Korešpondenčný seminár z programovania
XVIII. ročník, 2000/2001
Katedra vyučovania informatiky MFF UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje nadácia Open Society Fund Bratislava a
IUVENTA – zariadenie pre voľný čas detí, mládeže a dospelých

Vzorové riešenia 2. kola

Milí naši riešitelia,

tohoročná zima vyzerá veľmi neslávne – žiadny sneh, samý dážď. Tak, keď sa nie je poriadne ani kde lyžovať, tak sa môžete spokojne zahĺbiť do vzorových riešení. Tí, ktorí sa doteraz najviac snažili, môžu sa už teraz začať tešiť na nadchádzajúce jarné sústredenie KSP. Zároveň vás všetkých chceme povzbudiť do ďalšieho riešenia KSP v druhom polroku. Po zmene pravidiel máte aj vy, doteraz menej, ale odteraz iste viac snaživí riešitelia, šancu dostať sa na jesenné sústredenie KSP.

Nikdy nepľujte proti vetru.

KSPáci

1. O bojovej sekere

opravoval Goober
(max. 15 bodov)

Tento príklad zrejme na prvý pohľad vyzeral jednoducho, a preto ste ho mnohí podcenili. Vaše riešenia by sa dali rozdeliť do niekoľkých skupín:

- Riešenia, ktoré zväčša nenašli optimálne rozdelenie, pretože boli založené na nesprávnej myšlienke. Našli sa však aj optimisti, ktorí uviedli ešte aj dôkaz správnosti svojej myšlienky... Za takéto riešenia bolo možné získať nanajvýš **3 body**.
- Riešenia, ktoré našli najvýhodnejšie rozdelenie, no ich zložitosť bola horšia ako lineárna. Za takéto riešenia ste mohli dostať **6–11 bodov**.
- Riešenia optimálne, čiže lineárne, ktoré boli ohodnotené **15 bodmi**. Sem patria aj niektoré riešenia, ktoré síce o sebe vyhlasovali, že sú $O(N^2)$, no ich skutočná zložitosť je $O(N)$.

Ako už býva zvykom, udeľovali sa aj špeciálne prémie. Za chýbajúci alebo nesprávny dôkaz išiel tiež nejaký bodík dole. Podobne aj nadmerné využívanie pamäte bolo ocenené zníženým počtom bodov. Naopak, elegantné (ale v prvom rade funkčné) riešenie mohlo dostať o bod viac.

Ako sa to teda malo robiť? Základom úspechu je zistenie, že výrok medicinmana zabezpečuje, že nebude treba viac ako dve skupiny. Je zrejmé, že jedna skupina môže stačiť len vtedy, ak každý kmeň má nanajvýš jedného nepriateľa. Nasledujúci dôkaz indukciou bude súčasne aj návodom na riešenie.

Kmene rozdelíme na *úplne spokojné* (také nemajú v skupine nepriateľa), *spokojné* (keď majú nanajvýš jedného) a *nahnevané* (keď majú viac). Jeden kmeň môžeme umiestniť ľubovoľne a bude úplne spokojný. Predpokladajme teraz, že sme už umiestnili prvých $I-1$ kmeňov ($1 < I \leq N$), všetky sú spokojné a chceme umiestniť kmeň s číslom I. Keďže nemá viac ako troch nepriateľov, v niektorej skupine musí mať nanajvýš jedného (ak ho má, označíme ho P). Ak dáme I do tejto skupiny, bude určite spokojný. Nahnevať sme však mohli kmeň P, ktorý mohol takto dostať do svojej skupiny druhého nepriateľa (ak tam už predtým jedného, označme ho Q, mal). Ak je tak, vezmeme kmeň P a presunieme ho do opačnej skupiny. Tým

sme sa dostali do podobnej situácie ako predtým — opäť dávame jeden kmeň do skupiny, kde má nanajvýš jedného nepriateľa (lebo v pôvodnej mal dvoch). Nastala však určitá zmena — kmene I aj Q sú už úplne spokojné, lebo ich jediného nepriateľa (P) sme prehodili. Takto budeme pokračovať ďalej (nikdy nebudeme mať viac ako jeden nahnevaný kmeň, a keď už sú všetky spokojné, skončíme). Už len treba dokázať, že skutočne raz skončíme. Predpokladajme, že by to nebola pravda. Potom by sme sa niekedy museli vrátiť ku kmeňu, ktorý sme už predtým spracovali. Ako však bolo povedané, takýto kmeň je úplne spokojný, čiže keď mu pridáme jedného nepriateľa do skupiny, nenahnevá sa. Preto máme istotu, že prehadzovanie sa raz skončí (a všetky kmene budú potom spokojné) a navyše žiadny kmeň neprehodíme viac ako raz.

Odhad zložitosti: Keďže každé pridanie kmeňa vyžaduje najviac N prehodení, zdalo by sa, že popísaný postup má časovú zložitosť $O(N^2)$. Prefíkanejším rozborom (ktorý mnohí z vás mohli urobiť, ale zväčša neurobili) však zistíme, že tento odhad sa dá zlepšiť až na $O(N)$. Treba si všimnúť, že na každom úplne spokojnom kmeni sa prehadzovanie zastaví. Označme C počet neúplne spokojných kmeňov (t.j. takých, ktoré maju práve jedného nepriateľa v skupine). Pri každom jednotlivom prehodení (s výnimkou posledného) sa C zníži. Keďže C sa zvyšuje len pri pridávaní nového kmeňa, zvýši sa nanajvýš N-krát, a teda celkove bude prehodení $O(N)$.

Listing programu:

```
#include <stdio.h>
#define MAXN 100

int sek[MAXN+1][3];             /* Vykopane sekery */
int nepr[MAXN+1];               /* Pocet nepriatelov i-teho kmena */
char trebadve=0;                /* Treba dve skupiny ? */
char kde[MAXN+1];               /* Kde je i-ty kmen ? */
int nvs[MAXN+1];                /* Ak je uplne spokojny 0, inak cislo
                                   jedineho nepriatela v skupine */
int i,m,n,a,b;

/* Vykope bojovu sekeru medzi kmenmi 'a' a 'b' */
void Vykop(int a, int b) {
  sek[a][nepr[a]++]=b; sek[b][nepr[b]++]=a;
  if ((nepr[a]>1) || (nepr[b]>1)) trebadve=1;
}

void Pridaj(int k) {
  int s1=0,s2=0,p,i,celkove=k;

/* Najprv najdeme skupinu, v ktorej ma menej nepriatelov, a pridame ho */
  for (i=0;i<nepr[k];i++)
    { p=sek[k][i]; if (p<k) { if (kde[p]==0) s1++; else s2++; } }
  if (s2<s1) kde[k]=1; else kde[k]=0;

  while(1) {
/* Najdeme mu nepriatela (P) v skupine, ak ziadneho nema, tak koncime */
    for (i=0;i<nepr[k];i++)
      { p=sek[k][i]; if ((p<=celkove) && (kde[p]==kde[k])) break; }
    if (i==nepr[k]) { nvs[k]=0; break; }
/* Ak sa P nahneval, prehodime ho, inak iba upravime udaje */
```

```
    if (!nvs[p]) { nvs[k]=p; nvs[p]=k; break; }
    else { nvs[k]=nvs[nvs[p]]=0; k=p; kde[k]^=1; }
  }
}

void main() {
  scanf("%d %d",&n,&m);
  for (;m>0;m--) { scanf("%d %d",&a,&b); Vykop(a,b); }
  if (!trebadve) printf("Vsetky kmene mozu byt v jednej skupine\n");
  else {
    printf("Treba dve skupiny:\n");
    for (i=1;i<=n;i++) Pridaj(i);
    printf("Skupina 1: ");
    for (i=1;i<=n;i++) if (kde[i]==0) printf("%d ",i);
    printf("\nSkupina 2: ");
    for (i=1;i<=n;i++) if (kde[i]==1) printf("%d ",i);
    putchar('\n');
  }
}
```

