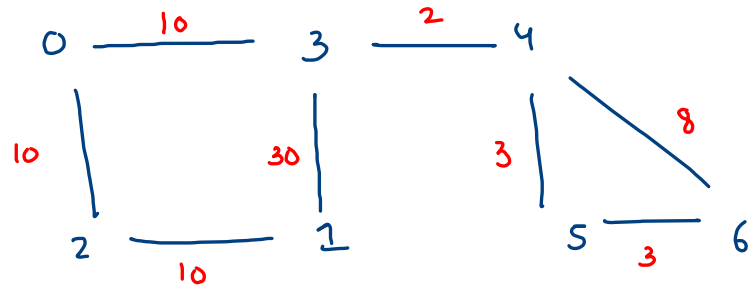
Graphs

(i) vertices

(ii) edges

u -> v

u -



V = 7

e = 8

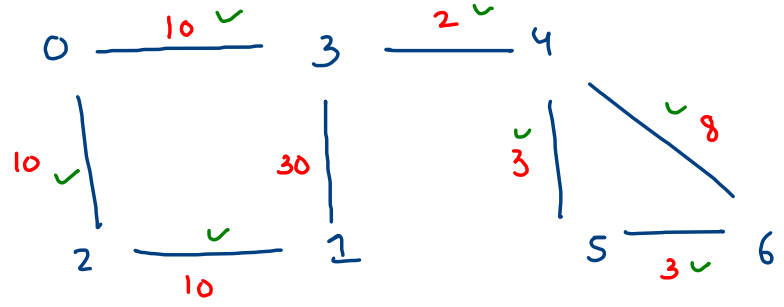
-> undirectional (edges don't have direction)

-> non - weighted

(i) paths

(ii) connected components

```
Edge {
    int src;
    int nbr;
    int wt;
}
```



Adjacency list

0 → 0-2 @10, 0-3 @10

1 → 1-2 @10, 1-3 @30

2 → 2-0 @10, 2-1 @10

3 → 3-0 @10, 3-1 @30, 3-4 @2

4 → 4-3 @2, 4-5 @3, 4-6 @8

5 → 5-4 @3, 5-6 @3

6 → 6-5 @3, 6-4 @8

edges

0	2	10
0	3	10
1	2	10
1	3	30
3	4	2
4	5	3
5	6	3
4	6	8

```
Edge {
    int src;
    int nbr;
    int wt;
}
```

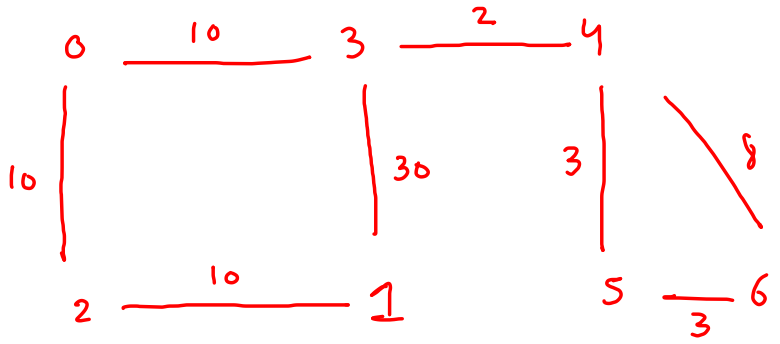
e

e1 -> (u v wt)

e2 -> (v u wt)

Adjacency list

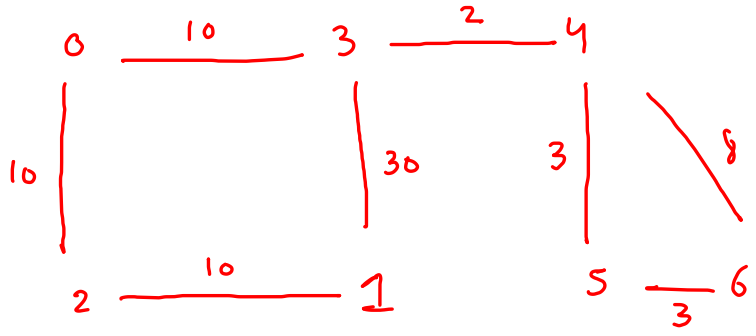
0	→	0-2 @10, 0-3 @10
1	→	1-2 @10, 1-3 @30
2	→	2-0 @10, 2-1 @10
3	→	3-0 @10, 3-1 @30, 3-4 @2
4	→	4-3 @2, 4-5 @3, 4-6 @8
5	→	5-4 @3, 5-6 @3
6	→	6-5 @3, 6-4 @8



v = 7

e = 8

ArrayList < Edges > [ ] graph ;



