GRAPH

$G = (E, V)$

Edge

- Directed $A \longrightarrow B$
- Bi-Directed $A \text{———} B$
- Weighted $A \xrightarrow{10} B$

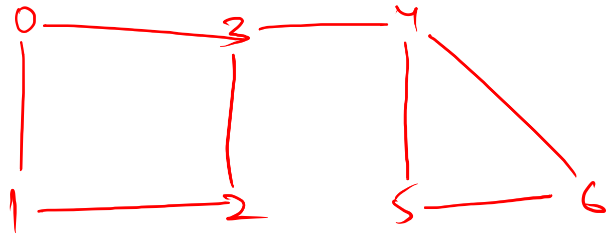
$A \overset{10}{\text{———}} B$

→ Social media

→ Telecom sector

→ Metro / Train paths

Imagine → facts/Numbers



has path

shortest path

All paths

Cycle

Travelling salesman problem

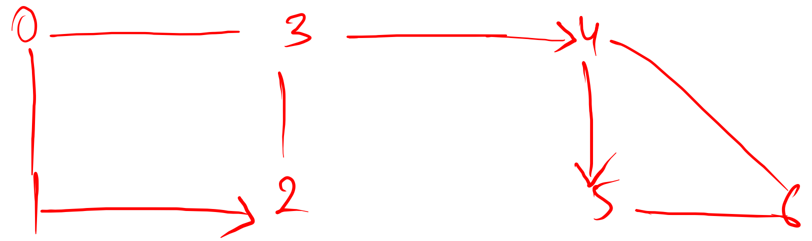
* Every Tree is a graph with an Exception of a special status sort.

* Every graph is not a tree.

graph → cycles allowed

trees → cycles not allowed.

Concept



Adjacency Matrix

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

[3] [5] → 0
↳ No direct Edge

[src][dest]

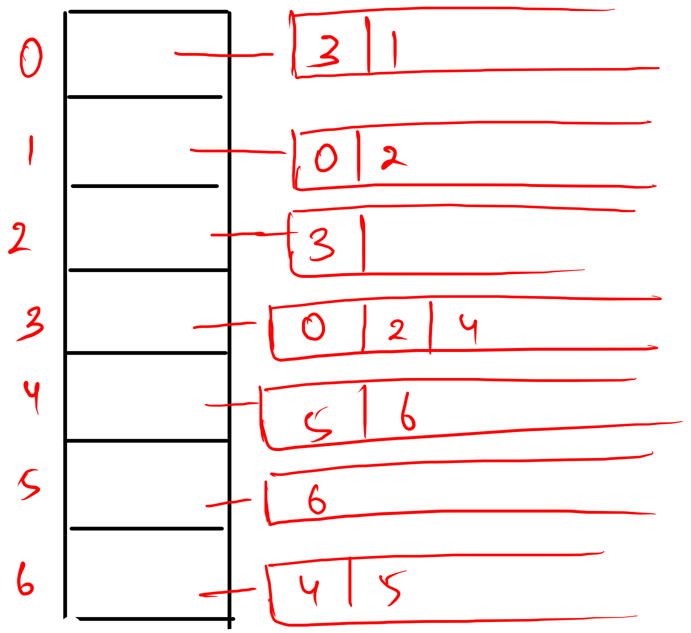
$$\begin{tikzcd} & {[src][dest]} \arrow[dl] \arrow[dr] & \\ 1 & & 0 \end{tikzcd}$$

