Linked list:

It is a sequence of data structures, which are connected together via links.
The elements are not stored at contiguous memory locations.

The code written for linked list is to perform the operation of reversing first N elements using Linked list.

Input:

1 2 3 4 5 6

Output:

Linked List before reversing first 3 elements: 1 2 3 4 5 6

Linked List after reversing first 3 elements: 3 2 1

Queue:

A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

Queue is referred to be as First In First Out list.

The code written for Queue is to reverse the elements of queue, and add another elements to the said queue and, then revers hem again.

Input: 1 2 3 4 5

Input: 100 200

Output: Queue elements are: 1 2 3 4 5

Reverse Queue, elements are: 5 4 3 2 1

Add two elements to the said queue:

Queue elements are: 5 4 3 2 1 100 200

Reverse Queue, elements are: 200 100 1 2 3 4 5

Stacks:

A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle. Stack has one end, whereas the Queue has two ends (front and rear). The code written for stacks is to perform the insertion and deletion of elements in Stacks.

Input: 10 20 30 40

We pushed the above values into the stacks.

Output: Pushed: 10

Pushed: 50

Pushed: 30

Pushed: 40

Stack items:

40 30 50 10

Popped: 40

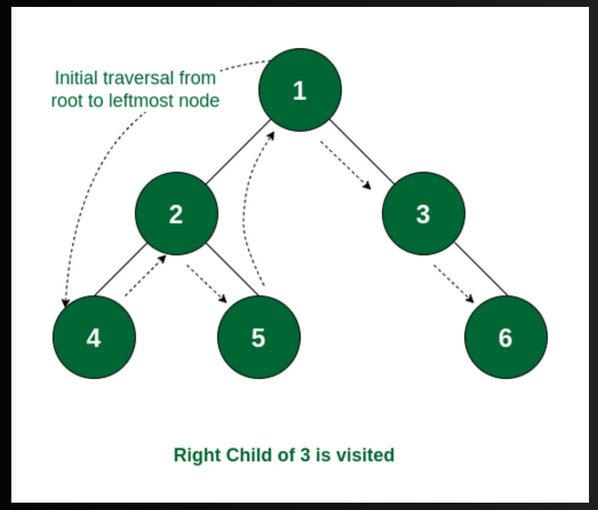
Popped: 30 Stack items: 50 10 Pushed: 50 Pushed: 60 Stack items: 60 50 50 10

Tree traversal:

In order traversal:

This technique follows the 'left root right' policy. It means that first left subtree is visited after that root node is traversed, and finally, the right subtree is traversed. The code written for tree traversal is to perform In-order traversal.

Input:



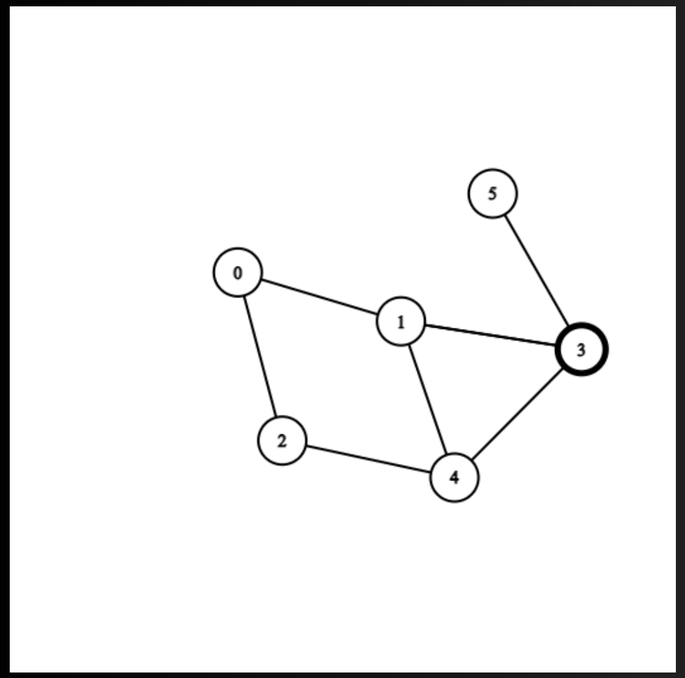
Output:

In order traversal of binary tree is: 4 2 5 1 3 6

Here we traversed the left node (4) first, then the current node, and finally, the right node.

