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

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

Vzorové riešenia 1. kola zimnej časti, kat. Z

Zima sa nám pomaličky blíži, aj prvý sneh sme tu už mali... Nás konkrétne stretol v Budapešti, kde sme si boli aj my zaprogramovať (na regionálnom kole veľkej svetovej súťaže ACM ICPC) – a hanbu sme si teda nespravili. A hanbu ste si nespravili ani vy, väčšina riešení prvého kola nám dávala zmysel `:-)` a bodov ste si nahonobili dosť a dosť. A ak sa vám máli, šup šup čítať tieto riešenia, keď ich prečítate, nabudúce budete múdrejší. Alebo aspoň sčítanejší `;-)`

Povedz dvakrát „povedz dvakrát povedz dvakrát“

KSPáci

1. Zas ten problém s medovníkmi

Opravoval Zemčo
(max. 15 bodov)

Napriek skutočnosti, že si väčšina z vás myslela (a aj ja som si to myslel), že príklad je ľahký, tak tip-top optimálne bezchybné voňavé riešenia prišli len dve. Identitu neznámych borcov môžete odhaliť vo výsledkovej listine. Ale po poriadku. Prišlo veľa riešení, ktoré boli všetky správne (alebo takmer správne). V zásade sa riešenia dali rozdeliť na tie bežiace v čase $O(\sqrt{N})$ a tie v čase $O(N)$.

Lineárne riešenie bolo naozaj veľmi jednoduché na realizáciu, avšak ani riešenie so zložitosťou $\sqrt{N}$ nebolo obtiažne. Mnohých z vás napadla optimalizácia typu „pozeráme len do polovice, ďalej celočíselné delitele už nebudú“. Takáto idea je správna, ale rádovo sa nelíši od toho najjednoduchšieho riešenia. Časová zložitosť je, aj napriek tomu, že v praxi by sa to pozitívne prejavilo, stále lineárne závislá od veľkosti čísla N (konštanty sa pri jej výpočte zanedbávajú). Preto som nemal takýto zlepšovák dôvod bodovo odmeňovať. Druhým významným kritériom bola pamäť. Mnohí síce prišli na riešenie v optimálnom čase, ale potrebovali si delitele pamätať. Toto som označoval ako $O(P)$ a je to síce lepšia pamäť ako lineárna, ale je rádovo horšia ako konštantá a ako sa ukáže, zaobídeme sa bez nej.

Ako bolo naznačené, optimálne riešenie pracuje v čase $O(\sqrt{N})$ a vystačí si len s dvoma premennými. Najskôr si treba uvedomiť, že keď platí $a \cdot b = N$ (a, b sú prirodzené čísla, $a \leq b$), tak potom $a \leq \sqrt{N} \leq b$. Všetky delitele N vieme takto rozdeliť do dvojíc, v každej dvojici je menšie z čísel nanajvýš rovné $\sqrt{N}$. (Napríklad $30 = 1.30 = 2.15 = 3.10 = 5.6$.) Špeciálny prípad je, keď N je štvorec, vtedy aj $\sqrt{N}$ je celočíselný deliteľ, ten ale chceme vypísať len raz.

Aby sme našli všetky delitele N, stačí nám teda prejsť všetky celé čísla po $\sqrt{N}$.

Ďalej si všimnime, že platí: keď $x < y < \sqrt{N}$, tak potom $\sqrt{N} < N/y < N/x$, skrátka že sa nám akosi zachováva poradie deliteľov. To znamená, že keby sme išli po deliteľoch od 1 po $\sqrt{N}$ a vypisovali ich „zrkadlové dvojičky“, tak ich dostaneme v zostupnom (klesajúcom) podarí. Z toho vyplýva, že keď budeme postupavať od $\sqrt{N}$ späť k 1, tak ich získame vo vzostupnom poradí. Keď napokon prídeme k 1 a vypíšeme jej zkadlového partnera, tak to bude určite ten najvačší deliteľ čísla N – samotné N. Optimálne riešenie prechádza najprv od 1 do $\sqrt{N}$, hľadá deliteľov a vypisuje ich. Potom sa začne vracať späť a hľadá deliteľov čísla

N znova, ale vypisuje už nie ich samotných, ale ich zrkadlové obrazy. Takto máme zaručené, že všetky delitele sa vypíšu vo vzostupnom poradí. Nepotrebujeme si ich pritom pamätať a časová zložitosť sa nám nezhorší, pretože robíme len dva prechody a každý trvá $\sqrt{N}$, takže rádovo to bude stále len $\sqrt{N}$.

