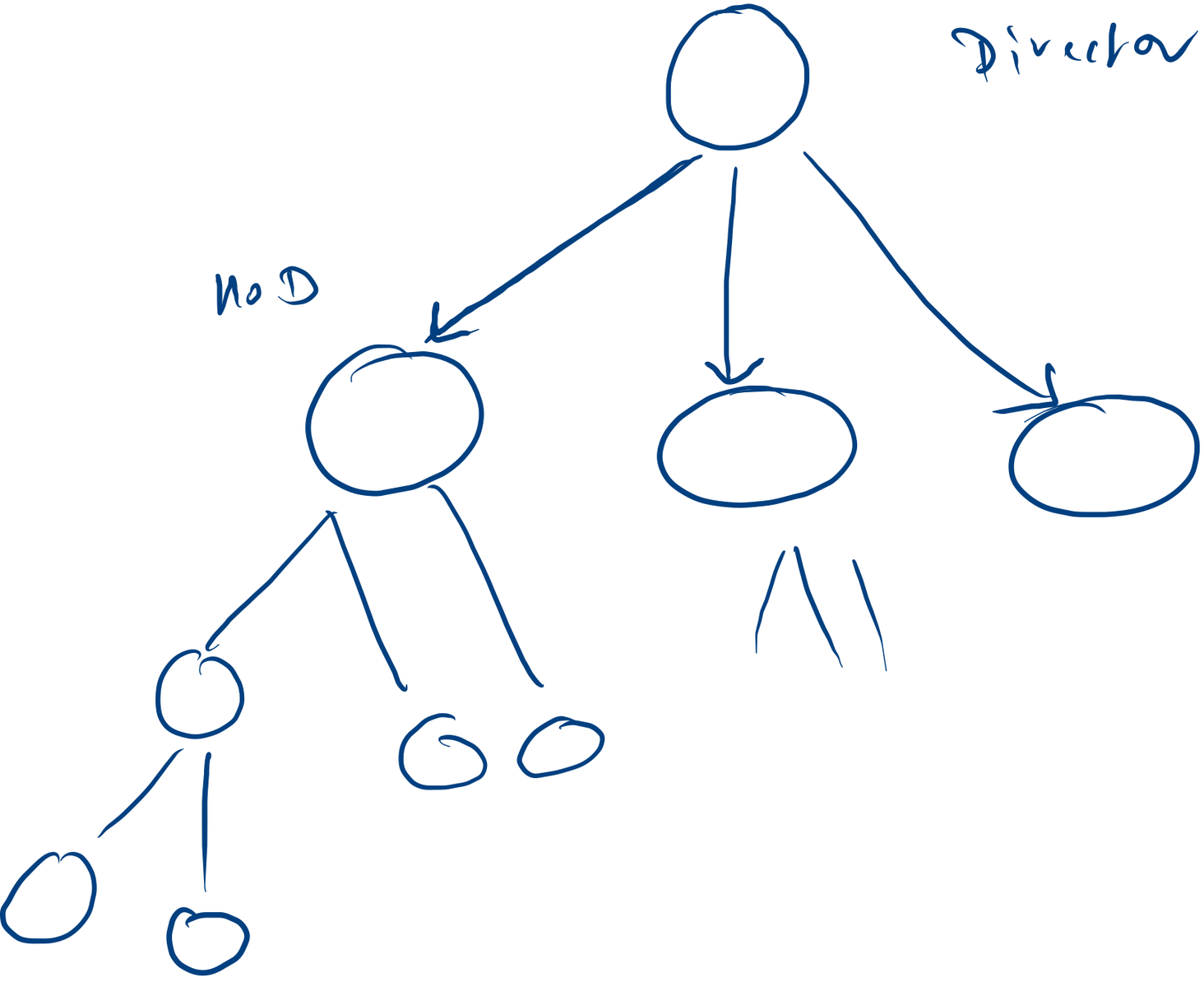
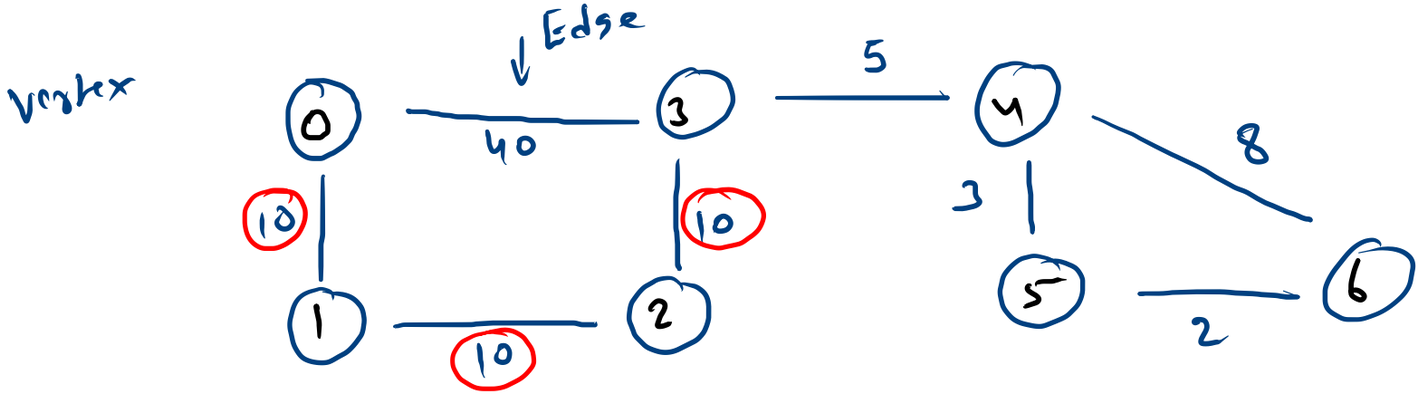
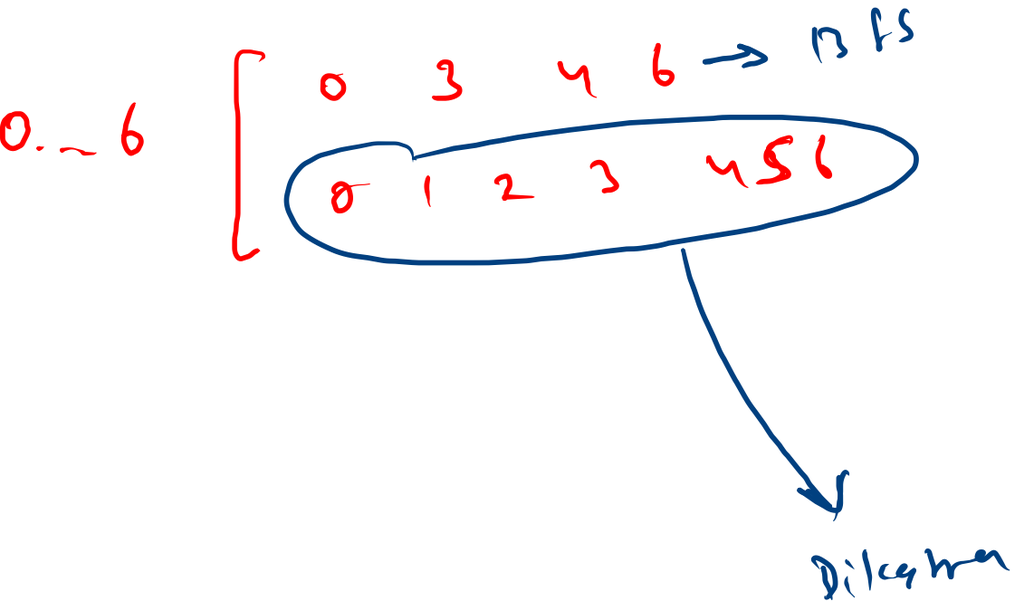
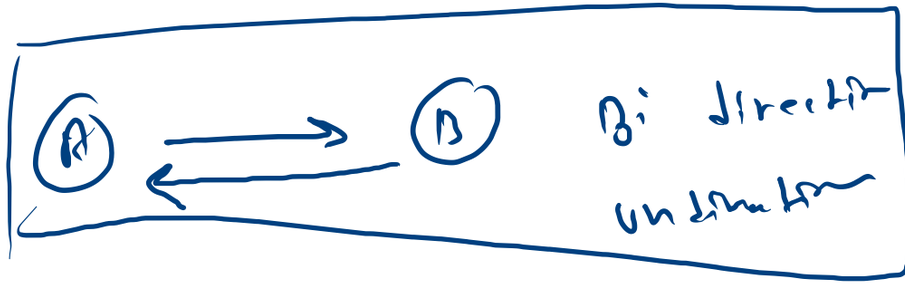

