**Korešpondenčný seminár z programovania**
**XXVII. ročník, 2009/10**
**Katedra základov a vyučovania informatiky FMFI UK,**
**Mlynská Dolina, 842 48 Bratislava**

*KSP finančne podporujú: MICROSTEP-MIS spol. s r.o.*
*Agentúra na podporu výskumu a vývoja*

# Vzorové riešenia 2. kola zimnej časti

Milé naše riešiteľky a iní!

Vojaci opravovateľského impéria (čítaj opravovatelia) sa pod hrozbou odobratia vifónu dali do pohybu a priniesli Vám nové vzoráky. Užite si ich!

Ak si dobrý, dajú ti veľa práce. Ak si naozaj dobrý, dokážeš sa jej zbaviť.

KSPáci

## 1. Zvaná túžba

opravovala Dominika
(max. 10 bodov)

Myšlienku riešenia ste mali všetci správne, ale niekomu chýbal popis k riešeniu (dostal 7 bodov) alebo mu chýbal odhad zložitosti (9b).

Tak teda v krátkosti, keďže ste to všetci vedeli. Na zistenie počtu čiernych pasažierov nám stačí zistiť celkový počet pasažierov (všetkých, čo vystúpili) a odrátať počet legálnych pasažierov (teda tých, čo si označili lístok). Keďže je sčítanie a odčítanie komutatívne (môžeme sčitovať a odčitovať v ľubovoľnom poradí), môžeme to robiť priamo v cykle. Časová zložitosť je lineárna a pamäťová konštantná.

### Listing programu:

```
program zvanaTuzba;
var n, listky, cestujuci, cierni, i: integer;

begin
  cierni := 0;
  readln(n);
  for i := 1 to n do begin
    readln(listky, cestujuci);
    cierni := cierni + cestujuci - listky;
  end;
  writeln(cierni);
end.
```

## 2. Zober si ma!

opravoval Mišo
(max. 10 bodov)

Najprv spravíme jedno základne pozorovanie. Pokiaľ začína Xavierov zoznam jedlom $A$, tak toto jedlo môžeme zjesť hneď keď ho prvý krát uvidíme na páse, keďže tým nič nepokazíme (dôkaz prečo je to tak, si skúste spraviť za domácu úlohu).

Poďme si tento postup preložiť do reči programu. Na vstupe máme dva stringy $j$ (zoznam jedál) a $x$ (Xavierov zoznam). Budeme mať 2 indexy, jeden vyjadruje posledné jedlo na zozname, kde sme sa pozreli a ukazuje na prvé jedlo, ktoré sme ešte nezjedli. Týmito indexami budeme prechádzať. Ak sú znaky na daných pozíciach zhodné, Xavier sa naje a môžeme



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

zvýšiť obidva indexy. Ak sú rôzne, Xavier sa nenaje, preto sa posunie index len v prvom stringu. Program vždy skončí, lebo vždy sa posunie aspoň jeden index a niekedy sa dostane nejaký index na koniec stringu. Ak sme prešli celý Xavierov zoznam, tak vypíšeme `ano`. Ak sme ho neprešli celý, to znamená, že sme prešli celý zoznam jedál, lebo aspoň jeden index je na konci stringu. V tomto prípade vypíšeme `nie`.

Časová zložitosť je $O(N)$, kde $N$ je veľkosť celého vstupu (súčet dĺžok oboch stringov). Pamäťová môže byť $O(N)$. Použili sa len tie 2 stringy.

Ale na pamäť sa dá pozrieť aj z iného pohľadu, v tom prípade ju môžeme považovať za konštantnú – $O(1)$. Predstavte si program ako čiernu skrinku. Dostane vstup, niečo šrotí a vypľuje nejaký výstup. Za pamäťovú zložitosť považujeme množstvo pracovnej pamäte, ktorú potrebuje pri tom šrotení. Čo teda môžeme v programe považovať za vstup? Do všetkých premenných, čo sú použité, sa niekedy zapisuje. Ak sa do nich zapisuje iba I/O príkazmi ako `readln()`, `scanf()`, `getchar()`, tak to nie je pracovná pamäť, to je len ten vstup, čo dostane čierna skrinka. V tomto programe sú to tie dva stringy, čierna skrinka si šrotí len s pomocnými premennými.

**Bodovanie:** Hlavnú myšlienku ste mali všetci dobre. Preto uvediem príčiny, za ktoré ste dostali menej bodov. Celkom častá chyba bola, že ste zvyšovali index v Xavierovom zozname vždy, keď bola zhoda. Potom sa ale môže stať, že budete týmto indexom pristupovať mimo poľa a hodí vám to segmentation fault/runtime exception. Preto tam treba dať podmienku, že ak už je Xavier spokojný, tak predčasne ukončíme cyklus. Za túto chybu som strhol pol boda, ale zaokrúhľoval som nahor, preto sa zmena v bodoch prejavila len spolu s ďalšou chybou. Za chýbajúci/zlý odhad zložitosti som strhol 1 bod. Pri pamäťovej som uznal obidva varianty.

### Listing programu:

```
var j, x: string;
    jj, xx: longint;
begin
  readln(j);
  readln(x);
  jj := 1;
  xx := 1;
  while (jj <= length(j)) and (xx <= length(x)) do begin
    if j[jj] = x[xx] then
      xx := xx + 1;
    jj := jj + 1;
  end;
  if xx > length(x) then
    writeln('ano')
  else
    writeln('nie');
end.
```

## 3. Zamakajme si

opravoval Kewo
(max. 10 bodov)

10 bodov dostali tip top riešenia s pamäťovou zložitosťou $O(1)$, časovou $O(N)$. Za pamäť $O(N)$ a čas $O(N)$ max 8 bodov, riešenia s časovou zložitosťou $O(D)$ max 7 bodov, s $O(N^2)$ max 6 bodov, s $O(N \cdot D)$ max 5 bodov, pri $O(N^3)$ som sa zľutoval - max 4 body. Ďalej sa dalo stratiť absenciou zdrojáku či popisu riešenia, chybami v kóde, neavizovanými zmenami na vstupe či výstupe.

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Vstup zväčša musíme prečítať celý (pokiaľ má Imp dostatok síl na prvý cvik) - časovú zložitosť pod $O(N)$ teda nedostaneme. Na rozvoj Impovho svalnatého ja máme $D$ dní. Cvik A je lepší/oplatí sa nám viac ako cvik B vtedy, ak cvikom A získame viac sily ako cvikom B. Niektorí z Vás sa rozhodli simulovať čo vlastne bude Imp cvičiť každý deň (v lepšom prípade ste to zvládli s časovou zložitostou $O(1)$). Takýto prístup po dňoch $(1+1+1+\ldots+1=D)$ vedie k riešeniu s časovou zložitosťou $O(N+D)$. Prístup v zásade nie je zlý, len existuje aj lepší :) Ako má vlastne vyzerať Impove cvičenie? Zatiaľ najlepší cvik praktizuje súvisle niekoľko dní po sebe dovtedy, kým nezosilnie dostatočne, aby dosiaľ najlepší cvik mohol vystriedať za lepší. Ak vieme rýchlejšie (napr. $O(1)$) zistiť, koľko dní po sebe musí cvičiť ten istý cvik, prístup po dňoch vieme nahradiť prístupom po cvikoch $(p_1+p_2+\ldots+p_j=D, j\leq N)$ a teda aj „simulačnú fázu“ stlačiť pod $O(N)$, celkovo teda $O(N)$. Počet dní od teraz po lepší cvik vieme zistiť ako hornú celú časť z $\frac{F_n-F}{P}$ (kde $F_n$ je sila potrebná na nový lepši cvik, $F$ aktuálna sila a $P$ príspevok aktuálneho cviku). Takto postupom - pamätám si zatiaľ najlepší cvik, načítavam ďalšie, ak nájdem lepší cvik, odcvičím si potrebný počet dní, aby som mohol realizovať lepší a tak ďalej až do konca dní - získavame riešenie s časovou zložitosťou $O(N)$ a pamäťovou $O(1)$ (pamätáme si zatiaľ najlepší cvik, najbližší lepší, koľko máme sily, koľko dní nám zostáva). To je asi tak všetko :)

## Listing programu:

```
program zamakajmeSi;
var s, n, d, sila, prir, i, k, ns, np, cv: integer;

begin
  read(s);  read(n);  readln(d);
  sila := s;  prir := 0;
  for i := 1 to n do begin
    read(ns);  readln(np);            { nova sila / prirastok }
    if (i = 1) then begin
      if (ns > sila) then begin       { nemame silu na pociatocne cvicenie }
        writeln('Imp nema dost sil na ziaden cvik :(');  d := 0;  break;
      end;
      prir := np;  k := i;            { update zatial najlepsieho cviku }
    end
    else begin
      if (np > prir) then begin       { ak sme nacitali lepsi cvik }
        if (ns > sila) then begin     { a musime nan este cvicit }
          cv := (ns - sila) div prir;
          if ((ns - sila) mod prir > 0) then inc(cv);
          if (cv > d) then cv := d; { nemozme cvicit viac ako sa da }
          writeln(cv, ' dni cvicil ', k, '. cvik');
          sila := sila + cv*prir;  d := d - cv;
          if (d = 0) then break;      { niet uz viac kedy cvicit }
        end;
        prir := np;  k := i;          { update zatial najlepsieho cviku }
      end;
    end;
  end;
  if (d > 0) and (i = n) then begin
writeln(d, ' dni cvicil ', k, '. cvik');
sila:=sila + d*prir;
  end;
  writeln(sila);
end.
```

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

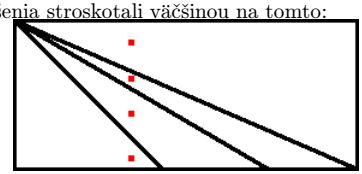
opravovali Mio a Feki
(max. 15 bodov)

# 4. Zemčove mosty

Riešenia, ktoré nám prišli, sa dajú rozdeliť do troch kategórií – logaritmické, lineárne a nefunkčné. Najskôr ako sme bodovali.

**Bodovanie** Riešenia s časovou zložitosťou $O(N+K\cdot\log N)$ a pamäťovou $O(N)$ mohli dostať najviac 15 bodov. Za riešenia s časovou zložitosťou $O(K\cdot N)$ a pamäťovou $O(N)$ bolo do 9 bodov. Riešenia, ktoré neboli korektné, nedostali viac ako 4 body. Za horšiu pamäť sme strhávali bod. Ďalšie tresty boli za slabý popis – chceli sme tam mať popis spôsobu, akým zisťujete, či je bod napravo alebo naľavo, popis binárneho vyhľadávania a časové a pamäťové zložitosti. Ak popis alebo kód chýbal, strhli sme veľa bodov. Toľko k bodovaniu, teraz k riešeniam.

**Riešenia** Nekorektné riešenia stroskotali väčšinou na tomto:



Predpokladali ste, že ak zoberiete obdĺžnik, ktorého je nejaká úsečka uhlopriečkou, tak sa v tomto obdĺžniku už nemôže nachádzať iná úsečka. Chyba! Môže. A nemusia tie úsečky vychádzať z jedného bodu, stačí aby začínali blízko seba. Prejdime ku korektným riešeniam. V oboch typoch riešení bolo treba zistiť, či leží bod napravo alebo naľavo od úsečky. Opäť ste to riešili dvojako. Prvý spôsob bol vyjadriť si úsečku pomocou rovnice, dosadiť Y-ovú súradnicu bodu a porovnať X-ové súradnice bodu. Druhý spôsob je použiť vektorový súčin. Keďže druhý spôsob sa vyskytoval menej, rozhodli sme sa, že vo vzorovom riešení využijeme práve ten.

Čo je to vektorový súčin? Začneme radšej tým, čo je to vektor. Vektor je orientovaná úsečka. Zapisujeme ho pomocou usporiadanej $n$-tice. Napríklad vektor $(1,2)$ znamená, že máme orientovanú úsečku z bodu $[0,0]$ do bodu $[1,2]$. Sčítanie vektorov je po súradniciach (čiže $(1,2)+(3,0)=(4,2)$). Vektor vieme vynásobiť skalárom ($2\times(1,2)=(2,4)$).

Dva vektory v 3D priestore vieme násobiť vektorovým súčinom. Dostaneme vektor, ktorý je kolmý na obidva vektory a má veľkosť rovnú obsahu rovnobežníka, ktorého strany určujú vstupné vektory. Smer vektora sa určuje podľa pravidla pravej ruky. Toľko k vektorovému súčinu, kto chce vedieť viac:

`http://sk.wikipedia.org/wiki/Vektorový_súčin`

Práve to ako sa nám určí smer vieme využiť. Vzhľadom na to, že máme úsečky v 2D, tak položíme ich 3. súradnicu 0. A teda jediná nenulová súradnica výsledného vektora bude tá 3. (keby bola iná nenulová, tak výsledný vektor nie je kolmý na pôvodné 2).

Podľa znamienka 3. súradnice vieme zistiť, ktorý vektor je napravo od druhého. Takže ak máme úsečku $AB$ a bod $C$, tak stačí vynásobiť napríklad vektory $\boldsymbol{AC}\times\boldsymbol{BC}$[1] a vieme, že ak je výsledok kladný, tak bod je napravo; ak je 0, tak na úsečke a ak záporný, tak je naľavo. Dobre, takže vieme zistiť, kde sa nachádza bod, pomocou takéhoto vzorca: $c=a_1b_2-a_2b_1$.

Lineárne riešenia spravili to, že postupne prechádzali úsečky a zisťovali, či už je bod konečne naľavo, vtedy skončili a vypísali výsledok.

Využime to, že úsečky sú utriedené. Keď chceme niečo nájsť v utriedenom poli, používame binárne vyhľadávanie. Algoritmus binárneho vyhľadávania vyzerá vo všeobecnosti takto: zoberieme stredný prvok a pozrieme sa, či je väčší ako prvok, ktorý hľadáme; ak áno, to isté spravíme v ľavej polovici poľa; ak je rovnaký – našli som ho; ináč to isté spravíme v

[1] Môžme vynásobiť aj iné vektory, napríklad $\boldsymbol{AB}$ a $\boldsymbol{AC}$ alebo $\boldsymbol{AB}$ a $\boldsymbol{BC}$, len si treba vždy uvedomiť, pri akom znamienku je bod napravo a pri akom naľavo.



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

pravej polovici poľa, až kým nenájdeme miesto, kde by mal daný prvok byť. To, či je bod väčší ako úsečka, znamená v tomto prípade, že je od úsečky napravo, menší je naľavo. To ako majú byť ošetrené body na úsečke zadanie neurčovalo, ani sme to nebrali pri hodnotení do úvahy.

Pretože sa počet prvkov, medzi ktorými hľadáme, zmenšuje v každom kroku na polovicu, časová zložitosť binárneho vyhľadávania je $O(\log N)$.

