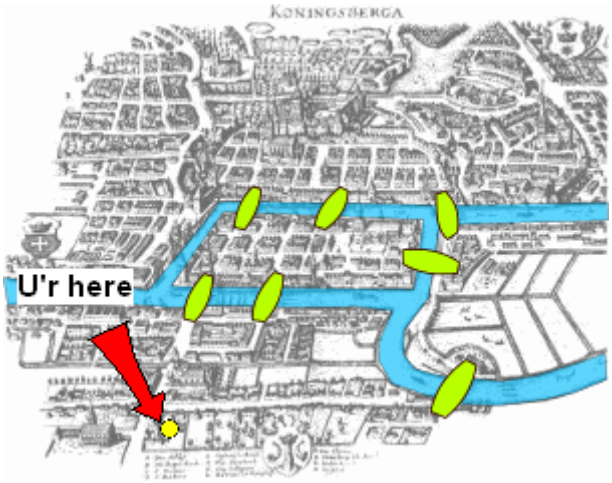
Теория на графите

Графите се една изключително полезна и доста разпространена структура от данни. Използват се за описването на най-разнообразни взаимовръзки между обекти от практиката, свързани с почти всичко. Както ще видим по-късно, дървета са подмножество на графите, т.е. графите представляват една обобщена структура, позволяваща моделирането на доста голяма съвкупност от реални ситуации.

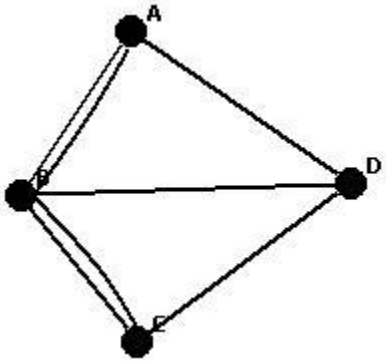
Теорията на графите е клон от съвременната математика. Графите са едни от най-полезните структури от данни в информатиката. Много задачи от различни области на науката и практиката могат да бъдат моделирани с граф и решени с помощта на съответен алгоритъм върху него.

Исторически първият проблем, който може да бъде отнесен към теория на графите, е задачата на Кьонигсбергските мостове на Ойлер (1736). През този пруски град минава реката Прегел, на която са построени седем моста. Пет от тях свързвали намиращия се в града остров с двата бряга. Хората си задавали въпроса дали е възможно да бъде извършена такава разходка през града, че всеки мост да бъде преминат точно веднъж. Отговаряйки на този въпрос, Ойлер поставя основите на теория на графите.

Ето как изглеждала обстановката по реката, съответно примерния граф с който Ойлер си е блъскал главата



С ребра и върхове, това изглежда така:



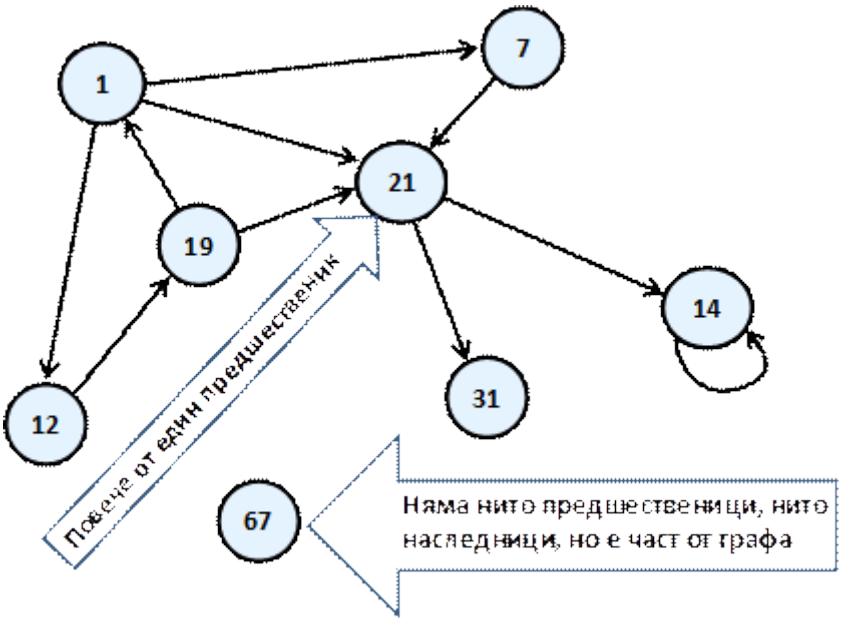
Наглед простата задача - "Може ли да обходя всички мостове и пак да съм на брега, от който съм тръгнал?"

Честата употреба на графите в практиката е довела до задълбочени изследвания в "теория на графите", в която са известни огромен брой задачи за графи и за повечето от тях има и добре известно решение.