Director
Node

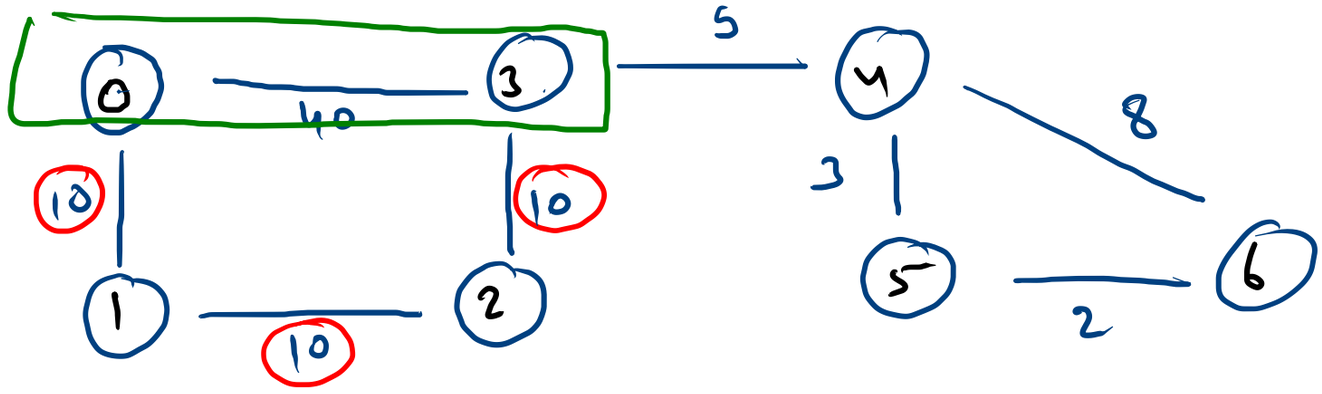

Vertex
Edge
0
3
4
5
6
1
2
40
10
10
10
5
3
8
2
10

MST


0 3 4 6 → BFS
0 1 2 3 4 5 6
0..6
Dictionary


A
B
Bi direction
undirection

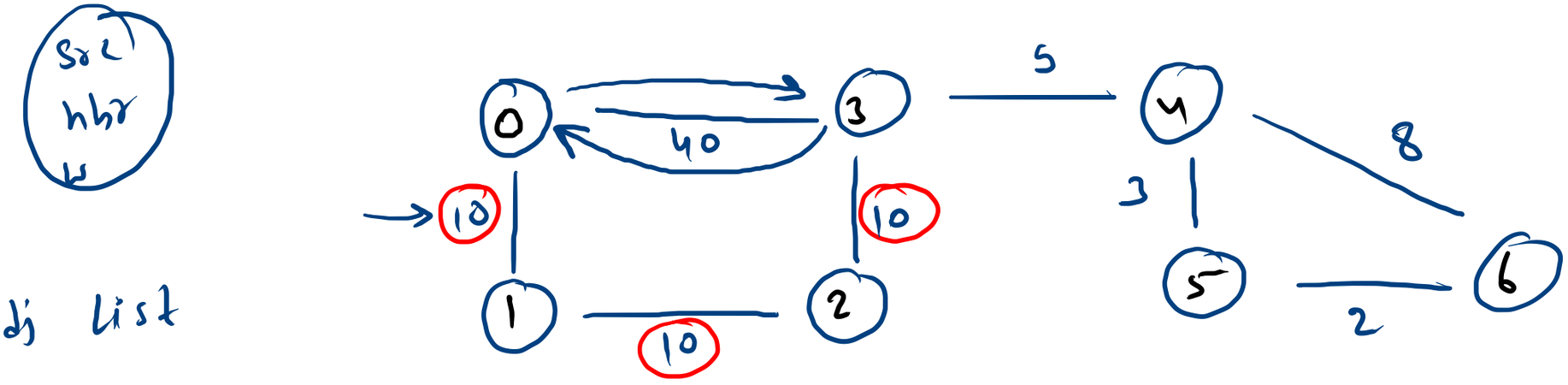

A
B
uni-direction



Vertex 2

Adj Matrix

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



Adj List

0 • — [{0, 3, 40}, {0, 1, 10}]

1 — [{1, 0, 10}]

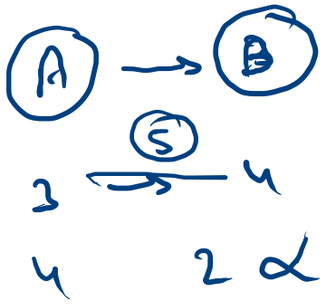
2 — [ ]

3 [{3, 0, 40}]

4 [ ]

5 [{5, 6, 2}]

6 [{6, 5, 2}]



2

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

0 — [{0, 3, 40}, {0, 1, 10}]
1 — [{1, 0, 10}]
2 — [ ]
3 — [{3, 0, 40}]
4 — [ ]
5 — [{5, 6, 2}]
6 — [{6, 5, 2}]