Adjacency matrix

V * V

(u v wt)

graph [u][v] = wt

graph [v][u] = wt

edges

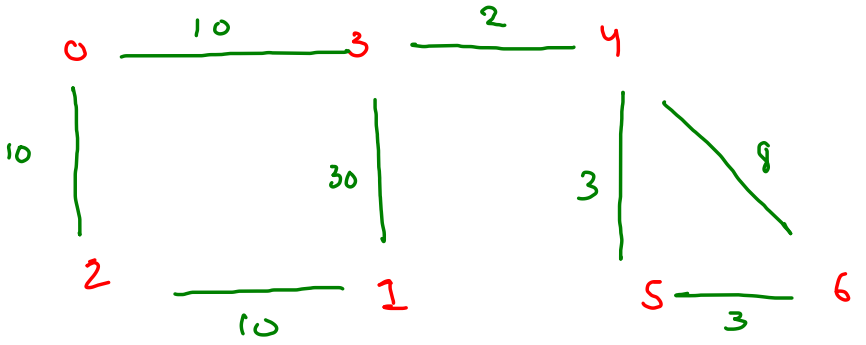
- 0 2 10
- 0 3 10
- 1 2 10
- 1 3 30
- 3 4 2
- 4 5 3
- 5 6 3
- 4 6 8

	0	1	2	3	4	5	6
0	∞	∞	10	10	∞	∞	∞
1	∞	∞	10	30	∞	∞	∞
2	10	10	∞	∞	∞	∞	∞
3	10	30	∞	∞	2	∞	∞
4	∞	∞	∞	2	∞	3	8
5	∞	∞	∞	∞	3	∞	3
6	∞	∞	∞	∞	8	3	∞

V > 1000

(matrix X)

	0	1	2	3	4	5	6
0	∞	∞	10	10	∞	∞	∞
1	∞	∞	10	30	∞	∞	∞
2	10	10	∞	∞	∞	∞	∞
3	10	30	∞	∞	2	∞	∞
4	∞	∞	∞	2	∞	3	8
5	∞	∞	∞	∞	3	∞	3
6	∞	∞	∞	∞	8	3	∞



```java
public static void main(String[]args) {
    Scanner scn = new Scanner(System.in);
    int vtx = scn.nextInt(); //total vertices
    int e = scn.nextInt(); //total edges

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

    while(e-- > 0) {
        int u = scn.nextInt();
        int v = scn.nextInt();
        int wt = scn.nextInt();

        addEdge(graph,u,v,wt);
    }

    display(graph);

}
```

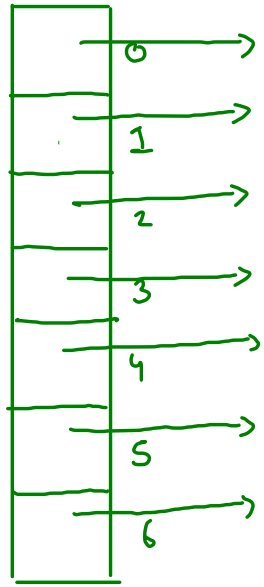
```
7 8
0 2 10
0 3 10
1 2 10
1 3 30
3 4 2
4 5 3
5 6 3
4 6 8
```

vtx = 7

e = 8

u = 0, v = 3, wt = 10

```java
public static void addEdge(ArrayList<Edge>[]graph,int u,int v,int wt) {
    Edge e1 = new Edge(u,v,wt);
    Edge e2 = new Edge(v,u,wt);

    graph[u].add(e1);
    graph[v].add(e2);
}
```



