## Olimpijada znanja 2014. – programiranje

### Zadatak 1 – Gusjenice (1 sec, 256 MB)

Po ploči sastavljenoj od N x N jediničnih kvadratića gmižu gusjenice. Svaka gusjenica zauzima niz od **bar dva** kvadratića, takav da svaka dva uzastopna kvadratića imaju **zajedničku stranicu**. Prvi vadratić u nizu nazivamo **glava**. Na svakom kvadratiću ploče se može nalaziti maksimalno jedna gusjenica. Kvadratići koje na početku ne zauzimaju gusjenice mogu biti ili prazni ili se u njima može nalaziti prepreka. Gusjenica gmiže tako da, u svakom koraku, najprije pomjeri glavu na neko **prazno susjedno** polje, a zatim povuče rep ispraznivši tako jedno polje. Smjer kretanja je na početku određen položajem glave u odnosu na drugi po redu kvadratić koji čini gusjenicu. U svakom koraku gusjenica se ponaša prema sljedećim pravilima:

- ako se može pomjeriti naprijed (i ne sudara se sa preprekom, drugom gusjenicom, samom sobom ili izlazi van granica ploče), onda se pomjeri naprijed;
- ako je to nemoguće onda **pokuša skrenuti udesno**;
- ako je to nemoguće onda **pokuša skrenuti ulijevo**;
- ako je to nemoguće onda **ostaje stajati na mjestu**, pa se u sljedećem koraku opet pokušava pomjeriti naprijed.

Na ploči se nalazi nekoliko gusjenica označenih slovima engleske abecede. U svakom koraku sve se pokušavaju se pomjeriti prema gore opisanim pravilima i to **abecednim redosljedom**. Svaki korak traje točno jednu sekundu. Napišite program koji će, za dati početni položaj gusjenica na ploči, utvrditi njihov položaj nakon T sekundi. **Ulaz:** U prvom redu ulaza nalaze se cijeli brojevi N i T, $2 \leq N \leq 1\,000$, $1 \leq T \leq 1\,000\,000$. Broj N opisuje dimenziju ploče, a T je broj sekundi nakon kojih treba utvrditi položaj gusjenica. U svakom od sljedećih N redova nalazi se po N znakova; ti redovi opisuju početni položaj gusjenica, praznih kvadratića i prepreka na ploči. Preciznije, svaki znak može biti: `'.'` (tačka) – odgovarajući kvadratić je prazan; `'#'` – prepreka; veliko slovo engleske abecede – glava gusjenice; malo slovo engleske abecede – dio gusjenice koji nije glava. Svi kvadratići u kojima se nalazi isto slovo abecede (bilo veliko ili malo) čine jednu gusjenicu. Ulazni podaci su takvi da svaki kvadratić koji čini neku gusjenicu biće susjedan **sa tačno dva** druga njena kvadratića, osim glave i vrha repa koji će biti susjedni **tačno jednom** njenom kvadratiću. Naravno, neće postojati dvije različite gusjenice označene istim slovom. **Izlaz**: Potrebno je štampati N redova, svaki sa po N znakova. Ti redovi trebaju opisivati položaj gusjenica po isteku T sekundi, u istom formatu kako je položaj bio opisan u ulaznim podacima.

| Primjer 1 | | Primjer 2 | |
|---|---|---|---|
| **Ulaz** | **Izlaz** | **Ulaz** | **Izlaz** |
| 4 8<br>.bB.<br>....<br>a.#.<br>aaA. | .baa<br>.BAa<br>..#.<br>.... | 7 100000<br>aA.....<br>a#####D<br>aa....d<br>.....d<br>.....d<br>.....d<br>..... | aaaaADd<br>.#####d<br>.....d<br>.....d<br>.....<br>.....<br>..... |

