**Seminar 10**

**Grafuri. Reprezentări. Metode de parcurgere.**

**Probleme propuse și rezolvate**

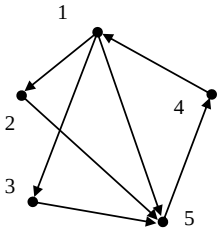
1. **Fie graful *G*=(*V*,*E*) graf, cu *V*={1,2,3,4,5,6}, *E*={(1,2),(1,3),(2,5),(2,6),(3,5),(3,6),(5,6)}. Care este matricea de adiacenţă a grafului?**

Raspuns: x`x`

2. **Fie *D*=(*V*,*E*) digraf, *V*={1,…,5}, *E*={(1,2), (1,3), (2,5), (3,5), (4,1), (5,1), (5,3), (5,4)}. Care este matricea de adiacenţă a digrafului?**

Raspuns: $A=\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$

3. **Fie *D*=(*V*,*E*) digraf, *V*={1,…,5}, *E*={(1,2), (1,3), (1,5), (2,5), (3,5), (4,1), (5,4)}. Digraful poate fi reprezentat grafic astfel,**




**Care este reprezentarea tabelară?**

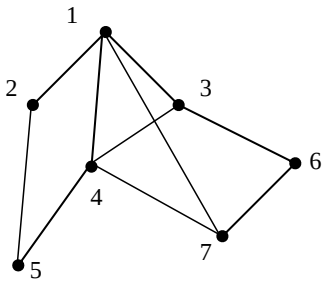
Raspuns: Digraful este reprezentat prin $A=\begin{pmatrix} 1 & 1 & 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 5 & 5 & 5 & 1 & 4 \end{pmatrix}^T$.

4. **Fie *G*=(*V*,*E*,*W*) graf ponderat, V={1,2,3,4,5}, E={(1,2), (1,4), (2,3), (2,4), (3,4)}, W((1,2))=5, W((1,4))=7, W((2,3))=4, W((2,4))=2, W((3,4))=1. Specificaţi matricea ponderilor.**

Raspuns: $W=\begin{pmatrix} \infty & 5 & \infty & 7 & \infty \\ 5 & \infty & 4 & 2 & \infty \\ \infty & 4 & \infty & 1 & \infty \\ 7 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$

**<u>Parcurgerea grafurilor in latime (Breadth First)</u>**

5. **Fie graful,**




**şi $v_0$=1. Realizați parcurgerea în lățime a grafului pornind din vârful 1 (tabelul distanțelor).**

Răspuns: Valorile calculate sunt,

| vîrf / d | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | 0 | 1 | 1 | 1 | ∞ | ∞ | 1 |
| 2 | 0 | 1 | 1 | 1 | 2 | 2 | 1 |
| | 0 | 1 | 1 | 1 | 2 | 2 | 1 |

Fie $G=(V,E)$ un graf, $|V|=n$. O alternativă de implementare a metodei BF este construită prin utilizarea următoarelor structuri de date,

- $A$ matricea de adiacenţă a grafului;
- o structură de tip coadă, $C$, în care sunt introduse vârfurile ce urmează a fi vizitate şi procesate (în sensul cercetării vecinilor lor);
- un vector $c$ cu $n$ componente, unde,

$$c_i = \begin{cases} 1, \text{ dacă } i \text{ a fost adăugat în coadă} \\ 0, \text{ altfel} \end{cases}$$

Componentele vectorului $c$ sunt iniţializate cu valoarea 0.

Parcurgerea BF poate fi descrisă astfel,

- coada $C$ este iniţializată cu vârful $v_0$;
- cât timp $C \neq \emptyset$, este extras şi vizitat un vârf $i$ din coadă, apoi sunt introduşi în coadă vecinii lui $i$ care nu au fost deja introduşi (acele vârfuri $k$ cu proprietatea că $c[k]=0$ şi $a[i][k]=1$). Vârfurile $i$ ce au fost introduse în coadă sunt marcate prin $c[i]=1$.

