Korešpondenčný seminár z programovania
XX. ročník, 2002/2003
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje MICROSTEP-MIS spol. s r.o.

Kategória Z

Vzorové riešenia 1. kola zimnej časti

Milí riešitelia!

Prší, prší len sa leje... Jesenné sústredenie za nami, do Vianoc ďaleko... Hoci už bolo Martina, ešte veľa snehu nie je, zato je pľušť jak sviňa. Tak nechoďte radšej von, aby ste neprechladli. Pekne sa zababušte do deky, prečítajte si vzorové riešenia a čakajte na sneh a Vianoce ako húska na nôž.

V nádržke už voda šumí, mydielkom si ruky umy.

KSPáci

O ideálnom riešení

Na úvod by sme vám radi ešte raz pripomenuli, 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.

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 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 závislosti od vstupných premenných.

Nezaujímajú nás presné hodnoty (ktoré je hlavne u časovej zložitosti navyše aj problém určiť), iba ich rádová veľkosť. Za týmto účelom bola zavedená tzv. O-notácia. Hovoríme, že funkcia $f(n)$ je $O(g(n))$, ak funkciu $f(n)$ vieme zhora odhadnúť nejakým násobkom funkcie $g(n)$.[1] Túto notáciu budeme používať na jednoduchý zápis zložitosti.

[1] Keby sme chceli byť formálni: $f(n) = O(g(n)) \Leftrightarrow \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+; \forall n > n_0; f(n) \leq c \cdot g(n)$

Napríklad ak program pre vstup veľkosti n nebeží nikdy viac ako $T(n) = 0.5n^3+5n \log n+$ 83 mikrosekúnd, vieme čas jeho behu zhora odhadnúť vhodným násobkom funkcie n^3. V takomto prípade hovoríme, že časová zložitosť nášho programu je $O(n^3)$. (Tento zápis môžeme tiež čítať nasledovne: Čas behu nášho programu je **(v najhoršom prípade) rádovo** n^3.) Všimnite si, 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.

Takto odhadnúť časovú a pamäťovú zložitosť programu nie je väčšinou nič ťažké. Ak napr. obsahuje program dva vnorené cykly, z ktorých sa každý vykonáva rádovo n-krát, bude jeho časová zložitosť $O(n^2)$. Zložitosti $O(n)$ zvykneme hovoriť lineárna, $O(n^2)$ kvadratická a pod. Konštantnú (časovú alebo pamäťovú) zložitosť môžeme značiť $O(1)$. Viacero príkladov odhadu časovej i pamäťovej zložitosti nájdete, ak si pozorne prečítate tieto vzorové riešenia.

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.

Opravoval Tom
(max. 15 bodov)

1. Zatopenie Brezna

Tento príklad bol asi dosť jednoduchý, kedže väčšina z vás ho vyriešila dobre. Takmer všetci ste prišli na to, že keď sa Hron vylieva, postupne zaplavuje krajinu, až sa hladina dostane na úroveň najvyššieho bodu krajiny. Ten vytvorí akúsi hrádzu, cez ktorú sa bude prelievať voda do ďalších častí krajiny. Vo zvyšku krajiny sa bude voda opäť rozlievať až kým nedosiahne najvyšší bod tohoto zvyšku krajiny, ten sa stane ďalšou hrádzou, atď. Takto vznikne systém jazierok a hrádz.

Toto pozorovanie sa dá využiť (a väčšinou ste to aj využili) na vyriešenie úlohy: Vždy si nájdem najvyšší bod zvyšku krajiny, zrátam, koľko vody bude v jazierku medzi poslednou dovtedy nájdenou hrádzou (na začiatku je to Hron) a ním, prehlásime tento bod za novú hrádzu a pokračujeme ďalej, až kým sa nedostanem ku Braňovej garáži. Takto dostaneme riešene s kvadratickou časovou zložitosťou, ktoré bolo hodnotené 10 bodmi.

Na podobnej predstave sa dalo založiť aj druhé často sa vyskytujúce (a vzorové) riešenie. Stačí sa na tie jazierka pozerať odzadu. Zrejme posledná hrádza bude Braňova garáž. Kde bude predposledná? Zjavne to bude najbližší vyšší bod krajiny. Až po túto hrádzu bude siahať jazierko, v ktorom bude hladina vo výške poslednej hrádze. (Objem tejto vody si teda vieme rátať počas hľadania najbližšej hrádze.) A od takto nájdenej hrádze pokračujeme rovnakým spôsobom ďalej, až kým sa nedostaneme k Hronu. Vtedy už mám zrátanú vodu vo všetkých jazierkách, t.j. riešenie úlohy. Takéto a všetky riešenia s lineárnou časovou zložitosťou boli hodnotené 15 bodmi.

Ďalej sa vyskytli také podivné modifikácie predchádzajúcich riešení ktoré rátali vodu po 1 metri kubickom. To im zhoršilo zložitosť, preto boli odmenené prémiou −2b. Za nedostatočný popis sa taktiež dávala prémia −2b.

