Korešpondenčný seminár z programovania
XXIII. ročník, 2005/06
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

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

Vzorové riešenia 2. kola zimnej časti, kat. Z

Milé naše riešiteľky!

Ták, ešte napíšem tento posledný úvod a môžem ísť spáááať... No, ako začať? Sme radi, že ste sa do KSP toľké zapojili, budúci polrok zavolajte aj kamarátky (a kamarátov) a bude nás ešte viac. Gratulujeme vám k dosiahnutým výsledkom, tie najlepšie z vás sa už môžu tešiť na pozvánku na sústredenie, ktoré bude v dňoch 13. až 19. marca pri Počúvadle.

A to je už na dnes od nás naozaj všetko.

Krásne sníčky...

KSPáci

1. Zanzibarské lamy

opravoval Bee
(max. 15 bodov)

Najprv menší úvod do písmeniek. Označíme počet lám N a počet operácii M. Drvivá väčšina riešení pozostávala z toho, že ste si v $O(N)$ spracovali informáciu o lamách a potom v $O(1)$ zisťovali, či sú lamy kamarátky. Také riešenia majú teda celkovú zložitosť $O(MN)$ Ak nič nechýbalo, tak za tento prístup sa dalo získať 12 bodov. Horšie zložitosti dostali primerane menej bodov – kvadratická 8 a kubická 7. Ak bol program nesprávny, viac ako 4 body som vám dať nemohol. Body som strhával za nedostatočné (alebo chýbajúce) popisy, poprípade chybičky v programoch.

Kto chcel dostať viac ako 12 bodov sa musel trochu viac posnažiť. Za program v zložitosti v $O(M \log N)$, ste si zaslúžili 14 bodov. No a napokon existuje optimálne riešenie v čase $O(M \log^* N)$. Za dôkaz zložitosti tohoto optimálneho riešenia by som pridal aj bonusové bodíky. Žiaľ, žiadny neprišiel. Čo je to $\log^*$ si povieme vo vhodnej chvíli.

Ale ešte predtým ako sa dostaneme k samotnému riešeniu, mám ešte jednu správu pre odpisovačov: ak už musíte odpisovať, tak aspoň zmeňte mená premenných a zarovnávanie, nech to tak nebije do očí. No nič, necháme odpisovačov nech sa nad sebou zamyslia a my ostatní sa vrhnime na riešenie.

Načrtnime si ako budeme postupovať. Zmeňme trochu terminológiu, aby sa nám lepšie pracovalo. Lamy budeme volať vrcholy. Každý vrchol bude mať svojho otca, čo bude odkaz na nejaký iný vrchol. Vrchol b je predkom vrcholu a a vrchol a je potomkom vrcholu b, ak existuje postupnosť vrcholov $a = v_1, v_2, \ldots, v_n = b$, kde vrchol v_i je otcom vrcholu $v_i - 1$. Strom je množina vrcholov, medzi ktorými existujú nejaké príbuzenské vzťahy. Konkrétne vrchol a patrí do stromu S, ak v S existuje iný vrchol b, ktorý je buď jeho predkom, alebo potomkom, ale nie obidvoje naraz. U nás budú do jedného stromu patriť tie lamy, ktoré sa znášajú. No a napokon koreň je vrchol, ktorý je predok všetkých vrcholov v strome. Od ostatných vrcholov sa bude odlišovať tým, že jeho otec bude on sám.

Budeme využívať dve funkcie. *find* nám nájde koreň stromu, v ktorom sa nachádza daný vrchol. *merge* spojí dva disjunktné stromy – to znamená dve skupiny lám, o ktorých sme predtým nevedeli, že sa znášajú – do jedného. Find sa dá implementovať tak, že stále prechádzame po otcoch, až kým neprídeme do koreňa. Pri *merge* jednoducho nastavíme

jeden z vrcholov ako otca tomu druhému. Pritom predpokladáme, že vrcholy, na ktoré voláme *merge* sú korene daných stromov. Všimnite si, že toto je rozumné, pretože koreň je predkom všetkých vrcholov vo svojom strome a tak nastavením jeho otca efektívne pripojíme celý strom.

Algoritmus potom bude pracovať tak, že si na začiatku vytvorí strom pre každú lamu. Každý strom bude mať jeden vrchol, ktorý bude aj jeho koreňom. Pri hlásení pozorovania dvoch lám sa funkciou $find$ zistí, či majú spoločný koreň. Ak nie, tak ich stromy spojíme funkciou *merge*. Ak budeme chcieť vedieť, či môžu byť lamy spolu, tak jednoducho znova funkciou $find$ zistíme, či majú spoločný koreň. Ak to ale naimplementujeme takto priamočiaro, tak dostaneme zložitosť $O(MN)$, pretože strom bude vznikať náhodne a $find$ môže trvať až $O(N)$ – príkladom je napríklad strom, kde každý vrchol má práve jedného syna.

Môžeme však spraviť nasledúcu vec: pri spájaní dvoch stromov pripojíme vždy ten s menším počtom vrcholov k tomu s väčším. Dá sa ľahko dokázať indukciou od hĺbky stromu, že takto vzniknutý strom bude mať hĺbku najviac $O(\log N)$ a teda žiadny $find$ nezaberie viac ako $O(\log N)$, čo nám celkovú zložitosť zlepší na pekných $O(M \log N)$.

Inou možnostou je zlepšiť funkciu $find$. Spravíme nasledujúcu úpravu: najprv nájdeme koreň daného vrchola, nazvime ho K. Potom tú istú cestu ku koreňu prejdeme ešte raz, ale všetkým vrcholom na nej nastavíme ako otca K. Dosiahli sme toľko, že ak nabudúce budeme niečo robiť s ľubovoľným z týchto vrcholov (a aj s ich potomkami), tak sa ku koreňu dostaneme rýchlejšie. Dokonca tak rýchlo, že sa dá ukázať, že teraz je výsledná zložitosť algoritmu $O(M \log N)$.

Poviete si: to je síce pekné, ale na čo je to dobré? Veď rovnakú zložitosť sme dosiahli aj pomocou jednoduchej úpravy funkcie *merge*. Dôvodom je, že uvedené optimalizácie sa vynikajúco dopĺňajú a ak ich implementujeme obe, dostávame oveľa lepšiu zložitosť $O(M \log^* N)$. Dôkaz je bohužiaľ pomerne zložitý a preto ho tu nebudem uvádzať.

Nakoniec vám teda ešte prezradím, čo to $\log^*$ vôbec je. Predstavme si čísla 1, 2, 2^2, $2^{(2^2)} = 2^4$, $2^{(2^{(2^2)})} = 2^{(2^4)} = 2^{16}$, $2^{(2^{(2^{(2^2)})})} = 2^{(2^{16})} = 2^{65536}$, ... $\log^* N$ je definovaný ako prvé číslo K také, že K-te číslo z hore vypísaných čísel je väčšie alebo rovné ako N. Ukážme si to teda na príklade:

$log^*N = 1$ ak $0 \leq N \leq 2$
$log^*N = 2$ ak $3 \leq N \leq 4$
$log^*N = 3$ ak $5 \leq N \leq 16$
$log^*N = 4$ ak $17 \leq N \leq 65536$
$log^*N = 5$ ak $65537 \leq N \leq 2^{65536}$

2^{65536} je číslo väčšie ako počet atómov vo vesmíre. Teda pre každé reálne použiteľné N je $\log^* N$ najviac 5. Lepšiu zložitosť už asi chcieť nemôžeme.