Takže do reči bodov: za riešenie v optimálnom čase sa dalo získať 15 bodov a za riešenie v lineárnom čase maximálne 9 bodov. Riešenia, ktoré sa neuspokojili s konštantnou pamäťou boli „odmenené“ stratou dvoch bodov. Pri mnohých riešeniach, pracujúcich v optimálnom čase sa vyskytla takáto chyba: Vypočítali sa delitele do odmocniny N a začali sa vypisovať ich zrkadlové obrazy. Avšak tu sa zabudlo na vyššie spomenutý špeciálny prípad, že zadané číslo ma celočíselnú druhú odmocninu. Potom platí, že $N/i = i$. Tie riešenia, ktoré na tento prípad zabudli, vypisovali druhú odmocninu dvakrát. Toto som označoval OK- (riešenia pamätajúce na tento prípad dostali OK+). Za tento nedostatok som strhával bod. Ďalšie bodíky sa mohli stratiť za chýbajúci popis alebo absentujúci či nesprávny odhad.

Ešte taký malinký pokecík pre pascalistov, prečo som vo vzorovom riešení použil trunc a nie round. Medzi týmito dvoma funkciami je rozdiel v tom, že trunc sa vždy posunie smerom dole – berie vlastne celočíselnú časť reálneho čísla. To znamená, že trunc($\sqrt{N}$) bude určite najväčšie prirodzené číslo pred, alebo v okrajovom prípade na úrovni „zrkadla“, zatiaľ čo round nás môže posunúť už na prvé číslo za zrkadlom. Neexistuje číslo, pri ktorom by sa to prakticky prejavilo, takže vaše riešenie je korektné a body som nestŕhal, ale teoreticky to trocha smrdí a treba si to uvedomiť.

Listing programu:

```
program medovniky;
var i,n:longint;

begin

  readln(n);

  for i:=1 to trunc(sqrt(n)) do
    if n mod i = 0 then write(i,' ');

  if trunc(sqrt(n))=sqrt(n) then
  begin

    for i:=trunc(sqrt(n))-1 downto 1 do
    if n mod i =  0 then write(n div i,' ');

  end
  else
  begin

    for i:=trunc(sqrt(n)) downto 1 do
    if n mod i =  0 then write(n div i,' ');

  end;
end.
```

2. Zúfalý slimák

opravoval Palo
(max. 15 bodov)

Tento príklad sa tváril, že sa dá ľahko vyriešiť, o čom svedčí počet odovzdaných riešení. Avšak opak sa ukázal byť pravdou. Za korektné riešenie bežiace v konštantnom čase sa dalo

získať 15 bodov. Za riešenia v čase $O(K)$ sa dalo získať maximálne 11 bodov. Body sa dali hlavne stratiť za nedostatočný alebo chýbajúci popis, chýbajúci odhad zložitosti alebo za chyby v programe.

Predstavíme si jednoduché riešenie v lineárnom čase: Budeme postupne k aktuálnemu dátumu pridávať jeden deň, až pokým ich nepridáme K. Vždy, keď prekročíme hranicu mesiaca alebo roka, tak si upravíme príslušné záznamy o dátume. Problém môže nastať iba v mesiaci február, ktorý má v priestupných rokoch o deň naviac. To, či je rok priestupný však vieme testovať jednoduchou podmienkou:

$$prestup(r) = ((r \bmod 4 = 0)\ and\ (r \bmod 100\ <>\ 0))\ or\ (r \bmod 400 = 0)$$

Skúsme daný program zlepšiť. Preveďme daný dátum na vstupe v tvare deň, mesiac, rok na počet dní od roku 0 (nula). Pripočítajme k nemu počet dní behu slimačieho programu a nakoniec výsledný počet dní prevedieme naspäť do normálneho formátu dátumu.

Prvá funkcia potrebuje vedieť, aký je počet priestupných rokov od roku nula (vrátane) do nejakého roku r (nevrátane). Na to slúži jednoduchý vzorec: $(r-1) \div 4 - (r-1) \div 100 + (r-1) \div 400 + 1$ (znak $\div$ značí celočíselné delenie). Rok nula považujeme za priestupný. Uvedený vzorec platí len pre kladné hodnoty r, ale to nám nevadí, pretože len pre tie to potrebujeme vedieť. Počet dní od začiatku roka si môžeme pre každý mesiac predrátať už pri písaní programu.

Ešte potrebujeme previesť výsledný počet dní v na dátum. Tu využijeme vlastnosť priestupných rokov. Totiž v každej štyristoročnici sa výskyty priestupných rokov opakujú na tých istých miestach, teda každých 400 rokov je počet priestupných rokov rovnaký. Konkrétne počet všetkých dní za 400 rokov je 146097. Teda si najprv zistíme počet štyristoročníc vo v. Ostane nám nejaký počet dní w z poslednej nedokončenej štyristoročnice, ktorý je určite menší ako 146097. Teda tých w dní môžeme k aktuálnemu dátumu pripočítať už známym algoritmom vyššie. V tomto prípade však nebudeme pridávať až k dní, ale len najviac 146097, čo je (aj keď trochu veľká) konštanta.

