

## ЛАБОРАТОРНО УПРАЖНЕНИЕ № 10

### ТЕМА: Граф. Представяне на графи. Обхождане на графи.

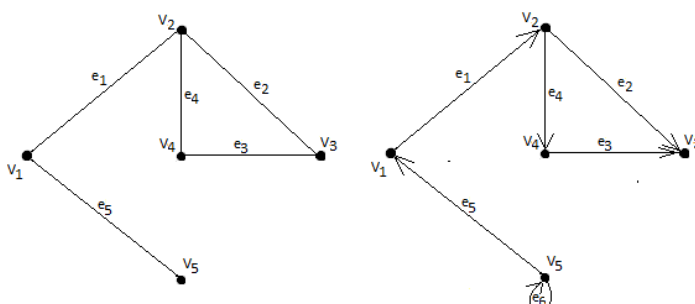
#### ЦЕЛ:

Целта на упражнението е студентите да се запознаят със основните елементи на езика РНР. След упражнението студентите би следвало да могат да боравят с променливи и константи, да пишат кратки изрази чрез РНР скриптове.

#### I. ТЕОРЕТИЧНА ЧАСТ

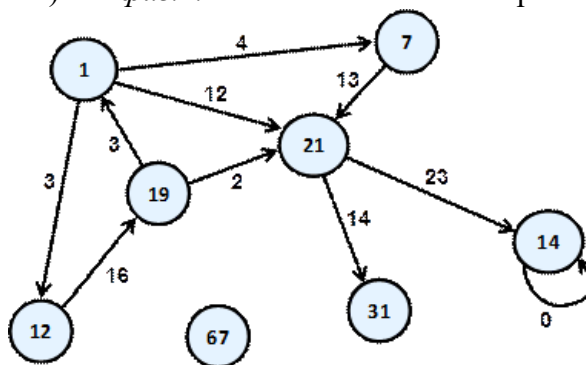
##### 1. Основни понятия за граф.

Графът е съвкупност от две групи елементи – точки (върхове) и линии (ребра), съединяващи някои двойки точки. Най – често точките се изобразяват в равнина, а линиите са прави. Множеството на върховете се бележи с  $V$ , а множеството на ребрата с  $E$ . Съвкупността  $G(V,E)$  ще наричаме **граф**. При това ако за всяко ребро има дефинирана посока (логическо „начало“ и „край“) се нарича **ориентиран граф**, в противен случай **неориентиран**. При ориентираните графи ребрата се наричат **дъги**. (фиг.1.)



Фиг.1. Илюстрация на неориентиран и ориентиран граф

За ребрата може да се зададе функция, която на всяко едно ребро съпоставя реално число. Тези така получени реални числа ще наричаме тегла. Като примери за тегла можем да дадем дължината на директните връзки между два съседни града, пропускателната способност на една тръба и др. Граф, който има тегла по ребрата, се нарича претеглен (weighted). На фиг.2. е показано как се изобразява такъв граф.



Фиг.2. Ориентиран граф с тегла на дъгите.

Някои основни понятия и определения при графите са:

- Съседни върхове – върхове свързани с общо ребро или дъга;
- Входящи и изходящи дъги – влизащи и излизащи дъги за връх  $v_i$ ;
- Инцидентни връх и дъга(ребро) – когато върха се явява начален или краен за дъгата (реброто).
- Съседни дъги (ребра) – две дъги (ребра) са съседни, ако са инцидентни с един и същи връх на графа
- Паралелни дъги (ребра) – когато два върха  $v_i, v_j$  са съединени с повече от една дъга (ребро)
- Степен на връх  $v_i$  – броят на инцидентните с върха  $v_i$  ребра (дъги) и се бележи с  $d(v_i)$ ;
- Висящи върхове – върхове с  $d(v_i)=1$ ;
- Изолирани върхове – върхове с  $d(v_i)=0$  (връх 67 от фиг.2.);
- Примка – когато началният и крайният връх на една дъга съвпадат (връх 14 от фиг.2.);
- Верига – Нека  $v_1, v_2, \dots, v_n, v_{n+1}$  е произволна последователност от върхове. Верига се нарича последователност от дъги  $e_1, e_2, \dots, e_n$ , за които  $e_i=(v_i, v_{i+1})$  или  $e_i=(v_{i+1}, v_i)$ ,  $i=1, 2, \dots, n$ . Върхът  $v_1$  се нарича начален връх за веригата, а върхът  $v_{n+1}$  – краен връх на веригата. Дължина на веригата се разбира броя участващи в нея дъги.
- Път – Верига за която  $e_i=(v_i, v_{i+1})$  за всяко  $i=1, 2, \dots, n$ , се нарича път (т.е. последователност от ребра спазваща посоката на ориентирания граф, позволяващи свързване на несъседни върхове). Понятието дължина на път, начален и краен връх на пътя се определят както при веригата;
- Цена на път – ще наричаме сумата на теглата на дъгите участващи в пътя.
- Контур – верига при която началния и крайният връх съвпадат.
- Цикъл – път при която началния и крайния връх съвпадат;
- Прост граф – това е граф без паралелни ребра(дъги) и примки
- Пълен граф – при който за всяка двойка върхове съществува ребро инцидентно с тях.

## 2. Представяне на граф

Съществуват много различни начини за представяне на граф в програмирането. Различните представяния имат различни свойства и кое точно трябва да бъде избрано, зависи от конкретния алгоритъм, който искаме да приложим. С други думи казано – представяме графа си така, че операциите, които алгоритъмът ни най-често извършва върху него, да бъдат максимално бързи. Без да изпадаме в големи детайли ще изложим някои от най-често срещаните представяния на графи.

- **Списък на ребрата** – представя се, чрез списък от наредени двойки  $(v_i, v_j)$ , където съществува ребро от  $v_i$  до  $v_j$ . Ако графът е претеглен, то вместо наредена двойка имаме наредена тройка, като третият ѝ елемент показва какво е теглото на даденото ребро.
- **Списък на наследниците** – в това представяне за всеки връх  $v$  се пази списък с върховете, към които сочат ребрата започващи от  $v$ . Тук отново, ако графът е претеглен, към всеки елемент от списъка с наследниците се добавя допълнително поле, показващо цената на реброто до него.
- **Матрица на съседство** – графът се представя като квадратна матрица  $g[N][N]$ , в която, ако съществува ребро от  $v_i$  до  $v_j$ , то на позиция  $g[i][j]$  в матрицата е записано 1. Ако такова ребро не съществува, то в полето  $g[i][j]$  е записано 0. Ако графът е

претеглен, в позиция  $g[i][j]$  се записва теглото на даденото ребро, а матрицата се нарича матрица на теглата. Ако между два върха в такава матрица не съществува път, то тогава се записва специална стойност, означаваща безкрайност.

- **Матрица на инцидентност между върхове и ребра** – в този случай отново се използва матрица, само че с размери  $g[M][N]$ , където  $M$  е броят на върховете, а  $N$  е броят на ребрата. Всеки стълб представя едно ребро, а всеки ред един връх. Тогава в стълба съответстващ на реброто  $(v_i, v_j)$  само и единствено на позиция  $i$  и на позиция  $j$  ще бъдат записани 1, а на останалите позиции в този стълб ще е записана 0. Ако реброто е примка, то на позиция  $i$  записваме 2. Ако графът който искаме да представим е ориентиран и искаме да представим реброто от  $v_i$  до  $v_j$ , то на позиция  $i$  пишем 1, а на позиция  $j$  пишем -1.

### 3. Обхождане на граф

- **Обхождане в дълбочина**

Обхождане в дълбочина (*Depth First Search - DFS*) е метод за систематично обхождане на всички върхове на даден граф, достижими от даден начален връх. Използвайки този метод върху дърво то той отговаря на префиксно обхождане. Метода се използва най често за намиране на покриващо дърво  $T$  за графа  $G$ .

Нека  $G(V, E)$  е свързан граф и  $r \in V$  (принадлежи).