Основни понятия

Графът се дефинира чрез съвкупност от върхове (наричани също и възли) и съвкупност от връзки между отделни двойки върхове. Тези връзки в някои задачи се разглеждат като притежаващи посока, а в други задачи – като ненасочени. Наричат се съответно дъги или ребра. Връзките обикновено се асоциират със свързващи пътища (далекопроводи и пр.) между върховете, а чрез самите върхове в приложенията се моделират градове, гари, водохранилища, електрически подстанции и т.н. В приложенията към всяка дъга или ребро се приписват една или повече числови стойности (наричани още и тегла), които имат смисъл да зададат дължината на разстоянието между двата свързани върха, цената за преминаване, максимален пропусквателен капацитет и пр.

Да разгледаме следният примерен граф, чийто тип по-късно ще наречем краен ориентиран. В него отново имаме номерация на върховете, която е абсолютно произволна и е добавена, за да може по-лесно да говорим за някой от тях конкретно.

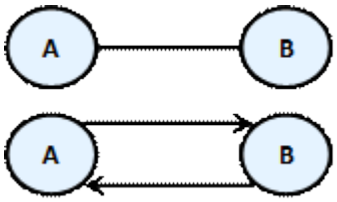


Кръгчетата на схемата, ще наричаме **върхове**, а стрелките, които ги свързват, ще наричаме **ориентирани ребра (дъги)**. Върхът, от който излиза стрелката ще наричаме **предшественик** на този, който стрелката сочи. Например "19" е предшественик на "1". "1" от своя страна се явява **наследник** на "19". За разлика от структурата дърво, сега всеки един връх може да има повече от един предшественик. Например "21" има три - "19", "1" и "7". Ако два върха са свързани с ребро, то казваме, че тези два върха са **инцидентни** с това ребро.

Краен ориентиран граф (finite directed graph) се нарича наредената двойката двойка **(V, E)**, където **V** е крайно множество от върхове, а **E** е крайно множество от ориентирани ребра. Всяко ребро ***e*** принадлежащо на **E** представлява наредена двойка от върхове ***u*** и ***v*** т.е. ***e*=(*u*, *v*)**, които еднозначно го определят.

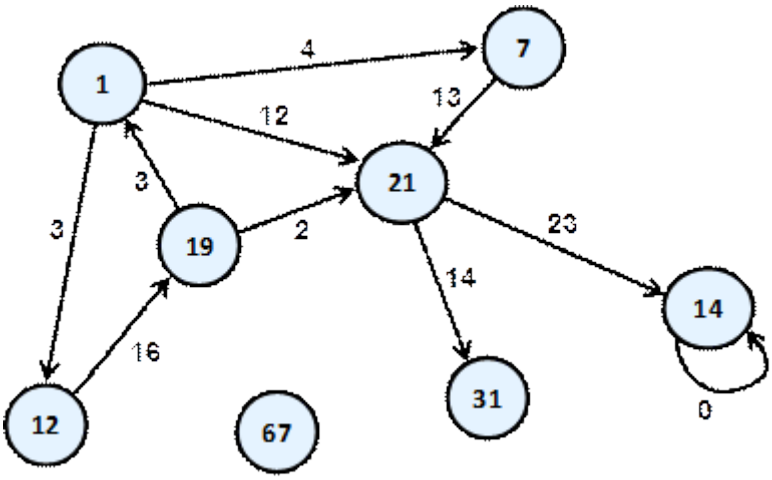
За по-доброто разбиране си представете върховете като градове, а ориентираните ребра като еднопосочни пътища. Така, ако единият връх е София, а другият е Велико Търново то еднопосочният път (дъгата) ще се нарича София-Велико Търново. Всъщност това е един от класическите примери за приложение на графите – в задачи свързани с пътища.

Ако вместо със стрелки върховете са свързани с отсечки, то тогава отсечките ще наричаме **неориентирани ребра**, а графът – **неориентиран**. На практика едно неориентирано ребро от връх A до връх B представлява двупосочно ребро, еквивалентно на две противоположни ориентирани ребра между същите два върха:



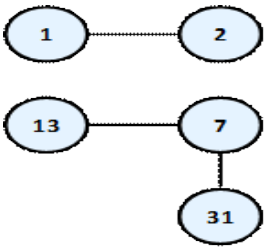
Два върха, свързани с ребро, ще наричаме **съседни**.