Listing programu:

```
program Zatopenie_Brezna;
const VELA=10000;

var F:array[0..VELA] of integer;   {profil krajiny}
    N,i,V,h:integer;

begin
  readln(N);
  F[0]:=0;
  for i:=1 to N do read(F[i]);  {nacitame ...}
  V:=0;h:=F[N];   {posledna hradza je garaz}
```

```
  for i:=N-1 downto 0 do
    if F[i]<h then V:=V+(h-F[i]) {cast jezierka}
    else h:=F[i];  {nova hradza}
  writeln(V);   {vypiseme vysledok}
end.
```

Opravovala Monika
(max. 13 bodov)

2. Z gaštanov

Miško skladá z gaštanov a zápaliek útvary, našou úlohou je zistiť ich počet. Na hľadanie počtu útvarov existujú štandardné metódy, zvané prehľadávanie do šírky a prehľadávanie do hĺbky, ktoré bežia v rovnakej (optimálnej) časovej zložitosti $O(N+M)$. Toto vzorové riešenie podrobne vysvetľuje algoritmus prehľadávania do šírky:

Budeme používať dátovú štruktúru fronta. Frontu si môžeme predstaviť ako rad ľudí u lekára. Keď nejaký človek odíde z ambulancie, po chvíli lekár zavolá dnu prvého človeka z radu. Keď nejaký človek príde do čakárne, postaví sa na koniec radu. Teda taký rad u lekára podporuje 2 operácie: *pridanie* na koniec radu a *vybratie* zo začiatku radu.

Takúto frontu si vieme naprogramovať ako jednorozmerné pole s dvoma indexami (v programe sa volajú `zac` a `kon`), ukazujúcimi na začiatok a tesne za koniec fronty. Pokiaľ chceme pridať na koniec nejaký prvok, tak ho uložíme na miesto, kam ukazuje index konca fronty a posunieme tento index o 1. Podobne keď chceme vybrať zo začiatku nejaký prvok, zoberieme ten, na ktorý ukazuje index začiatku fronty. Tento index následne zvýšime o 1, čím práve nájdený prvok z fronty odstránime.

Prehľadávanie do šírky funguje nasledovne: Budeme postupne farbiť gaštany na ružovo. Pre každý gaštan si pamätáme, či už bol zafarbený alebo ešte nie. Na začiatku nie je zafarbený žiaden. Kým ešte máme nejaké neofarbené gaštany, opakujeme nasledovný postup:

Vyberieme si ľubovoľný neofarbený gaštan, ofarbíme ho a vložíme do fronty.[2] Kým sa nám fronta nevyprázdni, opakujeme: Vyberieme z fronty (už ofarbený) gaštan. Teraz sa pozrieme na všetky gaštany, ktoré s ním susedia (sú spojené zápalkou). Ak niektoré z nich nie sú ofarbené, tak ich ofarbíme a vložíme do fronty. Keďže každý gaštan ofarbíme a vložíme do fronty najviac raz, časom sa nám fronta vyprázdni. Čo sa stalo? Najskôr sme ofarbili náš vybraný gaštan, potom sme postupne ofarbili jeho susedov, potom susedov jeho susedov, atď. Zjavne sme ofarbili práve všetky gaštany, ktoré patria do toho istého útvaru ako ten, ktorý sme si na začiatku vybrali.

Podľa tohoto algoritmu sa dá napísať program, ktorého časová aj pamäťová zložitosť bude priamo úmerná počtu gaštanov a zápaliek dokopy. Pre každý gaštan si budeme pamätať, koľko zápaliek z neho vychádza a do ktorých gaštanov vedú. (Takto vieme rýchlo prezrieť všetky susedné gaštany nami zvoleného gaštanu.) Pamäťová zložitost tohto programu je zjavne $O(N+M)$. Aj časová zložitosť nášho algoritmu je $O(N+M)$, lebo každý gaštan práve raz zafarbíme a vložíme do fronty a po každej zápalke ideme dvakrát (keď hľadáme susedov gaštanov, ktoré spája).[3]

Hodnotenie: podľa časovej zložitosti:

- $O(N+M)$: 13 bodov
- $O(N^2)$: 12-11 bodov
- $O(NM)$: 10-9 bodov
- $O(M^2)$: 8-7 bodov
- iné funkčné alebo polofunkčné riešenie s horšou zložitosťou: 6-5 bodov

[2] Niekedy hovoríme, že z tohto gaštanu spustíme prehľadávanie.

[3] Nepovinná domáca úloha: Zamyslite sa, nakoľko by sa časová a pamäťová zložitosť zhoršili, keby sme gaštany nefarbili pri vkladaní do fronty, ale až pri vyberaní.

- polofunkčné riešenia a nefunkčné riešenia menej

Listing programu:

```
Const max=100;
Var  vstup : array [1..max,1..max] of integer;
     deg : array [1..max] of integer;
     fronta : array [1..max] of integer;
     oznac : array [1..max] of boolean;
     utvary : integer;
     i,j,N,M,a,b: integer;
     zac,kon,pom: integer;

begin
   utvary:=0;
   for i:=1 to max do begin deg[i]:=0; oznac[i]:=false; end;
   readln(N,M);
   for i:=1 to m do begin
      readln(a,b);
      Inc(deg[a]); vstup[a,deg[a]]:=b;
      Inc(deg[b]); vstup[b,deg[b]]:=a;
   end;
   for i:=1 to n do if oznac[i]=false then begin
      Inc(utvary);
      zac:=1; kon:=2; fronta[kon]:=i; oznac[i]:=true;
      repeat
         pom:=fronta[zac]; Inc(zac);
         for j:=1 to deg[pom] do if not oznac[vstup[pom,j]] then begin
            fronta[kon]:=vstup[pom,j]; Inc(kon);
            oznac[vstup[pom,j]]:=true;
         end;
      until (zac=kon); { fronta je prazdna }
   end;
   writeln('Pocet figurok je ',utvary,'.');
end.
```

Opravoval Paľo

3. Zvianočnieva sa

(max. 13 bodov)

Väčšina vašich riešení postupne prechádzala cez všetky vetvy a zistila, kedy sa už strom nezmestí do dverí. Takéto riešenie potrebuje na svoju prácu čas priamo úmerný počtu vetiev t.j. má časovú zložitosť $O(N)$. Za takéto riešenie ste mohli získať 8 bodov. Bolo si však treba uvedomiť, že môže nastať niekoľko špeciálnych prípadov, ktoré sa musia ošetriť zvlášť. Za riešenia podobné vzorovému (t.j. s časovou zložitosťou $O(\log N)$) sa dalo získať 11 bodov. Za chyby v programe som strhával do 2 bodov. Za popis algoritmu ste mohli získať ďalšie 2 body.