Výsledná časová zložitosť je teda zjavne konštantná - $O(1)$. Pamätáme si len niekoľko pomocných premenných a počty dní v jednotlivých mesiacoch, takže aj pamäťová zložitosť je $O(1)$.

Uvedený algoritmus vieme ešte konštantne krát urýchliť napríklad tým, že nebudeme dátum zväčšovať po dňoch, ale po mesiacoch alebo rokoch (prípadne aj storočiach).

Listing programu:

```
var
   mesiac: array[0..1, 1..13] of longint =
((0,31,59,90,120,151,181,212,243,273,304,334,365),
(0,31,60,91,121,152,182,213,244,274,305,335,366));

function prestup(r: longint):longint;
begin
   if (r mod 400 = 0) or ((r mod 4 = 0) and (r mod 100 <> 0)) then
      prestup:=1
   else
      prestup:=0;
end;

function nadni(r,m,d: longint): longint;
var
   res: longint;
begin
   res:=(r - 1) div 4 - (r - 1) div 100 + (r - 1) div 400 + 1 + r * 365;
   res:=res+mesiac[prestup(r)][m]+d;
```

```
    nadni:=res;
end;

procedure nadatum(k: longint);
var
    r,m: longint;
begin
    dec(k);
    r:=k div 146097 * 400;
    k:=k mod 146097;
    while (k>=365+prestup(r)) do begin
        k:=k-365-prestup(r);
        r:=r+1;
    end;
    m:=1;
    while (mesiac[prestup(r)][m]<=k) do m:=m+1;
    dec(m);
    k:=k-mesiac[prestup(r)][m];
    writeln(r, ' ', m, ' ', k+1);
end;

var
    r,m,d,k: longint;
begin
    read(r,m,d,k);
    nadatum(nadni(r,m,d)+k);
end.
```

3. Zase som prehral...

opravoval Mirko
(max. 15 bodov)

Tento príklad sa opravoval veľmi ľahko, lebo ho mali takmer všetci dobre. Body sa strhávali jedine za nedostatočné komentáre, za zlé pochopenie zadania (keď niekto počítal niečo iné, ako bolo v zadaní), za odpisovanie alebo za väčšie chyby v programe.

V tejto úlohe sme od vás chceli, aby ste napísali program, ktorý postupne odsimuluje všetky možné priebehy hry, a tým pádom vypočíta ich počet. Ako urobiť takéto simulovanie? Najľahšie je asi také priamočiare riešenie, že máme jedno pole, v ktorom si pamätáme aktuálny stav plochy a nejakú rekurzívnu funkciu, ktorá bude uskutočňovať ťahy hráčov.

Určite treba vyskúšať všetky možnosti kam dá prvý hráč krížik. No a po každom umiestnení prvého krížika treba vyskúšať všetky možnosti umiestnenia kolieska, a zase po každom umiestnení kolieska vyskúšať všetky možnosti pre ďalší krížik... Asi nemá zmysel takto tu vypísať všetkých 9 ťahov (vnorení), a ani nemusím dodávať, že ak hra z nejakého dôvodu skončí, tak treba prestať.

A tak ako sa to dalo jednoducho vysvetliť, tak isto jednoducho sa to dá aj naprogramovať pomocou rekurzie. Rekurzia znamená, keď funkcia volá samu seba. Vďaka rekurzii môžeme zložité programy alebo úkony (v ktorých sa vlastne robí niekoľkokrát to isté) zapísať jednoducho. Napríklad tak ako som pri vysvetľovaní použil tri bodky, takým sposobom použijem rekurziu. Ak nechcem byť originálny, čo sa pri opravovani cení, použijem štandardnú metódu do-undo, ktorá je založená na tom, že bezhlavo vyskúšam zaradom všetko možné, vnorím sa ďalej, a po vynorení odčiním, čo som napáchal (čo niekedy netreba pri vhodnom prístupe).

Ostáva už len prísť na to ako, zistiť, či niekto nevyhral, stačí vyskúšať všetkých 8 možnosti, kde môžu byt tie tri rovnaké vedľa seba. Čo sa dá s pár ifmi, stačí ich presne 8. Tí lenivejší z vás si to nejako zovšeobecnili a potom potrebovali menej ifov. Namakanejšie riešenie ste mohli získať použitím logiky :), každému hráčovi sa zakódujú pozície jeho figúrok do

binárneho čísla, a keď poznáme správnych 8 vyhravajucich čísel, môžeme pomocou jedného cyklu a nejakých logických operácii zistiť či hráč vyhral... toť vše.