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)**. Ето как се изобразява претеглен граф:



- **Път в граф** ще наричаме последователност от върхове v_1, v_2, … , v_n, такава, че съществува ребро от v_i до v_{i+1} за всяко i от 1 до n-1. В нашия граф път е например последователността "1", "12", "19", "21". "7", "21" и "1" обаче не е път, тъй като не съществува ребро започващо от "21" и завършващо в "1".
- **Дължина на път** е броят на ребрата, свързващи последователността от върхове в пътя. Този брой е равен на броя на върховете в пътя минус единица. Дължината на примера ни за път "1", "12", "19", "21" е три.
- **Цена на път** в претеглен граф, ще наричаме сумата от теглата на ребрата участващи в пътя. В реалния живот пътят от София до Варна например е равен на дължината на пътя от София до Велико Търново плюс дължината на пътя от Велико Търново до Варна. В нашия пример дължината на пътя "1", "12", "19" и "21" е равна на 3 + 16 + 2 = 21.
- **Цикъл** е път, в който началният и крайният връх на пътя съвпадат. Пример за върхове, образуващи цикъл, са "1", "12" и "19". "1", "7" и "21" обаче не образуват цикъл.
- **Примка** ще наричаме ребро, което започва от и свършва в един и същ връх. В нашия пример върха "14" има примка.

Свързан неориентиран граф наричаме неориентиран граф, в който съществува път от всеки един връх до всеки друг. Например следният граф не е свързан, защото не съществува път от "1" до "7".



И така, вече имаме достатъчно познания, за да дефинираме понятието дърво по още един начин – като специален вид граф:

Дърво – неориентиран свързан граф без цикли.

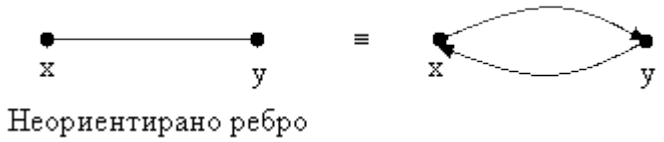
Граф - I част

1. Определения

Графът е нелинейна, динамична структура от данни, изградена от възли (върхове) и връзки между тях наречени ребра (клони, дъги).

В този смисъл графът се дефинира като съвкупност от две множества – множеството на възлите **V** и множеството на ребрата **E**:**G(V,E)**.

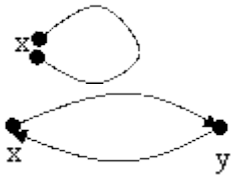
E може да се представи като множество от двойки **(x,y)**, където **x** и **y** принадлежат на **V. x** и **y** се наричат съседни възли**.** Когато двойката е наредена, т.е. **(x,y)≠(y,x)**, то реброто се нарича **ориентирано**. Когато двойката е ненаредена, т.е. **(x,y)=(y,x)**, то реброто се нарича **неориентирано**. Всяко неориентирано ребро може да се представи като съвкупност от две ориентирани ребра с различни посоки.



Неориентирано ребро

Ориентирано ребро

Бримка – ребро, за което началния и крайния възел съвпадат.



Паралелни ребра- ребра, които свързват едни и същи възли.

2. Неориентиран граф

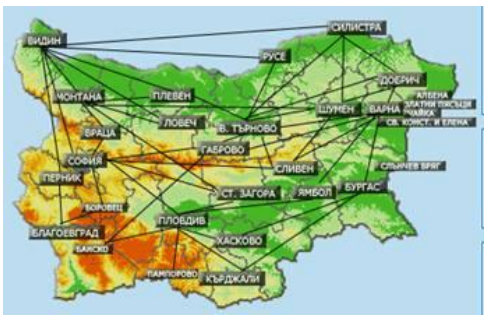
Дефиниция: Краен неориентиран граф се нарича наредената двойка *(V,E),* където:

- *V = {v1,v2,…,vn}* е крайно множество от върхове
- *E = {e1, e2, …,em}* е крайно множество от неориентирани ребра. Всеки елемент *ek ∈ E (k=1,2,…,m)* е ненаредена двойка *(vi,vj),* където *vi,vj∈V,1≤i,j≤ n.*

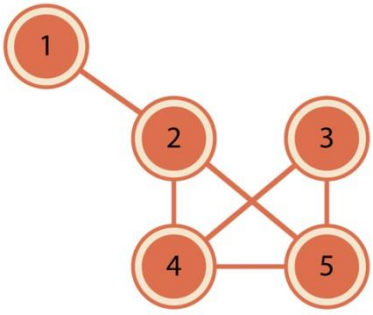
Ako освен това е зададена функция *f(i,j)*, съпоставяща целочислена стойност на всяко ребро *(i,j)∈E,f(i,j)=f(j,i),* графът се нарича претеглен неориентиран граф.

Почти всяка съвкупност от обекти с дефинирани връзки между тях може да бъде представена като граф. Примери:

1) Транспортната карта на България и по-големите градове в нея.



2) Компютърна мрежа може да бъде представена чрез неориентиран граф, в който компютрите са върхове, а всяко ребро между два върха показва, че съответните компютри са пряко свързани.
3) Множеството от страниците в Интернет.