**Rješenje**:

Zadatak se rješava direktnom simulacijom kretanja gusjenica kako je opisano u tekstu. Na početku odredimo sve gusjenice tako da, počevši od svakog velikog slova (koje predstavlja glavu), redom tražimo susjedna odgovarajuća mala slova. Svaku gusjenicu možemo sačuvati u memoriji kao listu koordinata kvadratića koje zauzima, redom od glave do repa. Simulaciju kretanja vršimo tako da, za svaku gusjenicu, promijenimo listu u kojoj su sačuvane njene koordinate, i istovremeno promjenimo polja tako da predstavljaju novu poziciju. Za jedan potez u jednom smjeru potrebno je:

- Provjeriti da li je slobodno polje na koje se glava treba pomjeriti.
- Pomjeriti glavu tako da dodamo nove koordinate na početak liste i označimo odgovarajuće polje na ploči kao zauzeto.
- Pomjeriti rep tako da izbrišemo koordinate sa kraja liste i označimo odgovarajuće polje kao slobodno.

## Olimpijada znanja 2014. – programiranje

Da bi rješenje bilo dovoljno brzo, potrebno je vrlo efikasno u listu dodati nove elemente na početak, te izbrisati stare sa kraja. Ovakva struktura podataka se lako implementira pomoću olančane liste ili cirkularnog niza ili strukture dequeue. Vremenska složenost ovakve simulacije je O(broj_kvadratića_na_tabli + vrijeme * broj_gusjenica), gdje prvi dio odgovara učitavanju ulaznih podataka, inicijalizaciji (traženje gusjenica po ploči) i štampanju rješenja, dok drugi dio odgovara samoj simulaciji.

```
#include <cstdio>
#include <cctype>
#include <queue>

using namespace std;

const int MAXN = 1000;
const int MAXS = 26;
const int DX[4] = {-1, 0, 1, 0};
const int DY[4] = {0, 1, 0, -1};
const int AT[3] = {0, 1, -1};
const char EMPTY = '.';

int n, t;
char g[MAXN][MAXN+1];

int s;
int dir[MAXS];
char name[MAXS];
deque<int> sx[MAXS];
deque<int> sy[MAXS];

void read_input() {
  scanf("%d%d", &n, &t);
  for (int i=0; i<n; i++)
    scanf("%s", g[i]);
}

void find_snake(int k, int x, int y) {
  int done = 0;
  int first = 1;
  int lastx = -1;
  int lasty = -1;
  name[k] = tolower(g[x][y]);
  while (!done) {
    sx[k].push_back(x);
    sy[k].push_back(y);
    done = 1;
    for (int i=0; i<4; i++) {
      int xx = x+DX[i];
      int yy = y+DY[i];
      if (xx < 0 || xx >= n || yy < 0 || yy >=n)
        continue ;
      if (g[xx][yy] != tolower(g[x][y]))
        continue ;
      if (xx == lastx && yy == lasty)
        continue ;
      done = 0;
      lastx = x;
      lasty = y;
      x = xx;
      y = yy;
      if (first)
```

```
        dir[k] = (i+2)%4;
      first = 0;
      break ;
    }
  }
}

void find_snakes() {
  s = 0;
  for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
      if (isupper(g[i][j]))
          find_snake(s++, i, j);
}

void move_snake(int k) {
  for (int i=0; i<3; i++) {
    int newdir = (dir[k]+AT[i]+4)%4;
    int xx = sx[k].front()+DX[newdir];
    int yy = sy[k].front()+DY[newdir];
    if (xx < 0 || xx >= n || yy < 0 || yy >=n)
      continue ;
    if (g[xx][yy] != EMPTY)
      continue ;
    dir[k] = newdir;
    // Head
    g[sx[k].front()][sy[k].front()] = name[k];
    g[xx][yy] = toupper(name[k]);
    sx[k].push_front(xx);
    sy[k].push_front(yy);
    // Tail
    g[sx[k].back()][sy[k].back()] = EMPTY;
    sx[k].pop_back();
    sy[k].pop_back();
    break ;
  }
}

void move_all(void) {
  int order[s];
  for (int i=0; i<s; i++) {
    int cnt = 0;
    for (int j=0; j<s; j++)
      if (name[j] < name[i])
        cnt++;
    order[cnt] = i;
  }

  for (int i=0; i<t; i++)
    for (int j=0; j<s; j++)
      move_snake(order[j]);
}

void output() {
  for (int i=0; i<n; i++)
    printf("%s\n", g[i]);
}

int main() {
```

```
  read_input();
  find_snakes();
  move_all();
  output();
  return 0;
}
```