1 Direct Edge

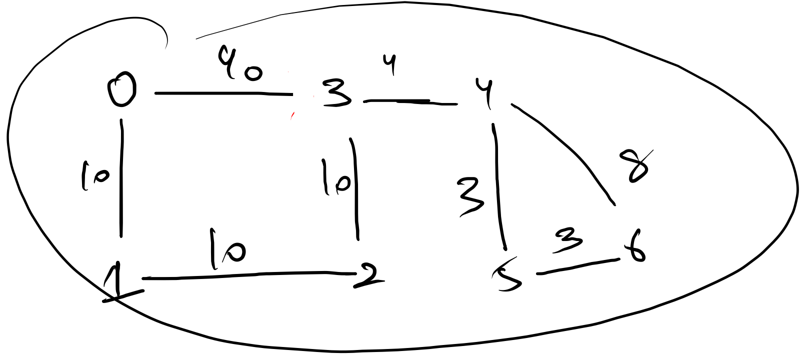
0 No direct Edge

Adjacency List

★ Space conserved which loosing ability of verifying direct Edge

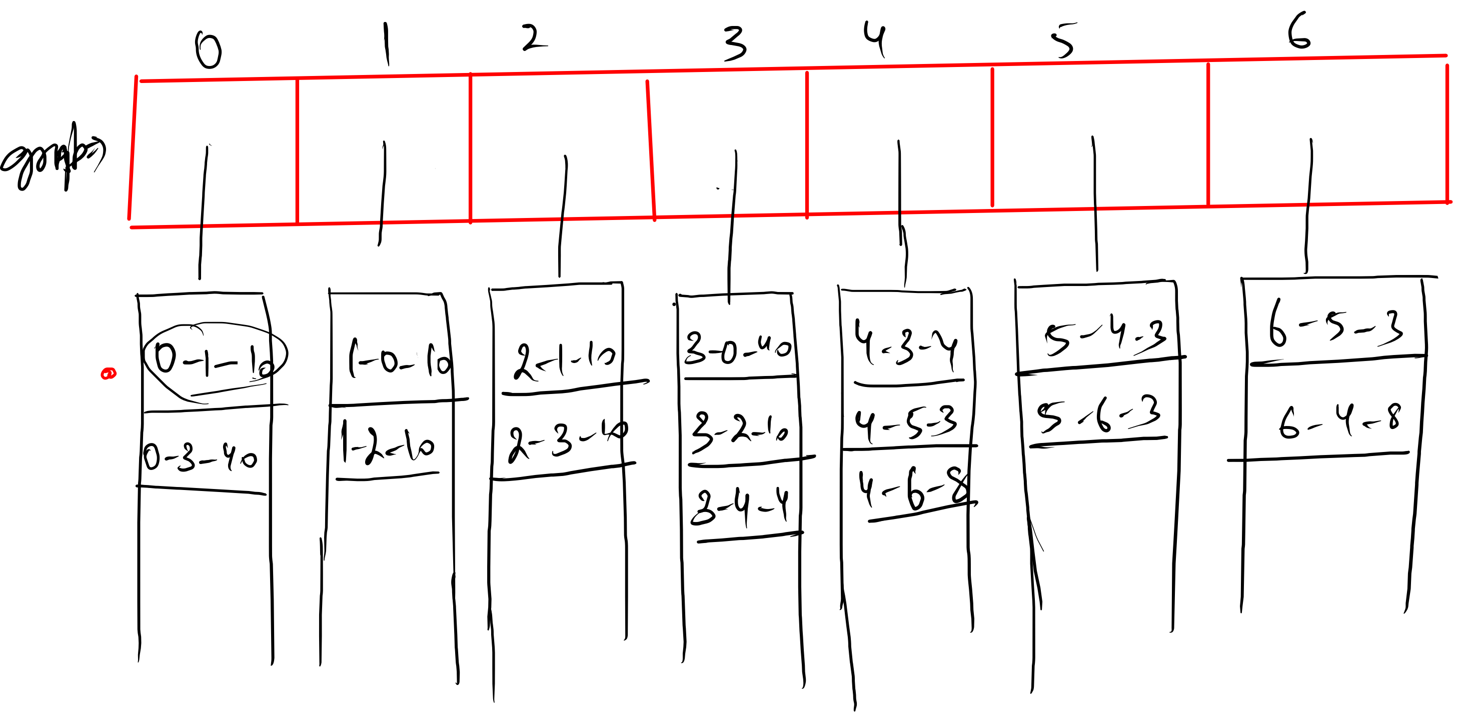


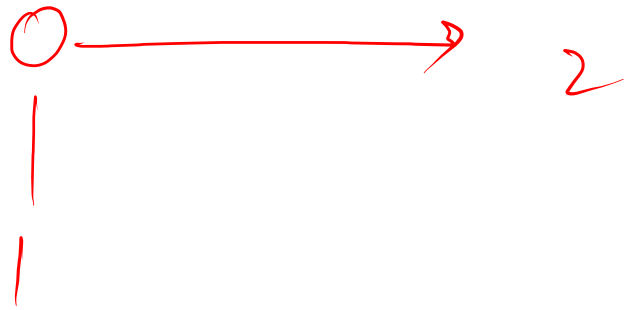
7
8
0 3 40 ✓
0 1 10 ✓
1 2 10 ✓
2 3 10 ✓
3 4 4 ✓
4 5 3 ✓
5 6 3 ✓
4 6 8 ✓



```
0 -> 3 1
1 -> 0 2
2 -> 1 3
3 -> 0 2 4
4 -> 3 5 6
5 -> 4 6
6 -> 5 4
```

```java
public static class Edge{
    int src,dest,wt;
    Edge(int src,int dest,int wt){
        this.src = src;
        this.dest = dest;
        this.wt = wt;
    }
}
Run | Debug
public static void main(String[] args) {
    Scanner  scn = new Scanner(System.in);

    int vtces = scn.nextInt(); // 7
    int edges = scn.nextInt(); // 8

    ArrayList<Edge>[] graph = new ArrayList[vtces]; ✓

    for(int i = 0 ; i < vtces ; i++){
        graph[i] = new ArrayList<Edge>();
    }

    for(int i = 0 ; i < edges ; i++){
        int v1 = scn.nextInt();
        int v2 = scn.nextInt();
        int wt = scn.nextInt();

        graph[v1].add(new Edge(v1,v2,wt));
        graph[v2].add(new Edge(v2,v1,wt));
    }
}
```