7 — V

8 — E

0 1 10

1 2 10

2 3 10

0 3 10

3 4 10

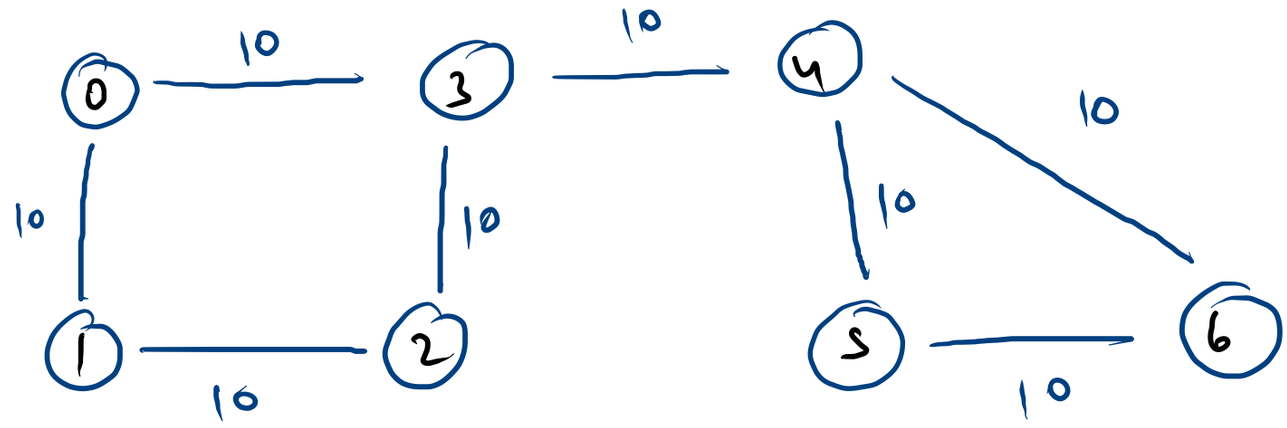
4 5 10

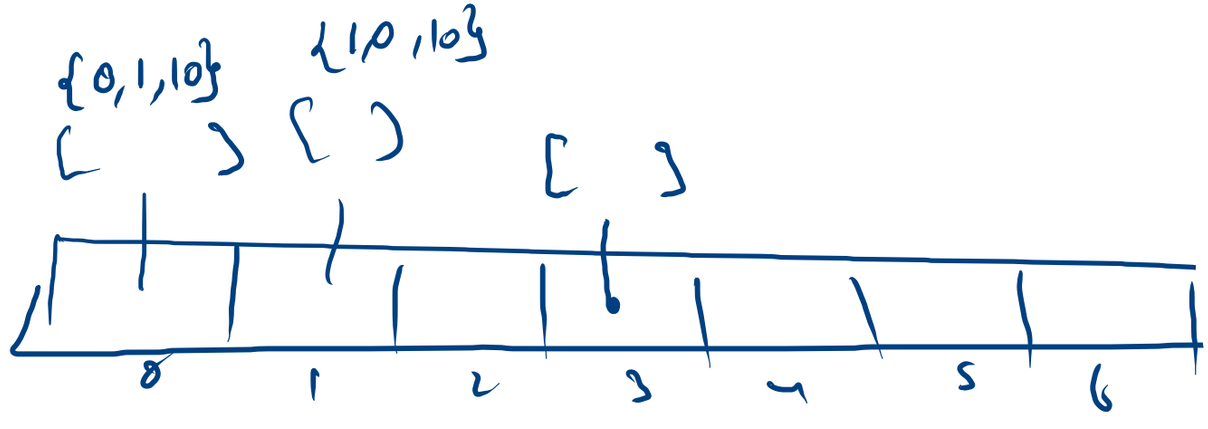
5 6 10

4 6 10

0

6





0 ← src

6 ← des

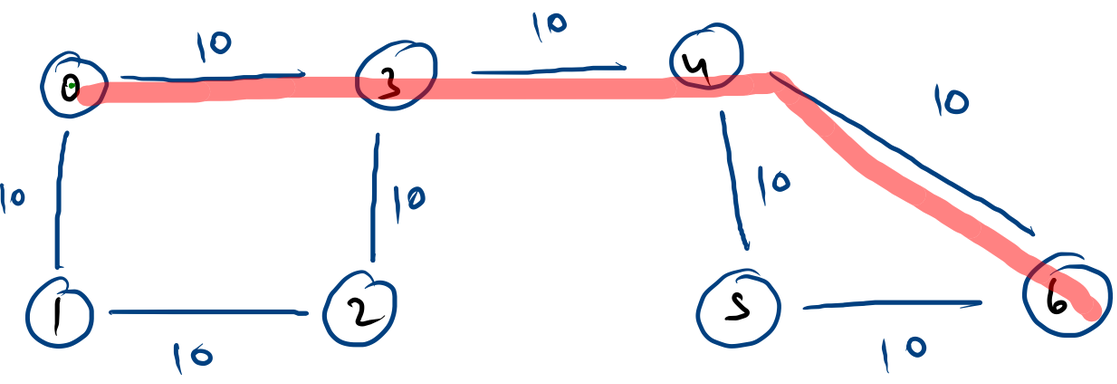
class Edge {

int src

int nbr

int w

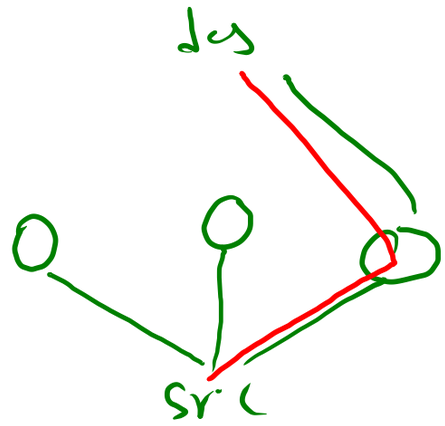
}



src = 0

des = 6

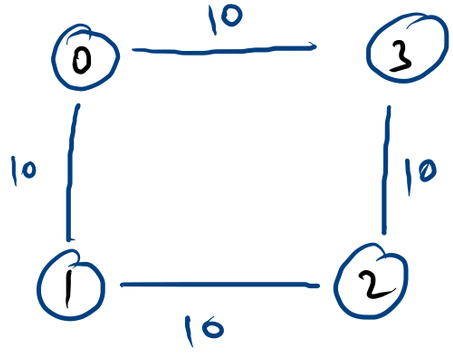
True



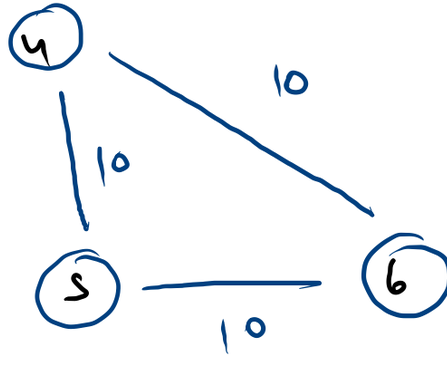
src = 0

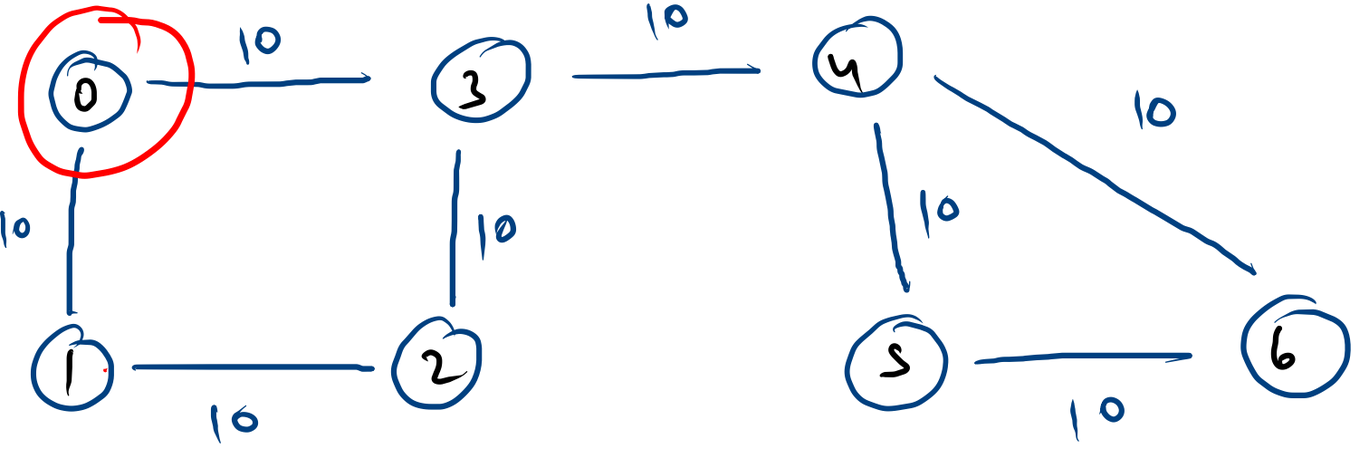
des = 6

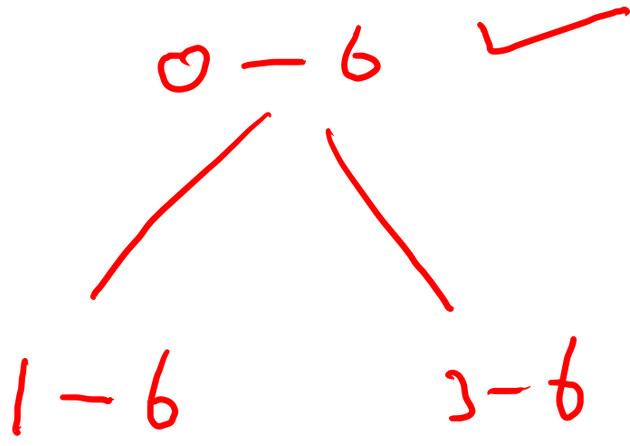
False

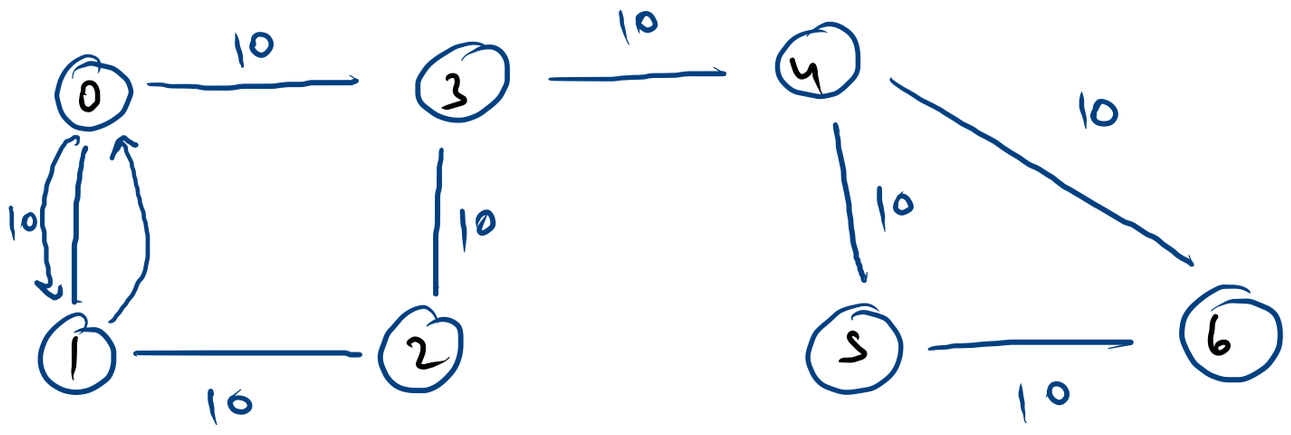


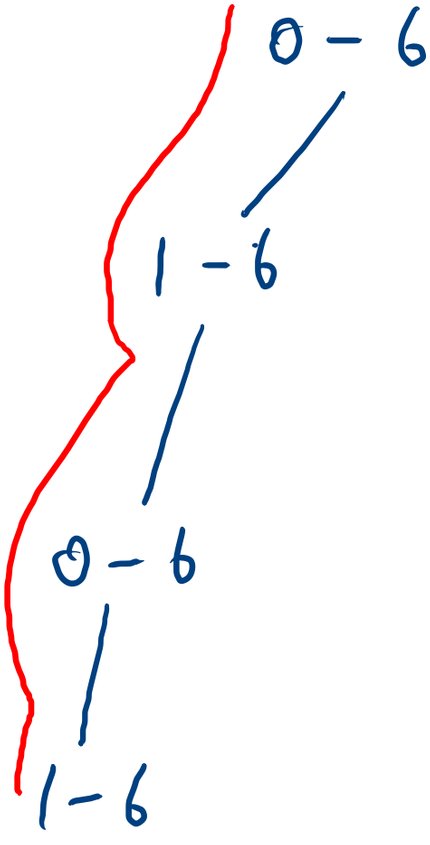
?



