**Korešpondenčný seminár z programovania**
**XXV. ročník, 2007/08**
**Katedra základov a vyučovania informatiky FMFI UK,**
**Mlynská Dolina, 842 48 Bratislava**

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

# Vzorové riešenia 1. kola letnej časti

Milé naše riešiteľky, milí naši riešitelia, milé naše riešiteľatá, dostáva sa Vám do rúk ďalšie vydanie populárneho bestselleru známeho ako Vzorové riešenia. Dúfame, že dosiahne čítanosť porovnateľnú s titulmi ako Pán Krúžkov, alebo Hrnčiar Harry. A nie že nad nimi strávite tak menej času ako my!

POZOR! Spôsobuje zdvojené videnie! POZOR! Spôsobuje zdvojené videnie!

KSPáci

## 1. Zlietajú sa lastovičky

opravovala Elfinka
(max. 15 bodov)

Tento príklad nebol ťažký, všetkým sa vám ho podarilo úspešne vyriešiť a väčšina z vás mala vzorové riešenie. Za lineárnu pamäť sa stratil bodík, za horší čas bolo 12 bodov. Nuž a body sa dali stratiť aj za zlé/chýbajúce odhady a popis.

Nuž ale pustime sa do vzoráku. Vieme, že lastovičky máme na vstupe utriedené podľa vzdialenosti od začiatku drôtu. To znamená, že lastovičky, ktoré sedia vedľa seba sú na vstupe za sebou. My chceme aby si ďalšie lastovičky posadali medzi tie, ktoré tam už sedia. Na toto nám stačí, aby sme pre každú dvojicu vedľa seba sediacich lastovičiek vyrátali, koľko sa medzi ne zmestí nových, a tieto hodnoty sčítali.

Teraz nám už stačí zistiť, koľko lastovičiek sa zmestí medzi dve. Zistíme si, koľkokrát je medzi nimi vzdialenosť $x$ – pokiaľ sú lastovičky na pozíciách $a$, $b$, $a < b$, bude to $(b-a)$ div $x$. Od tohto čísla ešte odčítame 1 (lebo aj lastovičky na pozíciách $a$, $b$ musia byť vzdialené najmenej $x$). Teraz nám už len stačí v cykle od 1 po $n-1$ vždy načítať lastovičku, vyrátať počet nových ktoré sa zmestia a prirátať ho k celkovému počtu lastovičiek.

Máme teda len 1 cyklus dĺžky $n$, časová zložitosť je teda $O(n)$. Keďže si pamätáme len pár premenných a vstup spracovávame už počas načítania, je pamäťová zložitosť $O(1)$.

### Listing programu:

```
var a,b,x,N,i,pocet:integer;

begin
  pocet:=0;
  readln(N,x);
  read(a);
  for i:=2 to N do begin
    read(b);
    pocet:=pocet+(b-a)div x-1;
    a:=b;
  end;
  writeln(pocet);
end.
```

## 2. Zmrznutá tybloň

opravoval Hermi
(max. 15 bodov)

Aj keď tento príklad vyzeral zložito, väčšina z Vás sa nedala zmiasť zadaním a vymyslela vzorové riešenie, alebo aspoň nejaké jemu pomerne blízke. Za riešenie v $O(N)$ čase s konštantnou pamäťou sa dal získať plný počet, za lineárnu pamäť 12 bodov, pomalšie riešenia dostali body zodpovedajúce svojej časovej zložitosti. Teraz sa už bez oštiepkania vrhnime priamo na riešenie ako malo vyzerať podľa nás.

Prvý a najdôležitejší krok bolo uvedomiť si, že nech je tyblko na strome umiestnené kdekoľvek, môžeme si vybrať, či z neho vezmeme šupku alebo červíka nezávisle od toho, čo si vyberieme u ostatných tybĺk. Stručne načrtnem ideu: postupujeme od tyblka na konci konára (t.j. od jablka ktorého odrezaním nič neodpadne), môžeme sa rozhodnúť či ho rozrežeme, alebo necháme na šupku. Ak ho rozrežeme, žiadne jablké neodpadnú (lebo bolo na konci) a ak sa rozhodneme nechať ho na šupku, je nám jedno či odpadne, alebo ostane na strome až do konca, preto môžeme postúpiť k tyblku, z ktorého vyrastá a pre neho zopakovať ten istý postup (ak odpadne jablko z ktorého chceme šupkui, neuškodí nám to).

Keď sme vyzbrojení takouto informáciou, môžeme úplne odignorovať štruktúru stromu, stačí nám načítavať údaje o cenách za jednotlivé tyblká a vždy sa rozhodneme pre výhodnejšiu možnosť. Navyše akonáhle raz rozhodneme o tyblku, údaje si nepotrebujeme už ďalej pamätať (toto bola druhá dôležitá vec ktorú si bolo treba uvedomiť pri implementovaní), preto si vystačíme s konštantným počtom premenných a pamäťová zložitosť je $O(1)$. Časová zložitosť algoritmu je $O(N)$, pretože iba načítame informácie o $N$ tyblkách a pre každé spravíme jedno porovnanie.

### Listing programu:

```
var
  i, n, cervik, supka, zbytocne, zisk: integer;

begin
  zisk := 0;
  readln(n);
  for i := 1 to n-1 do readln(zbytocne,zbytocne);
  for i := 1 to n do begin
    readln(cervik, supka);
    if cervik > supka then zisk := zisk + cervik
    else zisk := zisk + supka;
  end;
  writeln(zisk);
end.
```

## 3. Zase tie výroky...

opravoval Mic
(max. 15 bodov)

Prístupov ako riešiť túto úlohu bolo viac. Prvý z nich je taký, že si usporiadame všetky výroky a potom poľahky zrátame koľko výrokov každého typu je. Takéto riešenie nás vedie k časovej zložitosti $O(n \log n)$, ak použijeme efektívny triediaci algoritmus[1]. Avšak všimnime si nasledovnú vec: Ak máme $n$ výrokov a máme výrok tvaru „Práve $v$ z týchto výrokov je pravdivých“ a zároveň $v > n$, tak potom tento výrok nemôže byť pravdivý[2]. To znamená,

[1] napríklad Quicksort.

[2] Výrokov je predsa najviac $n$

že každý taký výrok môžeme pri načítavaní zahodiť. Teraz nám už stačí iba vytvoriť jedno pole o $n + 1$ prvkoch (pole budem nazývať `pocet`, indexuje sa v ňom od 0 po $n$ vrátane), ktoré bude na začiatku vyplnené nulami. Potom vždy keď načítame výrok s hodnotou $v$, skontrolujeme či $v \leq n$ a v našom poli na mieste $v$ zvýšime hodnotu o jedna. Takto nám `pocet`$[v]$ hovorí počet výrokov tvaru „Práve $v$ z týchto výrokov je pravdivých.“.

Ostáva nám nájsť najväčšie také číslo $v$, že výrok s číslom $v$ je pravdivý. To je najväčšie také $v$, pre ktoré platí `pocet`$[v] = v$. Toto sa dá spraviť jedným prechodom poľa, čiže napríklad jedným for-om s jedným if-om, alebo ako je to vo vzorovom programe, jedným cyklom while. Ostáva nám už iba skontrolovať jeden špeciálny prípad a to vtedy, keď 0 výrokov má byť pravdivých, ale na tabuli je aspoň jeden výrok tvaru „Práve 0 z týchto výrokov je pravdivých“. Vtedy 0 nie je správna odpoveď a odpovieme $-1$, čiže neexistuje také číslo $v$, že „Práve $v$ z týchto výrokov je pravdivých.“.

Koľko bude celá táto mašinéria trvať? Vynulovanie poľa a načítanie vstupu zvládneme v čase $O(n)$. Nájdenie odpovede je maximálne prechod celým poľom, čiže celková časová zložitosť je $O(n)$. Keďže si pamätáme jedno pole veľkosti $n$, tak pamäťová zložitosť bude tiež $O(n)$.

Ako sa bodovalo? Každý z vašich programov bol otestovaný na 15 testovacích sadách. Každá obsahovala aspoň jeden vstup. Ak váš program dal správnu odpoveď na každý vstup v danej testovacej sade, tak ste dostali za tú testovaciu sadu 1 bod, v opačnom prípade 0 bodov. V jednotlivých sadách postupne rástlo $n$, takže ak ste mali napríklad kvadratické riešenie, viac ako 10 bodov by ste dostať nemali. Niektoré sady obsahovali výrok s väčším číslom ako $n$, takže aj tam sa dali stratiť body a ďalšie body sa dali stratiť na zlom ošetrovaní výnimky, kedy neexistuje žiadna správna odpoveď.

Teraz sa už na liahni nachádzajú všetky testovacie vstupy, takže si môžete váš program opraviť a skúsiť nechať otestovať. Veľa šťastia.

## Listing programu:

```
program vyroky;
  var pocet:array[0..4000000] of longint;
  var i,N,max,a:longint;
begin
  readln(N);
  for i:=0 to N do
    pocet[i]:=0;
  for i:=1 to N do begin
    readln(a);
    if(a>=0)and(a<=N)then
      inc(pocet[a]);
  end;
  max:=N;
  while (max>=1) and (pocet[max]<>max) do
    dec(max);
  if(max<1)and(pocet[0]>0)then
    writeln('-1')
  else if max<1 then
    writeln(0)
  else
    writeln(max);
end.
```

# 4. Organizácia Kopanie Stôk Prievidza

opravoval Maják
(max. 15 bodov)

Poďme rovno zhurta na riešenie.

Použijeme prehľadávanie do hĺbky. Ako takého prehľadávanie funguje? O každom človeku si budeme pamätať, či sme mu už nejakého šéfa pridelili, aby sme vedeli, kým sa už nemusíme zaoberať. Ďalej budeme používať dátovú štruktúru zásobník, alebo tiež LIFO. Táto skratka znamená last in first out. Teda keď zo zásobníka budeme vyberať, tak vyberieme tie prvky, ktoré doňho boli vložené ako posledné.

Podstata celého prehľadávania je takáto: Na začiatku vložíme šéfa na zásobník, a nastavíme mu ako šéfa samého seba, aby sme ho už viackrát nespracovali. Funkčnosť nášho algoritmu nezávisí od výberu šéfa, čo neskôr ukážeme. Jadro algoritmu robí toto: Vyberieme jedného človeka zo zásobníka. Priradíme mu ako šéfa toho, kto ho tam vložil. Potom sa pozrieme na všetkých ľudí, s ktorými je rozhádaný, a ak niekto z nich ešte nemá šéfa, tak ho zas šupneme na zásobník. Skončíme vtedy, keď sa nám vyprázdni zásobník. Ak ešte existujú ľudia, ktorým sme nepridelili šéfa, tak vieme, že organizačná štruktúra sa nedá vytvoriť, pretože títo ľudia nie sú rozhádaný s tými, ktorých sme prezerali (ak by boli, tak sa k nim prehľadávaním dostaneme).