Breadth first search:

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes.
The code written for BFS to perform BFS on a binary tree.
Input:



Output:

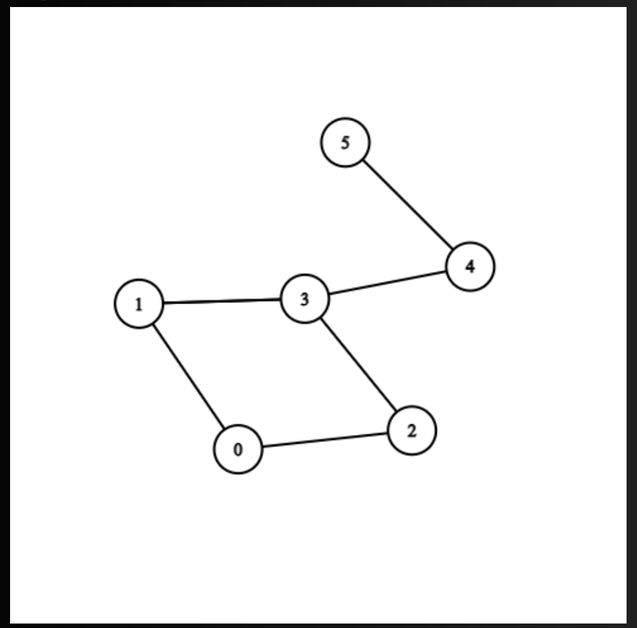
BFS Traversal starting from vertex 0:
Visited vertex: 0
Visited vertex: 1
Visited vertex: 2
Visited vertex: 3
Visited vertex: 4
Visited vertex: 5

Depth first search

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

The code written is to perform DFS on a binary tree.

Input:



Output:

Depth-First Traversal (starting from vertex 0):

Visited vertex: 0

Visited vertex: 2

Visited vertex: 3

Visited vertex: 4

Visited vertex: 5

Visited vertex: 1

Prim's Algorithm

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph.
The code written for Prim’s is to find MST using Prim’s algorithm.

Input:
Adjacency matrix

```
{ 0, 2, 0, 6, 0 },
{ 2, 0, 3, 8, 5 },
{ 0, 3, 0, 0, 7 },
{ 6, 8, 0, 0, 9 },
{ 0, 5, 7, 9, 0 } };
```

Output:

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

Kruskal's algorithm:

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph.
The code written is to find MST using Kruskal’ Algorithm

Input:
createEdge(0,1,4)
createEdge(0,2,3)
createEdge(1,2,1)
createEdge(1,3,2)
createEdge(2,3,4)
createEdge(3,4,2)
createEdge(4,5,6)

Output:
Minimum spanning Tree:
1 – 2 : 1
1 – 3 : 2
3 – 4 : 2
0 – 2 : 3
4 – 5 : 6

Dijkstra’s Algorithm:

It is used to find the shortest path in graphs

The code written is to find the shortest path using Dijkstra’s algorithm.

Input:

```
{ 0, 4, 0, 0, 0, 0, 0, 8, 0 },
 { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
 { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
 { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
 { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
 { 0, 0, 4, 14, 10, 0, 2, 0, 0 } ,
 { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
 { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
 { 0, 0, 2, 0, 0, 0, 6, 7, 0 };
```

Vertex	Distance
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

Bellmanford Algorithm:

Bellman ford algorithm is a single-source shortest path algorithm. This algorithm is used to find the shortest distance from the single vertex to all the other vertices of a weighted graph.

The code written is to find the shortest path from single source vertex to all other vertices using Bellmanford algorithm.

Input:

```
(graph, 0, 1, -1);
(graph, 0, 2, 4);
(graph, 1, 2, 3);
(graph, 1, 3, 2);
(graph, 1, 4, 2);
(graph, 3, 2, 5);
(graph, 3, 1, 1);
(graph, 4, 3, -3);
```

Output:

Shortest Paths from Source

Vertex	Distance from Source
0	0
1	-1
2	2
3	-2
4	1