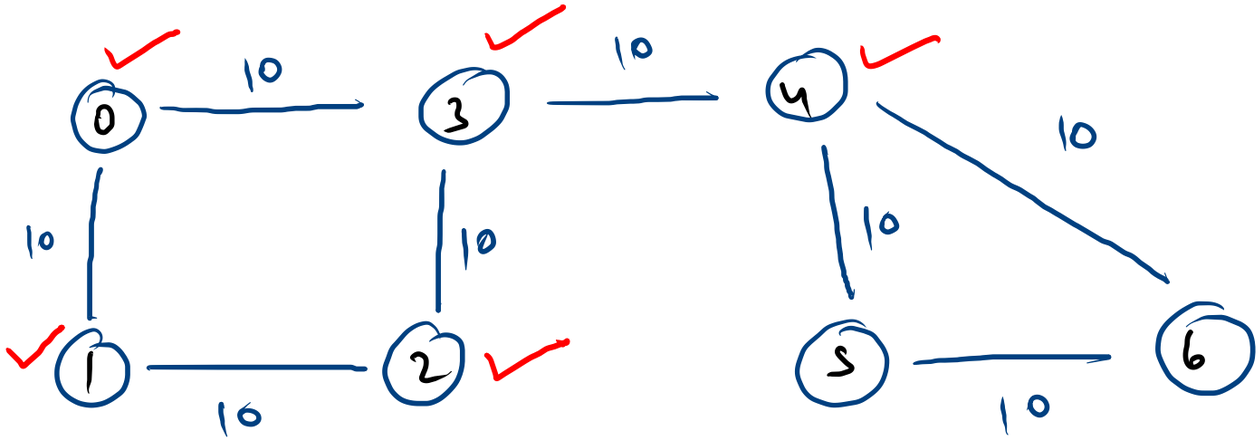

0
3
4
1
2
5
6
10
10
10
10
10
10
10


0 - 6
1 - 6
3 - 6

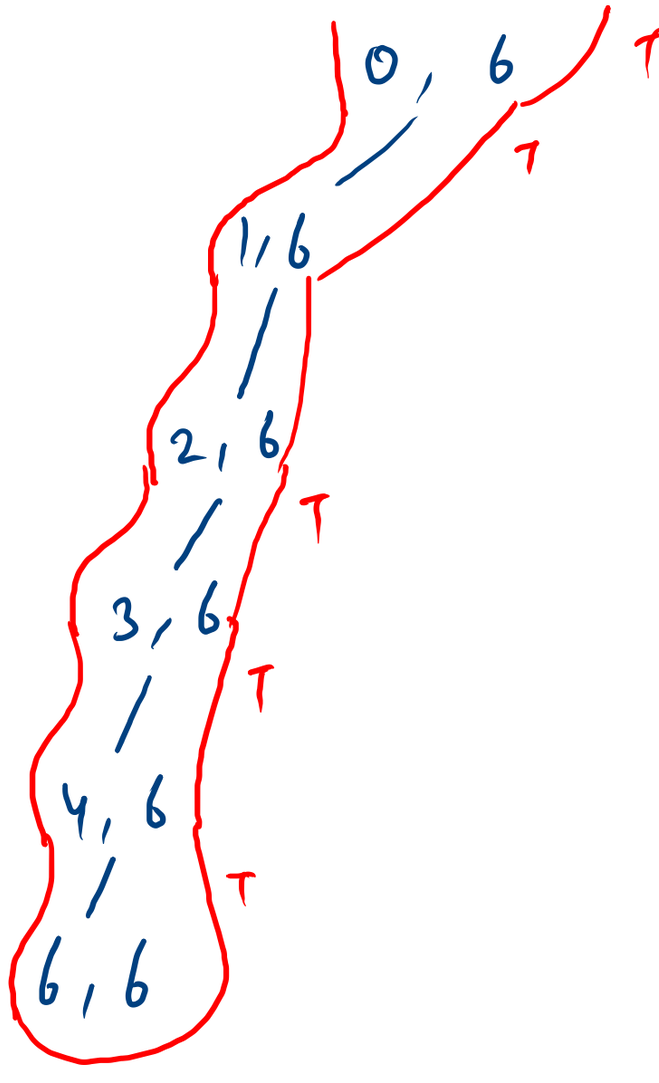


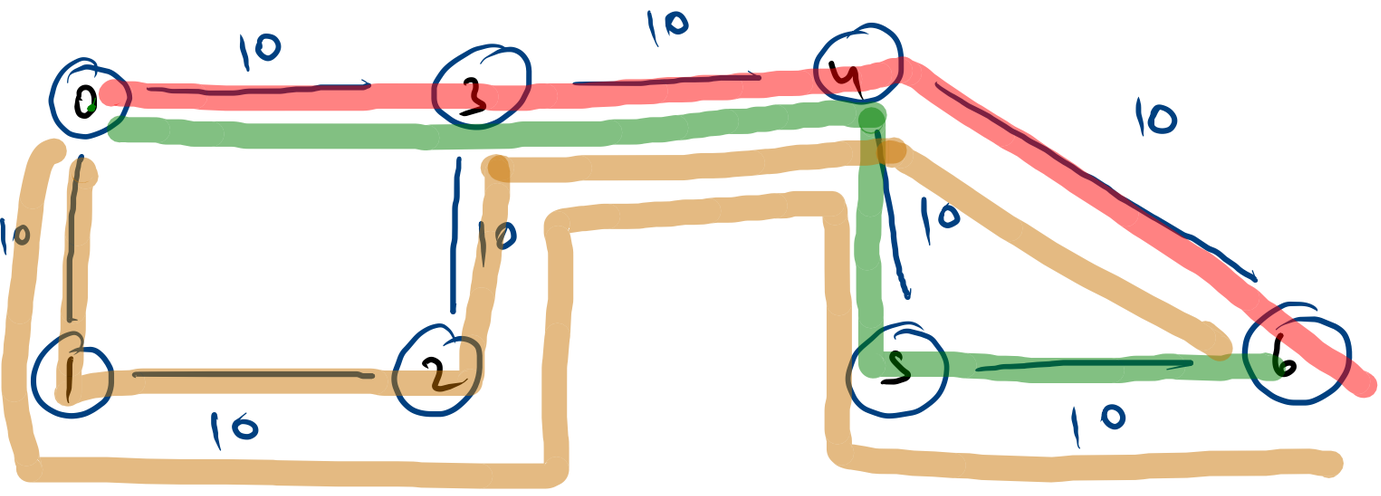



```
static boolean hasPath(ArrayList<Edge>[]graph, int src, int des){

    if(src == des)return true;

    for(Edge edge : graph[src]){
        boolean exist = hasPath(graph, edge.nbr, des);
        if(exist){
            return true;
        }
    }

    return false;
}
```




```
static boolean hasPath(ArrayList<Edge>[]graph, int src, int des

    if(src == des)return true;

    visited[src] = true;
    for(Edge edge : graph[src]){
        if(visited[edge.nbr] == true)continue;