Vzorové riešenie: Použijeme metódu binárneho vyhľadávania. Na začiatku vieme, že miesto rozpílenia leží niekde medzi vrchom a spodkom stromčeka. Pozrieme sa na strednú vetvu. Ak sa zmestí do dverí, tak vieme, že stromček musíme rozpíliť niekde pod ňou. Naopak, ak sa do dverí nezmestí, tak musíme stromček rozpíliť niekde nad ňou. (To vyplýva z toho, že vetvy sú v poli zaradom od vrcholu usporiadané podľa šírky.) Všimnime si, že sa nám v oboch prípadoch interval, na ktorom môže ležeť miesto rozpílenia, zmenšil na polovicu. Takto to budeme opakovať na tomto menšom intervale, až pokiaľ sa nám interval nezmenší na veľkosť 1. To znamená, že sme našli dvojicu vetiev, medzi ktorými treba strom rozpíliť.

Teraz už len pár slov k implementácii. Vytvorme si na vrchole stromu vetvu, ktorá ma nulovú šírku, a teda sa určite zmestí do dverí a na spodku stromu si vytvorme vetvu s väčšou

šírkou ako je šírka dverí, a teda sa do dverí nezmestí. Počas výpočtu si budeme pamätať index vetvy, ktorá sa určite zmestí do dverí a index vetvy, ktorá sa už určite do dverí nezmestí. Na začiatku inicializujme tieto dva indexy na novovytvorené vetvy. Pomocou vyššie uvedeného algoritmu sa budú tieto dva indexy k sebe približovať a ak budú ukazovať na dve za sebou idúce vetvy, tak skončíme, pretože vieme, že stromček treba rozpíliť niekde medzi nimi.

Stromček budeme vlastne píliť dvakrát. Najskôr ho rozpílime tesne nad prvou vetvou, ktorá sa už nezmestí cez dvere. (Ak je to vetva, ktorú sme pridali na spodok, znamená to, že ho nepílime vôbec.) Takto dostaneme najvyšší možný strom, ktorý prejde dverami na šírku. Ak je privysoký, musíme ho ešte spíliť na výšku dvier.

Pamäťová zložitosť je O(1), lebo potrebujeme len konštantný počet pomocných premenných. Časovú zložitosť zistíme nasledovnou úvahou: Po každom pozretí vetvy sa nám počet vetiev, o ktorých ešte nevieme, či prejdú cez dvere, zmenší približne na polovicu. Preto keď máme $N = 2^K$ vetiev, potrebujeme sa pozrieť na rádovo $K(= \log_2 N)$ vetiev stromu. Časová zložitosť je teda $O(\log N)$.

Listing programu:

```
program Zvianocnieva_sa;
var k, l, m, n: integer;
    vzd: array[1..100] of integer;
    sir: array[1..100] of integer;

procedure strom;
var
    vlavo, vpravo, stred: integer;
    akt, i: integer;
begin     {binarne vyhladavanie}
    vlavo := 0;  vpravo := n + 1;
    while vpravo - vlavo > 1 do begin
        {pozrieme sa do stredu intervalu}
        stred := (vlavo + vpravo) div 2;
        {podla toho, ci sa konar v strede zmesti do dveri zmensime interval}
        if (vzd[stred] > l) or (sir[stred] * 2 > k) then
            vpravo := stred else vlavo := stred;
    end;
    akt := vpravo;
    {test ci sa zmesti cez dvere kazda vetva}
    if akt = n + 1 then begin
        if m <= l then writeln('Stromcek netreba rozpilit.');
           else writeln('Stromcek treba rozpilit vo vzdialenosti ', l, 'cm od vrcholu.');
    end else if vzd[akt] <= l then begin
      {najvrchnejsia vetva, ktora sa nezmesti do dveri,
     sa nezmesti kvoli sirke, ale zmesti sa kvoli vyske}
         writeln('Stromcek treba rozpilit vo vzdialenosti ', vzd[akt], 'cm od vrcholu.');
        writeln('Je to tesne nad ', akt, '. vetvou.');
    end else begin
      {najvrchnejsia vetva, ktora sa nezmesti do dveri,
         je prilis nizko, a treba pilit nad nou (nie tesne)}
        writeln('Stromcek treba rozpilit vo vzdialenosti ', l, ' od vrcholu.');
        writeln('Je to ', vzd[akt] - l, 'cm nad ', akt, '. vetvou.');
    end;
end;

var i: integer;
begin
    readln(k, l, m, n); for i := 1 to n do read(vzd[i], sir[i]);
    strom;
end.
```

4. Žiarovky

Opravovala Katka
(max. 15 bodov)