3

3

0 1

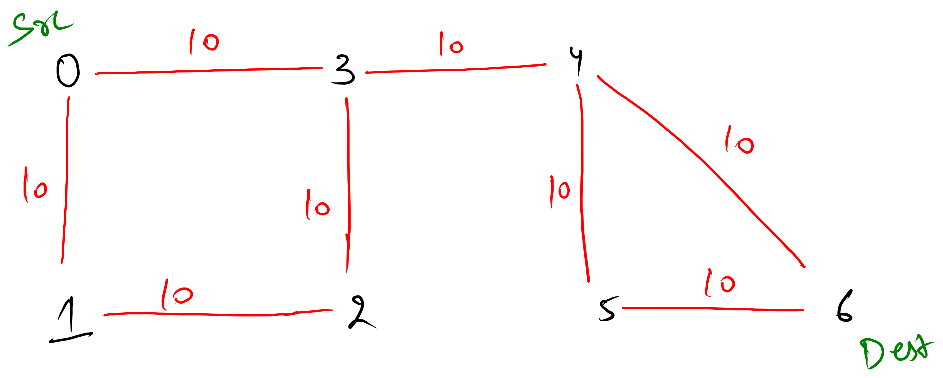
1 0

0 2

✓ ~~2 0~~

7 → vtces
8 → Edges

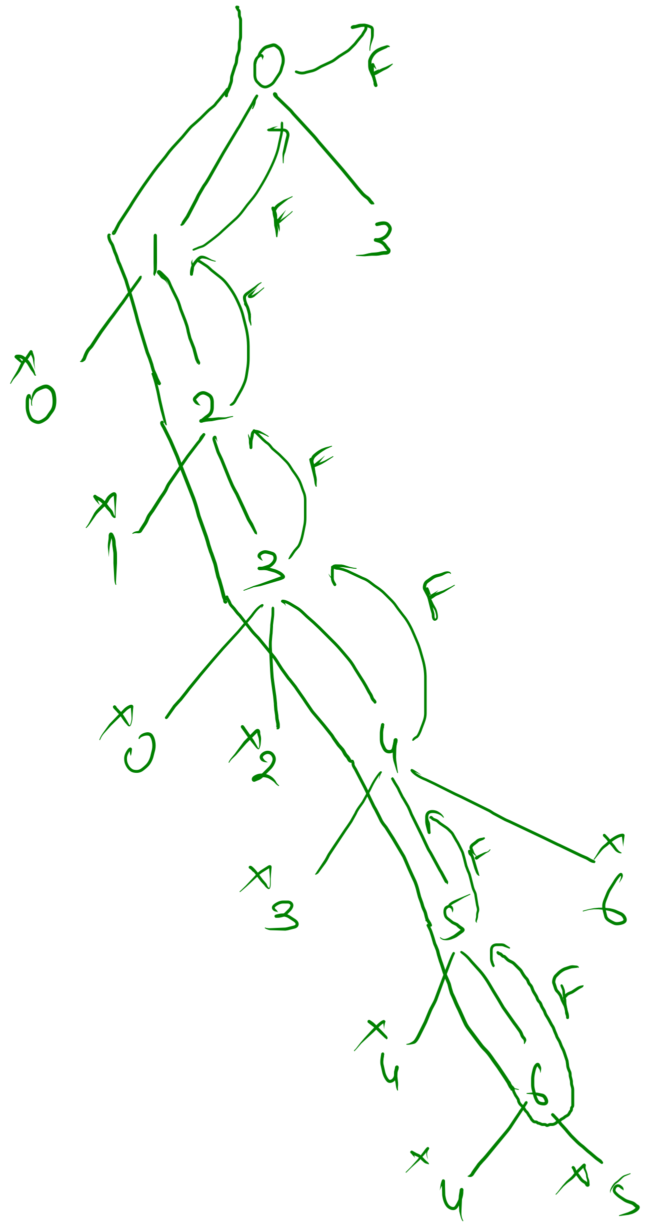
```
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
0
6
```

