Korešpondenn seminr z programovania
XVIII. ronk, 2000/2001
Katedra vyuovania informatiky MFF UK,
Mlynsk Dolina, 842 48 Bratislava

KSP finanne podporuje nadcia Open Society Fund Bratislava a
IUVENTA – zariadenie pre voľn as det, mldeže a dospelch

Vzorové riešenia 1. kola

Milí riešitelia,

pomaly sa zozimieva. Medvede si chystajú zásoby na zimu. Mali by ste sa snažiť aj vy, lebo ako hovorí popredný slovenský meteorológ Iľko, tohtoročná zima bude dlhá a krutá. Nakúpte si zásoby už teraz, aby vás nezaskočila.

Držte si nohy v teple a pite teplé mlieko, aby ste boli zdraví.

KSPci

Na úvod by sme vám radi povedali (rovnako ako minulý a predminulý rok) niekoľko slov o tom, ako si predstavujeme ideálne riešenie.

Nechceme, aby ste sa zbytočne hrali so vstupom a výstupom. Samozrejme, zakazovať vám to nebudeme, ale bodíky navyše za to nedostanete a ešte môžete znechutiť opravovateľa, ktorý musí čítať množstvo kódu nesúvisiaceho s riešením samotnej úlohy. Vo svojom riešení sa nemusíte zaoberať kontrolovaním správnosti vstupných údajov. Skôr by sme ocenili, keby ste inicializovli (napr. vynulovali) všetky premenné (a najmä polia), ktoré v programe použijete, aby sme videli, že ste na ne nezabudli.

Pod pojmom **popis algoritmu** *nerozumieme* preloženie programu z Pascalu, prípadne C-čka do slovenčiny. Popis by mal zhruba obsahovať: **Popis myšlienky**, na ktorej je založený algoritmus (bez detailov ako sú názvy premenných, procedúr...). **Popis dátových štruktúr**. Čo si vlastne chceme počas výpočtu pamätať a aké štruktúry sú na to vhodné. **Popis algoritmu**. Tu môžeme využívať už popísané dátové štruktúry a spomenúť dôležité procedúry a funkcie, ako aj najdôležitejšie premenné. **Zdôvodnenie správnosti**. Nemusí to byť formálne presný ani detailný dôkaz, avšak mal by obsahovať argumenty zdôvodňujúce každý dôležitý aspekt programu, ktorý nie je na prvý pohľad zrejmý. Sem zaradíme aj dôkaz konečnosti v tých prípadoch, keď nie je zrejmé, že sa program (resp. niektorá operácia, ktorú vykonáva) nezacyklí. Na koniec príde **odhad** časovej a pamäťovej zložitosti programu.

Takáto štruktúra popisu nie je univerzálna. Niektoré časti možno vynechať, niektoré zlúčiť, prípadne nejaké pridať. Poradie by však malo byť zhruba zachované. Popis by sa v žiadnom prípade nemal detailne venovať spracovávaniu vstupu, ani výpisu výstupu.

Niekoľko slov o odhade zložitosti. Pod odhadom časovej zložitosti rozumieme odhad času, ktorý bude program trvať, v závislosti od vstupných premenných. Tento čas je priamo úmerný počtu elementárnych operácií, ktoré program vykonáva – priradenie jednoduchej premennej, porovnanie jednoduchých premenných, aritmetické operácie, atď. Väčšinou sa zaujímame o to, ako dlho beží program v priemernom, alebo najhoršom prípade (čiže si všímame priemerný, alebo maximálny čas vykonávania programu). Analogicky odhad pamäťovej zložitosti je odhad veľkosti použitej pamäti v bajtoch v závislosti od vstupných premenných.

Väčšinou nás nezaujímajú presné hodnoty (niekto má rýchlejší počítač, na niektorých počítačoch má integer 2 a na iných 4 bajty a pod.), preto používame tzv. O-notáciu. Hovoríme, že funkcia $f(n)$ (napr. čas behu programu v milisekundách v závislosti od veľkosti vstupu n) je $O(g(n))$, ak existuje taká konštanta c, že pre všetky dostatočne veľké n ($n > n_0$) je $f(n) \leq c \cdot g(n)$.

Napríklad, ak program pre vstup veľkosti n nebeží nikdy viac ako $T(n) = 0.5n^3 + 5n \log n + 83$ mikrosekúnd, môžeme povedať, že má časovú zložitosť $O(n^3)$. Ak program potrebuje najviac $M(n) = 3n \log_2 n + 150n + 15$ bajtov pamäti, povieme, že má pamäťovú zložitosť $O(n \log n)$. [1]

Viacero príkladov odhadu časovej i pamäťovej zložitosti nájdete, ak si pozorne prečítate vzoráky.

A ešte jedno upozornenie pre **vás** – riešiteľov: uvedením nepostačujúceho popisu prípadne jeho vynechaním riskujete, že opravovateľ vaše riešenie nepochopí a dostanete napr. 0 (slovom **nula**) bodov.

1. O vybíjanej

opravoval Dávidko
(max. 15 bodov)

Vyskytlo sa vskutku pestré spektrum riešení:

- Nesprávne riešenia boli väčšinou založené na myšlienke, že družstvo s väčším počtom výhier musí byť nutne v poradí pred družstvom s menším počtom. Takéto a iné nefunčnosti získali maximálne 2 body.
- Backtracky s časovou zložitosťou $O(N!)$ a horšou získali maximálne po 5 bodov.
- Veľká bola aj druhá kopa riešení s časovou zložitosťou $O(N^2)$. Získala maximálne po 9 bodov. Išlo o postupné pridávanie do utriedenej postupnosti alebo bubblesorty zväčša bez dôkazu správnosti.
- 4 riešenia s časovou zložitosťou $O(N \log N)$ maximálne 15 bodov. Jedno riešenie s binárnymi vyvažovanými stromami. A tri riešenia à la Quicksort[2].

Po bode som strhal za chýbajúci alebo zlý popis, odhad časovej zložitosti alebo dôkaz správnosti, prípadne za evidentné chyby v programe.

A teraz už k vzorovému riešeniu.

Veta: *Pre ľubovoľné výsledky turnaja existuje poradie $d_1, d_2, \ldots d_n$ družstiev také, že družstvo d_i vyhralo nad družstvom d_{i+1} pre $1 \leq i < n$.*

Dôkaz: Tvrdenie vety dokážeme matematickou indukciou vzhľadom na n. Triviálne platí, že jedno alebo dve družstvá vieme usporiadať, ako sa požaduje. Predpokladajme teda, že vieme usporiadať ľubovoľný turnaj n družstiev. Ukážeme ako usporiadať ľubovoľný turnaj $n+1$ družstiev. Majme teda ľubovoľný turnaj $n+1$ družstiev. Vezmime prvých n družstiev tohto turnaja. Tie vieme podľa indukčného predpokladu požadovane usporiadať. Nech $d_1, d_2, \ldots d_n$ je ich poradie. Našou úlohou bude ukázať, že do tohto poradia sa dá „strčiť“ družstvo d_{n+1}.

Ak družstvo d_{n+1} vyhralo nad d_1, máme požadované poradie $d_{n+1}, d_1, d_2, \ldots d_n$. Podobne ak družstvo d_{n+1} prehralo s d_n, máme požadované poradie $d_1, d_2, \ldots d_n, d_{n+1}$. Inak nájdime najmenšie také i, že družstvo d_{n+1} vyhralo nad d_i[3]. Potom muselo družstvo d_{n+1} prehrať s družstvom d_{i-1}. Dostávame, tak opäť požadované poradie $d_1, d_2, \ldots d_{i-1}, d_{n+1}, d_i, \ldots d_n$.

