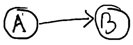
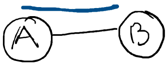
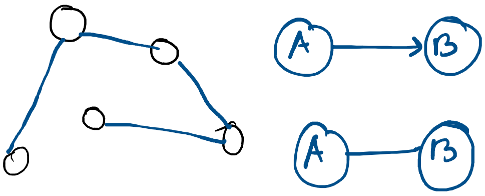
BFS DFS Z LAB

04 March 2025 06:26

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.

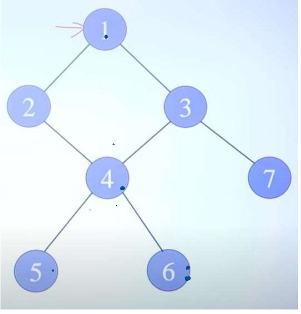


Graphs Traversal

To traverse a Graph means to start in one vertex, and go along the edges to visit other vertices until all vertices, or as many as possible, have been visited.

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.

Representation of graph in code.

① Adjacency Matrix

② Adjacency List.

OR

(1,1)

(1,2) → Is there an edge from node 1 to node 2?

(i,j)

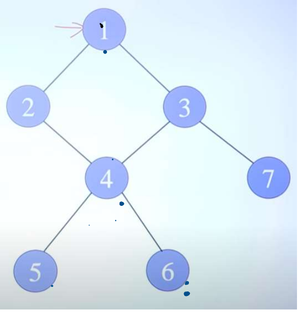
i==j → 0

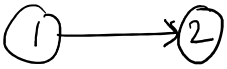
	0	1	2	3	4	5	6	7
0								
1		0	1	1	0	0	0	0
2		1	0	0	1	0	0	0
3		1	0	0	1	0	0	1
4					0			
5						0		
6							0	
7								0

← 0/1

0 → No edge from vertex i to j

1 → Edge from vertex i to j



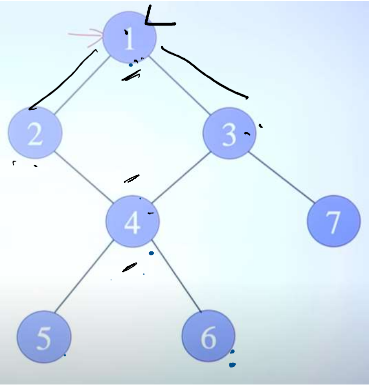


	1	2
1	0	1
2	0	0

(2,1)

2 1

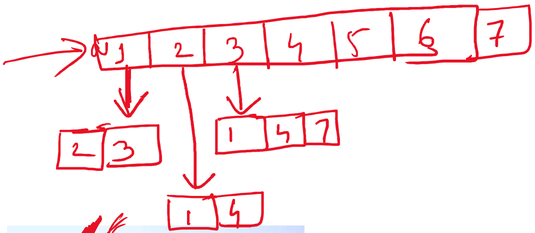
Adjacency List

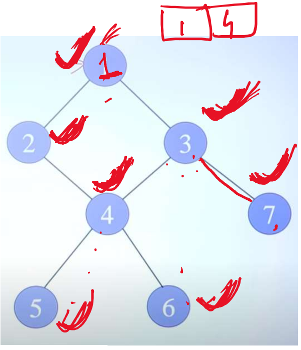


ArrayList<ArrayList<Integer>> adj;

1 →	2, 3
2 →	1, 4
3 →	1, 4, 7
4 →	2, 3, 5, 6
5 →	4
6 →	4
7 →	3.

adjacency list





BFS Algorithm

1. Push the starting node into the queue and mark it as visited.

BFS → Start from any node.

DFS → Start from node 1.

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

result is 1 2 3 4 7 5,

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, res-arr mein store.
   - Push all its unvisited neighboring nodes into the queue and mark them as visited.

result is 1 2 3 4 7 5

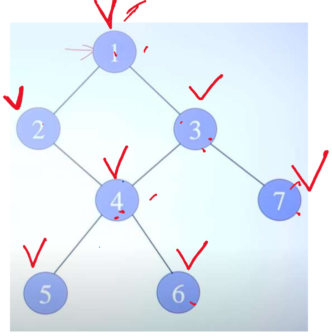
bfs : array.

visited

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

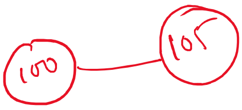
①

BFS traversal from node ②.



queue- ~~2~~ ~~4~~ ~~1~~ ~~3~~ ~~5~~ ~~6~~ ~~7~~

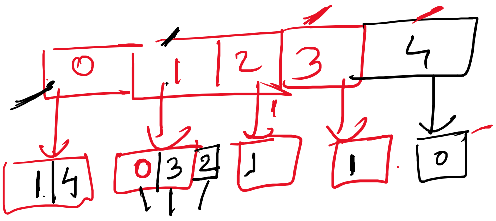
Result- 2 4 1 3 5 6 7.

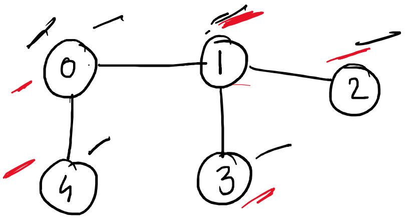


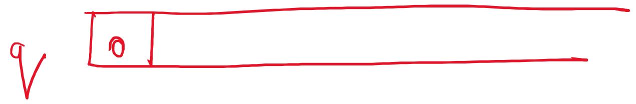
```
ArrayList < ArrayList < Integer >> adj = new ArrayList < > ();
        for (int i = 0; i < 5; i++) {
            adj.add(new ArrayList < > ());
        }
```

```
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);
```






q
0