        boolean exist = hasPath(graph, edge.nbr, des, visited);
        if(exist){
            return true;
        }
    }

    return false;
}
```





0 - 6 - "0"

1 - 6 - "0 1"

src → 0

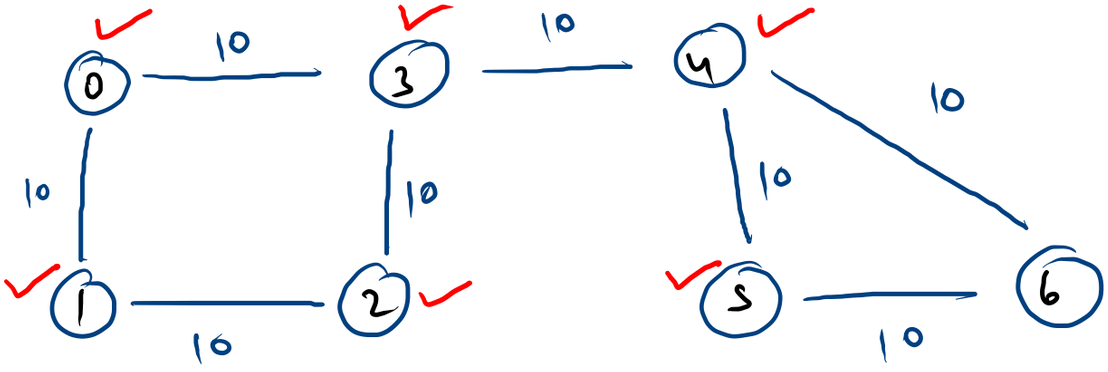
dest → 6

0 3 4 6

0 3 4 5 6

0 1 2 3 4 6

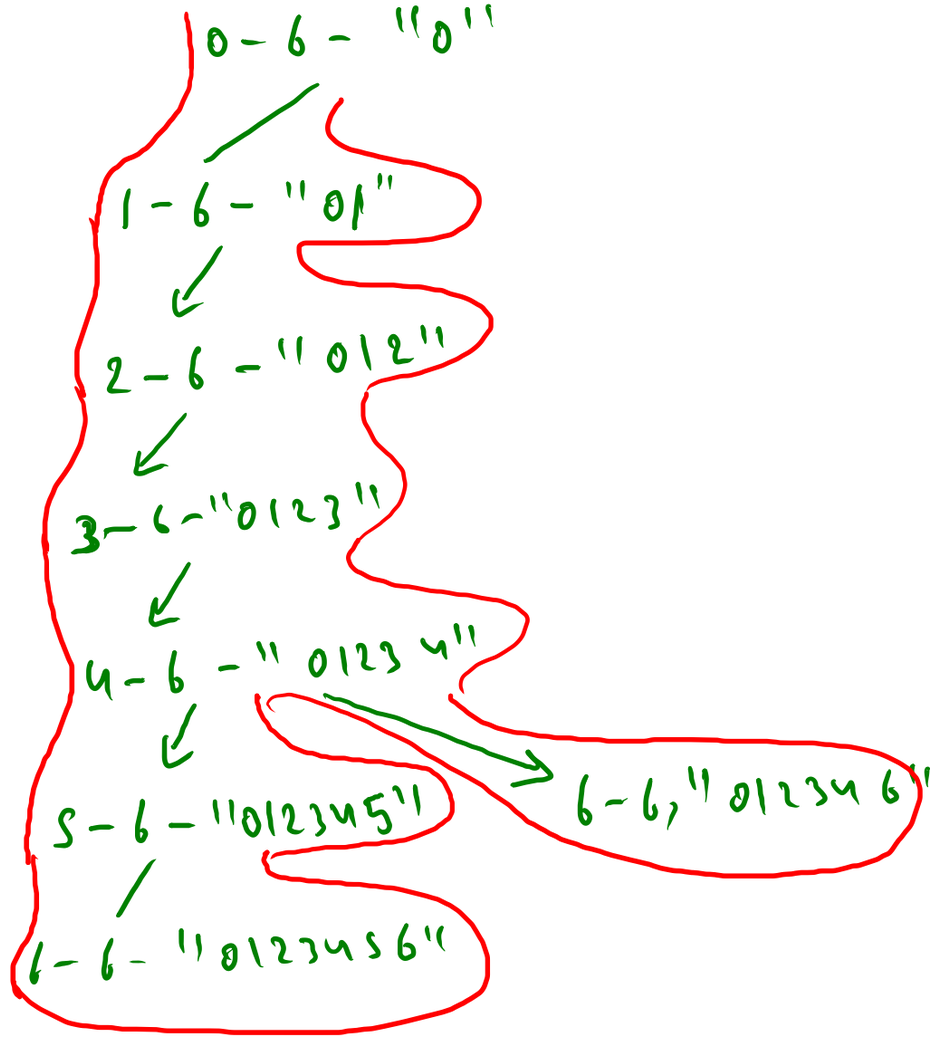
0 1 2 3 4 5 6



0123456

01246

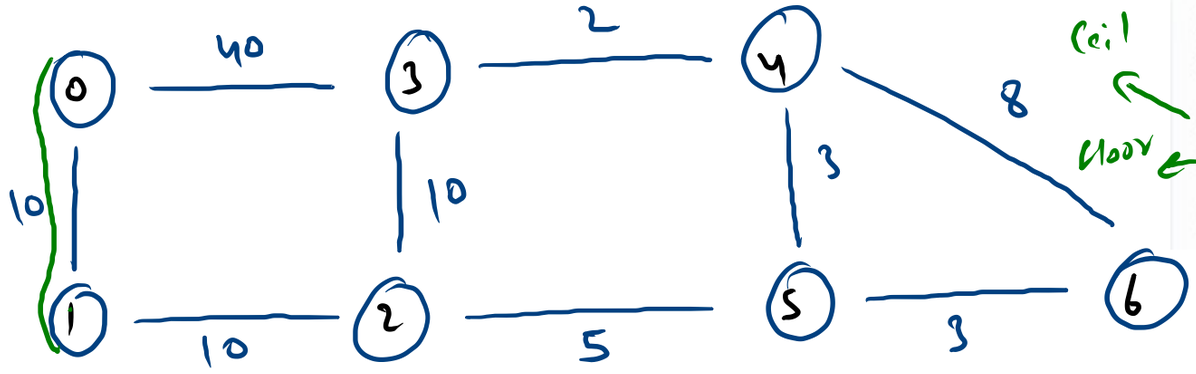
```
static void allPaths(ArrayList<Edge>graph[], int src, int des,
    if(src == des){
        System.out.println(psf);
        return;
    }

    visited[src] = true;

    for(Edge edge: graph[src]){
        if(visited[edge.nbr])continue;
        allPaths(graph, edge.nbr, des, psf+edge.nbr, visited);
    }
}
```