Dôkaz prechádzajúcej vety je konštruktívny t.j. dáva nám návod, ako poradie družstiev hľadať. Žial, priama aplikácia tohto postupu vedie k algoritmu, ktorého časová zložitosť je $O(n^2)$. Vzorový algoritmus bude však prefíkanejší a jeho časová zložitosť bude menšia – $O(n \log n)$.

Algoritmus je založený na metóde rozdeľuj a panuj[4]. Postup bude zhruba nasledovný: Máme turnaj n družstiev. Ak je $n = 1$ sme hotoví. Inak vezmeme prvých $p = \lfloor n/2 \rfloor$ družstiev,

[1] Poznamenávame, že ak je nejaká funkcia $O(n^2)$, tak je aj $O(n^3)$ atď. Vždy sa však snažíme nájsť čo najmenšie horné ohraničenie, t.j. funkciu, ktorá čo najpomalšie rastie.

[2] Vyberieme náhodne jedno družstvo a zvyšné rozdelíme na tie, čo s ním prehrali, a na tie, čo s ním vyhrali. Na tieto dve časti aplikujeme rekurzívne ten istý postup.

[3] Zrejme $i > 1$

[4] Veľmi podobný Mergesortu

tieto rekurzívne usporiadame. Podobne to spravíme pre zvyšných $q = n - p$ družstiev. Máme teraz usporiadanie $d_1, d_2, \ldots d_p$ prvej a usporiadnie $e_1, e_2, \ldots e_q$ druhej polovice družstiev. Našou úlohou bude poprestrkávať poradia medzi seba.

Postupovať budeme takto: Budeme medzi poradie družstiev $d_1, d_2 \ldots d_n$ „strkať“ jednotlivé družstvá e_i pre $1 \leq i \leq q$. Družstvo e_1 vieme podľa dôkazu vety niekde vopchať. Máme nejaké poradie $d_1, \ldots e_1, d_i, \ldots d_p$, kde $1 \leq i \leq p + 1$. Zoberme teraz družstvo e_2. To prehralo s e_1. Nám už stačí nájsť prvé družstvo d_j v tomto poradí od e_1 napravo, s ktorým e_2 vyhralo. Tam môžme e_2 „strčiť“, pretože s družstvom tesne pred d_j družstvo e_2 prehralo. Dostaneme tak poradie $d_1, \ldots e_1, d_i, \ldots e_2, d_j, \ldots d_p$, kde $i \leq j \leq p + 1$. Podobne e_3 vieme strčiť napravo od e_2 atď.

Takto vieme na jedno prejdenie poradia $d_1, d_2, \ldots d_p$ doňho postrkať družstvá $e_i, 1 \leq i \leq q$. Časová zložitosť takéhoto poprestrkávania bude preto len $O(n)$. Ostáva nám zistiť, aká bude celková časová zložitosť algoritmu t.j. zahrňujúc aj rekurziu.

Začnime teda jednoduchým pozorovaním, že pri každom rekurzívnom volaní sa počet družstiev v turnaji spoloviční. Hĺbka rekurzie bude preto najviac $O(\log_2 n)$.

Vezmime si teraz všetky rekurzívne volania v jednej pevnej hĺbke. Nech počty družstiev v jednotlivých turnajoch na tejto hĺbke vetvenia sú $n_1, n_2, \ldots n_k$. Súčet týchto počtov musí byť nutne $n = n_1 + n_2 + \ldots + n_k$. Keďže časové zložitosťi poprestrkávaní sú $O(n_1), O(n_2), \ldots O(n_k)$ bude ich celková časová zložitosť $O(n)$.

Hĺbok je v rekurzii najviac $O(\log_2 n)$. Dostávame tak celkovú časovú zložitosť $O(n \log n)$.

Listing programu:

```
{ KSP XVIII-1-1 O vybijanej }
Program O_Vybijanej;

const MaxN = 100; { maximalna hodnota N}

type PtrPoradie = ^ TPoradie;    { spajany zoznam druzstiev t.j. poradie }
     TPoradie = record
                  cislo:integer;    { cislo druzsvta }
                  next:PtrPoradie;  { pointer na dalsie druzstvo v poradi }
                end;

var Vys:array[1..MaxN,1..MaxN] of integer;
    N,i,j:integer;

    hlava,          { pointer na zaciatok spajaneho zoznamu }
    p,q:PtrPoradie;


{ Poradie(l,r) vrati spravne poradie druzstiev l..r }
Function Poradie(lavy,pravy:integer):PtrPoradie;
var hlava,p,q,r,s:PtrPoradie;
    stred:integer;
begin

  { ak je turnaj len s jednym druzstvom }
  if lavy=pravy then
  begin
    new(hlava);
    hlava^.cislo:=lavy;
```

```
    hlava^.next:=NIL;
    Poradie:=hlava;
    exit;
  end;

  { rozdelime problem na dva podproblemy }
  { dostaneme dva spajane zoznamy }
  stred:=(lavy+pravy) div 2;
  hlava:=Poradie(lavy,stred);   { poradie prvej  polovice druzstiev }
  p:=Poradie(stred+1,pravy);    { poradie druhej polovice druzstiev }

  { zoznamy zlucime }
  q:=p;
  r:=hlava;
  s:=NIL;

  repeat
    p:=q;
    q:=q^.next;

    { hladame, kam zaradit druzstvo p t.j. e[i] }
    while (r<>NIL) and (Vys[p^.cislo,r^.cislo]=0) do
    begin
     s:=r;
     r:=r^.next;
    end;

    p^.next:=r;

    if s=NIL then hlava:=p
             else s^.next:=p;

    s:=p;
  until q=NIL;

  Poradie:=hlava;   { vratime poradie druzstiev l..r }
end;

Begin
     assign(input,'vybijana.in');
     assign(output,'vybijana.out');
     reset(input);
     rewrite(output);

     { nacitame - POZOR !!! toto sa nerata do zlozitosti algoritmu }
     readln(N);
     for i:=1 to N do
     for j:=1 to N do read(Vys[i,j]);

     { zavolame hlavnu rekurzivnu proceduru }
     hlava:=Poradie(1,N);
```

```
    { vypiseme finalne poradie }
    p:=hlava;
    while p<>NIL do
    begin
      write(p^.cislo,' ');
      q:=p^.next;
      dispose(p); {znicime zaciatok zoznamu a posunieme sa dalej}
      p:=q;
    end;
End.
```

2. O prepúšťaní v TSSL

opravoval Goober
(max. 15 bodov)

Tento príklad patril medzi tie ľahšie, o čom svedčí aj vysoký počet a úspešnosť vašich riešení. Vaše riešenia by sa dali rozdeliť do troch skupín, podľa použitého algoritmu a správnosti:

- Riešenia nesprávne, za ktoré bolo možné získať nanajvýš **3 body** Je asi zrejmé, že tieto riešenia nemuseli obsahovať dôkaz.
- Správne riešenia s horšou ako kvadratickou zložitosťou. Použitá idea bola obyčajne natoľko triviálna, že nepotrebovala dôkaz.
- Správne riešenia založené na nie celkom triviálnej myšlienke, ktorú bolo nutné viac či menej dokázať. Tieto riešenia boli zväčša založené na myšlienke MaxMax alebo MaxMin (ktoré budú popísané neskôr) a podľa implementácie mohli dostať **9** (za $O(N^2)$) až **13** (za O(N)) **bodov**. Pamäťovú náročnosť mali tieto riešenia O(N).

Špeciálne bonusy sa udeľovali za dôkaz (0–2 body) a za chýbajúci popis (-1 bod). Niektoré riešenia navyše pracovali nesprávne pre isté špeciálne prípady (najčastejšie mali problémy s $N = 1$). Keď išlo len o chybu pri výpise (tj. program našiel korektné riešenie, ale napríklad pri výpise dal jednému vedcovi ten istý projekt dvakrát), pridávala sa prémia -1 bod. Ale dosť už bolo bodovania, poďme sa pozrieť na riešenia.

