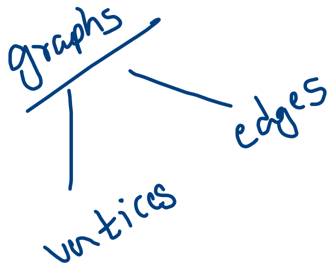
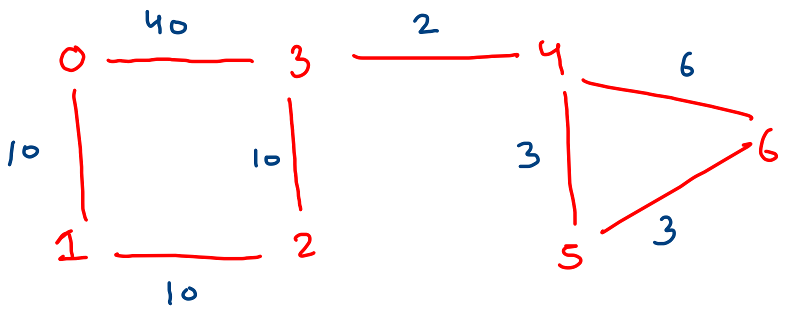


graphs

edges

vertices



undirected
weighted
graph

v = 7

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

e = 8

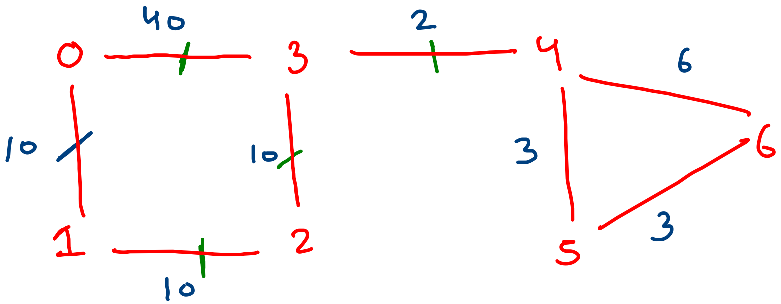
0-1, 0-3 . . . .

src = 0

dest = 6

(i) min stations

(ii) shortest path (wt

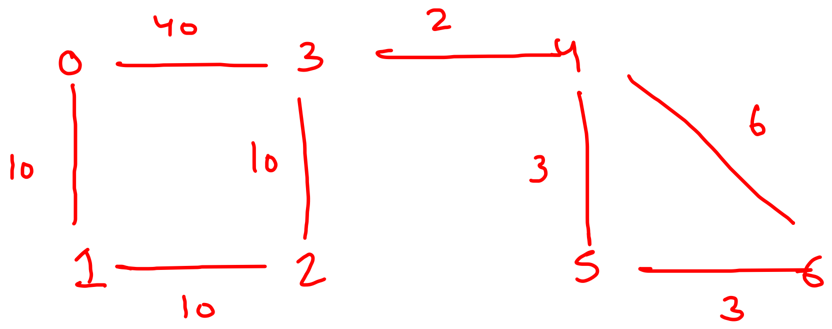


Adjacency matrix

	0	1	2	3	4	5	6
0	∞	10	∞	40	∞	∞	∞
1	10	∞	10	∞	∞	∞	∞
2	∞	10	∞	10	∞	∞	∞
3	40	∞	10	∞	2	∞	∞
4	∞	∞	∞	2	∞	3	6
5	∞	∞	∞	∞	3	∞	3
6	∞	∞	∞	∞	6	3	∞

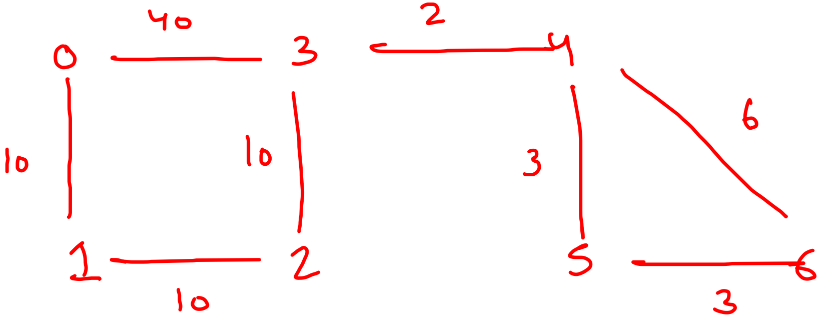
	0	1	2	3	4	5	6
0	∞	10	∞	40	∞	∞	∞
1	10	∞	10	∞	∞	∞	∞
2	∞	10	∞	10	∞	∞	∞
3	40	∞	10	∞	2	∞	∞
4	∞	∞	∞	2	∞	3	6
5	∞	∞	∞	∞	3	∞	3
6	∞	∞	∞	∞	6	3	∞

