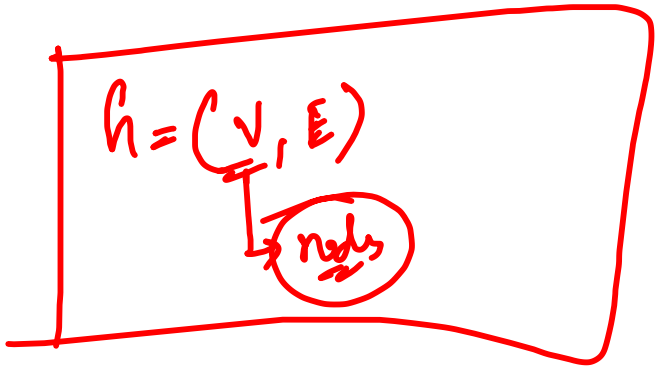
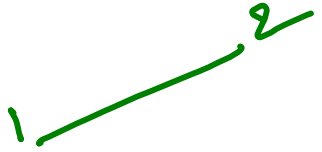
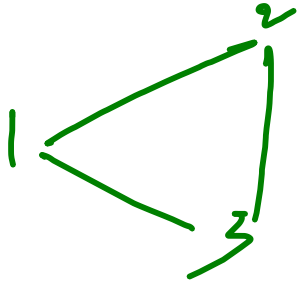
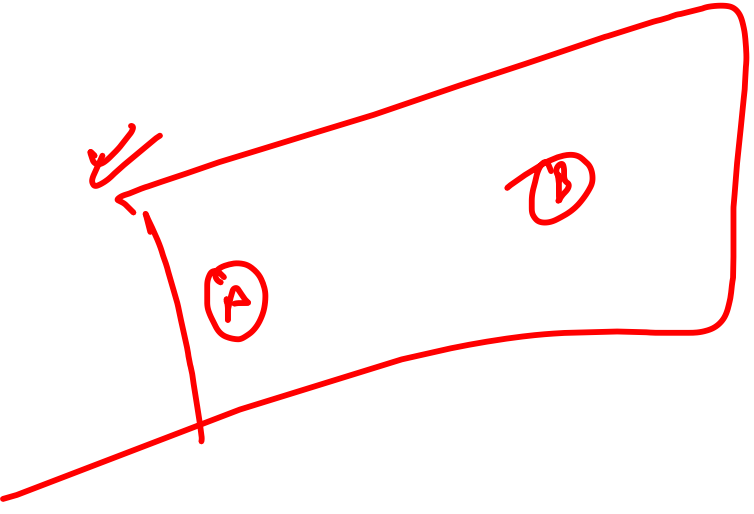
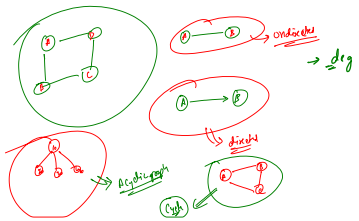
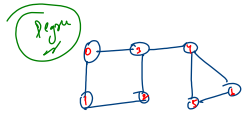