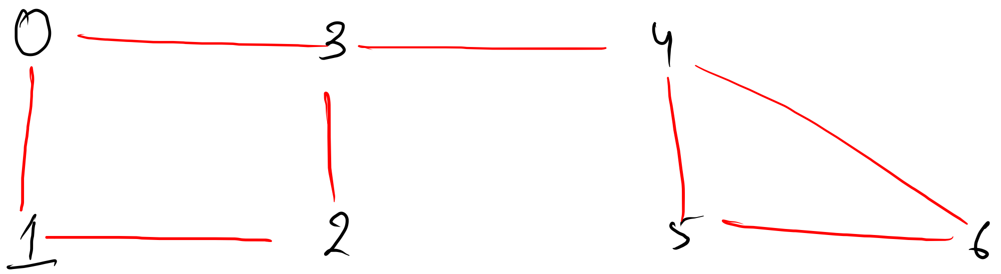


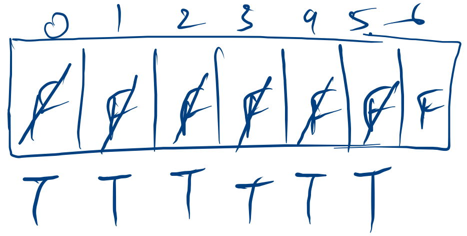
Has Path → True

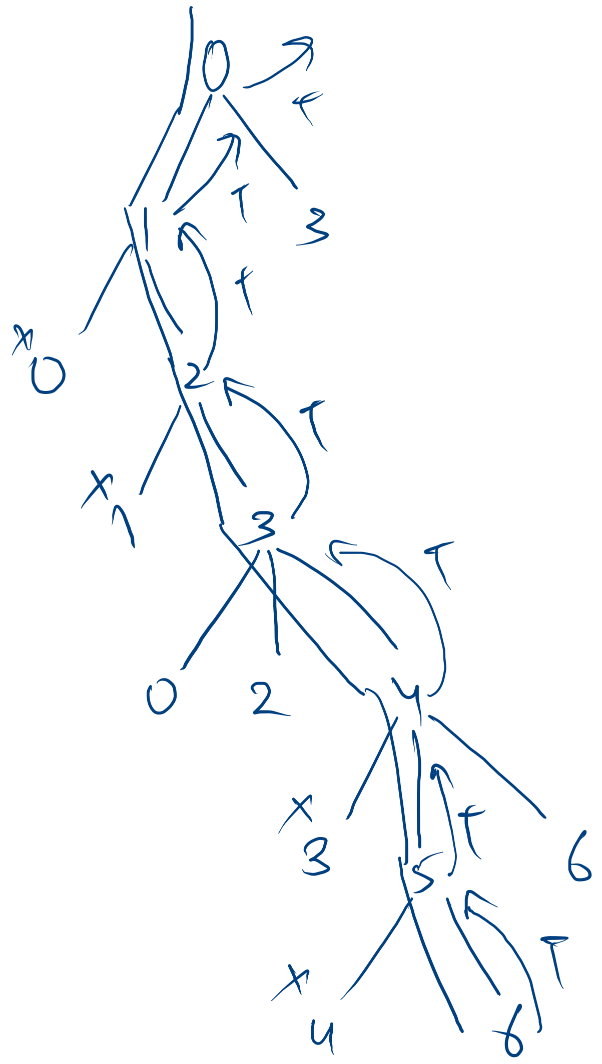
visited =

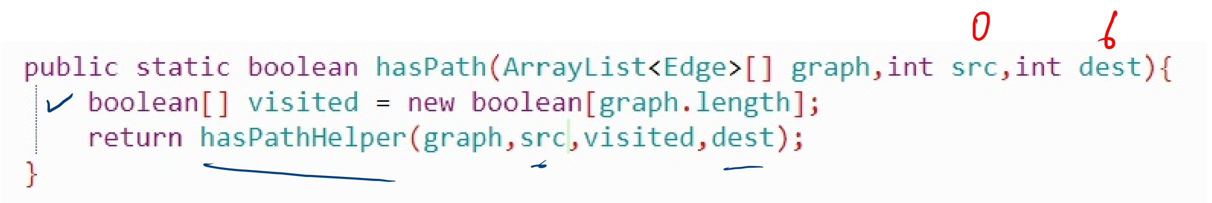
0	1	2	3	4	5	6
~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~
T	T	T	T	T	T	T

