	Škola pre mim. nadané deti BA	2	12			8		20
39	Krchniak Matej	Gym. Jura Hronca BA	2	5	1		8	4	18
39	Vadkerti Miroslav	SPŠE Nové Zámky	4	5	7	3	3		18
41	Hrobár Miso	Gym. Jura Hronca BA	4		11			1	12
41	Pančík Andrej	Gym. Š. Moyzesa B. Bystrica	1	3			3	6	12
43	Ragány Peter	SPŠE Prešov	???				8		8
43	Sedlák Štefan	Gym. Daxnera V.n. Topľou	3				8		8
45	Ďurfina Lukáš	Gym. Golianova Nitra	3				2		2
46	Oremus Vladimír	Gym. Jura Hronca BA	2				1		1