Neviem prečo, ale riešení tohto príkladu prišlo dosť málo, a ešte menej správnych. Správne riešenia boli tie, ktoré dávali správny výsledok, teda **najkratšiu** postupnosť prepnutí žiaroviek, pre každý korektný vstup. Najlepšie riešenia mali časovú náročnosť $O(MN)$ a pamäťovú $O(M+N)$. Za takéto riešenie ste získali 15 bodov. 1 bod ste mohli stratiť za použitú pamäť $O(MN)$, 2 body za horší čas $O((M+N)MN)$, za chýbajúci alebo nedostatočný popis som strhala minimálne 4 body, za chýbajúci program alebo jeho podstatnú časť minimálne 6 bodov, ak program nevypísal najkratšiu postupnosť, prišli ste aspoň o 6 bodov, atď. Jednoducho, body sa ľahšie strácajú ako získavajú. Najčastejšie chyby boli vysoká spotreba pamäte (všetky riešenia okrem jedného) a nezistenie najkratšej postupnosti.

Ako teda zistíme, či sa dajú žiarovky zasvietiť a ako? Väčšina z vás prišla na to, že žiarovky sa dajú zasvietiť práve vtedy, ak každý riadok/stĺpec je buď rovnaký ako prvý, alebo je jeho presnou negáciou. Myšlienka je to správna, ale málokoho napadlo ju využiť už pri načítavaní vstupu a ušetriť tým pamäť. A prečo to platí?

Všimnime si najskôr, že nezáleží na poradí, v akom riadky a stĺpce prepíname. Potom je zjavné, že každý riadok aj stĺpec sa oplatí prepnúť najviac raz. Bez ujmy na všeobecnosti prepnime najskôr všetky riadky. Keď skončíme prepínanie riadkov, musia byť v každom stĺpci buď len vypnuté, alebo len zapnuté žiarovky. To ale znamená, že na začiatku bol každý riadok buď rovnaký ako prvý, alebo jeho negáciou. Navyše po prepnutí riadkov je už jednoznačne určené, ktoré stĺpce treba prepnúť.

To ale znamená, že sú len dve možnosti, ako najkratšia postupnosť prepnutí vyzerá: buď poprepíname riadky tak, aby vyzerali ako prvý, alebo poprepíname riadky tak, aby vyzerali ako negácia prvého riadku (a zakaždým ešte zapneme nesvietiace stĺpce). V našom riešení stačí vyskúšať tieto dve možnosti a vybrať z nich tú kratšiu.

Najskôr teda overíme, či majú všetky riadky správny tvar. Od tohoto okamihu už potrebujeme len spočítať, koľko prepnutí potrebujeme na rozsvietenie všetkých žiaroviek v prvom riadku a stĺpci (potom už budú určite svietiť aj ostatné). Rozoberieme štyri možnosti – či prepneme prvý riadok a či prepneme prvý stĺpec. Dve z nich nebudú vyhovovať, lebo žiarovka v ľavom hornom rohu nebude svietiť. Pre každú zo zvyšných dvoch možností už len spočítame počet núl v (prípadne prepnutom) prvom riadku a stĺpci dokopy – toľkokrát ešte musíme prepnúť. A už len vyberieme tú z týchto dvoch možností, pre ktorú potrebujeme menej prepnutí.

A ešte poznámka pre tých (skoro všetkých), ktorí mali vysokú pamäťovú náročnosť: nechali ste sa zlákať myšlienkou „maticu treba načítať do matice“ (ako to napísal jeden z vás). Treba si uvedomiť, že okrem prvého riadku a prvého stĺpca nepotrebujeme žiarovky na nič iné, ako na zistenie, či je každý riadok rovnaký ako prvý, prípadne ako jeho negácia. Ak to teda zistíme už pri načítavaní vstupu, nemusíme si ich viac pamätať, stačí nám jediná premenná (vo vzoráku z), ktorá reprezentuje aktuálnu žiarovku, a po použití na porovnanie s prvým riadkom ju môžeme smelo hodiť do koša.

Listing programu:

```
program Ziarovky;
const MaxM=1000;
var R:array[1..MaxM] of byte;  {pole obsahujuce 1.riadok}
    S:array[1..MaxM] of byte;  {pole obsahujuce 1.stlpec}
    i,j,M,N,NR,NS,z:integer;
begin
   readln (M); readln(N);
   NR:=0; NS:=0;
   For i:=1 to M do begin    {nacita 1.riadok a zisti kolko je v nom nul}
      readln (R[i]); if R[i]=0 then inc(NR);
```

```
      end;
   S[1]:=R[1];
   if S[1]=0 then inc(NS);
   For i:=2 to N do begin
      readln (S[i]); if S[i]=0 then inc(NS);{pocita kolko je nul v 1.stlpci}
      For j:=2 to M do begin {nacitava znaky matice po riadkoch}
         readln (z);           {a zistuje, ci sa da rozsvietit}
         if ((S[i]=S[1])and(z<>R[j]))or((S[i]<>S[1])and(z=R[j])) then begin
            writeln ('Rozsvietit vsetky ziarovky sa neda');
            halt;            {ak sa neda, hned skonci}
            end;
         end;
      end;
   if R[1]=0 then begin      {ak je vlavo hore 0, treba si vybrat ,co je  }
      if (M-NR)>(N-NS) then begin  {lepsie prepnut:1.riadok alebo 1.stlpec}
         writeln ('1.riadok');
         for i:=2 to M do if R[i]=1 then writeln (i,'.stlpec');
         for i:=2 to N do if S[i]=0 then writeln (i,'.riadok');
         halt;
         end
      else begin
         writeln ('1.stlpec');
         for i:=2 to M do if R[i]=0 then writeln (i,'.riadok');
         for i:=2 to N do if S[i]=1 then writeln (i,'.stlpec');
         halt;
         end;
      end;
   if R[1]=1 then begin      {ak je vlavo hore 1, treba zistit, ci je lepsie}
      if (NR+NS-1)>(M+N-1-(NR+NS)) then begin {prepnut riadok aj stlpec}
         writeln ('1.riadok, 1.stlpec');        {alebo neprepnut nic}
         for i:=2 to M do if R[i]=1 then writeln (i,'.stlpec');
         for i:=2 to N do if S[i]=1 then writeln (i,'.riadok');
         halt;
         end
      else begin
         for i:=2 to M do if R[i]=0 then writeln (i,'.stlpec');
         for i:=2 to N do if S[i]=0 then writeln (i,'.riadok');
         end;
      end;
end.
```