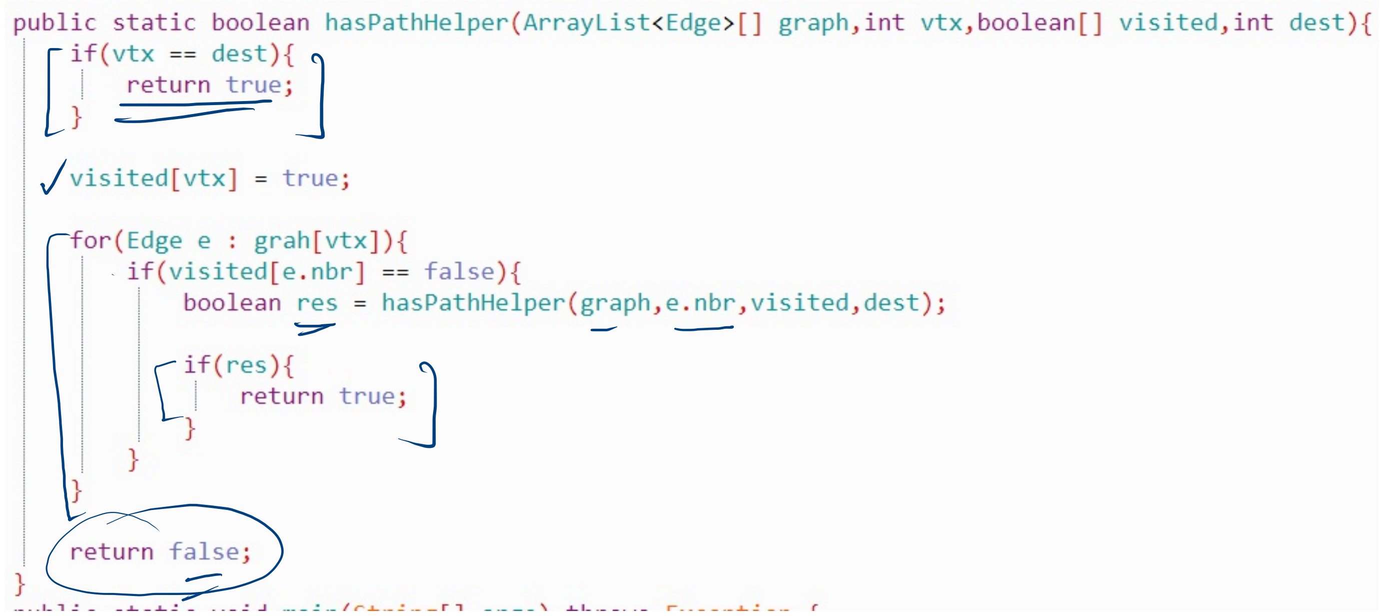




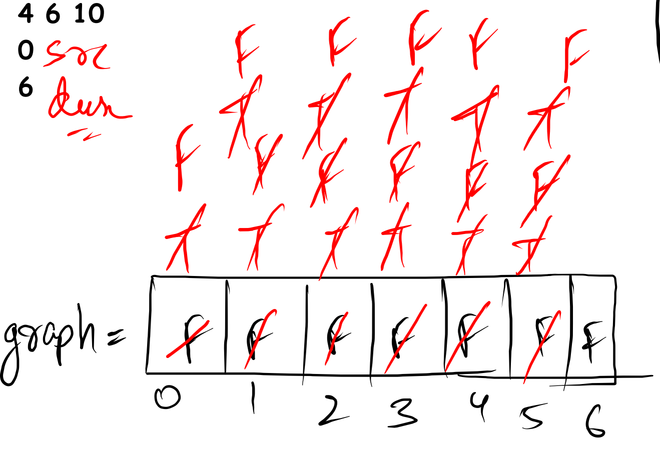


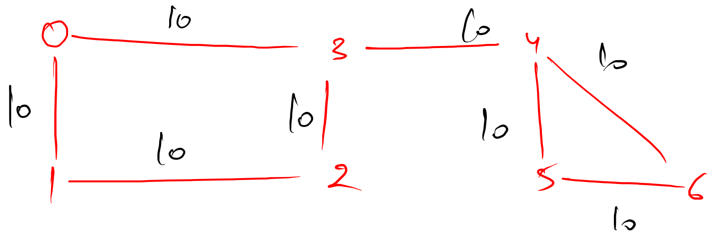



```
                                                                  0         6
public static boolean hasPath(ArrayList<Edge>[] graph,int src,int dest){
    boolean[] visited = new boolean[graph.length];
    return hasPathHelper(graph,src,visited,dest);
}
```

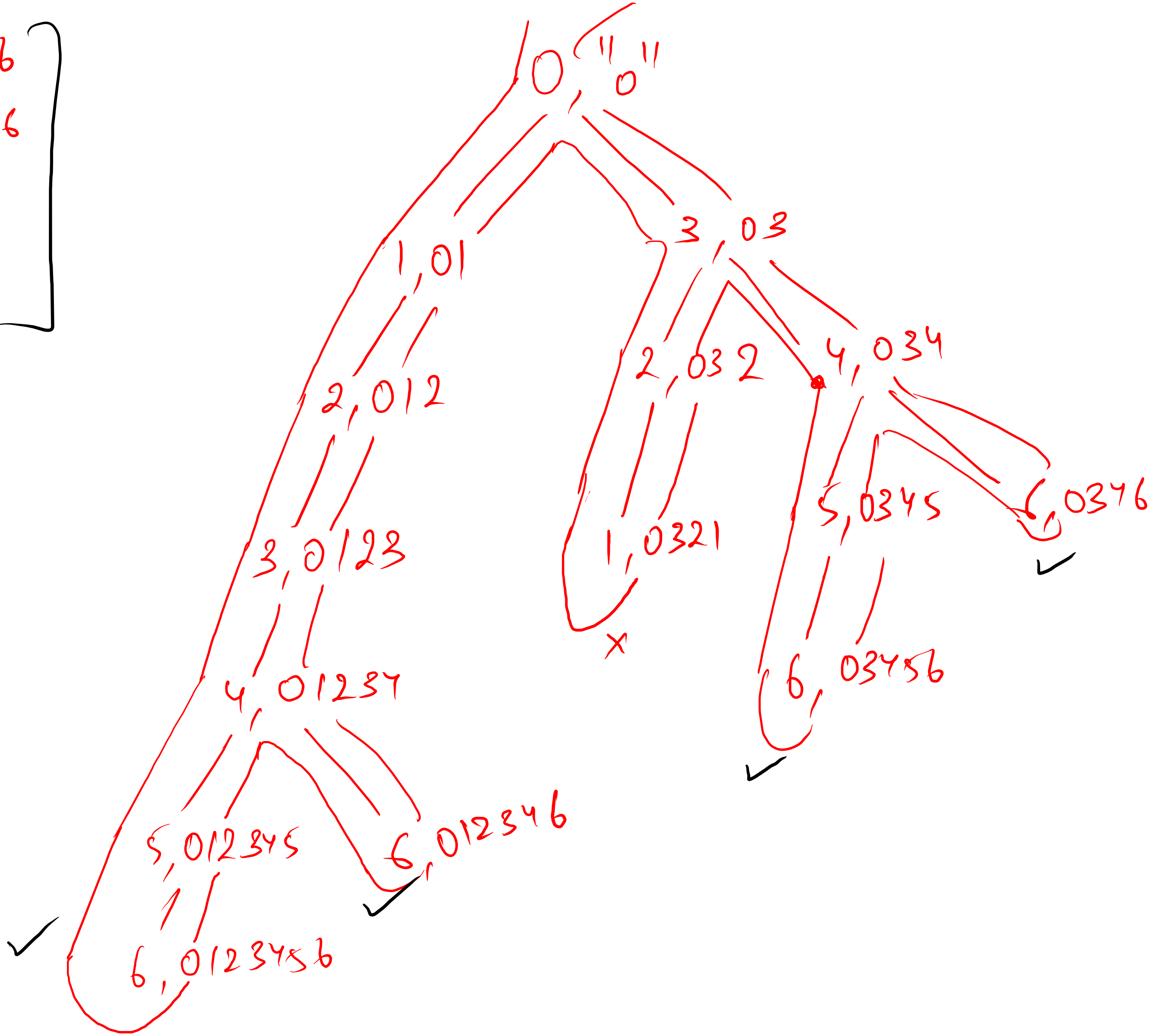

```
public static boolean hasPathHelper(ArrayList<Edge>[] graph,int vtx,boolean[] visited,int dest){
    if(vtx == dest){
        return true;
    }

    visited[vtx] = true;

    for(Edge e : grah[vtx]){
        if(visited[e.nbr] == false){
            boolean res = hasPathHelper(graph,e.nbr,visited,dest);

            if(res){
                return true;
            }
        }
    }

    return false;
}
public static void main(String[] args) throws Exception {
```