2. O byrokracii v Bratislave

opravoval Braňo
(max. 15 bodov)

S výkrikom „Vy ste prišli ku mne s vopred pripraveným úmyslom nevyplniť kolonku!? A to si ako predstavujete, vy hašišáčka jedna!“ vyrazil byrokrat Byrokrat ďalšiu obeť.

Hodnotenie:

- $O(n.\log n)$: **10b**
- $O(n^2)$: **7b**
- backtrack: **max. 2b**
- dôkaz správnosti algoritmu (ak bol správny): **+3b**
- popis algoritmu: **+2b**

Veta: Určite nič nepokazíme, keď občana platiaceho najväčšiu pokutu pošleme na úrad v minútu, kedy má termín.

Dôkaz: Keby prišiel skôr, tak ho môžme vymeniť s tým, čo prišiel v jeho termíne. Nič tým nestratíme a možno ten, s kým sa vymenil, nezaplatí pokutu.

Keby prišiel neskôr, tak ho tiež môžme vymeniť s tým, čo príde v minútu, kedy má termín. Možno potom bude musieť niekto iný namiesto neho zaplatiť pokutu, ale určite sa nezaplatí pokuta väčšia.

Na začiatku nech sú všetky minúty voľné. Vieme, že občana s najväčšou pokutou určite môžeme poslať v poslednú možnú voľnú minútu, aby ešte neplatil pokutu. Takto berieme postupne občanov zoradených od najväčšej pokuty. Vždy sa ho snažíme poslať na úrad v poslednú voľnú minútu, aby ešte stihol. Ak to nie je možné, tak ho necháme platiť pokutu. Správnosť každého kroku tohto algoritmu sa dá odôvodniť podobnými argumentami ako pri dôkaze vety.

Ukážeme si však ešte jeden algoritmus, o ktorom čitateľ po krátkom zamyslení zaiste zistí, že dáva rovnaký výstup. Autorovi sa zdal jednoduchší na efektívne implementovanie.

Pôjdeme po časovej osi od najneskoršej minúty. Nazvime ju t. Označme si tých občanov, ktorí majú termín t. V minúte t pošleme za Byrokratom označeného občana s najväčšou pokutou a odznačíme ho. Ďalej budeme vždy zmenšovať t o jedna a robiť presne to isté, pokiaľ $t >= 0$.

Nakoniec bude $t = 0$. Vtedy tých občanov, ktorí ostali označení, necháme platiť pokutu. A keď už platia pokutu, je úplne jedno kedy prídu. Priradíme im ľubovoľné (ale rôzne) časy.

A ako si už pozorný čitateľ asi všimol, môžme si označených občanov uchovávať v halde. Operácie pridanie označeného občana a vybratie občana s najväčšou pokutou budú trvať $O(\log n)$. Celkova časová zložitosťou algoritmu potom bude $O(n.\log n)$ a pamaťová $O(n)$.

A nakoniec poznámka. Zamyslite sa, či časová zložitosť vašich programov nezávisela od najneskoršej minúty. Síce som za to nestrhával body, ale nabudúce to platiť nemusí.

Listing programu:

```
#include <stdio.h>
#include <stdlib.h>
#include <values.h>
#define MAX 100

typedef struct {
  int deadline,pokuta,obcan,cas;
} TZaznam;

TZaznam A[MAX+2],                    //pracovne pole
  H[MAX+2]={{0,MAXINT,},};   //halda (so zarazkami)
int n, pocH,pokuta;    //n, pocet prvkov v halde, zaplatena pokuta

int deadline_sort_function( const void *a, const void *b) {
   return ((TZaznam *)a)->deadline - ((TZaznam *)b)->deadline;
}

int obcan_sort_function( const void *a, const void *b) {
   return ((TZaznam *)a)->obcan - ((TZaznam *)b)->obcan;
}

void bublaj_dole(int k){
  TZaznam v=H[1];

  while (k<=pocH/2) {
    int j=k+k;
    if (j<pocH)
      if (H[j].pokuta>H[j+1].pokuta) j++;
    if (v.pokuta>=H[j].pokuta)
      break;
    H[k]=H[j];
    k=j;
  }
  H[k]=v;
}

void bublaj_hore(int k){
  TZaznam v=H[k];

  while (H[k/2].pokuta<=v.pokuta) {  //mame zarazku
    H[k]=H[k/2];
    k=k/2;
```

```
  }
  H[k]=v;
}

void vloz(TZaznam v){  //vlozi k-ty prvok z pola A do haldy
  H[++pocH]=v;
  bublaj_hore(pocH);  //utapkame haldu
}

TZaznam vyber(void){ //vyberie najvacsi prvok a ulozi do A[k]
  TZaznam v=H[1];   //najvacsi vratime
  H[1]=H[pocH--];   //utapkame haldu
  bublaj_dole(1);
  return v;
}

void pracuj(void){
int i,j,      //i-miesto citania z A, j-miesto zapisu do A
    t,maxT;   //t - pracovny cas, maxT - maximalny cas

  i=n-1; j=n;
  maxT=t=A[i].deadline;
  A[n+1].pokuta=-1;  //zarazka do haldy
  vloz(A[n+1]);

  while (i>=0) {  //urcime ludi, ktori nebudu platit pokutu
    while (i>=0 && A[i].deadline>=t)  //pridame novych ludi do fronty
      vloz(A[i--]);

    A[--j]=vyber();  //pojde clovek s najvacsou pokutou
    A[j].cas=t;        //priradime mu cas
    t--;

    if (H[1].pokuta==-1 && i>=0)  //ak uz nik nie je vo fronte,
      t=A[i].deadline;            //ponunieme t.
  }

  //ti, ktori sa zvysili v halde, pokutu platit budu
  while (H[1].pokuta!=-1) {
    A[--j]=vyber();
    A[j].cas=maxT+j+1;  //priradime vsetkym casy>n
    pokuta+=A[j].pokuta;
  }
}

int main(void){
  int i;

  scanf("%d",&n);
  for (i=0; i<n; i++) {
    scanf("%d %d",&A[i].deadline,&A[i].pokuta);
```

```
    A[i].obcan=i;
  }

  qsort((void *)A, n, sizeof(A[0]), deadline_sort_function);
  pracuj();
  qsort((void *)A, n, sizeof(A[0]), obcan_sort_function);

  printf("Pokuta: %d\nCasy :",pokuta);
  for (i=0; i<n; i++)
    printf("%d ",A[i].cas);

  putchar('\n');
  return 0;
}
```

opravoval Dávidko
(max. 15 bodov)

3. O zabudnutej krajine

Prišla vcelku pekná hromádka riešení. Pestrosťou prístupu však neoplývali. Asi tretina z vás prišla na riešenie podobné vzorovému. Tým som s radosťou udeľoval po **15 bodov**. Zvyšné dve tretiny boli triviálne riešenia so zložitosťou $O(n^2)$. Tým som udeľoval po **9 bodov**. Dáke tie bodíky mohli zmiznúť, ako obyčajne, za chýbajúce popisy, odhady zložitosti alebo chybičky v programoch.