5. Zahltený prácou

opravoval Žužlo
(max. 18 bodov)

K našej radosti sa tento príklad tešil veľkej obľube. Niektorými svojimi bludiskami ste Harrymu riadne zamotali hlavu, a na niektoré si dokonca musel zavolať svojich kamarátov na pomoc. Ako sme túto zamotanú úlohu nakoniec hodnotili?

K hlavnému kritériu nepatril ani až tak program, alebo myšlienka riešenia, dôležité boli najmä výsledné bludiská. Zväčša sa však premyslená myšlienka rovnala dobrému bludisku. Hodnotilo sa predovšetkým estetické hľadisko. Za perfektné bludisko sme považovali také, ktoré malo medzi každými dvoma miestami práve jednu cestu a neobsahovalo nedosiahnuteľné miesta alebo veľké priestranstvá, ktoré len zbytočne plytvajú plochou. Prejdime k hodnoteniu:

- popis algoritmu a program: +4 body
- bludisko bez veľkých miestností: +4 body
- spôsob umiestnenia štartu a cieľu: +2 body

- medzi štartom a cieľom je práve jedna cesta: +2 body
- nemá nedosiahnuteľné oblasti: +2 body
- počet cukríkov v mojich ústach: najviac +4 body

Väčšina z vás začala najskôr s bludiskom plným múrov a ďalej ste, viacmenej premyslene, búraním múrov vytvárali chodby. Nakoniec sa umiestnil začiatok a koniec. Riešenia sme objektívne(dôležitý bol počet cukríkov) bodovali podľa kľúča vyobrazeného vyššie. Kvalitu svojich riešení ste vytrvalo testovali, za čo vám patrí Harryho vďaka.

Ťažko tu hovoriť o nejakom vzorovom riešení. Keď ale zabudneme na bludiská nejakého vzoru (keby Harry poznal vzor prešiel by ich už ľahko) a sústredíme sa na systematické budovanie perfektných bludísk – medzi ľubovoľnými dvoma miestami v bludisku existuje práve jedna cesta – dostaneme vcelku jednoduchý algoritmus. Bludisko si predstavíme ako mriežku prázdnych miestností 1×1 (toto budú práve políčka, ktorých obe súradnice sú párne) medzi ktorými sú múry (ostatné políčka). Tieto múry budeme postupne búrať tak, aby medzi každými dvoma miestnosťami stále existovala najviac jedna cesta. Skončíme, keď už sa bude dať prejsť medzi každými dvomi miestnosťami.

Ale ako tieto steny búrať? Jedna možnosť je upraviť algoritmus prehľadávania do hĺbky, ktorý bude fungovať nasledovne: Začneme v ľubovoľnej miestnosti, potom opakujeme: Ak má miestnosť, kde práve sme, nejakých susedov, kde sme ešte neboli, jedného z nich si vyberieme, zbúrame príslušnú stenu a prejdeme doň. V opačnom prípade sa vrátime do miestnosti, odkiaľ sme sem prišli. Skončíme, keď navštívime všetky políčka. Tento algoritmus sa dá ľahko naprogramovať za použitia rekurzie.

Týmto postupom síce vygenerujeme bludisko, ktoré bude mať veľa z vyššie popísaných vlastností, ale až také bludiskovité zase nebude. To preto, že v ňom bude málo vetvení a veľa dlhých chodieb. Preto si ukážeme ešte jeden prístup. Budeme naše bludisko generovať celé naraz. Vždy si náhodne vyberieme dve susedné miestnosti, medzi ktorými je stena. Ak medzi týmito miestnosťami už vedie cesta, nič sa nedeje, inak túto stenu zbúrame. Týmto postupom zostrojené bludisko bude mať všetky vlastnosti, ktoré má dobré bludisko mať. Jediný problém je ako ho efektívne naprogramovať.

Väčšina algoritmu je jednoduchá, problém bude jedine zisťovať, či medzi dvomi vybranými miestnosťami už vieme prejsť. Jedna možnosť je zakaždým spúšťať prehľadávanie celého dovtedy hotového bludiska. To bude samozrejme fungovať, ale bude to pomerne pomalé. Ako by sa to dalo spraviť lepšie? Všimnime si, ako také bludisko uprostred generovania vyzerá. Je rozpadnuté na niekoľko súvislých oblastí. Keby sme si pre každú miestnosť pamätali, do ktorej oblasti patrí, ľahko zistíme, či sa dá prejsť medzi dvoma miestnosťami – ide to práve vtedy, keď ležia v tej istej oblasti.