Tento program má ale jednu drobnú chybu. Tak ako je tu napísaný sa nám môže stať, že niektorého človeka vložíme do zásobníku viackrát. Napríklad hneď na začiatku, on nám tam potom na spodku bude kvasiť, a pridáme ho tam po čase zas, ak budeme spracovávať niekoho iného, kto je s ním rozhádaný. To vieme ale jednoducho opraviť, a to tak, že použijeme rekurziu. Prehľadávanie do šírky bude jedna procedúra, ktorej argumentom bude človek, ktorému sme práve určili šéfa. Pre každého z jeho potencionálnych podriadených zistíme, či už majú šéfa. Ak nie, tak túto procedúru rovno naňho spustíme. Tým docielime to, že kým sa nespracuje tento podriadený, tak o ďalších sa nebude uvažovať. Keď sa potom rekurzia vráti späť a budeme sa pozerať na zvyšných, tak len tý ktorých sme medzi časom nespracovali nemajú ešte šéfa.

Dôležitá otázka je, funguje naše riešenie? Robí to, čo po ňom chceme? To najjednoduchšie zistíme tak, či nami vytvorená organizačná štruktúra spĺňa požadované podmienky. Jedného šéfa máme, každý má práve jedného šéfa (okrem hlavného), a ani nikto nie je nadriadený sám sebe. Prvé tri podmienky teda spĺňame. Takisto je aj každý rozhádaný so svojim šéfom, pretože tak sme tú štruktúru vytvárali. Čo ale nie je jasné, je štvrtý bod. Ale aj ten platí. Ukážeme si, že situácia ktorá by ho porušovala nemôže nastať. Predpokladajme, že naše riešenie porušuje štvrtý bod. To znamená, že nastane situácia, pri ktorej existujú dvaja rozhádaní ľudia, ktorý sú v rozličných vetvách stromu [3]. Táto situácia ale nemohla nastať. Zoberme si z nich toho, ktorým sme sa pri prehľadávaní zaoberali ako prvým. Označme si ho $A$, druhého $B$. Keďže $A$ je rozhádaný s $B$, tak jediný dôvod, prečo $A$ sme nepriradili $B$ ako podriadeného, môže byť ten, že ho priradíme niekomu inému, napríklad $C$. $C$ ale musí byť podriadený $A$, pretože všetci ľudia ktorých určujeme šéfa pokým máme „rozpracovaného“ $A$ budú zároveň aj jeho podriadení.

Pre prehľadávanie do šírky si nepotrebujeme pamätať viac, ako kto je s kým rozhádaný, kto má koho ako šéfa, a zásobník. Na zásobník každého človeka vložíme naximálne raz, a preto pamäťová zložitosť je $O(N+K)$. A aká je časová? Náš algoritmus pre každého človeka pozrie všetky jeho rozhádania, či ten s kým je rozhádaný už má šéfa. Dokopy sa teda pozrieme na každého človeka a na každé rozhádanie (dvakrát, lebo rozhádanie je vzájomné). Časová zložitosť je teda tiež $O(N+K)$.

Na záver ešte pár slov k bodovaniu. Body som strhával najčastejšie na neukázanie, že váš algoritmus rieši danú úlohu (2–3b). Nestačí keď mu veríte vy, musíte to aspon nejak popísať,

[3] Pretože vtedy ani jeden nie je nadriadený tomu druhému.

prečo by mal fungovať. Druhé miesto zaujali algoritmy pomalšie ako prehľadávanie do hĺbky (2–7b). A zvyšok ste mohli statiť za iné chýbajúce časti, poprípade nefunkčný program.

### Listing programu:

```
program OKSP;
  const MAXLUDI = 100;
  var hadok:array[1..MAXLUDI] of integer;
  var hadky:array[1..MAXLUDI, 1..MAXLUDI] of integer;
  var sefovia:array[1..MAXLUDI] of integer;
  var N, K, a, b, i, j, nemajuSefa:integer;

procedure sefuj(sef:integer);
begin
  for i:=1 to hadok[sef] do begin
    if sefovia[hadky[sef, i]] = 0 then begin
      sefovia[hadky[sef, i]] := sef;
      sefuj(hadky[sef, i]);
      dec(nemajuSefa);
    end;
  end;
end;

begin
  readln(N, K);
  nemajuSefa := N;
  for i:=1 to N do sefovia[i] := 0;
  for i:=1 to N do hadok[i] := 0;
  for i:=1 to K do begin
    readln(a, b);
    inc(hadok[a]);
    hadky[hadok[a]] := b;
    inc(hadok[b]);
    hadky[hadok[b]] := a;
  end;

  sefuj(1);

  if nemajuSefa > 0 then writeln('Nie je možné vytvoriť organizačnú štruktúru.')
  else begin {toto vypisovanie je v O(N*N), nad rýchlejším môžete porozmýšlať sami}
    for i:=1 to N do begin
      write(i,':');
      for j:=1 to N do if sefovia[j] = i then write(' ',j);
      writeln;
    end;
  end;
end.
```

## 5. Opály, zafíry a iné drahé kamene

opravoval Miňo
(max. 15 bodov)

Tento príklad bol jednoznačne viac o matematike ako o programovaní – uvedomením si niekoľkých dôležitých faktov mohol riešiteľ získať postupne lepšie a lepšie riešenia. Nie vždy treba totiž použiť prvé dostatočne veľké kladivo, ktoré zíde na um.

Základnou myšlienkou je, že zadanie príkladu nám vlastne dáva rovnice v tvare $b_{i-1} + b_i + b_{i+1} = a_i$, kde $a_i$ je $i$-te číslo na vstupe (číslujeme od 0). Našou úlohou je vypočítať hodnoty $b_i$.

Zrejme najjednoduchším riešením je vziať Gaussovu elimináciu, zostaviť sústavu $n$ rovníc o $n$ neznámich a riešiť. Výsledok získame v $O(N^3)$ a pamäte spotrebujeme $O(N^2)$. Ide to aj rýchlejšie? Ide!

Pozrime sa bližšie na rovnice, ktoré máme, napríklad $a_{i+1} = b_i + b_{i+1} + b_{i+2}$ a $a_{i+2} = b_{i+1} + b_{i+2} + b_{i+3}$. Keď ich od seba odčítame a správne upravíme, dostaneme $a_{i+1} - a_{i+2} = b_i - b_{i+3}$, a teda $b_{i+3} = b_i + a_{i+2} - a_{i+1}$. To znamená, že ak vypočítame ľubovoľné $b_i$, zvyšné z neho už vieme dopočítať. Ďalej vieme zistiť zopár ďalších zaujímavých vecí – napríklad súčet pôvodných leskov. Na ten si stačí uvedomiť, že $\sum_{i=0} N - 1a_i = 3\sum_{i=0} N - 1b_i$.

Skúsme teraz vyriešiť prípad, ked $N \bmod 3$ je nenulové. Stále nám chýba vypočítať nejaké počiatočné $b_i$, z ktorého môžeme ďalej pokračovať. Skúsime sa teda zaobísť bez neho – namiesto toho si pevne určíme jedno $b_i$, napríklad $b_0 = 0$. Z neho potom môžeme vyjadriť ľubovoľné ďalšie dočasné $b_i$ – vďaka tomu, že $N \bmod 3$ nie je 0 vieme vypočítať ľubovoľné $b_i$. Týmto sme získali hodnoty relatívne ku $b_0$, teda vieme, že $b_i = b_0 + c_i$, pričom $c_i$ poznáme, ale $b_0$ ešte nie.

Teraz potrebujeme nájsť už iba správnu hodnotu pre $b_0$. Z nej potom vieme vypočítať všetky hodnoty $b_i$. A ako to dopočítať? Potrebujeme, aby súčet pôvodných leskov bol taký, ako sme si vypočítali zo zadaných hodnôt. Keďže súčet leskov vieme vyjadriť v tvare $\sum_{i=1}^{N} b_0 + c_i = N \cdot b_0 + \sum_{i=1}^{N} c_i$, tak hodnotu $b_0$ vyrátame jednoducho – zoberieme rozdiel súčtu, ktorý chceme dosiahnuť a súčet $c_i$.

Vypočítanie hodnôt pre $Nmod3 = 0$ je o čosi náročnejšie. Tu už nevieme vypočítať jedným priechodom celým poľom všetky hodnoty relatívne ku jednej $b_i$. Vieme však zobrať $b_0, b_1, b_2$ a počítať hodnoty relatívne k nim. Takto nám vzniknú 3 skupiny drahých kameňov (v jednej skupinke budú kamene, ktorých indexy dávajú rovnaký zvyšok po delení 3), ktorých relatívne hodnoty budeme vedieť. Zostáva už iba jediný problém – vypočítať skutočné hodnoty leskov (alebo zistiť, že náhrdelník je falošný).

Pre náhrdelník musí platiť, že lesky sú kladné celé čísla, pre každú skupinu teda nájdeme jej minimálny lesk. Ak je záporný, znamená to, že k nemu musíme prirátať také číslo, aby bol kladný, a toto isté číslo prirátať každému prvku skupiny. Naviac prirátame najmenšie číslo, ktoré spĺňa túto podmienku. Toto spravíme pre skupiny $b_0$ a $b_1$. A teraz nastane okamih pravdy – ku prvkom skupiny $b_2$ musíme prirátať také číslo, že platí $a_1 = b_0 + b_1 + b_2$ a zároveň v nej po jeho prirátaní budú iba kladné čísla. V opačnom prípade máme do činenia s falošným náhrdelníkom (čísla pripočítané ku ostatným skupinám boli minimálne možné, takže ich nemôžeme znížiť, aby sme prvkom poslednej skupiny pripočítali väčšie číslo).

Všimnime si, že týmto nepokazíme platnosť žiadnych ďalších rovníc. Každú totiž vieme vyjadriť v tvare $a_i = b_0 + b_1 + b_2 + c_i + d_{i+1} + e_{i+2} = a_1 + c_i + d_{i+1} + e_{i+2}$, kde $c_i, d_{i+1}, e_{i+2}$ sú rozdiely hodnôt $b_i, b_{i+1}, b_{i+2}$ oproti $b_0, b_1, b_2$ a tieto rozdiely máme už dávno správne vyrátané.

Riešení prišlo viacero druhov – našli sa Gaussove elminácie, našli sa riešenia hrubou silou, našli sa aj optimálne riešenia. Bohužiaľ, prišli aj nejake polo- či nefunkčné riešenia. Optimálne riešenie mohlo dostať najviac 15 bodov, Gaussova eliminácia bola odmeňovaná najviac 11-timi bodmi. V niektorých prípadoch sa stávalo, že program fungoval iba pre prípad $N \bmod 3! = 0$. Takéto riešenie vedelo získať najviac 9 bodov. Riešenie hrubou silou (skúšanie prvých dvoch leskov a kontrola ostatných) dostalo maximálne 7 bodov.