G = (V, E)
nodes


1
2


1
2
3

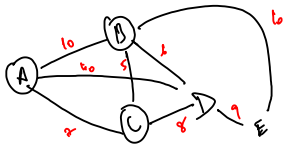

A
B



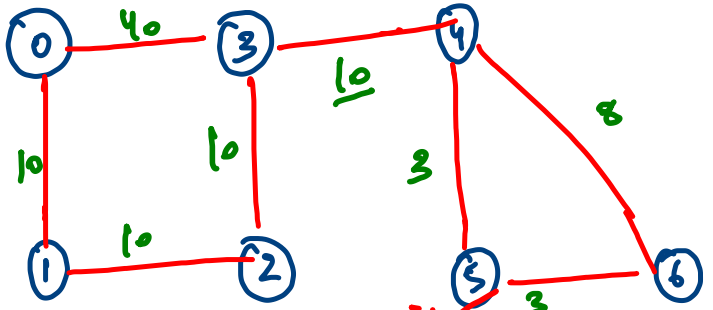


deg(0) = 2
deg(1) = 2
deg(2) = 2
deg(3) = 3

deg(5) = 2
deg(6) = 2
deg(4) = 3

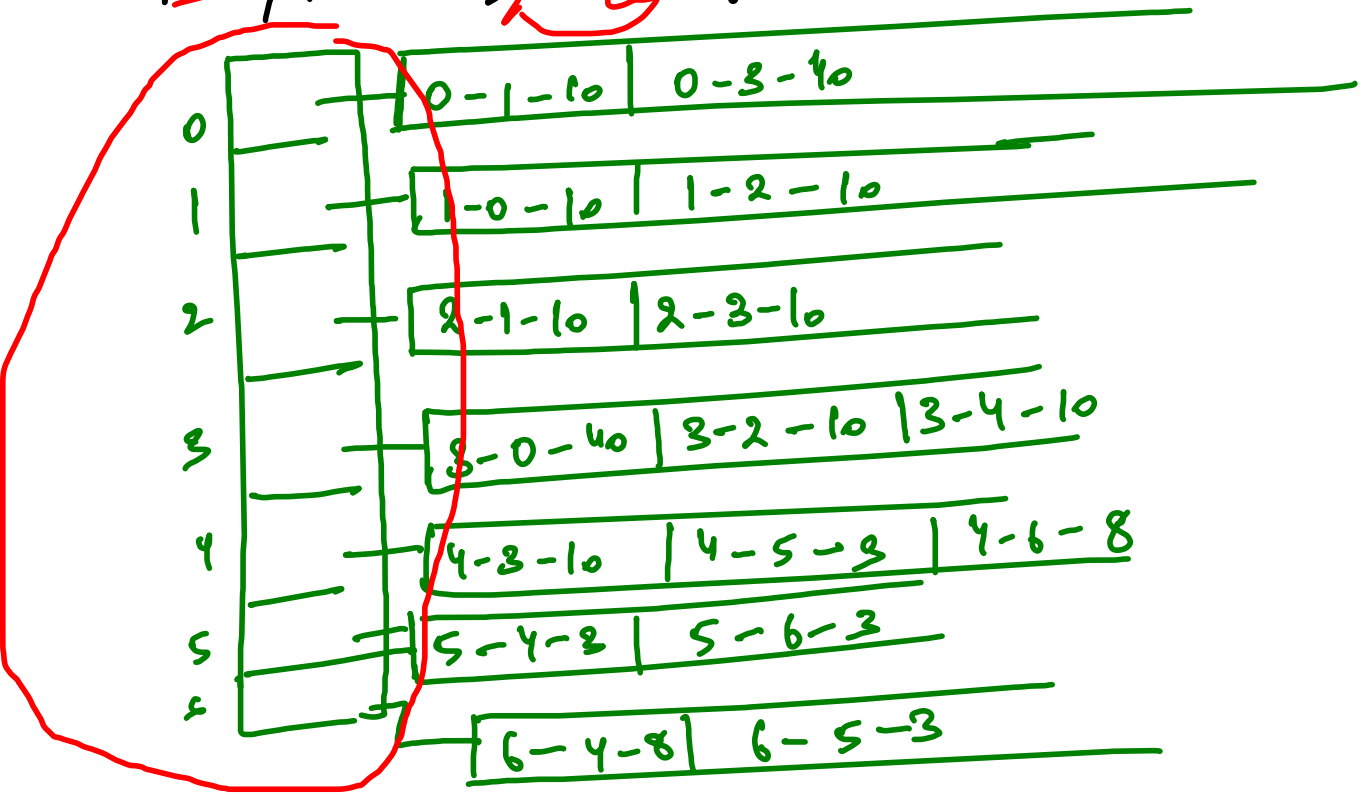


A → E
- has path
- All path
- min path (vtcs)
- min path (weight) (can)
- path → vtcs visit once exactly