0-2@10, 0-3@10

1-2@10, 1-3@30

2-0@10, 2-1@10

3-0@10, 3-1@30, 3-4@2

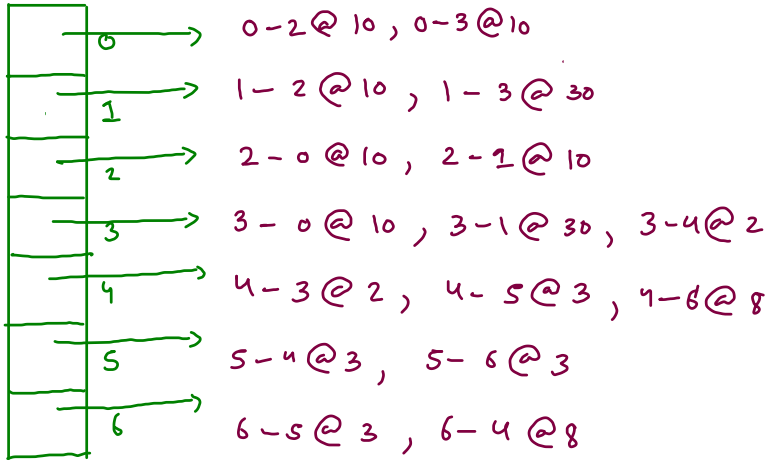
4-3@2, 4-5@3, 4-6@8

5-4@3, 5-6@3

6-5@3, 6-4@8

```java
public static void display(ArrayList<Edge>[]graph) {
    for(int i = 0; i < graph.length;i++) {
        System.out.print(i + " -> ");
        for(Edge ne : graph[i]) {
            int src = i;
            int nbr = ne.nbr;
            int wt = ne.wt;

            System.out.print(src + " - " + nbr + " @ " + wt + ", ");
        }
        System.out.println();
    }
}
```

```
0 -> 0 - 2 @ 10, 0 - 3 @ 10,
1 -> 1 - 2 @ 10, 1 - 3 @ 30,
2 -> 2 - 0 @ 10, 2 - 1 @ 10,
3 -> 3 - 0 @ 10, 3 - 1 @ 30, 3 - 4 @ 2,
4 -> 4 - 3 @ 2, 4 - 5 @ 3, 4 - 6 @ 8,
5 -> 5 - 4 @ 3, 5 - 6 @ 3,
6 -> 6 - 5 @ 3, 6 - 4 @ 8,
```