Mrknite sa na vzorové riešenie a osvojte si tento algoritmus, pretože sa vám v budúcnosti určite ešte zíde.

Listing programu:

```
const MAX = 10000;

var n, kind, a, b, koren_a, koren_b, i:integer;
    size:array[1..MAX] of integer;
    parent:array[1..MAX] of integer;

// vytvori nam strom pre jednu lamu
procedure makeset(n:integer);
begin
    parent[n] := n;
    size[n] := 1;
end;
```

```
procedure merge(a:integer; b:integer);
var c : integer;
begin
   // OPTIMALIZACIA 1: chceme mat v B cislo korena stromu s mensim poctom
potomkov
   if size[a] < size[b]  then
     begin c := a; a := b; b := c; end;
   // otec vrcholu B bude A, pretoze strom korena B ma menej vrcholov
   parent[b] := a;
   // a zvysime pocet potomkov stromu korena A
   size[a] := size[a] + size[b];
end;

function find(n:integer):integer;
var root, par:integer;
begin
   root := n;
   // najdeme koren
   while root <> parent[root] do
      root := parent[root];

   // OPTIMALIZACIA 2: nastavime vsetkym predkom vrcholu N ako otca ROOT
   while n <> parent[n] do begin
      par := parent[n];
      parent[n] := root;
      n := par;
   end;
   find := root;
end;

begin
   read(n);
   for i := 1 to n do makeset(i);
   while(true) do begin
      // nacitame typ operacie
      read(kind); if kind = 0 then break;
      // nacitame cisla lam
      read(a, b);
      if(kind = 1) then begin
         // najdeme korene stromov
         koren_a := find(a);
         koren_b := find(b);
         // a zavolame merge, ak su stromy disjunktne
         if koren_a <> koren_b then
            merge(koren_a, koren_b);
      end
      else begin
         // pozrieme sa, ci su lamy v tom istom strome
         if(find(a) = find(b)) then
           writeln('ANO')
         else
           writeln('RADSEJ NIE');
      end;
   end;
end.
```

2. Zabudnutý zošit

opravoval Zemčo
(max. 15 bodov)

Vaše riešenia sa dali rozdeliť do niekoľkých kategórii na základe toho, či ste si uvedomili hlavnú fintu problému a či ste si zvolili správny spôsob zapamätávania čísel. Hlavná myšlienka, na ktorú ale väčšina z vás prišla, spočíva v tom, že sčítavanie čísel v n-adickej sústave sa dá realizovať presne tak isto, ako klasické sčítanie pod seba v desiatkovej sústave. Keď sa dve čísla napíšu pod seba tak, že budú zarovnané sprava (jednotky pod sebou, desiatky pod sebou a tak ďalej), funguje to tak, ako sme zvyknutí. Vypočíta sa súčet na prvom mieste, ak je to viac ako sústava (tu je rozdiel, že nemáme 0, ale zato máme cifru n), tak sa započíta presun do vyššieho rádu a tak ďalej. Tu prichádza k druhému rozdielu, zatiaľ čo pri normálom sčítavaní v desiatkovej sústave môže prejsť do vyššieho rádu maximálne 1 (ak aj sčítavame $9 + 9$ a prišla nám jednotka z predchádzajúcich výpočtov, tak je to stále len 19, čiže 9 a jedna do vyššieho rádu), tu sa nám môže stať, že prejde aj viac. Napríklad pri sčítaní $\overline{33} + \overline{33}$ v 3-adickej sústave nám vyjde $\overline{213}$, prišlo k prechodu o dva. Viac ako dva to ale nemôže byť. Toto tvrdenie ľahko dokážeme indukciou od počtu cifier sčítavaných čísiel. Základná myšlienka spočíva v tom, že ak nám aj príde posun o dva a máme sčítať dve najväčšie možné cifry – n, tak výsledný posun bude tiež maximálne dva. Takže sme si ukázali, že žiadna konverzia do desiatkovej sústavy nebola nutná.

Povedzme si teraz niečo o časovej zložitosti. Na prvý pohľad sa zdá, že stále to bude lineárne závislé od počtu cifier, takže je to skoro jedno, či to budeme konvertovať, veď len spravíme trocha zbytočnej roboty. Avšak to nie je pravda. Poďme sa bližšie pozrieť na elementárne aritmetické operácie. Zatiaľ čo v zložitejších algoritmoch ich zjednodušene považujeme za konštanté, pri taktomto type príkladov, kde ide o kľúčovú súčasť programu, ich musíme vnímať ako zložitejšie. Veď aj sám príklad je o sčitovaní dvoch čísel (aj keď neštandardných) a vidíme, že to nejde konštantne, ale len lineárne. Sčítanie či iná aritmetická operácia s klasickými číslami je realizovaná počítačom v čase, ktorý zavisí od počtu cifier. Keď mu prikážeme vynásobiť dve jednociferné čísla, tak má menej práce ako v prípade stociferných. Ak sa takto pozrieme na aritmetické operácie v riešeniach nášho príkladu, úvahy o časovej zložitosti získajú nový rozmer.

Číslo sa dalo v zásade pamätať dvoma spôsobmi – ako číselná premenná a ako pole znakov, respektíve string. Ak si číslo zapamätáme po znakoch, máme napríklad k dispozícii i značne väčší rozsah, ale to som nehodnotil. Poďme si ukázať, čo sa deje s číslami v n-adickej sustave pri priamom sčitovaní, ak si ho pamätáme ako číselnú premennú. My musíme číslo zmodulovať desiatimi, aby sme získali poslednú cifru a potom ho vydeliť desiatimi, aby sme sa posunuli ďalej. Toto musíme robiť toľkokrát, koľko cifier zadané číslo má, a keďže operáciu delenia desiatimi považujeme za lineárnu od počtu cifier, čo je v tomto postupe v priemere polovica dĺžky slova (začíname s celým a každým krokom ho skrátime o jednu cifru), vychádza nám to na kvadratický čas od počtu cifier. Problém tkvie v tom, že my nemáme prístup k jednej cifre čísla v konštantom čase – keď si ja poviem, že chcem vedieť piatu cifru čísla, tak ho musím vydeliť 10000 a potom $\bmod 10$ (alebo $\bmod 100000$ a zvyšok vydeliť 10000). Rovnako aj akokoľvek inak realizovaná konverzia do desiatkovej sústavy pracuje v kvadratickom čase.

A čo ak si číslo zapamätáme ako pole znakov tak, že ho načítavame po jednotlivých cifrách (alebo do stringu, čo nie je nič iné ako pole znakov)? V tomto prípade získame konkrétnu cifru jediným zavolaním, čo je v konštantom čase. A to nás privádza k správnemu spôsobu zapamätávania čísel. Jednu cifru výsledku vieme spočítať v tomto prípade v konštantom čase (zavolanie sa na obe príslušné cifry, ich prevod na čísla, pripočítanie prípadneho prenosu z predchádzajúcich výpočtov, ich sčítanie v konštantom čase, ošetrenie prenosu do vyššieho rádu), pretože nemusíme nič deliť ani modovať. A preto je výsledný čas lineárny od počtu cifier. V poriadku je ešte riešenie, že si načítame string (alebo pole znakov) a prevedieme si ho na pole čísel.

