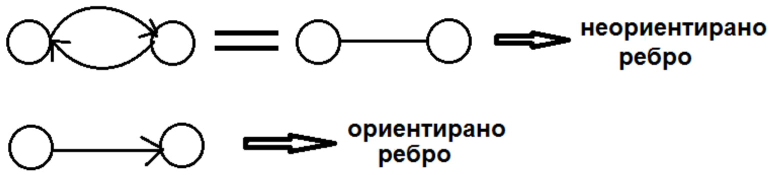
Графи

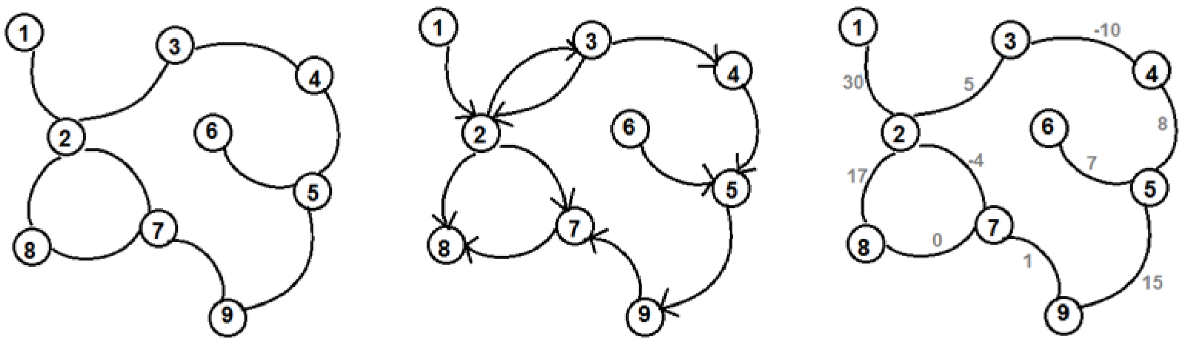
Граф е нелинейна структура данни, която се дефинира като множество върхове (V) и множество ребра (E) – наредени двойки върхове, между които има връзка. Бележим графа като G(V, E).

Спрямо вида на ребрата графите се разделят на няколко вида:

- неориентиран – ако от връх X до връх Y съществува ребро, то тогава и само тогава съществува и ребро от връх Y до връх X. Наричаме този тип ребра *неориентирани ребра* и всяко от тях може да се представи чрез две срещуположни ориентирани ребра.
- ориентиран – от връх X до връх Y съществува ребро, но не е задължително от връх Y до връх X също да съществува ребро. Наричаме този тип ребра *ориентирани ребра.*



- претеглен (тегловен) – съществува тегловна функция f: (x, y) → w, задаваща тегло на всяко ребро.



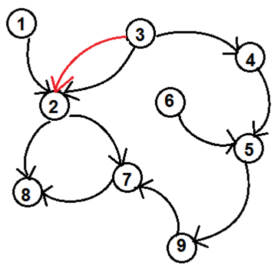
Неориентиран граф

Ориентиран граф

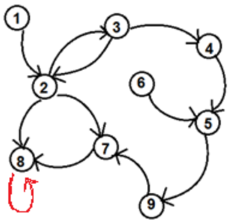
Претеглен граф

Основни понятия:

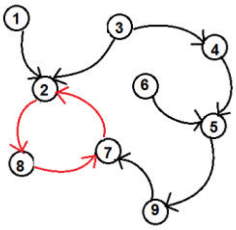
- мултиграф - ако E е мултимножество, т.е. се допуска повторение на двойките (X, Y), то G(V, E) наричаме мултиграф.



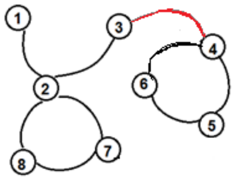
- <u>инцидентни ребра/върхове</u> – ако съществува ребро *e* между върховете X и Y, то X и Y се наричат инцидентни с *e* (краища на *e*).
- <u>съседни върхове</u> – ако върховете X и Y са инцидентни на едно и също ребро, то те са съседни.
- <u>примка</u> – ребро от вида *e*=(X, X).



- <u>път</u> – поредица от свързани върхове.
- <u>цикъл</u> – такъв път, при който първият и последният врръх съвпадат.



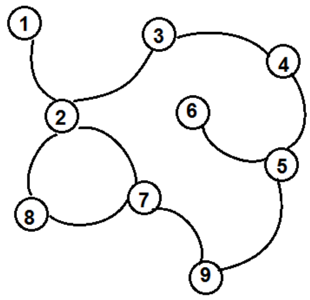
- <u>цикличен граф</u> – граф, съдържащ поне един цикъл.
- <u>ацикличен граф</u> – граф, който не съдържа цикли.
- <u>DAG (Directed Acyclic Graph)</u> – ориентиран нецикличен граф.
- <u>компонента</u> – един или повече върха, свързани с път помежду си.
- <u>мост</u> – ребро, при премахването на което една компонента се разделя на две нови.