Vzorové riešenie: Na začiatok kvôli jednoduchosti predpokladajme, že žiadna z priamok nie je zvislá. Na priamky v rovine sa potom možno dívať ako na lineárne funkcie. Našou úlohou je nájsť najpravejší priesečník funkcií t.j. priesečník s maximálnou x-ovou súradnicou. Uvážme zvislú priamku p a všímajme si, v akom poradí zhora nadol pretína jednotlivé funkcie. Na toto sa možno pozerať tak, že funkcie majú v nejakom bode x_0 nejaké poradie.

Posúvajme priamku p postupne zľava doprava. Keď priamka p prejde cez priesečník nejakých dvoch funkcií f a g, tak f a g si vymenia poradie. Aby sa funkcie f a g mohli vymeniť, museli byť v poradí susedné.

V tejto chvíli si stačí všimnúť, že napravo od najpravejšieho priesečníku funkcií sa poradie na priamke p nemení. Toto poradie volajme *finálne*. Navyše vieme, že v najpravejšom priesečníku si nejaké susedné funkcie vymenili poradie. Keby sme poznali finálne poradie, tak stačí skontrolovať priesečníky susedných funkcií a spomedzi týchto priesečníkov vybrať najpravejší.

Napíšme si naše lineárne funkcie v smernicovom tvare $y = kx + q$, kde číslo k je smernica a číslo q posun. Je ľahké všimnúť si, že vo finálnom poradí budú funkcie s väčšou smernicou vyššie a spomedzi funkcií s rovnakou smernicou budú funkcie s väčším posunom vyššie.

Algoritmus je už jasný. Najprv sa vysporiadame so zvislými priamkami. Z nich stačí vybrať najpravejšiu a vyrátať priesečníky s ostatnými priamkami, z nich stačí vybrať ľubovoľný. Zvislými priamkami sa od tejto chvíle netreba zaoberať. Zvyšné funkcie utriedime do finálneho poradia popísaného vyššie. Potom berieme susedné funkcie t.j. funkcie idúce vo finálnom poradí za sebou. Rátame ich priesečníky a spomedzi nich vyberieme najpravejší.

Časová zložitosť triedenia je $O(n.\log n)$, zvyšný výpočet je lineárny, a teda celkovo $O(n.\log n)$. Pamäťová zložitosť je lineárna.

Listing programu:

```
Program Priamky;

const MaxN = 100;
```

```
      Vela = 1e10;    { hrozne velke cislo }
      eps  = 1e-4;    { hrozne male cislo }

type priamka = record
                     K,Q:real; { y = k*x + q }
                     zvisla : boolean; { ak je priamka zvisla, tak x=q }
               end;

var x1,y1,x2,y2:real; { 2 body }
    a:array[1..MaxN] of priamka; { priamky }
    maxx,maxy:real; { suradnice najpravejesieho bodu }
    N,M,i:integer;  { N - pocet priamok, M - pocet nie zvislych priamok }

{ vymena dvoch priamok }
Procedure Vymen(var p,q:priamka);
var r:priamka;
begin
     r:=p; p:=q; q:=r;
end;

{ porovnanie dvoch priamok }
Function Porovnaj(p,q:priamka):boolean;
begin
     Porovnaj:=true;
     if p.k<q.k then exit;    { p ma mensiu smernicu }
     { ak maju rovnaku smernicu a p ma mensi posun }
     if (abs(p.k-q.k)<eps) and (p.q<q.k) then exit;
     Porovnaj:=false;
end;

{ triedenie priamok podla smernice }
Procedure Quicksort(l,r:integer);
var i,j: integer; p:priamka;
begin
  i:=l; j:=r; p:=a[(l+r) div 2];
  repeat
    while Porovnaj(a[i],p) do inc(i);
    while Porovnaj(p,a[j]) do dec(j);
    if i <= j then
    begin
      Vymen(a[i],a[j]);
      inc(i); dec(j);
    end;
  until i>j;
  if l<j then Quicksort(l,j);
  if i<r then Quicksort(i,r);
end;

Begin
     assign(input,'priamky.in');
     assign(output,'priamky.in');
```

```
reset(input);
rewrite(output);

readln(N);    { nacitame N }
for i:=1 to N do   { nacitame priamky }
with a[i] do
begin
     readln(x1,y1,x2,y2);

     zvisla:=abs(x1-x2)<eps;        { ci je priamka zvisla }
     { ak priamka nie je zvisla, zratame jej smernicu }
     if zvisla then q:=x1
     else
     begin
          K:=(y1-y2)/(x1-x2);
          Q:=x1-k*y1;
     end;
end;

maxx:= -Vela;
for i:=1 to N do        { zo zvislych priamok najdeme najpravejsiu - R }
  with a[i] do if zvisla and (q>maxx) then maxx:=q;

{ vyhodime zvisle priamky a najdeme priesecnik s R}
M:=0;
for i:=1 to N do
with a[i] do
if not zvisla then   { priamka nie je zvisla ! }
begin
     maxy:=k*maxx+q; { zratame priesecnik s R}
     inc(M);         { pocet nie zvislych priamok }
     a[M]:=a[i];
end;

Quicksort(1,M);    { utriedime priamky }

{ zratame priesecniky dvoch po sebe iducich priamok }
{ a najpravejsi z nich si zapamatame }
for i:=2 to M do
begin
     x1:=(a[i-1].q-a[i].q)/(a[i].k-a[i-1].k);
     y1:=a[i].k * x1 + a[i].q;
     if x1>maxx then
     begin
          maxx:=x1;
          maxy:=y1;
     end;
end;

{ vypiseme vystup }
if maxx>-Vela then
```

```
        Write('Najpravejsi priesecnik ma suradnice (',
        maxx:0:2 ,',', maxy:0:2,')')
     else
        Writeln('Priamky nemaju priesecnik.');
End.
```

opravoval MišoF
(max. 15 bodov)

4. O nu(d/t)nej prednáške

Nuž, takýto neobľúbený príklad sme v KSP už dávno nemali... Len 8 z vás vôbec niečo poslalo a ani tie riešenia neboli všetky dokonalé. Ale chápem vás, ani mne sa písať vzorák veľmi nechcelo. Aj keď nakoniec nie je až taký dlhý, ako som sa bál. Úbohých 200 riadkov, čo to je? Veď sa presvedčte sami. Večery sú stále ešte dlhé, a aby ste sa počas nich nenudili a mali čo čítať, bude dlhý aj tento vzorák.