Построяваме възможно най дълъг път в  $G$  с начало – върха  $r$ . Ако този път съдържа всички върхове от  $V$ , то тогава той е покриващо дърво  $T$  на  $G$  и с това алгоритъма приключва. В случай че построения път не съдържа всички върхове от графа, то движението се извършва в две посоки: напред – към някой от още не посетените върхове на графа и назад – връщане от текущия връх към неговия пряк родител и избор на някой от наследниците му, който още не е бил посетен. Ако няма такъв друг наследник, се връщаме назад по посетените предшественици, докато намерим такъв, който още има непосетени наследници и от него продължаваме да строим най – дълъг път, който съдържа тези непосетени върхове.

Тъй като графа  $G$  е краен и свързан то тази процедура ще приключи след краен брой стъпки и ще върне като резултат покриващо дърво  $T$  на  $G$  с корен  $r$ .

Ако графът е ориентиран, движението напред по дъгите се извършва в съответствие с тяхната ориентация.

Един от начините за описание на алгоритъма за обхождане на граф в дълбочина е чрез наредба на върховете му, за да се проследи по лесно кои от тях са посетени. За описанието на алгоритъма в процедурни стъпки са използвани два маркера – *посетен* и *непосетен*.

Алгоритъмът може да се опише чрез следните процедурни стъпки:

0. Нека  $G(V, E)$  е свързан граф и  $|V|=n$ ,  $V = \{v_1, v_2, \dots, v_n\}$ . Маркират се всички върхове като непосетени.
1. Полагаме текущ връх  $v=v_1$ . Инициализираме  $T=v$  – дърво, което има само един връх – неговия корен. Маркираме върхът  $v_1$  като посетен;
2. Избира се най – малкият индекс  $i$ ,  $2 \leq i \leq n$ , за който върхът  $v_i$  е с маркер непосетен и дъгата  $\{v, v_i\}$  принадлежи на  $E$ . Ако такъв индекс не съществува, то продължаваме към стъпка 3. В противен случай:
  - 1) Добавяме дъгата  $\{v, v_i\}$  към дървото  $T$ ;
  - 2) Маркираме върхът  $v_i$  с маркер посетен;
  - 3) Полагаме текущ връх  $v=v_i$ ;

- 4) Връщане в началото на **стъпка 2**;
3. Ако текущият връх  $v=v_1$ , то  $T$  е покриващо дърво на  $G$ . Преминава се към КРАЙ на алгоритъма;
4. Ако текущият връх  $v$  е различен от  $v_1$ , то се извършва връщане назад към родителя  $p(v)$  на текущия връх  $v$  в дървото  $T$  и полагаме  $v=p(v)$ . Извършва се връщане към **стъпка 2**.

- **Обхождане в ширина**

Обхождане в ширина (Breadth (Width) First Search - BFS) е метод за систематично обхождане на всички върхове на даден граф, при който върховете се обхождат по реда на нарастване на разстоянието ( в брой върхове) от началния връх. Този метод се нарича още метод на ветрилото или обхождане по нива, тъй като при обхождането се избира начален връх ( корен на покриващо дърво) и първо се посещават върховете, които са свързани чрез дъги с него. След това се посещават върховете които пряко са свързани с наследниците на корена, т.е. са свързани с пътища с дължина 2 от началния връх, и не са били посетени преди това. На следващото ниво се посещават всички върхове от непосетените до момента, които са свързани чрез пътища с дължина 3 от началния връх и т.н.

