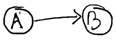
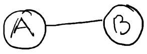
Bfs dfs Y section

06 March 2025 08:34

Graph Data Structure is a collection of **nodes**. Nodes are connected by **edges**. Edges represent connection between nodes.

Directed graph:  You can go from node A to B, but not B to A. Arrow will be present.

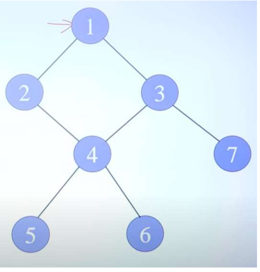
Undirected graph:  You can go from B to A and also from B to A. Arrow is absent.

Graphs Traversal

To traverse a Graph means to visit each and every node of graph only once.

2 techniques: BFS (Breadth first search), DFS (depth first search)

BFS is a graph traversal algorithm that explores all the neighbours of a node before moving on to their neighbours.
DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking.

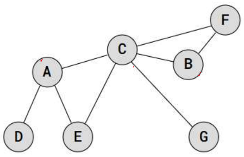


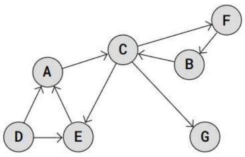
BFS Algorithm

1. **Push the starting node into the queue** and mark it as visited.
2. **While the queue is not empty, repeat:**
   - **Remove** an element (node) from the front of the queue.
   - **Process the node** (if required). Print it
   - **Push all its unvisited neighboring nodes** into the queue and mark them as visited.

Iterative DFS Algorithm (Using a Stack)

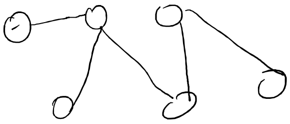
1. **Push start element in stack and print it.**
2. **Repeat till stack is not empty:**
   a. **See the top element in stack.**
   b. **If all its neighbours have been visited, remove the top item from stack.**
   c. **Else push one of its unvisited neighbours and continue the process.**





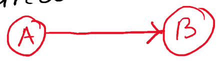
Graph- Non-Linear Data Structure.

- finite number of vertices/nodes.
- Nodes — Connections — Edges.

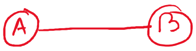


Types-

① Directed



② Undirected.



Representation of Graph-

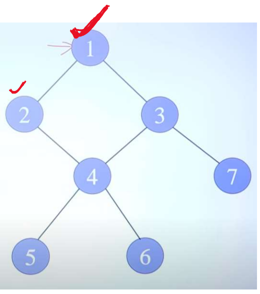
① Adjacency Matrix

② Adjacency List.

n = Number of nodes in your graph.

① Adjacency Matrix

② Adjacency List.



n = Number of nodes in your graph.

n = 7.

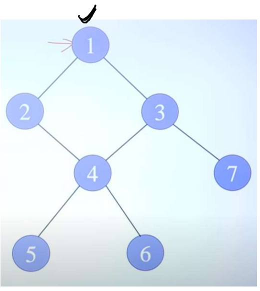
adj-matrix [7][7]. $n^2 = 49$ cells.

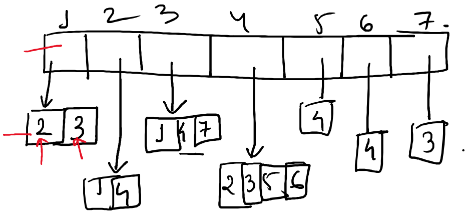
0/1

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

Adjacency List -

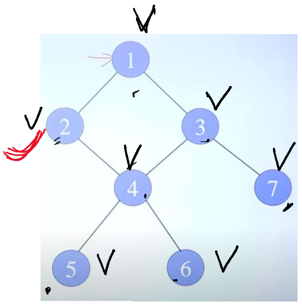
① List of list Arraylist<Arraylist<Integr>> adj list;





Graph Traversal - visit every node in the graph.

① BFS (Breadth First Search) ② DFS (Depth First Search)