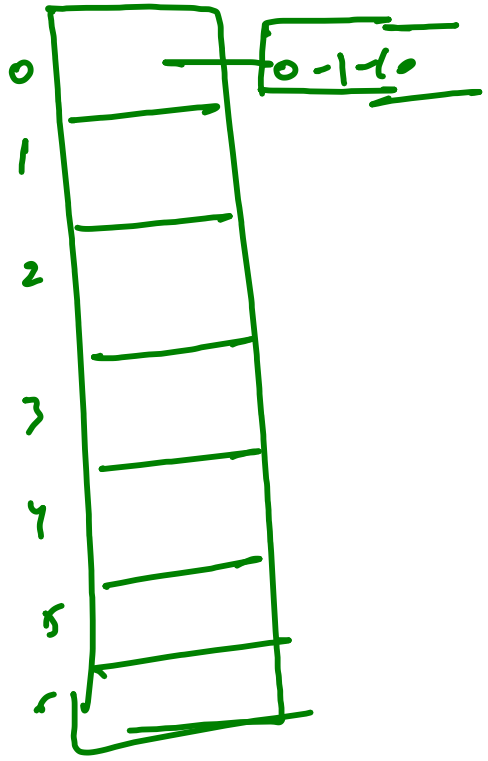


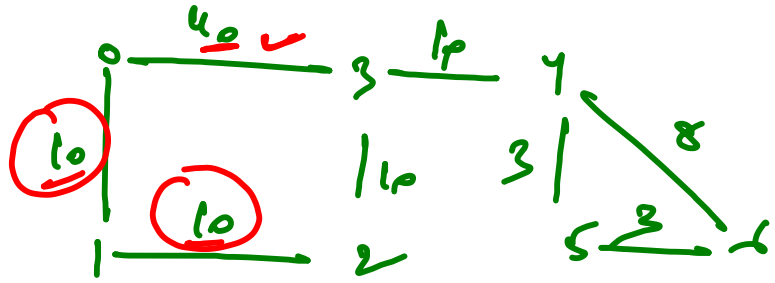
Edge
- src
- dest
- wt

ArrayList<Edge>[] graph



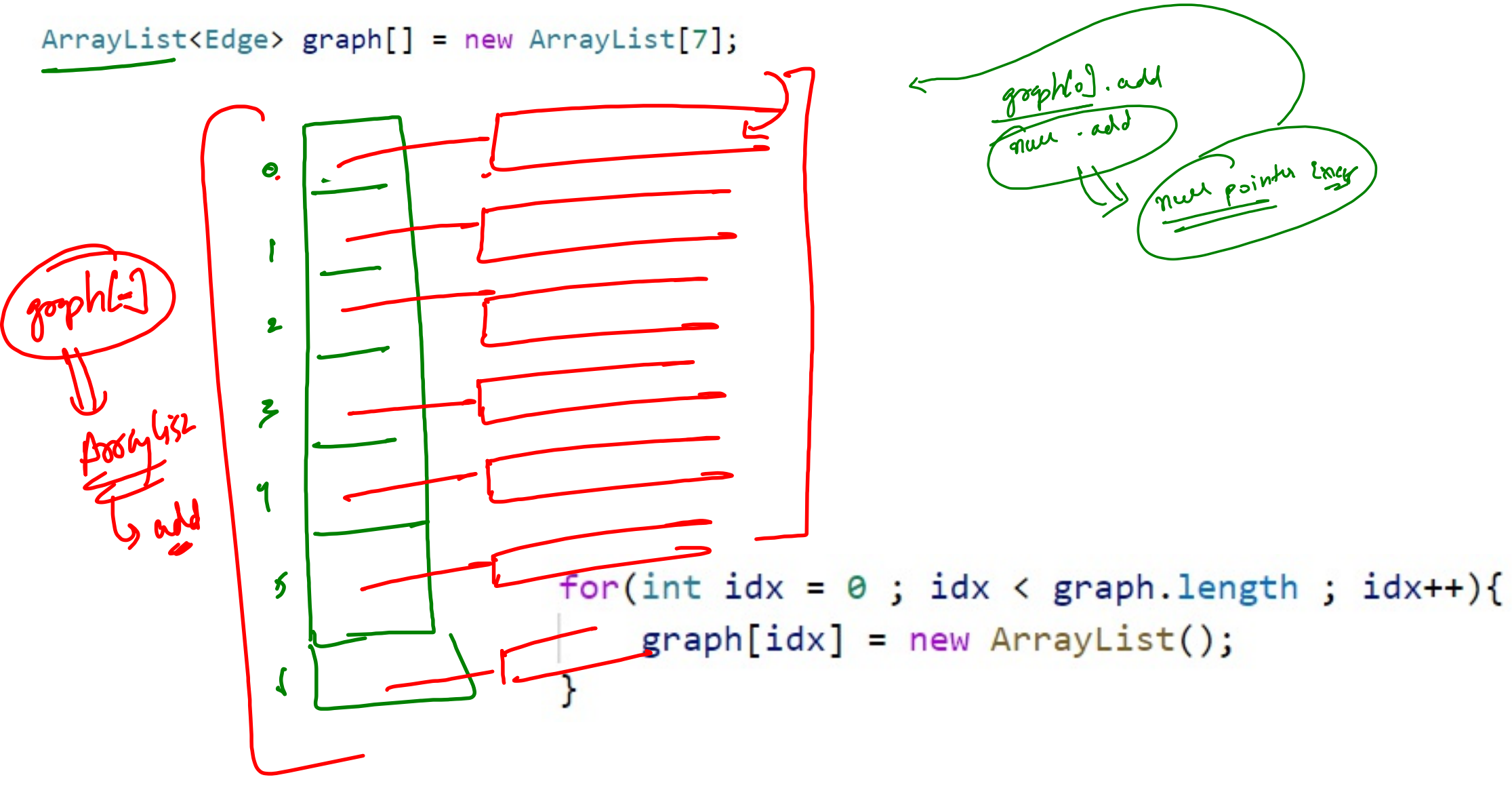
```java
public static void main(String[] args) {
    ArrayList<Edge> graph[] = new ArrayList[7];

    graph[0].add(new Edge(0,1,10));
    graph[0].add(new Edge(0,3,40));

    graph[1].add(new Edge(1,0,10));
    graph[1].add(new Edge(1,2,10));

    graph[2].add(new Edge(2,1,10));
    graph[2].add(new Edge(2,3,10));

    graph[3].add(new Edge(3,0,40));
    graph[3].add(new Edge(3,2,10));
    graph[3].add(new Edge(3,4,10));

    graph[4].add(new Edge(4,3,10));
    graph[4].add(new Edge(4,5,3));
    graph[4].add(new Edge(4,6,8));

    graph[5].add(new Edge(5,6,3));
    graph[5].add(new Edge(5,4,3));

    graph[6].add(new Edge(6,5,3));
    graph[6].add(new Edge(6,4,8));
}
```



