in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

Traversal algorithms And complexity analysis:

Graph Algorithms

In this article, you would be learning a brief explanation of some of the most used graph algorithms, which have massive applications in today's words. Graphs cover most high-level data structure techniques that one experiences while implementing them and to know which graph algorithm is best

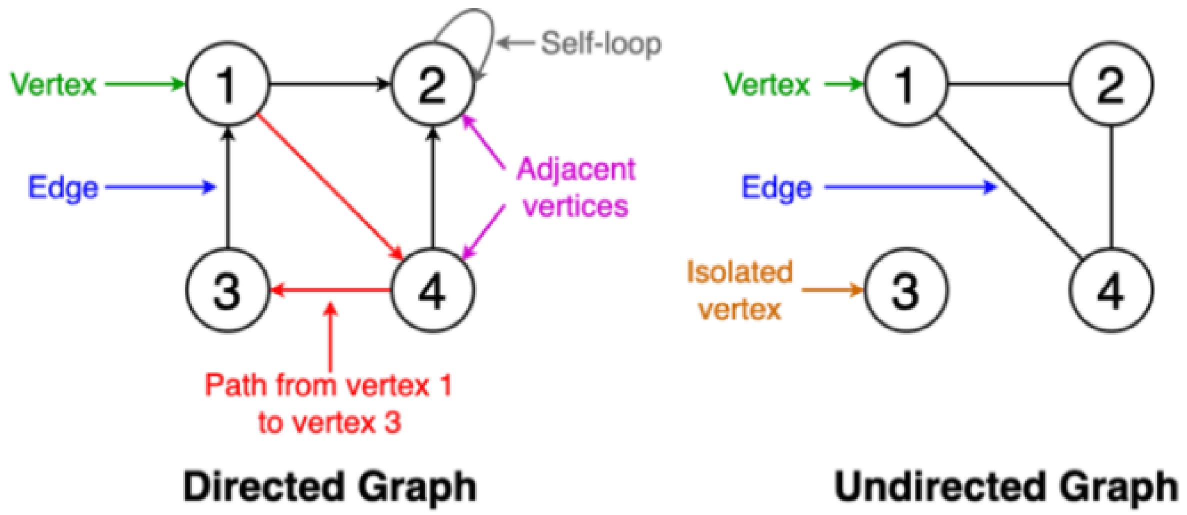
for the moment effectively is what you would be learning here. First, let's get a clear idea from the very basics about graphs.
What is a Graph?
A graph is a **unique data structure** in programming that consists of finite sets of nodes or vertices and a set of edges that connect these vertices to them. At this moment, adjacent vertices can be called those vertices that are connected to the same edge with each other. In simple terms, a graph is a visual representation of vertices and edges sharing some connection or relationship. Although there are plenty of graph algorithms that you might have been familiar with, only some of them are put to use. The reason for this is simple as the standard graph algorithms are designed in such a way to solve millions of problems with just a few lines of logically coded technique. To some extent, one perfect algorithm is solely optimized to achieve such efficient results.

Types of Graphs

There are various types of graph algorithms that you would be looking at in this article but before that, let's look at some types of terms to imply the fundamental variations between them.
Order: Order defines the total number of vertices present in the graph.

Size: Size defines the number of edges present in the graph.
Self-loop: It is the edges that are connected from a vertex to itself.
Isolated vertex: It is the vertex that is not connected to any other vertices in the graph.
Vertex degree: It is defined as the number of edges incident to a vertex in a graph.
Weighted graph: A graph having value or weight of vertices.
Unweighted graph: A graph having no value or weight of vertices.
Directed graph: A graph having a direction indicator.

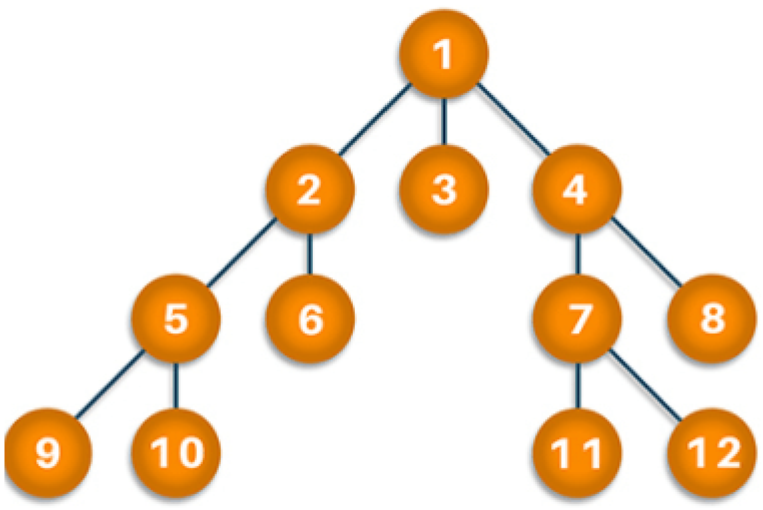
Undirected graph: A graph where no directions are defined.