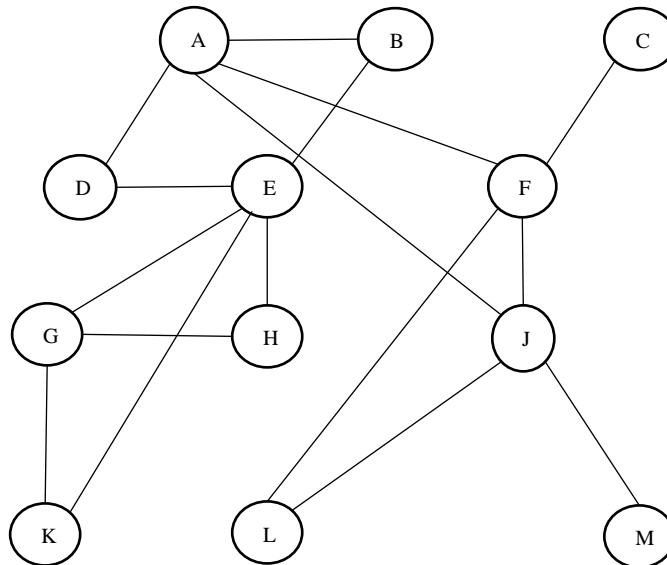
При обхождане на графа в ширина е удобно, както при предния алгоритъм, да се направи наредба на върховете в  $G$  и да се използват маркери. За този алгоритъм е подходящо да се използва и структурата от данни, опашка (Queue).

Алгоритъмът може да се опише чрез следните процедурни стъпки:

0. Нека  $G(V,E)$  е свързан граф и  $|V|=n$ ,  $V = \{v_1, v_2, \dots, v_n\}$ . Маркират се всички върхове като непосетени.
1. Полагаме текущ връх  $v=v_1$ . Инициализираме  $T=v$  – дърво, което има само един връх – неговия корен. Маркираме върхът  $v_1$  като посетен. Поставяме върхът най отзад на опашка  $Q$ , която първоначално е празна;
2. Ако опашката  $Q$  не е празна, прилагаме следното: Премахваме върхът, който стои най – отпред на опашката и асоциираме текущият връх  $v$  с този връх. Изследваме множеството  $C$  (съдържащо съседите на текущия връх) от всички върхове  $v_i$ ,  $2 \leq i \leq n$ , който са свързани с дъга с текущия връх  $v$ .  
За всеки връх  $v_i$  принадлежат на  $C$ , и ако  $v_i$  е с маркер непосетен:
  - 1) Добавяме дъгата  $\{v, v_i\}$  към дървото  $T$ ;
  - 2) Маркираме върхът  $v_i$  с маркер посетен;
  - 3) Поставяме върхът  $v_i$  най – отзад на опашката  $Q$ ;
3. Ако множеството от върхове на графа  $G$  съдържа само посетени върхове, то  $T$  е покриващо дърво на  $G$ . В този случай следва КРАЙ на алгоритъма в противен случай се извършва връщане към **стъпка 2**.

## II. ПРАКТИЧЕСКА ЧАСТ

Задачите в тази част са приложени върху следния граф на фиг.3.



Фиг.3. Граф за обхождане

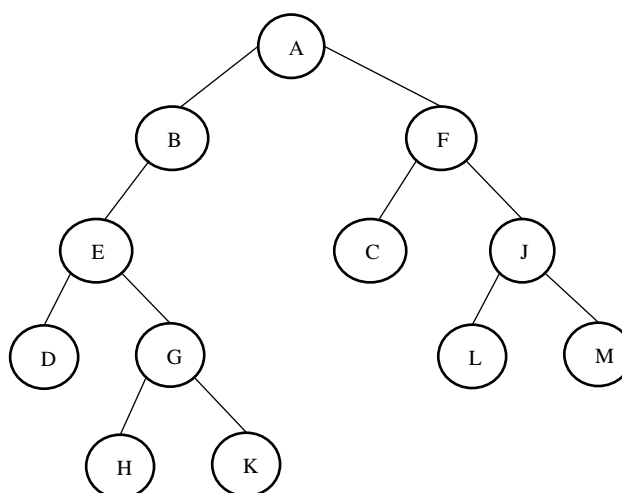
**ЗАДАЧА1:** За граф на фиг.3. да се намери покриващо дърво като се използва алгоритъма с обхождане в дълбочина (DFS).

### РЕШЕНИЕ:

Последователността за обхождане на върховете ще е следната:

{A , B, E, D, G, H, K, F, C, J, L, M}

Полученото покриващо дърво е показано на фиг.4.



Фиг.4. Покриващо дърво на графа G чрез алгоритъма DFS.