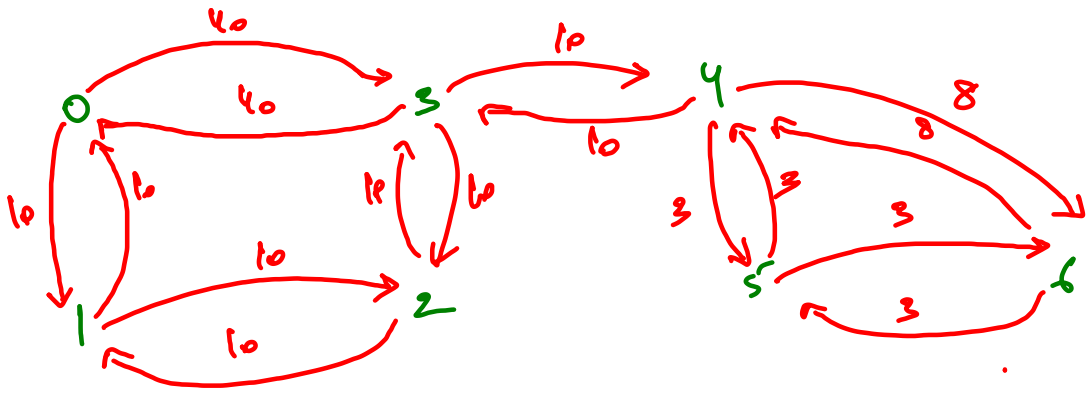


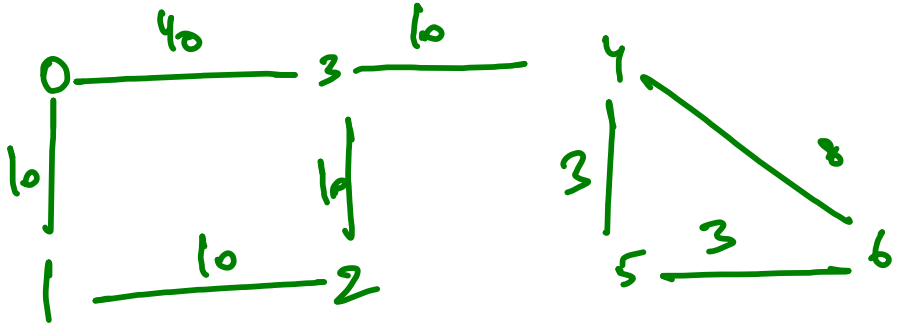

```java
public static class Edge{
    int src , dest , wt;

    Edge(int src, int dest , int wt){
        this.src = src;
        this.dest = dest;
        this.wt = wt;
    }
}
```

```
ArrayList<Edge> graph[] = new ArrayList[7];
```

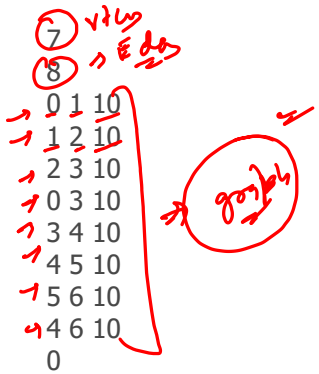


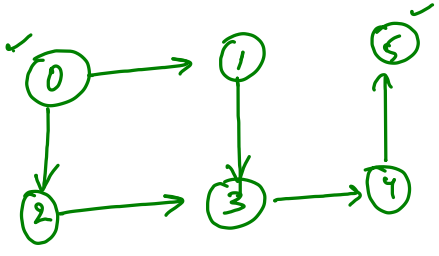

```
for(int idx = 0 ; idx < graph.length ; idx++){
    graph[idx] = new ArrayList();
}
```



