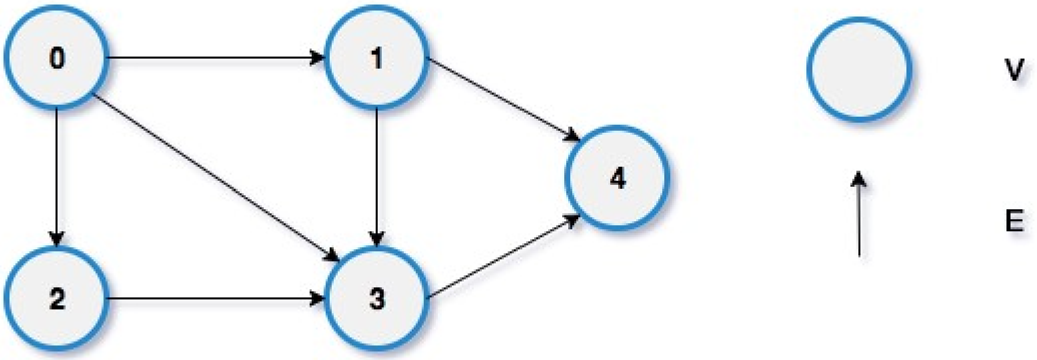
What is Graph Data Structure?

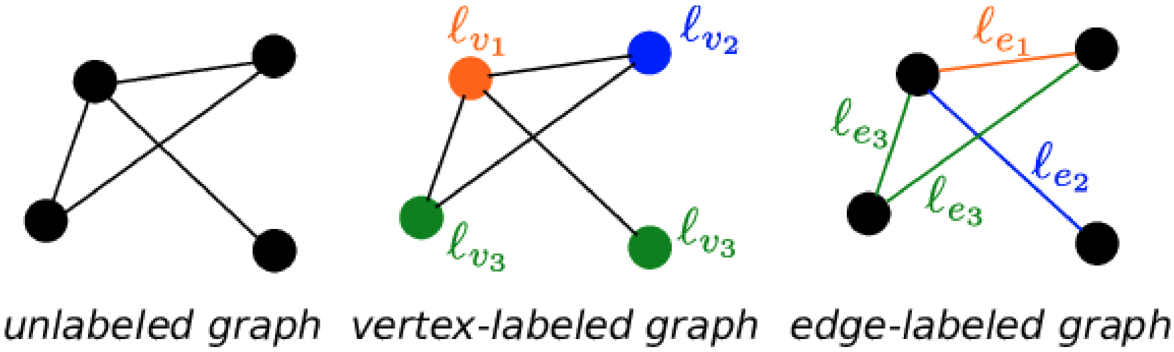
> A Graph is a non-linear data structure consisting of vertices and edges (or) Graph is composed of a set of vertices(V) and a set of edges(E).

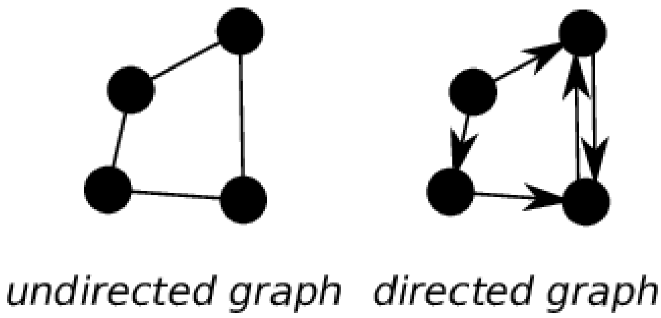
> The graph is denoted by G(E, V).



Vertices: Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes.Every node/vertex can be labeled or unlabelled.

Edges: Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph. Edges can connect any two nodes in any possible way. There are no rules. Sometimes, edges are also known as arcs.





Graphs are used to solve many real-life problems.Graphs are used to represent networks.,social networks like linkedIn, Facebook