0 - 6 - 0

1 - 6 - 0 of 10

2 - 6 - 10 of 6



Ceil

Floor

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

```
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 6
30
4
```

src = 0

dst = 6

criteria

k

0 3 4 6 → 50

0 3 4 5 6 → 48

0 3 2 5 6 → 58

0 3 2 5 4 6 → 66 → largest

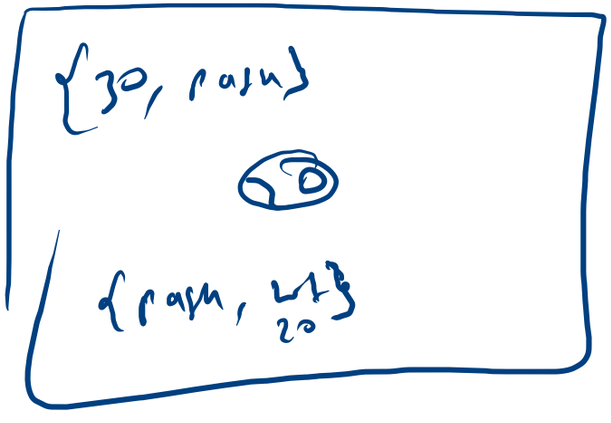
0 1 2 3 4 6 → 40

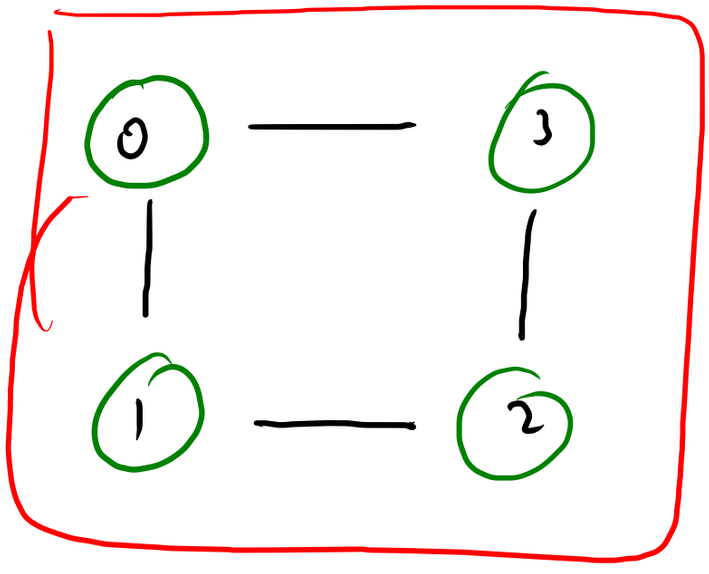
0 1 2 3 4 5 6 → 38

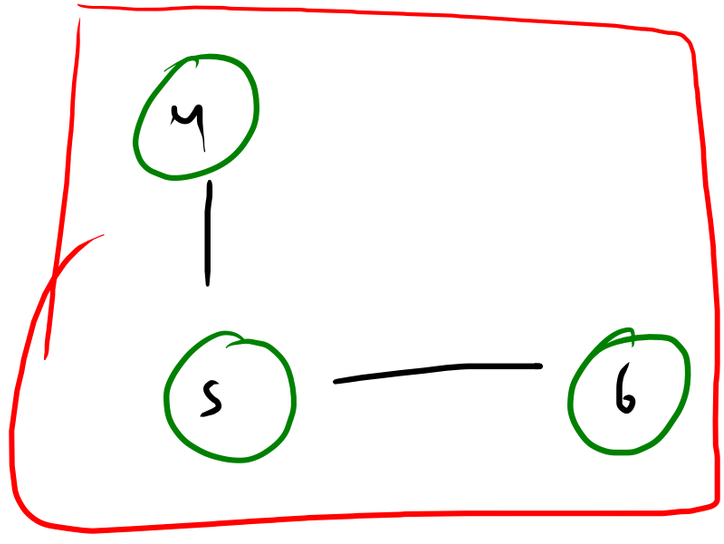
0 1 2 5 6 → 28 → smallest, just smaller

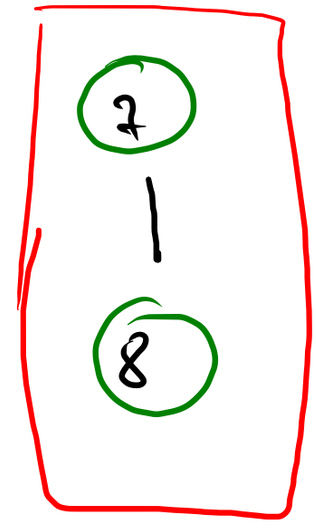
0 1 2 5 4 6 → 36 → just larger

10 20 15 30 5

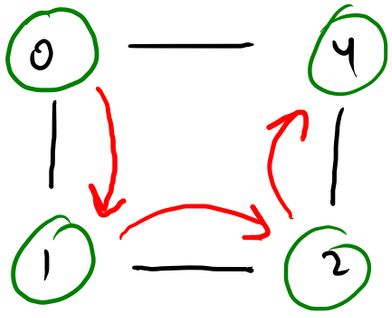


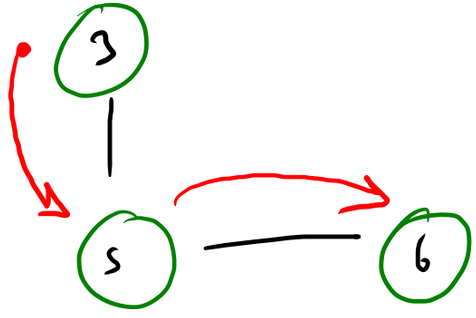


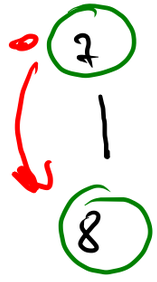


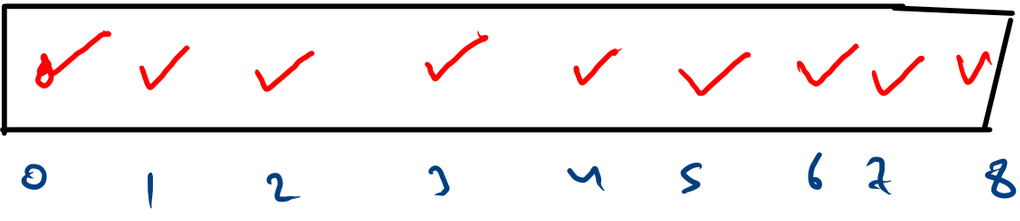


☆ [[0, 1, 2, 3] , [4, 5, 6] , [7, 8]]