### ПОЯСНЕНИЕ

В табл.1. е представено подробно описание на построяването на покриващото дърво при обхождане в дълбочина на графа от фиг.3. За върховете които не е възможно

продължение на текущия път, е отбелязано връщане назад и търсене на ново продължение (нов клон на дървото) от непосетени върхове.

Табл.1. Описание на алгоритъма DFS

Покриващо дърво $T(V,E)$	Текущ връх	Непосетени върхове
$V = \{ \}$ $E = \{ \}$		{ A, B, C, D, E, F, G, H, J, K, L, M }
$V = \{ A \}$ $E = \{ \}$	$V = A$	{ B, C, D, E, F, G, H, J, K, L, M }
$V = \{ A, B \}$ $E = \{ [A,B] \}$	$V = B$	{ C, D, E, F, G, H, J, K, L, M }
$V = \{ A, B, E \}$ $E = \{ [A,B], [B,E] \}$	$V = E$	{ C, D, F, G, H, J, K, L, M }
$V = \{ A, B, E, D \}$ $E = \{ [A,B], [B,E], [E,D] \}$	$V = D$	{ C, F, G, H, J, K, L, M }
$V = \{ A, B, E, D \}$ $E = \{ [A,B], [B,E], [E,D] \}$	$V = E$ <i>Връщане назад</i>	{ C, F, G, H, J, K, L, M }
$V = \{ A, B, E, D, G \}$ $E = \{ [A,B], [B,E], [E,D], [E,G] \}$	$V = G$	{ C, F, H, J, K, L, M }
$V = \{ A, B, E, D, G, H \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H] \}$	$V = H$	{ C, F, J, K, L, M }
$V = \{ A, B, E, D, G, H, \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H] \}$	$V = G$ <i>Връщане назад</i>	{ C, F, J, K, L, M }
$V = \{ A, B, E, D, G, H, K \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K] \}$	$V = K$	{ C, F, J, L, M }
$V = \{ A, B, E, D, G, H, K \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K] \}$	$V = G$ <i>Връщане назад</i>	{ C, F, J, L, M }
$V = \{ A, B, E, D, G, H, K \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K] \}$	$V = E$ <i>Връщане назад</i>	{ C, F, J, L, M }
$V = \{ A, B, E, D, G, H, K \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K] \}$	$V = B$ <i>Връщане назад</i>	{ C, F, J, L, M }
$V = \{ A, B, E, D, G, H, K \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K] \}$	$V = A$ <i>Връщане назад</i>	{ C, F, J, L, M }
$V = \{ A, B, E, D, G, H, K, F \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K], [A,F] \}$	$V = F$	{ C, J, L, M }
$V = \{ A, B, E, D, G, H, K, F, C \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K], [A,F], [F,C] \}$	$V = C$	{ J, L, M }
$V = \{ A, B, E, D, G, H, K, F, C \}$ $E = \{ [A,B], [B,E], [E,D], [E,G], [G,H], [G,K], [A,F], [F,C] \}$	$V = F$ <i>Връщане назад</i>	{ J, L, M }
$V = \{ A, B, E, D, G, H, K, F, C, J \}$	$V = J$	{ L, M }

$E=\{[A,B],[B,E],[E,D],[E,G],[G,H],[G,K],[A,F],[F,C],[F,J]\}$		
$V=\{A, B, E, D, G, H, K, F, C, J, L\}$ $E=\{[A,B],[B,E],[E,D],[E,G],[G,H],[G,K],[A,F],[F,C],[F,J],[J,L]\}$	$V=L$	$\{M\}$
$V=\{A, B, E, D, G, H, K, F, C, J, L\}$ $E=\{[A,B],[B,E],[E,D],[E,G],[G,H],[G,K],[A,F],[F,C],[F,J],[J,L]\}$	$V=J$ <i>Връщане назад</i>	$\{M\}$
$V=\{A, B, E, D, G, H, K, F, C, J, L, M\}$ $E=\{[A,B],[B,E],[E,D],[E,G],[G,H],[G,K],[A,F],[F,C],[F,J],[J,L], [J,M]\}$	$V=M$ <b>KRAJ</b>	$\{\}$