V tomto príklade sa kládol aj veľký dôraz na správne matematické zdôvodnenie a za popis sa dali stratiť až 3 body. Bohužiaľ veľa z vás túto možnosť aj čiastočne využilo – popisy niekedy iba jednoducho popísali, čo program robí, ale neobsahovali zdôvodnenie, prečo je takýto spôsob vypočtu ten správny a vždy vedie ku dobrému výsledku, poprípade

popis silne používal technológiu handwavingu[4], aby sa vyhol nejasným oblastiam. Pokiaľ používate netriviálne matematické úvahy, vždy sa oplatí presne ukázať, prečo to funguje. Raz za čas to dokonca ušetrí chybu v programe.

## Listing programu:

```
var   N,i,asum,bsum,add:integer;
  a:array[0..1000] of integer;
  b:array[0..1000] of integer;

function dalsie_b (poz:Integer):Integer;
begin
  dalsie_b:=b[poz mod N] + a[(poz+2) mod N] - a[(poz+1) mod N];
end;

function min3 (offset:Integer):Integer;
var i:Integer;
begin
  min3:=b[offset];
  for i:=0 to (N div 3)-1 do begin if b[3*i+offset]<min3 then min3:=b[3*i+offset];
end;
end;

procedure add3(offset, c:Integer);
var i:Integer;
begin
  for i:=0 to (N div 3)-1 do Inc(b[3*i+offset], c);
end;

begin
  asum:=0;  bsum:=0;
  readln(N);
  for i:=0 to N-1 do begin read(a[i]); Inc(asum, a[i]); end;
  if (N mod 3) > 0 then
  begin
    b[0]:=1;
    for i:=0 to N-2 do
    begin
      b[(3*i+3) mod N]:= dalsie_b(3*i);
      Inc(bsum, b[(3*i+3) mod N]);
    end;
    add:=((asum div 3) - bsum) div N;
    for i:=0 to N-1 do Inc(b[i],add);
  end
  else
  begin {N mod 3  = 0}
    b[0]:=0; b[1]:=0; b[2]:=0;
    for i:=0 to (N div 3)-2 do
    begin
      b[(3*i+3) mod N]:= dalsie_b(3*i);
      b[(3*i+4) mod N]:= dalsie_b(3*i+1);
      b[(3*i+5) mod N]:= dalsie_b(3*i+2);
    end;
    if (min3(0)<1) then add3(0,-min3(0)+1);
    if (min3(1)<1) then add3(1,-min3(1)+1);
    add3(2,a[1]-b[0]-b[1]-b[2]);
  end;
```

[4] http://en.wikipedia.org/wiki/Handwaving

```
  for i:=0 to N-1 do
    if (b[i] + b[(i+1) mod N] + b[(i+2) mod N] <> a[(i+1) mod N]) or (b[i]<1)
then
    begin
      writeln('Falzifikat'); halt;
    end;
  for i:=0 to N-1 do write(b[i],' '); writeln;
end.
```

opravoval MMx
(max. 15 bodov)

# 6. Obchod

Priamočiare riešenie tohoto príkladu mohlo vyzerať nasledovne: výsledok bude nejaká cena od 0 po maximálnu cenu, ktorú je niekto ochotný zaplatiť vrátane (z ceny menšej ako 0 zisk nebude a za väčšiu ako maximálnu cenu si to nikto nekúpi). Tak si pre každé číslo z tohoto intervalu vypočítajme, aký zisk pri takejto cene dosiahneme. To spravíme jednoduchým prechodom cez všetkých zákazníkov a spočítaním, koľko na kom zarobíme. Problém takéhoto riešenia je jeho časová zložitosť, ktorá je exponenciálna z nasledujúceho dôvodu. Pozrime sa na vstupy $a_0 = 10$, $b_0 = 0$ a $a_0 = 100$, $b_0 = 0$. Predĺžili sme vstup o jeden znak a čas výpočtu sa predĺžil desaťnásobne, teda rast dĺžky výpočtu od dĺžky vstupu určuje exponenciálna funkcia. Vymyslieť takéto riešenie nedalo veľa práce, takže sa za neho dalo získať najviac 7 bodov.

K plnohodnotnému riešeniu bolo treba spraviť niekoľko pozorovaní. Napríklad, že výsledok bude niektorá z maximálnych cien, ktorú je niekto ochotný zaplatiť ($a_i$ pre niektoré $i$). Pozrime sa, čo by sa stalo, ak by to tak nebolo – pre niektorý vstup by sme mali riešenie, ktoré sa nezhoduje so žiadnym $a_i$ a my by sme zvýšili cenu na najbližšie $a_i$. Možno tým získame nových zákazníkov, na ktorých zarobíme nezápornú sumu. Určite tým ale nestratíme žiadneho existujúceho zákazníka, lebo sme cenu zvyšovali len po najbližšie $a_i$ (zákazníka by sme stratili, keby sme nejaké $a_i$ presiahli). Teda rovnakému alebo väčšiemu počtu zákazníkov naúčtujeme vyššiu sumu, čím zvýšime zisk. To je ale spor, keďže pôvodná cena z ktorej sme vychádzali už mala byť optimálna. Uvedené úvahy by bolo dobré spresniť a spomenúť prípad, keď je počet pôvodných zákazníkov 0 a nezvýši sa, pretože vtedy zisk ostane 0. Nebudeme si to ale komplikovať a necháme technické detaily na čitateľa.

Priamočiara implementácia toho algoritmu teda prejde všetky $a_i$, pre každé z nich prejde všetkých zákazníkov a sčíta jednotlivé zárobky. Časová zložitosť bude kvadratická. Za takéto riešenie sa dalo získať najviac 11 bodov.

Ďalšie zlepšenie pochádza z pozorovania, že ako postupne zvyšujeme cenu, zisk na každom zákazníkovi bude chvíľu nulový (lebo je príliš drahá doprava), potom začne rásť, až začne byť zasa nulový (lebo cena je pre neho príliš vysoká). Princíp riešenia sa nazýva *zametanie* a je užitočné si ho osvojiť, pretože sa často využíva.

Okamihy, keď získame alebo stratíme zákazníka, budeme volať udalosti. Na začiatku si vstup rozbijeme na takéto udalosti a uložíme ich do poľa. Toto pole utriedime podľa ceny, pri ktorej daná udalosť nastane. Potom ho prejdeme od začiatku a každú udalosť spracujeme. Udalosti teda spracovávame systematicky od najskorších a preto ich všetky akokeby *pozametáme* z jednej strany.

Ak je spracovávaná udalosť získanie zákazníka, zvýšime si počet zákazníkov a zvýšime aj celkové náklady, ktoré treba na dopravu k aktívnym zákazníkom. Ak stratíme zákazníka, obidve premenné o príslušné hodnoty znížime. V každom okamihu bude teda platiť $zisk = cena \cdot pocet\_zakaznikov - naklady$. Zakaždým, keď uvidíme väčší zisk ako doterajšie maximum, zapamätáme si ho spolu s cenou, pri ktorej sme ho dosiahli. Časová zložitosť je

$O(N \log N)$, ktorú potrebujeme na utriedenie poľa udalostí, následný prechod poľa sa do nej skryje. Pamäťová zložitosť je $O(N)$. Zo zákazníkov, pre ktorých platí $a_i \leq b_i$ nebudeme mať zisk pri žiadnej cene, takže ich odignorujeme už pri načítaní vstupu. Toto riešenie je vzorové, teda za 15 bodov.

Na maximálny počet bodov bolo treba aj poriadne čítať zadanie, ktoré hovorí „Ak je riešení viac, vypíšte najmenšie nezáporné.“ To znamená, že zo všetkých potenciálnych výsledkov je správny len najmenší a vypísať čokoľvek iné je chyba. Za takéto a podobné chyby a nedostatočný popis sa dalo stratiť po bode.

## Listing programu:

```
#include <stdio.h>
#include <stdlib.h>

#define MAXEVENTS 100

int n, events, udalost[MAXEVENTS][3];

// porovnaj dve udalosti len podla ceny, pri ktorej nastanu
int compar(const void *aa, const void *bb) {
  int *a = (int *) aa, *b = (int *) bb;
  return a[0] - b[0];
}

void pridaj(int cena, int naklady, int pocet) {
  udalost[events][0] = cena;
  udalost[events][1] = naklady;
  udalost[events][2] = pocet;
  events++;
}

int main() {
  scanf("%d", &n);
  for (int i=0; i<n; i++) {
    int a,b;
    scanf("%d %d", &a, &b);
    // pridaj len zakaznikov, z ktorych moze byt zisk
    if (a>b) {
      pridaj(a, -b, -1);
      pridaj(b, b, 1);
    }
  }

  // utried udalosti podla ceny, pri ktorej nastanu
  qsort(udalost, events, sizeof(udalost[0]), compar);

  int zakaznikov = 0, naklady = 0, max_cena = 0, max_zisk = 0;
  for (int i=0; i<events; i++) {
    // spracuj udalost
    naklady += udalost[i][1];
    zakaznikov += udalost[i][2];
    // ak sme spracovali vsetky udalosti patriace k tejto cene, vypocitaj zisk
    // (posledna udalost je strata zakaznika, takze nema vyznam sa nou zaoberat)
    if (i+1<events && udalost[i][0]!=udalost[i+1][0]) {
      // zakaznikov, ktori boli ochotni zaplatit najviac udalost[i][0]
      // sme uz stratili, takze aktualna cena je udalost[i+1][0]
      int cena = udalost[i+1][0];
      int zisk = zakaznikov*cena-naklady;
      if (zisk>max_zisk) {
```

```
        max_cena = cena;
        max_zisk = zisk;
      }
    }
  }

  printf("%d\n", max_cena);

  return 0;
}
```

opravoval Mic

## 7. Ondrejova hra

(max. 15 bodov)

Pri takomto nie príliš ťažkom príklade ma nepotešil počet riešiteľov. Nebolo ich veru veľa. A ako som ich bodoval? Za úplne správne riešenie som dal plný počet bodov. Za riešenia v $O(n^2)$ som dával 10 bodov. Ak niekto nemal vypočítaný obsah mnohouholníka, udeľoval som bonus $-5$ bodov. Za nesprávne riešenia som dával najviac 4 body. Ostatné riešenia dostali body niekde medzi tým.