Listing programu:

```
#include<cstdio>
int p[9];
int h=0,w=0;
int vyhra(){
  int rval=0;
  if (p[0]==p[1] && p[1]==p[2]) rval |= p[0];
  if (p[3]==p[4] && p[4]==p[5]) rval |= p[3];
  if (p[6]==p[7] && p[7]==p[8]) rval |= p[6];

  if (p[0]==p[3] && p[3]==p[6]) rval |= p[0];
  if (p[1]==p[4] && p[4]==p[7]) rval |= p[1];
  if (p[2]==p[5] && p[5]==p[8]) rval |= p[2];

  if (p[0]==p[4] && p[4]==p[8]) rval |= p[0];
  if (p[2]==p[4] && p[4]==p[6]) rval |= p[2];

  return rval;
}
int tah=1;
void go(int pl){
  if (vyhra()) {
    w++;
    return;
  }
  if (tah==10) {
    h++;
    return;
  }
  for(int i=0;i<9;i++) if (!p[i]){
    tah++;
    p[i]=pl;
    go(3-pl);
    p[i]=0;
    tah--;
  }

}
int main(){
  go(1);
  printf("%d %d\n",w,h);
}
```

opravoval Miňo
(max. 15 bodov)

4. Zákerná štatistika

Z riešení, ktoré prišli, nebolo bohužiaľ príliš veľa správnych. Veľa z vás spravilo chybu hlavne pri počítaní najväčšej veľkosti radu, keď predpokladali, že stačí zistiť najviac príchodov za sebou – zabudlo sa však na to, že ľudia z fronty postupne aj odchádzajú. Ďalším dosť rozšíreným problémom bol tichý predpoklad, že vstupné pole je už zotriedené. Tu si treba zapamätať, že kým to nie je výslovne napísané v zadaní, rátame s neutriedeným poľom. Keďže ale zadanie bolo v tomto ohľade dosť „chytákové“ a ešte aj vzorový vstup bol utriedený, nebolo to považované za prílišnú chybu.

Ako sa bodovalo? Najlepšie riešenie so zložitosťou rádovo $N \log N$ mohlo dostať maximálne 15 bodov, N^2 riešenia mali hranicu 12 bodov, riešenia používajúce hrubú silu (kontrolovanie udalostí pre každú minútu dňa) boli ohodnotené 8 bodmi. Špeciálnou kategóriou boli nekompletné riešenia, ktorým nefungovalo napríklad práve zistenie dĺžky radu – ohodnotené maximálne 7 bodmi. Za chýbajúci popis išli dole 2 body, za chýbajúci odhad zložitosti alebo za tichý predpoklad, že vstup je utriedený, sa strhával 1 bod. Pokiaľ niekto aspoň napísal, že predpokladá utriedený vstup, bolo to v poriadku.

Ako vyzerá vzorové riešenie? Najskôr si načítame vstup do poľa a utriedime ho – na to sa hodí algoritmus QuickSort: Je to rekurzívny algoritmus (teda, zo svojho vnútra volá sám seba), a jeho parametrom je vždy časť poľa, nad ktorou má pracovať. Najskôr sa vyberie jeden prvok (odborne nazývaný pivot :-), pomocou ktorého budeme zoraďovať prvky. A teraz nasleduje trik: prehádžeme prvky poľa tak, aby na ľavej strane boli všetky prvky menšie alebo rovné ako pivot, na pravej väčšie alebo rovné. Pôjdeme po poli akoby dvoma prstami, jedným zľava a druhým spraa. Ľavý prst posúvame, až kým nenájdeme prvý prvok, ktorý nevyhovuje podmienke (je väčší ako pivot), potom posúvame pravý prst doľava, kým nenájdeme prvok menší ako pivot. Ak sa nám už prsty prekrížili, skončíme, inak tieto dva nájdené prvky vymeníme a pokračujeme ďalej. A čo sme tým dosiahli?

Poznáme miesto, kde sa končí ľavá časť poľa a začína pravá (nazvime ho hranica). Každé číslo z ľavej časti má byť vo výslednom poradí pred každým číslom z pravej časti. Na to, aby sme dostali utriedené pole, stačí samostatne utriediť každú jeho časť. No a na to, aby sme ich zoradili, zavoláme na obe časti takúto istú procedúru. A keď sa bude takto volať, stále na menšiu časť poľa, až kým sa nedostanem po tie najmenšie časti. A tie sa nám, prekvapivo, utriedia samé.