No ale prejdime k vzorovému riešeniu. Čo sme od vás vlastne chceli? Takéto niečo: Predstavme si, že by sme vedeli všetky možné rozloženia lodičiek, ktoré zodpovedajú hracej ploche na vstupe. Predpokladajme navyše, že sme sa už opýtali na tie políčka, na ktorých majú všetky možné rozloženia loďku. Ako teraz zistíme, či sa dá nájsť ich skutočné rozloženie s nanajvýš jednou chybou? Keď sa opýtame na nejaké políčko, sú dve možnosti – buď trafíme loďku a máme vybrať jedno z rozložení, ktoré na tom mieste majú loďku, alebo loďku netrafíme a v tomto prípade je správne niektoré z rozložení, ktoré na tomto mieste loďku nemajú. Lenže ak máme nájsť rozloženie nanajvýš s jednou chybou a tú sme práve spravili, už sa nemôžeme ďalej pýtať a musíme rovno povedať výsledok. To ale znamená, že nám mohlo zostať len jedno rozloženie.

To už dáva návod, ako sa pýtať. Pokiaľ existuje také políčko, na ktorom má jedno možné rozloženie vodu a všetky ostatné loďku, môžeme sa naň opýtať. Nič tým nestratíme – buď sa dozvieme, že je tam loďka, vypadne nám jedno rozloženie a pokračujeme, alebo sa dozvieme, že je tam voda a vyhrali sme. Pokiaľ niekedy také políčko neexistuje, zjavne sa na žiadne políčko nemôžeme opýtať – keby sme dostali odpoveď voda, nevedeli by sme sa rozhodnúť. Preto v takomto prípade sa to nedá.

Podľa vyššie uvedených myšlienok sa už dalo napísať funkčné riešenie. Nájdeme všetky rozloženia lodiek vyhovujúce vstupu. Potom simulujeme pýtajúceho sa hráča – vždy si nájdeme políčko, na ktorom má práve jedno rozloženie vodu a príslušné rozloženie vyhodíme. Pokiaľ skončíme s jedným rozložením, dá sa to, pokiaľ sa niekedy nevieme opýtať, nedá sa to.

Všetkých rozložení lodiek však môže byť nepríjemne veľa a keď už nič iné, nemusia sa nám vôjsť do pamäte. Jedna možnosť bola povedať si: „Bude ich najviac toľko, čo mi vôjde do pamäte, a ak náhodou nie, tak čo už...“ Korektnejšie (ale pomalšie) bolo vždy znova všetky generovať.[1] No a vzorové riešenie bolo ešte sa zamyslieť.

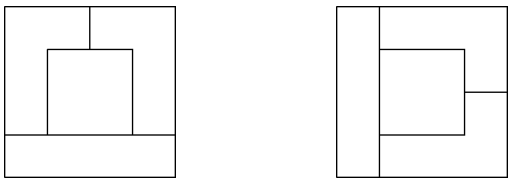
Tvrdím, že keď vstupu zodpovedá viac ako 29 rozložení, skutočné rozloženie už s najviac jednou chybou určiť nevieme. Prečo? Lebo zakaždým, keď sme jedno rozloženie vyhodili, znamenalo to, že sme sa pýtali na nejaké políčko a dostali odpoveď, že je na ňom loďka. No a loďky spolu zaberajú 28 políčok, preto sme mohli vyhodiť najviac 28 rozložení. V tomto okamihu nám však malo zostať práve 1 rozloženie. Preto ak sa to dá, je rozložení najviac 29.

Alebo pre zaujímavosť to isté trochu ináč. Každý pascalovský program je deterministický. (to si môžete predstaviť tak, že v okamihu, keď ho spustíte na nejaký vstup, viete povedať, aký dá výstup) Teraz si predstavme, že by sme mali program, ktorý dostane na vstupe pozíciu a simuluje hráča, čiže pýta sa na políčka a snaží sa nájsť správne rozloženie s najviac jednou chybou. Na svoje otázky môže tento program dostať 29 rôznych postupností odpovedí

[1] Tomuto sa hovorí trade-off (medzi časom a pamäťou) — zjednodušene povedané: čím menej máme pamäte na riešenie problému, tým dlhší čas nám to bude trvať.

— 0 . . . 27-krát loďka a nakoniec loďka alebo voda. (Akonáhle dostane odpoveď voda, musí povedať rozloženie, podobne aj keď odkryje všetky loďky.) Pre každú z nich vieme povedať, aký dá výstup, teda ktoré rozloženie vypíše. Keby to šlo pre nejaký prípad, v ktorom je možných rozložení viac ako 29, niektoré z nich by určite nevypísal nikdy, to je ale spor s tým, že je to možné rozloženie a že to v tomto prípade ide. Preto to pre viac ako 29 rozložení nejde. Podobná metóda sa používa napríklad v dôkaze, že každé triedenie porovnávaním má časovú zložitosť $\Omega(n.\log n)$.

Takže si stačí pamätať 29 rozložení, čo už do pamäte pohodlne vôjde. A akonáhle nájdeme tridsiate, môžeme skončiť s odpoveďou nie. A ešte k jednej drobnej chybičke. Zahľaďte sa na tieto dva obrázky:



Rozloženia lodiek sú síce iné, ale riešenie je to to isté! Ale keby sme si ho zapamätali dvakrát, dá náš program nesprávnu odpoveď, lebo by ich pochopiteľne nevedel rozlíšiť. Preto keď si ideme zapamätať rozloženie, musíme si najskôr skontrolovať, či také už náhodou nemáme.[2]

Na záver, ako som vaše riešenia hodnotil?

- Nesprávne riešenia – max. **2 body**
- Čiastočne správne riešenia – max. **5 bodov**
- Hľadanie všetkých rozložení – max. **12 bodov**
- Vzorové riešenie – max. **15 bodov**

Z tohto skromného počtu bodov sa ešte dalo stratiť za chýbajúci, prípadne nezrozumiteľný popis a za vyššie uvedenú drobnú chybičku. Do záporu sa dostať (našťastie `:-)`) nedalo. . .

Listing programu:

```
program Lodicky___KSP_18_2_4___by_MisoF;
const maxr = 15; {maximalny rozmer mapy}
const poc_poloh : array[1..7] of integer = (1,2,4,4,4,2,2);
      {kolko ma ktora lodka roznych poloh}
      lodky : array[1..7,1..4,1..4,1..4] of byte = (
      {cislo lodky, cislo polohy, suradnice}
      ( {lodka tvaru O}
       ((1,1,0,0),(1,1,0,0),(0,0,0,0),(0,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))),
      ( {lodka tvaru I}
       ((1,1,1,1),(0,0,0,0),(0,0,0,0),(0,0,0,0)),
       ((1,0,0,0),(1,0,0,0),(1,0,0,0),(1,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0)),
```