⇒



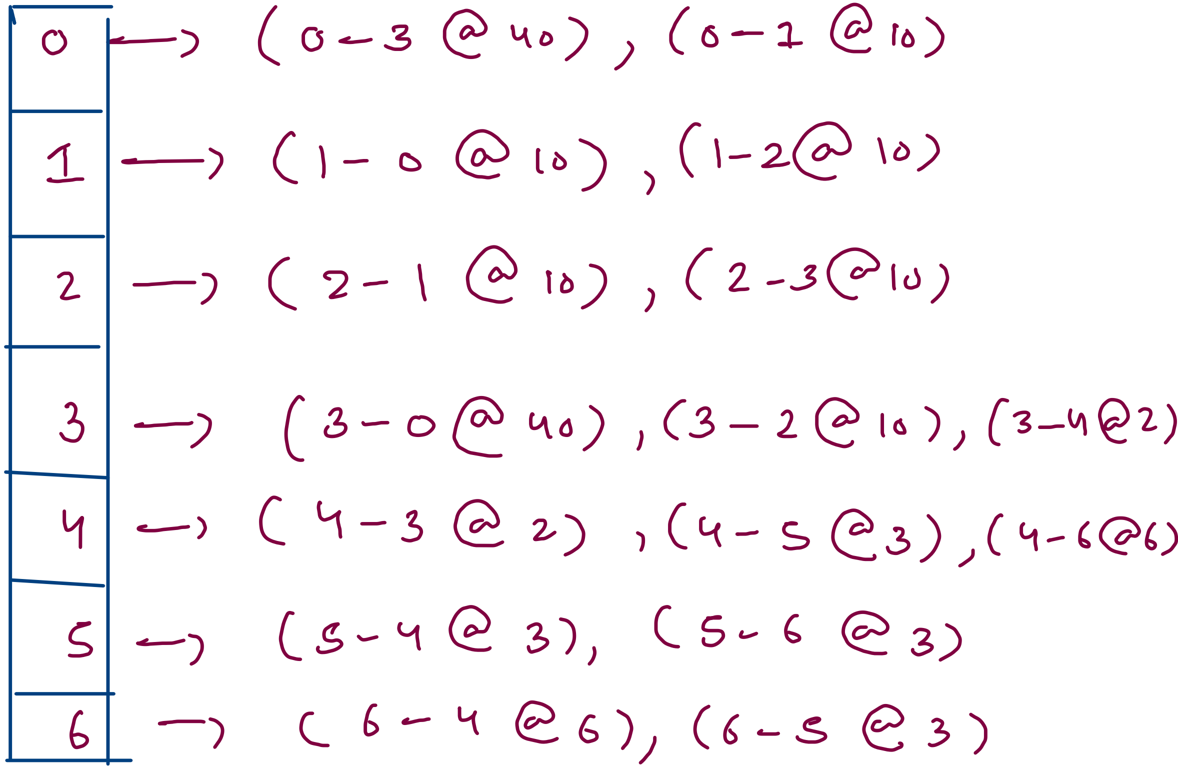
→ waste of space

→ $V \leq 1000$



ArrayList <Edge> [ ] graph;

Adjacency list :



0 → (0-3 @ 40), (0-1 @ 10)

1 → (1-0 @ 10), (1-2 @ 10)

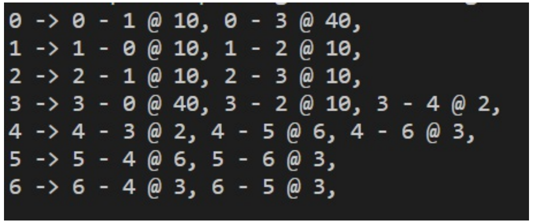
2 → (2-1 @ 10), (2-3 @ 10)

3 → (3-0 @ 40), (3-2 @ 10), (3-4 @ 2)

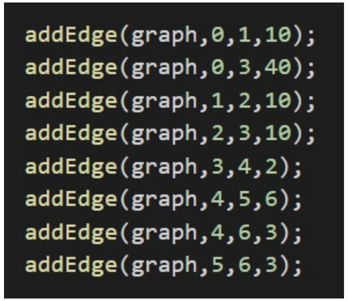
4 → (4-3 @ 2), (4-5 @ 3), (4-6 @ 6)

5 → (5-4 @ 3), (5-6 @ 3)