## Listing programu:

```
#include <iostream>
#include <vector>

#define VYSKA 1000
#define MAXW 1000000

using namespace std;

int n,k;
vector<pair<int,int> > usecky;

bool compare(pair<int, int> bod, int usecka)
{
    //zaciatok obdlznika ktoreho je usecka uhlopriecko
    //aby sme vedeli usecku posunut tak, aby zacinala na x = 0
    int l = min(usecky[usecka].first,usecky[usecka].second);

    //posunutie usecky tak aby zacinala na x = 0
    int u1 = usecky[usecka].first - l;
    int u2 = usecky[usecka].second - l;

    //posunutie bodu, aby bol vzhladom na usecku na rovnakom mieste
    int bx = bod.first - l;
    int by = bod.second;

    //pocitanie vektorov AC a BC
    int v1x = bx - u1;
    int v1y = by;
    int v2x = bx - u2;
    int v2y = by - VYSKA;

    //zratam vektorovy sucin
    if((v1x*v2y - v1y*v2x)>0) return true; //nalavo
    else return false; // napravo
}

//bin-search
int search(pair<int,int> bod, int start, int end)
{
    while (start < end)
    {
        //postavim sa do stredu rozsahu
        int u = (end + start) / 2;
        //zistim ci som mensi alebo vacsi a podla toho zmenim rozsah na lavu pripadne
pravu polovicu
        if (compare(bod, u)) end = u;
        else start = u+1;
    }
    return start;
}
```

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
int main()
{
    cin >> n >> k;
    usecky.resize(n+2);

    //pridanie zaciatku a konca mosta
    usecky[0].first = usecky[0].second = 0;
    usecky[n+1].first = usecky[n+1].second = MAXW;

    //nacitanie useciek
    for(int i = 1; i<=n; i++)
    {
        cin >> usecky[i].first >> usecky[i].second;
    }
    //nacitanie a spracovanie bodov
    pair<int,int> bod;
    for(int i = 0; i<k; i++)
    {
        cin >> bod.first >> bod.second;
        cout << search(bod, 0, n+1) << endl;
    }
    return 0;
}
```

opravoval Maják
(max. 15 bodov)

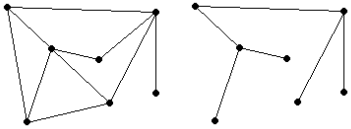
## 5. Zotni tie káble!

Drvivej väčšine z Vás, ktorí riešili tento príklad, sa podarilo spraviť nasledovné dôležité pozorovanie: ak nájdeme najlacnejšiu kostru zadaného grafu, tak sme vyhrali. Na začiatok trochu terminológie.

**Najlacnejšia kostra:** Graf $G$, ktorý pozostáva z vrcholov $V$ a hrán $E$, ktoré tieto vrcholy spájajú, budeme označovať ako $G = (V, E)$.

Strom je špeciálny typ grafu. Aby sa graf mohol nazvať stromom, tak pre jeho kaŽdé dva vrcholy musí platiť, že z jedného do druhého existuje len jedna cesta.

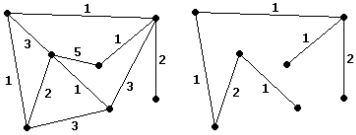
Na tamtom obrázku je vľavo nakreslený graf a vpravo jedna z jeho možných kostier.



Kostra spojitého grafu $G = (V, E)$ je taký podgraf $G'$ grafu $G$, ktorý obsahuje všetky vrcholy z $G$ a len tie hrany z $E$, aby $G'$ bol strom.

Minimálna, alebo aj najlacnejšia kostra grafu je taká kostra grafu, ktorá má zo všetkých kostier najmenšiu cenu.

Na nasledujúcom obrázku je pravo znázornená minimálna kostra ku grafu vľavo.




**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

A prečo nás vlastne tak veľmi zaujíma najlacnejšia kostra? Dôvod je jednoduchý. Ak ju nájdeme, tak hrany ňou obsiahnuté spájajú navzájom všetky zariadenia, ktoré boli spojené v pôvodonom grafe. Ostatné hrany sú preto vhodné na vytrhnutie. Kedže si ponechávame najlacnejšiu kostru, tak súčet nekostrových hrán (to sú tie vytrhnuté) bude najväčší možný. A to je presne to, o čo nám v tejto úlohe ide.

Pre hľadanie najlacnejšej kostry existuje viacero algoritmov s podobnou zložitosťou. Asi najznámejšie sú Primov a Kruskalov. My si ukážeme ten prvý z nich. Dočasne budeme predpokladať, že náš graf pozostáva len z jedného komponentu.

V priebehu tohto algoritmu budeme mať všetky vrcholy rozdelené na dve množiny – tie, ktoré už máme v kostre, budú $S$ a tie, čo ešte nie, budú $V \setminus S$. Z prvej, kostrovej časti vedú určité hrany do druhej, nekostrovej časti. Ukážeme si, že najlacnejšia z nich bude určite ležať v niektorej minimálnej kostre.

Sporom, predpokladajme, že v minimálnej kostre neleží. Označme si ju ako $e$ a vrcholy, ktoré spája, ako $v$ a $w$. Potom musí v minimálnej kostre existovať iná cesta, ktorá začína vo $v$, končí vo $w$ a prechádza aspoň jednou hranou z aktuálneho rozhrania $S$ a $V \setminus S$. Označme si túto hranu ako $f$.

Keď do tejto cesty pridáme hranu $e$, tak dostaneme cyklus $C$. Po odstránení jednej hrany z cyklu $C$ dostaneme opäť kostru grafu, pretože každá cesta, ktorá v pôvodnej kostre spájala vrcholy pomocou hrany $f$, ich teraz spája cez zvyšné hrany tohto cyklu $C$.

Platí $|f| \geq |e|$, a preto keď z cyklu $C$ odstránime $f$, dostaneme kostru, ktorá je menšia alebo rovná ako tá, o ktorej sme uvažovali ako o minimálnej. To je hľadaný spor, a preto nemôže platiť predpoklad, čo znamená, že hrana $e$ do minimálnej kostry patrí.

Počas behu algoritmu si preto potrebujeme udržiavať množinu hrán vedúcich z $S$ do $V \setminus S$. Takúto podmienku by ale bolo pri zmenách množín ťažké dodržiavať, preto sa uspokojíme s hranami vedúcimi z $S$ (a vchádzajúcimi nevedno kam). Vtedy stačí do množiny hrany pridávať, lebo $S$ sa len zväčšuje.

Do tejto množiny potrebujeme vedieť efektívne pridávať prvky a vyberať minimum. Preto použijeme haldu, do ktorej vieme vkladať prvky a vyberať minimum v čase $O(logN)$, kde $N$ je počet v nej sa nachádzajúcich prvkov.

Jadro algoritmu bude vyzerať takto:

1. Vyberám z haldy obsahujúcej hrany vychádzajúce z $S$, pokiaľ nenatrafím na hranu vedúcu do $V \setminus S$.
2. Pridám ju do minimálnej kostry. Nech vrchol, do ktorého vedie mimo $S$, je $v$. Potom do $S$ pridám $v$ a do haldy všetky hrany z neho vedúce mimo $S$.

Na začiatku bude $S$ množina obsahujúca jediný vrchol a v halde budú len hrany z neho vychádzajúce. Algoritmus ukončíme, keď budeme mať v kostre všetky vrcholy, teda $S = V$.

**Modifikácie pre náš problém:** Zatiaľ sme ignorovali skutočnosť, že náš graf môže pozostávať z viacerých komponentov. Ak by tomu tak bolo, tak zjavne Primov algoritmus nájde len kostru jedného komponentu, pretože z neho do iného nevedú žiadne hrany (inak by to neboli rôzne komponenty). Preto ho môžeme skúsiť spustiť zo všetkých jednotlivých vrcholov, ak už neboli niekedy zaradené do kostry.

**Odhad zložitosti:** Každú hranu pridáme do haldy maximálne raz, teda najviac ich tam môže byť spolu $|E|$. Zároveň každú vloženú hranu aj vyberieme. Do haldy vkladáme aj vyberáme prvky v logaritmickom čase od jej veľkosti, a preto celková zložitosť bude $O(|E| \log |E|)$. Môžme predpokladať, že graf nebude mať viac ako $|V|^k$ hrán. Potom z vlastnosti logaritmu platí $O(|E| \log(|V|^k)) = O(|E| \cdot k \log |V|) = O(|E| \log |V|)$.

Pamäťová zložitosť bude $O(|V| + |E|)$, pretože si musíme pamätať všetky hrany aj všetky vrcholy.

**Bodovanie** Za algoritmus s časovou zložitosťou $O(|E| \log |V|)$, popisom, funkčným kódom a dobrými odhadmi zložitostí ste mohli získať plný počet bodov – teda 15. Ak váš algoritmus



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

bežal v čase $O(|V|^2)$, tak ste mohli dostať do 13 bodov. Za $O(|E|\cdot|V|)$ som udeľoval 10 bodov a za $O(|V|^3)$ len 8.

Ak ste sa pri odhade rozumne odvolali na poznámku o hustote zo zadania, tak ste mohli získať aj viac, ako podľa tejto stupnice.

Samozrejme, za chýbajúce časti riešenia šli ďalšie body dole.

## Listing programu:

```
const MAX_VRCHOLOV=100;
      MAX_HRAN=20000;

type Tkabel = record
  odkial,kam,cena,poradie: longint;
end;

var hrany:array [1..MAX_VRCHOLOV, 1..MAX_HRAN] of Tkabel;
  pocet_hran: array[1..MAX_VRCHOLOV] of longint;
  v_kostre, vytrhnute: array[1..MAX_VRCHOLOV] of boolean;
  N,M: longint;
  stres: longint;
  i,j,a,b,cena: longint;
  min_hrana: Tkabel;
  Halda: array[1..MAX_VRCHOLOV*MAX_VRCHOLOV] of Tkabel;
  prvkov_v_halde: longint;


procedure vymen(var a,b : Tkabel);
var c : Tkabel;
begin c:=a; a:=b; b:=c; end;

procedure bubliHore;
var kde : longint; { kde je prvok, ktory bubleme }
begin
  kde := prvkov_v_halde;
  while true do begin
    if (kde = 1) then break; { uz sme uplne hore }
    if (Halda[kde div 2].cena < Halda[kde].cena) then break; { dalej sa neda }
    vymen( Halda[kde div 2], Halda[kde] ); { vymenime ho s otcom }
    kde := kde div 2; { a ideme ho bublat dalej }
  end;
end;

procedure bubliDole;
var kde, kam : longint; { kde sme, s kym ho vymenit }
begin
  kde := 1;
  while true do begin
    kam := kde;
    if (2*kde  <=prvkov_v_halde) then
        if (Halda[2*kde].cena  <Halda[kam].cena) then kam:=2*kde;
    if (2*kde+1<=prvkov_v_halde) then
        if (Halda[2*kde+1].cena<Halda[kam].cena) then kam:=2*kde+1;
    if (kde = kam) then break; { hlbsie to nejde }
    vymen( Halda[kde], Halda[kam] ); { vymenime ho s lepsim synom }
    kde := kam; { a ideme ho bublat dalej }
  end;
end;

procedure vlozDohaldy(co : Tkabel);
```



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
begin
  inc(prvkov_v_halde);
  Halda[prvkov_v_halde]:=co;
  bubliHore;
end;

function vyber_z_haldy() : Tkabel;
begin
  vyber_z_haldy:=Halda[1];
  vymen( Halda[1], Halda[prvkov_v_halde] );
  dec(prvkov_v_halde);
  bubliDole;
end;

begin
  stres:=0;
  prvkov_v_halde:=0;

  readln(N,M);
  for i:=1 to N do begin
    for j:=i to N do begin
      hrany[i,j].cena:=-1;    { vynulujem maticu susednosti }
      hrany[i,j].poradie:=-1; { -1 znamena neexistenciu hrany }
    end;
    pocet_hran[i]:=0; { zo ziadneho vrchola este nevedie hrana }
    v_kostre[i]:=false; { a ani ziaden vrchol este nie je v kostre }
  end;
  for i:=1 to M do begin
    vytrhnute[i]:=true; { na zaciatku nie je ziadna hrana v kostre }
    readln(a, b, cena);
    inc(stres, cena);
    inc(pocet_hran[a]);
    hrany[a,pocet_hran[a]].cena:=cena;
    hrany[a,pocet_hran[a]].poradie:=i;
    hrany[a,pocet_hran[a]].odkial:=a;
    hrany[a,pocet_hran[a]].kam:=b;
    inc(pocet_hran[b]);
    hrany[b,pocet_hran[b]].cena:=cena;
    hrany[b,pocet_hran[b]].poradie:=i;
    hrany[b,pocet_hran[b]].odkial:=b;
    hrany[b,pocet_hran[b]].kam:=a;
  end;

  for i:=1 to N do if not v_kostre[i] then begin { kostru skusam robit postupne z
kazdeho vrchola }
    v_kostre[i]:=true;
    for j:=1 to pocet_hran[i] do if not v_kostre[hrany[i,j].kam] then begin
      vlozDohaldy(hrany[i,j]); { do haldy vlozime hrany vychadzajuce zo zaciatocneho
vrchola }
    end;
    while prvkov_v_halde > 0 do begin
      min_hrana.cena:=-1;
      repeat { vyberiem najlacnejsiu hranu veducu do noveho vrchola, ak taka este je }
        min_hrana:=vyber_z_haldy;
      until (prvkov_v_halde = 0) or (not v_kostre[min_hrana.kam]);
      if (min_hrana.cena > -1) and (not v_kostre[min_hrana.kam]) then begin
        vytrhnute[min_hrana.poradie]:=false; { tato hrana je v kostre }
        dec(stres, min_hrana.cena); { preto nebude vytrhnuta a neznizi mi uvoleneny
stres}
        v_kostre[min_hrana.kam]:=true;
```



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
        for j:=1 to pocet_hran[min_hrana.kam] do if not
v_kostre[hrany[min_hrana.kam,j].kam] then begin
          vlozDohaldy(hrany[min_hrana.kam,j]); { do haldy vlozime hrany
vychadzajuce zo pridaneho vrchola }
        end;
      end;
    end;
  end;

  writeln(stres);
  for i:=1 to M do begin
    if (vytrhnute[i]) then writeln('Trhaj Dominika, trhaj!')
    else writeln('Ani omylom!');
  end;
end.
```

opravoval Stano
(max. 20 bodov)

## 6. Objednávka veže

Napriek tomu, že tento príklad vyzeral odradzujúco, nebol až taký zložitý a našlo sa pár odvážlivcov, ktorí sa odhodlali ho vyriešiť, podaktorí dokonca aj inak ako backtrackom.

Vzorové riešenie funguje pomocou dynamického programovania v časovej zložitosti $O(NK)$. Skúsme si predstaviť, že už máme nejak optimálne vyriešené veže, ktoré používajú prvý až $i$-ty valec. Čo s takouto vežou môžeme spraviť? Skúsime pridať valec na jednu vežu, na druhú vežu, a skúsime aj nechať pôvodné riešenie. Aby sme sa vedeli rozhodovať, ktoré riešenie si nechať, môžeme si pamätať rozdiel výšok veží, a vždy, keď máme rovnaký rozdiel výšok, oplatí sa nám nechať si riešenie, kde je jedna veža čo najvyššia – budeme si pamätať tú väčšiu.

Keď dostaneme na vstupe ďalší valec výšky $x$, prejdeme všetky možné rozdiely výšok, ktoré sme už predtým vedeli postaviť, a pozrieme sa, či nevieme niečo zlepšiť pridaním valca na menšiu (výšky $V_m$) alebo väčšiu vežu (výšky $V_v$).

Pridaním valca na väčšiu vežu sme dostali veže s rozdielom výšok $V_v + x - V_m$, ak sme zatiaľ s takýmto rozdielom výšok postavili iba nižšie veže, tak sa nám táto varianta oplatí viac.

Pridaním valca na menšiu vežu sa nám možno vymenia poradia výšky veží, tak ich prípadne otočíme, a postupujeme rovnako ako v minulom prípade – zase sa pozrieme na políčko s rozdielom výšok veží, a ak tam máme horšie riešenie, zapíšeme si aktuálne.

Aby sme sa vyhli tomu že použijeme jeden valec viac krát, musíme si pamätať 2 riadky, jeden predchádzajúci a jeden, kde už môžeme použiť aj $i+1$-vý valec, čím dostávame pamäťovú zložitosť $O(K)$.

Riešenie, ak existuje, sa potom nachádza na políčku s rozdielom výšok veží 0.

### Listing programu:

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
    int N,K,v,m,vyska;
    cin >> N >> K;
    vector<int> stare(K+1,-1),nove(K+1,-1);
    nove[0]=0;
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    for (int i=0;i<N;i++){
        cin >> vyska;
        for (int j=0;j<=K;j++) stare[j]=nove[j];
        for (int j=0;j<=K;j++){
            if (stare[j]==-1) continue;//ak sa nedali postavit veze s rozdielom j
            if (stare[j+vyska]<stare[j]+vyska) //skusime dat valec na vyssiu vezu
                nove[j+vyska]=stare[j]+vyska;
            m=stare[j]-j+vyska;//skusame postavit valec na mensiu vezu
            v=stare[j];
            if (v<m) {int t=v; v=m; m=t;};
            if (stare[v-m]<v) nove[v-m]=v;
        }
    }
    cout << nove[0] << endl;
    return 0;
}
```

opravoval U$Ama
(max. 20 bodov)

## 7. Odporúčací prístroj

Nazdar. Táto úloha nebola až tak ťažká. Poväčšinou ste ju zvládli, ale niekedy ste to riešili zbytočne komplikovane. Poďme sa pozrieť, ako som bodoval: riešenie, ktoré splní požiadavku v čase $O(N)$ (t.j. triviálnym spôsobom) dostalo najviac 12 bodov. Riešenie, ktoré plní požiadavky v čase $O(\log N)$, dostalo plný počet plus nejakú dávku hundrania v závislosti od toho, ako ste si to skomplikovali. Záporné odmeny som dával za slabý popis a za to, že ste vyberali najdlhší interval, čo nemusí byť vždy správne.