A ako sa bodovalo? Korektné riešenie dostalo aspoň 10 bodov. To, ktoré zabudlo na prípad, že môže vzniknúť presun dva, stratilo dva body. Je to síce príliš veľká chyba na to, aby som ho prehlásil za korektné, ale dnes mám dobrú náladu. Riešenia v kvadratickom čase delím na tie, ktoré riešili príklad prechodom do desiatkovej sústavy (maximum 10 bodov), a tie, ktoré počítali priamo (maximum 11, za myšlienku). No a tie v optimálnom čase mohli získať 15 bodov. Odhad časovej zložitosti bol obyčajne zlý, a ak bol dobrý, tak viac-menej len náhodou, že to bolo naozaj v lineárnom čase od počtu cifier. Na úroveň jednoduchých aritmetckých operácii sa nepreniesol nikto, a tak som bol k zlému odhadu časovej zložitosti troška tolerantnejší. Prísny som bol ale k tým, ktorí napísali, že pamäť je konštantná, pretože tá tiež lineárne závisela od počtu cifier vstupu. Síce tam mohlo byť len pár premenných konštantej veľkosti, ale dôležité je, čo sa s nimi robí. Keby som chcel sčítať dve 200-ciferné čísla, tak by som potreboval viac pamäte ako v prípade 5-ciferných. A to, že váš program nie je schopný počítať s 200-cifernými číslami, neznamená, že pamäťová zložitosť postupu je konštantná. Bodík mohol ešte uletieť za absenciu uspokojivého popisu.

Listing programu:

```
program sustavy;

var sustava,a,i,where,presun,dlzka1,dlzka2,pozicia1,pozicia2:integer;
    cislo1,cislo2:array[1..10000]of char;
    vysledok:array[1..10000]of integer;

function ChrToInt(a:char):integer;
begin
 chrtoint:=ord(a)-ord('0');
end;

begin
   readln(sustava);
   dlzka1:=1;   dlzka2:=1;
{Rozne dlzky cisel sa dali riesit viacerymi sposobmi,
ja som si dal na zaciatok kazdeho cisla jednu nulu,
neskor to vyuzijem}
   cislo1[1]:='0';  cislo2[1]:='0';
   while not eoln do begin
      inc(dlzka1);
      read(cislo1[dlzka1]);
   end;
   readln;
   while not eoln do begin
      inc(dlzka2);
      read(cislo2[dlzka2]);
   end;
   readln;
{Zacneme od konca kazdeho cisla}
   pozicia1:=dlzka1;
   pozicia2:=dlzka2;
{Vynulujem si vysledok}
   if dlzka1>dlzka2 then for I:=1 to dlzka1+1 do vysledok[i]:=0
   else for I:=1 to dlzka2+1 do vysledok[i]:=0;
   presun:=0;
   where:=1;
   repeat
       a:=ChrToInt(cislo1[pozicia1])+ChrToInt(cislo2[pozicia2])+presun;
       presun:=0;
       while a > sustava do begin
         a:=a-sustava;
```

```
        inc(presun);
      end;
      vysledok[where]:=a;
      inc(where);
{Ak som prisiel na prve miesto, kde je ta nula, tak sa
uz nehybem, stale pocitam s tou nulou.}
      if pozicia1>1 then dec(pozicia1);
      if pozicia2>1 then dec(pozicia2);
{A skoncim prave vtedy, ked som prisiel na koniec
oboch cisel, a presun je nula, co znamena, ze som uz
definitivne skoncil}
   until (pozicia1=1)and(pozicia2=1)and(presun=0);
   for i:=1 to where-1 do write(vysledok[where-i]);
   readln;
end.
```

3. Zase tie podvody...

opravoval Mic
(max. 15 bodov)

Najskôr si asi povieme, ako som bodoval.

1. Za riešenia bežiace v čase $O(N)$ a s pamäťou $O(\log N)$ som udeľoval 15 bodov.
2. Za čas $O(N)$ a pamäť $O(N)$ som dával 14 bodov.
3. Riešenia s časom $O(N \log N)$ získali najviac 12 bodov.
4. Kvadratické riešenia($O(N^2)$) získali najviac 10 bodov.
5. Ostatné, teda nefunkčné, riešenia získavali málo bodov...
6. Bonus -1 bod som udeľoval za chýbajúci odhad časovej a pamäťovej zložitosti.
7. 2 body som ďalej strhával za nekvalitný popis.

Takže teraz niečo k vzorovému riešeniu. Triviálne riešenie bolo vyskúšať sa postaviť na všetky možné pozície a pre každú si vyrátať, koľkokrát musí Banto podvádzať. Takéto riešenie malo časovú zložitosť $O(N^2)$. My si ale ukážeme lepšie[1] riešenie. Ako na to? Najskôr urobme menšiu úvahu. Predstavme si štandardného turnajového pavúka s nejakým rozmiestnením. Nech P je minimálny počet podvodov, ktoré musí Banto vykonať, G je počet zápasov, ktoré Banto absolvuje, a Z je poradie posledného zápasu, kedy nebude musieť Banto podvádzať. Potom $P \leq G - Z$ (to preto, lebo všetky ďalšie zápasy musí podvádzať). Čo ale znamená, že Banto nemusel podvádzať? No predsa, že všetci v danom "podpavúkovi" sú horší než on. Banto predsa nebude nejaká lama a tak sa tam postaví. Čiže $P \geq G - Z$. Z toho ale vyplýva, že $P = G - Z$, teda Banto bude najskôr stále čestne vyhrávať, a potom už iba stále podvádzať. Keby po nejakom podvádzaní zase raz nemusel podvádzať, tak by bolo pre neho výhodnejšie sa postaviť do "podpavúka", z ktoreho vyšiel ten, ktorého porazil, lebo v ňom sú všetci slabší než Banto.