Aplicarea metodei de traversare BF determină următoarea evoluţie,

| t / c | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| t=1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| t=2 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| t=3 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

| t / C | | | | |
|---|---|---|---|---|
| t=1 | 1 | | | |
| t=2 | 2 | 3 | 4 | 7 |
| t=3 | 3 | 4 | 7 | 5 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| t=4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| | | | | |
|---|---|---|---|---|
| t=4 | 4 | 7 | 5 | 6 |
| t=5 | 7 | 5 | 6 | |
| t=6 | 5 | 6 | | |
| t=7 | 6 | | | |
| t=8 | | | | |

Observaţie Deoarece graful este conex, traversarea BF realizează vizitarea tuturor vârfurilor grafului. Aplicarea metodei BF unui graf neconex nu determină vizitarea tuturor vârfurilor. Cu alte cuvinte, metoda BF aplicată unui graf determină vizitarea tuturor vârfurilor care sunt conectate cu vârful iniţial selectat.

**Implementarea metodei de parcurgere BF**

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include<malloc.h>

//inserarea unui varf a grafului in coada
int insereaza_in_coada(int *coada, int n, int vf)
{
	//se insereaza pe ultima pozitie din coada un nou element
	coada[n] = vf;
	n++;
	return n;
}

//extragerea unui varf din coada
int extrage_din_coada(int *coada, int n, int *vf)
{
	//se scoate din coada un element - adica primul element(FIFO)
	int i;
	*vf = coada[0];
	for (i = 0; i < n - 1; i++)
		coada[i] = coada[i + 1];
	n--;
	return n;
}

//algoritm de parcurgere in latime
//v0 => varful initial
//a => matricea de adiacenta
//n => nr de varfuri din graf
void breadth_first(int v0, int **a, int n)
{
	int *coada, *m, i, nr_c, k;
	coada = (int*)malloc(n * sizeof(int));

	//alocare dinamica si initializarea elementelor din vectorul m cu 0
	//m-este vectorul care indica daca un varf a fost adaugat in coada sau nu.
	m = (int*)malloc(n * sizeof(int));
	for (i = 0; i < n; m[i++] = 0);

	//se stie ca primul element inserat in coada este varful primit ca input
	//nr_c - repr. cate elemente sunt in coada si pozitia unde poate fi inserat un varf in
coada(adica la final)
	nr_c = 0;
```

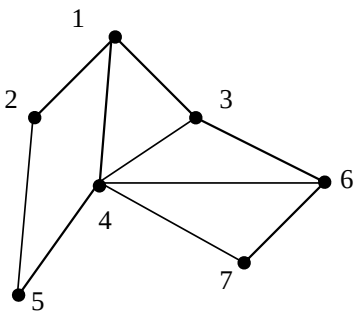
```
	nr_c = insereaza_in_coada(coada, nr_c, v0);
	m[v0] = 1;

	while (nr_c)//cat timp nr_c!=0 sau cat timp coada are elemente in ea
	{
		nr_c = extrage_din_coada(coada, nr_c, &i);
		printf("\n%i", i + 1);

		//cauta toti vecinii varfului care nu au fost adaugati deja in coada
		for (k = 0; k < n; k++)
			if ((a[i][k] == 1) && (m[k] == 0))
			{
				nr_c = insereaza_in_coada(coada, nr_c, k);
				m[k] = 1;
			}
	}

	//dezalocare memorie folosita
	free(coada);
	free(m);
}

void main()
{
	int n, v0, **a, m, i, j, vf1, vf2;
	printf("Numarul de varfuri:");
	scanf("%i", &n);
	//se va aloca dinamic memorie pentru matricea de adiacenta a
	a = (int **)malloc(n * sizeof(int*));
	for (i = 0; i < n; i++)
		a[i] = (int*)malloc(n * sizeof(int));

	//se initializeaza matricea cu 0....sau se putea folosea functia calloc in loc de malloc
si puteam sari de acest pas
	for (i = 0; i < n; i++)
		for (j = 0; j <= i; j++)
			a[j][i] = a[i][j] = 0;

	printf("\nNumarul de muchii:");
	scanf("%i", &m);
	for (i = 0; i < m; i++)
	{
		printf("Virf initial:"); scanf("%i", &vf1);
		printf("Virf final:"); scanf("%i", &vf2);
		a[vf1 - 1][vf2 - 1] = a[vf2 - 1][vf1 - 1] = 1;
	}

	printf("\nVirful initial pentru parcurgerea BF ");
	scanf("%i", &v0);

	printf("\nOrdinea de vizitare a virfurilor grafului este:");
	breadth_first(v0 - 1, a, n); //atentie aici....parcurgerea unui vector este de la
pozitia 0...==> v0 - 1