Zoberme si najskôr prvú úlohu a to zrekonštruovať Ondrejov $n$-uholník. Jednoduché je to v tom prípade, ak si všimneme nasledovnú vec. Zoberme si body, ktoré majú rovnakú $y$-ovú súradnicu. Z nich si vyberme ten najľavejší. Keďže taký $n$-uholník existuje a v každom bode je uhol 90 stupňov, tak nasledujúci (ten najbližší vpravo) bod s rovnakou $y$-ovou súradnicou s ním musí byť spojený. Keď zoberieme tretí bod v poradí, tak ten musí byť spojený so štvrtým bodom v poradí. Keďže máme zaručené, že taký $n$-uholník existuje, tak potom nutne musí existovať práve párny počet vrcholov s rovnakou $y$-ovou súradnicou. Keď si teraz usporiadame vrcholy s najskôr podľa $y$-ovej súradnice a potom podľa $x$ súradnice, tak zistíme, že prvý vrchol musí byť spojený s druhým, tretí zo štvrtým, piaty zo šiestym... Všimnite si, že sa nestane, že by sme spojili dva body s rôznou $y$-ovou súradnicou.

Čuduj sa svete, ale rovnako to bude fungovať aj s horizontálnymi hranami, ibaže budeme triediť najskôr podľa $x$-ovej súradnice a až potom podľa $y$-ovej. Takto sme pre každý bod zistili jeho dvoch susedov, čiže sme zrekonštruovali celý mnohouholník. Ak budeme triediť nejakým rýchlim triedením, napríklad Quicksortom, tak to potom zložitosť tejto časti bude $O(n \log n)$.

Teraz sa zaoberajme počítaním obsahu. Na to sa pozrime na všetky horizontálne hrany mnohouholníka a navyše aj na smer, v ktorom sú na obvode (ak budeme prstom prechádzať po obvode, niektoré budeme prechádzať zľava doprava a ostatné v opačnom smere). Nech $i$-ta horizontálna hrana má súradnice $(x_{i1}, y_i) \to (x_{i2}, y_i)$. Ak $x_{i1} < x_{i2}$, tak hrana ide zľava doprava a v opačnom prípade ide hrana naopak. Nech $k$ je počet horizontálnych hrán. Zoberme teraz

$$|\sum_{i=1}^{k}(x_{i2} - x_{i1})y_i|$$

Ja tvrdím, že sa rovná ploche, ktorú hľadáme. Dôkaz tohto tvrdenia je jednoduchý a preto vám dám iba nasledovný hint: pozrite sa na ľubovoľný vertikálny rez mnohouholníkom a na horizontálne hrany, ktoré tento rez pretínajú.

Každá časť riešenia okrem triedenia trvá lineárne veľa času a preto zložitosť závisí od triedenia, čo máme v tomto prípade $O(n \log n)$. Potrebujeme si zapamätať akurát tak každý bod a trošku vecí navyše a preto pamäťová zložitosť je $O(n)$.

A teraz sa môžete pokochať vzorovým programom.

## Listing programu:

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

vector<pair<int,int> > Body;
vector<vector<int> > Hrany;
vector<int> ref;

struct cmpx{
    bool operator()(const int &a,const int &b)const{
return Body[a]<Body[b];
    }
};

struct cmpy{
    bool operator()(const int &a,const int &b)const{
if(Body[a].second==Body[b].second)return Body[a].first<Body[b].first;
return Body[a].second<Body[b].second;
    }
};

int main(){
    int N;
    cin>>N;
    for(int i=0;i<N;i++){
int a,b;
cin>>a>>b;
ref.push_back(i);
Body.push_back(make_pair(a,b));;
    }
    Hrany.resize(N);
    sort(ref.begin(),ref.end(),cmpy());
    for(unsigned int i=0;i<ref.size();i+=2){
Hrany[ref[i]].push_back(ref[i+1]);
Hrany[ref[i+1]].push_back(ref[i]);
    }
    sort(ref.begin(),ref.end(),cmpx());
    for(unsigned int i=0;i<ref.size();i+=2){
Hrany[ref[i]].push_back(ref[i+1]);
Hrany[ref[i+1]].push_back(ref[i]);
    }
    cout<<"Ondrejov n-uholnik ma body v tomto poradi: "<<endl;
    int vrchol=0,typ=0;
    vector<int> poradie;
    do{
poradie.push_back(vrchol);
cout<<Body[vrchol].first<<" "<<Body[vrchol].second<<endl;
vrchol=Hrany[vrchol][typ];
typ=(typ+1)%2;
    }while (vrchol!=0);
    int plocha=0;
    poradie.push_back(poradie[0]);
    for(int i=1;i<=N;i++){
plocha+=(Body[poradie[i-1]].first-Body[poradie[i]].first)*Body[poradie[i]].second;
    }
    cout<<"Plocha Ondrejovho n-uholnika je: "<<abs(plocha)<<endl;}
```

## 8. O televíznej hre

opravoval Zemčo
(max. 15 bodov)

Prvým krokom k úspechu pri riešení tohto príkladu bolo uvedomiť si, že nie vždy sa oplatí zobrať z každého úseku najdrahší kufrík. Druhým krokom k úspechu bolo sa podľa uvedeného pozorovania aj správať. Tretím krokom potom bolo uvedomiť si, že tento problém má štruktúru, ktorá umožnuje jeho riešenie pomocou dynamického programovania. Posledným krokom bolo domyslieť technické detaily a nakódiť to. Aké jednoduché!

Úvodom treba povedať, že toto nebol jednoduchý príklad, čomu zodpovedalo aj jeho čislo 8. Napriek všetkému ma ale počet riešení 5 veľmi nemilo prekvapil[5]. Ak je dôvod nízkeho počtu riešení fakt, že väčšina z vás si nevedela dať rady, tak dúfam, že teraz si všetci tento vzorák prečítate a pochopíte.

Než sa dostanem k riešeniu, vybavím dvoma vetami bodovanie: až na jednu výnimku prišli len vzorové riešenia, ktoré mi navyše nedali ani možnost prejaviť svoju krutosť a strhnúť body a tak získali plný počet – 15. K tomu zvyšnému poznamenám, že posielať riešenia, o ktorých už dopredu viem, že nie sú korektné, sa veľmi neoplatí. Radšej pomaly a správne, ako nesprávne, hoci o triedu rýchlejšie ako vzorák.

Vrhnime sa teda na riešenie. Zo zadania vieme, že náš súper Maroško ťahá tak, aby nahral na konci čo najviac. Našou úlohou je maximalizovať našu výhru. Inými slovami: máme určiť, aké výbery kufríkov budeme robiť, aby bola naša nahraná suma na konci čo najvyššia.

Prvé dva kroky z úvodného odstavca sú ľahké a nabádal k nim aj vstup v zadaní. Zaujímavejšie to bude s tým tretím. Kedy má problém vhodnú štruktúru, aby mohol byť riešený pomocou dynamického programovania? Je to vtedy, keď si môžeme definovať podproblémy tak, že pomocou ich riešení efektívne vyskladáme riešenie problému. Toľko encyklopedická definícia, poďme to nejako skúsiť aplikovať na náš príklad. Aby sa mi lepšie vysvetľovalo, v ďalšom texte si úlohu zjednoduším: predstavme si, že kufríky nie sú rozložené do kruhu, ale do radu.

Rozhodujúce pozorovanie je, že keď mám nejaký rad, z ktorého vyberám kufrík a nejaký si vyberiem, nechám súperovi jeden alebo dva menšie rady, z ktorých vyberá on, pričom je vlastne v podobnej situácii. Ďalej je dôležité, že úseky kufríkov, ktoré ostali, sú nezávislé a preto ich môžeme riešiť osobitne. Presnejšie, ak stojíme pred dvoma úsekmi, tak najlepšie riešenie dostaneme, ak optimálne vyriešime prvý a zároveň optimálne vyriešime druhý. Práve toto budú naše podproblémy.

Už v tomto momente sme schopní vymyslieť riešenie. Spočívalo by v rekurzívnej funkcií $f$, ktorá by dostala na vstupe začiatok a koniec úseku kufríkov zo vstupu, ktorý máme riešiť a vrátila, koľko najviac dokáže hráč stojaci pred týmto úsekom nahrať za kufríky v tomto úseku. Ak by bol úsek dlhý jeden, mala by to ľahké. V opačnom prípade by pracovala tak, že by vyskúšala všetky možné kufríky v úseku a skúsila ich zobrať. Po ich odobratí by nám ostal jeden alebo dva úseky, na ktoré by sa funkcia rekurzívne zavolala a zistila by, koľko najviac vie v týcho úsekoch nahrať optimálne ťahajúci súper. Pre vybratý kufrík by potom získané peniaze boli rovné hodnote tohto kufríka plus hodnotám, ktoré by nám optimálne ťahajúci súper dovolil nahrať vo zvyšných radoch. Vyskúšaním všetkých možných kufríkov v rade získame maximálnu nahrateľnú hodnotu na tento úsek. Označme $a_i$ hodnotu kufríka na $i$-tom mieste. Potom je formálny zápis uvedenej funkcie nasledovný:

[5] A nielen mňa, ale aj celé KSP.

$$f(zac, kon) = \max(a_{zac} + (\sum_{k=zac+1}^{kon} a_k - f(zac+1, kon)), a_{kon} + (\sum_{k=zac}^{kon-1} a_k - f(zac, kon - 1)), \max_{zac<i<kon} : a_i + (\sum_{k=zac}^{i-1} a_k - f(zac, i-1)) + (\sum_{k=i+1}^{kon} a_k - f(i+1, kon))).$$

Prvý člen je hodnota pri vybraní prvého kufríka, druhý je hodnota pri vybratí posledného kufríka a ten tretí je maximum hodnôt, ktoré nahráme pri výberoch stredných kufríkov. Celý program by spravil iba toľko, že by vrátil hodnotu $f(1, N)$.

Takže by sme mali prvé riešenie, ktoré by ale trvalo exponencionálny čas. Jeho hlavný problém som sa pokúsil zachytiť na nasledovnom príklade: uvažujme rad kufríkov 7 5 5 7. Funkcia $f$ by začala tým, že by skúsila zobrať 7 a rekurzívne by sa zavolala na úsek 5 5 7. V tomto volaní by časom vyskúšala vziať zostávajúce číslo 7 a rekurzívne by sa zavolala na úsek 5 5. Tam by niekedy neskôr dorátala a povynárala sa späť hore k úseku 7 5 5 7. Niekedy neskôr by vyskúšala zobrať aj druhé číslo 7, čím by sa rekurzívne zavolala na úsek 7 5 5 a tam by hneď prvé skúsila vziať prvú sedmičku a opäť sa zavolala na úsek 5 5. Lenže tu už raz bola a zjavne nepríde na nič iné ako pri zavolaní predtým. No a iste si viete predstaviť situáciu, pri ktorej sa ten istý úsek ráta a tá istá hodnota vracia veľmi vela krát. Posledný krok k riešeniu je odstrániť tento nedostatok a spraviť tzv. *memoizáciu backtracku*. Miesto toho, aby sme niečo rátali viackrát si to jednoducho zapamätáme a keď to najbližsie budeme potrebovať, tak to budeme mať v konštantom čase. Všetkých možných úsekov je $O(N^2)$, pretože ku každému začiatku máme najviac $N$ koncov. Spravíme si dvojrozmernú tabuľku, kde si pamätáme pre každý úsek hodnotu funkcie $f$ na tom úseku, teda najvyššiu hodnotu, ktorú je schopný uhrať hráč, ked má pred sebou taký úsek. Keď stojíme pred úlohou nejaké políčko tabuľky vyplniť, tak si postupne v tabuľke hľadáme všetky podúseky, ktoré naša funkcia potrebuje. Ak sú už vyrátané, hodnotu môžeme použiť. Ak nie, tak sa rekurzívne na daný podúsek zavoláme. Takto nás každý úsek stojí čas úmerný jeho dĺžke a každý rátame len raz. Preto bude výsledný čas $O(N^3)$.

Dá sa na to ísť aj troška inak. Tabuľka sa dá rátať *zospodu*, teda od najmenších dĺžok úsekov smerom k najvyšším. Máme kde začať: úseky dlhé jedna su triviálne. Vezmime si, že chceme vyrátať políčko $[i, j]$. Na to potrebujeme len údaje o kratších úsekoch. A tieto už máme vyplnené. Toto je naozaj len prevrátaný postup k funkcii $f$ a výsledok úlohy by bola hodnota v tabuľke na políčku $[1, N]$. Oba uvedené postupy sa vyznačujú krátkym a elegantným kódom, náš vzorák je implementáciou druhého z nich.

Posledná vec: čo s tým, že kufríky sú do kruhu? Jedna možnosť je jednoducho funkciu rozšíriť. Nebudú existovať len úseky, kde je začiatok menšie číslo ako koniec, ale aj tie ostatné. Algoritmu to nikde neublíži. Iná možnosť je spraviť to ako náš vzorový program – úseky v tabuľke si indexovať nie podla začiatku a konca, ale podľa dĺžky a začiatku. Pôsobí to možno ešte troška intuitívnejšie. Schválne, v ktorom políčku tabuľky sa bude nachádzať výsledok?

## Listing programu:

```
#include <cstdio>
#define MAXN 100
int sucty[MAXN][MAXN],zisk[MAXN][MAXN],N;

int main(){
  scanf("%d ",&N);
  for (int i=0;i<N;i++){
    scanf("%d ",&sucty[1][i]);
    sucty[0][i]=0;
    zisk[0][i]=0;
    zisk[1][i]=sucty[1][i];
  }
```

```
  for (int i=2;i<=N;i++)for(int j=0;j<N;j++){
    sucty[i][j]=sucty[i-1][(j+1)%N]+sucty[1][j];
  }
  for (int i=2;i<=N;i++){//i - dlzka useku
     for (int j=0;j<N;j++){//j - zaciatok useku
        int best = -1;
        for (int k=j;k<j+i;k++){//k - ktory kufrik vezmeme
           int nahram = sucty[1][k%N]+(sucty[k-j][j]-zisk[k-j][j])+
                        (sucty[(j+i-k-1)%N][(k+1)%N]-zisk[(j+i-k-1)%N][(k+1)%N]);
           if (best < nahram){best = nahram;}
        }
        zisk[i][j]=best;
     }
  }
  printf("%d\n",zisk[N][0]);
  return 0;
}
```

## 9. Tóny ľubozvučných melódií

opravoval Lukáš
(max. 15 bodov)

Tento príklad riešili len dvaja riešitelia. Riešenie Peťa Ondrúšku boli jediné správne a zároveň dostatočne rýchle. Poďme sa naňho pozrieť.

Najprv sa zamyslíme nad jednoduchým faktom: každý tón bude spievať iba jeden spevák. Predstavme si, že máme nejaké optimálne riešenie, v ktorom nejaký tón spievajú viacerí speváci (po častiach). V momente, keď sa nejakí dvaja v tóne striedajú, majú rovnaký tón hlasu. Ak by sme ich od tohto okamihu vymenili, teda prvého nechali spievať tóny toho druhého a naopak, dostali by sme rovnako dobré riešenie. Opakovaním tohto postupu dosiahneme, že každý tón odspieva práve jeden spevák.

Teraz si pomocou tónov zo vstupu vyrobme graf. A nie hocijaký, ale bipartitný. Bipartitný graf je taký, že jeho vrcholy vieme rozdeliť na dve časti (partície) také, že hrany idú iba medzi vrcholmi týchto partícií. V našom prípade budú jednu partíciu tvoriť začiatky tónov a druhú konce tónov. Hrana z konca tónu $a$ ku začiatku tónu $b$ pôjde práve vtedy, ak po dospievaní $a$ stíhame zmeniť tón hlasu tak, aby sme odspievali tón $b$. Vrcholy obidvoch partícií budeme medzi sebou párіť. Každý vrchol môže mať len jeden pár a navyše s týmto párom musí byť spojený hranou. Na začiatku priradíme každému tónu jedného speváka, takže dokopy ich budeme mať $N$. Všiminime si, že ak spárujeme nejaký koniec tónu s iným začiatkom tónu, môžeme ušetriť jedného speváka, keďže on dokáže odspievať obidva tóny. Keďže nás zaujíma najmenší potrebný počet spevákov, potrebujeme v našom bipartitnom grafe nájsť najväčšie párenie.

Ukážeme si nejaké zaujímavé fakty o párení v bipartitnom grafe. Najprv si zadefinujeme zlepšujúcu cestu: je to taká cesta, ktorá sa začína v nespárovanom vrchole v jednej partícii, končí sa v nespárovanom vrchole v druhej partícií a striedajú sa na nej nespárované hrany s hranami z párenia. Všimnime si, že takáto cesta má nepárnu dĺžku. Začali sme totiž v jednej partícii a skončili v druhej. Zlepšujúca cesta sa začína aj končí nespárovanou hranou, čím sa dostávame k tomu, prečo sa volá zlepšujúca. Ak by sme totiž na tejto ceste vymenili spárované hrany s nespárovanými, dostali by sme korektné párenie a navyše počet hrán v párení by sa zvýšil o jedna.

Je celkom zrejmé, že ak máme najväčšie párenie, tak určite v grafe neexistuje zlepšujúca cesta, lebo by sme toto párenie vedeli zväčšiť. Opačná implikácia platí tiež: ak neexistuje zlepšujúca cesta, párenie je najväčšie. Ukážeme to sporom. Nech $M$ je najväčšie párenie a $K$ nie je najväčšie párenie. Ukážeme, že potom pre párenie $K$ existuje zlepšujúca cesta. Nechajme

v grafe iba hrany, ktoré sú práve v jednom z párení $M$ alebo $K$, ostatné zahoďme. Hrany z $M$ označme červenou farbou a hrany z $K$ modrou farbou. Vytvorili sa nám 4 typy útvarov: cykly, cesty s rovnakým počtom červených a modrých hrán, cesty s prevládajúcimi červenými hranami a cesty s prevládajúcimi modrými hranami. Každý cyklus obsahuje rovnaký počet modrých a červených hrán, keďže je párnej dĺžky a nemôžu byť vedľa seba dve červené alebo dve modré hrany. Keďže červených hrán je viac, musí existovať viac ciest takých, že na nich prevládajú červené hrany. No ale každá takáto cesta je zlepšujúca pre párenie $K$ (červené hrany vymeníme za modré). Dostali sme spor, čím sme dokázali tvrdenie.

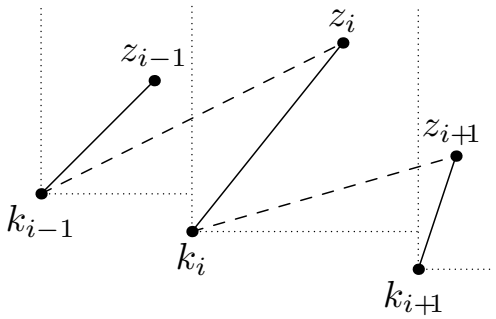
Vráťme sa teraz späť k našim tónom. Najprv si pomôžeme jedným trikom, aby sme mali ľahšiu robotu. Každý tón si predstavíme ako úsečku v rovine, pričom $x$-ová os je čas a $y$-ová os výška tónu. Všimnime si, že z konca jedného tónu vie spevák prejsť na začiatok iného tónu práve vtedy, ak je orientovaný uhol medzi týmito dvoma bodmi medzi 45 a $-45$ stupňov. Keď teraz otočíme body o 45 stupňov okolo bodu $[0,0]$ proti smeru chodu hodinových ručičiek, budeme to mať trochu ľahšie. Z konca tónu na začiatok iného sa dá prejsť práve vtedy, ak je začiatok napravo a hore od konca tónu. Teda hrana medzi začiatkom a koncom ide iba ak koniec má menšiu $x$-ovú aj $y$-ovú súradnicu ako koniec. Otočenie bodov o 45 stupňov spravíme tak, že bod so súradnicami $[x,y]$ premiestnime na súradnice $[x-y, x+y]$ (týmto sa nám natiahnu vzdialenosti o $\sqrt{2}$, ale to nám nevadí).

Po tejto transformácii súradníc zotriedime body najprv podľa $x$-ovej a potom podľa $y$-ovej súradnice. Potom postupne prechádzame body zľava doprava a keď narazíme na začiatok tónu $T$, nájdeme taký nespárený koniec tónu, ktorý je naľavo a dole od tónu $T$ (týmto zaručíme, že ich vieme spáriť), ale zároveň má najvyššiu $y$-ovú súradnicu. Ukážeme si, že týmto postupom nájdeme najväčšie párenie. Predpokladajme, že po skončení algoritmu nemáme najväčšie párenie. Potom nutne existuje zlepšujúca cesta. A keď existuje zlepšujúca cesta, potom existuje aj najkratšia zlepšujúca cesta. Označme si túto cestu $z_1k_1 \dots z_mk_m$ ($z_i$ sú začiatky tónov a $k_i$ sú konce tónov). Na tejto ceste sa striedajú hrany ktoré nie sú v párení $(z_ik_i)$ s hranami z párenia $(k_iz_{i+1})$ a vrcholy $z_1$ a $k_m$ sú nespárené.

Nech $m = 1$, potom hrana $z_1k_1$ nie je v párení, takže pre vrchol $z_1$ existuje nespárený vrchol vľavo dole od neho, čo je spor s tým, ako funguje náš algoritmus. Nech teda $m > 1$. Všimnime si, že napravo hore od vrcholu $k_i$ môžu byť spomedzi začiatkov len vrcholy $z_i$ a $z_{i+1}$, lebo ak by sa tam nachádzal aj vrchol $z_j$ a $j < i$ alebo $j > i+1$, tak medzi $k_i$ a $z_j$ je hrana a vynechaním úseku od $k_i$ po $z_j$ dostaneme kratšiu zlepšujúcu cestu. To sa však nedá, lebo táto cesta je najkratšia.

Všimnime vrcholy $z_{m-1}, k_{m-1}, z_m, k_m$. Vrchol $k_m$ je nespárený, preto $k_{m-1}$ musí mať vyššiu $y$-ovú súradnicu ako $k_m$. Ak by to tak nebolo, $z_m$ by sme spárili s $k_m$, pretože tak pracuje náš algoritmus. Navyše $k_{m-1}$ musí mať nižšiu $x$-ovú súradnicu ako $k_m$, lebo inak by $z_{m-1}$ bolo vpravo hore od $k_m$, čo sa nemôže stať. Potom aj $z_{m-1}$ musí mať vyššiu $y$-ovú súradnicu ako $k_m$. Ale $z_{m-1}$ sa nemôže nachádzať vpravo hore od $k_m$, takže sa nachádza vľavo hore od $k_m$. Zhrnuté a podčiarknuté: $k_{m-1}, z_{m-1}$ sa nachádzajú vľavo hore od $k_m$.

Teraz ukážeme, že podobná situácia nastane aj pre vrcholy $z_{i-1}, k_{i-1}, z_i, k_i$. Budeme dokazovať matematickou indukciou od konca cesty, pričom bázu indukcie (koniec cesty) sme už dokázali. Nech teda tvrdenie platí pre $i+1$: body $k_i, z_i$ sú vľavo hore od bodu $k_{i+1}$. Bod $k_{i-1}$ je spárený s $z_i$ a v čase, keď sme párili vrchol $z_i$ bol $k_i$ tiež nespárený. Takže $k_{i-1}$ má vyššiu $y$-ovú súradnicu ako $k_i$, a preto aj $z_{i-1}$ má vyššiu $y$-ovú súradnicu ako $k_i$. Zároveň $z_{i-1}$ nemôže byť vpravo hore od $k_i$, takže $z_{i-1}$ aj $k_{i-1}$ sú vľavo hore od $k_i$.




Vďaka dokázaným tvrdeniam platí, že $z_1, k_1$ sú vľavo hore od $k_2$ a navyše $z_1$ je naľavo od $z_2$. V čase, keď sme spracovávali $z_1$ a nechali ho nespárené, muselo zostať aj $k_1$ nespárené a to je spor s tým, ako pracuje náš algoritmus. Zlepšujúca cesta teda neexistuje a náš algoritmus nájde najväčšie párenie.

Dokázali sme si korektnosť algoritmu, môžeme sa vrhnúť na implementáciu. Na začiatku potrebujeme zmeniť súradnice a potom zotriediť body. To zvládame v čase $O(N \log N)$. Teraz body prechádzame zľava doprava a v nejakej dátovej štruktúre si pamätáme ešte nespárené konce tónov. Pre každý začiatok potrebujeme nájsť nespárený koniec s najvyššou $y$-ovou súradnicou menšou ako má spracovávaný začiatok tónu. Toto dokážeme zistiť pomocou vyváženého binárneho stromu v čase $O(\log N)$. Celková časová zložitosť je teda $O(N \log N)$ a pamäťová zložitosť $O(N)$. Implementácia využíva na tieto operácie štruktúru `multiset` z STL.

**Poznámka:** V zadaní sa vyskytla nešťastná formulácia, že spevák môže raz za časovú jednotku zmeniť tón hlasu. Toto robí príklad zbytočne zložitejším. V našom riešení uvažujeme také zadanie, že spevák za $x$ časových jednotiek môže zmeniť tón hlasu o $x$. Dôkazy a algoritmus v pôvodnom zadaní by vyzerali podobne, museli by sme však ošetrovať veľa špeciálnych prípadov. Už aj tak je tento vzorák dosť dlhý.

## Listing programu:

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

struct Bod
{
    int x, y;
    bool zaciatok;
};
bool operator<(const Bod &a, const Bod &b)
{
    if (a.x != b.x) return a.x < b.x;
    if (a.y != b.y) return a.y < b.y;
    return a.zaciatok < b.zaciatok;
}

int main()
{
    int n; cin >> n;
    vector<Bod> b;
    for (int i = 0; i < n; i++)
    {
        Bod a; int dlzka;
        cin >> a.x >> dlzka >> a.y;
```

```
        a.zaciatok = true;
        b.push_back(a);

        a.x += dlzka;
        a.zaciatok = false;
        b.push_back(a);
    }

    for (int i = 0; i < 2 * n; i++)
    {
        int nx = b[i].x - b[i].y, ny = b[i].x + b[i].y;
        b[i].x = nx;
        b[i].y = ny;
    }
    sort(b.begin(), b.end());

    multiset<int> konce;
    int pocet = n;
    for (int i = 0; i < 2 * n; i++)
    {
        if (b[i].zaciatok)
        {
            multiset<int>::iterator it = konce.upper_bound(b[i].y);
            if (it != konce.begin())
            {
                pocet--;
                konce.erase(--it);
            }
        }
        else konce.insert(b[i].y);
    }

    cout << "Treba " << pocet << " spevakov" << endl;
}
```