фиг.1 Неориентиран граф

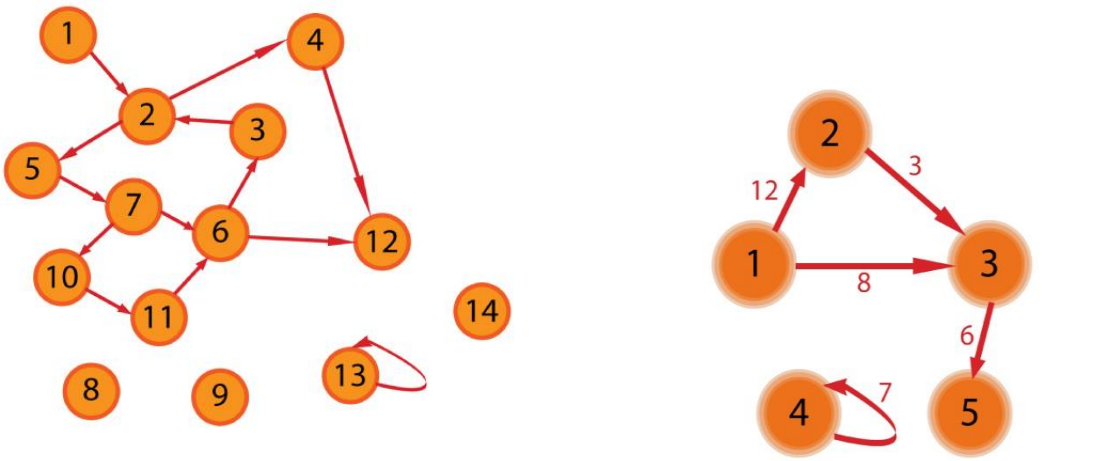
3. Ориентиран граф

Дефиниция: Краен ориентиран граф се нарича наредената двойка *(V,E),* където:

- *V = {v1,v2,…,vn}* е крайно множество от върхове.
- *E = {e1,e2,…,em}* е крайно множество от ориентирани ребра. Всеки елемент *ek ∈ E (k = 1,2,…,m)* е наредена двойка *(vi,vj), vi,vj∈V,1≤i,j≤n.*

Ако в допъление е дадена числова функция *f:E→R,* съпоставяща на всяко ребро *ек тегло f(ek),* графът се нарича претеглен. Понякога, когато няма опасност от двусмислие се казва само ребра вместо ориентирани ребра. Някои автори използват термина дъга за ориентирано ребро, а ребро - за неориентирано ребро.

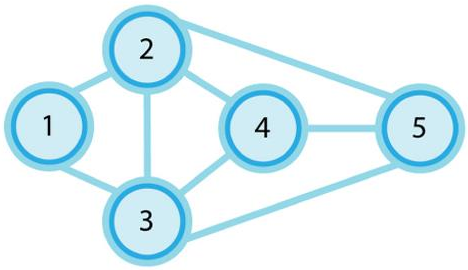
Най-често, ориентиран граф се представя графично в равнината чрез множество от точки (кръгчета), означаващи върховете му и свързващи ги стрелки - ребра на графа.



фиг.1 Ориентиран граф

4. Дефиниции

- Даден е ориентиран (неориентиран) граф *G(V,E).* Ако в множеството на ребрата му се допуска повторение (т.е. *E е* мултимножество*)*, *G* се нарича мултиграф.
- Даден е ориентиран граф *G(V,E).* Два върха *i* и *j (i,j ∈ V)* се наричат съседни*,* ако поне едно от ребрата *(i,j)* и *(j,i)* принадлежи на *E*. В такъв случай още казваме, че *i* и *j* са *краища* за ребрата *(i,j)* и *(j,i)* . За всяко ребро *(i,j)* върхът *i* се нарича предшественик на *j*, а *j* - наследник на *i*. Всеки от върховете *i* и *j* се нарича инцидентен с реброто *(i,j)*. Казваме, че две ребра са инцидентни, когато са инцидентни с един и същи връх. В случай на неориентиран граф, понятията съседен връх, край на ребро, инцидентност на върхове и ребра се дефинират аналогично.
- Път в ориентиран (неориентиран) граф G(*V*, *E)* се нарича последователност от върхове *v1,v2,…vk* такива, че за всяко *i = 1,2,…k-1* е изпълнено *(vi,vi-1) е от E.* Върховете *vi* и *vk* се наричат краища на пътя. Ако *v1 = vk*, пътят се нарича цикъл*.* Ако за всяко *i≠j(1≤i,j≤k)⇒vi≠vj*, то пътят се нарича прост. Съответно, ако *v1 = vk,* и всички останали върхове са различни, цикълът се нарича прост*.* Когато граф съдържа поне един цикъл, той се нарича цикличен, а в противен случай – ацикличен*.*
- **Цикъл (контур)-** път, за който началния и крайния възел съвпадат.
- **Прост цикъл** – цикъл, в който няма повтарящи се възли.