Poďme na to. Neudržujme si voľné záchody po jednom, ale v intervaloch. Pre interval si pamätáme miesta, kde začína a končí. Ja preferujem polootvorené intervaly (t.j. začiatok je tam, kde je a koniec ukazuje o 1 miesto za koniec intervalu – skupinu záchodov 1, 2, 3 si pamätám ako [1,4)). Ich výhodou je napr. to, že prázdny interval si pamätám ako napr. [4,4) a nie nejakým obludným spôsobom.

Postup, ako robiť operácie, sa ukazuje vcelku sám. Keď niekto príde, zoberieme interval, ktorý poskytuje najlepšiu vzdialenosť (pozor, nie najväčší interval)[2], keď je ich viac, tak ten, čo je najviac vzadu. Ak je interval vyhovujúci, tak určíme miesto pre človeka. Starý interval zrušíme a vyrobíme 2 nové.

Postup pre odchod človeka je obdobný. Zoberieme intervaly okolo obsadeného pisoára, zrušíme ich a vyrobíme 1 nový, ktorý je ich spojením (+ obsahuje ten záchod medzi nimi).

Ako to ale robiť rýchlo? To, ktoré intervaly obklopujú daný pisoár, si vieme pamätať v jednom poli s dĺžkou $N$ (pre každý pisoár si pamätáme, ktorý interval tam začína a končí, ak tam nič nezačína/nekončí, tak je tá hodnota pre nás nepodstatná). Ešte treba vyberať najvýhodnejší interval. Na to vie dobre poslúžiť napríklad halda[3] (podporuje výber maxima, vkladanie a vymazávanie v logaritmickom čase od počtu prvkov). Toto riešenie má ale niekoľko problémov. Hlavne si treba pre každý interval pamätať, kde v halde sa nachádza (prípadne nemať haldu v poli, ale robenú pomocou ukazovateľov). Proste nič moc.

Poďme si skúsiť pomôcť knižnicou STL. Niektorí sa na to možno budú krivo pozerať, ale STL je úplne legitímna vec, pokiaľ si uvedomujeme, čo robí a ako to robí (alebo aspoň ako rýchlo). To, čo budeme potrebovať, je set alias množina. V podstate je to čierna krabička, ktorá umožňuje vkladať, zmazávať a vyhľadávať prvky (aj štýlom: povedz mi prvý väčší

[2] Napríklad, keď máme intervaly [1,5) a [5,8), tak obidva poskytujú vzdialenosť 1.

[3] http://en.wikipedia.org/wiki/Heap_(data_structure)



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

alebo rovný ako toto). A ďalej umožňuje nájsť najmenší a najväčší prvok. A to všetko v čase $O(\log N)$. Reálne je implementovaná ako červeno-čierny vyhľadávací strom[4].

Použijeme 2 sety. Jeden na triedenie podľa výhodnosti (napíšeme si na to triediacu funkciu) a druhý bude triediť podľa miesta, kde začínajú intervaly (opäť si na to napíšeme triediacu funkciu). Postup potom použijeme podľa popisu vyššie. A keďže každá operácia robí konštantný počet vecí, ktoré trvajú $O(\log N)$, tak celkový čas operácie je $O(\log N)$. Mohli by sme si položiť otázku, prečo nevyužiť iba jeden set a na to, ktoré intervaly nasledujú za daným pisoárom použiť obyčajné pole. Odpoveď je jednoduchá. Časovú zložitosť nám to nepokazí, ale pri písaní programu je menšia šanca sa pomýliť.

Ešte jedna poznámka na záver. Na pamätanie si priradenia poradové číslo → záchod je najvýhodnejšie použiť asociatívne pole. V tomto prípade map (opäť je to vnútorne červeno-čierny strom a operácie trvajú $O(\log N)$). Vďaka tomu si môžeme pamätať len ľudí, ktorých máme pri záchodoch a nepokaziť si ani pamäťovú ani časovú zložitosť.

Ešte dodám, že pamäte budeme potrebovať $O(N)$.

## Listing programu:

```
#include <cstdio>
#include <set>
#include <map>

using namespace std;

int N;

class Interval
{
public:
    int begin, end;                    //begin - zaciatok, end - 1 miesto za koncom

    Interval(int b, int e)
    {
        begin = b; end = e;
    }

    Interval split(int &place)         //rozdelime interval na 2 kusy, 1 kus nechame tu,
2 vratime
    {
        if(end==N)          //special case
            place = N-1;
        else if(begin==0)          //a este jeden
            place = 0;
        else
            place = (begin+end)/2;
        Interval ret = Interval(place+1, end);
        end = place;
        return ret;
    }

    int good()              //nevracia velkosť, ale výhodnosť
    {
        if(begin == 0) return end;
        if(end == N) return N-begin;
        return (end-begin+1)/2;
    }
```

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


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
};

class OrderByGood{
public:
    bool operator()(Interval a, Interval b)
    {
        if(a.good() > b.good()) return true;
        if(a.good() < b.good()) return false;
        return a.begin > b.begin;          //ked su rovne berieme ten dalej
    }
};

class OrderByBegin{
public:
    bool operator()(Interval a, Interval b)
    {
        return a.begin < b.begin;
    }
};


int main()
{
    set<Interval, OrderByGood> byGood;
    set<Interval, OrderByBegin> byBegin;
    scanf("pocet %d", &N);
    Interval base = Interval(0, N);
    byGood.insert(base);
    byBegin.insert(base);
    map<int, int> results;
    int cpoz = 0;
    while(1){
        char buf[20];
        scanf("%s", buf);
        if(buf[0]=='v'){     //vstupil
            cpoz++;
            Interval i = (*byGood.begin());          //vyberieme najvacsi interval
            if(i.good()<2){                          //ak je malo miesta, tak lutujeme
                printf("lutujeme\n");
                continue;
            }
            int place;
            byGood.erase(i);                    //vymazeme interval
            byBegin.erase(i);
            Interval j = i.split(place);      //rozdelime ho

            byGood.insert(i); byGood.insert(j);
            byBegin.insert(i); byBegin.insert(j);      //vlozime ho
            printf("vase miesto: %d\n", place+1);      //vypiseme miesto
            results[cpoz]=place;      //ulozime, kde clovek skoncil
        }
        if(buf[0]=='o'){     //odisiel
            int x;
            scanf("%d", &x);
            int place = results[x];
            Interval del = Interval(place, place);
            set<Interval, OrderByBegin>::iterator it = byBegin.upper_bound(del);
            Interval i = *it;              //zoberieme interval za miestom
            --it;
            Interval j = *it;              //zoberieme interval pred miestom
```



*Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09*

```
            byGood.erase(i); byGood.erase(j);                    //vymazeme ich
            byBegin.erase(i); byBegin.erase(j);
            i.begin = j.begin;                    //spojime ich
            byGood.insert(i); byBegin.insert(i);     //a vlozime nazad
        }
    }
}
```

opravovalo sa samo, vzorák U$Ama

## 8. Totálne šialenstvo

(max. 25 bodov)

Tento príklad nedopadol úplne najlepšie. Poďme sa teda pozrieť, ako ho vyriešiť tak, aby nám testovač dal 25 bodov.

**Úvodné pozorovanie** Skúsme najprv napísať priamočiare riešenie. Vyskúšame hrubou silou všetky podmnožiny čísel zo zadania. Tu narazíme na prvý problém. Treba overiť, či súčin vybraných čísel dáva druhú mocninu. Priame násobenie je nepohodlné, keďže nám môžu vzniknúť pomerne veľké čísla. Ale už samotné zadanie ponúka „dobrý hint“. Treba rozložiť čísla na prvočísla. Teraz si ukážme odpoveď na niekoľko základných otázok života a smrti. Máme dva prvočíselné rozklady $x = 2^{a_1} \cdot 3^{b_1} \cdot 5^{c_1} \ldots$, $y = 2^{a_2} \cdot 3^{b_2} \cdot 5^{c_2} \ldots$ Ako bude vyzerať prvočíselný rozklad čísla $x \cdot y$? No bude to: $x \cdot y = 2^{a_1+a_2} \cdot 3^{b_1+b_2} \cdot 5^{c_1+c_2} \ldots$ A ako zo znalosti prvočíselného rozkladu rozhodneme, či je dané číslo druhou mocninou nejakého celého čísla? No predsa všetky exponenty musia byť párne, teda napríklad $2^4 \cdot 17^2$ druhá mocnina je, ale $2^4 \cdot 19^2 \cdot 47^3$ nie je.

Z tohoto pozorovania dostávame už vcelku jasné riešenie. Najprv si vygenerujeme všetky prvočísla menšie ako požadovaná hranica. Potom si rozložíme každé číslo na vstupe a zapamätáme si príslušné exponenty. A nakoniec prejdeme všetky podmnožiny. Pri každej podmnožine sčítame čísla v exponentoch a skontrolujeme, či sme dostali iba párne čísla. Ak áno, vyhrali sme. Návod, ako vyskúšať efektívne všetky podmnožiny, hľadajte vo vzoráku k 10. príkladu. Výsledný čas bude $O(2^N \pi(K))$, kde $\pi(K)$ je počet prvočísel menších alebo rovných ako $K$. Pamäťové nároky budú $O(N\pi(K))$. Toto riešenie dostane takmer 10 bodov.

**Lepšie riešenie** Ako naprogramovať niečo lepšie? Krutou pravdou je, že myšlienka lepšieho riešenia obsahuje jeden veľmi neintuitívny skok. Označme si to, či sme $i$-te číslo vybrali, ako $x_i$; čiže ak sme ho nevybrali, tak $x_i = 0$ a ak sme ho vybrali, tak $x_i = 1$. Ak toto spravíme, tak sa to už začína opäť sypať „samo“. Naše požiadavky na súčet exponentov vo vybraných číslach môžeme formulovať ako rovnice. Napríklad pre exponenty pri prvom prvočísle máme rovnicu (alebo skôr kongruenciu): $x_1a_1 + x_2a_2 + \ldots + x_na_n \equiv 0 \pmod 2$. Tá hovorí, že súčet exponentov vo vybraných číslach pri prvom prvočísle musí byť párny (teda po delení 2 dávať zvyšok 0). Takto zostavíme sústavu lineárnych kongruencií. Navyše vidíme, že si nepotrebujeme pamätať pri rozklade konkrétny exponent, ale len 0 alebo 1 podľa toho, či bol párny alebo nepárny. Keď nájdeme nenulové riešenie tejto sústavy, tak máme jasne popísanú potrebnú podmnožinu.

V našom príklade máme situáciu o trochu ľahšiu, keďže na ľavej strane rovníc máme neznáme a na pravej strane nulu. To nám jednak hovorí, že sústava má vždy aspoň 1 riešenie (samé nuly) a zároveň pokiaľ je neznámych viac ako rovníc, tak vieme nájsť aj nejaké nenulové riešenia. Na druhej strane trochu netypické je, že počítame modulo 2, ale to nám nerobí skoro žiadny problém. Za domácu úlohu si dokážte, že ak $a \equiv b \pmod 2$ a $c \equiv d \pmod 2$, tak $a + c \equiv b + d \pmod 2$. A ešte si dokážte, že $a + b \equiv a - b \pmod 2$. Navyše modulo 2 máme len 2 rôzne čísla, a to 0 a 1. Z toho vyplýva, že nepotrebujeme nikdy nič deliť, keď budeme v rovnici chcieť niekde dostať 1.

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

**Gaussova eliminačná metóda** Majme sústavu rovníc. V prvom rade si ju zapíšme do matice. Majme sústavu

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \equiv 0 \pmod 2$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \equiv 0 \pmod 2$$

$$\ldots$$

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \equiv 0 \pmod 2$$

Potom matica pre túto sústavu vyzerá nasledovne:

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

Naším cieľom ju bude upraviť na kultúrnejší tvar, tzv. trojuholníkovú redukovanú maticu (ďalej len TRM). Označme najľavejší nenulový prvok každého riadka ako jeho vedúci prvok. TRM musí spĺňať nasledovné podmienky:

1. Vedúci prvok každého riadka je 1.
2. Ak nejaký stĺpec obsahuje vedúci prvok nejakého riadka, tak jeho ostatné prvky sú nulové.
3. Vedúce prvky sú usporiadané zľava doprava.
4. Nulové riadky sú pod nenulovými riadkami.

Teraz si ukážeme, ako pomocou ekvivalentných úprav upraviť maticu na TRM. Využijeme nasledovné úpravy (keďže rátame modulo 2, tak nám stačia tieto 2):

- Výmena 2 riadkov.
- Pričítanie jedného riadka k druhému.

Za domácu úlohu si dokážte, že sú to ekvivalentné úpravy. Postup úpravy matice bude nasledovný. Nasledujúci postup opakuj pre $i$ od 1 do $m$.

1. Z riadkov $i$, $i+1$, ..., $m$ vyber riadok, ktorý má vedúci prvok najviac vľavo, pokiaľ je ich viac, vyber hociktorý z nich, nech je to riadok $j$ a vedúci stĺpec označme $k$.
2. Vymeň riadky $i$ a $j$.
3. Potom ku všetkým riadkom okrem $i$, ktoré v stĺpci $k$ majú 1, pripočítaj riadok $i$.

Z popisu postupu vyplýva, že čas úpravy bude $O(m^2n)$. Ešte treba ukázať, že úpravy boli korektné a výsledná matica je TRM. Použili sme iba výmenu a sčítanie 2 riadkov, takže sme boli korektní. Podmienka 1 z TRM vyplýva z toho, že počítame modulo 2. Podmienka 2 je splnená vďaka kroku 3, ktorý vynuluje patričný stĺpec v ostatných riadkoch. Podmienka 3 je splnená vďaka kroku 2. A splnenie podmienky 4 vyplýva priamo zo splnenia podmienky 3.

Teraz potrebujeme ešte popísať, ako dostať nenulové riešenie sústavy. Všimnime si, že vďaka tomu, že máme viac čísel ako počet prvočísel (teda $N > \pi(K)$), tak matica má viac stĺpcov ako riadkov. Takže musí existovať nejaký stĺpec, ktorý neobsahuje vedúci prvok. Označme ich $g_1, g_2, \ldots, g_f$. Ukážeme, že nech položíme $x_{g_1}, x_{g_2}, \ldots, x_{g_f}$ akékoľvek, stále vieme dopočítať korektné riešenie (potom stačí niektorej z týchto neznámych priradiť jednotku, aby sme dostali nenulové riešenie). Neznáme, ktoré musíme dopočítať, patria k tým stĺpcom, čo obsahujú vedúci prvok. Z podmienky 2 TRM vyplýva, že nenulové miesto je len v jednom riadku, teda len 1 rovnica popisuje podmienky pre konkrétnu neznámu. A z tej vieme hodnotu neznámej dorátať bez toho, aby sme došli k nejakému sporu. Hurá. Hotovo.

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

**Zhrnutie postupu** Najprv každé číslo zo vstupu rozložíme na prvočísla. Potom na základe týchto rozkladov vybudujeme patričnú sústavu rovníc a popíšeme ju maticou. Následne maticu upravíme na TRM. Z TRM už vieme nájsť nejaké nenulové riešenie a teda aj patričnú podmnožinu.

Časové nároky: Rozklad na prvočísla: $O(N\pi(K))$, Gausova eliminačná metóda $O(N\pi(K)^2)$, zbytok $O(N\pi(K))$, celkovo $O(N\pi(K)^2)$. Pamäťové nároky $O(N\pi(K))$.

**Niekoľko poznámok na záver** Existujú aj rýchlejšie algoritmy na riešenie sústav rovníc. Bohužiaľ ich popis je strašidelný. Niekto by mohol položiť otázku, načo je úloha tohoto typu dobrá. To, čo sme tu robili, sa využíva vo finálnej fáze niektorých faktorizačných algoritmov. Konkrétne odporúčam hľadať Dixon algorithm a Quadratic sieve.

## Listing programu:

```
#include <cstdio>
#include <vector>

using namespace std;


int mat[3010][3010];
int v[3010];
int n, k;

vector<int> primes;

void genPrimes(int lim)
{
    bool *p = new bool[lim+1];
    for(int i = 0; i <= lim; i++)
    {
        p[i]=true;
    }
    for(int i = 2; i <= lim; i++)
    {
        if(p[i]){
            primes.push_back(i);
            for(int j = i*2; j <= lim; j+=i)
            {
                p[j]=false;
            }
        }
    }
    delete[] p;
}

void solveMat()
{
    int r, temp, stop=primes.size();
    for(int i = 0; i < primes.size()+1;i++)
        v[i]=-1;
    int ro = 0;
    for(int i = 0; i < primes.size()+1; i++)
    {
        //najdi jednotku
        r=-1;
        for(int j = ro; j < primes.size(); j++)
        {
            if(mat[j][i]==1){ r = j; break; }
```



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
        }
        if(r==-1) {v[i]=1; continue;}
        //swapni riadky
        for(int j = 0; j < primes.size()+1; j++)
        {
            temp = mat[ro][j]; mat[ro][j]=mat[r][j]; mat[r][j]=temp;
        }
        //odcitaj prislusne veci
        for(int j = 0; j < primes.size(); j++)
        {
            if(j==ro) continue;
            if(mat[j][i]==0) continue;
            for(int k = 0; k < primes.size()+1; k++)
            {
                mat[j][k]^=mat[ro][k];
            }
        }
        ro++;
    }

    //spracuj vysledok matice (posledne 1)
    ro = 0;
    for(int i = 0; i < primes.size()+1; i++)
    {
        if(v[i]==-1)
        {
            for(int j = 0; j < primes.size()+1; j++)
            {
                v[i]+=mat[ro][j];
            }
            v[i]%=2;
            ro++;
        }
    }
}

int main()
{
    scanf("%d %d", &n, &k);
    genPrimes(k);
    vector<long long> cisla;
    long long a;
    int co;
    for(int i = 0; i < n; i++)
    {
        scanf("%lld", &a);
        cisla.push_back(a);
        for(int j = 0; j < primes.size(); j++)
        {
            co = 0;
            while(a%primes[j]==0)
            {
                co++;
                a/=primes[j];
            }
            mat[j][i]=co%2;
        }
    }
    solveMat();
    for(int i = 0; i < primes.size()+1; i++)
```



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
        if(v[i]==1) printf("%lld ", cisla[i]);
    printf("\n");
    return 0;
}
```

opravoval Bob
(max. 25 bodov)

## 9. Ťažká pyramída

Najjednoduchšie je začať stavať pyramídu zhora, od posledných kvádrov: na vrch dáme jeden ($N$-tý) kváder, pod neho dáme čo najmenej kvádrov, aby vytvorili poschodie prípustnej dĺžky, a rovnakým postupom staviame nižšie a nižšie poschodia. No toto riešenie nie je správne, zlyhá napríklad na vstupe (7, 6, 2, 3). Najvyššia pyramída má 3 poschodia a dá sa postaviť, len ak položíme na vrch kvádre s dĺžkami 2 a 3. Za toto a iné nekorektné riešenia ste mohli získať najviac 1 bod.

S použitím rekurzie sa dal naprogramovať tiež veľmi jednoduchý brute-force. Kvádre prechádzame spredu a vždy, keď je na výber, či stavať nové poschodie alebo ostať na aktuálnom, vyskúšame obe možnosti. Toto riešenie behá na náhodných vstupoch prekvapujúco rýchlo, ale v najhoršom prípade (keď sa vetví v každom kroku) má časovú zložitosť $\Theta(2^N)$. Exponenciálne riešenia získavali 8 bodov.

**Základ vzorového riešenia**

Pyramídu budeme stavať odhora. Označme $P_i$ najvyššiu pyramídu, ktorá sa dá postaviť z kvádrov $i, \ldots, N$; ak je takých viac, zoberieme tú najužšiu[5] z nich. Budeme postupne hľadať pyramídy od $P_N$ do $P_1$ a pri tom využívať už nájdené pyramídy (táto technika sa nazýva dynamické programovanie). $P_N$ má výšku 1 a tvorí ju $N$-tý kváder, iná pyramída sa z neho postaviť nedá. Keď hľadáme pyramídu $P_i$, vyskúšame pre všetky $j > i$ vytvoriť spodné poschodie pyramídy $P_i$ z kvádrov $i, \ldots, j-1$ a na to položiť $P_j$ (to sa dá, iba keď nie je súčet dĺžok kvádrov $i, \ldots, j-1$ menší ako šírka $P_j$). Pretože $j > i$, počas hľadania $P_i$ už poznáme $P_j$. Výstupom je $P_1$. Každú pyramídu $P_i$ hľadáme v čase $O(N)$, spolu má toto riešenie časovú zložitosť $O(N^2)$ a bolo za neho (alebo za iné s rovnakou zložitosťou) 17 bodov.

Otázkou je, či je toto riešenie korektné. Keď pri hľadaní $P_i$ skúsime pre nejaké $j$ postaviť spodné poschodie z kvádrov $i, \ldots, j-1$ a na to môžeme položiť $P_j$, je to zjavne optimálne – neexistuje vyššia pyramída z kvádrov $j, \ldots, N$ ako $P_j$. V prípade, že sa pyramída $P_j$ nedá položiť na spodné poschodie (lebo je príliš široká), nastáva problém: čo keď sa na neho dá položiť nejaká iná pyramída zložená z kvádrov $j, \ldots, N$? Nedá, pretože neexistuje užšia pyramída z kvádrov $j, \ldots, N$ ako $P_j$. Toto tvrdenie nie je úplne zrejmé; podľa definície máme zaručené iba to, že $P_j$ je najužšia z *najvyšších* pyramíd. Záujemcov čaká dôkaz na konci tohto vzoráku.

**Prefixové sumy**

Budeme potrebovať rýchlo zistiť súčet dĺžok nejakého súvislého úseku kvádrov. Predpočítame si preto prefixové sumy. Označme $S_i$ súčet dĺžok kvádrov $1, \ldots, i$; špeciálne $S_0 = 0$. Hodnoty $S_i$ vieme vypočítať v čase $O(N)$ jednoduchým prechodom, pretože $S_i = S_{i-1} +$ (dĺžka $i$-teho kvádra). Potom súčet dĺžok kvádrov $a, \ldots, b$ vypočítame ako $S_b - S_{a-1}$ v konštantnom čase.

**Užitočné pozorovanie**

Výšky pyramíd $P_i$ sa s klesajúcim $i$ nezmenšujú; po pridaní nejakých kvádrov na začiatok vieme postaviť aspoň rovnako vysokú pyramídu. Tiež platí, že ak sme pri hľadaní $P_i$ mohli položiť na spodné poschodie tvorené kvádrami $i, \ldots, j-1$ pyramídu $P_j$, budeme ju môcť

[5] Pod šírkou pyramídy rozumieme súčet dĺžok kvádrov najnižšieho poschodia.



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

použiť aj pri hľadaní $P_{i-1}$, keď ju položíme na spodné poschodie tvorené kvádrami $i-1, i, \ldots, j-1$.

**Vylepšenie vzorového riešenia**

Na základe predchádzajúceho odseku vieme získať rýchlejšie riešenie. Keď nájdeme pyramídu $P_i$, predpočítame si najväčšie $k$ také, že pri hľadaní $P_k$ budeme môcť využiť $P_i$. To znamená, že súčet dĺžok kvádrov $k, \ldots, i-1$ (teda $S_{i-1} - S_{k-1}$) je väčší alebo rovný šírke $P_i$. Nájsť $k$ dokážeme binárnym vyhľadávaním čísla $[S_{i-1} - (\text{šírka } P_i)]$ v poli prefixových súm v čase $O(\log N)$. Takto budeme už pri hľadaní pyramídy $P_i$ poznať všetky vhodné $j$, teda pyramídy $P_j$, ktoré môžeme položiť na spodné poschodie $P_i$. Určite nič nepokazíme, keď z nich zoberieme tú s najnižším indexom – žiadna z ostatných nemôže byť vyššia. Táto myšlienka sa dá jednoducho implementovať tak, aby časová zložitosť celého algoritmu bola $O(N \log N)$. Riešenia s takouto časovou zložitosťou získali 22 bodov.

**A ešte jedno**

Môžeme sa dokonca zaobísť bez binárneho vyhľadávania použitím podobnej techniky ako v príklade *O N mešcoch* z prvého kola. Budeme si udržiavať zoznam kandidátov (pyramídy $P_j$), ktorých môžme využiť pri hľadaní ďalších pyramíd $P_i$. Keď nájdeme ďalšiu pyramídu $P_i$, pridáme ju na koniec zoznamu. Predtým však odstránime z konca zoznamu všetky tie pyramídy, ktoré sa budú dať použiť až neskôr ako $P_i$, lebo určite nie sú vyššie a $P_i$ sa bude dať vždy použiť namiesto nich. Vďaka tomu budú kandidáti v zozname utriedení podľa výšky aj použiteľnosti: na jeho začiatku bude najnižšia, ale najskôr použiteľná pyramída. Pri hľadaní ďalšej pyramídy $P_i$ už stačí len vybrať zo začiatku zoznamu tých kandidátov, ktorí sa dajú položiť na spodné poschodie $P_i$. Posledného z nich použijeme (nie je nižší ako ostatní) a ostatných zahodíme.

*Použiteľnosť* pyramídy $P_i$ si reprezentujeme ako také číslo $x$, že $S_{i-1} - x = S_{j-1} - S_{i-1}$ ($j$ je prvý kváder v druhom poschodí pyramídy $P_i$). Pyramída $P_i$ bude potom použiteľná pri hľadaní pyramídy $P_k$ práve vtedy, keď $S_{k-1} \leq x$. Je to len nerovnosť „súčet dĺžok kvádrov $k, \ldots, i-1$ nesmie byť menší ako súčet dĺžok kvádrov $i, \ldots, j-1$“ prepísaná pomocou prefixových súm. Vďaka tejto reprezentácii budeme môcť ľahko porovnávať dve pyramídy podľa použiteľnosti: tá, ktorá má väčšie $x$, je použiteľná skôr.

Zoznam kandidátov sa dá implementovať pomocou spájaného zoznamu, takto dosiahneme vkladanie a odstraňovanie prvkov zo začiatku aj konca zoznamu v konštantnom čase. Vo vzorovom programe je namiesto toho použitá dátová štruktúra `deque` z knižnice STL. Každú z $N$ pyramíd $P_i$ vložíme aj vyberieme zo zoznamu kandidátov práve raz, preto je časová zložitosť celého algoritmu $O(N)$. Pamäťová zložitosť je tiež $O(N)$, lebo v zozname máme vždy najviac $N$ kandidátov. Riešenie v čase lineárnom od $N$ by samozrejme dostalo plných 25 bodov.

**Dôkaz nakoniec**

Ešte dokážeme pre všetky prirodzené čísla $N$ a dĺžky kvádrov $w_1, \ldots, w_N$, že najužšia z najvyšších pyramíd, ktoré sa dajú z týchto kvádrov postaviť, je súčasne aj najužšia zo všetkých pyramíd, ktoré sa dajú z týchto kvádrov postaviť (v tom zmysle, že neexistuje užšia). Takúto pyramídu nazvime *optimálna*. Dokazovať budeme matematickou indukciou vzhľadom na $N$. Pre jeden kváder tvrdenie zjavne platí, pyramída sa dá z neho postaviť jediným spôsobom. Keď už máme tvrdenie dokázané pre všetky počty kvádrov od 1 do $N-1$, dokážeme ho pre $N$ sporom.

Majme kvádre $1, \ldots, N$; označme $P$ optimálnu pyramídu a $Q$ nejakú pyramídu, ktorá je od $P$ užšia. Označme $i$ a $j$ posledné kvádre v spodných poschodiach pyramíd $P$ a $Q$. Platí $j < i$, lebo $Q$ je užšia ako $P$. Na spodnom poschodí pyramídy $P$ je položená nejaká pyramída z kvádrov $i+1, \ldots, N$ a pretože pre tieto kvádre ($N - i < N$) už máme tvrdenie dokázané, môžeme ju nahradiť optimálnou pyramídou z kvádrov $i+1, \ldots, N$; podobne pre $Q$. Vieme, že najvyššia pyramída z kvádrov $i+1, \ldots, N$ nie je vyššia ako najvyššia pyramída z kvádrov



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

$j+1, \ldots, N$. Z toho ale vyplýva, že $P$ nie je vyššia ako $Q$, čo je v spore s prepokladom, že $P$ je optimálna pyramída z kvádrov $1, \ldots, N$.

### Listing programu:

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

int main(){
    int N;
    cin >> N;
    vector<int> S(N + 1, 0);
    for(int i = 1; i <= N; ++i){
        cin >> S[i];
        S[i] += S[i - 1]; // prefixové sumy
    }

    vector<int> D(N + 1); // výšky pyramíd P
    vector<int> X(N); // od kedy sa dá pyramída použiť
    deque<int> Q; // zoznam kandidátov
    D[N] = 0;
    int j = N; // najlepší kandidát

    for(int i = N - 1; i >= 0; --i){
        while(!Q.empty() && (S[i] <= X[Q.front()])){
            j = Q.front(); // vyberie lepšieho kandidáta
            Q.pop_front();
        }

        D[i] = D[j] + 1;

        X[i] = S[i] * 2 - S[j]; // S[i] - X[i] == S[j] - S[i]
        while(!Q.empty() && (X[Q.back()] <= X[i]))
            Q.pop_back(); // vyhodí neskôr použiteľných
        Q.push_back(i);
    }

    cout << D[0] << endl;
}
```

opravoval MišoF
(max. 25 bodov)

## 10. Tajomstvá geológie

Začneme upútavkou a dobrou radou, a potom sa už pustíme na cestu možnými riešeniami.