opravoval kuko
(max. 15 bodov)

## 10. – je v morzeovke T

Tento príklad imho nebol ťažký – povedal by som priam priamočiary. Vstupný reťazec si môžeme predstaviť ako graf – jednotlivé bodky a čiarky budú vrcholy a medzi dvoma vrcholmi bude hrana vtedy, keď v slovníku existuje slovo, ktoré sa medzi týmito vrcholmi nachádza. Presnejšie, nech $x_1x_2\cdots x_L$ je reťazec bodiek a čiarok. Predstavme si graf s vrcholmi od 0 po $L$, pričom hrana pôjde z $i$ do $j$, ak sa podreťazec $x_{i+1}x_{i+2}\cdots x_j$ nachádza v slovníku (prípadne, ak má viacero slov rovnaký zápis v morzeovke, cena hrany bude počet takých slov v slovníku; napr. `MILENKA` a `ZASKRT` zo zadania budú tvoriť hrany s cenou 2).

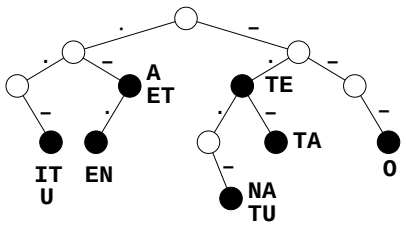
Keď problém formulujeme takto, rôzne otázky nám zodpovie teória grafov. Napríklad, dá sa nejaká veta zakódovať na tento reťazec? Dá sa vtedy, ak existuje cesta z vrcholu 0 do $L$. Koľko viet (zo slov zo slovníka) sa zakóduje na daný reťazec? To je počet ciest z 0 do $L$. Ktorá veta, čo sa zakóduje na daný reťazec, je najkratšia? Ktorá je najdlhšia? Nuž to je najkratšia/najdlhšia cesta z 0 do $L$ (čo sa týka počtu hrán). Pripomeňme, že náš graf je acyklický, všetky hrany sú „zľava doprava“, takže všetky úlohy (aj najdlhšia veta) sú ľahko riešiteľné. Zafunguje dynamické programovanie. Pôjdeme zľava doprava a pre každé $k$ si budeme počítať hodnotu $P[k]$ – koľkými spôsobmi sa dá reťazec $x_1x_2\cdots x_k$ rozložiť na postupnosť slov zo slovníka. Platí, že $P[0] = 1$ a $P[k]$ spočítame ako súčet $P[k - \text{length}(y)] \cdot \text{count}(y)$, kde $y$ sa nachádza v reťazci $x$, končí na pozícii $k$ (teda začína na

pozícií $k - \text{length}(y) + 1$) a v slovníku sa $y$ nachádza $\text{count}(y)$ krát (viacero slov môže mať v morzeovke rovnaký zápis). Alebo v grafovej terminológii: cez všetky hrany $j$–$k$ robíme súčet $P[j]\times$cena hrany.

Jediný problém je uvedený graf zostrojiť (nie nutne explicitne: hrany si nemusíme pamätať, stačí ich pre každý vrchol vedieť vypočítať). A to je úloha tiež pomerne štandardná.

Najjednoduchšie riešenie je nájsť postupne všetky výskyty každého slova v slovníku tak nejak naivne. $N$ slov dĺžky $O(D)$ sa môže nachádzať na $L$ rôznych pozíciách – dostávame algoritmus v čase $\Theta(LND)$, ktorý, ako už bolo deklarované v zadaní, nedostal veľa bodov. Trochu lepšie je použiť trochu lepšie vyhľadávanie, napr. KMP. Jedno slovo potom nájdeme v čase $O(L+D)$ namiesto $O(LD)$ a výsledný čas bude $O(N(L+D))$. V skutočnosti nič moc. Ak chceme niečo lepšie, všetky slová v slovníku musíme hľadať naraz. Nebudeme teda každé slovo hľadať v stringu[6], ale skúšať, či sa nejaké úseky stringu nachádzajú v slovníku.

Na to je dobre si slovník reprezentovať ako písmenkový strom. Môžeme si ho predstaviť ako binárny strom (lebo máme iba dve písmená), kde hrana vľavo predstavuje bodku a hrana vpravo čiarku. Takto každému vrcholu prislúcha nejaká postupnosť bodiek a čiarok (dostaneme ju, keď pôjdeme postupne z koreňa do daného vrcholu, pričom si čítame značky na hranách). V strome vyznačíme tie vrcholy, ktoré patria do slovníka. Príklad pre slovník {`A`, `EN`, `ET`, `IT`, `NA`, `O`, `TA`, `TE`, `TU`, `U`} je na obrázku vľavo:




Vrcholy, ktoré patria do slovníka, sú zafarbené na čierno. V skutočnosti si nemusíme pre každý vrchol pamätať, aké slová (v latinke) predstavujú, stačí ich počet. Ak chceme v tomto slovníku nájsť nejaké slovo, začneme v koreni, čítame slovo a podľa toho, či je písmeno bodka alebo čiarka ideme doľava alebo doprava. Napríklad ak hľadáme `-..-`, ideme z koreňa doprava, doľava, doľava a doprava. Dostávame sa do čierneho vrcholu, takže vieme, že také slovo je v slovníku, navyše si pamätáme, že sú dve také. Keď chceme pridať nejaké slovo, ideme po rovnakej ceste ako pri hľadaní, pričom ak taký vrchol neexistuje, tak ho vytvoríme. Posledný vrchol označíme (resp. zvýšime počet slov, ktoré tu končia).