Teraz si ukážeme prefíkaný spôsob ako si pamätať, ktorá miestnosť patrí do ktorej oblasti. Potrebujeme totiž navyše to, aby sme vedeli rýchlo spojiť dve oblasti. Pre každú miestnosť si budeme pamätať jednu inú, ktorú budeme volať jej *šéfom*. *Nadriadení* miestnosti budú jej šéf a všetci nadriadení jej šéfa. Táto „hierarchia“ sa nebude smieť zacykliť, teda nesmú existovať dve miestnosti, ktoré by si navzájom boli nadradené. Všimnime si teraz, čo sa stane, keď pre nejakú miestnosť M zoberieme postupnosť M, šéf(M), šéf(šéf(M)), ... Časom musíme prísť do miestnosti, ktorá je sama sebe šéfom (inak by sme sa zacyklili). Túto miestnosť budeme volať *najvyšší šéf* miestnosti M.

Na začiatku bude každá miestnosť sama sebe šéfom, a teda aj najvyšším šéfom. Všimnime si, že teraz (triviálne) platí, že dve miestnosti sú v tej istej oblasti práve vtedy, keď majú toho istého najvyššieho šéfa. Toto bude platiť počas celého výpočtu. Nájsť najvyššieho šéfa miestnosti vieme ľahko. Zostáva ukázať, ako zmeniť túto štruktúru, keď spojíme dve miestnosti a neporušiť si pri tom spomínané vlastnosti.

Majme teda dve susedné miestnosti A, B. Nájdime najskôr ich najvyšších šéfov A', B'. Ak $A' = B'$, patria miestnosti do tej istej oblasti a nič sa nedeje. V opačnom prípade zbúrame stenu medzi nimi. Teraz by sme mali spojiť oblasti, „patriace“ najvyšším šéfom A'

a B'. Nie je nič ľahšie – stačí povedať, že odteraz už B' nebude sám sebe šéfom, ale jeho nový šéf bude A'. Takto A' bude najvyšším šéfom všetkým miestnostiam, ktorým bol najvyšším šéfom doteraz a navyše sa stane najvyšším šéfom miestností, ktorých najvyšší šéf bol doteraz B'. Tým sme ale dosiahli presne to, čo sme chceli – nezacyklili sme sa a naďalej platí, že miestnosti patria do tej istej oblasti práve vtedy, keď majú rovnakého najvyššieho šéfa.

Práve popísaný postup sa volá algoritmus Union-Find. Jeho implementácia je veľmi jednoduchá. Stačí nám jedno pole, v ktorom si pre každú miestnosť pamätáme jej šéfa a jedna rekurzívna funkcia, ktorá nájde najvyššieho šéfa miestnosti. Keď sa zamyslíme, dá sa tento algoritmus ďalej vylepšiť. Problém je v tom, že keď chceme nájsť najvyššieho šéfa miestnosti, môže sa nám stať, že budeme musieť prejsť cez veľa miestností. Budeme sa preto snažiť, aby dĺžky ciest do najvyšších šéfov boli čo najkratšie.

Spomenieme si dve zlepšenia. Prvé z nich bude nasledovné: Pre každého najvyššieho šéfa si budeme pamätať, koľkým miestnostiam je najvyšší šéf. Keď teraz máme spojiť oblasti patriace najvyšším šéfom A', B', najvyšším šéfom zostane ten z nich, ktorý má viac podriadených. (Podriadeným toho druhého sa totiž o 1 predĺži cesta do najvyššieho šéfa, takto sa cesta predĺži menej miestnostiam.) Prípadne si nemusíme pamätať nič, iba si hodíme mincou, kto bude najvyšší šéf. Náhoda zabráni tomu, aby nastal nejaký patologický prípad.

Druhé zlepšenie bude trochu prefíkanejšie. Všimnime si rekurzívnu procedúru, ktorá hľadá najvyššieho šéfa. V okamihu, keď sme ho našli, si ho zapamätáme. Ako sa vynárame z rekurzie, prechádzame postupne späť miestnosťami, ktorými sme išli, keď sme ho hľadali. Vieme teda, že nájdená miestnosť je najvyšším šéfom každej z nich. Tak prečo si túto informáciu nezapamätať? Odteraz si ako šéfa každej z týchto miestností budeme pamätať práve nájdeného najvyššieho šéfa. (Takto sa vzdialenosť každej z nich od najvyššieho šéfa zmenší na 1.)

Dá sa dokázať, že s týmito vylepšeniami nám **v priemere** bude nájdenie jedného najvyššieho šéfa trvať čas $O(\log^*(MN))$. *(Hodnota* $\log^* N$ *je počet, koľkokrát sa musí číslo* N *zlogaritmovať, aby sme dostali číslo mešie ako 1. Pre všetky hodnoty do* 2^{65536} *je to do* 4. *Predstavte si, že napíšete číslo* N *na kalkulačke a potom stláčate tlačítko „log“, kým nedostanete chybu. Počet stlačení tlačítka je približne hodnota* $\log^* N$*.)* Môžeme teda smelo povedať, že v priemere vieme nájsť najvyššieho šéfa miestnosti takmer v konštantnom čase.

Časovú náročnosť nám mierne komplikuje fakt, že vždy vyberáme náhodnú dvojicu susedných miestností, a teda nie je zjavné, či vôbec skončíme. Program by sa dal jednoducho modifikovať tak, aby sme vyberali len medzi tými, ktoré sme ešte nikdy nevybrali. Dá sa však ukázať, že v priemernom prípade sa na každú dvojicu miestností pozrieme len konštantný počet krát. Preto nám to na časovej zložitosti nič nezmení, zostáva $O(MN\log^*(MN))$.