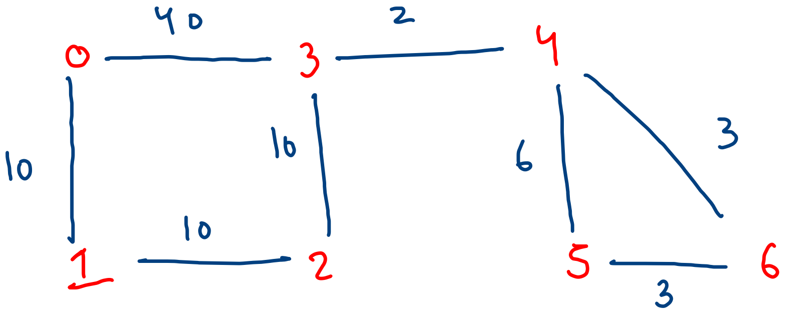
6 → (6-4 @ 6), (6-5 @ 3)

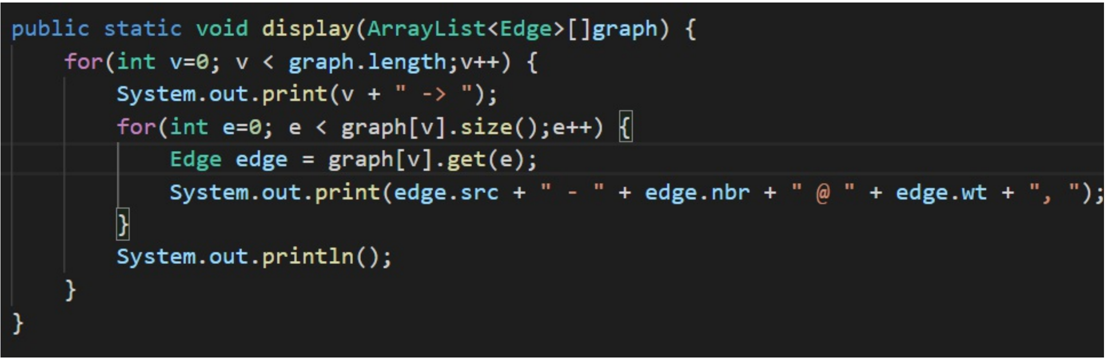


```
0 -> 0 - 1 @ 10, 0 - 3 @ 40,
1 -> 1 - 0 @ 10, 1 - 2 @ 10,
2 -> 2 - 1 @ 10, 2 - 3 @ 10,
3 -> 3 - 0 @ 40, 3 - 2 @ 10, 3 - 4 @ 2,
4 -> 4 - 3 @ 2, 4 - 5 @ 6, 4 - 6 @ 3,
5 -> 5 - 4 @ 6, 5 - 6 @ 3,
6 -> 6 - 4 @ 3, 6 - 5 @ 3,
```

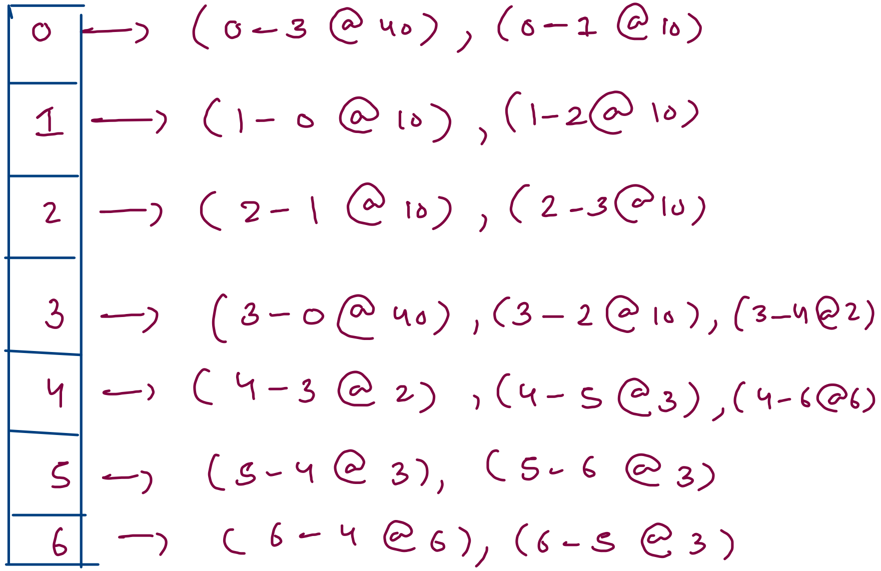


```
addEdge(graph,0,1,10);
addEdge(graph,0,3,40);
addEdge(graph,1,2,10);
addEdge(graph,2,3,10);
addEdge(graph,3,4,2);
addEdge(graph,4,5,6);
addEdge(graph,4,6,3);
addEdge(graph,5,6,3);
```





```java
public static void display(ArrayList<Edge>[]graph) {
    for(int v=0; v < graph.length;v++) {
        System.out.print(v + " -> ");
        for(int e=0; e < graph[v].size();e++) {
            Edge edge = graph[v].get(e);
            System.out.print(edge.src + " - " + edge.nbr + " @ " + edge.wt + ", ");
        }
        System.out.println();
    }
}
```