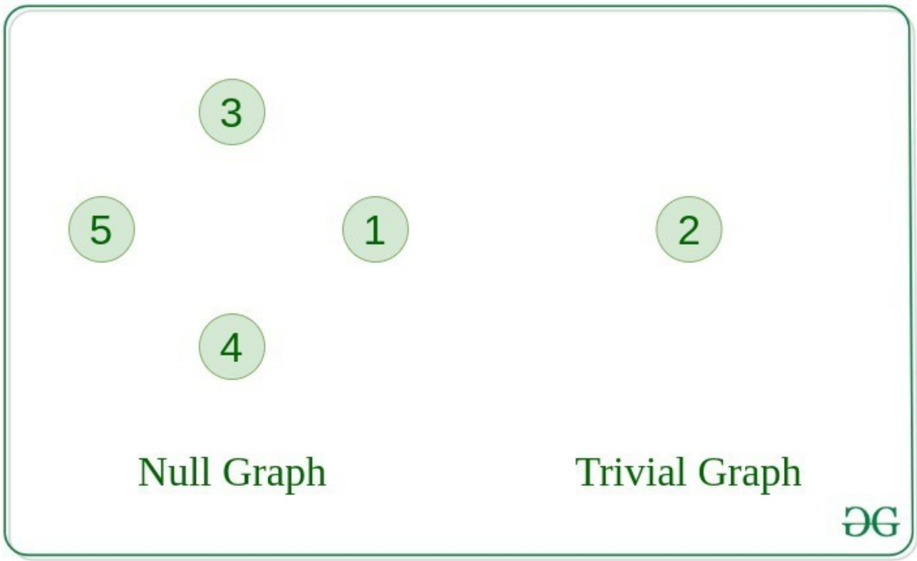
Types Of Graph

1. Null Graph

A graph is known as a null graph if there are no edges in the graph.

2. Trivial Graph

Graph having only a single vertex, it is also the smallest graph possible.

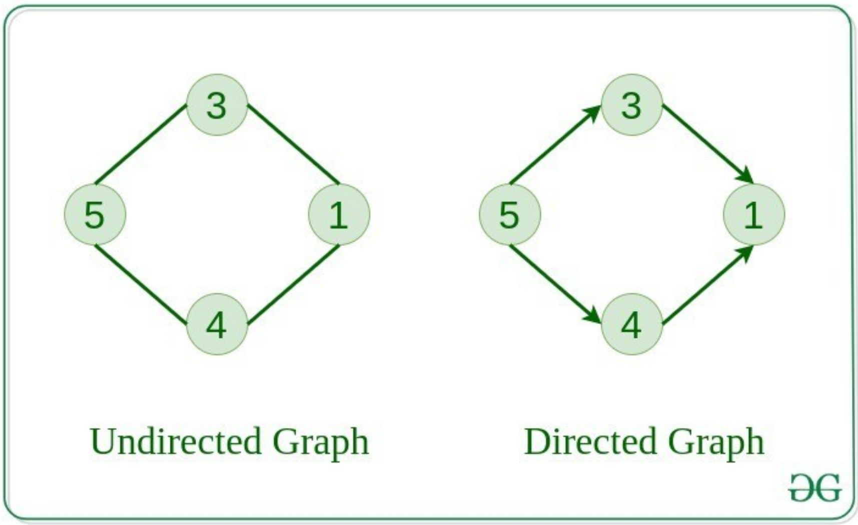


3. Undirected Graph

A graph in which edges do not have any direction. That is the nodes are unordered pairs in the definition of every edge.

4. Directed Graph

A graph in which edge has direction. That is the nodes are ordered pairs in the definition of every edge.

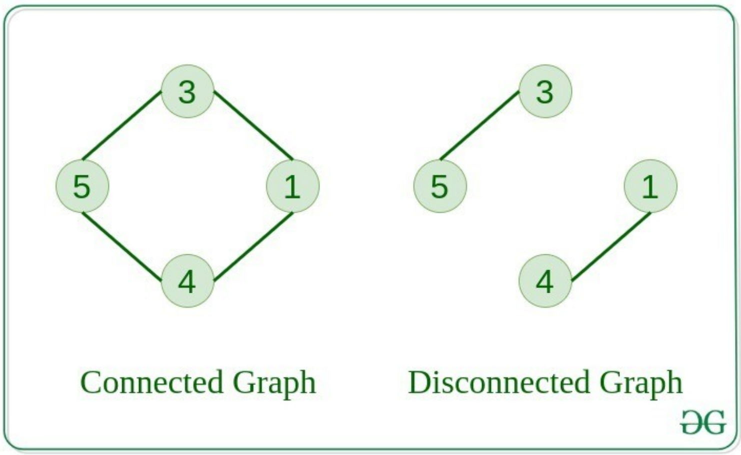


5. Connected Graph

The graph in which from one node we can visit any other node in the graph is known as a connected graph.

6. Disconnected Graph

The graph in which at least one node is not reachable from a node is known as a disconnected graph.