Keď už máme zákazníkov zotriedených, môžeme začať hľadať veci, ktoré potrebujeme. Čas príchodu prvého zistíme triviálne. Ďalšie údaje sú o čosi pracnejšie, preto budeme simulovať, ako počas dňa prichádzajú zákazníci. Konkrétne si budeme pamätať začiatok a koniec fronty počas vybavovania zákazníka na začiatku. Zoberieme si prvého zákazníka a nastavíme naň začiatok fronty. Potom budeme hľadať koniec fronty – všetkých zákazníkov, ktorí prišli počas vybavovania začiatku fronty, pričom zároveň aj aktualizujeme čas začiatku vybavovania posledného zákazníka – vždy pripočítavam čas potrebný na vybavenie predposledného zákazníka vo fronte. Toto robíme až do chvíle, kym nezískame maximálnu frontu pre vybavovaného zákazníka – ďalší zákazníci by prišli až po jeho odchode, takže skontrolujeme maximálny rad a čakanie. Potom posuniemw začiatok fronty na ďalšieho zákazníka a toto celé opakujeme. Pozor si ale treba dať na prípad, kedy počas zákazníkovho vybavovania nikto nepríde. Vtedy máme frontu veľkosti 1, no nesmie sa stať, aby bol koniec fronty pred jej začiatkom, treba ho upraviť spolu so začiatkom fronty. Po prejdení celého poľa máme zozbierané všetky štatistiky – najväčšiu dĺžku radu aj najdlhšie čakanie.

Čo sa týka ideálnej zložitosti – v pamäti potrebujeme mať všetkých N záznamov, aby sme ich mohli triediť, pričom QuickSort ma zložitosť $N \log N$. Vlastné spracovanie poľa má lineárnu zložitosť, čo sa dá dokázať celkom jednoducho: začiatkom aj koncom poľa hýbem len dopredu, nikde ich nemením na 0 ani nezmenšujem, takže každým prejdem N prvkov.

Výsledná časová zložitosť je teda $O(N \log N)$, pamäťová $O(N)$.

Listing programu:

```
const
  maxn = 100;

type TZakaznik = record
  prichod, vybavenie:Integer;
  meno:String[15];
  end;
```

```
var
  n, i, hod, min: Integer;
  zakaznici: array[1..maxn] of TZakaznik;
  zac, kon, maxdlzka, poslZacina, maxcakanie, maxcakajuci: Integer;

procedure napisCas(cas:Integer);
begin  write((cas div 60):2,':', (cas mod 60):2); end;

procedure tried(l, p:Integer);
var
  li, pi, med: Integer;
  tmp: TZakaznik;
begin
  li := l;
  pi := p;
  med := zakaznici[(l+p) div 2].prichod;

  if l>=p then exit;
  while li<pi do
  begin
    while (zakaznici[li].prichod<=med) and (li<p) do Inc(li);
    while (zakaznici[pi].prichod>med) and (pi>l) do Dec(pi);
    if (li<pi) then
    begin
      tmp := zakaznici[li];
      zakaznici[li] := zakaznici[pi];
      zakaznici[pi] := tmp;
    end;
    writeln('partition');
  end;
  tried(l, li-1); tried(li+1, p);
end;

begin
  readln(n);
  for i:=1 to n do
  begin
    read(hod, min, zakaznici[i].vybavenie, zakaznici[i].meno);
    zakaznici[i].prichod := hod*60 + min;
  end;

  tried(1,n); { utriedime si vstup }

  zac:=1; kon:=1; maxdlzka:=0; maxcakanie:=0; poslZacina:=zakaznici[1].prichod;
  while zac<n do
  begin
    if zac > kon then kon:=zac;
    while kon<n do
      if (poslZacina+zakaznici[kon].vybavenie > zakaznici[kon+1].prichod) then
      begin {vybavovania nasho cloveka prisiel do fronty dalsi - zaradime}
Inc(poslZacina, zakaznici[kon].vybavenie); Inc(kon);
      end
      else break;

    if maxdlzka<kon-zac+1 then maxdlzka:=kon-zac+1;
    if poslZacina-zakaznici[kon].prichod > maxcakanie then
    begin { najdlhsie cakanie - rozdiel realneho zaciatku a prichodu }
      maxcakanie:=poslZacina-zakaznici[kon].prichod;
      maxcakajuci:=kon;
    end;
```

```
    Inc(zac);
    if (zac>kon) then
    begin {ak sa zac dostal pred za koniec, musime ho posunut}
      kon:=zac;
      poslZacina:=zakaznici[zac].prichod;
    end;
  end;

  write('Prichod prveho: ');napisCas(zakaznici[1].prichod); writeln;
  write('Odchod posledneho: ');napisCas(poslZacina+zakaznici[n].vybavenie); writeln;
  writeln('Najdlhsi rad: ',maxdlzka);
  writeln('Najdlhsie cakajuci: ', zakaznici[maxcakajuci].meno, ' - ', maxcakanie, '
minut');

end.
```

opravoval Peťo

5. Z koláča dieru

(max. 15 bodov)

Súdiac podľa počtu správnych riešení, musím prehlásiť úlohu za pomerne ľahkú. Občas sa našli nejaké drobné chybičky, najmä pri odhadoch zložitostí, ale inak ste to zvládli celkom dobre.