<u>Пътища</u>
П1: 5-4-2-1
П2: 1-2-3-4-5
П3: 1-2-4-3-2-1
П4: 1-2-4-3-1
П3 и **П4** - цикли
П4 - прост цикъл

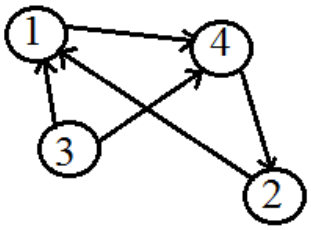
- **Хамилтонов цикъл** – цикъл, който включва всички възли на графа точно по веднъж.
- **Ойлеров цикъл** – цикъл, който включва всички ребра на графа точно по веднъж.

Не всички графи притежават хамилтонов и ойлеров цикъл. Поради това, графите които притежават хамилтонов цикъл се наричат **хамилтонови графи**, а тези които притежават ойлеров цикъл- **ойлерови графи**

Граф - II част

1. Логическо представяне

Граф се задава с двойка числа от множество от върхове и множество от ребра. Ако двойката е наредена, върховете се наричат дъги, а графът се нарича ориентиран. Ако двойката е ненаредена графът е неориентиран. Ребрата /дъгите могат да се свързват с етикети от имена или числа, наречени тегла. Такъв граф се нарича притеглен.



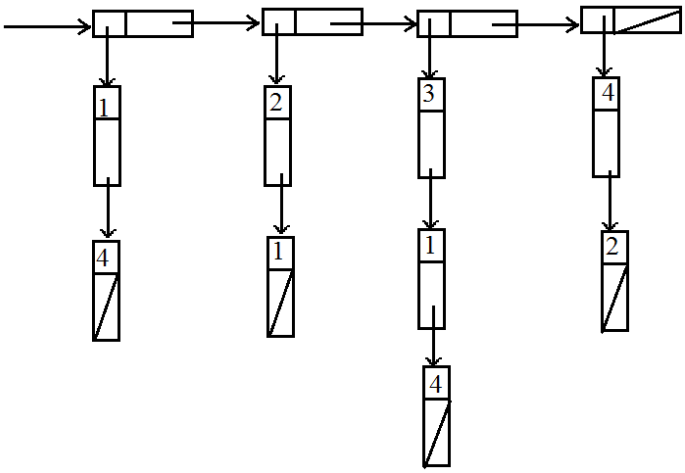
2. Физическо представяне

а) Последователно представяне - Реализира се чрез матрица на наследство. За граф с n върха матрицата има n реда и n стълба. Това представяне не се използва често,

тъй като при големи графи матриците заемат много памет. Матрица с голям брой нулеви елементи се нарича разредена.

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

b) Свързано представяне - Реализира се чрез списък на наследници. Използва свързани списъци от толкова елемента, колкото са върховете на графа. Елементите на списъка са списъци, като всеки подспидък съдържа всички върхове, до които има дъги от съответния връх



3. *Видове представения на графи*

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 $\mathbf{v_i}$ до $\mathbf{v_j}$. 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 $\mathbf{v_i}$ до $\mathbf{v_j}$, 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. а матрицата се нарича
- **Матрица на теглата -** ако графът е претеглен, в позиция **g[i][j]** се записва теглото на даденото ребро. 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 ($\mathbf{v_i}$, $\mathbf{v_j}$) 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 т.е. е ($\mathbf{v_i}$, $\mathbf{v_i}$), то на позиция **i** записваме 2. Ако графът, който искаме да

представим е ориентиран и искаме да представим ребро от $\mathbf{v_i}$ до $\mathbf{v_j}$, то на позиция **i** пишем 1, а на позиция **j** пишем -1.

4. Основни приложения и задачи

Графите се използват за моделиране на много ситуации от реалността, а задачите върху графи моделират множество реални проблеми, които често се налага да бъдат решавани. Ще дадем само няколко примера:

- **Карта на град** може да се моделира с ориентиран претеглен граф. На всяка улица се съпоставя ребро с дължина съответстваща на дължината на улицата и посока – посоката на движение. Ако улицата е двупосочна може да й се съпоставят две ребра за двете посоки на движение. На всяко кръстовище се съпоставя връх. При такъв модел са естествени задачи като търсене на най-кратък път между две кръстовища, проверка дали има път между две кръстовища, проверка за цикъл (дали можем да се завъртим и да се върнем на изходна позиция), търсене на път с минимален брой завои и т.н.
- **Компютърна мрежа** може да се моделира с неориентиран граф, чиито върхове съответстват на компютрите в мрежата, а ребрата съответстват на комуникационните канали между компютрите. На ребрата могат да се съпоставят различни числа, примерно капацитет на канала или скорост на обмена и др. Типични задачи при такива модели на компютърна мрежа са проверка за свързаност между два компютъра, проверка за двусвързаност между две точки (съществуване на двойно-подсигурен канал, който остава при отказ на който и да е компютър) и др. В частност Интернет може да се моделира като граф, в който се решават задачи за маршрутизация на пакети, които се моделират като задачи за графи.
- **Речната система** в даден регион може да се моделира с насочен претеглен граф, в който всяка река се състои от едно или няколко ребра, а всеки връх съответства на място, където две или повече реки се вливат една в друга. По ребрата могат да се съпоставят стойности, свързани с количеството вода, което преминава по тях. Естествени при този модел са задачи като изчисление на обемите вода, преминаващи през всеки връх и предвиждане на евентуални наводнения при увеличаване на количествата.

5. Операции с графи

Основните операции, свързани с построяване и модифициране на граф, са следните:

- създаване на празен граф.
- добавяне/премахване на връх.
- добавяне/премахване на ребро.
- проверка за съществуване на връх
- проверка за съществуване на ребро
- намиране на наследниците на даден връх

Пример: реализация за граф, представен с матрица на съседство (матрица на теглата).

```
#define MAXN 200 /* Максимален брой върхове в графа */
unsigned n; /* Брой върхове в графа */
int A[MAXN][MAXN]; /* Матрица на теглата на графа */

A[i][i] = k; /* Модифициране на теглото на върха i */
A[i][j] = k; /* Добавяне на ребро с тегло k, свързващо върховете i и j */
A[j][i] = k; /* ако графът не е ориентиран */

A[i][j] = 0; /* Премахване на ребро, свързващо върховете i и j */
/* Проверка дали има ребро между върховете i и j */
if (A[i][j] != 0) { /* има ребро */;
}
 else { /* няма ребро */; }
/* Намиране на всички наследници на върха i */
for (k = 0; k < n; k++) if (k != i) {
if (A[i][k] != 0) { /* върхът k е наследник на i */ ; }
```

}

Дефиниция - *Диаметър* на граф се нарича максималното число *k*, такова че най-крат-кия прост път (пътят, в който участват минимален брой върхове) между произволни два върха *i*,*j*□*V* съдържа поне *k* върха. Така например, ако разглеждаме страниците в Интернет и връзките помежду им като ори-ентиран граф, то диаметърът му *k* е максималният брой страници, през които трябва да преминем, започвайки от произволна страница и следвайки хипервръзки, за да достигнем до произволна друга страница в мрежата. Проучванията показват, че "диаметърът на Интернет" е приблизително равен на 19

Дефиниция - *Силно* (*слабо*) *свързана компонента* в ориентиран граф *G*(*V*, *E*) се нарича всеки *подграф G*'(*V*', *E*'), за който е изпълнено:

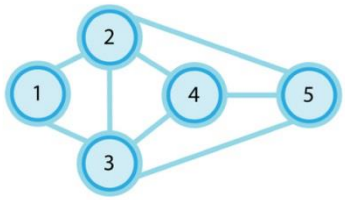
1) *G*' е силно (слабо) свързан граф.
2) Не съществува друг силно (слабо) свързан собствен подграф *G*''(*V*'',*E*'') на *G*, такъв, че *G*' да бъде подграф на *G*''.

Аналогично се дефинира *свързана компонента* в неориентиран граф.
Дефиниция - Неориентиран свързан граф без цикли се нарича *дърво*. Ако допълни-телно изберем някой връх от дървото за корен, то получената структура се нарича *кореново дърво*.
Дефиницията е алтернативна на разгледаната в *2.3.* дефиниция на кореново дърво.
Дефиниция - *Покриващо* (*обхващащо*) *дърво* в свързан неориентиран граф *G*(*V*,*E*) се нарича всеки свързан ацикличен подграф *G*'(*V*, *E*') на *G*.

6. Представяне на графи

Графът може да бъде записан (представен) във външната памет (външно представяне) и във вътрешната памет (вътрешно представяне).

1) **Външно представяне – 2 подхода:**



- записва се номера на възела и след тире се изреждат неговите съседи;

Пример: 1 - 2 3; 2 - 1 3 4 5; 3 - 1 2 4 5; 4 - 2 3 5; 5 - 2 3 4

- на всеки ред от файла се записват съседите на възела, чийто номер съответства на номера на реда.