[2] Pamätáte si ešte myšlienku neukladať všetky konfigurácie, ale vždy ich znovu generovať? Teraz vidíme, že keď vygenerujeme konfiguráciu, potrebujeme ešte overiť, či sme ju už nemali – teda znovu generovať všetky od začiatku až po ňu a porovnať ju s nimi. Takéto riešenie je natoľko pomalé, že má iba teoretický význam, ale stále je polynomiálne.

```
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))),
      ( {lodka tvaru L}
       ((1,0,0,0),(1,0,0,0),(1,1,0,0),(0,0,0,0)),
       ((0,0,1,0),(1,1,1,0),(0,0,0,0),(0,0,0,0)),
       ((1,1,0,0),(0,1,0,0),(0,1,0,0),(0,0,0,0)),
       ((1,1,1,0),(1,0,0,0),(0,0,0,0),(0,0,0,0))),
      ( {lodka tvaru J}
       ((0,1,0,0),(0,1,0,0),(1,1,0,0),(0,0,0,0)),
       ((1,1,1,0),(0,0,1,0),(0,0,0,0),(0,0,0,0)),
       ((1,1,0,0),(1,0,0,0),(1,0,0,0),(0,0,0,0)),
       ((1,0,0,0),(1,1,1,0),(0,0,0,0),(0,0,0,0))),
      ( {lodka tvaru T}
       ((1,1,1,0),(0,1,0,0),(0,0,0,0),(0,0,0,0)),
       ((0,1,0,0),(1,1,0,0),(0,1,0,0),(0,0,0,0)),
       ((0,1,0,0),(1,1,1,0),(0,0,0,0),(0,0,0,0)),
       ((1,0,0,0),(1,1,0,0),(1,0,0,0),(0,0,0,0))),
      ( {lodka tvaru S}
       ((0,1,1,0),(1,1,0,0),(0,0,0,0),(0,0,0,0)),
       ((1,0,0,0),(1,1,0,0),(0,1,0,0),(0,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))),
      ( {lodka tvaru Z}
       ((1,1,0,0),(0,1,1,0),(0,0,0,0),(0,0,0,0)),
       ((0,1,0,0),(1,1,0,0),(1,0,0,0),(0,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0)),
       ((0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))));

type tpole = array[1..maxr,1..maxr] of integer;
var M,N : integer; {rozmery plochy}
    pov : array[0..maxr+1,0..maxr+1] of integer;
    {povodne postavenie}
    teraz : tpole; {aktualne postavenie}
    roz : array[1..29] of tpole;
    {vyhovujuce rozlozenia lodiciek}
    nasiel : integer;
    {kolko vyh. rozlozeni uz nasiel}

procedure Nacitaj;
var F : text;
    i,j : integer;
    c : char;
begin
   assign(F,'lodicky.in'); reset(F);
   readln(f,m,n);
   for i:=1 to m do begin
      for j:=1 to n do begin
         read(f,c);
         case c of
            '+' : pov[i,j]:=1;
            '-' : pov[i,j]:=-1;
            '.' : pov[i,j]:=0;
```

```
         end;
      end; readln(F);
   end;
   close(F);
   {spravime si okolo plochy zarazku z -1, aby lodky
    nemohli trcat von}
   for i:=0 to m+1 do begin pov[i,0]:=-1; pov[i,n+1]:=-1; end;
   for j:=0 to n+1 do begin pov[0,j]:=-1; pov[m+1,j]:=-1; end;
   nasiel:=0;
end;

function Kontrola(co,ako,x,y : integer) : boolean;
{ci na dane miesto mozemedat lodku}
var i,j : integer;
begin
   Kontrola:=false;
   for i:=1 to 4 do for j:=1 to 4 do
      if lodky[co,ako,i,j]=1 then
         if pov[x+i-1,y+j-1]=-1 then exit;
         {je niekde, kde nema byt}
   for i:=1 to 4 do for j:=1 to 4 do
      if lodky[co,ako,i,j]=1 then
         if teraz[x+i-1,y+j-1]=1 then exit;
         {je niekde, kde uz nejaka je}
   Kontrola:=true;
end;

function Hotovo : boolean;
{ci sme pokryli vsetko, co sme mali}
var i,j : integer;
begin
   Hotovo:=false;
   for i:=1 to m do for j:=1 to n do
      if (pov[i,j]=1) and (teraz[i,j]=0) then exit;
      {nie je niekde, kde ma byt}
   Hotovo:=true;
end;

function Este_nemam : boolean;
{ci si danu konfiguraciu este nepamatam}
var k,i,j : integer;
    rozna : boolean;
begin
   Este_nemam:=false;
   for k:=1 to nasiel do begin
      rozna:=false;
      for i:=1 to m do for j:=1 to n do
         if roz[k][i,j]<>teraz[i,j] then begin
            rozna:=true; break; break;
         end;
      if not rozna then exit;
```

```
   end;
   Este_nemam:=true;
end;

procedure Skus(co,ako,x,y : integer);
var i,j,k : integer;
begin
   {skontroluje, ci ju tam moze dat}
   if not Kontrola(co,ako,x,y) then exit;
   {zaznaci si ju}
   for i:=1 to 4 do for j:=1 to 4 do
      if lodky[co,ako,i,j]=1 then teraz[x+i-1,y+j-1]:=1;
   if co=7 then begin
      {umiestnili sme poslednu lodku, treba
       skontrolovat, ci sme pokryli vsetky polia,
       ktore sme mali}
      if Hotovo then if Este_nemam then begin
         inc(nasiel);
         if nasiel=30 then begin {uz ich je privela}
            writeln('NIE'); halt;
         end;
         roz[nasiel]:=teraz;
      end;
   end else begin
      for k:=1 to poc_poloh[co+1] do
         for i:=1 to m do for j:=1 to n do Skus(co+1,k,i,j);
   end;
   {opaet odznaci lodku}
   for i:=1 to 4 do for j:=1 to 4 do
      if lodky[co,ako,i,j]=1 then teraz[x+i-1,y+j-1]:=0;
   {a vrati sa spaet skusat dalej}
end;

procedure Najdi_pozicie;
var i,j : integer;
begin
   fillchar(teraz,sizeof(teraz),0);
   {skusame postupne umiestnit kazdu zo 7 lodiciek}
   for i:=1 to m do for j:=1 to n do Skus(1,1,i,j);
   {cislo lodicky, cislo polohy, sur. laveho horneho rohu}
end;

procedure Over_riesitelnost;
var pocty : tpole;
    i,j,k,x,y : integer;

begin
   fillchar(pocty,sizeof(pocty),0);
   for k:=1 to nasiel do
      for i:=1 to m do for j:=1 to n do
         pocty[i,j]:=pocty[i,j]+roz[k][i,j];
```

```
   x:=-1;
   {najde vzdy odobratelnu poziciu a odstrani prislusnu
    konfiguraciu (ktora tam jedina ma vodu)}
   while nasiel>1 do begin
      for i:=1 to m do for j:=1 to n do
         if pocty[i,j]=nasiel-1 then begin
            x:=i; y:=j; break;
         end;
      if x=-1 then begin   {nenasiel policko, na ktorom nelezi len 1 z nich}
         writeln('NIE'); halt;
      end;
      for k:=1 to nasiel do if roz[k][x,y]=0 then break;
      {k-tu konfiguraciu mozeme vyhodit}
      for i:=1 to m do for j:=1 to n do
         pocty[i,j]:=pocty[i,j]-roz[k][i,j];
      if k<nasiel then roz[k]:=roz[nasiel];
      dec(nasiel);
   end;
end;

begin
   Nacitaj;
   Najdi_pozicie;
   {nasiel nieco, je ich najviac 29}
   if nasiel=0 then begin
      writeln('Pozicia nema riesenie !!!'); halt;
   end;
   Over_riesitelnost;
   {a ak sa dostal az sem, vyhrali sme}
   writeln('ANO');
end.
```

opravoval Rišo
(max. 15 bodov)