Ale nikto z vás neposlal úplne správne a najlepšie riešenie. Väčšina z vás našla algoritmus pracujúci v čase $O(N \times M)$, ale pamäte využívate rovnako veľa. Sú však aj riešenia, ktorým si stačí pamätať iba jeden alebo dva riadky vstupu. Existujú napríklad riešenia asymptoticky trochu pomalšie využívajúce *Union-Find Set*. Ale, dá sa nájsť aj riešenie, ktoré má časovú zložitosť $O(N \times M)$ a pamäťovú zložitosť $O(M)$.

To však už bolo spomínané v 16. ročníku KSP medzi „ostrými“ príkladmi a naozaj sa v tejto chvíli nehodí ako vzorové riešenie do kategórie Z. A preto naše vzorové riešenie bude rovnako plytvať pamäťou ako tie vaše ;-). Ešte by som si dovolil poznamenať, že ako ste si iste všetci všimli, tak úloha spadá do teórie grafov. Jednotlivé pozície sú vrcholy a hrany sú vždy medzi susednými pozíciami a našou úlohou je nájsť súvislé komponenty, ale načo to písať tak zložito.

Vzorové riešenie bude využívať algoritmus prehľadávania do šírky, ale poďme pekne od začiatku. Našou úlohou je na každý kus koláča uložiť práve jedno hrozienko. A práve to slovo *práve* nám tu robí problémy. Musíme si dať pozor na to, aby sme na jeden kúsok koláča nedali náhodou dve hrozienka. Náš program bude robiť nasledovné. Postupne prechádzam koláč (to znamená pozerám si ho po znakoch). Ak nájdem voľný kúsoček koláča (znak „.“), tak tam uložím hrozienko. Teraz si však musím označiť celý kus koláča, ku ktorému náš kúsoček patrí. A tu prichádza prehľadávanie do šírky.

Tento algoritmus vie napríklad zistiť či existuje cesta medzi dvoma pozíciami na mape ako je táto. Takisto vie povedať kam všade sa vieme z nejakej pozície dostať, a práve túto vlastnosť využijem. Budem prehľadávať koláč a značiť si všetky kúsočky, na ktoré sa viem dostať z nášho ohrozienkovaného kúsočku tak, že prejdem iba po znakoch „.“ . No a vy už iste tušíte, že ak viem niekam preskákať iba po bodkách, tak to určite patrí do jedného kusu koláča.

A toto naše prehľadávanie funguje úplne jednoducho. Zoberiem si prvý kúsoček, z ktorého chcem prehľadávať a dám si ho do fronty. Fronta je vlastne dáke pole kam budem pridávať kúsočky a časom ich (v poradí, v akom som ich vkladal) odoberať. Konkrétne v tejto fronte budem mať vždy kúsočky, do ktorých sa viem nejako dostať, ale ešte neviem kam sa z nich viem dosatať. Teraz spustím cyklus, ktorý pôjde kým budem mať vo fronte nejaké kúsočky.

Ak tam ešte sú, tak viem, že existujú kúsočky, z ktorých neviem, kam sa viem dostať. A teda zoberiem nasledujúci kúsoček z fronty a pozrem sa kam sa viem z neho dostať. Ak sa viem dostať niekam, kde som ešte nebol, tak si to miesto označím, že som tam už bol a prihodím ho do fronty. Teda každý kúsoček z nášho kusu bude označený ako už navštívený a okrem toho sa pozriem, kam sa z neho viem dostať. Takto prehľadám všetky kúsočky nášho kusu.

No a poďme hneď aj na odhad zložitostí. Začnem tou jednoduchšou stránkou a to je pamäťová. V podstate mám dve veľké polia. Jedno je samotný koláč, ktorý musí byť veľký $N \times M$. Druhé je fronta. Najhorší prípad pre frontu je, keď v jednom prehľadaní pozriem celý koláč (teda koláč obsahoval iba znaky „.“). Vtedy musím mať vo fronte $N \times M$ znakov. Teda obe polia sú rovnako veľké a pamäťová zložitosť je $O(N \times M)$. Pri odhadovaní časovej zložitosti sa musím pozrieť na to, že čo robím koľkokrát. Konkrétne v tomto prípade najviackrát asi kontrolujem, ako mám označené jednotlivé kúsočky. Túto kontrolu robím asi päťkrát. A to konkrétne keď pozerám celý koláč a potom počas všetkých prehľadávaní sa na každý kúsoček pozriem len štyrikrát a to keď pozerám zhora, zdola, zľava a zprava. Teda dokopy urobím asi len $5 \times (N \times M)$ pozretí. Načítanie vstupu a výstupu má tiež iba $K \times (N \times M)$ kde K je iba nejaká trápna konštanta. Teda celková časová zložitosť je $O(N \times M)$.

