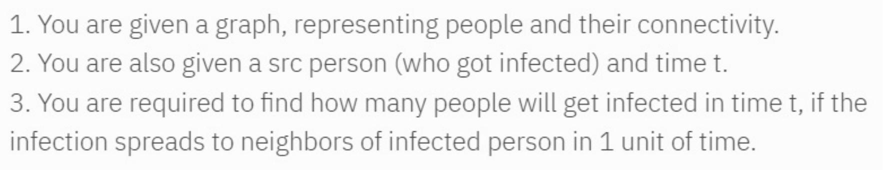
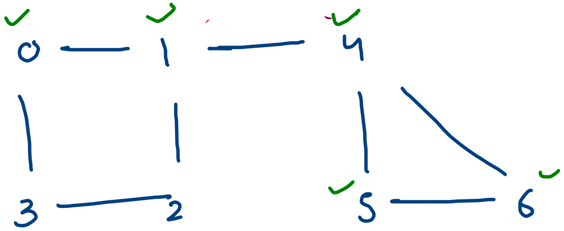
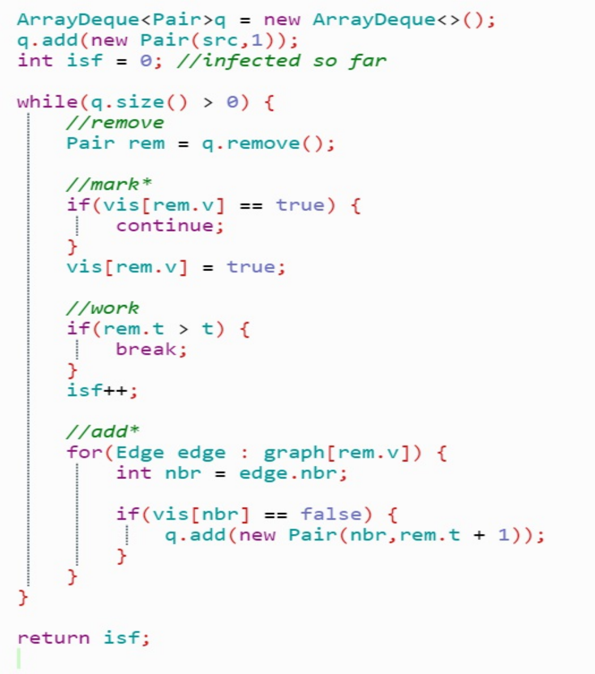
Spread Of Infection



1. You are given a graph, representing people and their connectivity.
2. You are also given a src person (who got infected) and time t.
3. You are required to find how many people will get infected in time t, if the infection spreads to neighbors of infected person in 1 unit of time.



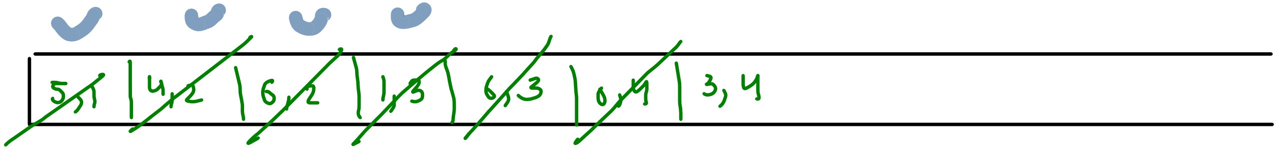


```
ArrayDeque<Pair>q = new ArrayDeque<>();
q.add(new Pair(src,1));
int isf = 0; //infected so far

while(q.size() > 0) {
    //remove
    Pair rem = q.remove();

    //mark*
    if(vis[rem.v] == true) {
        continue;
    }
    vis[rem.v] = true;

    //work
    if(rem.t > t) {
        break;
    }
    isf++;

    //add*
    for(Edge edge : graph[rem.v]) {
        int nbr = edge.nbr;

        if(vis[nbr] == false) {
            q.add(new Pair(nbr,rem.t + 1));
        }
    }
}

return isf;
```

src = 5 (5 is already infected at t = 1)

t = 3

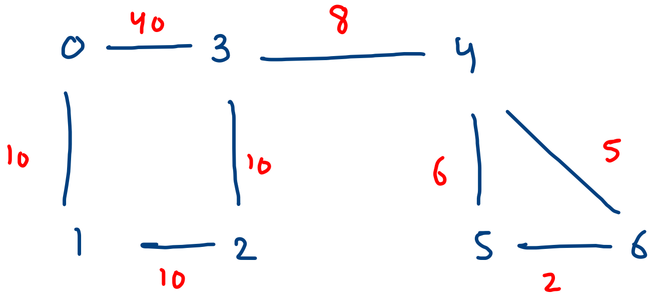
isf = 1 + 1 + 1 + 1



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

Shortest Path In Weights

(i) single src all dest shortest path
(edge wt)



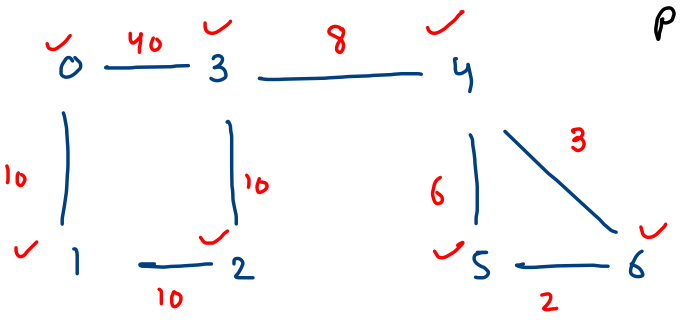
Dijkstra → single traversal

BFS
└ dont use queue, priority queue

premise :

what ?