0 → 0 - 2 @ 10 , 0 - 3 @ 10

1 → 1 - 2 @ 10 , 1 - 3 @ 30

2 → 2 - 0 @ 10, 2 - 1 @ 10

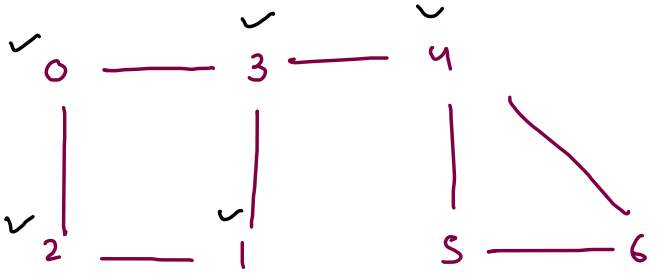
3 → 3 - 0 @ 10, 3 - 1 @ 30 , 3 - 4 @ 2

4 → 4 - 3 @ 2, 4 - 5 @ 3, 4 - 6 @ 8

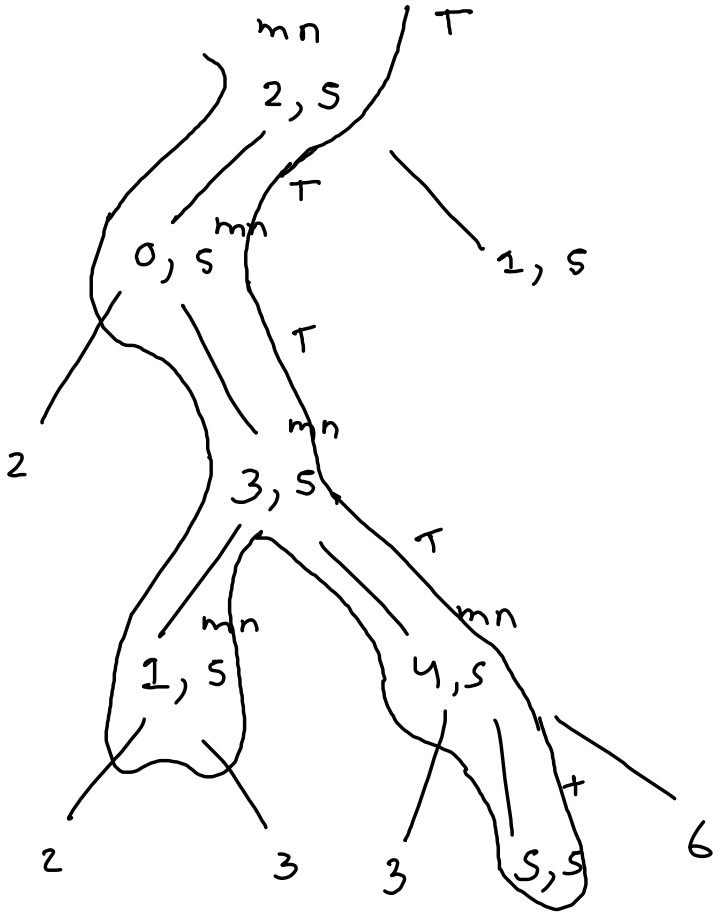
5 → 5 - 4 @ 3, 5 - 6 @ 3

6 → 6 - 5 @ 3, 6 - 4 @ 8

hasPath

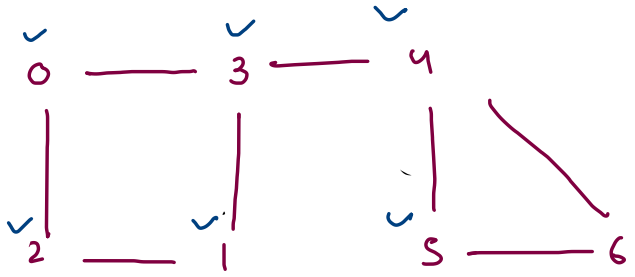


T	T	T	T	T	F	F
0	1	2	3	4	5	6

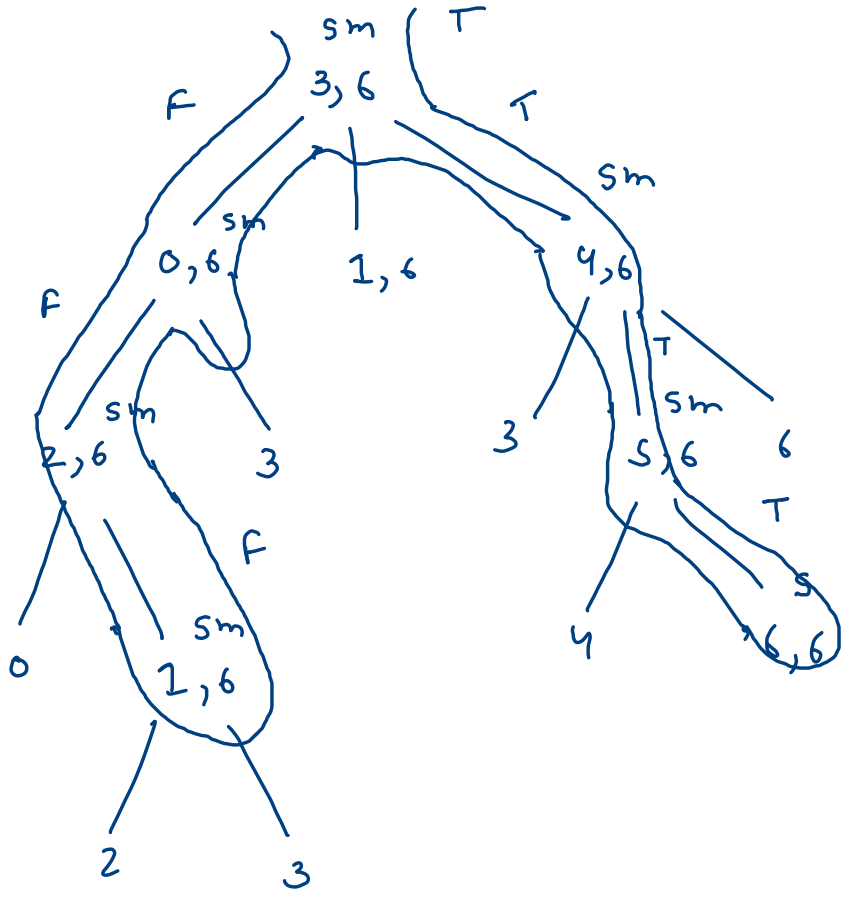


src = 2

dest = 5

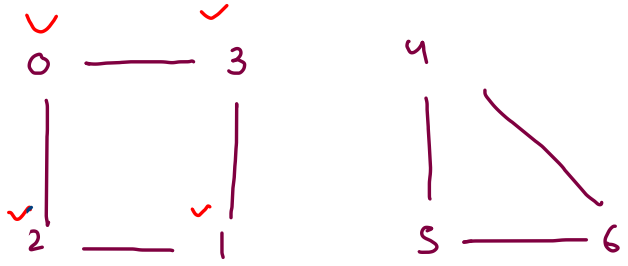


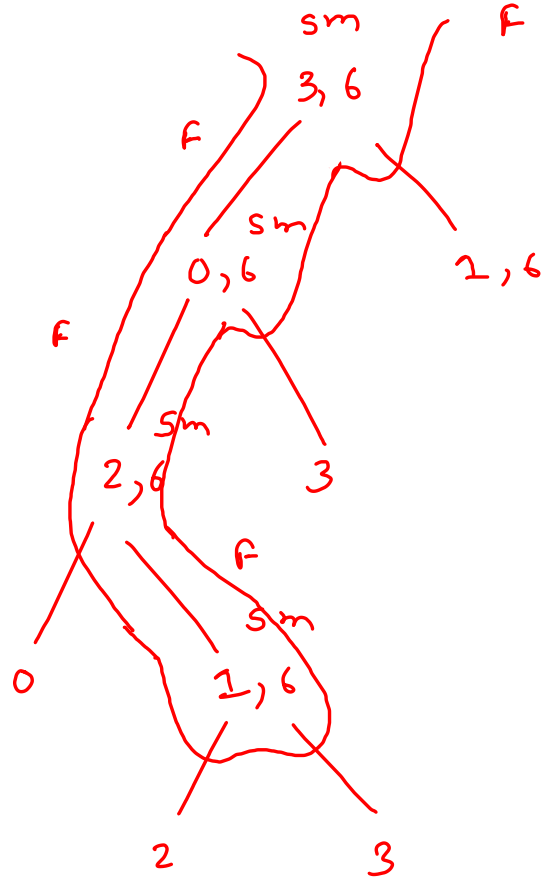

```java
public static boolean hasPath(ArrayList<Edge>[]graph,int src,int dest,boolean[]vis) {

    //self check
    if(src == dest) {
        return true;
    }

    //mark
    vis[src] = true;

    //go to your unvisited nbr
    for(Edge ne : graph[src]) {
        int nbr = ne.nbr;

        if(vis[nbr] == false) {
            boolean hpntod = hasPath(graph,nbr,dest,vis); //has path from nbr to dest
            if(hpntod == true) {
                return true;
            }
        }

    }

    return false;
}
```