Takže čo teraz budeme s tým robiť? No stačí nám zistiť, ako najvyššie sa vie dostať bez podvádzania. No a začneme pochopiteľne presne opačne, a to zvrchu. Použijeme metódu rozdeľuj a panuj. Ako táto metóda funguje? Kedy je vhodná? Je vhodná práve vtedy, keď si vieme rozdeliť problém na dva menšie podproblémy, ktoré po vyriešení vieme efektívne spojiť do riešenia nášho problému. Znie to hrozne, tak si to ukážme na príklade. Chceme zistiť, či bude Banto podvádzať vo finále. Vo finále nemusí podvádzať práve vtedy, keď sú všetci menší ako on. No ale ako by sme na to napasovali rozdeľuj a panuj? Rozdeľme si jeho protihráčov na ľavú polovicu, pravú polovicu (alebo tiež semifinálove zápasy) a stredného hráča, ktorý sa pridá do tej polovice, v ktorej Banto neštartuje. Ak je stredný protihráč

[1] rýchlejšie a jednoduchšie napísateľné

horší ako Banto, a v ľavej aj pravej polovici Banto nemusel podvádzať (čiže všetci sú tam horší než on), tak potom ani vo finále Banto nemusí podvádzať, čiže počet podvodov je nula. Inak bude musieť vo finále podvádzať. Na to, aby sme zistili, či musí Banto podvádzať v semifinále, použijeme rovnaký postup ako pri zisťovaní, či bude Banto podvádzať vo finále, čiže si každe semifinále rozdelíme na dve polovice (štvrťfinále) a stredneho hráča. Ak Banto v oboch poloviciach nemusel podvádzať a aj stredný hŕáč je slabší než Banto, tak v semifinále nemusel podvádzať. To znamená, že na to, aby som vedel, či Banto bude musieť podvádzať vo finále, potrebujem vedieť, či musí podvádzať v semifinále, a na to, aby som vedel, či bude musieť podvádzať v semifinále, musím vedieť či bude musieť podvádzať v štvrťfinále,...

Časom sa dostaneme do takého stavu, že sa budeme zaoberať tým ako dopadol Bantov prvý zápas. To bude práve vtedy, keď sa budeme pytať na zápas s jediným hráčom. Vtedy si to vieme jednoducho vyrátať, stačí nám porovnať Bantovu silu so súperovou silou. Určite ste si asi všimli, že pri riešení použijeme rekurziu. Ako teda bude naša rekurzívna funkcia fungovať?

Najskôr si ešte všimnime jednu vec. Nemusíme si dopredu načítať celý vstup. Prečo? Lebo vieme program napísať tak, že budeme načítavať prvky po poradí, a to tak, že najskôr sa zavoláme na ľavú polovicu, potom načítame stredný prvok, a nakoniec sa zavoláme na pravú polovicu. Zavolanie na ľavú polovicu načíta všetky prvky pred stredným prvkom a zavolanie na pravú polovicu načíta všetky prvky za stredným prvkom. To znamená, že dávať funkcií za parameter interval, pre ktorý zisťuje, či tam musí podvádzať, je zbytočné, lebo sa nepozeráme do poľa, ale prvky načítavame len vtedy, keď ich potrebujeme. Postačí nám veľkosť intervalu, na ktorý sa voláme – počet prvkov, ktoré treba načítať. Ak tá veľkosť bude rovná nule (to znamená, že Banto by súťažil sám zo sebou) tak to znamená, že Banto určite nemusí podvádzať. A ešte za parameter dáme aj hĺbku rekurzie (funkciu budeme rekurzívne volať vždy o 1 väčšou hĺbkou ako je aktuálna hĺbka, na čo ju treba pochopíte o chvíľku).

Takže funkcia sa zavolá na ľavú polovicu, načíta stredný prvok, a zavolá sa na pravú polovicu. Stačí uvažovať prípady, že ak raz Banto podvádza, tak bude podvádzať už stále. Teda si stačí pamätať najmenšiu hĺbku, v ktorej nemusíme podvádzať, čo bude aj náš výsledok. Hĺbka rekurzie nám hovorí koľko zápasov Banta ešte čaká. Teda v prípade, že Banto nemusí podvádzať, pozrieme sa na doteraz najmenšiu takú hĺbku a prípadne ju aktualizujeme. A teraz si pokojne môžete prečítať kód vzorového riešenia. Ale ešte si povieme niečo o časovej a pamäťovej zložitosti.

Časová zložitosť tohto riešenia je $O(N)$. Urobíme konštantný počet operácií pre finále, dvojnásobok pre semifinále, štvornásobok pre kolá predtým,..., až nakoniec N-násobok. A keďže platí $N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} \cdots = 2N$ tak aj celkový čas je $O(N)$. [2] Pamätáme si konštantný počet premenných, a funkcia si pamätá tiež konštantný, čiže pamäťová zložitosť závisí len od hĺbky rekurzie(keďže pri každom volaní potrebujeme konštantný počet premenných), ktorá je $\log N$. Prečo? Nech h je hĺbka rekurzie. Koľko krát sa teda zavoláme? Najskôr sa zavoláme na polovicu, potom štvrtinu, osminu,..., až sa nakoniec zavoláme na jeden prvok, keďže N je mocninou 2. Čiže rekurzia nám zastane vtedy, keď $\frac{n}{2^h} = 1$. Nás zaujíma akú hodnotu má h. K funkcií 2^x existuje inverzná funkcia $\log_2(x)$. Použijeme ju na vyriešenie našej rovnice. Obidve strany zlogaritmujeme a dostaneme $\log_2(\frac{N}{2^h}) = \log_2(1)$, a teda $\log_2(N) - \log_2(2^h) = 0$. Z toho dostaneme že $\log_2(N) - h = 0$, čo znamená že $h = \log_2(N)$. Teda pamäťová zložitosť je $O(\log N)$.

Listing programu:

```
program Banto_velky_cheater;
```

[2] Nakreslite si úsečku a prikreslite k nej polovicu úsečku. Potom k tomu prikreslite štvrtinu úsečky. Takto pokračujte ďalej, až kým vás to neomrzí, a potom to porovnajte s úsečkou, ktorá je dvojnásobne dlhá ako pôvodná úsečka. Vždy keď prikreslíte ďalší zlomok úsečky, tak rozdiel veľkostí sa vydelí dvoma. Čiže celkový súčet dĺžok úsečiek nikdy neprekročí dvojnásobok dĺzky pôvodnej úsečky

```
const MAX=100;
var N, B, min : integer;

function can_win(s,d:integer):boolean;
var p:integer;
    cheat_left,cheat_right:boolean;
begin
  if s=1 then begin
    can_win:=true;
    exit;
  end;
  cheat_left:=can_win(s div 2,d+1);
  read(p);
  cheat_right:=can_win(s div 2,d+1);
  if(cheat_left)and(cheat_right)and(p<B)then begin
    can_win:=true;
    if min>d then min:=d;
  end else can_win:=false;
end;

begin
  readln(N,B);
  min:=round(ln(N)/ln(2));
  can_win(N,0);
  writeln(min);
end.
```

opravovali Stano a Elφnqa
(max. 18 bodov)

4. Zase ty!

Ako vypočítame obsah trojuholníka, na to máme vzorec už v zadaní. Problém nám robí iba prienik dvoch trojuholníkov. Obsah zjednotenia vyrátame ako súčet obsahov trojuholníkov mínus prienik. Pustime sa teda do prieniku. Aké body môžu byť jeho vrcholmi? Budú to buď priesečníky strán alebo vrchol trojuholníka, ležiaci vnútri druhého trojuholníka, na čo mnohí zabúdali (kto tomu neverí, môže si nakresliť všetky prípady).

Potrebujeme teda nájsť všetky takéto body. Najprv nájdeme vrcholy ležiace v druhom trojuholníku: Pre každý vrchol trojuholníka (jedného aj druhého) otestujeme, či neleží v tom druhom. Bod X leží v trojuholníku ABC vtedy, ak leží na tej istej strane od vektorov $\overrightarrow{AB}$, $\overrightarrow{AC}$, $\overrightarrow{CA}$, alebo na niektorom z nich. To, na ktorú stranu leží bod X od vektora AB, zistíme vektorovým súčinom. Ak je ten kladný, bod X leží na jednej strane, ak je 0, tak leží na vektore, a ak záporný, tak leží na druhej strane.