Väčšina správnych riešení bola založená na metóde greedy[5], tj. každému vedcovi dáme čo najviac projektov a dúfame, že to bude optimálne. Metóda greedy je však veľmi záludná, a preto je dôkaz nevyhnutnou súčasťou riešenia. A ako to prideľovanie projektov realizovať? Bez ujmy na všeobecnosti môžeme dať najťažší projekt vedcovi číslo 1. Keďže pracujeme metódou greedy, skúsime mu dať nejaký ďalší projekt (samozrejme nesmieme prekročiť maximálnu pracovnú dobu a porušiť tak stanovy TSSL). Riešenia MaxMax hľadali najťažší povolený projekt a riešenia MaxMin naopak projekt najľahší (zanedlho zdôvodníme, prečo sú obidva postupy správne). Ak vhodný projekt nenájdeme, vedcovi zostane len ten jeden (najťažší). Napokon odstránime projekty (alebo projekt), ktoré sme už pridelili, čím dostaneme redukovanú úlohu, ktorú vyriešime rovnako.

Lema 1: Nech A je množina ešte nerozdelených projektov a B je minimálny počet vedcov, ktorí zvládnu všetky projekty z množiny A. Ak A' vznikne z A odstránením ľubovoľného počtu projektov, tak pre počet vedcov B', ktorí zvládnu všetky projekty z A', platí $B' \leq B$. Dôkaz: Dúfam, že toto tvrdenie je zrejmé, pretože B vedcov stačilo na všetky projekty z A, a teda stačia aj na všetky projekty z A' (prinajlepšom sa niektorý vedec môže stať nepotrebným, ale určite nám nebude žiadny chýbať).

Ak teda vedcovi 1 dáme najťažší projekt (niektorý vedec tento projekt aj tak musí dostať, preto ho môžeme dať rovno prvému vedcovi), tak nám zostane množina A projektov. Ak mu

[5] greedy = chamtivý, pažravý

dáme ešte jeden projekt (z množiny A), tak si tým určite nepohoršíme (viď Lema 1). Týmto sa dokázala oprávnenosť metódy greedy.

Lema 2: Nazvime U-rozdelením také rozdelenie projektov, v ktorom vedec 1 dostal ako druhý projekt U. Potom pre všetky X (samozrejme, ak vedec 1 môže dostať projekt X) je X-rozdelenie rovnako dobré. Dôkaz: Predpokladajme, že existujú (aspoň) dva rôzne projekty I,J, ktoré možno pridať vedcovi 1 ako druhý projekt. Nepochybne platí, že projekty I aj J možno skombinovať s ľubovoľným projektom (lebo ich bolo možné skombinovať aj s najnáročnejším). Vezmime si teraz I-rozdelenie. Niektorý vedec v tomto rozdelení musí mať aj projekt J a ako sme už dokázali, tieto projekty môžeme vzájomne vymeniť, čím dostaneme rovnako dobré J-rozdelenie. Teda I-rozdelenie a J-rozdelenie je rovnako dobré pre každé I,J.

Dôsledok: Keďže metóda greedy je oprávnená (Lema 1) a všetky výbery druhého projektu pre vedca 1 sú rovnocenné (Lema 2), nájdu vždy algoritmy MaxMin aj MaxMax optimálne riešenie.

Na záver pár slov k implemetácii: Máme dva ukazovatele (max, min), ktoré ukazujú na najnáročnejší, resp. najmenej náročný projekt. Podľa popísaného postupu ich približujeme a priebežne vypisujeme rozdelenie projektov medzi vedcov.

Listing programu:

```
program O_Prepustani_V_TSSL;
const MAXN=1000;
var
 i,j,n,m:integer;
 p:array[1..MAXN] of integer;
begin
 readln(n,m);
 for i:=1 to n do read(p[i]);
 i:=1; j:=n;
 while (i<j) do
  begin
   write('Vedec #',i,': ',i);
   if (p[i]+p[j]<=m) then begin writeln(', ',j); dec(j); end
   else writeln;
   inc(i);
  end;
 if (i=j) then writeln('Vedec #',i,': ',i);
 writeln('Ostatnych (',n-i,') vedcov mozno prepustit.');
end.
```

3. O najkratšom tuneli

opravoval Rišo
(max. 15 bodov)

Hlavná myšlienka tohoto príkladu bola pomerne jednoduchá a väčšina z vás na ňu prišla. Bolo treba zistiť vzdialenosť medzi každou úsečkou prvého mnohouholníka a každou úsečkou druhého mnohouholníka. Mnohí z vás si však neuvedomili, čo to znamená, že mnohouholníky nie sú disjunktné. Neošetrenie prípadu, keď jeden mnohouholník celý leží v druhom, bolo vašou najčastejšou chybou. Bodovanie bolo nasledovné:

- **15 bodov** – časová zložitosť $O(mn)$, správne riešenie bez závažných chýb
- **12 bodov** – časová zložitosť $O(mn)$, neošetrený prípad, keď nie sú mnohouholníky disjunktné

- **8 bodov** – časová zložitosť $O(mn)$, závažné chyby vo výpočte vzdialeností medzi úsečkami (napr. za počítanie vzdialeností medzi priamkami namiesto vzdialeností medzi úsečkami)
- **7 bodov** – neexaktné riešenia, ktoré sa snažili hľadať vzdialenosť úsečiek iba približne (časová zložitosť závisí od požadovanej presnosti)
- **6 bodov** – časová zložitosť $O(mn)$, závažné chyby vo výpočte vzdialeností medzi úsečkami a neošetrený prípad, keď nie sú mnohouholníky disjunktné
- **4 body** – myšlienka skúšania všetkých dvojíc úsečiek, bez špecifikovania ich zisťovania vzdialenosti

Pamäťová zložitosť bola u všetkých riešení $O(m + n)$, hoci vaše odhady neboli vždy správne. Za chýbajúci popis, nesprávny alebo zlý odhad zložitosti a zlý subjektívny dojem ste mohli stratiť 1-2 body.

A teraz vzorové riešenie: Najskôr treba overiť, či sú mnohouholníky disjunktné. Mnohouholníky sú disjunktné práve vtedy, ak sa žiaden vrchol jedného mnohouholníka nenachádza v druhom a zároveň sa žiadna strana prvého mnohouholníka nepretína s nejakou stranou druhého mnohouholníka.[6]

V prípade, že mnohouholníky nie sú disjunktné, existuje bod, ktorý im obom patrí. Tento bod zároveň tvorí tunel s nulovou dĺžkou. Ak mnohouholníky sú disjunktné, najkratší tunel bude mať iste nenulovú dĺžku. Teraz sa budeme zaoberať situáciou, keď mnohouholníky nie sú disjunktné.

Jeden koncový bod najkratšieho tunela bude ležať na obvode jedného mnohouholníka a druhý na obvode druhého mnohouholníka. Keby tomu tak nebolo (t.j. oba krajné body tunela ležia vo vnútri nejakého mnohouholníka), potom tunel určite pretne obvody oboch mnohouholníkov. Spojnica týchto priesečníkov však tvorí kratší tunel, čo je spor.

Aspoň jeden z krajných bodov najkratšieho tunela bude určite vrcholom nejakého mnohouholníka. Predpokladajme, že by to tak nebolo. Predpokladajme, že strany mnohouholníkov, medzi ktorými vedie tunel (označme ich AB a CD), sú rôznobežné. Potom môžme tunel posunúť smerom k vrcholu uhlu, ktorý zvierajú priamky AB a CD. Urobíme to tak, že zachováme jeho smer, jeho krajné body budú stále ležať na úsečkách AB a CD a aspoň jeden jeho krajný bod bude totožný s nejakým vrcholom niektorého mnohouholníka (bude to niektorý z bodov A, B, C, D). Takýmto posunutím sa však dĺžka tunela zmenší, čo je spor. Ak by boli strany AB a CD rovnobežné, tunel môžme podobne posunúť (tentoraz ľubovoľným smerom), pričom jeho dĺžka zostane rovnaká. Teda najkratší tunel sa dá zostrojiť tak, aby aspoň jeden z jeho krajných bodov bol vrcholom nejakého mnohouholníka.