Queue Data Structure??? FIFO

Queue ~~2~~ | ~~1~~ | ~~4~~ | ~~3~~ | ~~6~~ | ~~5~~ | ~~7~~ |

Visited array

1	2	3	4	5	6	7
1	1	~~0~~ 1	1	1 ~~0~~	1 ~~0~~	1 ~~0~~

Print: 2 1 4 3 6 5 7

n = 7 = no of nods

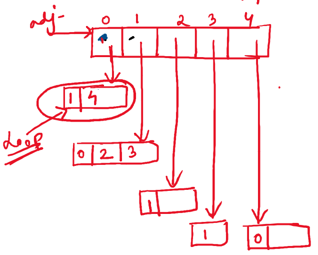
BFS Algorithm

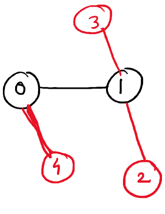
1. **Push the starting node into the queue** and mark it as visited.
2. **While the queue is not empty**, repeat:
   - **Remove** an element (node) from the front of the queue.
   - **Process the node** (if required).
   - **Push all its unvisited neighboring nodes** into the queue and mark them as visited.

```
ArrayList < ArrayList < Integer >> adj = new ArrayList < > ();
```

```
  for (int i = 0; i < 5; i++) {
      adj.add(new ArrayList < > ());
  }
adj.get(0).add(1);
adj.get(1).add(0);
adj.get(0).add(4);
adj.get(4).add(0);
adj.get(1).add(2);
adj.get(2).add(1);
adj.get(1).add(3);
adj.get(3).add(1);
```

→ Initialise with an empty list.

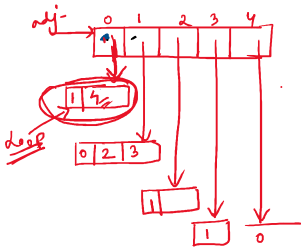




```java
public ArrayList<Integer> bfsOfGraph(int V,
ArrayList<ArrayList<Integer>> adj) {

    ArrayList < Integer > bfs = new ArrayList < > ();
    boolean vis[] = new boolean[V];
    Queue < Integer > q = new LinkedList < > ();

    q.add(e:0);
    vis[0] = true;

    while (!q.isEmpty()) {
        Integer node = q.poll(); //Remove front element
        bfs.add(node);  → Print it
        for (Integer it: adj.get(node)) {  ← loop.
            if (vis[it] == false) {  ← Unvisited.
                vis[it] = true;
                q.add(it);
            }
        }
    }

    return bfs;
```

1. **Push the starting node into the queue** and mark it as visited.
2. **While the queue is not empty, repeat:**
   - **Remove** an element (node) from the front of the queue.
   - **Process the node** (if required). Print it
   - **Push all its unvisited neighboring nodes** into the queue and mark them as visited.

Print: 2 1



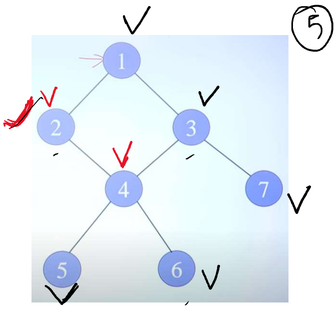
```java
q.add(0);
vis[0] = true;
while (!q.isEmpty()) {
    Integer node = q.poll();
    bfs.add(node);
    for (Integer it: adj.get(node)) {
        if (vis[it] == false) {
            vis[it] = true;
            q.add(it);
        }
    }
}
```

q | ~~0~~ | 1 | 4.

visited | 1~~0~~ | ~~0~~1 | 0 | 0 | ~~0~~1

0 1 2 3 4

print: 0



5 6 3

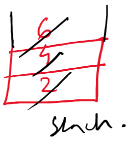
Iterative DFS Algorithm (Using a Stack)

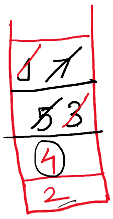
1. **Push start element in stack and print it.** mark it as visited.
2. **Repeat till stack is not empty:**
   - a. **See the top element in stack.**
   - b. **If all its neighbours have been visited, remove the top item from stack.**
   - c. **Else push one of its unvisited neighbours and continue the process.** and mark it as visited and print it.

Stack Data Structure → FILO / LIFO.

Print: 2 4 5 3 7 1 6

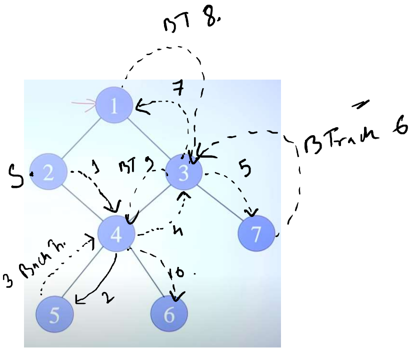
x = ~~2~~ ~~4~~ ~~5~~ ~~4~~ ~~3~~ ~~7~~ 3 ~~1~~ ~~3~~ ~~4~~ ~~6~~ ~~4~~ ~~2~~





visited

~~0~~1	1~~0~~	~~0~~1	1~~0~~	~~0~~1	~~0~~1	~~0~~1
1	2	3	4	5	6	7


BT 8.
7
1
S
2
1
BT 2
3
5
BTrack 6
4
4
7
3 Brch
5
2
10
6