Src = 3

dest = 6

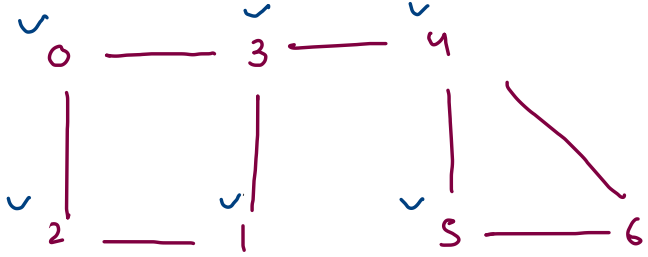




src = 3

dest = 6

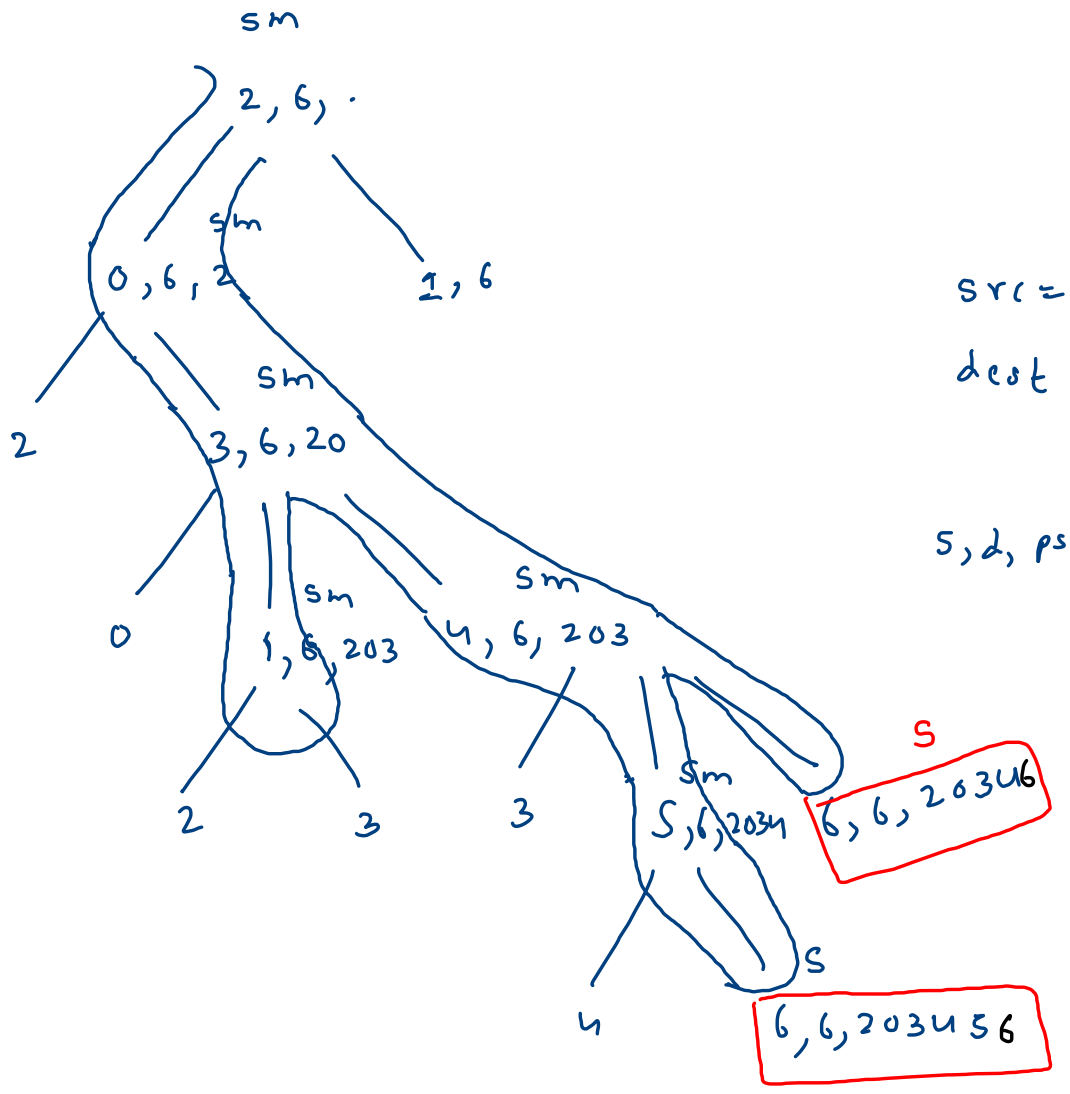
```java
public static boolean hasPath(ArrayList<Edge>[]graph,int src,int dest,boolean[]vis) {

    //self check
    if(src == dest) {
        return true;
    }

    //mark
    vis[src] = true;

    //go to your unvisited nbr
    for(Edge ne : graph[src]) {
        int nbr = ne.nbr;

        if(vis[nbr] == false) {
            boolean hpntod = hasPath(graph,nbr,dest,vis); //has path from nbr to dest
            if(hpntod == true) {
                return true;
            }
        }

    }

    return false;
}
```




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

vis[src] = true;

for(Edge ne : graph[src]) {

    int nbr = ne.nbr;