Na záver už len pár slov k hodnoteniu. Za riešenia v čase $O(N \times M)$ ste mali možnosť získať 15 bodov. Nejaké tie bodíky sa strhávali za zlý popis, čiastočne zlý popis, nesprávny odhad zložitostí prípadne nefunkčnosť.

- $O(N \times M)$: 15
- $O((N \times M)^2)$: 9
- tradične za slabý až žiadny popis: -1 až -2
- za nesprávny alebo žiadny odhad zložitosti: -1

Listing programu:

```
const maxM=40;
      maxN=20;

type kusok=record
  x,y:integer;
end;

var n,m,i,j,k,zac,kon,x,y:integer;
  kolac:array[0..maxN+1,0..maxM+1] of char;{mapa kolaca}
  fronta:array[0..maxN*maxM] of kusok;{fronta na prehladavanie}
  okolo:array[0..3,0..1] of integer= {pole so suradnicami doprava dolava dole a hore}
  ((1,0),(-1,0),(0,1),(0,-1));
begin
  readln(n,m);
  for j:=0 to m+1 do begin
    kolac[0,j]:='$';
    kolac[n+1,j]:='$';
  end;
  for i:=1 to n do begin
    kolac[i,0]:='$';
    kolac[i,m+1]:='$';
    for j:=1 to m do read(kolac[i,j]);
    readln;
  end;
  {nacitanie a pripravenie mapy je hotove a teraz samotny program:}
  for i:=1 to n do
    for j:=1 to m do
      if(kolac[i,j]='.')then begin
      {nasiel som neoznaceni kusocek teda zaciatok dalsieho kusu}
        kolac[i,j]:='*';
```

```
          fronta[0].x:=i;
          fronta[0].y:=j;
          zac:=0;
          kon:=1;
          {prehladavam az kym frontanik nie je prazdny}
          while(zac<kon)do begin
            x:=fronta[zac].x;
            y:=fronta[zac].y;
            {mozny  sposob ako pozerat pozicie okolo seba}
            {v poli okolo mam ulozene konstanty kam sa chcem 'pozret'}
            for k:=0 to 3 do
              if kolac[x+okolo[k,0],y+okolo[k,1]]='.' then begin
              {nasiel som novy kusocek patriaci mojmu kusu
              tak si ho oznacim a hodim do frontanika:}
                kolac[x+okolo[k,0],y+okolo[k,1]]:='X';
                fronta[kon].x:=x+okolo[k,0];
                fronta[kon].y:=y+okolo[k,1];
                inc(kon);
              end;
            inc(zac);
          end;
        end;

  for i:=1 to n do begin
    for j:=1 to m do
      if kolac[i,j]='X' then write('.')
      else write(kolac[i,j]);
    writeln;
  end;
end.
```