**Upútavka**

V tomto vzorovom riešení sa stretneme s mnohými užitočnými vecami. Naučíme sa, ako efektívne zostrojiť nasledujúcu permutáciu, povieme si niečo o kódovaní množín do bitov a aj o tom, ako súvisí naša súťažná úloha s teóriou grafov. A na záver možno príde aj kúzelník. Alebo nie.

**Dobrá rada**

Pri čítaní tohto vzorového riešenia je vhodné konzumovať. Odporúčaný pokrm je palacinka s náplňou „geologická špecialita“ z palacinkárne Lacinka, ale v princípe aj každý iný pokrm je lepší ako prázdne brucho. A že je vzorové riešenie dlhšie, prázdne brucho reálne hrozí každému, kto nedá na moje rady :)

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

**Základné pozorovania a dohody**

Kvôli ľahšiemu vyjadrovaniu si dohodnime trochu terminológie. Micova skúška nech trvá $N$ dní, pričom každý deň bude odpovedať na jednu otázku. Keďže listingy programu budeme písať v C++[6], oboje – aj dni, aj otázky – si očíslujeme od 0 po $N-1$.

Formát vstupu a výstupu (pravdepodobnosti udané v percentách) je dosť nešikovný, pravdepodobnosti si radšej interne budeme reprezentovať ako čísla z intervalu $[0,1]$. Presnejšie, hodnota $P_{i,j}$ bude pravdepodobnosť, že Mic správne zodpovie otázku $i$ v deň $j$.

V našej úlohe hľadáme to najlepšie spomedzi $N!$ poradí odpovedania na otázky. Keď si vyberieme jedno konkrétne poradie, jeho pravdepodobnosť úspechu nájdeme jednoducho tak, že vynásobíme pravdepodobnosti správneho zodpovedania jednotlivých otázok.

**Základné riešenie**

Teraz už vieme úlohu *nejako* riešiť: stačí vyskúšať všetky permutácie množiny otázok a vybrať najlepšiu. Každá permutácia totiž predstavuje jedno možné poradie, v ktorom môže Mic odpovedať.

V C++ toto ide veľmi ľahko:

```
// vyrobime si prvu permutaciu
int perm[N];
for (int i=0; i<N; ++i) perm[i] = i;

// a prejdeme cez vsetky permutacie a najdeme najlepsiu
double best = 0;
do {
  double current = 1;
  for (int i=0; i<N; ++i) current *= P[ i ][ perm[i] ];
  best = max( best, current );
} while (next_permutation(perm,perm+N));
```

Funkcia `next_permutation` vyrobí zo zadanej permutácie nasledujúcu v lexikografickom poradí a vráti `true`. Jedinou výnimkou je situácia, kedy je na vstupe posledná permutácia. Vtedy z nej `next_permutation` vyrobí prvú permutáciu a vráti `false`. Uvedený cyklus sa teda práve raz vykoná pre každú permutáciu. (Uvedomte si, že testovanie podmienky `while` musí byť na konci cyklu – ak by sme ju testovali na začiatku, telo cyklu by sa nevykonalo pre úplne prvú permutáciu.)

**Ako na nasledujúcu permutáciu**

Ak používame jazyk, ktorý funkciu `next_permutation` nemá, ľahko si ju naprogramujeme sami. Ako na to? Pokúsme sa napríklad vyrobiť permutáciu, ktorá v lexikografickom poradí nasleduje hneď po $(4,0,6,2,3,7,5,1)$. Spomedzi permutácií začínajúcich $(4,0,6,2,3,\ldots)$ je táto permutácia posledná možná, lebo jej „chvost“ $7,5,1$ je už v klesajúcom poradí. Ešte ale existujú ďalšie permutácie začínajúce $(4,0,6,2,\ldots)$, lebo $3<7$. Ktorá je nasledujúca z nich?

Namiesto hodnoty 3 zjavne musíme dať najbližšiu väčšiu hodnotu – čo je v našom prípade 5. Nasledujúca permutácia teda začína $(4,0,6,2,5,\ldots)$. A spomedzi týchto permutácií chceme tú najmenšiu možnú, ostatné hodnoty teda umiestnime v rastúcom poradí: $(4,0,6,2,5,1,3,7)$.

Celý postup teda môžeme všeobecne formulovať nasledovne:

- Idúc od konca nájdi najväčšie $i$ také, že $P[i]<P[i+1]$.
  (Ak také $i$ neexistuje, máme na vstupe poslednú permutáciu, ošetríme špeciálne.)
- Idúc od konca nájdi najväčšie $j$ také, že $P[j]>P[i]$.
- Vymeň $P[j]$ a $P[i]$.

[6] A navyše je toto číslovanie intuitívnejšie akonáhle kódujeme množiny do bitov. O tom viac o chvíľu.



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

- Hodnoty od pozície $i+1$ do konca sú aj po tejto výmene utriedené zostupne, stačí teda tento úsek poľa reverznúť.

Jedno volanie tejto funkcie má teda časovú zložitosť lineárnu od dĺžky časti, ktorú je potrebné zmeniť. Všimnite si, že toto je lepšie ako časová zložitosť lineárna od $N$. Dá sa ukázať, že celková časová zložitosť $N!$ volaní tejto funkcie je vďaka tomu len $\Theta(N!)$, nie $\Theta(N \cdot N!)$.

V našom programe z predchádzajúcej časti tohto vzorového riešenia však pre každú permutáciu lineárne prejdeme všetky jej prvky, takže jeho časová zložitosť je $\Theta(N \cdot N!)$.

**Nie je exponenciála ako faktoriál**

Riešenia, ktoré skúšajú všetky permutácie, sa skoro vždy dajú takmer zadarmo zlepšiť. To neplatí len v tejto úlohe, ale aj v mnohých iných – napríklad taký problém obchodného cestujúceho („nájdite takú permutáciu miest, aby nás okružná cesta v uvedenom poradí stála čo najmenej“).

Predstavme si, že si Mic najskôr tvrdohlavo povedal: „Posledné tri otázky chcem hovoriť o kremeni, o grafite a úplne na koniec o ametyste.“ Prezrel teda všetky možné poradia ostatných $N-3$ otázok a z nich si vybral to najlepšie.

Potom si ale dal palacinku a pri nej si to celé rozmyslel úplne ináč. Zmení poradie posledných troch otázok! Najskôr ametyst, potom kremeň a úplne na konci grafit!

Už-už sa chcel Mic pustiť do toho, aby ešte raz preskúšal poradie prvých $N-3$ otázok... keď si zrazu uvedomil, že to vôbec nemusí robiť. Optimálne poradie ostatných otázok bude predsa to isté, ako predtým!

A na tomto Micovom pozorovaní môžeme postaviť lepší algoritmus pre našu úlohu. Nepotrebujeme totiž prezerať všetkých $N!$ permutácií, bude nám stačiť pozerať sa na všetky podmnožiny množiny otázok.

Jeden možný spôsob, ako takéto riešenie sformulovať: nech $M$ je ľubovoľná podmnožina množiny otázok, ktoré má Mic zodpovedať. Označme $R(M)$ optimálnu pravdepodobnosť, že sa Micovi podarí v dňoch 0 až $|M|-1$ zodpovedať všetky tieto otázky.

Nás zaujíma hodnota $R(V)$, kde $V = \{0, 1, 2, \ldots, N-1\}$. Aby sme zistili túto hodnotu, postupne spočítame hodnoty $R(M)$ pre všetky $M \subseteq V$. Tento algoritmus je teda istou formou dynamického programovania.

Ako budeme hodnoty $R(M)$ počítať? Napríklad pomocou nasledujúcej úvahy: niektorú z otázok $x \in M$ musíme zodpovedať ako poslednú, v deň $|M|-1$. Vyskúšajme teda všetky možnosti, ktorá to bude. Zakaždým takto dostaneme nový, menší problém: zostala nám množina otázok $M - \{x\}$. No a pre tie je optimálna pravdepodobnosť $R(M - \{x\})$. Preto platí:

$$R(M) = \max_{x \in M} R(M - \{x\}) \cdot P_{x,|M|-1}$$

Slovne: Pre každé možné $x$ vynásobíme dve pravdepodobnosti: optimálnu pravdepodobnosť, že za prvých $|M|-1$ dní Mic správne zodpovie otázky z $M - \{x\}$ a pravdepodobnosť, že v nasledujúci deň správne zodpovie otázku $x$. Spomedzi týchto možností si vyberieme tú, ktorá je pre nás najlepšia – čiže tú, v ktorej je výsledná pravdepodobnosť najvyššia.

Teraz si už len stačí uvedomiť, že každú z hodnôt $R(M)$ nám stačí vypočítať raz a uložiť si ju do vhodného poľa v pamäti.

Celý algoritmus teda môžeme zhrnúť napríklad nasledovne:

- Zoradíme si všetkých $2^N$ podmnožín množiny $V$ podľa veľkosti.
- Postupne pre každú množinu $M$ vypočítame hodnotu $R(M)$ a zapamätáme si ju

Všimnite si, že vďaka zoradeniu množín podľa veľkosti platí, že v okamihu, keď spracúvame ľubovoľnú množinu $M$, už sme spracovali všetky množiny tvaru $M - \{x\}$. Vieme teda všetko potrebné na to, aby sme vypočítali hodnotu $R(M)$.

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

**Efektívna a efektná implementácia**

Každú podmnožinu množiny $\{0, 1, \ldots, N-1\}$ si môžeme predstaviť ako postupnosť $N$ núl a jedničiek: 0 znamená, že dotyčný prvok do našej množiny nepatrí, 1 že áno. Napríklad pre $N = 5$ množine $\{0, 1, 3\}$ zodpovedá postupnosť $(1, 1, 0, 1, 0)$.

No a postupnosť núl a jedničiek, to je predsa postupnosť bitov. A tieto bity predstavujú v dvojkovej sústave nejaké číslo. Napríklad našej postupnosti bitov $(1, 1, 0, 1, 0)$ zodpovedá číslo $1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 = 11$.

Takto sme vlastne definovali jednoznačné priradenie (matematici to volajú *bijekcia*) medzi všetkými množinami, ktoré nás zaujímajú, a všetkými $N$-bitovými celými číslami – presnejšie teda číslami od 0 po $2^N - 1$.

Takáto forma reprezentácie množín je vynikajúca pre ich reprezentáciu v počítači. Vieme totiž s úspechom použiť viaceré existujúce operácie. Napríklad taký bitový and predstavuje prienik množín, bitový or ich zjednotenie. Pomocou bitových posunov vieme ľahko testovať, či konkrétny prvok do množiny patrí alebo nie – stačí sa pozrieť na príslušný bit čísla, ktoré ju reprezentuje.

A navyše si môžeme všimnúť jednu veľmi dôležitú vlastnosť tejto reprezentácie: keď si zoberieme ľubovoľné dve množiny $A \neq B$ také, že $A \subseteq B$, tak $A$ má určite menšie číslo ako $B$. To je zjavné: postupnosť bitov predstavujúca $A$ vznikne z postupnosti bitov predstavujúcej $B$ tak, že niektoré 1 zmeníme na 0 – a tým príslušné číslo zmenšíme.

Pri implementácii predchádzajúceho algoritmu môžeme teda spokojne vynechať krok „Zoradíme si všetkých $2^N$ podmnožín množiny $V$ podľa veľkosti.“ Pokojne nám stačí spracúvať množiny usporiadané podľa ich čísla. Aj v tomto poradí bude pre ľubovoľnú množinu $M$ platiť, že všetky jej potrebné podmnožiny spracujeme skôr ako ju samú.

Takto teda dostávame až neuveriteľne stručnú implementáciu:

```
int R[1<<N]; // pole velkosti 2^N na uz spocitane vysledky
R[0] = 1;
for (int M=1; M<(1<<N); ++M) { // pre kazdu mnozinu M
  R[M] = 0;
  int velkost = __builtin_popcount( M );
  for (int x=0; x<N; ++x) {     // pre kazdy prvok x
    if (M & 1<<x) {             // ak mnozina M obsahuje prvok x
      R[M] = max( R[M], R[M ^ 1<<x] * P[x][velkost-1] );
    }
  }
}
```

Dovysvetlíme detaily:

- Operátor `<<` je bitový posun doľava. Číslo `1 << x` má teda hodnotu $2^x$ a pre nás predstavuje množinu $\{x\}$.
- Výraz `(M & 1<<x)` má dve možné hodnoty: ak je v $M$ prvok $x$, má premenná `M` nastavený príslušný bit, a teda bitovým and-om čísel `M` a `1<<x` je kladné číslo `1<<x`. V tomto prípade je teda naša podmienka vyhodnotená ako splnená. V opačnom prípade je výsledkom 0 a podmienka je nesplnená.
- Operátor `^` je bitový xor. Výraz `M ^ 1<<x` teda môžeme čítať: v premennej `M` zneguj bit `x`. Keďže v dotyčnej situácii vieme, že $M$ obsahuje bit $x$, zmení táto operácia bit $x$ z 1 na 0 – a teda výsledné číslo je práve číslom množiny $M - \{x\}$.
  (Namiesto tohto zápisu sme mohli použiť aj zápis `M & ~(1<<x)`, teda `M` and not `1<<x`.)
- Funkcia `__builtin_popcount(M)` je neštandardným rozšírením v kompilátore gcc. Vracia počet bitov čísla `M`, ktoré majú hodnotu 1. V prípade, že túto funkciu tvoj kompilátor nepozná, môžeš ju priamočiaro implementovať nasledovne:
  `int velkost=0; for (int i=0; i<N; ++i) if (M & 1<<i) ++velkost;`



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Alebo o niečo efektívnejšie takto:

```
int velkost=0, pom=M; while (pom) ++velkost, pom &= pom-1;
```

(Rozmyslite si, že `x & (x-1)` vznikne z `x` tak, že vynulujeme posledný jednotkový bit.)

Toto riešenie má pamäťovú zložitosť $\Theta(2^N)$ a časovú zložitosť $\Theta(N \cdot 2^N)$. To je výrazné zlepšenie oproti predchádzajúcemu riešeniu – toto je prakticky použiteľné tak po $N = 30$, to prvé tak po $N = 12$.

Naša úloha však mala aj omnoho, omnoho efektívnejšie riešenie. A práve tam teraz smerujeme.

**Logaritmy**

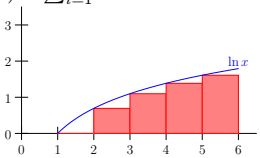
Každý logaritmus (s bázou väčšou ako jedna) má dve príjemné vlastnosti. Prvá vlastnosť je, že je rastúci. A druhá, že pre ľubovoľné kladné $a$ a $b$ platí $\log(ab) = \log a + \log b$.

Tieto dve vlastnosti dokopy majú na svedomí skutočnosť, že s logaritmami sa často stretneme, keď sa snažíme optimalizovať hodnotu nejakého súčinu. Keďže logaritmus je rastúci, je jedno, či hľadáme optimálnu hodnotu nejakého výrazu $Z$ alebo výrazu $\log Z$. No a ak $Z$ je nejaký súčin, tak $\log Z$ je rovný súčtu logaritmov jeho činiteľov. A so súčtom sa hneď pracuje lepšie ako so súčinom.