5. O Santolande II

Podľa počtu riešiteľov tohto príkladu vidno, že tento príklad patril medzi tie ťažšie. Viacerí z vás vyriešili aspoň časť a), aj keď sa im nepodarilo zovšeobecniť svoj postup a vytvoriť algoritmus na prevod ľubovoľnej jaskyne. Žiaľ, vyskytli sa aj opačné prípady – niektorí z vás zle pochopili zadanie a mysleli si, že keď vyriešia úlohu b), nemusia riešiť úlohu a). Keďže však obe úlohy boli bodované nezávisle, mohli ste takto stratiť 5 bodíkov.

A teraz k bodovaniu: Za časť a) ste mohli získať najviac **5 bodov**. Toľko bodov ste mohli dostať za ľubovoľnú jaskyňu, ktorá spĺňa podmienky v zadaní. Za drobné chyby ste mohli stratiť **2 body** (aj drobné chyby totiž robia jaskyňu nevyhovujúcu zadaniu). V prípade celkom zlej jaskyne ste mohli získať za snahu najviac **1 bod**.

Za časť b) ste mohli získať najviac **10 bodov**. Za program s časovou zložitosťou v najhoršom prípade $O(2^n(m+n))$, kde n je počet komôr v pôvodnej jaskyni a m je počet tunelov v pôvodnej jaskyni ste mohli získať plný počet bodov. Za horšie riešenia (jediná horšia časová zložitosť korektných programov, ktorá sa vyskytla, bola $O(4^n(m+n))$) ste mohli získať **8 bodov**. Za zlé riešenia ste mohli za snahu získať najviac **1 bod**. **1 až 2 body** ste mohli stratiť za nepresnosti v odhade zložitosti (ako ešte spomeniem, v tomto príklade je odhad najmä pamäťovej zložitosti nepríjemný, preto som bol pri hodnotení zložitosti dosť tolerantný), drobné chyby a subjektívny dojem.

Vzorové riešenie:

Najskôr sa zamyslime, čo by urobil Alibaba a jeho nekonečno zbojníkov, keby zavítali do Santovej jaskyne. Najskôr by všetci vošli do prvej komory. Potom by sa presunuli do ďalšej komory podľa prvého písmena na papieriku. V prípade, že by mali viac možností, do ktorého tunela vojsť, rozdelili by sa na príslušný počet skupín. Keďže zbojníkov je nekonečno, bude v každej komore, kam sa mohli dostať, opäť nekonečno zbojníkov. Ďalej všetci zbojníci vo všetkých komorách naraz zopakujú tento postup pre druhé písmenko na papieriku, atď. Po prečítaní celého papieriku teda bude vo všetkých komorách, kam sa s papierikom dá dostať, nekonečno zbojníkov a vo všetkých ostatných komorách nebude nikto. Ak teda niektorý zo zbojníkov našiel fľašu rumu, Alibaba mohol s pôvodným papierikom túto fľašu nájsť, v opačnom prípade rum nájsť nemohol.

Keď bude Santo prerábať jaskyňu, môže sa inšpirovať týmto postupom. Pre každú podmnožinu komôr pôvodnej jaskyne vytvorí komoru v novej jaskyni. Zatiaľ teda budeme v našich úvahách označovať komory novej jaskyne nie číslami, ale podmnožinami komôr pôvodnej jaskyne. Ak sa bude Alibaba nachádzať v niektorej komore $M = \{a_1, \ldots, a_k | a_1 \ldots a_k \in \{1 \ldots n\}\}$ novej jaskyne, bude to zodpovedať situácii, že sa Alibabovi zbojníci nachádzajú v komorách $a_1 \ldots a_k$ pôvodnej jaskyne a nijakých iných.

Ak sa Alibaba nachádza v komore M, môže sa po prečítaní nejakého písmena p dostať do takej komory M', kam prejdú jeho zbojníci z komôr $a_1 \ldots a_k$ po prečítaní písmena p. Teda $M' = \{b | \text{existuje } a \in M \text{ také, že z } a \text{ sa dá dostať do } b \text{ písmenom } p \text{ v pôvodnej jaskyni.}\}$. Takýmto spôsobom sme jednoznačne určili, kam má Alibaba prejsť po prečítaní písmena p. Nová jaskyňa teda bude spĺňať podmienku, aby na jedno písmeno z určitej komory nevychádzal viac ako jeden tunel.

A do ktorých komôr novej jaskyne má Santo umiestniť rum? Zjavne ho musí umiestniť do tých komôr novej jaskyne M, ktoré zodpovedajú situácii, že nejaký Alibabov zbojník našiel rum. Teda ak $M = \{a_1, \ldots, a_k\}$, v komore M je fľaša rumu práve vtedy, ak v niektorej z komôr $a_1 \ldots a_k$ v pôvodnej jaskyni bola fľaša rumu.

Ako už bolo povedané, Alibaba sa nachádza po prečítaní niekoľkých písmen v takej komore novej jaskyne, ktorá zodpovedá tej množine komôr pôvodnej jaskyne, kam by sa mohol po prečítaní tých istých písmen dostať. V komore je rum práve vtedy, ak by po prečítaní príslušných písmen mohol v pôvodnej jaskyni rum nájsť. Preto nová jaskyňa spĺňa podmienku, že rum sa v nej dá nájsť práve s tými papierikmi, s ktorými sa dal nájsť v pôvodnej jaskyni.

Poznámka pre zvedavých: Ak si ešte spomínate na vzorové riešenia prvej série, bolo v nich povedané, že Santove jaskyne zodpovedajú v teórii formálnych jazykov konečným automatom. Pôvodná Santova jaskyňa v tomto príklade zodpovedá nedeterministickému konečnému automatu. Vašou úlohou bolo nedeterministický automat previesť na deterministický, t.j. taký, kde je výpočet jednoznačne určený. Ak vás táto téma zaujala, môžeme vám odporučiť napr. knižku Hopcroft, Ullman: Formálne jazyky, alebo si počkať na prednášku v druhom ročníku FMFI UK.

Implementácia: Pri programovaní môžme použiť dva rôzne postupy. Prvý z nich je, že budeme vytvárať vždy jaskyňu s 2^n komorami, kde n je počet komôr pôvodnej jaskyne. Pri tomto postupe stačí generovať všetky podmnožiny komôr pôvodnej jaskyne a pre každú z nich pre každé písmeno vypočítať, do akej komory v novej jaskyni sa dá prejsť. Pre tento výpočet treba v najhoršom prípade prezrieť všetky tunely pôvodnej jaskyne. Pri výpise musíme previesť podmnožinu čísel 1 až n do čísla 1 až 2^n. Toto môžme spraviť napríklad binárnym kódovaním. Vo výslednom čísle bude $i - 1$ bit rovný 1 práve vtedy, ak sa číslo i nachádzalo v podmnožine. Nakoniec ešte musíme 0 zmeniť na 2^n. Pri tomto kôdovaní zároveň zostane zachované, že začiatočná komora bude mať číslo 1. Časová zložitosť tohoto algoritmu je $O(2^n(m+n))$ a pamäťová $O(n+m)$.