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Danilák Michal	Gym. Hubeného BA	3	13	10	14	15	15	67
2	Košinárová Alena	Gym. Grösslingová BA	2	15	15	7	13	15	65
3	Szabados Michal	Škola pre mim. nadané deti BA	3	13	12	14	15	8	62
4	Hapák Samuel	Gym. Grösslingová BA	2	13	13	4	15	15	60
5	Kucharík Marcel	Gym. Športová Nové Mesto n.V.	4	12	10	10	13	14	59
5	Sucha Martin	Gym. Jura Hronca BA	2	13	10	7	15	14	59
7	Petrucha Michal	Gym. Metodova BA	1	13		14	14	15	56
8	Kočiský Tomáš	Gym. Grösslingová BA	2	13	8		15	14	50
9	Novák Ján	Gym. Ľudovíta Štúra Trenčín	4	12	10	11		14	47
10	Fecko Stanislav	Gym. Pankúchova Bratislava	3	8	9		15	14	46
11	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	2	13	6	5	7	14	45
11	Mészároš Roman	SPŠ Levice	3	9	8	9	15	4	45
13	Beno Miroslav	Gym. Jura Hronca BA	4	9	9	11		14	43
14	Baumann Martin	Gym. Jura Hronca BA	3	13	11	4		14	42
14	Hlavatý Peter	Gym. Jura Hronca BA	2	8	8	13		13	42
14	Nikodem Peter	Gym. Školská Spiš. Nová Ves	2	13	9	5		15	42
14	Saleh Adam	Gym. Jura Hronca BA	2	12	10	8		12	42
14	Švec Marcel	Gym. Jura Hronca BA	3	13	12	3		14	42
19	Kostolányi Peter	Gym. Jura Hronca BA	2	9	10	11	2	9	41
20	Mego Marek	Gym. Jura Hronca BA	4	9	10	5	5	11	40
20	Nagy Kristián	Gym. Jura Hronca BA	1	8	7	9	2	14	40
20	Novacek Milos	Gym. Jura Hronca BA	4	9	8	9		14	40
23	Synak Peter	Gym. Jura Hronca BA	2	13	9	3		14	39
24	Holla Kamila	Gym. Jura Hronca BA	4	9	9		4	14	36
25	Janošík Jozef	Gym. Varšavská Žilina - Vlčince	1	8	9	4		14	35
25	Orihel Lukas	Gym. Jura Hronca BA	4	9	9	4		13	35
25	Schiffer Juraj	Gym. Jura Hronca BA	4	8	10		5	12	35
28	Fedáková Dominika	Gym. Stará Ľubovňa	2	14	10	10			34
28	Trančík Ivan	Gym. Jura Hronca BA	2	13	7			14	34
30	Jakabovič Juraj	Gym. Jura Hronca BA	2	9	11			13	33
30	Kvasnička Igor	Gym. Jura Hronca BA	2	6	6	8		13	33
32	Košdy Martin	Gym. Jura Hronca BA	1	11		7		14	32
33	Bachratý Martin	ZŠ Gaštanová Žilina	0	8	10	8	5		31
33	Hanes Filip	Gym. Tajovského B. Bystrica	3	13	11	7			31
33	Takács Michal	Gym. Tajovského B. Bystrica	4	9	10	6		6	31
33	Valenčík Ivan	Gym. Ul. L. Sáru BA	3	9	10	12			31
37	Hreha Ján	Gym. Liptovský Hrádok	3	9			15	3	27
37	Korcsok Peter	Gym. Mládežnícka Šahy	2	9	5	13			27
37	Vojtko Martin	Gym. Jura Hronca BA	3	7	7			13	27
40	Zajacová Danica	Gym. Jura Hronca BA	3	15				11	26
41	Betík Roman	SPŠ Levice	4	9	15				24
41	Lachata Adrián	Gym. Svidník	4	9	6	5	4		24
41	Móro Róbert	Gym. Ul. L. Sáru BA	3	9	10	5			24
44	Dullová Veronika	Gym. Ul. L. Sáru BA	3	9	13				22
45	Bašista Peter	Gym. P. Horova Michalovce	4	8	9	4			21
45	Berthoty Adam	Gym. Ľudovíta Štúra Trenčín	4	13	8				21
45	Hlaváček Filip	Gym. Hubeného BA	4	11	10				21
45	Hojčková Martina	Gym. Jura Hronca BA	3	7	10	4			21
49	Langer Lukáš	Súkromná SOŠ Humanus Via	2	7	9	4			20
49	Popovič Viktor	Gym. Mudroňova Prešov	0	9	11				20

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
51	Beňo Fratišek	SPŠ Humenné	2	8	7	4			19
51	Fojtík Michal	Gymnázium	4	13	6				19
51	Lacko Michal	Gym. Veľký Krtíš	4	8	7			4	19
51	Šmeringai Peter	Neznama skola	3	9	10				19
55	Ihnát Peter	Gym. Sečovce	2	9	9				18
55	Koval Michal	Gym. Jura Hronca BA	2	9	9				18
55	Kundis Tomáš	SOU, Kukučínova 483, Poprad	3	9	9				18
55	Kurpel Matej	Gym. Ľudovíta Štúra Trenčín	4	9	9				18
55	Molčan Dávid	Gym. Slovenská Bardejov	3	9	9				18
60	Doležal Martin	Gym. Alejová Košice	2	8	9				17
60	Kučin Alexander	Gym. Lipany	2	9	8				17
62	Kovács Vince	Gym. Štúrovo	2	8	6	2			16
62	Šuranyi Peter	Gym. Jura Hronca BA	4	7	9				16
64	Kukan Matúš	Gym. Grösslingová BA	2	13					13
64	Morvay Peter	Gym. Jura Hronca BA	3	13					13
66	Hattas Marek	SPŠ Nitra	3	8		4			12
67	Buchovecká Simona	Gym. Medzilaborce	3	8	2	0	0	0	10
67	Rybár Michal	Gym. sv. Uršule BA	3	8	2				10
69	Gálik Marian	Gym. Jura Hronca BA	2	9					9
69	Godány Martin	Škola pre mim. nadané deti BA	3	9					9
69	Kelemeca Patrik	SPŠ Snina	3	9					9
69	Šrámek Radoslav	Gym. Jura Hronca BA	3		9				9
69	Vnuk Marek	Gym. Jura Hronca BA	2	9					9
74	Sember Matej	Gym. sv. Uršule BA	2	7	1				8
75	Lesnák Stefan	Gym. Jura Hronca BA	2	6					6
76	Schlosáriková Lucia	Gym. P. de Coubertina Piešťany	3		5				5