Z dokázaných tvrdení vyplýva, že stačí pre každý bod prvého mnohouholníka a každú úsečku druhého mnohouholníka (a naopak) zistiť ich vzdialenosť a vybrať z nich najmenšiu. Pritom vzdialenosť bodu P od úsečky AB je buď vzdialenosťou bodu P od priamky AB (ak kolmica z P na priamku AB leží na úsečke AB), alebo kratšou vzdialenosťou zo vzdialeností $|AP|$, $|BP|$ (v prípade, že kolmica na úsečke AB neleží).

A ako to implementovať? Na zistenie, či daný bod P leží v danom mnohouholníku, použijeme nasledujúci postup: z bodu vedieme polpriamku ľubovoľným smerom (pre jednoduchosť implementácie si môžme zvoliť napr. smer osi y – smerový vektor je $(0, 1)$) a zistíme, s koľkými stranami mnohouholníka sa polpriamka z bodu P v smere $(0, 1)$ pretína. Bod P leží vnútri mnohouholníka práve vtedy, ak je počet týchto priesečníkov nepárny.

Okrajové prípady, ak P leží na strane mnohouholníka netreba ošetrovať (nulovú vzdialenosť by sme našli aj vtedy, ak by sme mnohouholníky považovali za disjunktné). Na ošetrenie

[6] Na prvú časť podmienky ste mnohí zabudli. Ak však jedenmnohouholník leží celý v druhom, druhá časť podmienky na overenie disjunktnosti nestačí.

prípadov, keď polpriamka prechádza nejakým krajným bodom strany mnohouholníka, posunieme všetky body o malú vzdialenosť ε doprava. Keď bude ε menší, ako presnosť čísel na vstupe, takéto prípady určite nenastanú.

Pretože parametrická rovnica úsečky AB je $x = A_x + t(B_x - A_x), y = A_y + t(B_y - A_y), t \in \langle 0, 1 \rangle$, na zistenie, či sa úsečky AB a CD pretínajú, nám stačí riešiť sústavu dvoch rovníc o dvoch neznámych $A_x + k(B_x - A_x) = C_x + l(D_x - C_x), A_y + k(B_y - A_y) = C_y + l(D_y - C_y)$ a overiť, či k aj l budú patriť do intervalu $\langle 0, 1 \rangle$. Rovnobežné úsečky nebudeme považovať za pretínajúce sa. Ak by aj mali spoločný bod, bude to určite aj nejaký ich koncový bod. Nulovú vzdialenosť medzi mnohouholníkmi teda nájdeme aj vtedy, keď ich budeme pokladať za disjunktné.

Ak máme daný bod P a úsečku AB, päta kolmice vedená z P na priamku AB bude na úsečke AB práve vtedy, ak uhly ABP aj BAP budú ostré (teda ich kosínus bude kladný). Kosínus príslušných uhlov možno vypočítať pomocou skalárneho súčinu (pre vektory $\boldsymbol{u}, \boldsymbol{v}$, ktoré zvierajú uhol α platí $\boldsymbol{u} \cdot \boldsymbol{v} = |\boldsymbol{u}||\boldsymbol{v}| \cos \alpha$).

Na vypočítanie vzdialenosti bodu P od priamky AB vypočítame koeficienty rovnice priamky $ax + by + c = 0$ pre priamku AB. Pretože vektor (a, b) musí byť kolmý na smer priamky, môžme položiť $a = B_y - A_y$ a $b = A_x - B_x$. Koeficient c dopočítame tak, aby bod A patril priamke. Potom podľa známeho vzorca je vzdialenosť bodu P od priamky AB rovná $\frac{|aP_x + bP_y + c|}{\sqrt{a^2 + b^2}}$.

Čo dodať na záver? Zistenie, či jeden bod patrí mnohouholníku s n stranami, trvá čas $O(n)$. Ak majú mnohouholníky n a m hrán, na zistenie, či existuje dvojica pretínajúcich sa strán, potrebujeme čas $O(mn)$. Rovnaký čas potrebujeme aj na nájdenie najkratšieho tunela preskúšaním všetkých dvojíc strán. Celková časová zložitosť algoritmu je preto $O(mn)$. Potrebujeme si pamätať súradnice vrcholov oboch mnohouholníkov. Okrem toho vystačíme s konštantným množstvom pomocnej pamäte. Pamäťová zložitosť algoritmu je preto $O(m+n)$.

Listing programu:

```
program O_najkratsom_tuneli;

const MaxN=100;
      Epsilon=1e-8;

type bod=record
          x,y:real;
         end;
     ostrov=array[1..MaxN] of bod;

var m,n:integer;
    O1,O2:ostrov;

procedure Nacitaj;
var i:integer;

begin
 writeln('Zadajte pocet vrcholov 1. ostrova: ');
 read(n);
 writeln('Zadajte suradnice vrcholov:');
 for i:=1 to n do
  read(O1[i].x,O1[i].y);
 writeln('Zadajte pocet vrcholov 2. ostrova: ');
```

```
 read(m);
 writeln('Zadajte suradnice vrcholov:');
 for i:=1 to m do
  read(O2[i].x,O2[i].y);
end;

function Vnutri(A:bod;O:ostrov;n:integer):boolean; { Ci bod lezi vnutri }
var i,c:integer;                                   { ostrova }
    B1,B2,P:bod;

begin
 c:=0;
 for i:=1 to n do
  begin
   B1:=O[i];
   B2:=O[(i mod n)+1];
   B1.x:=B1.x+Epsilon;          { Posunutie o Epsilon }
   B2.x:=B2.x+Epsilon;          { odstrani okrajove pripady }
   if B2.x<B1.x then
    begin
     P:=B1; B1:=B2; B2:=P;
    end;
   if (B1.x<A.x) and (A.x<B2.x) and (abs(B2.x-B1.x)>Epsilon) then
    begin
     if (B2.y-B1.y)/(B2.x-B1.x)*(A.x-B1.x) + B1.y > A.y then inc(c);
    end;                        { Zistenie, ci existuje priesecnik }
  end;                          { s polpriamkou }
 Vnutri:= c mod 2 = 1;
end;

function Pretina(A1,A2,B1,B2:bod):boolean; { Zistenie, ci sa 2 usecky }
var D,k,l:real;                            { pretinaju }

begin
                                     { Sustava rovnic je riesena }
 Pretina:=false;
         { vzorcom pomocou determinantov }
 D:=(A2.x-A1.x)*(B1.y-B2.y)-(A2.y-A1.y)*(B1.x-B2.x);
  { Mozne su aj ine }
 if abs(D)<Epsilon then exit;
           { riesenia }
 k:=((B1.x-A1.x)*(B1.y-B2.y)-(B1.y-A1.y)*(B1.x-B2.x))/D;
 l:=((A2.x-A1.x)*(B1.y-A1.y)-(A2.y-A1.y)*(B1.x-A1.x))/D;
 Pretina:= (0<k)and(k<1)and(0<l)and(l<1);
end;

function Disjunktne:boolean;    { Zistenie, ci su ostrovy disjunktne }
var i,j:integer;

begin
 Disjunktne:=false;
```

```
 for i:=1 to n do
  if Vnutri(O1[i],O2,m) then exit;
 for i:=1 to m do
  if Vnutri(O2[i],O1,n) then exit;
 for i:=1 to n do
  for j:=1 to m do
   if Pretina(O1[i],O1[(i mod n)+1],O2[j],O2[(j mod m)+1]) then exit;
 Disjunktne:=true;
end;

function Vzd(P,B1,B2:bod):real; { Zistenie vzdialenosti usecky a bodu }
var a,b,c:real;
begin
 if ( (B2.x-B1.x)*(P.x-B1.x)+(B2.y-B1.y)*(P.y-B1.y) > 0 ) and
    ( (B1.x-B2.x)*(P.x-B2.x)+(B1.y-B2.y)*(P.y-B2.y) > 0 ) then
  begin                    { Ak pata kolmice lezi na usecke }
   a:=B2.y-B1.y;           { Je to vzdialenost od priamky }
   b:=B1.x-B2.x;
   c:=-(a*B1.x+b*B1.y);
   Vzd:=abs(a*P.x+b*P.y+c)/sqrt(sqr(a)+sqr(b));
  end
 else                      { Inak je to mensia zo vzdialenosti ku koncovym }
  begin                    { bodom usecky }
   a:=sqrt(sqr(P.x-B1.x)+sqr(P.y-B1.y));
   b:=sqrt(sqr(P.x-B2.x)+sqr(P.y-B2.y));
   if a<b then Vzd:=a
          else Vzd:=b;
  end;
end;

function Dlzka:real;       { Zistenie minimalnej dlzky tunela }
var i,j:integer;
    min,act:real;

begin
 min:=1e30;
 for i:=1 to n do
  for j:=1 to m do
   begin
    act:=Vzd(O1[i],O2[j],O2[(j mod m)+1]);
    if act<min then min:=act;
   end;
 for i:=1 to m do
  for j:=1 to n do
   begin
    act:=Vzd(O2[i],O1[j],O1[(j mod n)+1]);
    if act<min then min:=act;
   end;
 Dlzka:=min;
end;
```

```
begin
 Nacitaj;
 if not(Disjunktne)
  then writeln('Minimalna dlzka tunela je 0.')
  else writeln('Minimalna dlzka tunela je ',Dlzka:0:5,'.');
end.
```