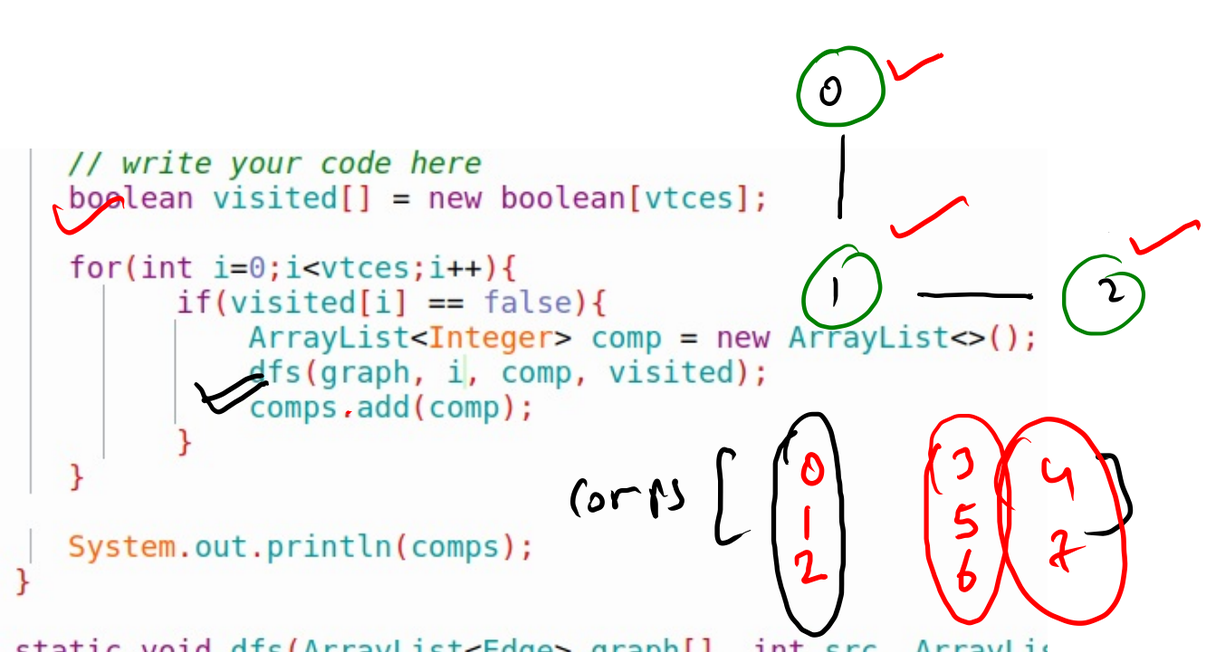


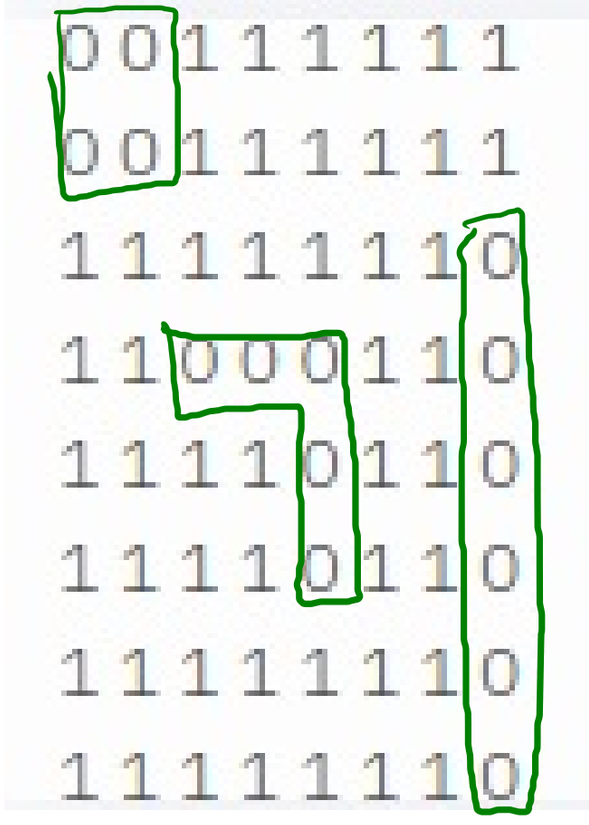




comps [0 1 2 4 / 3 5 6]

```java
    // write your code here
    boolean visited[] = new boolean[vtces];

    for(int i=0;i<vtces;i++){
        if(visited[i] == false){
            ArrayList<Integer> comp = new ArrayList<>();
            dfs(graph, i, comp, visited);
            comps.add(comp);
        }
    }

    System.out.println(comps);
}

static void dfs(ArrayList<Edge> graph[], int src, ArrayLis

    visited[src] = true;
    comp.add(src);

    for(Edge edge: graph[src]){
        if(visited[edge.nbr])continue;
        dfs(graph, edge.nbr, comp, visited);
    }

}
```

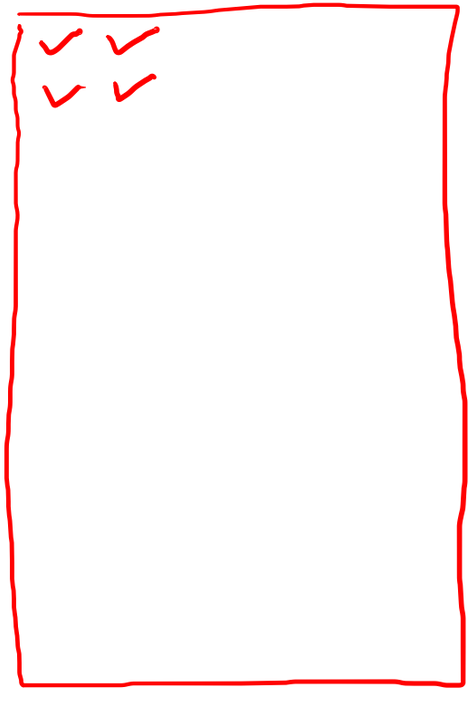




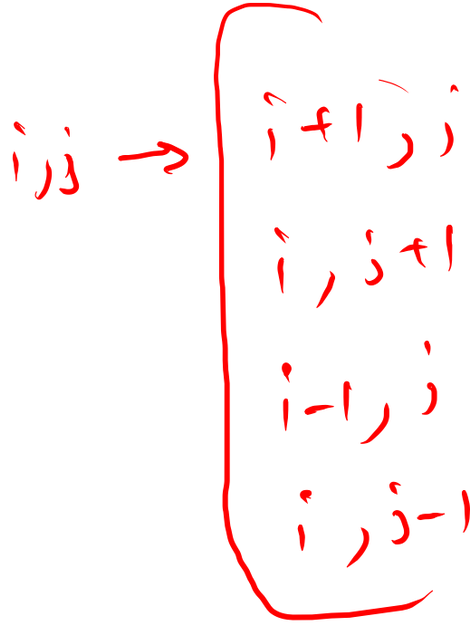
0 → land

1 → water

3 ←



00111111
00111111
11111110
11000110
11110110
11110110
11111110
11111110

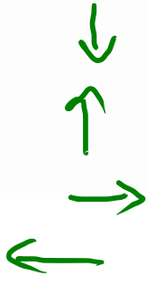


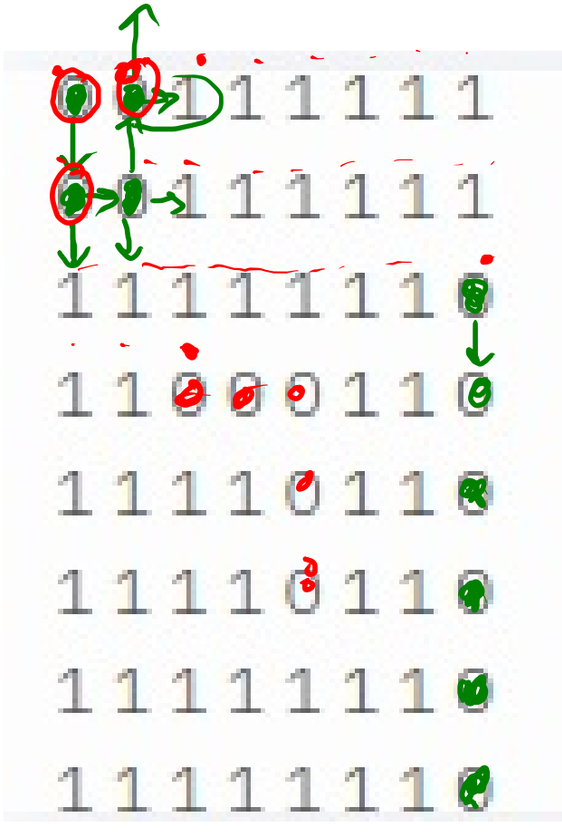
count = 0 + 2 ?

```
boolean visited[][] = new boolean[m][n];

int count=0;
for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
        if(arr[i][j] == 0 && visited[i][j] == false){
            dfs(arr, i, j, visited);
            count++;
        }
    }
}
System.out.println(count);
```