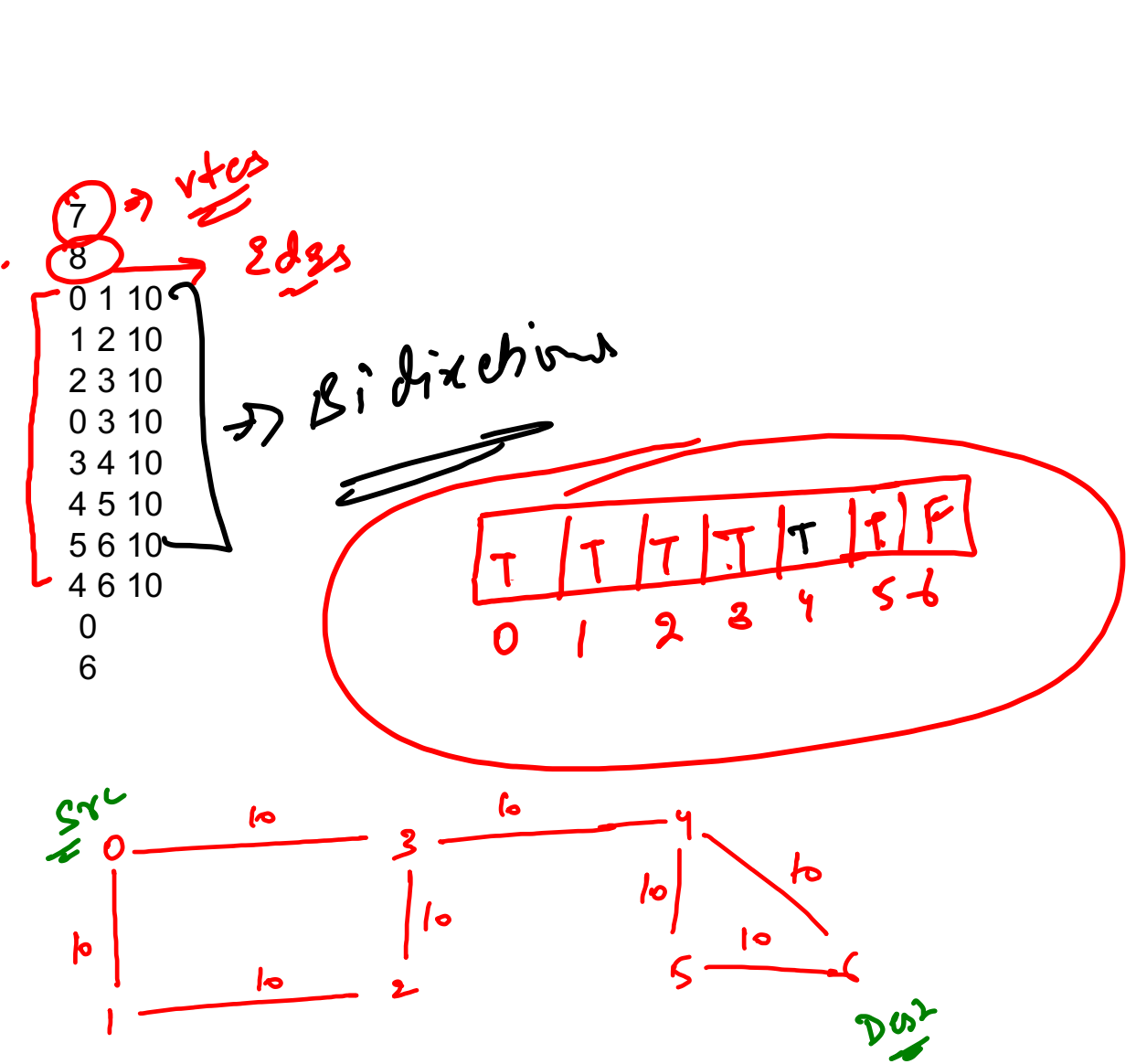


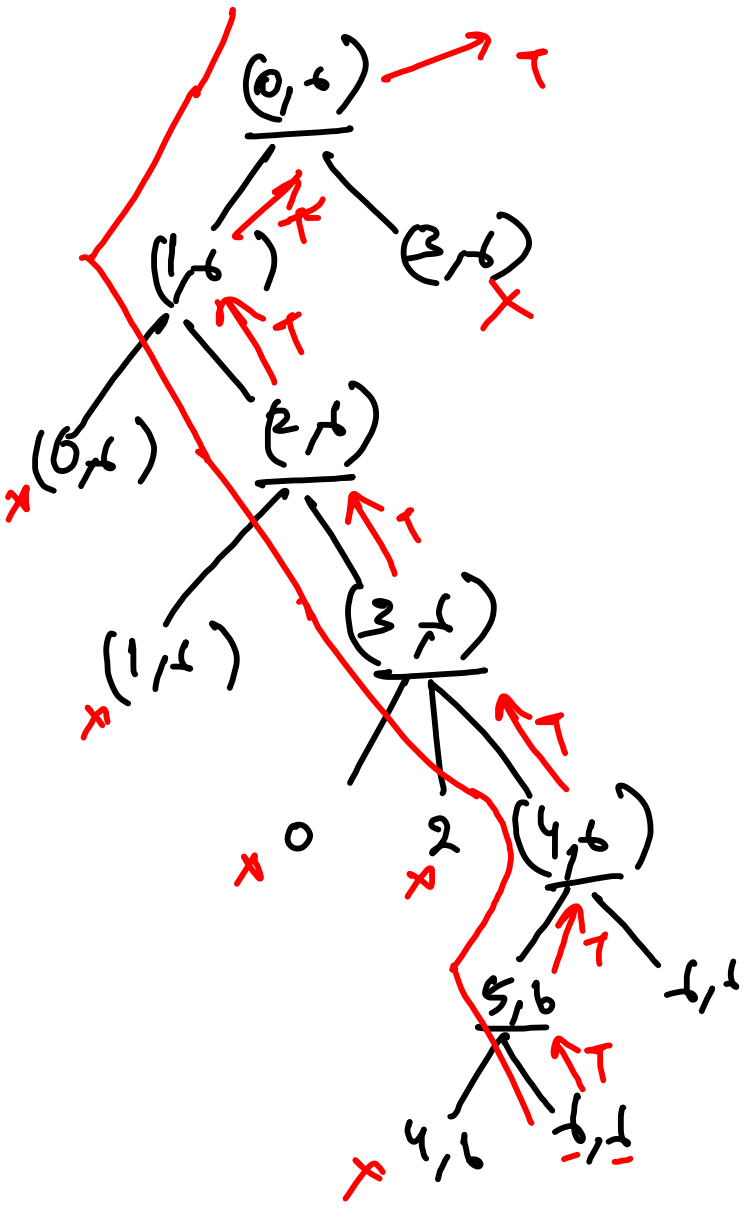

```
0-->[0 -> 1 @ 10, 0 -> 3 @ 40]
1-->[1 -> 0 @ 10, 1 -> 2 @ 10]
2-->[2 -> 1 @ 10, 2 -> 3 @ 10]
3-->[3 -> 0 @ 40, 3 -> 2 @ 10, 3 -> 4 @ 10]
4-->[4 -> 3 @ 10, 4 -> 5 @ 3, 4 -> 6 @ 8]
5-->[5 -> 6 @ 3, 5 -> 4 @ 3]
6-->[6 -> 5 @ 3, 6 -> 4 @ 8]
```