**Zadatak 2 – Princ (2 sec, 64MB)**

Princ Persije nalazi se na najvišem nivou podzemnog lavirinta koji je osmislio Džafar. Lavirint se sastoji od h nivoa koji se nalaze jedan ispod drugog. Svaki nivo je pravougaonik podijeljen na m×n dijelova. Neki od dijelova sadrže stubove koji drže plafon i po takvim dijelovima Princ ne može da se kreće. Princ se može pomjerati sa jednog dijela na drugi ako ta dva dijela imaju zajedničku stranicu i nijedan od dijelova nije stub. Takvo pomjeranje traje 5 sekundi. Podovi su veoma tanki, pa Princ udarcem nogom može probiti pod i pasti za jedan nivo ispod, ne premještajući se u horizontalnoj ravni. Takva operacija takođe traje 5 sekundi. Ako je Princ na najnižem nivou, ne može slomiti pod. Na jednom od dijelova na najnižem nivou Princa čeka Princeza. Pomozite Princu da za najkraće moguće vrijeme dođe do Princeze.

**Ulaz**: Prvi red sadrži cijele brojeve h, m i n – visinu i horizontalne dimenzije lavirinta ($2 \leq h, m, n \leq 50$). Zatim je dato h blokova koji opisuju nivoe lavirinta, od najvišeg ka najnižem. Svaki blok sadrži m redova sa po n simbola: '.' (tačka) označava slobodni dio, 'o ' (malo slovo o) označava dio sa stubom, '1 ' označava dio u kojem se na početku nalazi Princ, '2' označava dio gdje se nalazi Princeza. Simboli 1 i 2 pojavljuju se tačno jednom: simbol 1 na najvišem nivou a simbol 2 na najnižem nivou. Susjedni blokovi su razdvojeni jednim praznim redom.

**Izlaz**: Štampati jedan cio broj – minimalni broj sekundi potrebnih Princu da nađe Princezu. Garantuje se da Princ može ostvariti zadatak (jer dobro uvijek pobjeđuje).

**Primjer**:

| Ulaz | Izlaz |
|---|---|
| 3 3 3<br>1..<br>oo.<br>...<br><br>ooo<br>..o<br>.oo<br><br>ooo<br>o..<br>o.2 | 60 |

**Rješenje**: Čvorovi grafa su dijelovi lavirinta a grane postoje ako su dva dijela susjedna na istom nivou ili na susjednim nivoima. Sada se zadatak svodi na klasično traženje u grafu primjenom BFS-a ili DFS-a. Čak nije potrebno ni kreirati graf, već se sve može odraditi na trodimenzionalnom nizu koji predstavlja lavirint.

```
#include <stdio.h>

#define PROBLEM_ID          "maze"
#define MAXN                50
#define INF                 1000000

char f[MAXN][MAXN][MAXN];
int ans[MAXN][MAXN][MAXN];
int q[MAXN * MAXN * MAXN];
char mark[MAXN][MAXN][MAXN];

int dx[] = { -1,  0,  0,  1, 0 };
int dy[] = {  0, -1,  1,  0, 0 };
```