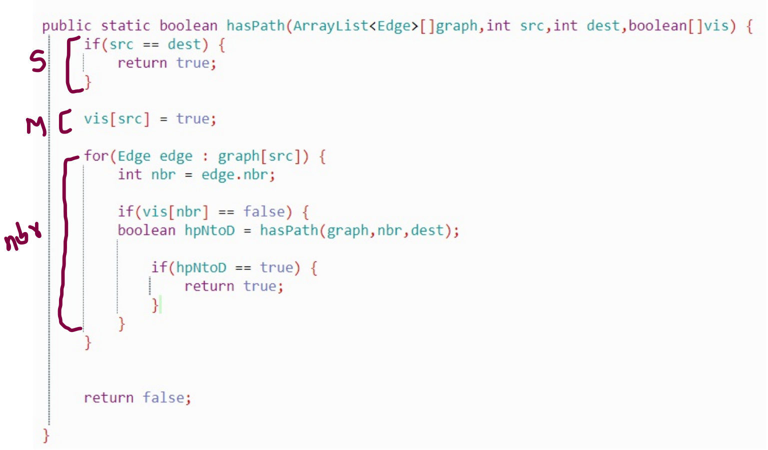


0 → (0 - 3 @ 40), (0 - 1 @ 10)

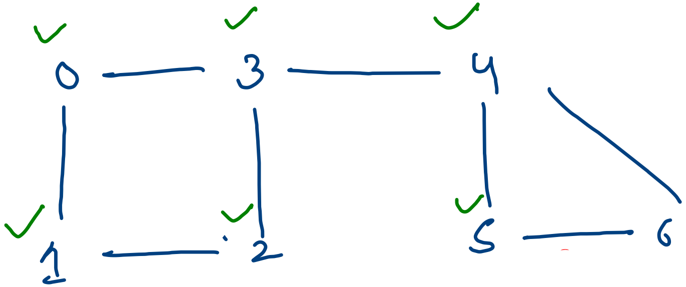
1 → (1 - 0 @ 10), (1 - 2 @ 10)