Let's now carry forward the main discussion and learn about different types of graph algorithms.

Breadth-First Search

Traversing or searching is one of the most used operations that are undertaken while working on graphs. Therefore, in **breadth-first-search** (BFS), you start at a particular vertex, and the algorithm tries to visit all the neighbors at the given depth before moving on to the next level of traversal of vertices. Unlike trees, graphs may contain cyclic paths where the first and last vertices are remarkably the same always. Thus, in BFS, you need to keep note of all the track of the vertices you are visiting. To implement such an order, you use a queue data structure which First-in, First-out approach. To understand this, see the image given below.



BREADTH FIRST SEARCH

Algorithm

1. Start putting anyone vertices from the graph at the back of the queue.
2. First, move the front queue item and add it to the list of the visited node.
3. Next, create nodes of the adjacent vertex of that list and add them which have not been visited yet.
4. Keep repeating steps two and three until the queue is found to be empty.

Pseudocode

```
1. Set all nodes to "not visited";
2.    q = new Queue();
3.    q.enqueue(initial node);
4.    while ( q ? empty ) do
5.    {
6.       x = q.dequeue();
7.       if ( x has not been visited )
8.       {
9.          visited[x] = true;         // Visit node x !
10.
11.             for ( every edge (x, y)  /* we are using all edges
   ! */ )
12.               if ( y has not been visited )
13.              q.enqueue(y);      // Use the edge (x,y) !!!
14.           }
15.        }
```

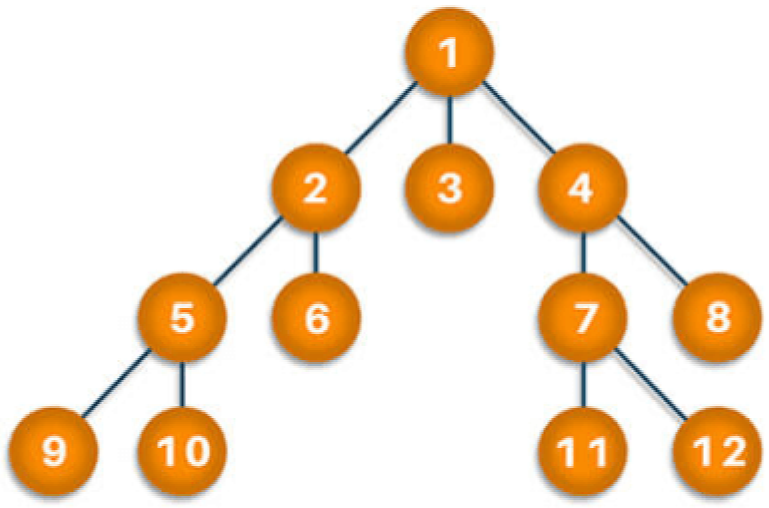
Complexity: 0(V+E) where V is vertices and E is edges.

Applications

BFS algorithm has various applications. For example, it is used to determine the **shortest path** and **minimum spanning tree.** It is also used in web crawlers to creates web page indexes. It is also used as powering search engines on social media networks and helps to find out peer-to-peer networks in BitTorrent.

Depth-first search

In depth-first-search (DFS), you start by particularly from the vertex and explore as much as you along all the branches before backtracking. In DFS, it is essential to keep note of the tracks of visited nodes, and for this, you use stack data structure.



DEPTH FIRST SEARCH

Algorithm

1. Start by putting one of the vertexes of the graph on the stack's top.
2. Put the top item of the stack and add it to the visited vertex list.
3. Create a list of all the adjacent nodes of the vertex and then add those nodes to the unvisited at the top of the stack.
4. Keep repeating steps 2 and 3, and the stack becomes empty.