Пример:
```
2 3
1 3 4 5
1 2 4 5
2 3 5
2 3 4
```

2) **Вътрешно представяне**

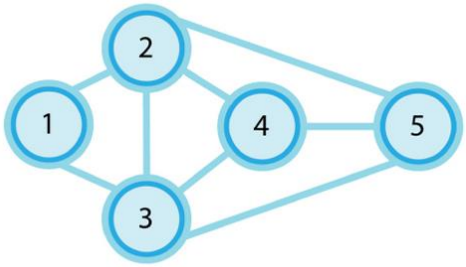
Вътрешното представяне може да се извърши по два основни начина: чрез матрица на съседство и чрез списъци на съседство.

В компютърните реализации един граф се представя с *множеството от върховете* си V={1,2,...,n} и *матрица на съседство* a[i][j], в която a[i][j]=1, когато съществува дъга от връх i към връх j и a[i][j]=0 в противен случай. В графите, където са зададени тегла по ребрата, съответната стойност може да бъде записана като стойност a[i][j], при което този масив няма да съдържа само нули и единици. Алтернативен начин за представяне на граф е чрез използване на множествата s[j], съдържащи съседите за всеки връх j. Множеството s[j] съдържа номерата на всички върхове, които са непосредствено свързани с върха, номериран с j. В програмните реализации елементите на множеството s[j] обикновено се представят като елементи на двуиндексен масив s[j][1], s[j][2], ..., s[j][s[j][0]], където s[j][0] съдържа броя на съседите на върха с номер j.

7. Матрица на съседство

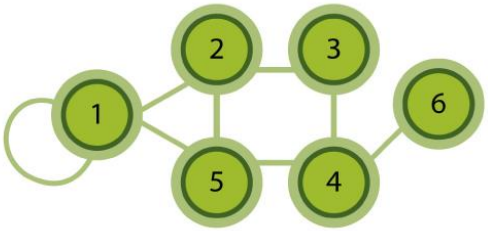
Един от най-често използваните начини за представяне на граф *G(V,E)* е посредством матрица на съседство. Нека *V={1,2,...,n}*. Матрицата на съседство за *G(V,E)* е *n* x *n* матрица *A*, където *a[i][j]=1* ако има дъга от *i* до *j* и *a[i][j]=0* в противен случай.

Предимството на представянето чрез матрица на съседство е, че се изисква постоянно време (само еднократен достъп до паметта) за определянето дали има или не дъга между два дадени върха. В случаите, когато с всяка дъга е свързана дължина или тегло, представянето с матрица на съседство може да се модифицира подходящо, така че *a[i][j]* да съдържа дължината или теглото вместо единица. Недостатък на представянето с матрица на съседство е, че използваната памет е n^2 клетки, даже и в случаите, когато графът има твърде малко ребра. Освен това, само еднократното преглеждане на елементите на матрицата изисква n^2 стъпки, което автоматично води до това, че използваните алгоритми ще бъдат квадратични, даже и за разредени графи.
Създава се матрица по редовете и колоните на която се поставят номерата на възлите. При битоничните матрици в местата на връзка се записва 1, а при тегловните матрици (за тегловен граф)- теглото на реброто.



	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**0**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**1**
4	**0**	**1**	**1**	**0**	**1**
5	**0**	**1**	**1**	**1**	**0**

Ако в графа няма бримки, по главния диагонал трябва да има само нули. Ако графът е неориентиран, матрицата е симетрична спрямо главния диагонал.
За представянето на графа са необходими n^2 клетки в паметта – доста неикономично.

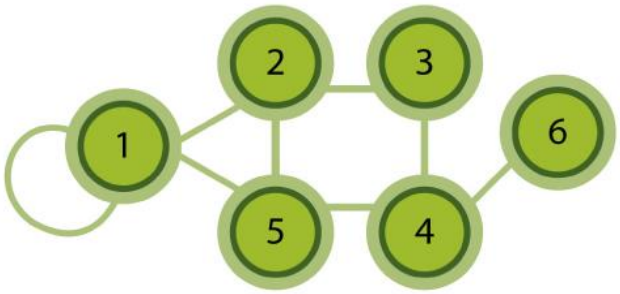


	1	2	3	4	5	6
1	1	**1**	**0**	**0**	**1**	0
2	**1**	**0**	**1**	**0**	**1**	0
3	**0**	**1**	**0**	**1**	**0**	0
4	**0**	**0**	**1**	**0**	**1**	1
5	**1**	**1**	**0**	**1**	**0**	0
6	0	0	0	1	0	0

8. Списъци на съседство

Алтернативно представяне на граф *G(V,E)* е посредством списъци на съседство. Казваме, че връх *j* е съседен на връх *i*, когато има дъга от *i* до *j*. За представянето на графа използваме масив от *n* елемента, в който *i*-тия елемент указва списъка от съседи (в произволен ред) на върха *i*. Следователно необходимата памет за представяне на граф с *n* върха и *m* ребра е от порядъка на *n+m*. Така се избягва недостатъка от използването на повече памет, отколкото е необходима. Недостатък на представянето чрез списъци на

съседство е, че определянето дали има ребро от *i* до *j* може да изисква до *n* стъпки, защото е свързано с последователно обхождане на списъка от съседи на връх *i*. За приложения, където тясното място е определянето дали има ребро между два върха, за предпочитане е използването на матрица на съседство.