arr[i][j]==1

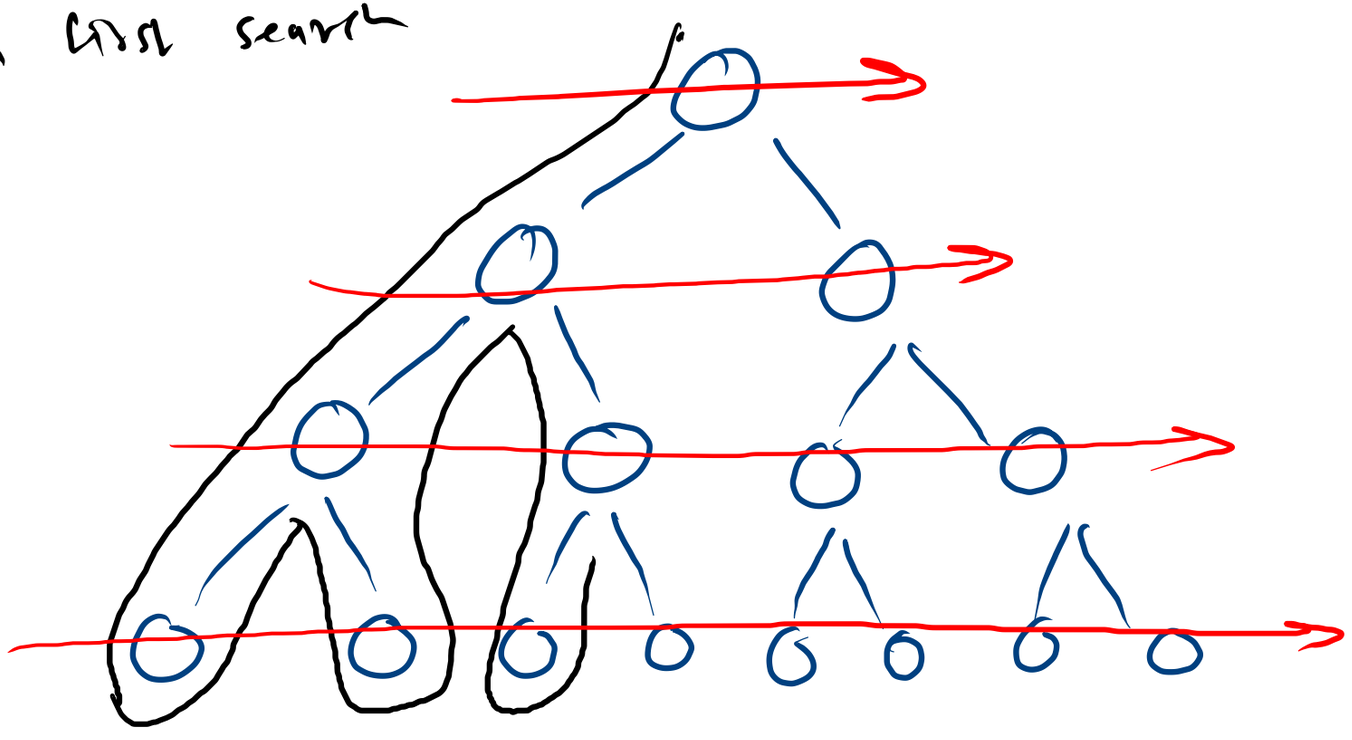
```
atic void dfs(int arr[][], int i, int j, boolean visited[][]){
  if(i < 0 || j < 0 || i >= visited.length || j >= arr[0].length || visited[i][j] == true)return;

  visited[i][j] = true;

  dfs(arr, i+1, j, visited);
  dfs(arr, i-1, j, visited);
  dfs(arr, i, j+1, visited);
  dfs(arr, i, j-1, visited);
```



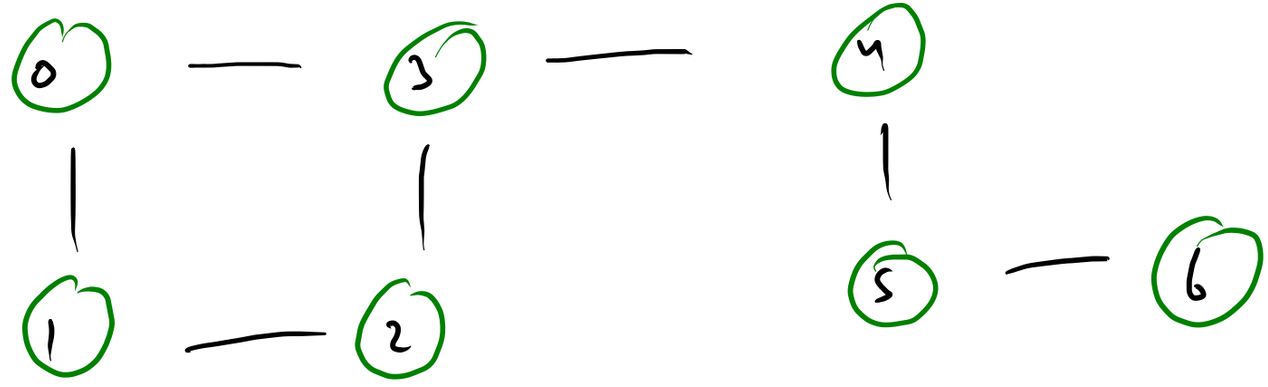


✓ DFS → depth first search

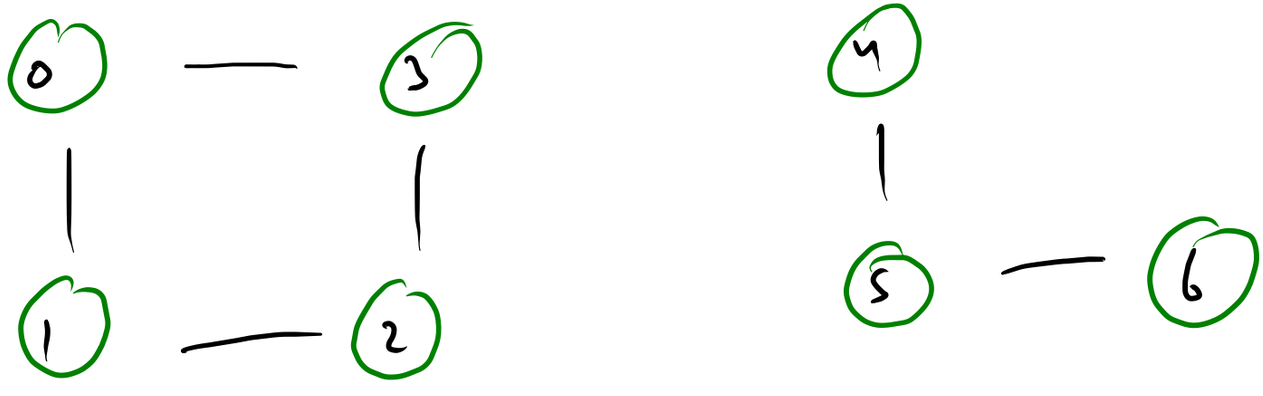
✓ BFS →



M.V is graph connected



→ True



→ False