opravoval MišoF
(max. 16 bodov)

4. O slepačej farme

Nuž, musím povedať, že som bol (príjemne) prekvapený, keď som videl, koľko z vás tento príklad riešilo. Takže k bodovaniu. Vaše riešenia sa dali rozdeliť do niekoľkých hlavných skupín:

- riešenia, ktoré vždy potrebovali minimálny počet hodov **12–16 bodov**
- riešenia využívajúce myšlienku binárneho vyhľadávania **7–10 bodov**
- riešenia idúce s krokom rovným počtu zostávajúcich vajec **5–6 bodov**

Bodík navyše sa dal získať za rôzne vylepšenia, body sa dali stratiť za chýbajúci, prípadne nezrozumiteľný popis riešenia a tiež za chyby v programe.

Poďme sa teraz lepšie pozrieť na vaše riešenia. Všetci (nuž, takmer všetci) ste prišli na to, že keď mi ostáva len jedno vajíčko, musím výšku postupne zvyšovať o 1 a väčšina z vás si tiež uvedomila to, že ak mám dosť vajec (čiže keď pre počet vajec platí $M \geq \lg N$ [7]), najefektívnejším algoritmom je binárne vyhľadávanie. Pri ňom robíme to, že keď máme interval $\langle a, b\rangle$ taký, že z a sa vajíčko ešte nerozbije a z b už áno, opýtame sa na výšku $v = (a + b)/2$. Ak sa z nej vajce rozbije, ostal nám interval $\langle a, v\rangle$, ak nie, $\langle v, b\rangle$. V oboch prípadoch sa tým dĺžka intervalu zmenší na polovicu. Keď máme interval s $b = a+1$, môžeme skončiť, lebo najvyššia výška, z ktorej sa vajce nerozbije, je a.

Na tomto sa zakladali riešenia, využívajúce myšlienku binárneho vyhľadávania. Boli ich dva typy: prvý typ riešenia začal hádzať vajíčko od jednotky s krokom K, keď sa mu prvýkrát rozbilo, zostal mu interval dĺžky K a $M-1$ vajíčok. Tento interval potom prehľadal binárnym vyhľadávaním. Zjavne vyhovuje krok $K = 2^{M-1}$, ktorý mala väčšina z vás, ale vyskytlo sa aj vylepšenie, ktoré šlo s krokom $K = 2^M - 1$. Druhý typ riešenia začal binárne vyhľadávanie na celom intervale a pokračoval v ňom, kým mal aspoň dve vajíčka. Keď mu zostalo už len jedno, celý interval ním prešiel lineárne. Výhoda tohto prístupu je v tom, že koľkokrát sa nám vajíčko nerozbije, toľko ďalších otázok môžeme položiť a toľkokrát sa nám podarí zmenšiť interval na polovicu. Oba typy riešení potrebovali na výpočet výšky ďalšieho hodu konštantný čas aj pamäť a v najhoršom prípade potrebovali $\frac{N}{2^{M-1}} + M - 1$ hodov.

Okrem toho sa vyskytlo zopár heuristík (odhadov), ktoré na základe výšky intervalu a počtu vajec nejakým spôsobom určili, z akej výšky sa má hodiť ďalšie vajce. Z tejto skupiny spomeniem len myšlienku ísť po odmocnine. Totiž keď máme dve vajíčka, jedno možné riešenie je ísť prvým po odmocnine výšky a druhým potom v získanom intervale lineárne. Takto budeme určite potrebovať najviac $2\sqrt{N}$ hodov. Tento postup sa dá zovšeobecniť aj pre viac vajíčok (zoberie sa delenie intervalu na $\sqrt[M]{N}$ častí) a aj keď nie je optimálny, dáva dosť dobré výsledky.

Prvá myšlienka vzorového riešenia znie: no dobre, chce sa po mne, aby bolo hodov čo najmenej. Keby som vedel minimálny počet hodov, ktorý na takto veľký interval s toľkými vajcami potrebujem a odkiaľ mám hodiť vajíčko, aby som ho dosiahol, mal by som vyhraté – na menej ako minimálny počet hodov to určite nejde `:-)`. Označme si teda $A[i, j]$ najmenší počet hodov, ktorý mi určite stačí na nájdenie kritickej výšky v intervale $\langle 0, i\rangle$, ak mám ešte j vajec. Ako ho zrátať? Určite budeme hádzať vajíčko z výšky $0 < v < i$, lebo o

[7] $lg\ N = \log_2 N$

ostatných výškach už vieme, či sa z nich rozbije alebo nie. Keď hodíme vajíčko z výšky v, môžu nastať dva prípady – buď sa rozbije, nám ostane interval $\langle 0, v\rangle$ a $j-1$ vajec, alebo nie, potom máme interval $\langle v, i\rangle$ (čo je to isté, ako interval $\langle 0, i-v\rangle$) a j vajec. V prvom prípade budeme potrebovať ešte $A[v, j-1]$, v druhom $A[i-v, j]$ hodov. To znamená, že na to, aby sme určite vedeli určiť kritickú výšku, ak hodíme vajíčko z výšky v, budeme potrebovať $\max(A[v, j-1], A[i-v, j])+1$ hodov. My si vyberieme takú výšku, aby tento počet hodov bol čo najmenší. Potom bude $A[i, j] = \min\{\max(A[v, j-1], A[i-v, j]) \mid 0 < v < i\}$. Podľa tohto vzťahu si už ale vieme postupne vyplniť tabuľku A a pre každé jej políčko si v druhej tabuľke $B[i, j]$ pamätať výšku, pre ktorú toto minimum nastáva (a teda z ktorej máme prvýkrát hodiť vajíčko.) Takémuto prístupu (keď zo známych riešení pre menšie úlohy určíme riešenie väčšej) hovoríme dynamické programovanie. V tejto podobe má toto riešenie časovú zložitosť $O(M.N^2)$ na predvýpočet a pamäťovú $O(M.N)$. Jednotlivé skoky už potom vieme vyhodnotiť v konštantnom čase – keď máme interval $\langle a, b\rangle$ a j vajíčok, opýtame sa na výšku $a + B[b-a, j]$.