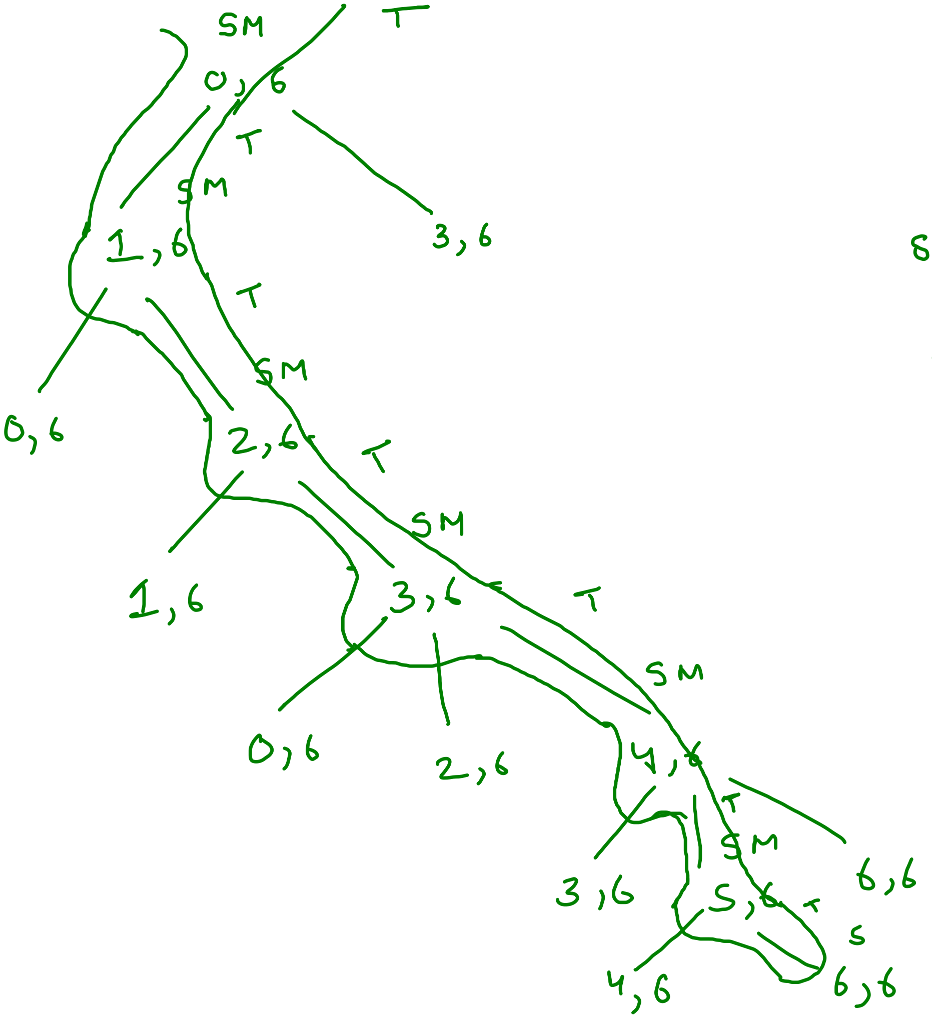
2 →

haspath



```
public static boolean hasPath(ArrayList<Edge>[]graph,int src,int dest,boolean[]vis) {
    if(src == dest) {
        return true;
    }

    vis[src] = true;

    for(Edge edge : graph[src]) {
        int nbr = edge.nbr;

        if(vis[nbr] == false) {
        boolean hpNtoD = hasPath(graph,nbr,dest);

            if(hpNtoD == true) {
                return true;
            }
        }
    }


    return false;
}
```





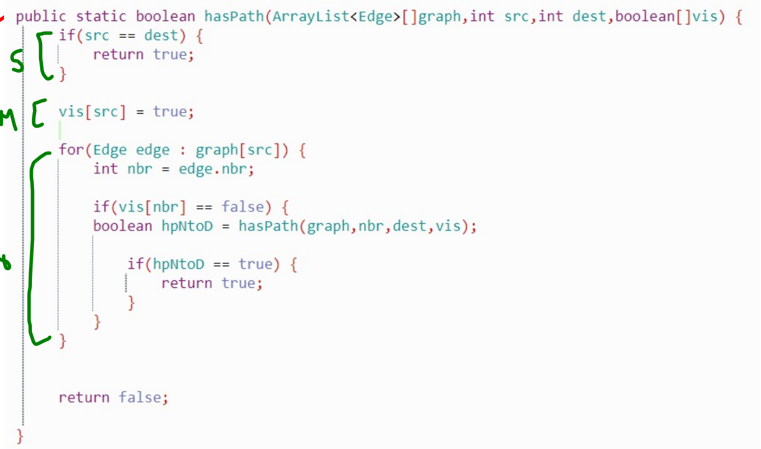
src = 0

dest = 6

haspath

src = 4

dest = 1

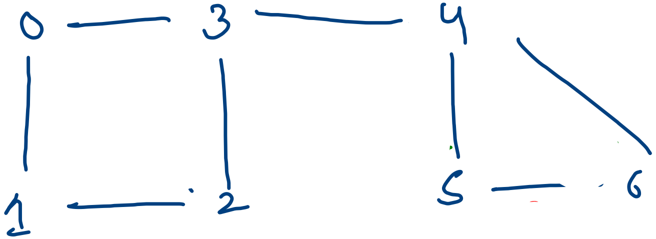


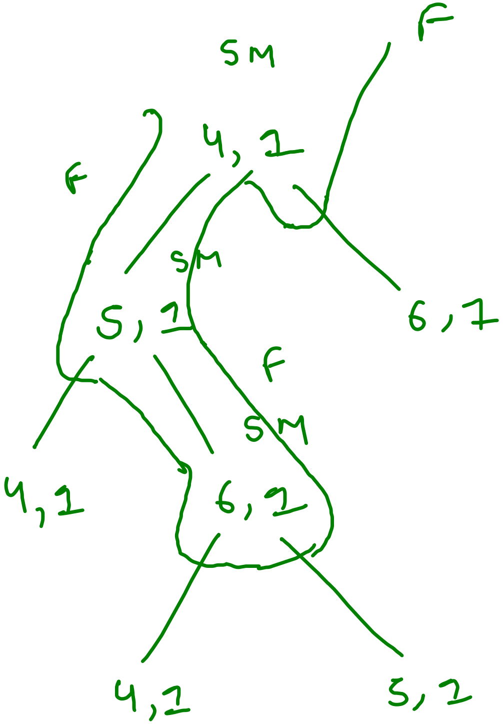
```
public static boolean hasPath(ArrayList<Edge>[]graph,int src,int dest,boolean[]vis) {
    if(src == dest) {
        return true;
    }

    vis[src] = true;

    for(Edge edge : graph[src]) {
        int nbr = edge.nbr;

        if(vis[nbr] == false) {
        boolean hpNtoD = hasPath(graph,nbr,dest,vis);

            if(hpNtoD == true) {
                return true;
            }
        }
    }


    return false;
}
```