**Matematická odbočka**

Ukážeme si príklad, ktorý s našou úlohou síce nijak nesúvisí, ale pekne ukáže, ako môžu byť niekedy logaritmy užitočné. V našom príklade nájdeme horný odhad čísla $N!$. To spravíme tak, že nájdeme horný odhad čísla $\ln N!$.

Platí $\ln N! = \ln(1 \cdot 2 \cdot \ldots \cdot N) = \sum_{i=1}^{N} \ln i$.




Graficky si túto sumu znázornime ako $N$ obdĺžničkov: nakreslíme si os $x$ a nad úsečkou od $i$ po $i+1$ nakreslíme obdĺžnik výšky $\ln i$, pre $i$ od 1 po $N$. Súčet plôch týchto obdĺžničkov je zjavne $\sum_{i=1}^{N} \ln i$.

Keď teraz dokreslíme do nášho grafu funkciu $\ln x$, vidíme, že na intervale od 1 po $N+1$ všetky naše obdĺžničky ležia pod ňou. Súčet plôch obdĺžničkov teda môžeme zhora odhadnúť plochou pod grafom $\ln x$, inými slovami, integrálom:

$$\ln N! = \sum_{i=1}^{N} \ln i \leq \int_{1}^{N+1} \ln x \,\mathrm{d}x = (N+1)\ln(N+1) - N$$

Poznámka: Integrály v skutočnosti nevie rátať nik, s výnimkou prvákov na výške v okolí skúšky z analýzy. Pre nás ostatných sú tu nástroje ako `http://integrals.wolfram.com/`.

No a keď vieme, že $\ln Z \leq Y$, tak potom $Z \leq e^Y$. V našom prípade teda dostávame

$$N! \leq e^{(N+1)\ln(N+1)-N} = \frac{e^{(N+1)\ln(N+1)}}{e^N} = \frac{\left(e^{\ln(N+1)}\right)^{N+1}}{e^N} = \frac{(N+1)^{N+1}}{e^N} = e\left(\frac{N+1}{e}\right)^{N+1}$$

Ako cvičenie si môžete rovnakou metódou odvodiť dolný odhad: $N! \geq e\,(N/e)^N$.

**Späť k našej úlohe**

Našou úlohou je nájsť tú permutáciu, pre ktorú je súčin pravdepodobností maximálny možný. Ak teda všetky pravdepodobnosti zlogaritmujeme, prevedieme tým našu úlohu na novú: nájsť tú permutáciu, pre ktorú je súčet nových hodnôt najväčší možný.

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Pozrime sa teraz poriadnejšie na tabuľku $logP$, v ktorej máme logaritmy pravdepodobností zo vstupu. Toto je tabuľka $N \times N$. A my v nej chceme vyznačiť $N$ políčok tak, aby žiadne dve neležali v tom istom riadku ani stĺpci a aby súčet ich čísel bol najväčší možný.

Zabudnime na chvíľu na čísla. Vždy, keď sa stretneme s požiadavkou „v žiadnom riadku ani stĺpci nesmieme vybrať viac ako jedno políčko“, malo by nám to pripomenúť podobný koncept z teórie grafov: párovania. Tam tiež nesmie zo žiadneho vrcholu viesť viac ako jedna hrana.

A skutočne, tieto dve situácie spolu súvisia.

Predstavme si bipartitný graf, tvorený dvomi množinami vrcholov. V ľavej množine $L$ je $N$ vrcholov, každý z nich zodpovedá jednému riadku našej tabuľky. V pravej množine $R$ je tiež $N$ vrcholov, každý z nich zodpovedá jednému stĺpcu našej tabuľky. Každé dva vrcholy $l \in L$ a $r \in R$ sú spojené hranou. A zjavne teda každá hrana zodpovedá jednému políčku v našej tabuľke.

A ľahko nahliadneme, že povolené výbery políčok v tabuľke zodpovedajú *úplným párovaniam* v práve definovanom bipartitnom grafe. Totiž keď máme vybranú vhodnú množinu $N$ políčok, z ktorých žiadne dve neležia v tom istom riadku ani stĺpci, tak im zodpovedá $N$ hrán, pričom žiadne dve hrany nemajú spoločný ľavý ani pravý vrchol. A naopak, každej takejto množine $N$ hrán zodpovedá vyhovujúca $N$-tica políčok.

Spomeňme si teraz opäť na čísla na políčkach. Každej hrane v našom grafe priradíme *váhu* rovnú číslu na políčku, ktorému zodpovedá. Našou úlohou je teraz v tomto grafe nájsť *úplné párovanie s najväčšou možnou váhou*.

**Všeobecnejšia úloha – MinCost MaxFlow**

Namiesto toho, aby sme popisovali algoritmy na hľadanie párovania s najväčšou váhou, vyriešime rovno úlohu ostro všeobecnejšiu.

Daný je orientovaný graf $G$, v ktorom sú vyznačené dva vrcholy $a$ a $b$. Každá hrana $e$ tohto grafu má priradené dve čísla: nezápornú celočíselnú kapacitu $k_e$ a nezápornú reálnu cenu $c_e$.

My sa nachádzame vo vrchole $a$ a máme tu nevyčerpateľnú zásobu tovaru, ktorý chceme v čo najväčšom množstve dodávať do vrcholu $b$. Hrany grafu si môžeme predstaviť ako navzájom nezávislé jednosmerky. Kapacita hrany udáva, koľko kamiónov s naším nákladom môže po danej ceste denne prejsť, a cena hrany udáva, koľko nás stojí poslanie jedného kamiónu po tejto hrane.

Ak by sme na chvíľu zabudli na ceny, prirodzená otázka znie: koľko najviac našich kamiónov môže denne prichádzať do $b$? Tento problém sa volá úloha o maximálnom toku, skrátene MaxFlow.

Akonáhle máme aj ceny, vzniká množstvo zložitejších otázok. Jedna z najľahšie zodpovedateľných je práve tá naša: za predpokladu, že chcem z $a$ do $b$ denne posielať najväčší možný počet kamiónov, koľko najmenej ma to môže stáť a kadiaľ ich mám posielať? Hľadáme teda ten maximálny tok, ktorý má spomedzi všetkých maximálnych tokov najmenšiu možnú celkovú cenu. (Pripomíname, že treba zaplatiť za každý prejazd kamióna po každej hrane. Cenu toku teda dostaneme tak, že pre každú hranu vynásobíme jej cenu a počet kamiónov, ktoré ňou pôjdu, a všetky tieto poplatky sčítame.)

**Hľadanie párovania pomocou toku**

Skôr, než si ukážeme, ako MinCost MaxFlow hľadať, ukážeme si, ako pomocou neho nájsť párovanie s najväčšou váhou.

Majme náš úplný bipartitný graf $G$. Všetky hrany v ňom zorientujeme tak, aby viedli zľava doprava. Každá z týchto hrán bude teraz mať kapacitu 1 a cenu $-v$, kde $v$ je jej doterajšia váha.[7]

[7] Uvedomte si, že keďže pravdepodobnosti boli všetky z intervalu $[0, 1]$, ich logaritmy boli nekladné, a teda pre každú hranu je teraz jej cena nezáporná.



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Ku $G$ ešte pridáme dva nové vrcholy $a$ a $b$. Z vrcholu $a$ do každého vrcholu ľavej partície pridáme hranu s kapacitou 1 a cenou 0. A podobne, z každého vrcholu pravej partície pridáme hranu s kapacitou 1 a cenou 0 do $b$.

Malo by byť zjavné, že MinCost MaxFlow v takto upravenom grafe $G$ skutočne zodpovedá úplnému párovaniu s maximálnou váhou v pôvodnom grafe $G$. (Stačí sa pozrieť na $N$ hrán, ktorými v optimálnom riešení idú kamióny z ľavej do pravej partície.)

**Maximálny tok, zvyšková sieť a zlepšujúce cesty**

Formálne, „nájsť tok“ znamená priradiť každej hrane $e$ veľkosť toku po nej $f_e$, pre ktorú platí $0 \leq f_e \leq k_e$. Navyše v každom vrchole okrem $a$ a $b$ musí platiť prvý Kirchhoffov zákon – teda súčet tokov na prichádzajúcich hranách musí byť rovný súčtu tokov na odchádzajúcich hranách. Veľkosť celého toku je rozdiel medzi množstvom odtekajúcim z $a$ a množstvom pritekajúcim do $a$ (ktoré je pri praktických aplikáciách väčšinou nulové).

Predstavme si, že hľadáme maximálny tok a už máme naplánované, ako poslať niekoľko kamiónov. V tomto okamihu môžeme zobrať graf $G$ a vytvoriť z neho nový graf $G'$: pre každú hranu $e = (u \to v)$ v $G$, ktorá má kapacitu $k_e$ a aktuálne už po nej ide $f_e$ kamiónov, pridáme do $G'$ dve hrany: modrú hranu z $u$ do $v$ s kapacitou $k_e - f_e$ a červenú z $v$ do $u$ s kapacitou $f_e$. Tento nový graf $G'$ voláme *zvyšková sieť*. Predstavuje naše aktuálne možnosti, ako meniť tok.

Cestu z $a$ do $b$ po aktuálnej zvyškovej sieti nazveme *zlepšujúca cesta*. Prečo? Hrany, ktoré sme dali do $G'$, predstavujú naše možnosti, ako zmeniť aktuálny tok. Každá zlepšujúca cesta nám predstavuje jeden možný spôsob, ako cez $G$ pretlačiť ešte jeden kamión. Stačí, keď prejdeme po hranách nájdenej zlepšujúcej cesty a podľa toho upravíme tok v $G$ – vždy, keď ideme po modrej hrane $G'$, na príslušnej hrane $G$ tok zvýšime, a vždy, keď ideme po červenej hrane $G'$, na príslušnej hrane $G$ tok znížime. Takto opäť dostaneme platný tok, ktorého veľkosť bude o 1 väčšia.

No a práve tento postup využíva najjednoduchší algoritmus na hľadanie maximálneho toku (ešte bez cien!) od pánov Forda a Fulkersona. Ten funguje nasledovne:

Ak v zvyškovej sieti existuje cesta z $a$ do $b$ (čo vieme zistiť jednoduchým prehľadávaním), práve sme našli spôsob, ako aktuálny tok vylepšiť. Pošleme teda po uvedenej ceste o jeden kamión viac. A toto opakujeme, kým to ide. Ford s Fulkersonom dokázali, že nech si vyberáme zlepšujúce cesty z $a$ do $b$ ako len chceme, tento proces je vždy konečný a jeho výsledkom je vždy tok najväčšej možnej veľkosti.

Myšlienka dôkazu: nech už neexistuje cesta z $a$ do $b$ v aktuálnej zvyškovej sieti. Rozdeľme si všetky vrcholy $G$ na dve množiny: do $A$ dáme všetky vrcholy vrátane $a$, do ktorých takáto cesta existuje, a do $B$ dáme všetky ostatné. Keďže do žiadneho vrcholu $B$ sa nevieme dostať, tak musí platiť, že každá hrana z $A$ do $B$ je už plne vyťažená a zároveň po žiadnej hrane z $B$ do $A$ neposielame vôbec nič. Tým pádom sme v $G$ našli tzv. *rez*, cez ktorý sa už toho viac pretlačiť nedá – a teda práve nájdený tok je nutne optimálny.

**Cykly so zápornou cenou**

Ak v pôvodnom grafe majú hrany priradené ceny, premietnu sa tieto ceny aj do toho, ako vyzerajú zvyškové siete. Každá modrá hrana (zodpovedajúca zvýšeniu počtu kamiónov na hrane) bude mať cenu rovnú cene príslušnej hrany. Naopak, červené hrany dostanú negáciu pôvodnej ceny – keď dotyčnou hranou pošleme o kamión menej, ušetríme tým.

Teraz ukážeme nasledujúce pomocné tvrdenie: nech $f$ je ľubovoľný tok v našom grafe. Potom platí, že $f$ je najlacnejší spomedzi všetkých tokov rovnakej veľkosti práve vtedy, keď zvyšková sieť zodpovedajúca toku $f$ neobsahuje žiadny záporný cyklus.

Jedna polovica tohto tvrdenia je zjavná: ak by sme v zvyškovej sieti mali záporný cyklus, môžeme po ňom poslať jeden kamión (t. j. upraviť pôvodný tok rovnako ako pri použití zlepšujúcej cesty), čím nezmeníme veľkosť toku, ale znížime jeho cenu.

Opačný smer tvrdenia dokážeme nasledovne: Nech $\overline{f}$ je najlacnejší tok rovnakej veľkosti ako $f$. Zostrojme teraz graf $G''$ podobne ako sme zostrojovali zvyškovú sieť: Pre každú hranu



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

$e = u \to v$ spočítame rozdiel $\overline{f}_e - f_e = \Delta_e$. Ak $\Delta_e > 0$, dáme do $G''$ modrú hranu z $u$ do $v$, a ak $\Delta_e < 0$, dáme do $G''$ červenú hranu z $v$ do $u$. V oboch prípadoch pôjde po tejto hrane tok veľkosti $f''_e = |\Delta_e|$.

Hrany, ktoré máme v $G''$, nám ukazujú, aké zmeny treba spraviť, aby sme z nášho toku $f$ dostali rovnako veľký, ale lacnejší tok $\overline{f}$. Keďže $f$ a $\overline{f}$ boli rovnako veľké, tak v $G''$ má každý vrchol (vrátane $a$ a $b$) rovnaký súčet prichádzajúcich a odchádzajúcich tokov. Z toho vyplýva, že tok v $G''$ sa vždy dá rozbiť na niekoľko jednotkových cyklov.[8] A keďže $\overline{f}$ je lacnejší ako $f$, aspoň jeden z týchto cyklov musí mať záporný súčet cien hrán.

**Algoritmus na hľadanie MinCost MaxFlowu**

Základné tvrdenie, na ktorom bude založený náš algoritmus: nech $f$ je najlacnejší spomedzi všetkých tokov jeho veľkosti. Nech teraz vo zvyškovej sieti pre $f$ nájdeme najlacnejšiu zlepšujúcu cestu $\delta$ a po nej zväčšíme tok o 1. Potom nový tok $f + \delta$ je opäť najlacnejší spomedzi všetkých tokov jeho veľkosti.

Dôkaz: Zoberme ľubovoľný tok $\overline{f}$, ktorý má rovnakú veľkosť ako $f + \delta$. Opäť uvažujme rovnaký graf $G''$, ako v predchádzajúcom prípade. Tentokrát je ale rozdiel tokov $\overline{f}$ a $f$ rovný jednej, preto keď rozbijeme graf $G''$ na cykly, zostane nám ešte jedna nejaká zlepšujúca cesta.

Čo teraz ale vieme povedať? Každý z cyklov, na ktoré sme $G''$ rozbili, musí mať nezápornú cenu, lebo $f$ bol najlacnejší tok jeho veľkosti. A tá zlepšujúca cesta, ktorá nám zostala, má aspoň takú cenu ako $\delta$, lebo $\delta$ je najlacnejšia zo všetkých zlepšujúcich ciest. Dokopy dostávame, že cena $\overline{f}$ je aspoň o cenu $\delta$ väčšia od ceny $f$. Na druhej strane, náš tok $f + \delta$ je od $f$ presne o cenu $\delta$ drahší, a teda je najlacnejší možný.

MinCost MaxFlow teda vieme hľadať nasledovne: začneme s všade nulovým tokom. Teraz dokola hľadáme a používame najlacnejšiu zlepšujúcu cestu, kým to ide. Keď už to nejde, Fordova a Fulkersonova veta nám zaručí, že veľkosť práve nájdeného toku je najväčšia možná. A z práve dokázaného tvrdenia vyplýva, že tento tok je najlacnejší možný tok danej veľkosti.