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

2 0 3 4 5 6

2 0 3 4 6



src = 2

dest = 6

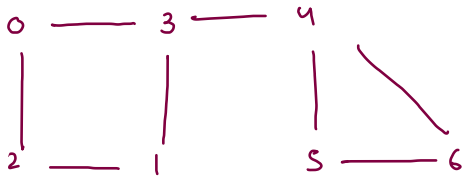
s, d, psf

2 1 3 4 5 6

2 0 3 4 5 6

2 1 3 4 6

2 0 3 4 6



src = 2, dest = 6

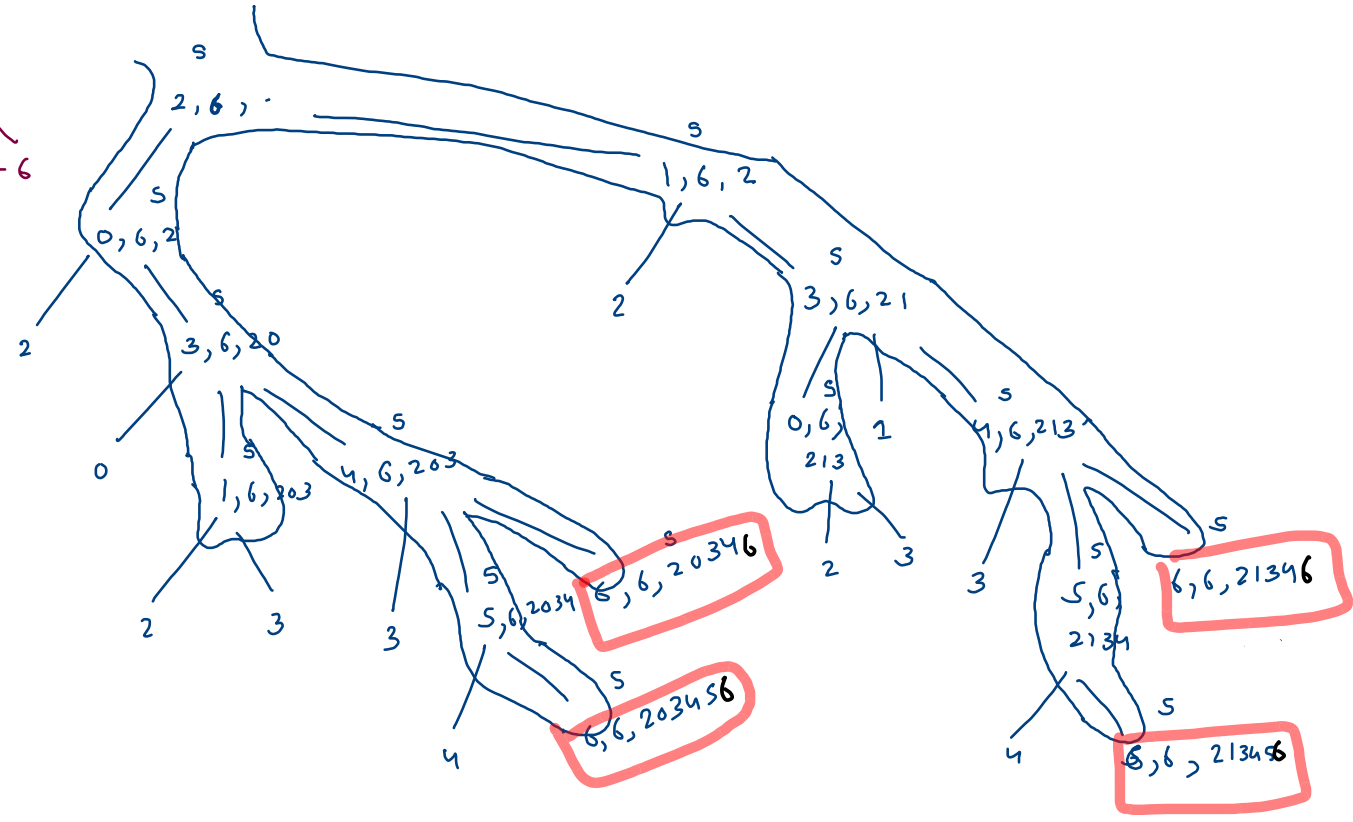
```
if(src == dest) {
    psf += src;
    System.out.println(psf);
    return;
}[]

vis[src] = true;

for(Edge ne : graph[src]) {

    int nbr = ne.nbr;

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

vis[src] = false;
```