S

M

nbr

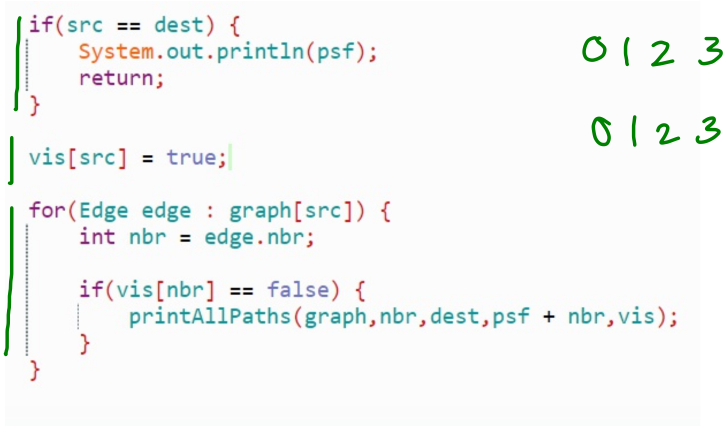




S

M

nbr



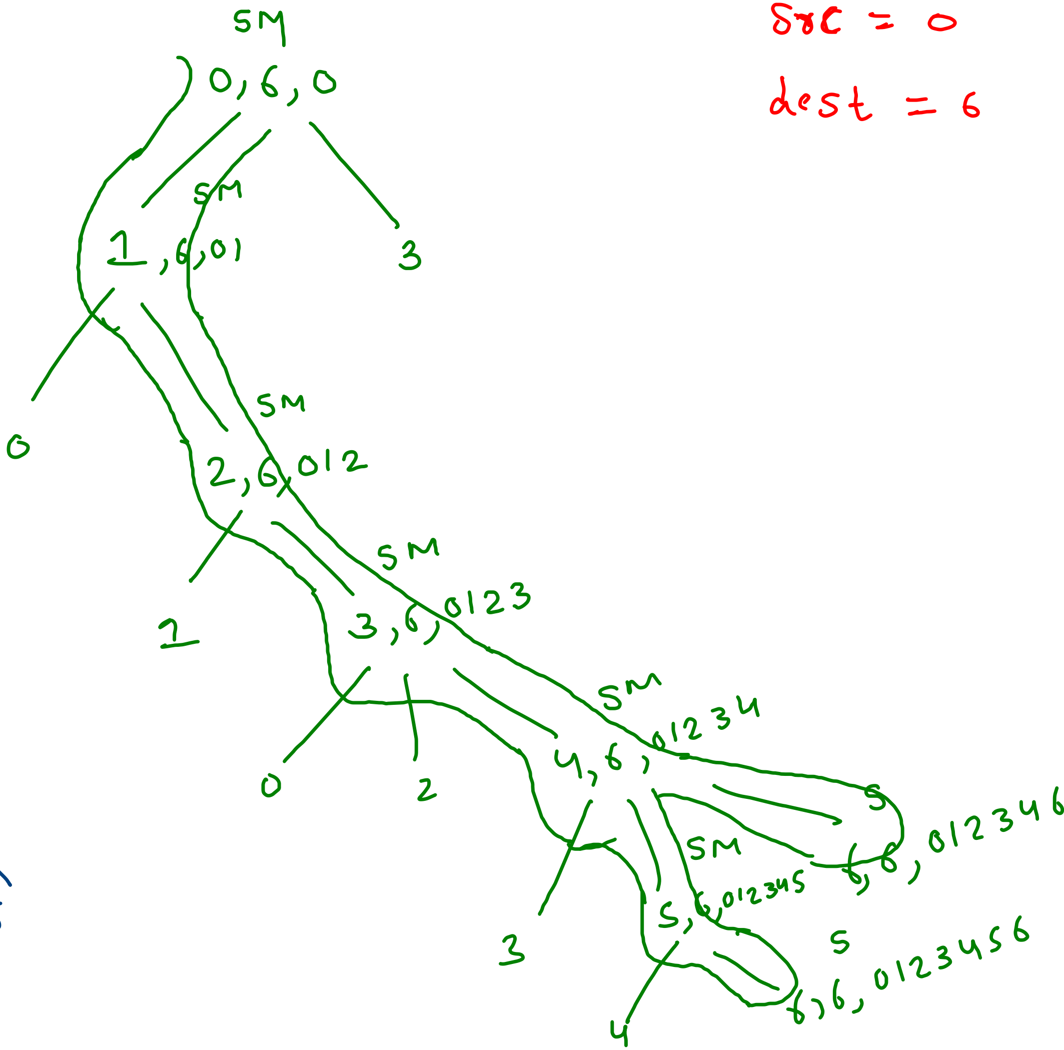
```
if(src == dest) {
    System.out.println(psf);
    return;
}

vis[src] = true;

for(Edge edge : graph[src]) {
    int nbr = edge.nbr;

    if(vis[nbr] == false) {
        printAllPaths(graph,nbr,dest,psf + nbr,vis);
    }
}
```