A čo to vlastne je taký vektor a vektorový súčin? Vektor znázorňuje smer alebo posun. Napríklad v trojrozmernom priestore vektor $[a, b, c]$ znázorňuje smer, ktorý má polpriamka vedúca z bodu $[0, 0, 0]$ do bodu $[a, b, c]$. A podobne, ak máme ľubovoľné dva body A, B (zadané súradnicami), vektor zodpovedajúci smeru „z A do B“ ľahko určíme ako rozdiel súradníc bodu B a bodu A.

S vektormi môžeme robiť dve zaujímavé operácie. Obe sa zvyknú označovať „súčin“. No a aby sme vedeli rozlíšiť, ktorú máme na mysli, jednu voláme skalárny súčin (lebo jej výsledkom je skalár, teda číslo) a tú druhú vektorový súčin (hádaj prečo).

Vektorový súčin definujeme len pre trojrozmerné vektory. Ak $u = [u_1, u_2, u_3]$ a $v = [v_1, v_2, v_3]$, tak ich vektorový súčin je $u \cdot v = [u_2v_3 - u_3v_2, u_3v_1 - v_3u_1, u_1v_2 - u_2v_1]$ (čiže tiež vektor). A to nie vektor hociaký ale rovno kolmý na oba z násobených vektorov. O

vektorovom súčine ešte vieme nasledovné: Polovica veľkosti vektorového súčinu vektorov $\overrightarrow{AB}$ a $\overrightarrow{AC}$ je obsah trojuholníka ABC (takto sme dostali vzorec pre výpočet obsahu trojuholníka v zadaní).

Keď sa postavíme na spoločný začiatok vektorov (každé vektory sa dajú presunúť v súradnicovom systéme tak, aby začínali v tom istom bode), pozrieme sa smerom prvého vektora a keď smer druhého vektora vidíme na ľavej strane, výsledok ich vektorového súčinu bude vektor, ktorý bude smerovať „hore“. Naopak, ak vektor, ktorý pri násobení dávame na druhé miesto, vidíme napravo, výsledný vektor bude smerovať „dole“.

Takže máme tri body (A, B, X), z ktorých odvodíme dva vektory ($\overrightarrow{AB}$, $\overrightarrow{AX}$). Tým pridáme tretí rozmer jednoducho tak, že pridáme ako tretiu súradnicu nulu, čo nám určite nič nepokazí. No a teraz treba spraviť len jeden vektorový súčin, ktorým zistíme, či je prvý vektor od druhého napravo alebo naľavo. Ba čo viac, nás vlastne výsledný vektor nezaujíma, nás zaujíma len jeho tretia súradnica, respektíve jej kladnosť či zápornosť (ostatné súradnice vďaka nulovej tretej súradnici vektorov $\overrightarrow{AB}$, $\overrightarrow{AX}$ budú nulové). Takže potrebudeme spraviť len jeden výpočet, a to $w_3 = u_1v_2 - u_2v_1$.

Kedy sa stretneme so situáciou, že nám leží tretí bod na priamke AB (takým vravíme kolineárne)? Na jednej priamke vtedy budú ležať aj vektory, s ktorými pracujeme. Takým vektorom hovoríme lineárne závislé a znamená to aj, že jeden vektor vieme napísať ako druhý vynásobený vhodnou reálnou konštantou. A čudujsa svete, vektorový súčin dvoch lineárne závislých vektorov nám vráti nulový vektor.

Ak teda chceme zistiť, či bod X leží v trojuholníku ABC, vypočítame vektorové súčiny: $\overrightarrow{AX} \times \overrightarrow{AB}$, $\overrightarrow{BX} \times \overrightarrow{BC}$ a $\overrightarrow{CX} \times \overrightarrow{CA}$.

Aké môžu nastať prípady pre polohu bodu vzhľadom na trojuholník?

1. bod leží v trojuholníku – v tom prípade bude na tej istej strane od každého z vektorov zodpovedajúcich stranám, teda všetky tri vypočítané čísla budú nenulové a budú mať rovnaké znamienko.
2. bod leží na jednej strane trojuholníka – vtedy bude jedno číslo 0, a ďalšie dve budú mať rovnaké znamienko.
3. bod leží na vrchole druhého trojuholníka – v takomto prípade budú dve čísla 0 a tretie nenulové.
4. bod leží mimo trojuholníka – v takomto prípade budú mať dve z vypočítaných čísel rôzne znamienka.

Nájdené body si odložíme a ideme hľadať prieniky. Toto mala väčšina z vás dobre. Jedným možným pohľadom na vec je zistiť si rovnice priamok, na ktorých ležia dané úsečky a vyriešením sústavy rovníc nájsť prienik. Čo je to rovnica priamky? Každú priamku v rovine vieme určiť (nie nutne jednoznačne) tromi číslami a, b, c a rovnicou $ax + by + c = 0$. Teda ak za x, y dosadíme súradnice ľubovolného bodu priamky, tak rovnica platí, a pre všetky ostatné body v rovine rovnica neplatí.

Ako získame takú rovnicu priamky? Vieme, že priamku máme určenú smerovým vektorom $\overrightarrow{AB}$ a bodom, ktorý na nej leží A. Nech má vektor $\overrightarrow{AB}$ súradnice (s_1, s_2). Potom rovnicu dostaneme ako $s_2x - s_1y + c = 0$, pričom c dopočítame dosadením súradníc bodu A do rovnice. Keď takto zistíme rovnice oboch priamok, ktorých priesečník chceme nájsť, riešime sústavu 2 rovníc o 2 neznámych. Nebolo však treba zabúdať na kontrolu, či je prienik priamok aj prienikom úsečiek. Napríklad takto: nájdený bod $X = [x, y]$ leží na úsečke AB ($A = [ax, ay]$, $B = [bx, by]$), ak platí: $x > \min\{ax, bx\}$ a $x < \max\{ax, bx\}$ a podobne pre y-ové súradnice. Pozor si len treba dať na prípady, kedy je koeficient pri niektorej z premenných 0.

Keď máme všetky dôležité body, chceli by sme spočítať obsah mnohouholníka, ktorý nám vyšiel. Čo však mnohí zabúdali, body nemáme utriedené tak, ako idú po obvode, ale môžu byť poprehadzované. Vieme však, že prienik je určite konvexný, a navyše vieme, že je to nanajvýš šesťuholník. Ako zistiť správne poradie bodov na obvode? Nájdeme si bod B, ktorý je v mnohouholníku, napríklad ťažisko nejakého trojuholníka (to získame ako aritmetický

priemer súradníc vrcholov) a body utriedime podľa uhla, ktorý zviera kladná poloos osi x a polpriamka z bodu B do daného bodu. To nám nájdené body zoradí pekne po obvode.