	//se va elibera zona de memorie alocata pentru matricea a
	for (i = 0; i < n; i++) free(a[i]);
	free(a);
}
```

```
Numarul de virfuri:7

Numarul de muchii:10
Virf initial:1
Virf final:2
Virf initial:1
Virf final:3
Virf initial:1
Virf final:4
Virf initial:1
Virf final:7
Virf initial:2
Virf final:5
Virf initial:3
Virf final:4
Virf initial:3
Virf final:6
Virf initial:4
Virf final:5
Virf initial:4
Virf final:7
Virf initial:6
Virf final:7

Virful initial pentru parcurgerea BF 1

Ordinea de vizitare a virfurilor grafului este:
1
2
3
4
7
5
6
Process returned 1 (0x1)   execution time : 110.516 s
Press any key to continue
```

Ordinea de vizitare a vârfurilor este 1,2,3,4,7,5,6 care poate fi văzută atât din exemplul numeric, cât și prin execuția programului pentru același graf și vârf de pornire.

## Metoda de parcurgere DF (Depth First)

6. Pentru graful,




şi $v_0$=1, prin aplicarea metodei descrisa in continuare , rezultă următoarea evoluţie.

O variantă de implementare a metodei DF rezultă prin gestionarea stivei *S* în modul următor. Iniţial vîrful $v_0$ este unicul component al lui *S*. La fiecare etapă se preia, fără ştergere, ca vîrf curent vârful stivei. Se introduce în stivă unul dintre vecinii vârfului curent încă nevizitat. Vizitarea unui vârf revine la introducerea lui în *S*. Dacă vârful curent nu are vecini încă nevizitaţi, atunci el este eliminat din stivă şi este efectuat un nou acces de preluare a noului vârf al stivei ca vârf curent. Calculul se încheie în momentul în care este efectuat un acces de preluare a vârfului stivei ca vârf curent şi se constată că *S* este vidă. Evident, nici în cazul acestei variante nu vor fi vizitate vârfurile care nu sunt conectate cu vârful iniţial ales.

| t \ c | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| t=1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| t=2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| t=3 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| t=4 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| t=5 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| t=6 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| t=7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=11 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=12 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=13 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| t=14 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| T \ S | | | | | | | |
|---|---|---|---|---|---|---|---|
| t=1 | 1 | | | | | | |
| t=2 | 2 | 1 | | | | | |
| t=3 | 5 | 2 | 1 | | | | |
| t=4 | 4 | 5 | 2 | 1 | | | |
| t=5 | 3 | 4 | 5 | 2 | 1 | | |
| t=6 | 6 | 3 | 4 | 5 | 2 | 1 | |
| t=7 | 7 | 6 | 3 | 4 | 5 | 2 | 1 |
| t=8 | 6 | 3 | 4 | 5 | 2 | 1 | |
| t=9 | 3 | 4 | 5 | 2 | 1 | | |
| t=10 | 4 | 5 | 2 | 1 | | | |
| t=11 | 5 | 2 | 1 | | | | |
| t=12 | 2 | 1 | | | | | |
| t=13 | 1 | | | | | | |
| t=14 | | | | | | | |

Ordinea în care sunt vizitate vârfurile corespunzător acestei variante este: 1,2,5,4,3,6,7

Implementarea metodei DF

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include<malloc.h>

//inserarea in stiva se bazeaza pe conceptul LIFO - ultimul intrat, primul iesit
//inserarea adauga la inceputul vectorului "stiva" un nou varf "vf"