0 1 2 3 4 5 6

0 1 2 3 4 6

src = 0

dest = 6

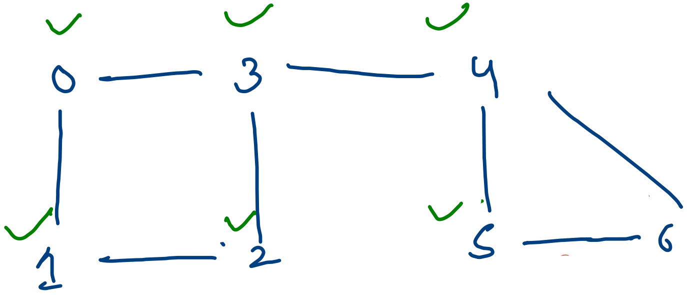


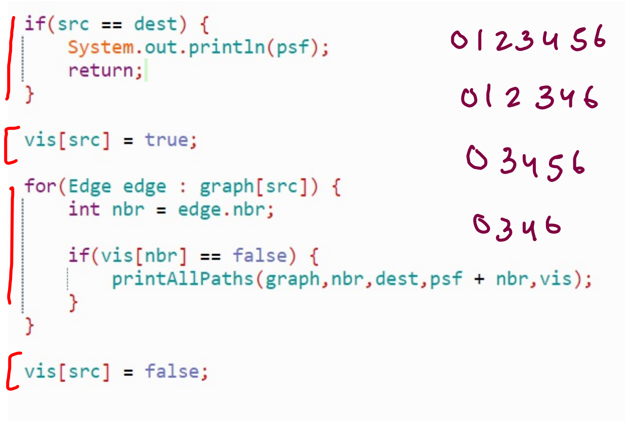
0 1 2 3 4 5 6

0 1 2 3 4 6

0 3 4 5 6

0 3 4 6





S

```java
if(src == dest) {
    System.out.println(psf);
    return;
}
```

M

```java
vis[src] = true;
```

nbr

```java
for(Edge edge : graph[src]) {
    int nbr = edge.nbr;

    if(vis[nbr] == false) {
        printAllPaths(graph,nbr,dest,psf + nbr,vis);
    }
}
```