1 --> {1, 2, 5}

2 --> {1, 3, 5}

3 --> {2, 4}

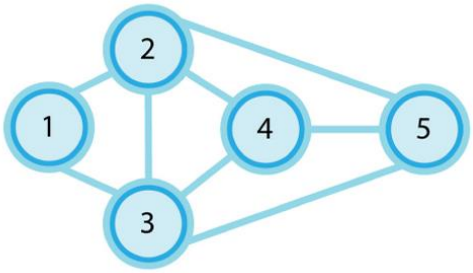
4 --> {3, 5, 6}

5 --> {1, 2, 4}

6 --> {4}

При списъците на съседство са възможни 3 подхода: статично(два масива), полудинамично (масив+списъци) и динамично (списъчни структури).

1) Статично представяне – използват се два масива: масив "съседи", където последователно се изреждат номерата на съседите на всеки един възел в графа. Масив "начало" - във всеки елемент се записва индекса, от който в масив "съседи" започва изреждането на съседите на възела, който има номер, равен на индекса на елемента в масив "начало".



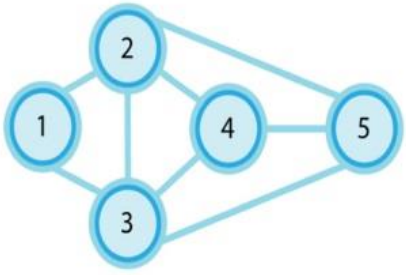
"начало"

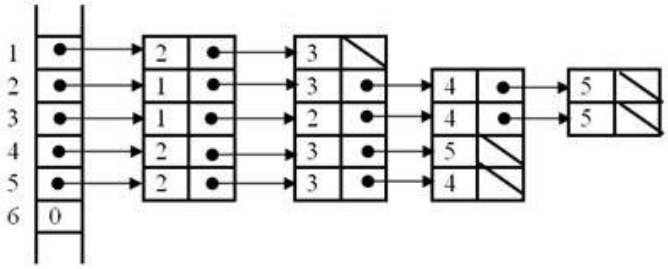
i	
1	1
2	3
3	7
4	11
5	14
6	0

"съседи"

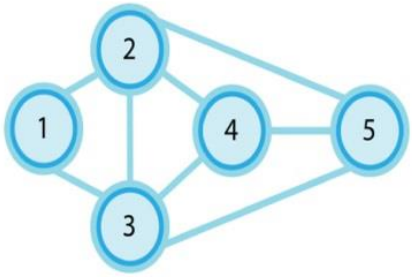
	j
възел 1	2
	3
възел 2	1
	3
	4
	5
възел 3	1
	2
	4
	5
възел 4	2
	3
	5
възел 5	2
	3
	4
	0

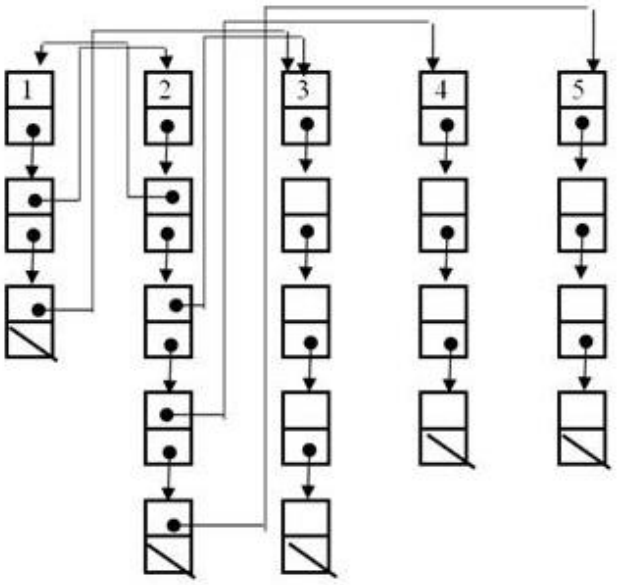
2) Полудинамично – в масива "начало" се записва адресът на записа на първия съсед. Всеки запис на съсед притежава по две полета: номер на възела и указател към следващия съсед. Указателят за следващ съсед на последния от списъка е нула.





3) Динамично – за всеки възел се създава запис с две полета: номер на възела и указател към двойка указатели. Първият указател от двойката съдържа адреса на съсед, а втория- адреса на следващата двойка указатели.





http://tu-utc.com/Webpages/E_learning/SAA/Tema5-TREE1.htm - линк в интернет с теория