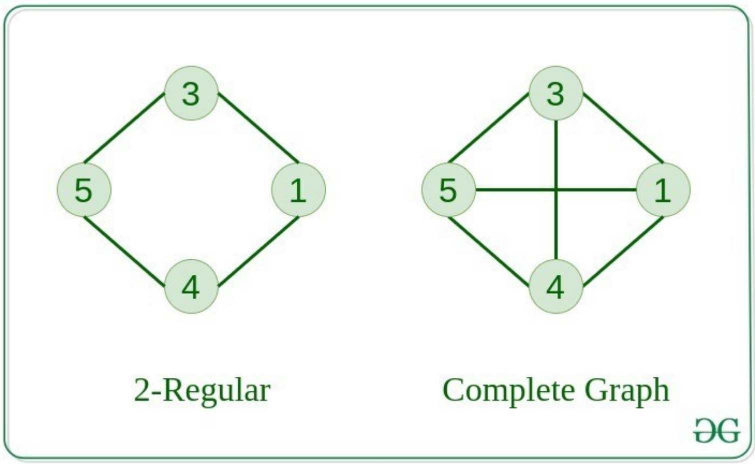


7. Regular Graph

The graph in which the degree of every vertex is equal to K is called K regular graph.

8. Complete Graph

The graph in which from each node there is an edge to each other node.

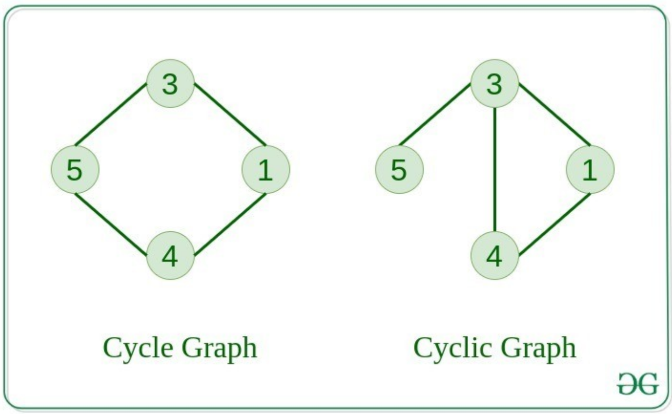


9. Cycle Graph (loop)

The graph in which the graph is a cycle in itself, the degree of each vertex is 2.

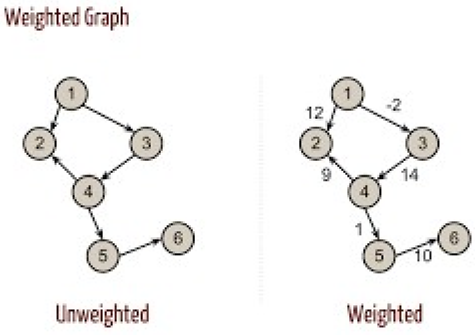
10. Cyclic Graph

A graph containing at least one cycle is known as a Cyclic graph.



11. Weighted Graph

- A graph in which the edges are already specified with suitable weight is known as a weighted graph.
- Weighted graphs can be further classified as directed weighted graphs and undirected weighted graphs

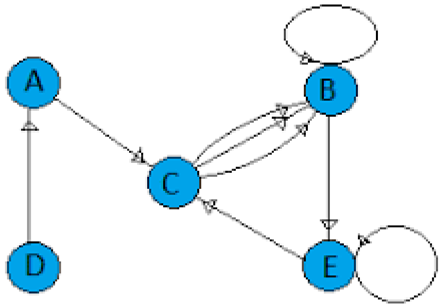


12. In-degree

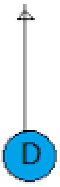
- In-degree of a vertex is the number of edges coming to the vertex.

13. Out-degree

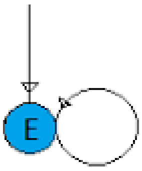
Out-degree of a vertex is the number edges which are coming out from the vertex.



Directed Graph



Ou-degree



In-degree

14. Pendant Vertex

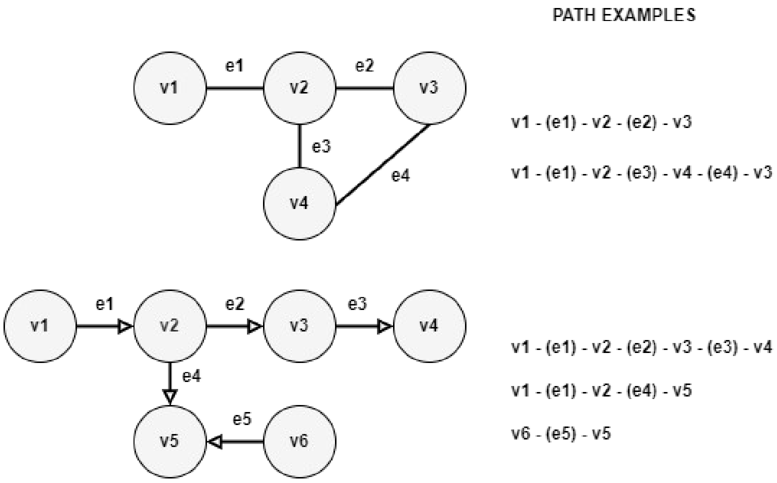
- A vertex with degree one is called a pendant vertex.