U

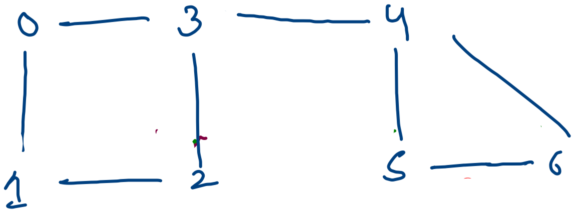
```java
vis[src] = false;
```

0123456

012346

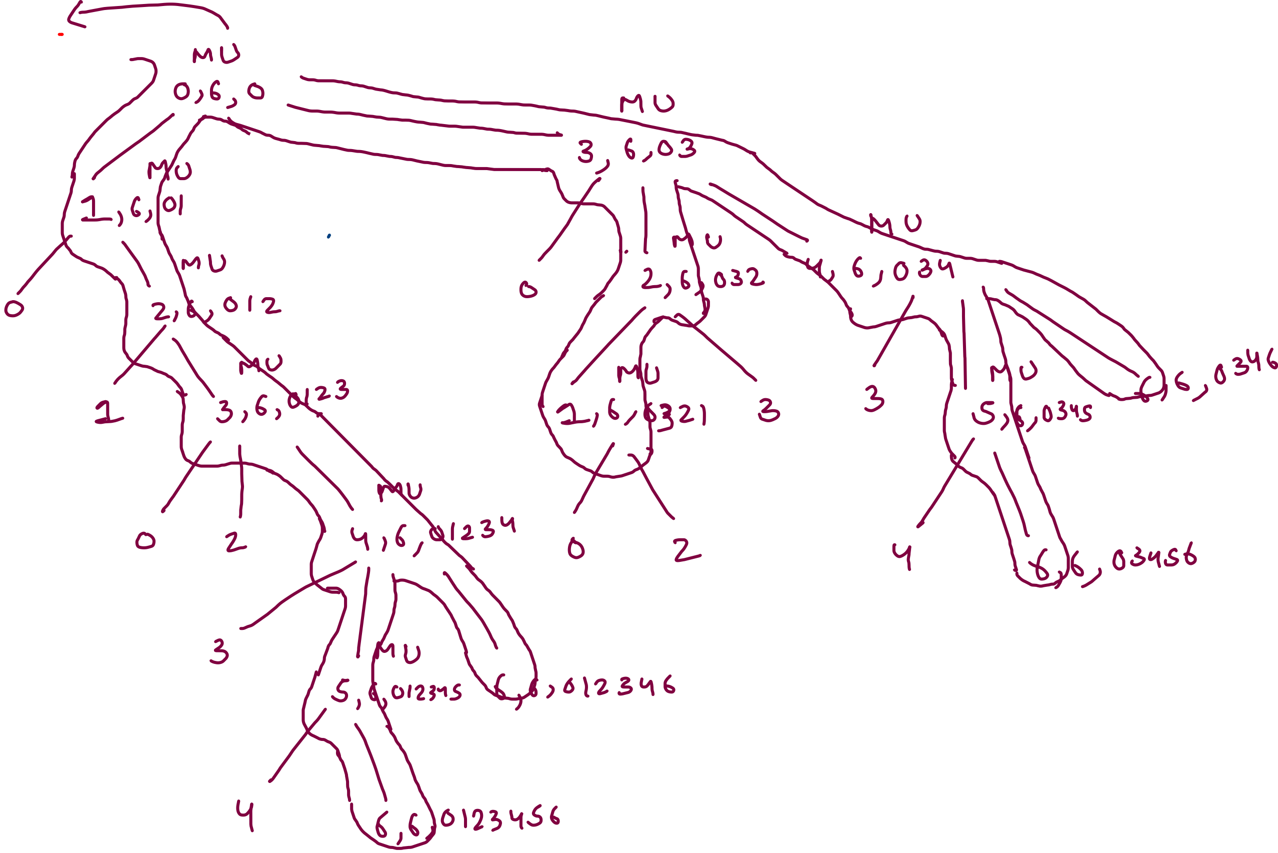
03456

0346

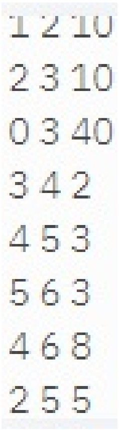


src = 0

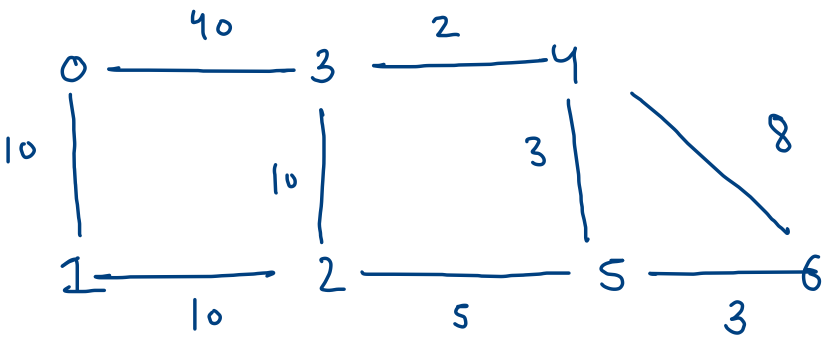
dest = 6



0 1 10

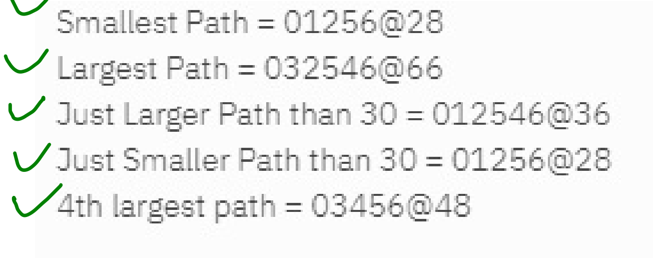


```
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
2 5 5
```



src = 0

dest = 6



✓ Smallest Path = 01256@28

✓ Largest Path = 032546@66

✓ Just Larger Path than 30 = 012546@36

✓ Just Smaller Path than 30 = 01256@28

✓ 4th largest path = 03456@48

0 1 2 3 4 5 6 @ 38

0 1 2 3 4 6 @ 40

0 1 2 5 6 @ 28

0 1 2 5 4 6 @ 36

0 3 4 6 @ 50

0 3 4 5 6 @ 48

0 3 2 5 4 6 @ 66

0 3 2 5 6 @ 58

0 1 2 3 4 5 6 @ 38

0 1 2 3 4 6 @ 40

0 1 2 5 6 @ 28

0 1 2 5 4 6 @ 36

0 3 4 6 @ 50

0 3 4 5 6 @ 48

0 3 2 5 4 6 @ 66

0 3 2 5 6 @ 58

$\infty \xrightarrow{F} 66$ largest

$66 \xrightarrow{F} 58$ (2nd L)

$58 \xrightarrow{F} 50$ (3rd L)

$50 \xrightarrow{F} 48$ (4th L)