2 0 3 4 5 6

2 0 3 4 6

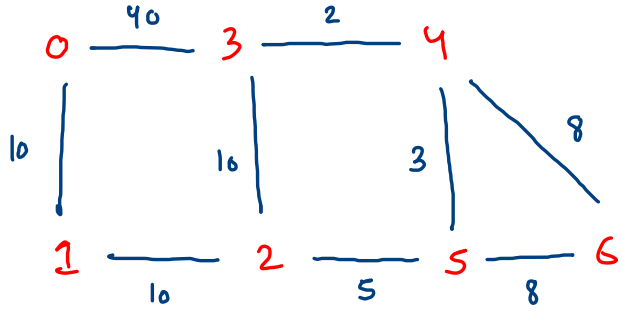
2 1 3 4 5 6

2 1 3 4 6



```
7
9
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
```

```
0 s
6 d
30 cr
4 k
```



src = 0

dest = 6

0 1 2 5 6 @ 28

0 1 2 5 4 6 @ 36

0 1 2 3 4 6 @ 40

0 1 2 3 4 5 6 @ 43

0 3 4 6 @ 50

0 3 4 5 6 @ 53

0 3 2 5 6 @ 63

0 3 2 5 4 6 @ 66

max paths : 0 3 2 5 4 6 @ 66

min paths : 0 1 2 5 6 @ 28

ceil paths (30) : 0 1 2 5 4 6 @ 36

floor path (30) ; 0 1 2 5 6 @ 28

4th largest path : 0 3 4 6 @ 50

0346 @ 50

01256 @ 25

01254 6 @ 36

03456 @ 53

01236 @ 40

03256 @ 28

01234 6 @ 43

032546 @ 66

cr = 30

Smallest	Largest	ceil	floor
sp = 01256	lp = 032546	cp = 012546	fp = 03256
spw = 25	lpw = 66	cpw = 36	fpw = 28

```
//smallest
if(spathwt > wsf) {
    spath = psf;
    spathwt = wsf;
}

//largest
if(lpathwt < wsf) {
    lpath = psf;
    lpathwt = wsf;
}

//ceil -> qualified min
if(wsf > criteria && wsf < cpathwt) {
    cpath = psf;
    cpathwt = wsf;
}

//floor -> qualified max
if(wsf < criteria && wsf > fpathwt) {
    fpath = psf;
    fpathwt = wsf;
}
```

0346 @ 50

01256 @ 25

01254 6 @ 36

0345 6 @ 53

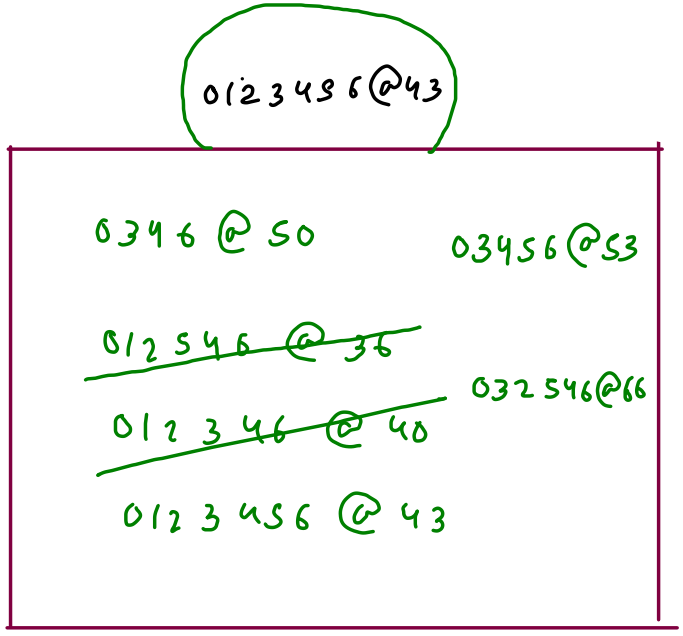
0 1 2 3 4 6 @ 40

03256 @ 29

01 2 3 4 5 6 @ 43

032546 @ 66

k = 4




```
//smallest
if(spathwt > wsf) {
    spath = psf;
    spathwt = wsf;
}

//largest
if(lpathwt < wsf) {
    lpath = psf;
    lpathwt = wsf;
}

//ceil -> qualified min
if(wsf > criteria && wsf < cpathwt) {
    cpath = psf;
    cpathwt = wsf;
}

//floor -> qualified max
if(wsf < criteria && wsf > fpathwt) {
    fpath = psf;
    fpathwt = wsf;
}
```