Tento algoritmus má z hľadiska najhoršieho prípadu najlepšiu časovú aj pamäťovú zložitosť. Pre rozumné využitie výslednej jaskyne je však lepšie nevytvárať zbytočné komory.

Pri takomto riešení vlastne prehľadávame jaskyňu do šírky resp. do hĺbky a vytvárame len tie komory, ktoré sú potrebné (to však neznamená, že jaskyňa obsahuje najmenší možný počet komôr!). Pri tomto prehľadávaní si musíme pamätať komory novej jaskyne, ktoré sme vytvorili a keďže ich je najviac 2^n, vzrastie nám pamäťová náročnosť.

Pri odhade pamäťovej náročnosti takýchto riešení ste sa však dopúštali takejto nepresnosti. Tým, že ste stav novej jaskyne kódovali do jedného čísla, predpokladali ste, že komora novej jaskyne sa dá zapamätať v pamäti $O(1)$. To však nie je pravda, lebo komora novej jaskyne predstavuje podmnožinu čísel 1 až n, a teda na jej zapamätanie je treba pamäť $O(n)$!

Riešenia druhého typu mohli byť implementované rôzne. Najlepšie riešenie využívalo okrem fronty na prehľadávanie do šírky binárny strom, v ktorom boli zapamätané vytvorené komory novej jaskyne. Na zistenie, či určitá komora bola vytvorená, potom stačil čas $O(n)$. Najmenej prefíkané riešenia na zistenie, či bola daná komora už vytvorená, potrebovali čas $O(n2^n)$, lebo prezerali všetky doteraz vytvorené komory. Tým ich časová zložitosť vzrástla na $O(4^n(m+n))$.

A teraz je už čas na listing vzorového riešenia (je to riešenie prvého typu).

Listing programu:

```
program O_Santolande_II;

const MAX=30;                  { Maximalny pocet stavov povodneho }
                               { automatu }
type pzoznam=^tzoznam;         { Zoznam stavov }
     tzoznam=record
               v:integer;
               n:pzoznam;
             end;
     tmnozina=set of 1..MAX;   { Mnozina stavov }

var N:integer;                 { Pocet stavov povodneho automatu }
    A:array[1..MAX,'a'..'z'] of pzoznam; { Tabulka prechodov }
    Rum:tmnozina;              { Koncove vrcholy povodneho automatu }
    nrum:longint;              { Pocet koncovych vrcholov noveho automatu }
    act:tmnozina;              { Spracovavany vrchol noveho automatu }

procedure Pridaj(var z:pzoznam; v:integer); { Prifaj vrchol do spajaneho }
var p:pzoznam;                              { zoznamu }
begin
 new(p);
 p^.v:=v;
 p^.n:=z;
 z:=p;
end;

procedure Nacitaj;      { Nacitaj vstup }
var i,j,k,m:integer;
    ch,p:char;
begin
 read(N);
 for i:=1 to N do
   for ch:='a' to 'z' do A[i,ch]:=nil;
```

```
 read(m);
 for k:=1 to m do
 begin
   readln(i,j,p,ch);
   Pridaj(A[i,ch],j);
 end;
 read(m);
 for k:=1 to m do
 begin
   read(i);
   Rum:=Rum+[i];
 end;
end;

function CisloStavu(s:tmnozina):longint; { Preved mnozinu stavov }
var c:longint;                           { povodneho automatu }
    i:integer;                           { na cislo stavu noveho automatu }
begin
 c:=0;
 for i:=N downto 1 do
 begin
   c:=2*c;
   if i in s then c:=c+1;
 end;
 if c=0 then c:=(1 shl N);
 CisloStavu:=c;
end;

procedure PisHrany;           { Vypis prechody zo stavu act noveho automatu }
var ch:char;
    stav:tmnozina;
    i:integer;
    p:pzoznam;

begin
 for ch:='a' to 'z' do        { Pre kazde pismeno }
 begin
   stav:=[];                  { Zistime, kam sa da dostat }
   for i:=1 to n do
    if i in act then
    begin
      p:=A[i,ch];
      while (p<>nil) do
      begin
        stav:=stav+[p^.v];
        p:=p^.n;
      end;
    end;
   writeln('(',CisloStavu(act),',',CisloStavu(stav),',',ch,')');
 end;
```

```
 for i:=1 to n do             { Zistime, ci je stav koncovy }
  if i in act then
   if i in Rum then
   begin
     inc(nrum);               { Ak ano, zvysime pocet koncovych stavov }
     break;
   end;
end;

procedure PisVrcholy(v:integer);  { Prejdi vsetky vrcholy noveho automatu }
begin
 if v>N then
 begin
   PisHrany;                  { A pre kazdy vypis prechody na vsetky pismena }
   exit;
 end;
 PisVrcholy(v+1);
 act:=act+[v];
 PisVrcholy(v+1);
 act:=act-[v];
end;

procedure PisRum(v:integer);  { Vypis koncove stavy noveho automatu }
var i:integer;
begin
 if v>N then
 begin
   for i:=1 to N do
    if (i in act)and(i in Rum) then
    begin
      writeln(CisloStavu(act));
      break;
    end;
   exit;
 end;
 PisRum(v+1);
 act:=act+[v];
 PisRum(v+1);
 act:=act-[v];
end;

begin
 Nacitaj;                                 { Nacitaj vstup }
 writeln('Pocet komor: ',(1 shl N));      { Novy automat ma 2^N stavov }
 act:=[];
 nrum:=0;
 PisVrcholy(1);                           { Vypis prechody }
 writeln('Pocet komor s rumom: ',nrum);
 act:=[];
 PisRum(1);                               { Vypis koncove stavy }
end.
```