15. Isolated Vertex

- A vertex with degree zero is called an isolated vertex.

16. path in a graph

- path in a graph is a finite or infinite sequence of edges which joins a sequence of vertices which, by most definitions, are all distinct


PATH EXAMPLES
v1
e1
v2
e2
v3
e3
v4
e4
v1 - (e1) - v2 - (e2) - v3
v1 - (e1) - v2 - (e3) - v4 - (e4) - v3
v1
e1
v2
e2
v3
e3
v4
e4
v5
e5
v6
v1 - (e1) - v2 - (e2) - v3 - (e3) - v4
v1 - (e1) - v2 - (e4) - v5
v6 - (e5) - v5

<u>Graph data structure:</u>

- Representation Graph (Adjacency matrix, Linked List)
- *Breadth-First* Search (*BFS*)
- Depth-First Search (DFS)
- Dijkstra’s Algorith
- All pairs shortest path - Floyd-Warshall algorithm

<u>Sorting:</u>

- Insertion Sort.
- Shell Sort.
- Radix Sort.

Breadth-First Search Algorithm (BFS) :

```
BFS(input : graph G, int : x)
{
    Queue q;
    int y,z;
    print x;
    Enqueue(q,x);
    while(q is not empty)
    {
        z = gettop(q);
        Dequeue(q);
        for(unvisited neighbors y of z)
        {
            print y;
            Enqueue(q,y);
        }
    }
}
```

Depth-First Search Algorithm (DFS) :

```
DFS(input : graph G, int : x)
{
    Stack s;
    int t;
    print x;
    push(s,x);
    while(s not empty)
    {
        t = peek(s);
        Dequeue(q);
        if( t has  unvisited neighbors)
        {
            print y;
            push(s,y);
        }
        else pop(s);
    }
}
```

```
DFS(input : graph G, int : x)
{
    print x;
    for each  unvisited adjacent vertex w of x
    {
         DFS(G,w);
    }
}
```

Dijkstra’s Algorithm :

```
inittable(input : graph G, vertex start)
{
        int i;
        for( i=0; i<numofvertex; i++){
                vertex[i].known  = false;
                vertex[i].dist  = 99999;
                vertex[i].path  = -1;
        }
        vertex[start].dist  = 0;
}

dijkstra(input : graph G, vertex start)
{
        while(there is an unknown distance vertex)
        {
                vertex v = smallest unknown distance vertex;
                v.know = true;
                for each vertex w adjacent to v
                        if(!w.know)
                        {
                                int $c_{vw}$ = cost of edge from v to w;
                                if(v.dist + $c_{vw}$ < w.dist)
                                {
                                        w.dist = v.dist + $c_{vw}$ ;
                                        w.path = v;
                                ]
                        }
        }
}
```

Dijkstra’s PathPrinting Algorithm :

```
PathPrint(input : graph G, vertex V)
{
        if(G->path != -1)
        {
                printpath(G->path);
                print “to”;
        }
        print G->path;
}
```

Floyd-Warshall Algorithm :

```
Floyd-Warshall()
{
    for k=1 to n do
    {
        for i=0 to n do
        {
            for j=1 to n do
            {
                if(d[i,j] > d[i,k] + d[k,j];
                pred[i,j] = k;
            }
        }
    }
}
```

Find Path in Floyd-Warshall Algorithm :

```
FindPath(input : int i,j)
{
    static t = o;
    if(pred[i,j] == 0)
    {
        print i,j;
        t += dist(i,j);
    }
    else
    {
        FindPath(i, pred[i,j]);
        FindPath(pred[i,j], j);
    }
}
```

Insertion Sort:

```
insertion_sort(arrADT)
{
    a[0] = -99;
    for(i=2 to n)
    {
        j=i;
        while(arr[j] < arr[j-1])
        {
            Swap(arr[i], arr[j-i]);
            j = j-1;
        }
    }
}
```

Shell Sort:

```
shell_sort(var A : array[n] of integer)
{
    int var i,j;
    incr = n/2;
    while(incr > 0)
    {
        for i=incr+1 to n do
        {
            j = i-incr;
            while(j > 0){
                if(a[j] > a[j +incr]){
                    Swap(A[i], A[j-incr]);
                    j = j – incr;
                }
                else  j =0;
        }
        incr = incr / 2;
    }
}
```

Redix Sort:

```
redix_sort(arr)
{
    max = largest element in the given array;
    d = no.of.digits in the largest element;
    create d buckets of size 0-9;
    for i=0 to d;
        sort the array element according to the digits at the ith place;
}
```