Pseudocode

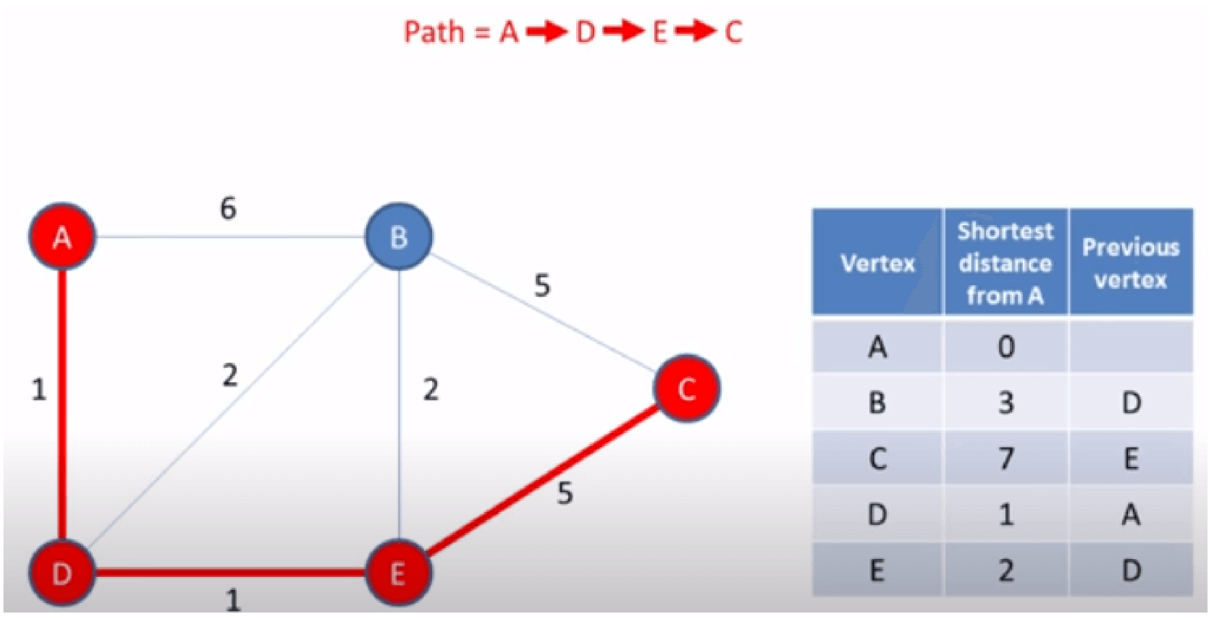
```
1. DFS(G,v)   ( v is the vertex where the search starts )
2.        Stack S := {};   ( start with an empty stack )
3.        for each vertex u, set visited[u] := false;
4.        push S, v;
5.        while (S is not empty) do
6.           u := pop S;
7.           if (not visited[u]) then
8.              visited[u] := true;
9.              for each unvisited neighbour w of uu
10.                        push S, w;
11.                end if
12.             end while
13.          END DFS()
```

Applications

DFS finds its application when it comes to finding paths between two vertices and detecting cycles. Also, topological sorting can be done using the DFS algorithm easily. DFS is also used for one-solution puzzles.

Dijkstra's shortest path algorithm

Dijkstra's shortest path algorithm works to find the minor path from one vertex to another. The sum of the vertex should be such that their sum of weights that have been traveled should output minimum. The shortest path algorithm is a highly curated algorithm that works on the concept of receiving efficiency as much as possible. Consider the below diagram.



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Algorithm

1. Set all the vertices to infinity, excluding the source vertex.
2. Push the source in the form (distance, vertex) and put it in the min-priority queue.
3. From the priority, queue pop out the minimum distant vertex from the source vertex.
4. Update the distance after popping out the minimum distant vertex and calculate the vertex distance using (vertex distance + weight < following vertex distance).
5. If you find that the visited vertex is popped, move ahead without using it.
6. Apply the steps until the priority queue is found to be empty.

Pseudocode

```
function dijkstra(G, S)
    for each vertex V in G
```

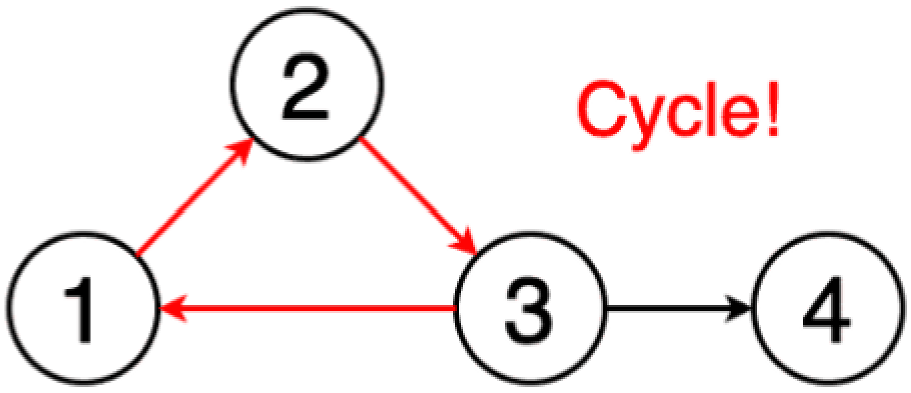
```
3.        distance[V] <- infinite
4.        previous[V] <- NULL
5.        If V != S, add V to Priority Queue Q
6.    distance[S] <- 0
7.    while Q IS NOT EMPTY
8.        U <- Extract MIN from Q
9.        for each unvisited neighbour V of U
10.            tempDistance <- distance[U] + edge_weight(U, V)
11.            if tempDistance < distance[V]
12.                distance[V] <- tempDistance
13.                previous[V] <- U
14.        return distance[], previous[]
```

Applications

Dijkstra's shortest path algorithm is used in finding the distance of travel from one location to another, like Google Maps or Apple Maps. In addition, it is highly used in networking to outlay min-delay path problems and abstract machines to identify choices to reach specific goals like the number game or move to win a match.