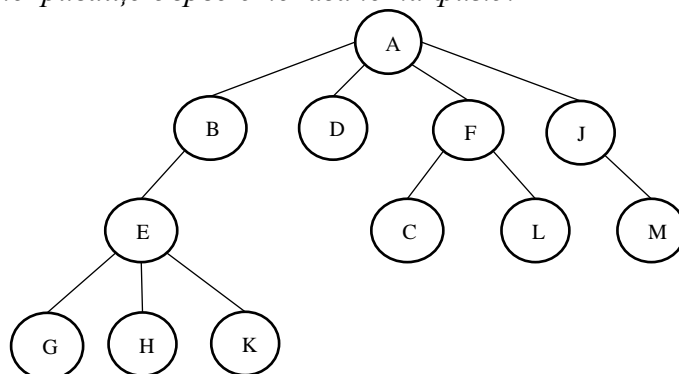
**ЗАДАЧА2:** За граф на фиг.3. да се намери покриващо дърво като се използва алгоритъма с обхождане в ширина (BFS).

**РЕШЕНИЕ:**

Последователността за обхождане на върховете ще е следната:

$\{A, B, D, F, J, E, C, L, M, G, H, K\}$

Полученото покриващо дърво е показано на фиг.5.



Фиг.5. Покриващо дърво на графа G чрез алгоритъма BFS.

**ПОЯСНЕНИЕ:**

В табл.2. е представено подробно описание (стъпка по стъпка) на построяването на покриващото дърво при обхождане в ширина на графа от фиг.3.

Табл.1. Описание на алгоритъма DFS

Покриващо дърво $T(V,E)$	Опашка Q	Текущ връх и списък с неговите наследници	Спусък с непосетени върхове
$V=\{\}$ $E=\{\}$	$\{\}$		$\{A, B, C, D, E, F, G, H, J, K, L, M\}$
$V=\{A\}$ $E=\{\}$	$\{A\}$	$V=A$	$\{B, C, D, E, F, G, H, J, K, L, M\}$
$V=\{A, B, D, F, J\}$ $E=\{[A,B],[A,D], [A,F],[A,J]\}$	$\{B, D, F, J\}$	$V=A$ $C=\{B, D, F, J\}$	$\{C, E, G, H, K, L, M\}$

V={A, B, D, F, J, E} E={ [A,B],[A,D], [A,F],[A,J],[B,E] }	{D, F, J, E}	V=B C={A, E}	{C, G, H, K, L, M}
V={A, B, D, F, J, E} E={ [A,B],[A,D], [A,F],[A,J],[B,E] }	{F, J, E}	V=D C={A, E}	{C, G, H, K, L, M}
V={A, B, D, F, J, E, C, L} E={ [A,B],[A,D], [A,F],[A,J],[B,E], [F,C],[F,L] }	{J, E, C, L}	V=F C={A, C, J, L}	{G, H, K, M}
V={A, B, D, F, J, E, C, L, M} E={ [A,B],[A,D], [A,F],[A,J],[B,E], [F,C],[F,L], [J,M] }	{E, C, L, M}	V=J C={A, F, L, M}	{G, H, K}
V={A, B, D, F, J, E, C, L, M, G, H, K} E={ [A,B],[A,D], [A,F],[A,J],[B,E], [F,C],[F,L], [J,M],[E,G],[E,H],[E,K] }	{C, L, M, G, H, K}	V=E C={B, D, G, H, K}	{ } КРАЙ