**Olimpijada znanja 2014. – programiranje**

```
int dz[] = {  0,  0,  0,  0, 1 };

int main() {
    int x, y, z, xx, yy, zz;
    int h, n, m;
    int i, j, k;
    int low, hi;
    int cur;

    freopen(PROBLEM_ID".in", "r", stdin);
    freopen(PROBLEM_ID".out", "w", stdout);

    scanf("%d %d %d", &h, &n, &m);
    for (i = 0; i < h; i++) {
        for (j = 0; j < n; j++) {
            for (k = 0; k < m; k++) {
                char ch = getc(stdin);
                while (ch != '.' && ch != '1' && ch != '2' && ch != 'o')
                    ch = getc(stdin);

                f[i][j][k] = ch;
            }
        }
    }

    low = 0;
    hi = 0;

    for (i = 0; i < h; i++) {
        for (j = 0; j < n; j++) {
            for (k = 0; k < m; k++) {
                cur = (i << 16) + (j << 8) + k;
                ans[i][j][k] = INF;
                mark[i][j][k] = 0;
                if (f[i][j][k] == '1') {
                    ans[i][j][k] = 0;
                    q[hi++] = cur;
                    mark[i][j][k] = 1;
                }
            }
        }
    }

    while (low < hi) {
        cur = q[low++];
        y = cur & 255;
        x = (cur >> 8) & 255;
        z = (cur >> 16) & 255;

        for (i = 0; i < 5; i++) {
            xx = x + dx[i];
            yy = y + dy[i];
            zz = z + dz[i];
            if (xx < 0 || xx >= n || yy < 0 || yy >= m || zz < 0 || zz >= h)
                continue;

            if (f[zz][xx][yy] == 'o' || mark[zz][xx][yy])
                continue;
```

```
            q[hi++] = (zz << 16) + (xx << 8) + (yy);
            ans[zz][xx][yy] = ans[z][x][y] + 1;
            mark[zz][xx][yy] = 1;
            if (f[zz][xx][yy] == '2') {
                printf("%d\n", 5 * ans[zz][xx][yy]);
                return 0;
            }
        }
    }

    return 0;
}
```

**Zadatak 3 – Grafika (2 sec, 256 MB)**

Potrebno je napraviti grafički prostorni prikaz gomile poslaganih kockica pomoću ASCII simbola. Kocke su pravilno složene u M redova i N kolona, a na nekim kockama se nalazi jedna ili više drugih kocaka koje na taj način formiraju tornjeve. Jednu kocku prikazujemo pomoću znakova '**+**' (**plus**), '**-**' (**minus**), '**|**' (**vertikalna crta**), '**/**' (**kosa crta**) i ' ' (**razmak**) u **6 redova** i **7 kolona** na sljedeći način.

```
  +---+
 /   /|
+---+ |
|   | +
|   |/
+---+
```

Neke kocke **zaklanjaju** druge kocke koje su onda **djelomično ili potpuno sakrivene**. Napišite program koji će odrediti grafički prikaz cijele zadate konfiguracije, koristeći pri tome **što je moguće manje redova i kolona**. Prazna polja označite simbolom '**.**' (**tačka**). **Ulaz**: U prvom redu nalaze se prirodni brojevi M i N, $1 \leq M,N \leq 50$. U svakom od sljedećih M redova nalazi se N prirodnih brojeva. Svaki od njih je manji od ili jednak 50, a označava visinu tj. ukupni broj naslaganih kockica na toj poziciji. Redovi su označeni redom brojevima od 1 do M tako da **red broj 1 označava najudaljeniji**, a **red broj M najbliži red** na grafičkom prikazu. **Kolone su označene** redom brojevima od 1 do N, **slijeva nadesno**. **Izlaz**: Štampati grafički prikaz zadatih kocaka u prostoru kako je opisano u tekstu zadatka.

**Primjer 1**

**Ulaz**

```
3 1
2
1
3
```

**Izlaz**

```
......+---+
..+---+  /|
./   /|-+ |
+---+ | | +
|   | + |/|
|   |/|-+ |
+---+ |/| +
|   | + |/.
|   |/| +..
+---+ |/...
|   | +....
|   |/.....
+---+......
```

**Primjer 2**

**Ulaz**

```
3 3
2 3 2
1 2 1
1 1 1
```

**Izlaz**

```
..........+---+....
........./   /|....
........+---+ |....
......+-|   | +---+
...../  |   |/   /|
....+---+---+---+ |
....|  /   /|   | +
....| +---+ |   |/|
....+-|   | +---+ |
.../  |   |/   /| +
..+---+---+---+ |/.
./   /   /   /| +..
+---+---+---+ |/...
|   |   |   | +....
|   |   |   |/.....
+---+---+---+......
```