Keď si takto dostané tabuľky lepšie pozrieme, zistíme, že v nich platia určité závislosti, ktoré nám môžu pomôcť rýchlejšie ich vypĺňať, takto sa dá dosiahnuť riešenie s časovou zložitosťou $O(M.N)$ a pamäťovou $O(N)$.

Druhá myšlienka vzorového riešenia spočíva v tom, že nebudeme pre výšku určovať, na koľko najmenej hodov ju určite vieme spraviť, ale pre daný počet hodov budeme chcieť spočítať, pre aký najdlhší interval vieme na toľkoto hodov určite nájsť kritickú výšku. Ako na to? Budeme hľadať najvyššiu výšku, z ktorej sa ešte vajíčko nerozbije. Tá je na začiatku v intervale[8] $\langle 0, N-1\rangle$. Keď teraz máme interval $\langle a, b\rangle$, vieme, že z a a menej vajíčko prežije a že z viac ako b sa rozbije, preto budeme určite hádzať z výšky $a < v \leq b$. Ak sa nám vajíčko rozbije, hľadaná výška leží v intervale $\langle a, v-1\rangle$, ak nie, v intervale $\langle v, b\rangle$. Keď dostaneme interval dĺžky 0, našli sme hľadanú výšku. Opäť si môžeme všimnúť, že je jedno, či máme interval $\langle a, b\rangle$ alebo $\langle 0, b-a\rangle$, rozhodujúca je len jeho dĺžka.

Označme si $T[i, j]$ dĺžku najdlhšieho intervalu, v ktorom vieme určite nájsť kritickú výšku, ak máme k dispozícii i hodov a j vajíčok. Zjavne bez vajíčok ani bez pokusov toho moc nenarozhodujeme, preto nutne $T[i, 0] = T[0, j] = 0$. Ako teraz porátať ostatné $T[i, j]$? Keď si zoberieme interval $\langle 0, T[i, j]\rangle$, určite nemôžeme pustiť vajíčko z výšky $v > T[i-1, j-1]+1$, lebo ak by sa rozbilo, ostal by nám interval dlhší, ako vieme na $i-1$ pokusov s $j-1$ vajíčkami rozhodnúť. Rovnako ho nemôžeme pustiť z výšky $v < T[i, j] - T[i-1, j]$, lebo ak by sa nerozbilo, ostal by nám pridlhý interval. Preto $T[i, j] \leq T[i-1, j]+T[i-1, j-1]+1$. A zjavne keď zoberieme interval dĺžky $d \leq T[i, j]$ a hodíme vajíčko z výšky $\min(T[i-1, j-1]+1, d)$[9], nech sa stane, čo chce, dostaneme opäť interval vyhovujúcej dĺžky.

Toto by nám už stačilo na riešenie podobné tomu predchádzajúcemu – postupne počítame hodnoty $T[i, j]$ pre j od 0 po M a pre i od 0 až kým nebude $T[i, M] \geq N-1$. Keď teda označíme H minimálny počet hodov potrebný na nájdenie kritickej výšky v intervale $\langle 0, N-1\rangle$, bude mať takéto riešenie časovú aj pamäťovú zložitosť $O(H.M)$, čo je lepšie ako v predchádzajúcom prípade.

No a takmer na záver – ako to ide ešte lepšie? Takto: je $T[i, j] = \sum_{k=1}^{j} \binom{i}{k}$. Dôkaz:

$$T[i, j] = T[i-1, j] + T[i-1, j-1] + 1 = \sum_{k=1}^{j} \binom{i-1}{k} + \sum_{k=1}^{j-1} \binom{i-1}{k} + 1 =$$

$$= \binom{i-1}{1} + \sum_{k=1}^{j-1} \left(\binom{i-1}{k+1} + \binom{i-1}{k} \right) + 1 =$$

[8] Pozor, teraz berieme trochu iný interval, ako v predchádzajúcom riešení, pomôže nám to pri výpočtoch.

[9] To d je tam len pre prípad, že by sme mali príliš krátky interval, aby to bolo korektné – keď máme krátky interval a kopu pokusov, pokojne pustíme jedno vajíčko z najvyššej výšky.

$$= i + \sum_{k=1}^{j-1} \binom{i}{k+1} = \sum_{k=1}^{i} \binom{i}{k}$$

Potom z toho dostaneme, že platí nasledovný vzťah: $T[i+1,j] = 2.T[i,j] + 1 - \binom{i}{j}$. Dôkaz nechávam na vás, v podstate stačí rozpísať $T[i,j]$ a upravovať. Tento vzťah môžeme prepísať do tvaru $T[i-1,j] = \frac{1}{2} \cdot (T[i,j] + \binom{i-1}{j} - 1)$ a tiež vieme, že je $T[i,j-1] = T[i,j] - \binom{i}{j}$. To nám už stačí ako delostrelecká príprava a teraz hor sa k riešeniu:

Budeme si počas riešenia udržiavať v pamäti aktuálne čísla $T[i,j]$ a $\binom{i}{j}$. Na výpočet počiatočného $T[i,j]$ použijeme prvý vzťah a vzťahy medzi kombinačnými číslami. Potom si vždy po opýtaní upravíme i a j a podľa posledných dvoch vzťahov porátame aktuálnu hodnotu $T[i,j]$. Takéto riešenie má časovú zložitosť úmernú minimálnemu počtu otázok a pamäťovú konštantnú, teda je zjavne optimálne.

Listing programu:

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

long Tij,Cij;
int N,M,i,j,a,b,v,result;

int hod(int h) {
  char c;
  printf("Hod z vysky %d. Rozbilo sa ?\n",h);
  c = getch();
  if (c=='n') return 0;
  return 1;
}

int main(void) {

  printf("N = "); scanf("%d",&N);
  printf("M = "); scanf("%d",&M);
  Cij=1; Tij=0; i=1;
  for (j=1;j<=M;j++) {
    Cij = ( (Cij * (i-j+1)) / j );
    Tij += Cij;
  } // mame zratane T[1,M] a (1 nad M)
  j=M;
  while (Tij<N-1) {
    Tij=2*Tij + 1 - Cij;
    i++;
    if (i==j) Cij=1; else Cij = ( (Cij * i) / (i-j) );
    // zvacsujeme i, kym nebude T[i,M] dost velke
  }

  a=0; b=N-1;
  while (b>a) {
    // vyratame T[i,j-1]
    Tij -= Cij;
    if (i!=j-1) Cij = ( Cij * j ) / (i-j+1); else Cij=1;
```