int insereaza_in_stiva(int *stiva, int n, int vf)
{
	int i;
	for (i = n - 1; i >= 0; i--)
		stiva[i + 1] = stiva[i];
	stiva[0] = vf;
	n++;
	return n;
}
```

```
//stergerea din stiva presupune scoatea ultimului varf inserat in vectorul "stiva"(adica
primul element din vector)
int sterge_din_stiva(int *stiva, int n)
{
	int i;
	for (i = 0; i < n - 1; i++)
		stiva[i] = stiva[i + 1];
	n--;
	return n;
}

//returneaza ultimu element introdus in stiva(adica primul element din vectorul "stiva")
int citeste_din_stiva(int *stiva, int n)
{
	return stiva[0];
}

void depth_first(int v0, int **a, int n)
{
	int *stiva, *m, i, nr_c, gasit;
	// se aloca memorie pentru stiva – n elemente
	stiva = (int*)malloc(n * sizeof(int));

	//alocare dinamica si initializarea elementelor din vectorul m cu 0
	//m-este vectorul care indica daca un varf a fost adaugat in stiva sau nu.
	m = (int*)malloc(n * sizeof(int));
	for (int i = 0; i < n; m[i++] = 0);

	//primul element din stiva este varful v0
	nr_c = 0;
	nr_c = insereaza_in_stiva(stiva, nr_c, v0);
	m[v0] = 1;
	printf("\n%i", v0 + 1);

	while (nr_c)//cat timp nr_c!=0 sau cat timp stiva are elemente in ea
	{
		i = citeste_din_stiva(stiva, nr_c);
		gasit = 0;

		for (int k = 0; (k < n) && !gasit; k++)
		{
			if ((a[i][k] == 1) && (m[k] == 0))//insereaza primul vecin gasit al
varfului in stiva
			{
				nr_c = insereaza_in_stiva(stiva, nr_c, k);
				m[k] = 1;
				printf("\n%i", k + 1);  gasit = 1;//in acest moment se va iesi din
for
			}
		}

		if (!gasit)//daca gasit=0 dupa parcurgerea for-ului, adica daca varful curent nu
mai are vecini, atunci el este scos din stiva
			nr_c = sterge_din_stiva(stiva, nr_c);
	}

	//eliberare memorie alocata – pentru stiva si pentru m;
	free(stiva); free(m);
}

void main()
{
	int n, v0, **a, m, i, j, vf1, vf2;;
	printf("Numarul de virfuri:");
	scanf("%i", &n);
```

```
	//se va aloca dinamic memorie pentru matricea de adiacenta a
	a = (int **)malloc(n * sizeof(int*));
	for (i = 0; i < n; i++)
		a[i] = (int*)malloc(n * sizeof(int));

	//se initializeaza matricea cu 0....sau se putea folosea functia calloc in loc de malloc
si puteam sari de acest pas
	for (i = 0; i < n; i++)
		for (j = 0; j <= i; j++)
			a[j][i] = a[i][j] = 0;

	printf("\nNumarul de muchii:");
	scanf("%i", &m);
	for (i = 0; i < m; i++)
	{
		printf("Virf initial:"); scanf("%i", &vf1);
		printf("Virf final:"); scanf("%i", &vf2);
		a[vf1 - 1][vf2 - 1] = a[vf2 - 1][vf1 - 1] = 1;
	}

	printf("\nVirful initial pentru parcurgerea DF ");
	scanf("%i", &v0);

	printf("\nOrdinea de vizitare a virfurilor grafului este:");
	depth_first(v0 - 1, a, n);

	//se va elibera zona de memorie alocata pentru matricea a
	for (i = 0; i < n; i++) free(a[i]);
	free(a);

	getch();
}
```

```
Numarul de virfuri:7

Numarul de muchii:10
Virf initial:1
Virf final:2
Virf initial:1
Virf final:3
Virf initial:1
Virf final:4
Virf initial:2
Virf final:5
Virf initial:3
Virf final:4
Virf initial:3
Virf final:6
Virf initial:4
Virf final:5
Virf initial:4
Virf final:6
Virf initial:4
Virf final:7
Virf initial:6
Virf final:7

Virful initial pentru parcurgerea DF 1

Ordinea de vizitare a virfurilor grafului este:
1
2
5
4
3
6
7
Process returned 13 (0xD)   execution time : 99.359 s
Press any key to continue.
```