7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
0 src
6 dest

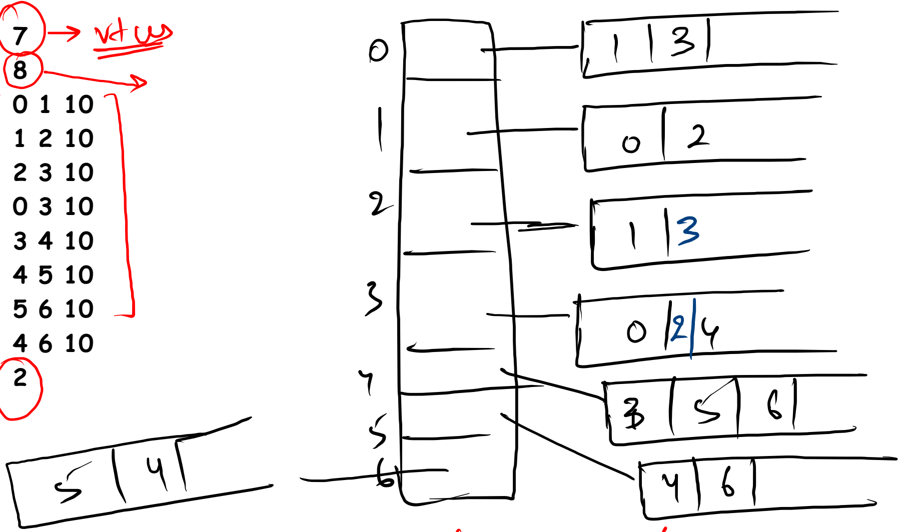


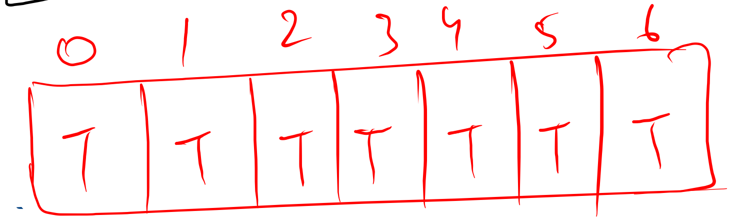


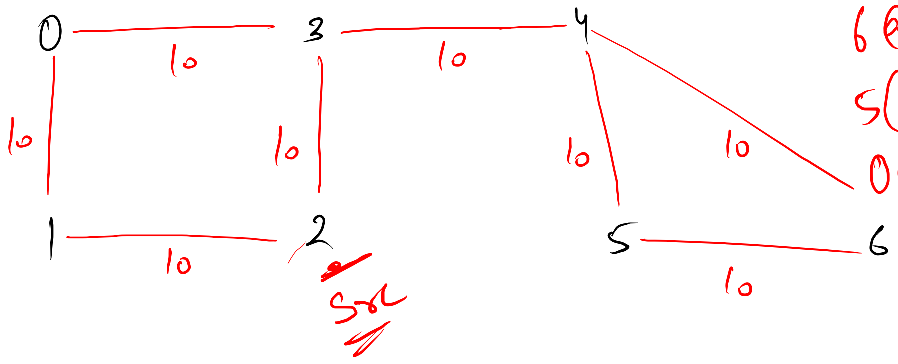
$$\begin{bmatrix} 0123456 \\ 012346 \\ 03456 \\ 0346 \end{bmatrix}$$



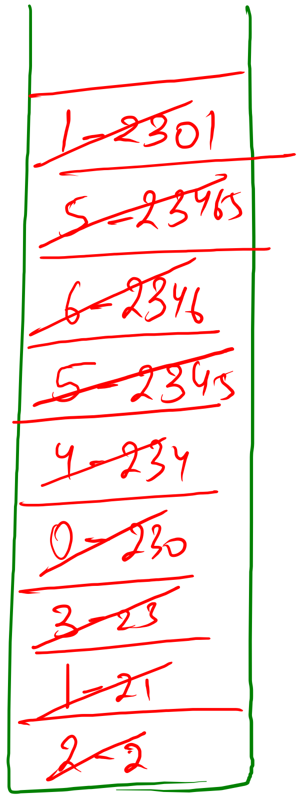
7 → vtces
8 →
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
2