```
    j--;
    // vyratame T[i-1,j-1]
    Cij = ( Cij * (i-j) ) / i;
    Tij = 0.5 * ( Tij - 1 + Cij );
    i--;

    v=a + Tij + 1; if (b<v) v=b;
    result=hod(v);

    if (result==0) {
      a=v;
      // vyratame T[i-1,j]
      if (i!=j) Cij = ( Cij * j ) / (i-j); else Cij=0;
      Tij = Tij + Cij;
      j++;
    } else {
      b=v-1;
    }
  }

  printf("Hladana vyska je %d.\n",a);
  return 0;
}
```

opravovali Braňo a JanoS
(max. 15 bodov)

5. O Santolande I

Túto úlohu ste v zásade riešili dvoma spôsobmi. Jedno z toho boli viac-menej vzorové riešenia, to druhé backtracky. Vyskytlo sa aj niekoľko pokusov o lineárne riešenia, prípadne heuristiky, ktoré však boli načisto zlé a boli ohodnotené 1–3 bodmi.

Riešenia backtrackingom (prehľadávaním do hĺbky, príp. skúšaním všetkých možností) tvorili asi polovicu všetkých riešení. K týmto riešeniam treba dodať už len toľko, že občas by sa zišlo zamyslieť nad počtom vnáraní a ušetriť počítaču kus zásobníka. Dobrý backtrack bol ohodnotený 7 bodmi.

Viacerých z vás napadlo použiť frontu a v nej si pamätať prezreté komôrky. Takéto riešenie, dobre napísané, by sa vyrovnalo vzorovému, ale dotiahnuť do konca sa ho podarilo iba jednému z vás. Všetci ostatní ste urobili tú istú chybu – nepamätali ste si, ktoré komôrky pre dané písmenko z papierika už vo fronte máte a kľudne ste ich vložili viackrát. Tu ale prichádza problém, lebo počet komôrok vo fronte vám mohol exponenciálne rásť. Z toho potom vyplýva exponenciálna časová aj pamäťová zložitosť – čo je dokonca horšie ako backtrack. Takéto riešenia boli ohodnotené najviac 7 bodmi.

Ostatné riešenia spadali do troch kategórií, a podľa časovej zložitosti boli aj bodované. Označme si N počet komôrok, M počet tunelov a L dĺžku papierika. Riešenie s časovou zložitosťou $O(MNL)$ dostalo 10 bodov, takmer vzorové $O(N^2L)$ 13 bodov a vzorové $O(ML)$ 14 bodov. Riešili ste to buď dynamickým programovaním, frontou alebo tak ako to je vo vzorovom riešení.

Na záver sme ešte popis ocenili -1 až 1 bodmi, no a teraz hor sa na vzorák!

Dôležité bolo uvedomiť si, že nik od vás nechce zistiť presne cestu, akou treba ísť. Nás zaujímalo iba to, kde všade sa môže skončiť. Pozrime sa na to takto: Na začiatku môžeme byť iba v prvej komôrke. Po prvom kroku sa môžeme nachádzať v tých komôrkach, do ktorých vedie z prvej chodba, na ktorej je prvé písmenko. Po druhom kroku v tých, do ktorých vedie na druhé písmeno chodba z nejakej komôrky, do ktorej sme sa vedeli dostať v prvom kroku.

Ak tento postup opakujeme pre všetky písmenká, dostaneme množinu všetkých komôrok, kde môžeme skončiť po prečítaní celého papieriku. Zamyslime sa teraz nad zložitosťou takéhoto algoritmu. Pre každé písmenko z papieriku a pre každú komôrku, do ktorej sme sa vedeli dostať v minulom kroku, kontrolujeme, do ktorých komôrok sa môžeme dostať teraz. Ak by sme to robili hlúpo, dostávame časovú zložitosť $O(N^2L)$ a pamäťovú zložitosť $O(LN + M)$

Zlepšenie časovej zložitosti: Ak si tunely utriedime podľa začiatočnej komôrky, a zapamätáme si kde sú tunely idúce z i-tej komôrky, vieme robiť pre každé písmenko iba M operácií. Teda N krát zopakujeme prezeranie tunelov, ale vždy prezeráme iné tunely, ktorých je dokopy len M ($M < N^2$).

Zlepšenie pamäťovej zložitosti: Náš algoritmus nepotrebuje vedieť kam sa mohol dostať v i-tom kroku pre každé i. Podstatné je len to, kde môže byť v aktuálnom a predchádzajúcom kroku. Preto použijeme dve jednorozmerné polia, ktoré budeme navzájom striedať. Týmto ušetríme jeden rozmer poľa a výsledná pamäťová zložitosť bude $O(L + M)$.

Ešte jedno drobné zlepšenie ste mohli dosiahnuť, ak ste si nepamätali celý papierik. Stačilo si pamätať jedno písmenko, ktoré práve spracuvávame.

Niektorí ste si všimli, že Santove bludisko je podľa terminológie formálnych jazykov a automatov nedeterministický konečný automat, a my sme vlastne zisťovali, že či patrí dané slovo do jazyka alebo nie. Ak chcete o danej téme vedieť viac, počkajte na ďalšiu sériu, alebo na druhý ročník na FMFI UK.

Listing programu:

```
Program O_Santolande_I;
Const Max=50;
Var N,M:Integer;
    S:String;                          {Papierik     }
    Rum:Array[1..Max] Of Boolean;      {Kde je rum   }
    Zac:Array[1..Max] Of Integer;      {Zaciatky hran}
    A:Array[0..2*Max*Max] Of Record    {Tunely v bani}
                               A,B:Integer;
                               C:Char;
                             End;
Procedure Nacitaj;  {Nacitanie vstupu}
Var I,J,Rumu:Integer;
    Medzera:Char;
Begin
   Readln(N);
   Readln(M);
   For I:=1 To M Do
      Readln(A[I].A,A[I].B,Medzera,A[I].C);
   Readln(Rumu);
   FillChar(Rum,Sizeof(Rum),False);
   For I:=1 To Rumu Do Begin
      Readln(J);
      Rum[J]:=True;
   End;
   Readln(S);
End;

Procedure QSort(L,R:Integer); {Klasicky QuickSort}
Var I,J,X:Integer;            {v case O(N log N) }
Begin
```

```
   I := L; J := R; X := A[(L+R) Div 2].A;
   Repeat
      While A[I].A < X      Do I:=I+1;
      While      X < A[J].A Do J:=J-1;
      If I <= J Then Begin
         A[0]:=A[I]; A[I]:=A[J]; A[J]:=A[0];
         I:=I+1; J:=J-1;
      End;
   Until I > J;
   If L < J Then QSort(L, J);
   If I < R Then QSort(I, R);
End;

Procedure Pocitaj;
Var Pozri:Array[1..2,1..Max] Of Boolean;
    Ktore,I,J,K:Integer;
    OK:Boolean;
Begin
   For I:=1 To N Do Zac[I]:=1;
   For I:=M DownTo 1 Do Zac[A[I].A]:=I;
   Ktore:=1;
   FillChar(Pozri[1],Sizeof(Pozri[1]),False);
   Pozri[1,1]:=True;
   For I:=1 To Length(S) Do Begin
      FillChar(Pozri[3-Ktore],Sizeof(Pozri[3-Ktore]),False);
      For J:=1 To Max Do
         If Pozri[Ktore,J] Then Begin
            K:=Zac[J];
            While (A[K].A=J) Do Begin
               If A[K].C=S[I] Then Pozri[3-Ktore,A[K].B]:=True;
               Inc(K);
            End;
         End;
      Ktore:=3-Ktore;       {Prehadzujem si dva polia, 1 a 2}
   End;
   OK:=False;
   For I:=1 To N Do
      If Pozri[Ktore,I] And Rum[I] Then OK:=True;
   If OK Then Writeln('Mali ste lepsie hladat cestu!')
         Else Writeln('Santo vas podviedol...');
End;

