Strukture podataka i algoritmi 2
Avgust, 2012

1. Mapa zamka je data u vidu matrice dimenzija *MxN*, tako što su sobe obeležene nulama, a zidovi jedinicama.

a. Napraviti mapu zamka u vidu stabla, tako da svaki čvor u stablu predstavlja jednu sobu. Potomci nekog čvora predstavljaju sobe u koje se iz tog čvora može stići kretanjem levo desno, gore ili dole. Ukoliko do neke sobe već postoji put kroz stablo, svaki sledeći pronađen put do nje treba zanemariti. Za početni čvor izabrati proizvoljnu sobu.
(10 poena)

b. Napraviti funkciju *Pronadji* koja vraća pokazivač na zadatu sobu ukoliko do nje postoji put.
(10 poena)

c. Napraviti funkciju *Izlaz* koja štampa put kojim se iz zadate sobe stiže do izlaza (početne sobe).
(štampanje soba 10 poena, štampanje koraka (levo,desno, gore, dole) 15 poena)

U nastavku je dato jedno od rešenja studenata.

Rešenje
(Darko Antonijević)

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

#define N 100
#define M 100
#define DESNO 0
#define DOLE 1
#define LEVO 2
#define GORE 3


typedef struct s
{
	int x, y;
	struct s *p[4];
} soba;

void ucitajMatricu(FILE *f, int mat[N][M], int *n, int *m)
{
	int i, j;

	fscanf(f, "%d%d", n, m);

	for(i = 0; i < *n; i++)
		for(j = 0; j < *m; j++)
			fscanf(f, "%d", &mat[i][j]);
}


soba *novaSoba(int x, int y)
{
	soba *nova;
	int i;

	nova = (soba *)malloc(sizeof(soba));
	if(nova == NULL)
	{
		printf("Greska pri alociranju memorije.\n");
```

```
		exit(0);
	}

	nova->x = x;
	nova->y = y;

	for(i = 0; i < 4; i++)
		nova->p[i] = NULL;

	return(nova);
}

soba *prvaSoba(int mat[N][M], int n, int m)
{
	int i, j;
	for(i = 0; i < n; i++)
		for(j = 0; j < m; j++)
			if(mat[i][j] == 0) return(novaSoba(i, j));
}

int ista(soba *t, int x, int y)
{
	if(t->x == x && t->y == y)
		return 1;

	return 0;
}

soba *pronadji(soba *k, int x, int y)
{
	soba *pom = k, *temp;
	int i;

	if(pom != NULL)
	{
		if(ista(pom, x, y))
			return pom;

		for(i = 0; i < 4; i++)
		{
			temp = pronadji(pom->p[i], x, y);
			if(temp != NULL)
				return temp;
		}

	}

	return NULL;
}


soba *sledeca(soba *koren, soba *t, int smer, int mat[N][M], int n, int m)
{
	int x, y;

	soba *pom = NULL;

	x = t->x;
	y = t->y;

	switch (smer)
	{
		case DESNO:
		{
			if(y+1 < m && mat[x][y+1] == 0)
				if(pronadji(koren, x, y+1) == NULL)
					pom = novaSoba(x, y+1);
```

```
                        break;
                }
                case DOLE:
                {
                        if(x+1 < n && mat[x+1][y] == 0)
                                if(pronadji(koren, x+1, y) == NULL)
                                        pom = novaSoba(x+1, y);
                        break;
                }
                case LEVO:
                {
                        if(y-1 >= 0 && mat[x][y-1] == 0)
                                if(pronadji(koren, x, y-1) == NULL)
                                        pom = novaSoba(x, y-1);
                        break;
                }
                case GORE:
                {
                        if(x-1 >= 0 && mat[x-1][y] == 0)
                                if(pronadji(koren, x-1, y) == NULL)
                                        pom = novaSoba(x-1, y);
                        break;
                }
                default:
                        return NULL;
        }

        return pom;
}


void dodeliSobe(soba *koren, soba *t, int mat[N][M], int n, int m)
{
        int i;
        if(t != NULL)
        {

                for(i = 0; i < 4; i++)
                {
                        t->p[i] = sledeca(koren, t, i, mat, n, m);
                        dodeliSobe(koren, t->p[i], mat, n, m);
                }
        }
}

soba *mapa(int mat[N][M], int n, int m)
{
        soba *t;

        t = prvaSoba(mat, n, m);

        if(t != NULL)
                dodeliSobe(t, t, mat, n, m);

        return t;
}

soba *izlaz(soba *k, int x, int y)
{
        soba *pom = k, *temp;
        int i;

        if(pom != NULL)
        {
                if(ista(pom, x, y))
                {
                        return pom;
```

```
		}

		for(i = 0; i < 4; i++)
		{
			temp = izlaz(pom->p[i], x, y);
			if(temp != NULL)
			{
				switch(i)
				{
					case GORE:
					{
						printf("dole\n");
						return temp;
						break;
					}
					case DOLE:
					{
						printf("gore\n");
						return temp;
						break;
					}
					case LEVO:
					{
						printf("desno\n");
						return temp;
						break;
					}
					case DESNO:
					{
						printf("levo\n");
						return temp;
						break;
					}
				}
			}
		}
	}

	return NULL;
}

void stampaj(soba *t)
{
	int i;
	if(t != NULL)
	{
		printf("%d\t%d\n", t->x, t->y);
		for(i = 0; i < 4; i++)
		{
			stampaj(t->p[i]);
		}
	}
}

int main()
{
	soba *koren;
	int n, m, i, j;
	int mat[N][M];
	FILE *f;

	f = fopen("mapa.txt", "r");
	if(f == NULL)
	{
		printf("Greska prilikom otvaranja fajla!\n");
		exit(0);
	}
```

```
        ucitajMatricu(f, mat, &n, &m);

        fclose(f);

        koren = mapa(mat, n, m);

        izlaz(koren, 4, 6);

        return;
}
```