Máme teda zoradené vrcholy prieniku a máme vnútri bod B. Teraz už môžeme vypočítať obsah prieniku ako súčet obsahov trojuholníkov BA_iA_{i+1}. Keďže sme mali na vstupe iba 2 trojuholníky, väčšia ako konštantná časová a pamäťová zložitosť sa asi ani nedala vyrobiť.

K hodnoteniu: Body ste mohli podostávať za nájdenie priesečníkov, vnútorných bodov v druhom trojuholníku, a korektné vypočítanie obsahu mnohouholníka. Za absenciu kódu bol koeficient $\frac{1}{2}$. Bodíky hore dole sa dali získať za pekné riešenie (a samozrejme dobrý, všetko obsahujúci a vysvetľujúci popis).

Listing programu:

```
uses math;
const inf=maxint;
type Tbod=record x,y:real; end;
     Ttroj=record b:array[0..2] of Tbod; end;
     Tvektor=record
       A,B:Tbod;
       x,y:real;
     end;

var t1,t2,pom:Ttroj;
    dol:array[0..11]of Tbod;
    i,q,j:integer;
    pri:Tbod;
    obs,cele:real;

{na ktoru stranu od priamky ab je polozeny bod c. ak na nej lezi, vrati 0, ak je na
jednu stranu 1, ak na druhu -1}
function strana(a,b,c:Tbod):integer;
var res:real;
begin
  res:=(c.x-a.x)*(c.y-b.y)-(c.x-b.x)*(c.y-a.y);
  if res=0 then strana:=0
  else if res>0 then strana:=1 else strana:=-1;
end;

{touto funkciou zistujeme ci sa nachadza bod(bod) vnutri trojuholnika(t)}
function vnutri(t:Ttroj; bod:Tbod):boolean;
var i,j,k:integer;
begin
  vnutri:=false;
  i:=strana(t.b[0],t.b[1],bod);
  j:=strana(t.b[1],t.b[2],bod);
  k:=strana(t.b[2],t.b[0],bod);
  if (i*j>=0) and (i*k>=0) and (j*k>=0) then vnutri:=true;
end;

function prienik(A,B,C,D:Tbod):Tbod;
var u1,u2,v1,v2,c1,c2,surx,sury:real;
    jezavisly:boolean;
begin
  jezavisly:=false;
{vypocitame rovnice priamok AB, CD}
  u1:=B.y-A.y;  u2:=A.x-B.x;
  v1:=C.y-D.y;  v2:=D.x-C.x;
  c1:=u1*A.x+u2*A.y;  c2:=v1*C.x+v2*C.y;
{overime, ci nie su linearne zavisle}
  if (v1<>0) and (v2<>0) then begin
```

```
    if abs((u1/v1) - (u2/v2))<0.000001 then jezavisly:=true;
  end;
  if (v1=0) and (u1=0) then jezavisly:=true;
  if (v2=0) and (u2=0) then jezavisly:=true;
  prienik.x:=inf;  prienik.y:=inf;

{ak niesu, najdeme ich prienik}
  if not jezavisly then begin
    if v2=0 then begin
      surx:=c2/v1;  sury:=(c1-u1*surx)/u2;
    end else if u1=0 then begin
      sury:=c1/u2;  surx:=(c2-v2*sury)/v1;
    end else begin
      sury:=(c1*v1-c2*u1)/(v1*u2-u1*v2);  surx:=(c1-u2*sury)/u1;
    end;
{zistime, ci ich prienik patri aj useckam AB, CD}
    if ( ((A.x<surx) and (surx<B.x)) or ((B.x<surx) and (surx<A.x)) ) and
       ( ((A.y<sury) and (sury<B.y)) or ((B.y<sury) or  (sury<A.y)) ) then begin
      prienik.x:=surx;  prienik.y:=sury;
    end;
  end;
end;

{najdeme tazisko vsetkych dolezitych bodov}
function tazisko:Tbod;
var i:integer;
    x,y:real;
begin
  x:=0;  y:=0;
  for i:=0 to q-1 do begin
    x:=x+dol[i].x;  y:=y+dol[i].y;
  end;
  tazisko.x:=x/q;  tazisko.y:=y/q;
end;

{utriedime dolezite body tak ako idu do kruhu}
procedure sort(t:Tbod);
var i,j:integer;
    uhol:real;
    b:Tbod;
    uhly:array[0..11]of real;
begin
  uhly[0]:=arctan2(dol[0].x-t.x,dol[0].y-t.y);
  for i:=1 to q-1 do begin
    b:=dol[i];
    uhol:=arctan2(b.x-t.x,b.y-t.y);
    j:=i;
    while (uhol<uhly[j]) and (j>0) do begin
      uhly[j]:=uhly[j-1];
      dol[j]:=dol[j-1];
      dec(j);
    end;
    uhly[j]:=uhol;
    dol[j]:=b;
  end;
end;

{obsah trojuholnika vyratame podla vzorca co sme mali v zadani
(kto by mu neveril, moze si ho odvodit sam, obsah trojuholnik sa rata
ako polovica vektoroveho sucinu dvoch stran)}
```

```
function obsah(t:Ttroj):real;
begin
  obsah:=abs(t.b[0].x*t.b[1].y + t.b[1].x*t.b[2].y + t.b[2].x*t.b[0].y -
  t.b[1].x*t.b[0].y - t.b[2].x*t.b[1].y - t.b[0].x*t.b[2].y) / 2;
end;

begin
  readln(t1.b[0].x, t1.b[0].y, t1.b[1].x, t1.b[1].y, t1.b[2].x, t1.b[2].y);
  readln(t2.b[0].x, t2.b[0].y, t2.b[1].x, t2.b[1].y, t2.b[2].x, t2.b[2].y);
  q:=0;
  for i:=0 to 2 do begin
    if vnutri(t1,t2.b[i]) then begin
      dol[q]:=t2.b[i];
      inc(q);
    end;
    if vnutri(t2,t1.b[i]) then begin
      dol[q]:=t1.b[i];
      inc(q);
    end;
  end;
  {pre kazdu dvojicu stran vyratame ich priesecnik}
  for i:=0 to 2 do
    for j:=0 to 2 do begin
      pri:=prienik (t1.b[i],t1.b[(i+1)mod 3],t2.b[j],t2.b[(j+1)mod 3]);
      if pri.x<inf then begin
        dol[q]:=pri;
inc(q);
      end;
    end;
  pri:=tazisko;
  sort(pri);
  obs:=0;
  pom.b[0]:=pri;
  for i:=0 to q-1 do begin
    pom.b[1]:=dol[i];
    pom.b[2]:=dol[(i+1) mod q];
    obs:=obs+obsah(pom);
  end;
  writeln(obs:2:3);
  cele:=obsah(t1)+obsah(t2)-obs;
  writeln(cele:2:3);
end.
```

5. Z koláča dieru II

opravoval MMx
(max. 15 bodov)

Na správnu myšlienku tohoto príkladu prišla väčšina z vás, ťažšie sa ukázalo správne ju naprogramovať. Zabúdali ste na drobnosti, napríklad výsledná dvojica kúskov môže mať súčet hrozienok na sebe nula (za toto som body nestŕhal), alebo kúsok nemôže susediť sám so sebou. Takisto sa nedá predpokladať, že prvé dva nájdené kúsky budú susediť.