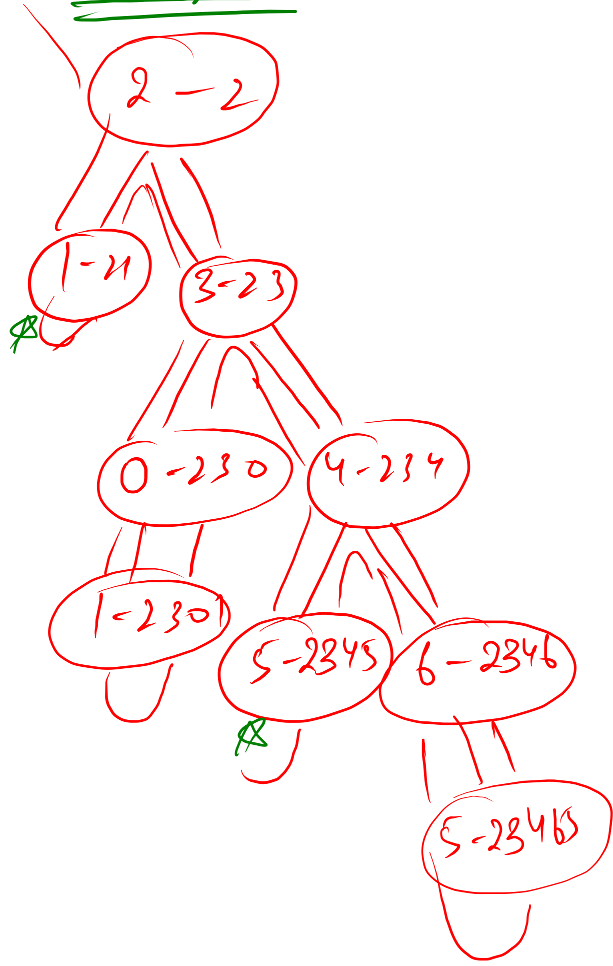


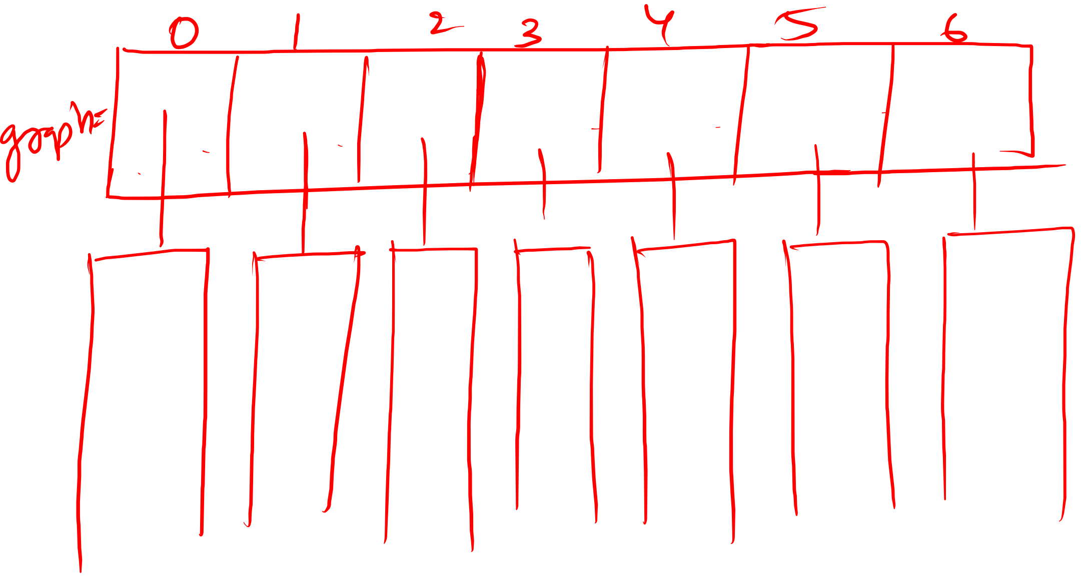


2@2
3@23
4@234
6@2346
5@23465
0@230
1@2301



Imagine





```
7
8
0 3 40
0 1 10
1 2 10
2 3 10
3 4 4
4 5 3
5 6 3
4 6 8
```

```java
Scanner  scn = new Scanner(System.in);

int vtces = scn.nextInt(); // 7
int edges = scn.nextInt(); // 8

* ArrayList<Edge>[] graph = new ArrayList[vtces];

for(int i = 0 ; i < vtces ; i++){
    graph[i] = new ArrayList<Edge>();
}

for(int i = 0 ; i < edges ; i++){
    int v1 = scn.nextInt();
    int v2 = scn.nextInt();
    int wt = scn.nextInt();

    graph[v1].add(new Edge(v1,v2,wt));
    graph[v2].add(new Edge(v2,v1,wt));
}

display(graph);
```