Poznámka pre fajnšmekrov: Najjednoduchšie je umiestniť štart a cieľ do protiľahlých rohov bludiska. Ide to však aj prefíkanejšie. Najvhodnejšie by bolo zvoliť najdlhšiu cestu a jej koncové vrcholy zvoliť za štart a cieľ. Vo všeobecnom grafe sa táto úloha nedá dobre riešiť. Graf bludiska je ale strom (medzi ľubovoľnými dvoma vrcholmi existuje práve jedna cesta) a v stromoch sa dá nájsť najvzdialenejšia cesta v lineárnom čase od veľkosti stromu, a teda keby sme v našom bludisku hľadali najdlhšiu cestu, zložitosť by nám to nezhoršilo. To je ale zasa iná rozprávočka, deti.

Listing programu:

```
program Zahlteny_pracou;
const
  d:array[1..4,1..2]of integer = ((-1,0),(1,0),(0,-1),(0,1));
  inv_dir:array[1..4]of integer = (2,1,4,3);
  pow2:array[1..4]of integer = (1,2,4,8);

var r,s,nc:integer;
    a:array[1..100,1..100]of integer;
```

```
    dad:array[0..100*100-1]of integer;

function otec(v:integer):integer;
begin
  while dad[v]<>v do begin dad[v]:=dad[dad[v]]; v:=dad[v]; end;
  otec:=v;
end;

procedure spoj(v,dir:integer);
var i,j,ni,nj:integer;
begin
  i:=(v div s)+1; j:=(v mod s)+1;
  if (a[i,j] and pow2[dir])>0 then begin
      ni:=i+d[dir,1];nj:=j+d[dir,2];
      i:=otec(v); j:=otec((ni-1)*s+nj-1);
      if i<>j then begin
          dad[i]:=j;
          dec(a[(v div s)+1,(v mod s)+1],pow2[dir]); {nahrada za xor}
          dec(a[ni,nj],pow2[inv_dir[dir]]); {nahrada za xor}
          dec(nc); {zniz pocet komponentov}
  end;
    end;
end;

var i,j:integer;

begin
  readln(r,s);
  r:=(r div 2)+2; s:=(s div 2)+2; nc:=r*s-4;
  for i:=0 to r*s-1 do dad[i]:=i;
  for i:=1 to r do for j:=1 to s do a[i,j]:=15;

  while nc>1 do begin
    i:=2+random(r-2);j:=2+random(s-2);
    spoj((i-1)*s+j-1,1+random(4));
  end;

  for j:=2 to s-1 do write('##'); writeln('#');
  for i:=2 to r-1 do begin
      write('#');
      for j:=2 to s-2 do begin
          if(i=2)and(j=2) then write('s')else write(' ');
          if (a[i,j] and 8)>0 then write('#')else write(' ');
      end;
      if i=r-1 then write('c#')else write(' #'); writeln;

      write('#');
      if i<>r-1 then begin
          for j:=2 to s-1 do
              if (a[i,j] and 2)>0 then write('##')
                 else write(' #');
          writeln;
      end;
  end;
  for j:=2 to s-1 do write('##'); writeln;
end.
```

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Perešíni Peter	Gym. Tajovského B. Bystrica	1	14	11	11	14	18	68
2	Simančík František	Gym. Grösslingová BA	2	15	13	12	15	12	67
3	Labuda Tomáš	Gym. Grösslingová BA	3	15	10	13	14	14	66
4	Cicko Miroslav	Gym. Tajovského B. Bystrica	2	10	10	11	14	18	63
5	Porubský Juraj	Gym. Cyrila a Metoda Nitra	3	15	10	11	14	12	62
6	Nánási Michal	Gym. Jura Hronca BA	2	15	13	11	5	17	61
7	Petruľák Matúš	Gym. Grösslingová BA	2	15	13	10	6	16	60
8	Szórád Pavol	Gym. Cyrila a Metoda Nitra	3	15	6	12	13	11	57
9	Petráš Daniel	Gym. Šk. Bratov BA	2	15	2	10	13	15	55
10	Klátik Peter	Gym. Grösslingová BA	3	15	5	9	9	16	54
11	Imriška Jakub	Gym. Jura Hronca BA	1	15	10	12	14		51
12	Gottweis Martin	Gym. Jura Hronca BA	1	15	4	8	13	10	50
13	Krchniak Matej	Gym. Jura Hronca BA	1	15	6	8	11	9	49
14	Beleš Lukáš	Gym. Čadca	2	10	10	8	13	7	48
15	Sedlák Milan	Gym. Liptovský Hrádok	3	13	10	8	7	9	47
16	Beňo Michal	Gym. Kremnica	3	15	10	9	12		46
16	Mikulík Andrej	Gym. Grösslingová BA	3	15	13	11	7		46
18	Blénessy Tibor	Gym. Poštová Košice	3	15	12	8	7		42
18	Plžík Milan	Gym. Ľ. Štúra Zvolen	2	10	10	8	9	5	42
18	Ľudvík Martin	Gym. Školská Spiš. Nová Ves	2	15	5	9	13		42
21	Dzetkulič Michal	Gym. P. Horova Michalovce	2	15	5	11	10		41
21	Hasaj Martin	Gym. Trebišovská Košice	4	10	5	8	7	11	41
23	Kuna Matej	Gym. Golianova Nitra	2	13	10	9	8		40
24	Bundala Daniel	Gym. Jura Hronca BA	1	8	7	7	9	8	39
25	Legény Jozef	Gymnázium	2	10	12	8	8		38
25	Schlosáriková Eva	Gym. SNP Piešťany	2	13	5	8		12	38
25	Zeman Marek	Gym. Jura Hronca BA	2	15	12	11			38
28	Krištofič Milutín	Gym. Grösslingová BA	3	13	5	8	11		37
29	Budáč Ondrej	Gym. Haličská Lučenec	1	15	10	11			36
29	Zaťko Tomáš	Gym. 17. novembra Topoľčany	3	10	10	9	7		36
31	Buštor Stano	Gym. Jura Hronca BA	2	15	4	11		5	35
32	Ilavský Milan	SPŠE Liptovský Hrádok	3	10	5	8	7	4	34
32	Varga Ľubomír	SPŠ Nitra	3	10	5	8		11	34
34	Kažimír Pavol	Gym. Tornaľa	3	15	4	9	5		33
34	Morihladko Peter	Gym. Školská Spiš. Nová Ves	2	15	10	8			33
34	Vitásek Matej	Gym. Grösslingová BA	2	15	10	8			33
34	Zelinka Ľuboš	Gym. Golianova Nitra	3	15	8	10			33
38	Baník Dušan	Gym. Popradské nábr. Poprad	3	15		10	7		32
38	Nadanyi Peter	Gym. Tvrdošín	4	15		10	7		32
40	Fiala Martin	Gym. Grösslingová BA	3	15	5	11			31
41	Tomori Alexander	Gym. Humenné	3	10	10	10			30
42	Elman Michal	Gym. Golianova Nitra	3	10	10	9			29
42	Rjabinin Igor	Gym. Cyrila a Metoda Nitra	3	15	4	10			29
44	Ambrošová Lucia	Gym. Grösslingová BA	1	15		11			26
45	Hennel Matúš	Neznama skola	3	13	3	9			25
45	Prievalský Juraj	Gym. Nedožerského Prievidza	3	10	7	8			25
45	Zápotocký Radoslav	Gym. Alejová Košice	3	15		10			25
48	Andruška Igor	SPŠ Levice	2	10	3	11			24
48	Batmendijnová Zuzana	Gym. Stará Ľubovňa	3	10	6	8			24
48	Bielik Peter	Gym. SNP Piešťany	2	15	3	6			24