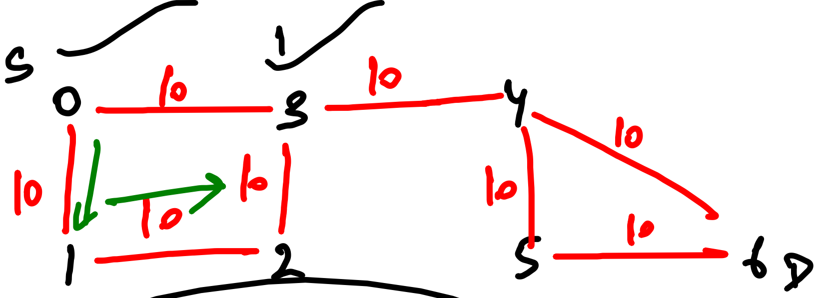

7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
0
6

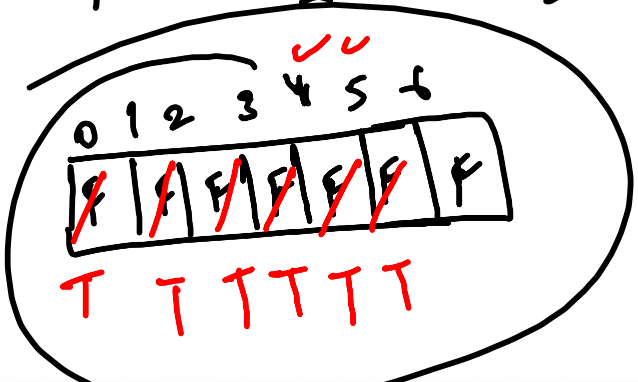

0
1
5
2
3
4

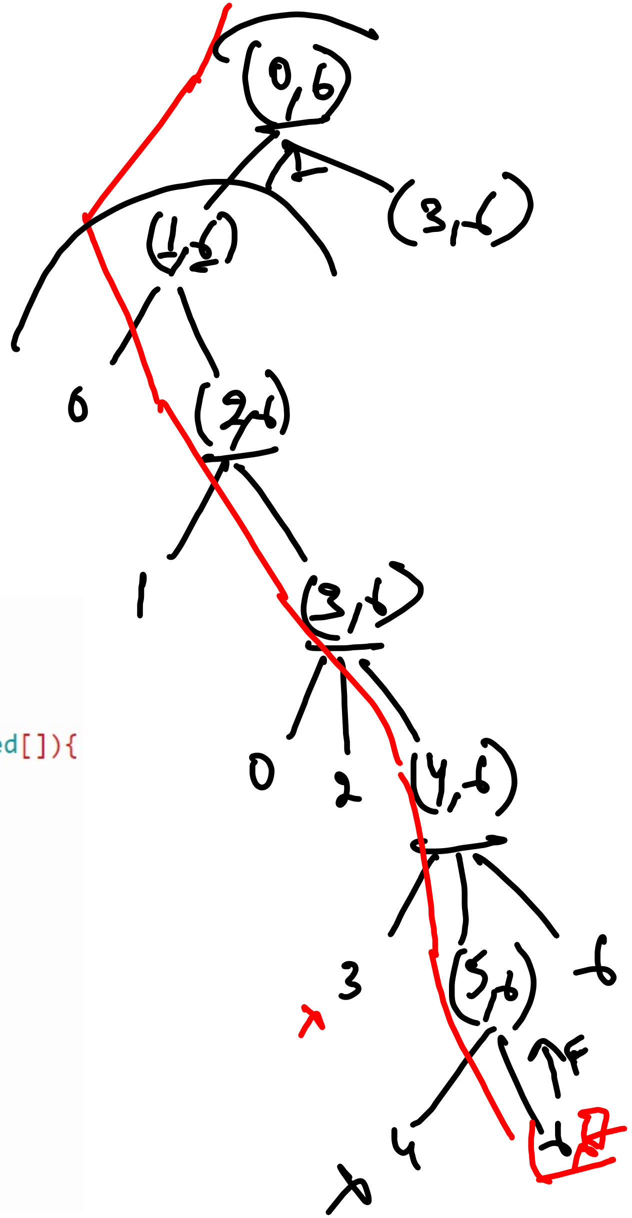

6


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
6
vtces
Edges
Bidirectional
T T T T T T F
0 1 2 3 4 5 6
Src
Dst
10