Výsledková listina po 2. sérii kategórie KSP

	Meno a priezvisko		Škola		21	22	23	24	25	Σ
1.	Záthurecký Tomáš	6F	Gym. V.Paulínyho-T Martin	70	15	15	15	14	13	142
2.	Dvorský Marián	4C	Gym. Košice, Šrobárova	70	15	15	15	1	15	131
3.	Bella Peter	3C	Gym. J. Hronca Bratislava	60	12	12	15		13	112
4.	Juhos Pavol	3	Gym. A.Markuša Bratislava	56	15	15	15		9	110
5.	Glaus Michal	4A	Gym. J. Hronca Bratislava	56	15	15	15		5	106
6.	Rudišin Miroslav	4	Gym. Košice, Šrobárova	68		13	15		5	101
7.	Tvarožek Jozef	3C	Gym. J. Hronca Bratislava	61		12	15		7	95
8.	Tvarožek Michal	4C	Gym. J. Hronca Bratislava	61		12	15			88
9.	Gaži Peter	4	Gym. A.Markuša Bratislava	39		12	15		12	78
10.	Krčah Peter	4	Gym. Nitra, Párovská 1	44		12	9	10		75
	Dzetkulič Tomáš	3A	Gym. P.Horova Michalovce	52	2	9	9	3		75
	Katrenič Ján	3B	Gym. Spiš. Nová Ves, Školská	43	6	12	9		5	75
13.	Oravec Ján	4	Gym. Tajovského B.Bystrica	72						72
	Pokorný Michal	4	Gym. A.Markuša Bratislava	33		15	15		9	72
15.	Hríbik Lukáš	1B	Gym. A.Merici Trnava	48	1	12	9			70
	Horváth Anton	4A	Gym. Sereď	42	2	11	9	1	5	70
17.	Truchlý Peter	3	Gym. Nitra, Párovská 1	43	2	9	15			69
18.	Burdiliak Boris	4C	Gym. J. Hronca Bratislava	46		6	15			67
	Veselý Jozef	2B	Gym. Nitra, Golianova 68	47	1	2	9		8	67
20.	Sághy Tomáš	3	SPŠE Nové Zámky, Pod kopcom	51	2	2	9			64
21.	Klempai Michal	4B	Gym. A.Bernoláka Námestovo	37	2	2	9	11	1	62
22.	Haraga Dávid	4	Gym. Tajovského B.Bystrica	61						61
	Macko Martin	4C	Gym. Spiš. Nová Ves, Školská	61						61
24.	Vešelényi Michal	5L	Gym. Tajovského B.Bystrica	42		9	9			60
25.	Bauer Radovan		Gym. Košice, Poštová	57						57
	Čulák Radovan	1B	Gym. Nitra, Golianova 68	39	2	8	8			57
27.	Müller Pavol	3	Gym. A.Markuša Bratislava	55						55
	Bombiak Tibor	4	Gym. A.Bernoláka Námestovo	55						55
	Glaus Peter	2B	Gym. J. Hronca Bratislava	32		9	9		5	55
30.	Ludha Marek	1F	Gym. Tajovského B.Bystrica	37		7	9			53
31.	Bednárik Ľuboš	3F	Gym. Ľ.Štúra, Trenčín	41			9			50
	Palát Michal	4	Gym. A.Markuša Bratislava	37	1	2	9	1	0	50
33.	Miklian Peter	4B	Gym. B.S.Timravy Lučenec	37			9		3	49
34.	Havlůj František	4	.....	48						48
35.	Kucharík Ladislav		SPŠE Piešťany, SNP	45						45
	Lehotský Jakub	4A	Gym. Liptovský Hrádok	45						45
	Steskal Ľuboš	4	Gym. A.Markuša Bratislava	34			10	1		45
38.	Buranský Rado	4C	Gym. J. Hronca Bratislava	44						44
	Rjaško Michal	2A	Gym. Vranov n/T, Daxnerova	25	2	8	9			44
	Kolínsky Miro	3	Gym. A.Markuša Bratislava	21	2	12	9			44
41.	Riha Ondrej	3	Gym. A.Markuša Bratislava	33			9			42
42.	Mikuš Michal	3C	Gym. J. Hronca Bratislava	24	2		15			41
43.	Nourani Sana		Gym. J. Hronca Bratislava	40						40
	Florek Martin	4	Gym. A.Markuša Bratislava	31			9			40
45.	Vlčková Zuzana	4	Gym. Košice, Alejová	22		10	5		2	39
46.	Rodák Roman	2B	Gym. A.Merici Trnava	26	2		9			37
47.	Revay Marián	3B	Gym. Čadca	35						35
	Psotka František	4A	Gym. Košice, Opatovská	35						35
	Sekera Klement		Gym. A.Markuša Bratislava	35						35
50.	Lakatos Tomáš	4E	Gym. Nové Zámky	32						32

	Meno a priezvisko		Škola		21	22	23	24	25	Σ
51.	Slovik Vojtech	4C	Gym. J. Hronca Bratislava	31						31
	Berta Radovan	2A	Gym. Vranov n/T, Daxnerova	24		7				31
	Chromek Daniel		Gym. B.S.Timravy Lučenec	31						31
	Burdiliak Branislav		Gym. J. Hronca Bratislava	0	4	12	15			31
55.	Baka Kristián	4C	Gym. Levice	20		2	8			30
56.	Kollár Ivor	2C	Gym. J. Hronca Bratislava	29						29
	Pelák Jakub	1D	Gym. L.Stockela Bardejov	26	1	2				29
58.	Havlíček Libor	3B	.....	28						28
	Mazák		.....	28						28
	Plavčan Juraj	4A	Cirk. gym. Bratislava, Čachtická	28						28
	Lovíšek Peter	4	Gym. Považská Bystrica	28						28
62.	Bobok Pavol	4C	Gym. Levice	17		2	8			27
	Urban Jaroslav	3A	Gym. Liptovský Hrádok	27						27
	Farinič Igor	4A	Gym. Snina	27						27
65.	Cifra Marek	1B	Gym. A.Merici Trnava	15		9				24
66.	Piatka Peter	4E	Gym. Ľ.Štúra, Trenčín	22						22
67.	Rybár Pavol		Cirk. gym. Bratislava, Čachtická	21						21
	Kopčanský Peter	3A	Gym. Krompachy	19		2				21
	Sýkora Peter	4	Ev. gym. B.Bystrica	21						21
	Žbodák Michal	4	Gym. Považská Bystrica	21						21
	Tekeľ Michal	3E	Gym. Prešov, Konštantínova	15	2	2			2	21
72.	Vranec Maroš	2A	Gym. Košice, Alejová	20						20
	Haluzová Soňa	2C	Gym. J. Hronca Bratislava	20						20
	Rybár Tomáš	2C	Gym. J. Hronca Bratislava	20						20
75.	Pekar Andrej	8A	Gym. A.Merici Trnava	19						19
76.	Lipková Juliana		Gym. J. Hronca Bratislava	8		0	8			16
	Vasilová Eva	4A	Gym. P.Horova Michalovce	16						16
78.	Zika Martin	2C	Gym. J. Hronca Bratislava	14						14
79.	Šišková Jana	3	Ev. gym. B.Bystrica	13						13
	Novotný Lukáš	2C	Gym. J. Hronca Bratislava	13						13
81.	Dzurjanin Peter	3	Gym. A.Markuša Bratislava	11						11
82.	Mikuš Tomáš	2D	Gym. J. Hronca Bratislava	10						10
83.	Hudák Miroslav	3B	Gym. Nitra, Golianova 68	9						9
	Trubeňová Barbora	1A	Gym. J. Hronca Bratislava	0			9			9
85.	Totušek Zdenko	6F	Gym. V.Paulínyho-T Martin	8						8
86.	Tomeček	2D	Gym. J. Hronca Bratislava	0			7			7
	David Martin	2D	Gym. J. Hronca Bratislava	0			7			7
	Janiga Peter		Gym. J. Hronca Bratislava	0			7			7
	Merschitz Mirko		Gym. J. Hronca Bratislava	0			7			7
90.	Cupper Ľubomír	3A	Gym. Krompachy	5						5
91.	Konečný Štefan	2D	Gym. J. Hronca Bratislava	0		2				2