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
48	Filo Michal	Gym. Golianova Nitra	4	10		9		5	24
48	Krištofík Štefan	SPŠ Levice	4	10	4			10	24
48	Sedlák Štefan	Gym. Daxnera V.n. Topľou	2	15		9			24
54	Balga Jozef	Gym. - maďarské Šahy	4	8	0	6	9		23
54	Budáčová Hana	Gym. Haličská Lučenec	3	15		8			23
54	Strobl Radoslav	Gym. Okružná Zvolen	3	10		8	5		23
57	Folkman Lukáš	Gym. Lettricha Martin	3	10		10		1	21
57	Potisková Lucia	Gym. Cyrila a Metoda Nitra	3	10		11			21
59	Hirjak Miroslav	Gym. Humenné	3	10		9			19
59	Smetana Róbert	Gym. Grösslingová BA	2	13		6			19
59	Ďuďák Juraj	Gym. Golianova Nitra	1	6	7	6			19
62	Doršic Matej	Gym. Grösslingová BA	2	10		8			18
63	Šimko Jakub	Gym. Jura Hronca BA	2	8		9			17
64	Kezeš Július	Gym. Golianova Nitra	3	8	8				16
65	Borsuk Andrej	Gym. Grösslingová BA	1	15					15
65	Goga Peter	Gym. 17. novembra Topoľčany	3	1		9		5	15
65	Lebedová Barbora	Gym. Jura Hronca BA	1	15					15
65	Marek Pavol	Gym. A. Merici Trnava	4	15					15
65	Martin Kováčik	Neznama skola	4	15					15
65	Mikuláš Ján	Gym. Haličská Lučenec	1		6	9			15
65	Vranec Maroš	Gym. Alejová Košice	4	15					15
72	Golier Richard	Gym. Alejová Košice	3	3	5	6			14
72	Lacková Veronika	Gym. Jura Hronca BA	2	8		6			14
72	Poláčik Michal	GLN Tomášikova BA	3	10	4				14
72	Šuška Martin	Gymnázium Levice	4	10	4				14
76	Antonič Michal	Gym. Grösslingová BA	1					13	13
76	Cyprich Peter	Gym. Čadca	4	13					13
76	Minár Tomáš	Gym. Golianova Nitra	3	13					13
76	Ruska Štefan	Gym. Alejová Košice	?	13					13
80	Fzeši Štefan	Gym. - maďarské Šahy	?	10	0	2			12
81	Husár Radoslav	Bilingválne gym. Sučany	2	10					10
82	Polovková Darina	Gym. Školská Spiš. Nová Ves	3			8			8
82	Rusinko Rastislav	Gym. Alejová Košice	?	8					8
82	Salaj Martin	Gym. Čadca	4			8			8
82	Tvrdý Lukáš	Gym. Čadca	2			8			8
82	Červenec Peter	Gym. Čadca	4			8			8
87	Černík Peter	Gym. Trebišovská Košice	2	1		6			7
88	Petrezsél Tibor	Gym. - maďarské Šahy	3			6			6
88	Strama Ján	Gym. Alejová Košice	4			6			6
90	Hrabčák Ján	Gym. Alejová Košice	3	1	1	3			5
91	Sakáč Pavol	Gym. Alejová Košice	3	1		3			4
92	Polák Matúš	Gymnázium	2					3	3