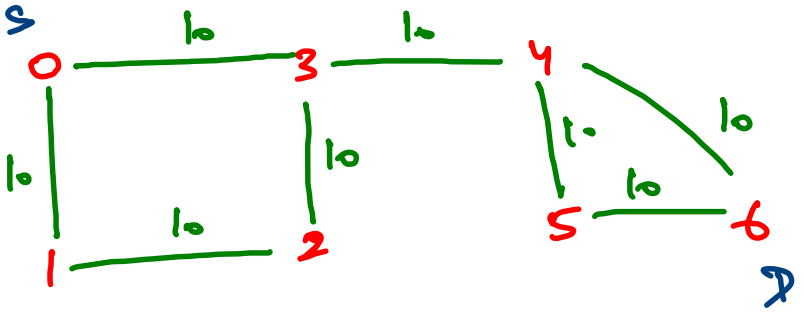

(0,6)
T
(1,6)
(3,6)
(0,6)
(2,6)
(1,1)
(3,6)
0
2
(4,6)
(5,6)
(4,6)
(5,5)

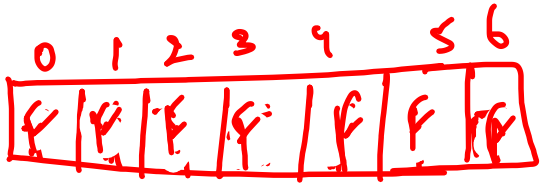






```java
    boolean res = hasPath(graph,src,dest,new boolean[vtces]);

    System.out.println(res);
}
public static boolean hasPath(ArrayList<Edge>[] graph,int vtx,int dest,boolean visited[]){
    // logic

    visited[vtx] = true;
    for(Edge e : graph[vtx]){
        if(visited[e.nbr] == false){
            boolean rres = hasPath(graph,e.nbr,dest,visited);
            if(rres){
                return true;
            }
        }
    }

    return false;
}
```