Основни представяния на графи

- Матрица на съседство

Представяне на релацията на съседство в таблична форма, бележейки с 0 отсъствието на ребро, а с 1 (или съответното тегло при претеглен граф) – наличието му.

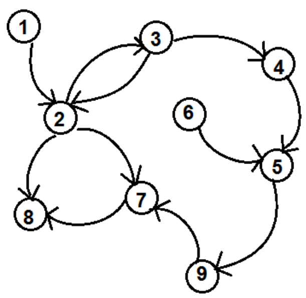


	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0
5	0	0	0	1	0	1	0	0	1
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	1	1
8	0	1	0	0	0	0	1	0	0
9	0	0	0	0	1	0	1	0	0

Пример:

```
bool graph[128][128];
int n, m;                          //n - брой върхове, m – брой ребра
cin>>n>>m;
for(int i=0; i<m; i++){  //въвеждаме всяко ребро
    int x, y;
    cin>>x>>y;
```

Ако графът е ориентиран, можем също така да построим т.н. матрица на инцидентност – отбелязваме ребро с 1 (или съответното тегло w при претеглен граф), а обратното му – с -1 (или –w). Отсъствието на ребро бележим с 0.



	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	-1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	0	0	0
4	0	0	-1	0	1	0	0	0	0
5	0	0	0	-1	0	-1	0	0	1
6	0	0	0	0	1	0	0	0	0
7	0	-1	0	0	0	0	0	1	-1
8	0	-1	0	0	0	0	-1	0	0
9	0	0	0	0	-1	0	1	0	0

<u>Предимства:</u>
+ лесна проверка за наличие на ребро между два върха

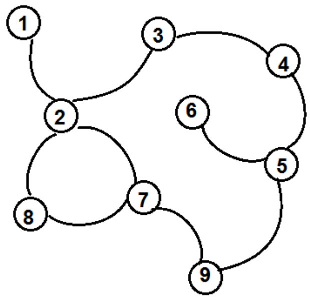
<u>Недостатъци:</u>
- може да бъде много разредена (да има повече върхове, отколкото ребра)

- бавно откриване на съседите на даден връх

- Списък на съседите

Представяне на релацията на съседство като списъци на съседни върхове за всеки връх.

1	2			
2	1	3	7	8
3	2	4		
4	3	5		
5	5	6	9	
6	5			
7	2	8	9	
8	2	7		
9	5	7		



Пример:

```
vector<vector<int> > graph;
int n, m;                          //n - брой върхове, m – брой ребра
cin>>n>>m;
graph.resize(n+1);                 //заделяме достатъчно място за всички списъци
for(int i=0; i<m; i++){            //въвеждаме всяко ребро
   int x, y;
   cin>>x>>y;
   graph[x].push_back(y);  //добавяме Y като съсед на X
```

Аналогично се прилага за ориентиран граф. Ако графът е претеглен, то се пази информация и за теглото на реброто.

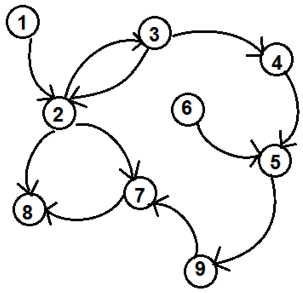
Пример:

```
vector<vector<pair<int, int> > > graph;
int n, m;                          //n - брой върхове, m – брой ребра
cin>>n>>m;
graph.resize(n+1);                 //заделяме достатъчно място за всички списъци
for(int i=0; i<m; i++){            //въвеждаме всяко ребро
   int x, y, w;
   cin>>x>>y>>w;
```

<u>Предимства:</u>	<u>Недостатъци:</u>
+ пести памет	- бавна проверка за наличието на ребро между два върха
+ лесно обхождане на съседите на даден връх	

- Списък на ребрата

Представяне на множеството E като списък от наредени (ненаредени за неориентиран граф) двойки върхове.



1	2
2	3
3	2
3	4
4	5
6	5
5	9
9	7
2	7
2	8
7	8

Пример:

```
vector<pair<int, int> > graph;
int n, m;                       //n - брой върхове, m – брой ребра
cin>>n>>m;
for(int i=0; i<m; i++){         //въвеждаме всяко ребро
    int x, y;
    cin>>x>>y;
```

Ако графът е претеглен, то се пази информация и за теглото на реброто.

Пример:

```
vector<pair<pair<int, int>, int > > graph;
int n, m;                       //n - брой върхове, m – брой ребра
cin>>n>>m;
for(int i=0; i<m; i++){         //въвеждаме всяко ребро
    int x, y, w;
    cin>>x>>y>>w;
```

<u>Предимства:</u>
+ пести памет

<u>Недостатъци:</u>
- бавна проверка за наличието на ребро между два върха
- бавно обхождане на съседите на даден връх