**Rješenje:**

Zadatak rješavamo tako da crtamo kocku po kocku, pazeći pri tome da su prije crtanja određene kocke već nacrtane sve kocke koje ona možda zaklanja. Jedan način da to osiguramo jeste da crtamo red po red od

najdaljeg prema najbližem. U redu crtamo tornjeve slijeva na desno, a svaki toranj odozdo prema gore. Širina i visina štampe se lako izračunava iz dimenzija i sadržaja matrice.

```
import java.util.Scanner;

public class Kocka {

	final static int REDOVA_U_STRANICI = 4;
	final static int KOLONA_U_STRANICI = 5;
	final static int DUBINA_KOCKE = 2;
	final static int MAX_DIMENZIJE = 50;

	String [] prikaz_kocke = {
		"..+---+",
		"./   /|",
		"+---+ |",
		"|   | +",
		"|   |/.",
		"+---+.."
	 };

	int broj_redova,broj_kolona;
	int [][] visine; // = new int [MAX_DIMENZIJE][MAX_DIMENZIJE];

	int redova_u_rjesenju,kolona_u_rjesenju;
	char [][] rjesenje = new char[500][500];

	void ucitaj()
	{
	  int i,j;
	  int temp_redova_u_rjesenju;
	  Scanner in = new Scanner(System.in);
	  broj_redova = in.nextInt();
	  broj_kolona = in.nextInt();
	  visine = new int [broj_redova][broj_kolona];
	  kolona_u_rjesenju = broj_kolona * (KOLONA_U_STRANICI - 1) + 1 +
	                      broj_redova * DUBINA_KOCKE;

	  for (i = 0;i < broj_redova;++i)
	    for (j = 0;j < broj_kolona;++j)
	    {
	      visine[i][j] = in.nextInt();
	      temp_redova_u_rjesenju = (broj_redova - i - 1) * DUBINA_KOCKE +
visine[i][j] * (REDOVA_U_STRANICI - 1) + 1 + DUBINA_KOCKE;
	      if (temp_redova_u_rjesenju > redova_u_rjesenju)
	        redova_u_rjesenju = temp_redova_u_rjesenju;
	    }
	}

	void crtaj(int redak,int stupac)
	{
	  int i,j;

	  for (i = 0;i < REDOVA_U_STRANICI + DUBINA_KOCKE;++i)
	    for (j = 0;j < KOLONA_U_STRANICI + DUBINA_KOCKE;++j)
	      if (prikaz_kocke[i].charAt(j) != '.')
	        rjesenje[redak + i][stupac + j] = prikaz_kocke[i].charAt(j);
	}
```

**Olimpijada znanja 2014. – programiranje**

```
      void rijesi()
      {
        int i,j,k;
        int red,kolona;

        for (i = 0;i < redova_u_rjesenju;++i)
          for (j = 0;j < kolona_u_rjesenju;++j)
            rjesenje[i][j] = '.';

        for (i = 0;i < broj_redova;++i)
          for (j = 0;j < broj_kolona;++j)
            for (k = 1;k <= visine[i][j];++k)
            {
              red = redova_u_rjesenju -
                      ((broj_redova - i - 1) * DUBINA_KOCKE + (k - 1) *
(REDOVA_U_STRANICI - 1) + REDOVA_U_STRANICI + DUBINA_KOCKE);
              kolona = j * (KOLONA_U_STRANICI - 1) + (broj_redova - i - 1) *
DUBINA_KOCKE;
              crtaj(red,kolona);
            }
      }

      void stampa()
      {
        int i,j;

        for (i = 0;i < redova_u_rjesenju;++i)
        {
          //rjesenje[i][kolona_u_rjesenju] = '\0';
            for (j = 0;j < kolona_u_rjesenju;++j)
                  System.out.printf("%c",rjesenje[i][j]);
            System.out.println();
        }
      }

      public static void main(String[] args) {
            Kocka ko = new Kocka();
            ko.ucitaj();
            ko.rijesi();
            ko.stampa();
      }

}
```