**Hľadanie najlacnejšej zlepšujúcej cesty**

Posledné, čo nám zostáva ukázať, je algoritmus, ktorým takúto zlepšujúcu cestu vo zvyškovej sieti nájdeme. Inými slovami, hľadáme vo zvyškovej sieti cestu s najmenším súčtom cien. Toto podozrivo pripomína hľadanie najkratšej cesty – až natoľko, že jediný rozdiel je v tom, že číselká priradené hranám nevoláme „dĺžky“, ale „ceny“.

Pozor, nemôžeme ale len tak použiť Dijkstrov algoritmus – niektoré hrany v našom grafe totiž majú ceny záporné.[9] My preto použijeme o rád pomalší algoritmus pánov Bellmana a Forda, ktorý funguje na každom grafe bez záporných cyklov.

Tento algoritmus je veľmi jednoduchý. Na začiatku nastavíme, že do vrcholu $a$ sa vieme dostať s cenou 0, a do každého iného s cenou $\infty$. Teraz budeme dokola opakovať postup, ktorý si nazveme „vylepšenie“. Jedno vylepšenie vyzerá nasledovne: V ľubovoľnom poradí prejdeme všetky hrany grafu. Pre každú hranu $u \to v$ sa pozrieme, či náhodou neplatí, že (doteraz najlepšia cena do $u$) plus (cena hrany $u \to v$) je menej ako (doteraz najlepšia cena do $v$). Ak áno, tak cenu do $v$ príslušne znížime.

Ľahko dokážeme nasledujúce tvrdenie: Po $k$ vylepšeniach pre každý vrchol $v$ platí, že cena, ktorú si práve pre $v$ pamätáme, je menšia alebo rovná ako cena najlacnejšej cesty z $a$ do $v$ spomedzi všetkých ciest, ktoré majú nanajvýš $k$ hrán.

No a keďže sme o vstupe predpokladali, že neobsahuje záporné cykly, má pre každý vrchol $v$ najkratšia cesta doň nanajvýš $N - 1$ hrán, kde $N$ je počet vrcholov grafu. Ak teda postupne spravíme $N - 1$ vylepšení, všetky spočítané hodnoty už budú optimálne.

[8] Dôkaz: začnem hocikde a chodím hocijako po hranách s kladným tokom. Keď sa mi prvýkrát nejaký vrchol zopakuje, úsek mojej cesty medzi týmito opakovaniami tvorí cyklus. Ten odstránim a začnem odznova, a tak dokola, až kým sa všetko neminie.

[9] Existuje však finta, ktorú vieme urobiť a potom použiť Dijkstrov algoritmus. Ak to niekoho zaujíma, nech sa opýta na fóre, doplním.

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Keďže jedno vylepšenie má časovú zložitosť $O(M)$, je celková časová zložitosť algoritmu Bellmana a Forda $O(MN)$.

No a čas nájdenia najlacnejšieho maximálneho toku opakovaním tohto algoritmu má potom časovú zložitosť $O(MNT)$, kde $T$ je veľkosť dotyčného toku.

**Záver pre našu úlohu**

V našej úlohe potrebujeme nájsť najdrahšie párovanie v našom bipartitnom grafe s $N+N$ vrcholmi. Keď túto úlohu prevedieme na MinCost MaxFlow, dostaneme graf s $\Theta(N)$ vrcholmi a $\Theta(N^2)$ hranami. A vieme, že veľkosť maximálneho toku bude určite presne $T = N$. Preto časová zložitosť vyššie uvedeného algoritmu bude $O(N^4)$.

(Ak by sme použili spomínanú fintu a Dijkstrov algoritmus, dostali by sme časovú zložitosť $O(N^3)$. Ani táto ale ešte nie je optimálna, existujú aj efektívnejšie algoritmy. Tie sú však na toto vzorové riešenie už príliš zložité.)

## Listing programu:

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

class MincostMaxflow {
  public:
  class edge {
    public:
      int source, destination, capacity, residue;
      double cost;
      edge(int s, int d, int cap, int res, double cost)
        : source(s), destination(d), capacity(cap), residue(res), cost(cost) { }
  };

  vector< vector<int> > V;
  vector<edge> E;
  void addArc(int source, int destination, int capacity, double cost);
  pair<int,double> getFlow(int source, int sink);
};

void MincostMaxflow::addArc(int source, int destination, int capacity, double cost)
{
  int e = E.size();
  if (source >= int(V.size())) V.resize(source+1);
  if (destination >= int(V.size())) V.resize(destination+1);
  V[source].push_back( e );
  V[destination].push_back( e+1 );
  E.push_back(edge(source,destination,capacity,capacity,cost));
  E.push_back(edge(destination,source,capacity,0,-cost));
}

pair<int,double> MincostMaxflow::getFlow(int source, int sink) {
  if (source >= int(V.size())) V.resize(source+1);
  if (sink >= int(V.size())) V.resize(sink+1);

  int N = V.size(), M = E.size();
  int flowSize = 0;
  double flowCost = 0;
  vector<int> flow(M,0);
  vector<double> potential(N,0);
  while (1) {
    // pomocou bellman-forda najdeme najlacnejsiu zlepsujucu cestu
```



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    vector<int> from(N,-1);
    vector<double> dist(N,1e30);
    dist[source] = 0;

    for (int i=0; i<N-1; ++i)
      for (int j=0; j<M; ++j) {
        if (E[j].residue == 0) continue; // neda sa nou ist
        if (dist[ E[j].destination ] <= dist[ E[j].source ] + E[j].cost + 1e-9)
            continue; // nezlepsi
        dist[ E[j].destination ] = dist[ E[j].source ] + E[j].cost;
        from[ E[j].destination ] = j;
      }

    // ak sme ziadnu cestu nenasli, koncime
    if (from[sink] == -1) return make_pair(flowSize,flowCost);

    // prejdeme celu najdenu zlepsujucu cestu a zistime, kolko sa po nej da pretlacit
    int canPush = 987654321, where = sink;
    while (1) {
      if (from[where]==-1) break;
      canPush = min(canPush, E[ from[where] ].residue );
      where = E[ from[where] ].source;
    }

    // este raz ju prejdeme a pretlacime nou dane mnozstvo toku
    where = sink;
    while (1) {
      if (from[where]==-1) break;
      E[ from[where] ].residue -= canPush;
      E[ from[where]^1 ].residue += canPush; // uprava hodnoty pre zodpovedajucu
hranu opac. farby
      flowCost += canPush * E[ from[where] ].cost;
      where = E[ from[where] ].source;
    }
    flowSize += canPush;
  }
}

int main(void) {
  int N;
  cin >> N;
  MincostMaxflow MCMF;
  // nacitame maticu a zostrojime nas bipartitny graf
  for (int i=0; i<N; i++)
    for (int j=0; j<N; j++) {
      int x;
      cin >> x;
      if (x) MCMF.addArc(1+i,N+1+j,1,-log(x/100.));
    }
  // pridame zaciatok a=0 a koniec b=2N+1
  for (int i=0; i<N; i++) MCMF.addArc(0,1+i,1,0);
  for (int i=0; i<N; i++) MCMF.addArc(N+1+i,2*N+1,1,0);
  // najdeme maximalny tok a vypocitame vysledok
  pair<int,double> res = MCMF.getFlow(0,2*N+1);
  printf("%.10f\n",(res.first==N)*100*exp(-res.second));
}
```

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

# Výsledková listina po 2. kole kategórie KSP-Z

| | Meno a priezvisko | Škola | Trieda | | 1 | 2 | 3 | 4 | 5 | Σ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Balog Matej | Gym. Grösslingová BA | 3 | 59 | 10 | 10 | 10 | 15 | 15 | 119 |
| 2 | Rohár Pavol | Gym. M. R. Štefánika, Košice | 4 | 59 | 10 | 10 | 9 | 15 | 14 | 117 |
| 3 | Horňák Marián | Gym. Párovská Nitra | 2 | 55 | 10 | 10 | 10 | 15 | 15 | 115 |
| 4 | Mariš Andrej | Gym. Piaristická Nitra | 2 | 55 | 10 | 10 | 9 | 15 | 15 | 114 |
| 5 | Brandys Jozef | Gym. Námestovo | 2 | 59 | 10 | 10 | 9 | 14 | | 102 |
| 6 | Pulmann Jan | Gym. Grösslingová BA | 3 | 55 | 10 | 10 | 6 | 9 | 7 | 97 |
| 7 | Babiak Jakub | Gym. Žiar nad Hronom | 3 | 45 | 10 | 10 | 7 | 14 | 10 | 96 |
| 8 | Greššák Jerguš | Gym. Mudroňova Prešov | 1 | 44 | 10 | 10 | 9 | 14 | 6 | 93 |
| 9 | Kováč Ondrej | SKŠ Nitra, Gym. sv. Cyrila a Metoda | 3 | 42 | 10 | 10 | 9 | 8 | 11 | 90 |
| 10 | Birkus Róbert | SPŠE Nové Zámky | 3 | 54 | 10 | 10 | 8 | 1 | 3 | 86 |
| 10 | Porubsky Martin | SKŠ Nitra, Gym. sv. Cyrila a Metoda | 3 | 42 | 10 | 10 | 4 | 15 | 5 | 86 |
| 10 | Sedláček Lukáš | Gym. Žiar nad Hronom | 3 | 50 | 10 | 10 | 4 | 7 | 5 | 86 |
| 13 | šinogl jakub | Gym. Žiar nad Hronom | 3 | 55 | 10 | 9 | 6 | | | 80 |
| 14 | Smolik Michal | Gym. Grösslingová BA | 1 | 50 | 10 | 8 | 4 | 7 | | 79 |
| 15 | Livora Tomáš | Gym Javorová Spiš. Nová Ves | 3 | 32 | 10 | 10 | 7 | 6 | 13 | 78 |
| 16 | Anderle Michal | Gym. Haličská Lučenec | 3 | 33 | 10 | 10 | 9 | 4 | 6 | 72 |
| 16 | Masár Juraj | Gym. Jura Hronca BA | 3 | 33 | 10 | 10 | 4 | 9 | 6 | 72 |
| 18 | Jurových Jakub | Gym. Okružná Zvolen | 3 | 36 | 10 | 10 | 8 | 5 | | 69 |
| 18 | Novella Tomáš | Gym. Alejová Košice | 4 | 46 | 10 | 10 | 3 | | | 69 |
| 18 | Čačko Marek | Gym. M. Galandu Turčianske Teplice | 4 | 42 | 10 | 10 | 7 | | | 69 |
| 21 | Jurenka Vladimír | Gym. Grösslingová BA | 4 | 34 | 10 | 10 | 7 | 3 | 4 | 68 |
| 22 | Sebechlebský Ján | Gym. Žiar nad Hronom | 3 | 40 | | | | 15 | 10 | 65 |
| 23 | Minárik Jakub | Gym. Jura Hronca BA | 3 | 33 | 10 | 7 | 5 | 8 | | 63 |
| 23 | Takács Gabriel | Gym. Fándlyho Šaľa | 2 | 33 | 10 | 10 | 9 | 1 | | 63 |
| 25 | Šuppa Marek | SKŠ Nitra, Gym. sv. Cyrila a Metoda | 1 | 35 | 10 | 9 | 6 | | | 60 |
| 26 | Petrucha Jaroslav | Gym. Metodova BA | 1 | 20 | 10 | 10 | 10 | 9 | | 59 |
| 27 | Pokorný Fridolín | Gym. Haličská Lučenec | 4 | 28 | 10 | 10 | 7 | 3 | | 58 |
| 28 | Toman Viktor | Gym. Golianova Nitra | 3 | 25 | 10 | 10 | 5 | 7 | | 57 |
| 29 | Krajčovič Matej | Gym. Jura Hronca BA | 1 | 14 | 10 | 10 | 7 | 9 | 5 | 55 |
| 30 | Durčák Dávid | Gym. Námestovo | 4 | 10 | 10 | 10 | 9 | 15 | | 54 |
| 31 | Kučera Martin | Gym. Golianova Nitra | 3 | 25 | | | | 12 | 15 | 52 |
| 32 | Šormanová Mária | Škola pre mim. nadané deti BA | 3 | 25 | 10 | 10 | 4 | | | 49 |
| 33 | Vargovčík Matej | Gym. Sabinov | 3 | 48 | | | | | | 48 |
| 34 | Plavák Dušan | Gym. Trstená | 3 | 16 | 10 | 10 | 7 | 4 | | 47 |
| 35 | Magdolen Matej | Gym. Golianova Nitra | 4 | 46 | | | | | | 46 |
| 36 | Kutaj Tomáš | Gym. Jura Hronca BA | 3 | 31 | 10 | 4 | | | | 45 |
| 37 | Klc Dano | GLN Tomášikova BA | 4 | 42 | | | | | | 42 |
| 38 | Rabatin Rastislav | Gym. Jura Hronca BA | 1 | 0 | 10 | 9 | 8 | 14 | | 41 |
| 39 | Kovár Martin | Gym. Jura Hronca BA | 3 | 15 | 10 | 9 | 4 | 2 | | 40 |
| 39 | Krumpal Rudo | Gym. Jura Hronca BA | 3 | 29 | 7 | 2 | 2 | | | 40 |
| 41 | Ille Ondrej | Gym. Jura Hronca BA | 3 | 10 | 10 | 10 | 4 | 5 | | 39 |
| 41 | Jankovič Radovan | Gymnázium Levice | 3 | 34 | | | | | 5 | 39 |
| 43 | Mutný Mojmír | Gym. Jura Hronca BA | 2 | 38 | | | | | | 38 |
| 44 | Marko Jozef | SPŠ Myjava | 3 | 20 | 6 | 7 | 3 | 1 | | 37 |
| 45 | Šeliga Adam | Súkromná SOŠ Humanus Via | 2 | 10 | 10 | 8 | 6 | 2 | | 36 |
| 46 | Varga Michal | SPŠE Nové Zámky | 3 | 0 | 10 | 10 | 6 | 7 | | 33 |
| 47 | Hornyák Zsolt | Gymnázium | 4 | 15 | 10 | 7 | | | | 32 |
| 47 | Šafin Jakub | Gym. P. Horova Michalovce | 1 | 4 | 9 | 9 | 4 | 3 | 3 | 32 |
| 49 | Součková Kamila | Ev. lýceum BA | 1 | 0 | 10 | 10 | 4 | 6 | | 30 |
| 50 | Kurtulík Matej | Gym. Námestovo | 2 | 0 | 10 | 8 | 6 | 5 | | 29 |