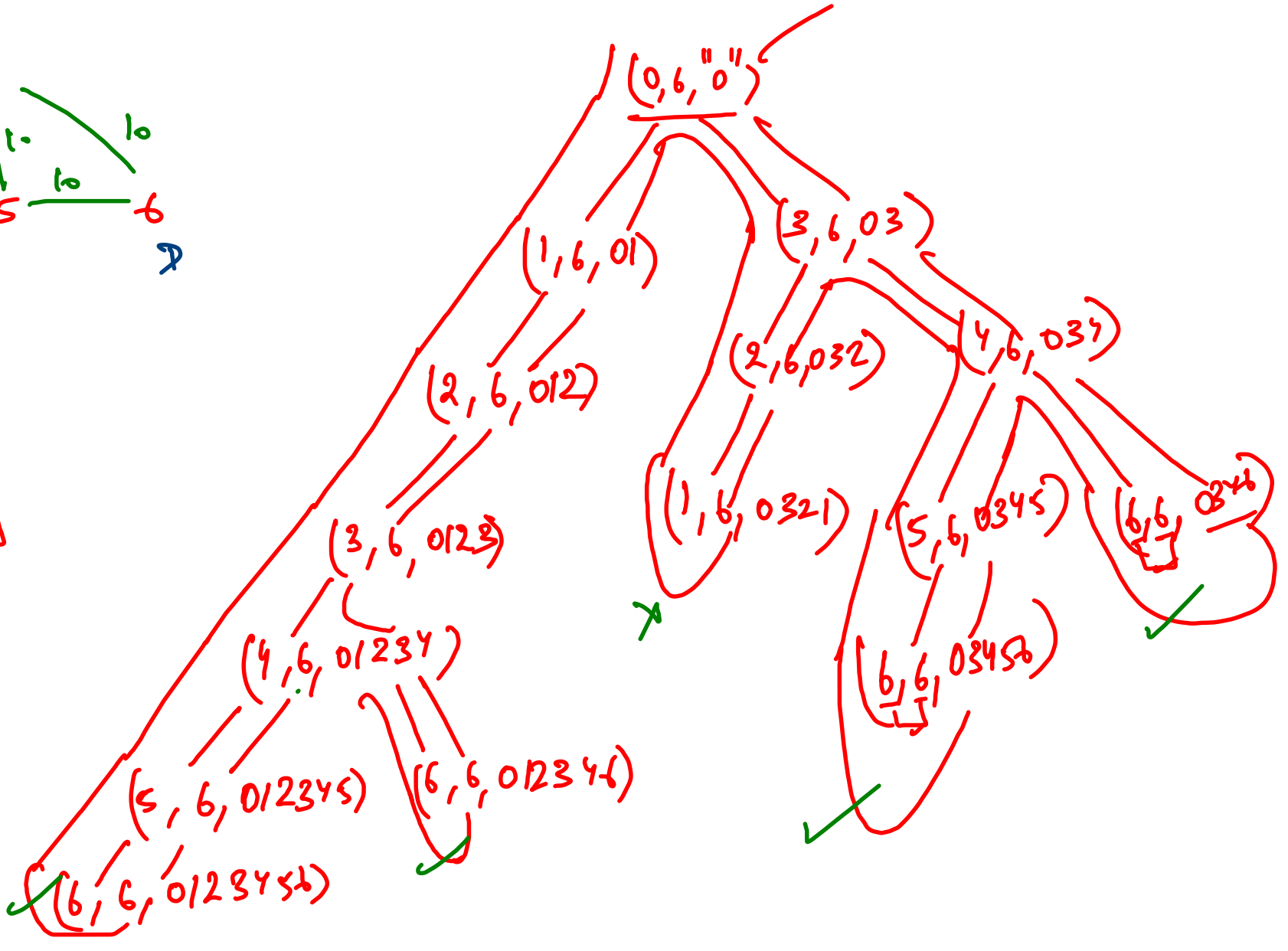


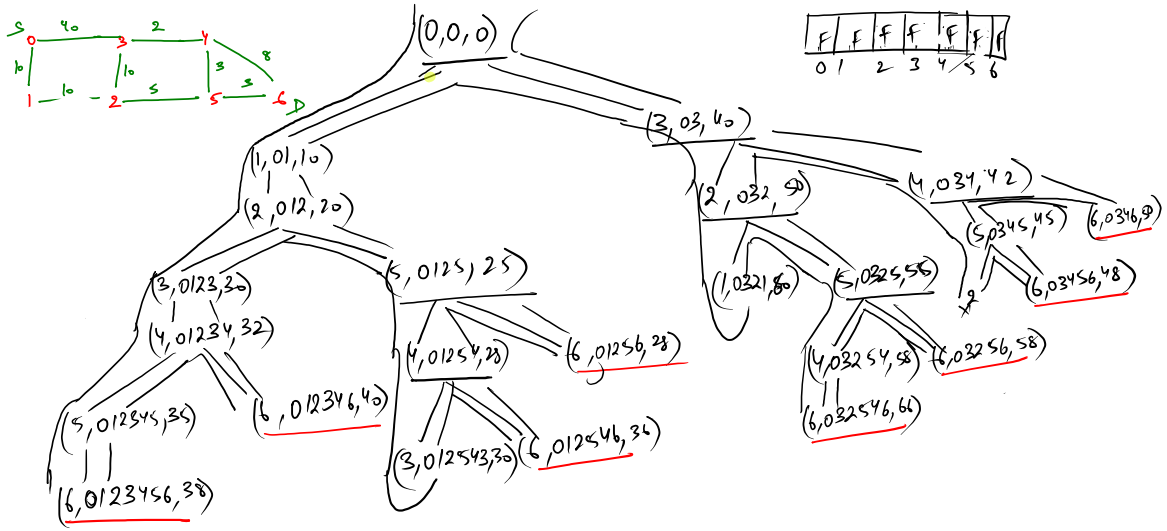
0 1 2 3 4 5 6

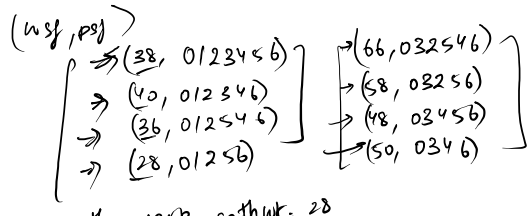
0 1 2 3 4 6

0 3 4 5 6

0 3 4 6





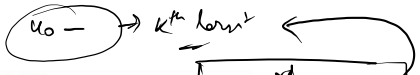


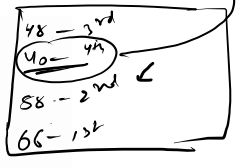
spath = 01256 spathwt = 28
lpath = 0123456 lpathwt = 40
cpath = 012546 cpathwt = 36
fpath = 01256 fpathwt = 28

```
static class Pair implements Comparable<Pair> {
    int wsf;
    String psf;

    Pair(int wsf, String psf){
        this.wsf = wsf;
        this.psf = psf;
    }

    public int compareTo(Pair o){
        return this.wsf - o.wsf;
    }
}
```

just larger → minimum among larger

just smaller → maximum among smaller






```
if(vtx == dest){
    if(wsf < spathwt){
        spathwt = wsf;
        spath = psf;
    }

    if(wsf > lpathwt){
        lpathwt = wsf;
        lpath = psf;
    }

    if(pq.size() < k){
        pq.add(new Pair(wsf,psf));
    }else{
        if(pq.peek().wsf < wsf){
            pq.remove();
            pq.add(new Pair(wsf,psf));
        }
    }

    if(wsf > criteria && wsf < cpathwt){
        cpathwt = wsf;
        cpath = psf;
    }

    if(wsf < criteria && wsf > fpathwt){
        fpathwt = wsf;
        fpath = psf;
    }

    return;
}
```