Vzorové riešenie bude využívať prehľadávanie do hĺbky. Rovnako dobre sa dá použiť aj prehľadávanie do šírky z minulého kola, ale to tu už popísané bolo ;-). Praktický rozdiel medzi prehľadávaním do šírky a do hĺbky je ten, že pri prehľadávaní do šírky navštevujeme políčka v poradí podľa ich rastúcej vzdialenosti od prvého políčka (to znamená, že najprv

navštívime všetky susedné políčka, potom políčka so vzdialenosťou dva, atď.). Pri prehľadávaní do hĺbky sa rozbehneme jedným smerom a až keď narazíme na prekážku, zmeníme smer (teda najprv ideme smerom k 'najvzdialenejšiemu' políčku).

Program si najprv načíta celý koláč do pamäte (v tomto prípade sa tomu nedalo vyhnúť). Potom ho bude prechádzať po znakoch a hľadať zárezy. Keď nejaký nájde, pozrie sa, či vľavo a vpravo alebo hore a dole nie sú dva rôzne kúsky, ktorých súčet hrozienok je väčší, ako doteraz našiel. Ak áno, potom si túto dvojicu zapamätá. Dvojicu, ktorú si bude pamätať na konci, prehlási za výslednú.

Pri takomto postupe bude treba počítať, koľko hrozienok je na jednotlivých kúskoch a či sú dva kúsky rôzne. Keď budeme potrebovať zistiť počet hrozienok pre nejaký kúsok, najprv sa pozrieme, či sme ho už prehľadali. Ak áno, vrátime zapamätanú hodnotu. V opačnom prípade mu priradíme nejaké číslo (iné ako ostatným kúskom) a spustíme na neho prehľadávanie. Pomocou tohoto čísla budeme vedieť takisto zistiť, či sú dva kúsky rôzne.

Prehľadávanie kúsku do hĺbky bude vyzerať nasledovne: Na začiatku máme súradnice nejakého políčka, ktoré prehľadávanému kúsku patrí. Tak si toto políčko označíme príslušným číslom. Ak je na ňom hrozienko, potom zvýšime počet hrozienok pre tento kúsok. Teraz sme v podobnej situácii ako na začiatku: potrebujeme označiť všetky susedné políčka, pretože takisto patria prehľadávanému kúsku. Ako správne tušíte, použijeme na to rekurziu. To znamená, že jeden takýto krok implementujeme pomocou procedúry (v tomto prípade *spocitaj*), ktorá bude pokračovať vo výpočte tým, že na susedné políčka zavolá seba. Prehľadávanie sa zastaví, keď narazí na rez, na políčko, ktoré už bolo označené, alebo keď sa zavolá mimo koláča. Aby sme nemuseli pre všetky štyri smery kontrolovať, či jeden krok daným smerom je alebo nie je mimo koláča, ohraničíme si na začiatku celý koláč rezmi.

Pozrime sa na odhady zložitostí. Okrem jednoduchých premenných používame dve polia veľkosti N^2. V tomto prípade však treba pri odhade zložitosti brať ohľad aj na počet volaní procedúry *spocitaj*, pretože tá síce používa len konštantne veľkú pamäť, ale keď zavolá sama seba, tak v pamäti budú premenné z prvého aj druhého volania. Čiže treba odhadnúť, koľko volaní *spocitaj*, ktoré sa ešte nevrátili, môže nastať. V tomto prípade sa *spocitaj* môže zavolať na každé políčko najviac raz, čiže toto nám pamäťovú zložitosť $O(N^2)$ nezhorší. Takisto si treba dávať pozor na načítavanie vstupu zo súboru, pretože ak vstupný súbor prečítate viac ako raz, jeho veľkosť sa započíta do celkovej zložitosti (pretože ho vlastne používate ako pamäť).

Časová zložitosť bude takisto $O(N^2)$, pretože prechádzame celý koláč a keď pri tom nájdeme neoznačenú časť, celú ju prehľadáme (každú časť nanajvýš raz).

Hodnotenie bolo principiálne rovnaké ako v predchádzajúcom kole, teda 15 bodov za riešenie s časovou zložitosťou $O(N^2)$, 9 bodov za $O(N^4)$, -1 až -2 body za zlý popis a -1 bod za nesprávny odhad. Body som strhával aj za drobné chyby v programe.

Listing programu:

```
const maxn=100;
var kolac       :array[0..maxn+1,0..maxn+1] of integer;
    hrozienok   :array[1..maxn*maxn] of integer; //pocty hrozienok pre kusky kolaca
    n,i,j,kuskov :integer;
    c           :char;
    maxc,maxx1,maxy1,maxx2,maxy2 :integer; //najlepsi doteraz najdeny kusok

procedure spocitaj(x,y,n :integer); //rekurzivne prehladaj cely kusok
begin
  if kolac[x,y]>=0 then exit;
  if kolac[x,y]=-2 then inc(hrozienok[n]);
  kolac[x,y]:=n;
  spocitaj(x-1,y,n); spocitaj(x+1,y,n);
  spocitaj(x,y-1,n); spocitaj(x,y+1,n);
```

```
end;

function pocet(x,y :integer):integer; //zisti pocet hrozienok na kusku, pripadne spusti
prehladavanie
begin
  if kolac[x,y]<0 then begin
    inc(kuskov); spocitaj(x,y,kuskov);
  end;
  pocet:=hrozienok[kolac[x,y]];
end;

procedure testmax(x1,y1,x2,y2 :integer); //vyskusaj, ci tato dvojica kuskov nie je
lepsia
var spolu :integer;
begin
  if (kolac[x1,y1]<>0) and (kolac[x2,y2]<>0) then begin
    spolu:=pocet(x1,y1)+pocet(x2,y2);
    if (spolu>maxc) and (kolac[x1,y1]<>kolac[x2,y2]) then begin
      maxx1:=x1; maxy1:=y1;
      maxx2:=x2; maxy2:=y2;
      maxc:=spolu;
    end;
  end;
end;

begin
  readln(n);
  for i:=0 to maxn+1 do begin//ohranic kolac rezmi
    kolac[0,i]:=0; kolac[n+1,i]:=0;
    kolac[i,0]:=0; kolac[i,n+1]:=0;
  end;
  for i:=1 to n do begin
    for j:=1 to n do begin
      read(c);
      case c of
       '.': kolac[i,j]:=-1;
       '*': kolac[i,j]:=-2;
       '#': kolac[i,j]:=0;
      end;
    end;
    readln;
  end;

  maxc:=-1; kuskov:=0; //inicializovat maxc na 0 je chyba
  for i:=1 to n do //pre kazdy rez vyskusaj obidve dvojice
    for j:=1 to n do
      if kolac[i,j]=0 then begin
        testmax(i,j-1,i,j+1);
        testmax(i-1,j,i+1,j);
      end;
  writeln('Prvy kusok: ', maxx1, ', ', maxy1);
  writeln('Druhy kusok: ', maxx2, ', ', maxy2);
end.
```

Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
1	Danilák Michal	Gym. Hubeného BA	3	67	14	15	14	13	15	138
2	Hapák Samuel	Gym. Grösslingová BA	2	69	14	13	1	12	14	123
3	Szabados Michal	Škola pre mim. nadané deti BA	3	62	9	9	14	11	13	118
4	Kucharík Marcel	Gym. Športová Nové Mesto n.V.	4	59	11	8	10	15	14	117
5	Sucha Martin	Gym. Jura Hronca BA	2	59	13	10		10	14	106
6	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	2	45	15	15	4	17	7	103
7	Kočiský Tomáš	Gym. Grösslingová BA	2	50	12	11	4	8	14	99
7	Košinárová Alena	Gym. Grösslingová BA	2	65	12	10	12			99
9	Nikodem Peter	Gym. Školská Spiš. Nová Ves	2	42	10	13	12		15	92
10	Novák Ján	Gym. Ľudovíta Štúra Trenčín	4	47	12	12			14	85
11	Mészároš Roman	SPŠ Levice	3	45	6	14	10		9	84
12	Hanes Filip	Gym. Tajovského B. Bystrica	3	31	12	12	10	14		79
13	Petrucha Michal	Gym. Metodova BA	1	56		10		12		78
13	Saleh Adam	Gym. Jura Hronca BA	2	42	12	12			12	78
15	Nagy Kristián	Gym. Jura Hronca BA	1	40	12	12	4		9	77
16	Fedáková Dominika	Gym. Stará Ľubovňa	2	34	12	11	11	7		75
17	Betík Roman	SPŠ Levice	4	24	12	9	12		14	71
17	Košdy Martin	Gym. Jura Hronca BA	1	32	12	14			13	71
17	Kvasnička Igor	Gym. Jura Hronca BA	2	33	12	14	12			71
20	Synak Peter	Gym. Jura Hronca BA	2	39	4	11	9		6	69
21	Kostolányi Peter	Gym. Jura Hronca BA	2	41	12	10	4			67
22	Korcsok Peter	Gym. Mládežnícka Šahy	2	27	10	15	12			64
23	Hlavatý Peter	Gym. Jura Hronca BA	2	42	11	9				62
24	Mego Marek	Gym. Jura Hronca BA	4	40	10	11				61
25	Schiffer Juraj	Gym. Jura Hronca BA	4	35		11	12			58
26	Baumann Martin	Gym. Jura Hronca BA	3	42		15				57
27	Vojtko Jakub	Gym. Jura Hronca BA	1	0	12	15	12	15		54
28	Švec Marcel	Gym. Jura Hronca BA	3	42	11					53
29	Molčan Dávid	Gym. Slovenská Bardejov	3	18	12	10	12			52
30	Dullová Veronika	Gym. Ul. L. Sáru BA	3	22	10	10	7			49
31	Fecko Stanislav	Gym. Pankúchova Bratislava	3	46						46
32	Beňo Fratišek	SPŠ Humenné	2	19		10	11	4		44
32	Schlosáriková Lucia	Gym. P. de Coubertina Piešťany	3	5	12	9	4		14	44
34	Bachratý Martin	ZŠ Gaštanová Žilina	0	31	12					43
34	Beno Miroslav	Gym. Jura Hronca BA	4	43						43
36	Vojtko Martin	Gym. Jura Hronca BA	3	27		15				42
37	Novacek Milos	Gym. Jura Hronca BA	4	40						40
38	Holla Kamila	Gym. Jura Hronca BA	4	36						36
39	Janošík Jozef	Gym. Varšavská Žilina - Vlčince	1	35						35
39	Orihel Lukas	Gym. Jura Hronca BA	4	35						35
41	Kučin Alexander	Gym. Lipany	2	17	7	10				34
41	Móro Róbert	Gym. Ul. L. Sáru BA	3	24		10				34
41	Trančík Ivan	Gym. Jura Hronca BA	2	34						34
44	Jakabovič Juraj	Gym. Jura Hronca BA	2	33						33
45	Takács Michal	Gym. Tajovského B. Bystrica	4	31						31
45	Valenčík Ivan	Gym. Ul. L. Sáru BA	3	31						31
47	Čevorová Kristína	Škola pre mim. nadané deti BA	3	0	8	10	12			30
48	Hojčková Martina	Gym. Jura Hronca BA	3	21		8				29
48	Liska Igor	Gym. Jura Hronca BA	1	0	11	11	7			29
48	Popovič Viktor	Gym. Mudroňova Prešov	0	20		9				29

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
51	Hreha Ján	Gym. Liptovský Hrádok	3	27						27
52	Magyar Vladimír	SPŠE Nové Zámky	2	0	12	10	4			26
52	Zajacová Danica	Gym. Jura Hronca BA	3	26						26
54	Lachata Adrián	Gym. Svidník	4	24						24
55	Bašista Peter	Gym. P. Horova Michalovce	4	21						21
55	Berthoty Adam	Gym. Ľudovíta Štúra Trenčín	4	21						21
55	Hlaváček Filip	Gym. Hubeného BA	4	21						21
58	Langer Lukáš	Súkromná SOŠ Humanus Via	2	20						20
59	Fojtík Michal	Gymnázium	4	19						19
59	Lacko Michal	Gym. Veľký Krtíš	4	19						19
59	Morvay Peter	Gym. Jura Hronca BA	3	13	6					19
59	Šmeringai Peter	Neznama skola	3	19						19
63	Ihnát Peter	Gym. Sečovce	2	18						18
63	Koval Michal	Gym. Jura Hronca BA	2	18						18
63	Kundis Tomáš	SOU, Kukučínova 483, Poprad	3	18						18
63	Kurpel Matej	Gym. Ľudovíta Štúra Trenčín	4	18						18
67	Doležal Martin	Gym. Alejová Košice	2	17						17
68	Jančigová Jarmila	Gym. Jura Hronca BA	2	0	4	12				16
68	Kovács Vince	Gym. Štúrovo	2	16						16
68	Šuranyi Peter	Gym. Jura Hronca BA	4	16						16
71	Kukan Matúš	Gym. Grösslingová BA	2	13						13
72	Hattas Marek	SPŠ Nitra	3	12						12
72	Trnovec Matúš	Gym. Jura Hronca BA	2	0	12					12
74	Buchovecká Simona	Gym. Medzilaborce	3	10						10
74	Rybár Michal	Gym. sv. Uršule BA	3	10						10
74	Tóth Ladislav	SPŠE Nové Zámky	3	0		10				10
74	Urbanová Elena	Gym. Jura Hronca BA	3	0	6				4	10
78	Gálik Marian	Gym. Jura Hronca BA	2	9						9
78	Godány Martin	Škola pre mim. nadané deti BA	3	9						9
78	Kelemeca Patrik	SPŠ Snina	3	9						9
78	Šrámek Radoslav	Gym. Jura Hronca BA	3	9						9
78	Vnuk Marek	Gym. Jura Hronca BA	2	9						9
83	Sember Matej	Gym. sv. Uršule BA	2	8						8
84	Lesnák Stefan	Gym. Jura Hronca BA	2	6						6
84	Mazal Tomáš	Gym. Jura Hronca BA	3	0	6					6
86	Dittrich Andrej	Gym. Jura Hronca BA	2	0	2					2