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

| | Meno a priezvisko | Škola | Trieda | | 1 | 2 | 3 | 4 | 5 | Σ |
|---|---|---|---|---|---|---|---|---|---|---|
| 51 | Strapko Jakub | SPŠ dopravná BA | 0 | 0 | 9 | 4 | 6 | 5 | 4 | 28 |
| 52 | Vozárová Zuzana | Gym. Jura Hronca BA | 2 | 23 | | 4 | | | | 27 |
| 53 | Ivanov Marián | Gym. Jura Hronca BA | 1 | 25 | | | | | | 25 |
| 53 | Pinter Martin | Gym. Jura Hronca BA | 3 | 25 | | | | | | 25 |
| 55 | Lami Jozef | Gym. Poštová Košice | 2 | 24 | | | | | | 24 |
| 56 | Bezek Matúš | Gym. Jura Hronca BA | 1 | 12 | 10 | | | | | 22 |
| 56 | Viskup Michal | Gym. Jura Hronca BA | 3 | 14 | 7 | 1 | | | | 22 |
| 58 | Pistrakova Alexandra | Gym. Poštová Košice | 2 | 8 | 10 | | | | | 18 |
| 59 | Chudjak Martin | SPŠ Martin | 4 | 17 | | | | | | 17 |
| 60 | Anderko Maroš | Gym. Konštantínova Prešov | 4 | 15 | | | | | | 15 |
| 61 | Krasnayová Dáša | Gym. Alejová Košice | 3 | 14 | | | | | | 14 |
| 62 | Matuš Juraj | Gym. Jura Hronca BA | 2 | 10 | | | | | | 10 |
| 62 | Václavík Lukáš | SSOŠ Via Humana | 2 | 10 | | | | | | 10 |
| 64 | Dang Quoc Trung | Gym. Jura Hronca BA | 2 | 9 | | | | | | 9 |
| 65 | Kucerová Bibiana | Gym. Alejová Košice | 3 | 8 | | | | | | 8 |
| 66 | Krotký Miroslav | Gym. Spišská Stará Ves | 2 | 7 | | | | | | 7 |
| 67 | Bobuľa Daniel | Spojená škola Kráľovnej pokoja Žilina | 2 | 6 | | | | | | 6 |
| 67 | Mesároš Tomáš | Gym. Jura Hronca BA | 3 | 6 | | | | | | 6 |
| 69 | Lieskovsky Adam | Gym. Jura Hronca BA | 3 | 5 | | | | | | 5 |
| 69 | Orenič Jozef | Gym. Slovenská Bardejov | 4 | 5 | | | | | | 5 |
| 71 | Kollár Dan | Gym. Grösslingová BA | 2 | 3 | | | | | | 3 |



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

# Výsledková listina po 2. kole kategórie KSP-O

| | Meno a priezvisko | Škola | Trieda | | 4 | 5 | 6 | 7 | 8 | Σ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Hudec Tobiáš | Gym. Partizánske | 4 | 91 | 12 | 15 | 20 | 19 | 25 | 182 |
| 2 | Belan Tomáš | Škola pre mim. nadané deti BA | 4 | 83 | 14 | 14 | 20 | 20 | 25 | 176 |
| 3 | Rohár Pavol | Gym. M. R. Štefánika, Košice | 4 | 67 | 15 | 14 | 3 | 18 | 9 | 126 |
| 4 | Mariš Andrej | Gym. Piaristická Nitra | 2 | 48 | 15 | 15 | 8 | 12 | 1 | 99 |
| 5 | Korbaš Rafael | Gym. Hronská BA | 3 | 41 | 14 | 13 | 8 | 20 | 1 | 97 |
| 5 | Sebechlebský Ján | Gym. Žiar nad Hronom | 3 | 49 | 15 | 10 | 8 | 12 | 3 | 97 |
| 7 | Kekely Michal | Gym. Varšavská Žilina - Vlčince | 3 | 39 | 15 | 11 | | 12 | | 77 |
| 8 | Pitoňák Martin | Gym. Tajovského B. Bystrica | 4 | 44 | 13 | 15 | | | 4 | 76 |
| 9 | Kučera Martin | Gym. Golianova Nitra | 3 | 28 | 12 | 15 | 7 | 12 | | 74 |
| 10 | Dresslerova Anna | Gym. Jura Hronca BA | 3 | 30 | 15 | 13 | 15 | | | 73 |
| 10 | Mrocková Mária | Gym. Jura Hronca BA | 3 | 35 | 15 | 5 | | 18 | | 73 |
| 12 | Varga Mátyás | Gym. H. Selyeho Komárno | 3 | 38 | 2 | 9 | 8 | 12 | 1 | 70 |
| 13 | Hozza Jan | Gym. Jura Hronca BA | 3 | 50 | 15 | | | | | 65 |
| 14 | Cuc Bruno | Gym. Grösslingová BA | 4 | 33 | 7 | 10 | | 14 | 0 | 64 |
| 15 | Balog Matej | Gym. Grösslingová BA | 3 | 30 | 15 | 15 | | | | 60 |
| 16 | Horňák Marián | Gym. Párovská Nitra | 2 | 28 | 15 | 15 | | | | 58 |
| 17 | Greššák Jerguš | Gym. Mudroňova Prešov | 1 | 21 | 14 | 6 | | 12 | | 53 |
| 18 | Miklovič Tomáš | Gym. Nové Zámky | 4 | 27 | 9 | | | 12 | | 48 |
| 18 | Večerík Matej | Škola pre mim. nadané deti BA | 3 | 28 | 14 | 6 | | | | 48 |
| 20 | Pulmann Jan | Gym. Grösslingová BA | 3 | 30 | 9 | 7 | | | | 46 |
| 21 | Gandžala Jozef | Gym. Tajovského B. Bystrica | 4 | 30 | 15 | | | | | 45 |
| 22 | Brandys Jozef | Gym. Námestovo | 2 | 30 | 14 | | | | | 44 |
| 22 | Kováč Ondrej | SKŠ Nitra, Gym. sv. Cyrila a Metoda | 3 | 25 | 8 | 11 | | | | 44 |
| 24 | Babiak Jakub | Gym. Žiar nad Hronom | 3 | 16 | 14 | 10 | | | | 40 |
| 25 | Ziman Michal | Gym. Haličská Lučenec | 4 | 17 | 9 | | | 10 | | 36 |
| 26 | Porubsky Martin | SKŠ Nitra, Gym. sv. Cyrila a Metoda | 3 | 14 | 15 | 5 | | | | 34 |
| 27 | Sedláček Lukáš | Gym. Žiar nad Hronom | 3 | 21 | 7 | 5 | | | | 33 |
| 28 | Birkus Róbert | SPŠE Nové Zámky | 3 | 28 | 1 | 3 | | | | 32 |
| 29 | Hruby Tomas | Gym. Jura Hronca BA | 3 | 7 | 15 | 9 | | | | 31 |
| 29 | Minárik Jakub | Gym. Jura Hronca BA | 3 | 23 | 8 | | | 0 | | 31 |
| 29 | Smolik Michal | Gym. Grösslingová BA | 1 | 24 | 7 | | | | | 31 |
| 32 | Habovštiak Martin | Gym. Tvrdošín | 4 | 30 | | | | | | 30 |
| 33 | Vargovčík Matej | Gym. Sabinov | 3 | 28 | | | | | | 28 |
| 34 | Anderle Michal | Gym. Haličská Lučenec | 3 | 17 | 4 | 6 | | | | 27 |
| 34 | šinogl jakub | Gym. Žiar nad Hronom | 3 | 27 | | | | | | 27 |
| 36 | Jankovič Radovan | Gymnázium Levice | 3 | 18 | | 5 | | | 3 | 26 |
| 36 | Masár Juraj | Gym. Jura Hronca BA | 3 | 11 | 9 | 6 | | | | 26 |
| 38 | Livora Tomáš | Gym Javorová Spiš. Nová Ves | 3 | 6 | 6 | 13 | | | | 25 |
| 39 | Krumpal Rudo | Gym. Jura Hronca BA | 3 | 23 | | | | | | 23 |
| 40 | Tóthová Tatiana | Gym. Jura Hronca BA | 4 | 0 | 9 | 1 | | 11 | | 21 |
| 41 | Kutaj Tomáš | Gym. Jura Hronca BA | 3 | 20 | | | | | | 20 |
| 42 | Magdolen Matej | Gym. Golianova Nitra | 4 | 19 | | | | | | 19 |
| 42 | Novella Tomáš | Gym. Alejová Košice | 4 | 19 | | | | | | 19 |
| 44 | Pokorný Fridolín | Gym. Haličská Lučenec | 4 | 15 | 3 | | | | | 18 |
| 45 | Klc Dano | GLN Tomášikova BA | 4 | 17 | | | | | | 17 |
| 45 | Kovár Martin | Gym. Jura Hronca BA | 3 | 15 | 2 | | | | | 17 |
| 45 | Strapko Martin | Gym. Jura Hronca BA | 3 | 0 | 4 | 6 | 7 | | | 17 |
| 45 | Čačko Marek | Gym. M. Galandu Turčianske Teplice | 4 | 17 | | | | | | 17 |
| 49 | Jurenka Vladimír | Gym. Grösslingová BA | 4 | 9 | 3 | 4 | | | | 16 |
| 49 | Toman Viktor | Gym. Golianova Nitra | 3 | 9 | 7 | | | | | 16 |

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

| | Meno a priezvisko | Škola | Trieda | | 4 | 5 | 6 | 7 | 8 | Σ |
|---|---|---|---|---|---|---|---|---|---|---|
| 49 | Vozárová Zuzana | Gym. Jura Hronca BA | 2 | 16 | | | | | | 16 |
| 52 | Durčák Dávid | Gym. Námestovo | 4 | 0 | 15 | | | | | 15 |
| 52 | Ivanov Marián | Gym. Jura Hronca BA | 1 | 15 | | | | | | 15 |
| 52 | Jurových Jakub | Gym. Okružná Zvolen | 3 | 10 | 5 | | | | | 15 |
| 52 | Mutný Mojmír | Gym. Jura Hronca BA | 2 | 15 | | | | | | 15 |
| 56 | Krajčovič Matej | Gym. Jura Hronca BA | 1 | 0 | 9 | 5 | | | | 14 |
| 56 | Rabatin Rastislav | Gym. Jura Hronca BA | 1 | 0 | 14 | | | | | 14 |
| 56 | Spano Marek | Gym. Jura Hronca BA | 3 | 14 | | | | | | 14 |
| 56 | Šuppa Marek | SKŠ Nitra, Gym. sv. Cyrila a Metoda | 1 | 14 | | | | | | 14 |
| 60 | Ille Ondrej | Gym. Jura Hronca BA | 3 | 8 | 5 | | | | | 13 |
| 61 | Marko Jozef | SPŠ Myjava | 3 | 11 | 1 | | | | | 12 |
| 62 | Petrucha Jaroslav | Gym. Metodova BA | 1 | 0 | 9 | | | | | 9 |
| 62 | Strapko Jakub | SPŠ dopravná BA | 0 | 0 | 5 | 4 | | | | 9 |
| 62 | Takács Gabriel | Gym. Fándlyho Šaľa | 2 | 8 | 1 | | | | | 9 |
| 65 | Holas Juraj | Gym. Jura Hronca BA | 3 | 8 | | | | | | 8 |
| 65 | Viskup Michal | Gym. Jura Hronca BA | 3 | 8 | | | | | | 8 |
| 67 | Hornyák Zsolt | Gymnázium | 4 | 7 | | | | | | 7 |
| 67 | Plavák Dušan | Gym. Trstená | 3 | 3 | 4 | | | | | 7 |
| 67 | Varga Michal | SPŠE Nové Zámky | 3 | 0 | 7 | | | | | 7 |
| 70 | Lukča Pavol | Gym. sv. Františka z Assisi V.n. Topľou | 4 | 0 | | | | | 6 | 6 |
| 70 | Součková Kamila | Ev. lýceum BA | 1 | 0 | 6 | | | | | 6 |
| 70 | Šafin Jakub | Gym. P. Horova Michalovce | 1 | 0 | 3 | 3 | | | | 6 |
| 73 | Bezek Matúš | Gym. Jura Hronca BA | 1 | 5 | | | | | | 5 |
| 73 | Kurtulík Matej | Gym. Námestovo | 2 | 0 | 5 | | | | | 5 |
| 75 | Dang Quoc Trung | Gym. Jura Hronca BA | 2 | 4 | | | | | | 4 |
| 75 | Pinter Martin | Gym. Jura Hronca BA | 3 | 4 | | | | | | 4 |
| 77 | Šeliga Adam | Súkromná SOŠ Humanus Via | 2 | 0 | 2 | | | | | 2 |
| 78 | Hruška Eugen | Gym. Hlohovec | 2 | 0 | | | | | 1 | 1 |
| 78 | Krotký Miroslav | Gym. Spišská Stará Ves | 2 | 1 | | | | | | 1 |
| 78 | Kucerová Bibiana | Gym. Alejová Košice | 3 | 1 | | | | | | 1 |
| 78 | Lieskovsky Adam | Gym. Jura Hronca BA | 3 | 1 | | | | | | 1 |
| 78 | Matuš Juraj | Gym. Jura Hronca BA | 2 | 1 | | | | | | 1 |
| 83 | Kollár Dan | Gym. Grösslingová BA | 2 | 0 | | | | | | 0 |
| 83 | Nánási Michal | Gym. Jura Hronca BA | 9 | 0 | | | | | 0 | 0 |
| 83 | Orenič Jozef | Gym. Slovenská Bardejov | 4 | 0 | | | | | | 0 |
| 83 | Václavík Lukáš | SSOŠ Via Humana | 2 | 0 | | | | | | 0 |
| 83 | Šormanová Mária | Škola pre mim. nadané deti BA | 3 | 0 | | | | | | 0 |

**http://www.ksp.sk/ksp2.0**

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

# Výsledková listina po 2. kole kategórie KSP-T

| | Meno a priezvisko | Škola | Trieda | | 8 | 9 | 10 | Σ |
|---|---|---|---|---|---|---|---|---|
| 1 | Hudec Tobiáš | Gym. Partizánske | 4 | 25 | 25 | 22 | 20 | 92 |
| 2 | Belan Tomáš | Škola pre mim. nadané deti BA | 4 | 34 | 25 | 17 | 9 | 85 |
| 3 | Rohár Pavol | Gym. M. R. Štefánika, Košice | 4 | 8 | 9 | 8 | 4 | 29 |
| 4 | Jankovič Radovan | Gymnázium Levice | 3 | 24 | 3 | | | 27 |
| 5 | Mariš Andrej | Gym. Piaristická Nitra | 2 | 0 | 1 | 12 | 6 | 19 |
| 6 | Lukča Pavol | Gym. sv. Františka z Assisi V.n. Topľou | 4 | 0 | 6 | | | 6 |
| 6 | Pitoňák Martin | Gym. Tajovského B. Bystrica | 4 | 2 | 4 | | | 6 |
| 8 | Novella Tomáš | Gym. Alejová Košice | 4 | 5 | | | | 5 |
| 9 | Sebechlebský Ján | Gym. Žiar nad Hronom | 3 | 0 | 3 | | | 3 |
| 10 | Varga Mátyás | Gym. H. Selyeho Komárno | 3 | 1 | 1 | | | 2 |
| 11 | Cuc Bruno | Gym. Grösslingová BA | 4 | 1 | 0 | | | 1 |
| 11 | Hruška Eugen | Gym. Hlohovec | 2 | 0 | 1 | | | 1 |
| 11 | Ille Ondrej | Gym. Jura Hronca BA | 3 | 0 | | 1 | | 1 |
| 11 | Kekely Michal | Gym. Varšavská Žilina - Vlčince | 3 | 1 | | | | 1 |
| 11 | Korbaš Rafael | Gym. Hronská BA | 3 | 0 | 1 | | | 1 |
| 11 | Livora Tomáš | Gym Javorová Spiš. Nová Ves | 3 | 0 | | 1 | | 1 |
| 11 | Minárik Jakub | Gym. Jura Hronca BA | 3 | 0 | | 1 | | 1 |
| 11 | Mrocková Mária | Gym. Jura Hronca BA | 3 | 1 | | | | 1 |
| 11 | Vozárová Zuzana | Gym. Jura Hronca BA | 2 | 1 | | | | 1 |
| 20 | Dang Quoc Trung | Gym. Jura Hronca BA | 2 | 0 | | | | 0 |
| 20 | Lampanen Mika | Gym. Pankúchova Bratislava | 3 | 0 | | | | 0 |
| 20 | Nánási Michal | Gym. Jura Hronca BA | 9 | 0 | 0 | | | 0 |
| 20 | Večerík Matej | Škola pre mim. nadané deti BA | 3 | 0 | | 0 | | 0 |



Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09