Už len využitím tohto stromu dostaneme algoritmus s časom $O(D(N+L))$ (čo je oveľa lepšie ako $O(N(L+D))$): Stačí „naivne“ skúšať všetky možné začiatky $p$ a zisťovať, či sa nejaké zo slov $x_p x_{p+1} x_{p+2} \cdots$ nenachádza v slovníku (ak áno, máme hranu). Ako to zistíme? Pre každé $p \leq L$ sa postavíme do koreňa, čítame písmená $x_p x_{p+1} x_{p+2} \cdots$ a hýbeme sa podľa nich, až kým nevýjdeme zo stromu von (kým nenarazíme na nilový pointer), pritom vždy keď sa dostaneme do čierneho vrcholu, zaznačíme si hranu. Keďže slová v slovníku majú dlžku maximálne $4D$ (maximálne $D$ písmen a jedno písmeno môže mať do 4 bodiek/čiarok), toto vyhľadávanie spravíme v čase $O(DL)$.

Toto sa dá ešte trochu zlepšiť na $O(DN+L+z)$, kde $z$ je počet výskytov slov v slovníku danom reťazci. Takýto algoritmus vymysleli ujo Alfred Aho a teta Margaréta Corasicková. Myšlienka je rovnaká ako pri KMP algoritme a pokúsim sa ju vysvetliť cez Mišofových lemmingov (pozri `http://ksp.sk/~misof/MFnotes/0002.pdf`). Na hľadanie slov v stringu sa môžeme pozerať ako na takúto hru: vrcholy nášho stromu (slovníku) sú miestnosti, ktoré sú pospájané chodbami (hranami). V každej miestnosti sú najviac dvoje dvere – jedny označené bodkou, jedny čiarkou. Každú sekundu prečítame z textu jedno písmeno – bodku alebo

[6] hoci aj toto by sa dalo dokopať do čohosi rýchleho, ak by sme si spravili sufixový strom

čiarku, čím sa všetky dvere takto označené otvoria a ostatné zavrú. Lemmingovia sa pohybujú týmto stromom z koreňa smerom nadol rýchlosťou 1 miestnosť za sekundu. Ak je lemming v nejakej miestnosti a otvoria sa mu dvere (najviac jedny), za 1 sekundu dobehne do nasledujúcej miestnosti. Ak sa mu žiadne dvere neotvoria, zahynie (v miestnostiach sú pasce). Ak sa však dostane do označeného vrcholu, vyhrali sme – našli sme slovo (zatiaľ sa uspokojme s tým; ako nájsť všetky výskyty všetkých slov si povieme neskôr).

Ako dosiahneme, aby sa nejaký lemming dostal do cieľa (označeného vrcholu)? Nuž keďže dopredu nevieme, kde sa nejaké slovo nachádza, aby nám žiadne neuniklo, budeme z koreňa každú sekundu púšťať jedného lemminga. Takto možno veľa z nich zahynie, ale určite nám žiadny výskyt neunikne.

Všimnite si, že na každej úrovni stromu sa môže vyskytovať najviac 1 lemming! Keby sme niečo takéto priamo simulovali, dostali by sme algoritmus s časom $O(D(N + L))$. Všimnime si ale pozíciu najhlbšieho lemminga. Ak je vo vrchole $v$ v hĺbke $h$, vieme povedať posledných $h$ písmen textu a teda aj pozície všetkých ostatných lemmingov s menšou hĺbkou. Teda pozícia najhlbšieho lemminga celkom určuje postupnosť všetkých. Namiesto posúvania všetkých lemmingov nám stačí posunúť toho najhlbšieho. Čo sa stane, keď najhlbší zakape? Pre každý vrchol $v$ si spočítame hodnotu $f(v)$ – ak je $v$ najhlbší lemming, v ktorom vrchole je druhý najhlbší. Toto si vieme predpočítať v lineárnom čase od veľkosti slovníka (pozri stredný obrázok). Ak teda najhlbší lemming zakape, pozrieme sa na druhého ($f(v)$). Ak aj ten zakape, na tretieho ($f(f(v))$), atď. Napríklad ak bude v texte `-..-`, v slovníku na obrázku bude najhlbší lemming vo vrchole prislúchajúcom slovám `NA` a `TU`. Tento lemming jednoznačne určuje pozície ostatných, stačí sledovať šípky: nasledujúci lemming na tretej úrovni zodpovedá posledným trom písmenám `..-`, teda slovám `IT` a `U`. Posledné dve písmená sú `.-`, vrchol `A` a `ET`. Ďalší dvaja lemmingovia sú v koreni a jeho ľavom synovi. Keby v tejto situácií nasledovala bodka, prvý dvaja lemmingovia vo vrcholoch prislúchajúcich slovám `NA`, `TU` a `IT`, `U` by zakapali a tretí lemming by sa z vrcholu `A`, `ET` pohol do vrcholu `EN`. Keby nasledovala čiarka, zakapal by aj tretí lemming a štvrtý by sa pohol do vrcholu `A`, `ET`. Ak si pre každý vrchol a pre každý znak – bodku a čiarku – predpočítame, aká bude pozíciu najhlbšieho lemminga (ak bol predtým najhlbší lemming v danom vrchole a prišiel daný znak), dostaneme konečný automat zobrazený na obrázku vpravo – bodkované hrany zodpovedajú bodke, čiarkované čiarke. S takouto mašinou (ba priam mašinériou) už ľahko nájdeme výskyty slov v reťazci. Stačí sa na začiatku postaviť do koreňa a keď príde bodka ísť po bodkovanej, ak čiarka, po čiarkovanej čiare.

Ak chceme nájsť všetky výskyty všetkých slov, treba ešte pre každý vrchol predpočítať zoznam slov, ktoré sa končia na danej pozícií. Napríklad ak sa v texte nachádza `.-.`, najhlbší lemming bude vo vrchole prislúchajúcom slovu `EN`. V čiernom vrchole však bude aj druhý lemming – vo vrchole `TE`. Vo všeobecnosti zoznam slov, ktoré končia na danej pozícií získame tak, že zoberieme zoznam slov, ktoré prislúchajú danému vrcholu $v$ a spojíme ho so zoznamom slov $f(v)$. (Tu v skutočnosti potrebujeme iba zoznam počtu slov danej dĺžky.) Uvedomte si však, že aj keď je najhlbší lemming v bielom vrchole, iný môže byť v čiernom. Nuž a toto je teda algoritmus bežiaci v čase $O(ND + L + z)$, kde $z$ je počet výskytov slov v reťazci, resp. počet hrán v grafe. Slovník, resp. automat pripravíme v čase $O(ND)$ (resp. lineárne od jeho veľkosti, t.j. súčtu dĺžok slov), vyhľadanie je v čase $O(L + z)$ a v rovnakom čase počas vyhľadávania počítame tiež hodnotu $P[i]$.

V skutočnosti algoritmus nižšie, Ahov-Corasickovej algoritmus, nevyrába úplný automat – obrázok vpravo, predpočíta iba $f(v)$ – stredný obrázok. Aký je medzi tým rozdiel? Obrázok vpravo bude rýchlejší. V tomto prípade si ho môžeme dovoliť a je to aj lepšie riešenie. Avšak vo všeobecnosti, keby sme mali veľkú abecedu, napr. `A`–`Z`, museli by sme pre každý znak predpočítať, kam treba ísť a veľkosť automatu by sa nafúkla (krát počet písmen abecedy; v našom prípade je to 2, čo je úplne v pohode, ale 26 alebo 256 už stojí za zamyslenie). Na druhej strane stredný obrázok bude vždy lineárny od veľkosti slovníka (súčtu dĺžok slov).

Navyše časová zložitosť sa nezhorší (v každom kroku môžeme spraviť viacero krokov nahor a jeden krok nadol; avšak krokov nahor nemôže byť viac ako krokov nadol a tých je $L$).

## Listing programu:

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

#define MAXL 10000
#define MAXV 20*4*10000
#define REP(i,n) for (int i=0; i<(n); ++i)
#define SIZE(x) ((int)x.size())
#define NIL 0
#define ROOT 1

const char morse[26][5] = {".-", "-...", "-.-.", "-..", ".", "..-.", "--.",
  "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.",
  "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.."};

int g[MAXV][2], f[MAXV], out[MAXV], len[MAXV],
    cnt[MAXV], next[MAXV], tv=1, tl=1;
int P[MAXL+10];

string tomorse (string s) {
  string t;
  REP(i,SIZE(s)) t += morse[s[i]-'A'];
  return t;
}

void dictionary() {
  int n; cin >> n;
  while (n--) {
    string s; cin >> s; s = tomorse(s);
    int q=ROOT;
    REP(i,SIZE(s)) {
      int a = (s[i]=='-');
      if (g[q][a] == NIL) g[q][a] = ++tv;
      q = g[q][a];
    }
    if (!out[q]) out[q] = ++tl;
    len[out[q]] = SIZE(s);
    ++cnt[out[q]];
  }
}

void ACautomaton() {
  queue<int> Q;
  REP(a,2) {
    if (g[ROOT][a] == NIL) g[ROOT][a] = ROOT;
    int q = g[ROOT][a];
    f[q] = ROOT; Q.push (q);
  }
  while (!Q.empty()) {
    int q=Q.front(), r; Q.pop();
    REP(a,2) if ((r = g[q][a]) != NIL) {
      int v = f[q];
      while (g[v][a] == NIL) v = f[v];
      f[r] = g[v][a];
      if (out[r]==NIL) out[r] = out[f[r]];
```