Begin
   Assign(Input,'KSP-1-5.IN'); Reset(Input);
   Nacitaj;
   QSort(1,M);
   Pocitaj;
End.
```

Chvíľka napätia pred výsledkovou listinou...

Vsledkov listina po 1. srii kateg¡=rie KSP

	Meno a priezvisko		řkola	11	12	13	14	15	Σ
1.	Oravec Jn	4	Gym. Tajovskho B.Bystrica	15	15	15	12	15	72
2.	DvorskÀ Marin	4C	Gym. Koice, robrova	9	15	15	16	15	70
	ZthureckÀ Tom	6F	Gym. V.Paulnyho-T Martin	15	15	12	13	15	70
4.	Rudiin Miroslav	4	Gym. Koice, robrova	9	15	15	14	15	68
5.	Tvaroek Michal	4C	Gym. J. Hronca Bratislava	5	15	14	13	14	61
	Tvaroek Jozef	3C	Gym. J. Hronca Bratislava	5	14	15	12	15	61
	Haraga Dvid	4	Gym. Tajovskho B.Bystrica	9	13	14	10	15	61
	Macko Martin	4C	Gym. Spi. Nov Ves, kolsk	8	15	14	9	15	61
9.	Bella Peter	3C	Gym. J. Hronca Bratislava	5	14	12	14	15	60
10.	Bauer Radovan		Gym. Koice, Potov	9	15	8	10	15	57
11.	Juhos Pavol	3	Gym. A.Markua Bratislava	9	12	12	9	14	56
	Glaus Michal	4A	Gym. J. Hronca Bratislava	8	13	10	10	15	56
13.	Mller Pavol	3	Gym. A.Markua Bratislava	9	11	15	9	11	55
	Bombiak Tibor	4	Gym. A.Bernolka Nmestovo	8	13	14	12	8	55
15.	Dzetkuli Tom	3A	Gym. P.Horova Michalovce	8	15	4	11	14	52
16.	Sghy Tom	3	SPE Nov Zmky, Pod kopcom	2	13	11	10	15	51
17.	Hrbik Luk	1B	Gym. A.Merici Trnava	8	12	11	10	7	48
	Havlj Frantiek	4	.....	8	9	15	8	8	48
19.	VeselÀ Jozef	2B	Gym. Nitra, Golianova 68	4	12	13	10	8	47
20.	Burdiliak Boris	4C	Gym. J. Hronca Bratislava	9	13		9	15	46
21.	Kuchark Ladislav		SPE Pie»any, SNP	2	14	12	10	7	45
	LehotskÀ Jakub	4A	Gym. LiptovskÀ Hrdok	5	9	14	10	7	45
23.	BuranskÀ Rado	4C	Gym. J. Hronca Bratislava	4	13	11	9	7	44
	Krah Peter	4	Gym. Nitra, Provsk 1	5	9	12	10	8	44
25.	Katreni Jn	3B	Gym. Spi. Nov Ves, kolsk	4	9	12	10	8	43
	TruchlÀ Peter	3	Gym. Nitra, Provsk 1	5	9	11	10	8	43
27.	Horvth Anton	4A	Gym. Sere	13	11		10	8	42
	Veelnyi Michal	5L	Gym. Tajovskho B.Bystrica	5	6	13	10	8	42
29.	Bednrik -ubo	3F	Gym. -.tra, Trenn	2	13	11	8	7	41
30.	Nourani Sana		Gym. J. Hronca Bratislava	4	9	11	10	6	40
31.	Gai Peter	4	Gym. A.Markua Bratislava	9	14		9	7	39
	ulk Radovan	1B	Gym. Nitra, Golianova 68	8	12	2	10	7	39
33.	Miklian Peter	4B	Gym. B.S.Timravy Luenec	2	12	12	9	2	37
	Ludha Marek	1F	Gym. Tajovskho B.Bystrica	2	9	8	10	8	37
	Klempai Michal	4B	Gym. A.Bernolka Nmestovo	2	13	10	4	8	37
	Palt Michal	4	Gym. A.Markua Bratislava	4	13	5	7	8	37
37.	Revay Marin	3B	Gym. adca	4	13		10	8	35
	Psotka Frantiek	4A	Gym. Koice, Opatovsk	14	9		4	8	35
	Sekera Klement		Gym. A.Markua Bratislava	2	13	10	10		35
40.	Steskal -ubo	4	Gym. A.Markua Bratislava	9	15		10		34
41.	PokornÀ Michal	4	Gym. A.Markua Bratislava	9	15		9		33
	Riha Ondrej	3	Gym. A.Markua Bratislava	5	3	11	6	8	33
43.	Glaus Peter	2B	Gym. J. Hronca Bratislava	7	13		10	2	32
	Lakatos Tom	4E	Gym. Nov Zmky		9	6	9	8	32
45.	Slovik Vojtech	4C	Gym. J. Hronca Bratislava	4	10		9	8	31
	Florek Martin	4	Gym. A.Markua Bratislava		13		10	8	31
	Chromek Daniel		Gym. B.S.Timravy Luenec	4	9	11	5	2	31
48.	Kollr Ivor	2C	Gym. J. Hronca Bratislava	6	9	0	9	5	29
49.	Havlek Libor	3B	.....	4	8		10	6	28
	Plavan Juraj	4A	Cirk. gym. Bratislava, achtick		12		8	8	28

	Meno a priezvisko		řkola	11	12	13	14	15	Σ
	Mazk Jn	3A	Gym. Koice, Potov		11	8	9		28
	Lovek Peter	4	Gym. Povask Bystrica	6	9	7		6	28
53.	Urban Jaroslav	3A	Gym. LiptovskÀ Hrdok		9		10	8	27
	Farini Igor	4A	Gym. Snina	4	9	8	nn	6	27
55.	Pelk Jakub	1D	Gym. L.Stockela Bardejov	4	9		5	8	26
	Rodk Roman	2B	Gym. A.Merici Trnava	6	9		5	6	26
57.	Rjako Michal	2A	Gym. Vranov n/T, Daxnerova	9			9	7	25
58.	Miku Michal	3C	Gym. J. Hronca Bratislava	4	12			8	24
	Berta Radovan	2A	Gym. Vranov n/T, Daxnerova		13	6	5		24
60.	Vlkov Zuzana	4	Gym. Koice, Alejov	3	13		5	1	22
	Piatka Peter	4E	Gym. -.tra, Trenn	7	9		6		22
62.	Rybr Pavol		Cirk. gym. Bratislava, achtick		13		8		21
	SÀkora Peter	4	Ev. gym. B.Bystrica		8		6	7	21
	Kolnsky Miro	3	Gym. A.Markua Bratislava	2	13		6		21
	bodk Michal	4	Gym. Povask Bystrica		9		9	3	21
66.	Baka Kristin	4C	Gym. Levice	3	12		5		20
	Vranec Maro	2A	Gym. Koice, Alejov		13		7		20
	Haluzov Soa	2C	Gym. J. Hronca Bratislava		13			7	20
	Rybr Tom	2C	Gym. J. Hronca Bratislava	6	9		nn	5	20
70.	KopanskÀ Peter	3A	Gym. Krompachy	2	9	1	5	2	19
	Pekar Andrej	8A	Gym. A.Merici Trnava	2	9	6	2		19
72.	Bobok Pavol	4C	Gym. Levice	3	9		5		17
73.	Vasilov Eva	4A	Gym. P.Horova Michalovce		9		5	2	16
74.	Cifra Marek	1B	Gym. A.Merici Trnava	2	9		4		15
	Teke Michal	3E	Gym. Preov, Kontantnova				9	6	15
76.	Zika Martin	2C	Gym. J. Hronca Bratislava		8			6	14
77.	ikov Jana	3	Ev. gym. B.Bystrica	4	9				13
	NovotnÀ Luk	2C	Gym. J. Hronca Bratislava		13				13
79.	Dzurjanin Peter	3	Gym. A.Markua Bratislava		11				11
80.	Miku Tom	2D	Gym. J. Hronca Bratislava	3	7				10
81.	Hudk Miroslav	3B	Gym. Nitra, Golianova 68		9				9
82.	Totuek Zdenko	6F	Gym. V.Paulnyho-T Martin	2	1		3	2	8
	Lipkov Juliana		Gym. J. Hronca Bratislava	2	1		5		8
84.	Cupper -ubomr	3A	Gym. Krompachy				5		5