Cycle detection

A cycle is defined as a path in graph algorithms where the first and last vertices are usually considered. For example, if you start from a vertex and travel along a random path, you might reach the exact point where you eventually started. Hence, this forms a chain or cyclic algorithm to cover along with all the nodes present on traversing. Therefore, cycle detection is based on detecting this kind of cycle. Consider the below image.



Pseudocode

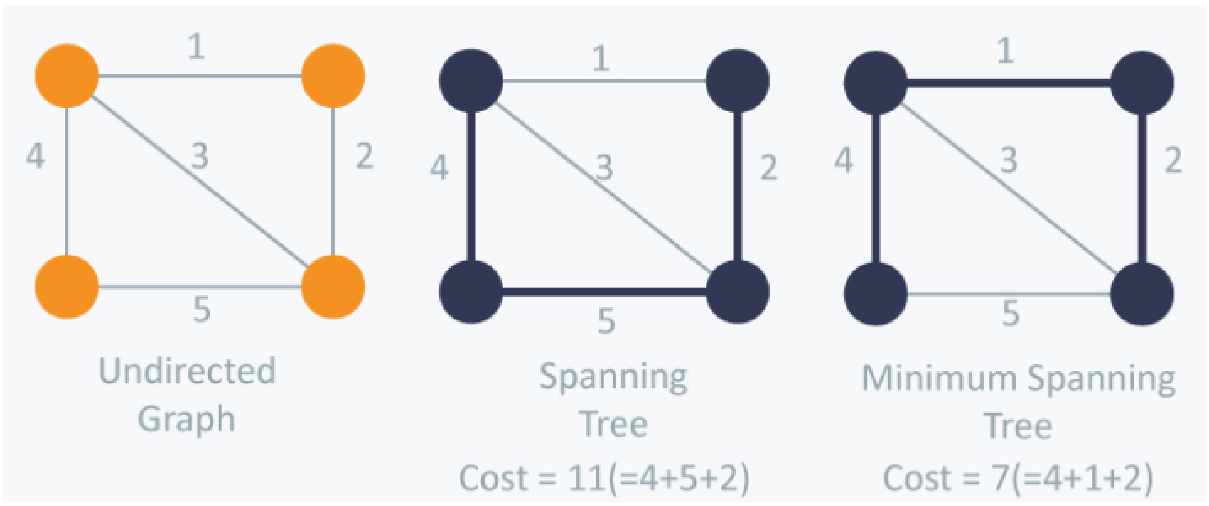
```
Brent's Cycle Algorithm Example
def brent(f, x0):
    # main phase: search successive powers of two
    power = lam = 1
    tortoise = x0
    hare = f(x0)  # f(x0) is the element/node next to x0.
    while tortoise != hare:
        if power == lam:  # time to start a new power of two?
            tortoise = hare
                power *= 2
                lam = 0
            hare = f(hare)
            lam += 1
        # Find the position of the first repetition of length ?
        tortoise = hare = x0
        for i in range(lam):
        # range(lam) produces a list with the values 0, 1, ... , lam-1
            hare = f(hare)
        # The distance between the hare and tortoise is now ?.
        # Next, the hare and tortoise move at same speed until they agree
        mu = 0
        while tortoise != hare:
            tortoise = f(tortoise)
            hare = f(hare)
            mu += 1
        return lam, mu
```

Applications

Cyclic algorithms are used in message-based distributed systems and large-scale cluster processing systems. It is also mainly used to detect deadlocks in the concurrent system and various cryptographic applications where the keys are used to manage the messages with encrypted values.

Minimum Spanning Tree

A minimum spanning is defined as a subset of edges of a graph having no cycles and is well connected with all the vertices so that the minimum sum is availed through the edge weights. It solely depends on the cost of the spanning tree and the minimum span or least distance the vertex covers. There can be many minimum spanning trees depending on the edge weight and various other factors.


1
4
3
2
5
Undirected Graph
1
4
3
2
5
Spanning Tree
Cost = 11(=4+5+2)
1
4
3
2
5
Minimum Spanning Tree
Cost = 7(=4+1+2)