**ЗАДАЧА3:** Да се направи програмна реализация за представяне на граф чрез матрица на съседство.

### РЕШЕНИЕ:

```
using System;
using System.Collections.Generic;

public class Graph
{
    // Съдържа върховете на графа
    private int[,] vertices;

    public Graph(int[,] vertices)
    {
        this.vertices = vertices;
    }

    // Добавяне на ново ребро от i до j
    public void AddEdge(int i, int j)
    {
        vertices[i,j] = 1;
    }

    public void RemoveEdge(int i, int j)
    {
        vertices[i,j] = 0;
    }

    //Проверява дали има ребро между върховете.
    public bool HasEdge(int i, int j)
    {
        return vertices[i,j] == 1;
    }

    // Връща съседите на даден връх.
```



```

public IList<int> GetSuccessors(int i)
{
    IList<int> successors = new List<int>();

    for (int j = 0; j < vertices.GetLength(1); j++)
    {
        if (vertices[i,j] == 1)
        {
            successors.Add(j);
        }
    }

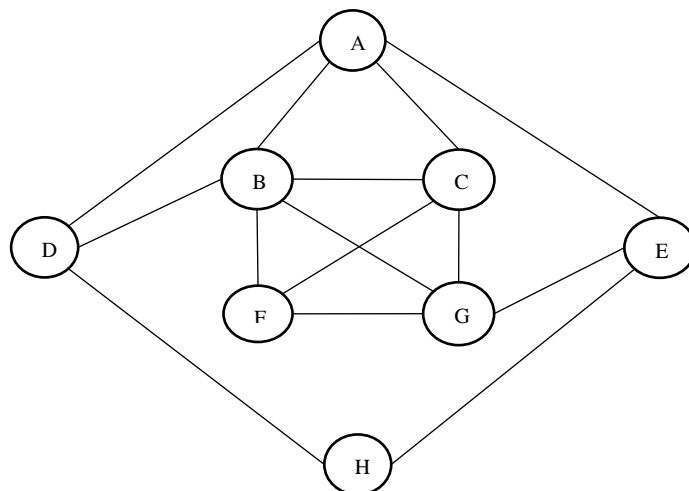
    return successors;
}
}

```

### ПОЯСНЕНИЕ:

Чрез горния фрагмент код е направена примерна програмна реализация за представяне на граф чрез матрица на съседство. Чрез конструктура Graph се създават върховете на графа чрез матрицата на съседство. Методите „AddEdge()“ и „RemoveEdge()“ служат за добавяне и премахване на ребро на графа. Метода „HasEdge()“ проверява дали съществува ребро между посочените върхове и метода „GetSuccessors()“ връща съседите на даден връх.

### III. Задачи за семинарни упражнения



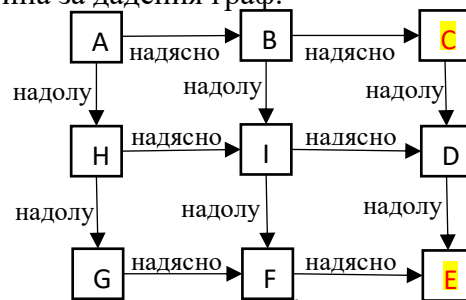
Фиг.6. Граф за обхождане

1. Да се намери покриващо дърво на свързания граф от фиг.6. като се използва алгоритъма DFS.
2. Да се намери покриващо дърво на свързания граф от фиг.6. като се използва алгоритъма BFS.
3. Да се създаде матрица на съседство за графа на фиг.6.

### IV. Задача за лабораторни упражнения.

1. Използвайки задача 3 от практическата част и получената матрица от задача 3 в раздел III, представете графа на фиг.6 програмно. Направете така че потребителя да може да проверява между кои върхове съществува ребро и между кои не.
2. Към предходната задача добавете метод за премахване на връх.

3. Да се създаде програмна реализация на графа от фиг.6, като се използват списъци на ребрата.
4. Създайте програма която ще изведе реда на обхождане на елементите от алгоритъма DFS.
5. Създайте програма която ще изведе реда на обхождане на елементите от алгоритъма BFS.
6. Даден е граф на фиг.7. с начален връх А и крайни цели – върховете С и Е. Ако при еквивалентни алтернативи се избира първо движение надолу, а после надясно, да се създаде програма която ще определи до коя крайна цел ще се достигне първо при:
  - А) обхождане в дълбочина за дадения граф;
  - Б) обхождане в ширина за дадения граф.



Фиг.7. Граф за зад. 5