Doubt

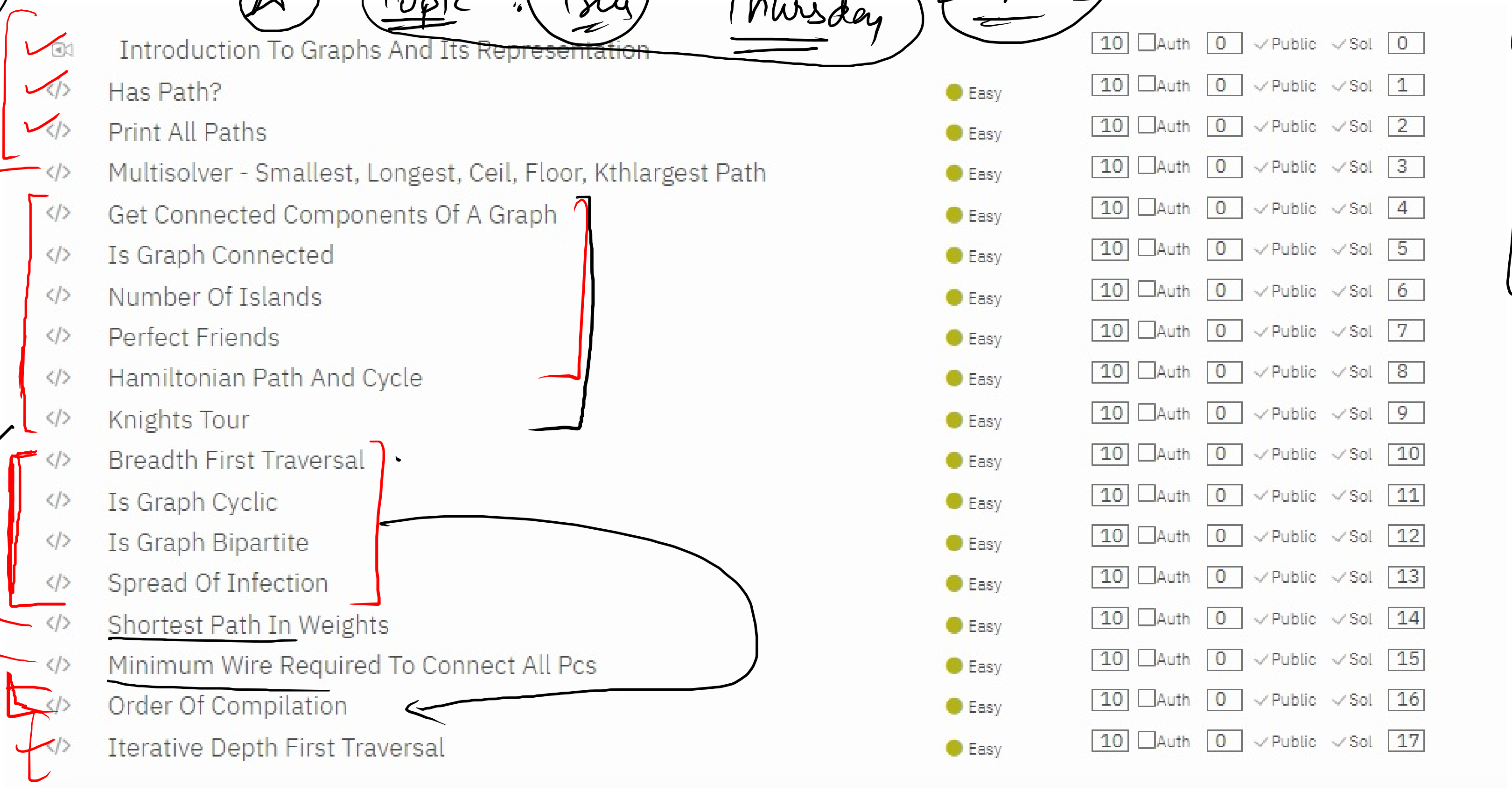
Topic : Try Thursday 3-4 2 hours

Graph

Implementation

General Traversal

HM & Heap



Introduction To Graphs And Its Representation		10	Auth	0	✓ Public	✓ Sol	0
Has Path?	● Easy	10	Auth	0	✓ Public	✓ Sol	1
Print All Paths	● Easy	10	Auth	0	✓ Public	✓ Sol	2
Multisolver - Smallest, Longest, Ceil, Floor, Kthlargest Path	● Easy	10	Auth	0	✓ Public	✓ Sol	3
Get Connected Components Of A Graph	● Easy	10	Auth	0	✓ Public	✓ Sol	4
Is Graph Connected	● Easy	10	Auth	0	✓ Public	✓ Sol	5
Number Of Islands	● Easy	10	Auth	0	✓ Public	✓ Sol	6
Perfect Friends	● Easy	10	Auth	0	✓ Public	✓ Sol	7
Hamiltonian Path And Cycle	● Easy	10	Auth	0	✓ Public	✓ Sol	8
Knights Tour	● Easy	10	Auth	0	✓ Public	✓ Sol	9
Breadth First Traversal	● Easy	10	Auth	0	✓ Public	✓ Sol	10
Is Graph Cyclic	● Easy	10	Auth	0	✓ Public	✓ Sol	11
Is Graph Bipartite	● Easy	10	Auth	0	✓ Public	✓ Sol	12
Spread Of Infection	● Easy	10	Auth	0	✓ Public	✓ Sol	13
Shortest Path In Weights	● Easy	10	Auth	0	✓ Public	✓ Sol	14
Minimum Wire Required To Connect All Pcs	● Easy	10	Auth	0	✓ Public	✓ Sol	15
Order Of Compilation	● Easy	10	Auth	0	✓ Public	✓ Sol	16
Iterative Depth First Traversal	● Easy	10	Auth	0	✓ Public	✓ Sol	17