```
      else next[out[r]] = out[f[r]];
      Q.push(r);
    }
  }
}

int main() {
  dictionary();
  ACautomaton();
  string s; cin >> s;
  P[0] = 1; int q=ROOT;
  for (int i=1; i<=SIZE(s); ++i) {
    int a = (s[i-1] == '-');
    while (g[q][a] == NIL) q = f[q];
    q = g[q][a];
    int j = out[q];
    while (j != NIL) {
      P[i] += cnt[j] * P[i-len[j]];
      j = next[j];
    }
  }
  cout << P[SIZE(s)-1] << endl;
  return 0;
}
```

# Výsledková listina po 1. kole kategórie KSP-Z

| | Meno a priezvisko | Škola | Trieda | 1 | 2 | 3 | 4 | 5 | Σ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Hozza Ján | Gym. Jura Hronca BA | 1 | 15 | 15 | 5 | 13 | 15 | 63 |
| 2 | Kuzma Tomáš | Gym. Alejová Košice | 3 | 15 | 15 | 13 | | 15 | 58 |
| 3 | Bačo Ladislav | Gym. Poštová Košice | 2 | 15 | 15 | 15 | | 11 | 56 |
| 4 | Kekely Michal | Gym. Varšavská Žilina - Vlčince | 1 | 15 | 15 | 2 | 13 | 7 | 52 |
| 4 | Rohár Pavol | Gym. Alejová Košice | 2 | 14 | 15 | 15 | 8 | | 52 |
| 6 | Proksa Ondrej | Gym. Párovská Nitra | 3 | 15 | 15 | 2 | 10 | 7 | 49 |
| 7 | Kováč Jakub | Gym. sv. Cyrila a Metoda Nitra | 3 | 15 | 15 | 5 | 2 | 9 | 46 |
| 8 | Macháč Juraj | Gym. Jura Hronca BA | 1 | 15 | 15 | 6 | | 9 | 45 |
| 9 | Vavrík Boris | Gym. Jura Hronca BA | 1 | 15 | 15 | 5 | | 9 | 44 |
| 10 | Csiba Peter | Škola pre mim. nadané deti BA | 3 | 15 | 15 | 13 | | | 43 |
| 11 | Špánik Marián | Gym. sv. Františka Žilina | 3 | 13 | 12 | 9 | 3 | 5 | 42 |
| 12 | Hašík Juraj | Gym. Grösslingová BA | 2 | 15 | 14 | 0 | | 10 | 39 |
| 13 | Phuong Mariana | Gym. Jura Hronca BA | 1 | 15 | 15 | 8 | | | 38 |
| 14 | Kučera Martin | Gym. Golianova Nitra | 1 | 15 | 11 | 1 | 7 | | 34 |
| 14 | Páleník Martin | Gymnázium Levice | 3 | 15 | 14 | 5 | | | 34 |
| 16 | Habovštiak Martin | Gym. Tvrdošín | 2 | 15 | 12 | 5 | | | 32 |
| 16 | Kikta Michal | Gym. Tatarku Poprad | 1 | 15 | 15 | 2 | | | 32 |
| 16 | Morvay Tomáš | Gym. Golianova Nitra | 1 | 15 | 15 | 0 | | 2 | 32 |
| 16 | Čerman Ondrej | Gym. Partizánske | 0 | 10 | 12 | | 10 | | 32 |
| 20 | Ort Miroslav | Gym. Pankúchova Bratislava | 3 | 11 | 11 | 1 | | 8 | 31 |
| 21 | Majdiš Mojmír | Gym. Dolný Kubín | 2 | 15 | | 15 | | | 30 |
| 21 | Pinter Martin | Gym. Jura Hronca BA | 1 | 15 | 15 | 0 | | | 30 |
| 21 | Špano Marek | Gym. Jura Hronca BA | 1 | 15 | 15 | | | | 30 |
| 24 | Baxová Katarína | Gym. Ľudovíta Štúra Trenčín | 3 | 11 | 15 | | | 2 | 28 |
| 25 | Holas Juraj | Gym. Jura Hronca BA | 1 | 13 | 14 | | | | 27 |
| 25 | Jursa Jakub | Gym. Alejová Košice | 3 | 14 | | 13 | | | 27 |
| 25 | Ziman Michal | Gym. Haličská Lučenec | 2 | 13 | 14 | | | | 27 |
| 28 | Iring Andrej | Gym. Jura Hronca BA | 1 | 15 | 11 | 0 | | | 26 |
| 28 | Kudláč Jakub | Gym. Bilíkova BA | 3 | 14 | 12 | | | | 26 |
| 28 | Stančiaková Katarína | Gym. Jura Hronca BA | 2 | 10 | 12 | 4 | | | 26 |
| 31 | Piovarči Michal | Gym. Hronská BA | 3 | 11 | 14 | | | | 25 |
| 31 | Sayedová Monika | Gym. Grösslingová BA | 4 | 10 | 15 | | | | 25 |
| 33 | Polák Lukáš | Gym. Jura Hronca BA | 2 | 11 | | 13 | | | 24 |
| 34 | Korbaš Rafael | Gym. Hronská BA | 1 | 11 | 8 | | | 4 | 23 |
| 34 | Macháč Matej | Gym. P. de Coubertina Piešťany | 2 | 10 | | 8 | | 5 | 23 |
| 34 | Mudrík Richard | Gym. J. M. Hurbana Čadca | 4 | 12 | 10 | 1 | | | 23 |
| 37 | Popovič Viktor | Gymnázium J.A. Raymana | 2 | 14 | | 8 | | | 22 |
| 38 | Hozzánková Hana | Gym. Jura Hronca BA | 2 | 15 | | 3 | | | 18 |
| 38 | Miklovič Tomáš | Gym. Nové Zámky | 2 | 13 | 5 | 0 | | | 18 |
| 40 | Sabo Michal | Gym. Jura Hronca BA | 4 | 15 | | | | | 15 |
| 41 | Chrást Lukáš | Gym. Golianova Nitra | 3 | | | 13 | | | 13 |
| 41 | Cudrák Miloš | Gymnázium | 4 | 12 | | 1 | | | 13 |
| 41 | Dávid Tóth | Gym. Daxnera V.n. Topľou | ? | | | 13 | | | 13 |
| 41 | Kentoš Michal | Gymnázium | 3 | | | 13 | | | 13 |
| 41 | Masár Juraj | Gym. Bilíkova BA | 1 | 11 | | 2 | | | 13 |
| 46 | Michalek Martin | Gym. P. de Coubertina Piešťany | 3 | 12 | | | | | 12 |
| 46 | Táňa Tóthová | Gym. Jura Hronca BA | 2 | 12 | | 0 | | | 12 |
| 48 | Haffner Oto | Gym. Hlohovec | 4 | 10 | | | | | 10 |
| 48 | Paulech Matej | Gym. P. de Coubertina Piešťany | 3 | | | 10 | | | 10 |
| 48 | Tomkovič Viktor | Gym. Jura Hronca BA | 3 | | | 10 | | | 10 |

| | Meno a priezvisko | Škola | Trieda | 1 | 2 | 3 | 4 | 5 | Σ |
|---|---|---|---|---|---|---|---|---|---|
| 48 | Žák Samuel | Gym. Rajec | 2 | 10 | | | | | 10 |
| 52 | Bubnár Michal | Gym. sv. Františka z Assisi V.n. Topľou | 3 | | | 9 | | | 9 |
| 53 | Kovaľ Anton | Gym. L. Stöckela Bardejov | 2 | 8 | | | | | 8 |
| 54 | Kučera Erik | Gym. Hlohovec | 4 | | | 5 | | | 5 |
| 55 | Sebechlebský Ján | Gym. Žiar nad Hronom | 1 | | | 1 | | | 1 |
| 56 | Kudláč Matúš | Gym. Bilíkova BA | 4 | | | 0 | | | 0 |
| 56 | Pásztor Bálint | Gym. maďarské Šahy | 3 | | | 0 | | | 0 |

# Výsledková listina po 1. kole kategórie KSP-O

| | Meno a priezvisko | Škola | Trieda | 4 | 5 | 6 | 7 | 8 | Σ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Fulla Peter | SPŠ Spišská Nová Ves | 3 | 15 | 15 | 15 | 15 | 15 | 75 |
| 2 | Hudec Tobiáš | Gym. Partizánske | 2 | 15 | 15 | 15 | 10 | 15 | 70 |
| 3 | Hozza Michal | Gym. Jura Hronca BA | 3 | 15 | 14 | 15 | 4 | 15 | 63 |
| 4 | Belan Tomáš | Škola pre mim. nadané deti BA | 2 | 12 | 15 | 10 | 10 | | 47 |
| 5 | Fekiač Jozef | Gym. Grösslingová BA | 3 | 13 | 7 | 14 | 10 | | 44 |
| 6 | Hozza Ján | Gym. Jura Hronca BA | 1 | 13 | 15 | 10 | | | 38 |
| 7 | Proksa Ondrej | Gym. Párovská Nitra | 3 | 10 | 7 | 10 | 4 | 1 | 32 |
| 8 | Kuzma Tomáš | Gym. Alejová Košice | 3 | | 15 | 11 | | | 26 |
| 9 | Csiba Peter | Škola pre mim. nadané deti BA | 3 | | | 11 | 12 | | 23 |
| 10 | Kekely Michal | Gym. Varšavská Žilina - Vlčince | 1 | 13 | 7 | | | | 20 |
| 11 | Macháč Juraj | Gym. Jura Hronca BA | 1 | | 9 | 10 | | | 19 |
| 12 | Kučera Martin | Gym. Golianova Nitra | 1 | 7 | | 11 | | | 18 |
| 13 | Petrucha Michal | Gym. Metodova BA | 3 | | 6 | 11 | | | 17 |
| 14 | Hojčka Michal | Gym. Partizánske | 3 | 0 | 3 | 11 | 2 | | 16 |
| 15 | Ondrúška Peter | SPŠ Dubnica nad Váhom | 4 | | | | | 15 | 15 |
| 16 | Ort Miroslav | Gym. Pankúchova Bratislava | 3 | | 8 | | 4 | | 12 |
| 17 | Bačo Ladislav | Gym. Poštová Košice | 2 | | 11 | | | | 11 |
| 17 | Kováč Jakub | Gym. sv. Cyrila a Metoda Nitra | 3 | 2 | 9 | | | | 11 |
| 19 | Hašík Juraj | Gym. Grösslingová BA | 2 | | 10 | | | | 10 |
| 19 | Čerman Ondrej | Gym. Partizánske | 0 | 10 | | | | | 10 |
| 21 | Vavrík Boris | Gym. Jura Hronca BA | 1 | | 9 | | | | 9 |
| 22 | Rohár Pavol | Gym. Alejová Košice | 2 | 8 | | | | | 8 |
| 22 | Špánik Marián | Gym. sv. Františka Žilina | 3 | 3 | 5 | | | | 8 |
| 24 | Pinter Martin | Gym. Jura Hronca BA | 1 | | | 6 | | | 6 |
| 25 | Macháč Matej | Gym. P. de Coubertina Piešťany | 2 | | 5 | | | | 5 |
| 26 | Korbaš Rafael | Gym. Hronská BA | 1 | | 4 | | | | 4 |
| 27 | Baxová Katarína | Gym. Ľudovíta Štúra Trenčín | 3 | | 2 | | | | 2 |
| 27 | Morvay Tomáš | Gym. Golianova Nitra | 1 | | 2 | | | | 2 |

# Výsledková listina po 1. kole kategórie KSP-T

| | Meno a priezvisko | Škola | Trieda | 8 | 9 | 10 | Σ |
|---|---|---|---|---|---|---|---|
| 1 | Ondrúška Peter | SPŠ Dubnica nad Váhom | 4 | 15 | 14 | 15 | 44 |
| 2 | Fulla Peter | SPŠ Spišská Nová Ves | 3 | 15 | | 6 | 21 |
| 3 | Hozza Michal | Gym. Jura Hronca BA | 3 | 15 | | | 15 |
| 3 | Hudec Tobiáš | Gym. Partizánske | 2 | 15 | | | 15 |
| 5 | Szabó Peter | SPŠE Nové Zámky | 3 | | 1 | 1 | 2 |
| 6 | Proksa Ondrej | Gym. Párovská Nitra | 3 | 1 | | 0 | 1 |