Greedy based



src = 0

```
Pair {
    int v;
    int wt;
    String psf;
}
```

0 -> 0 @ 0

1 -> 01 @ 10

2 -> 012 @ 20

3 -> 0123 @ 30

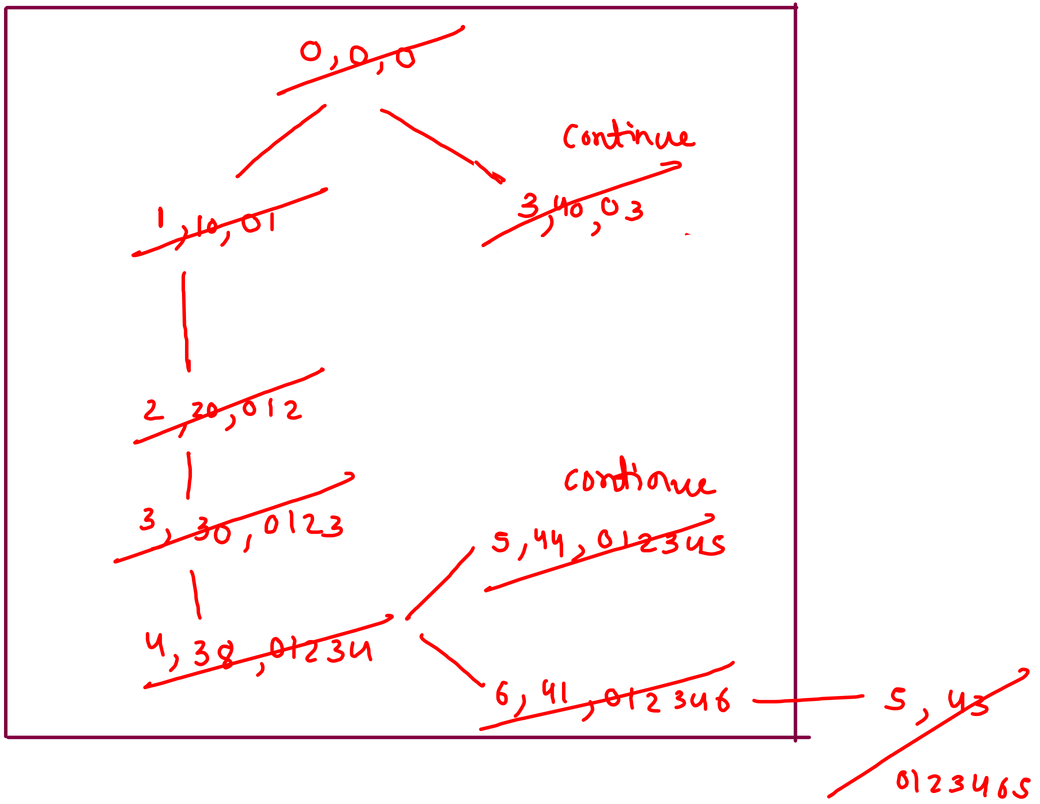
4 -> 01234 @ 38

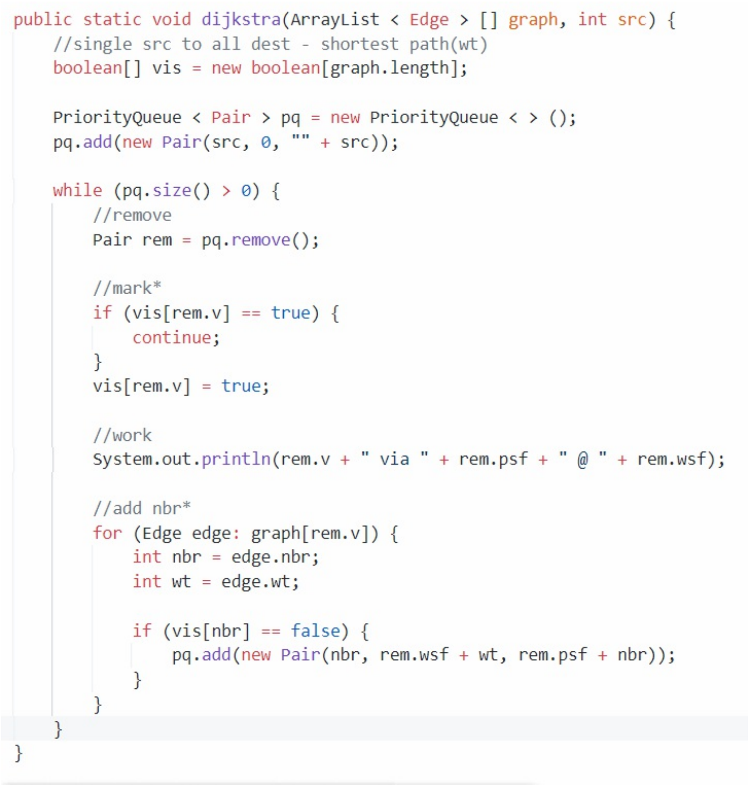
6 -> 012346 @ 41

5 -> 0123465 @ 43

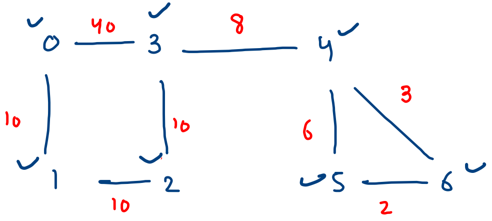
minheap

(wt)

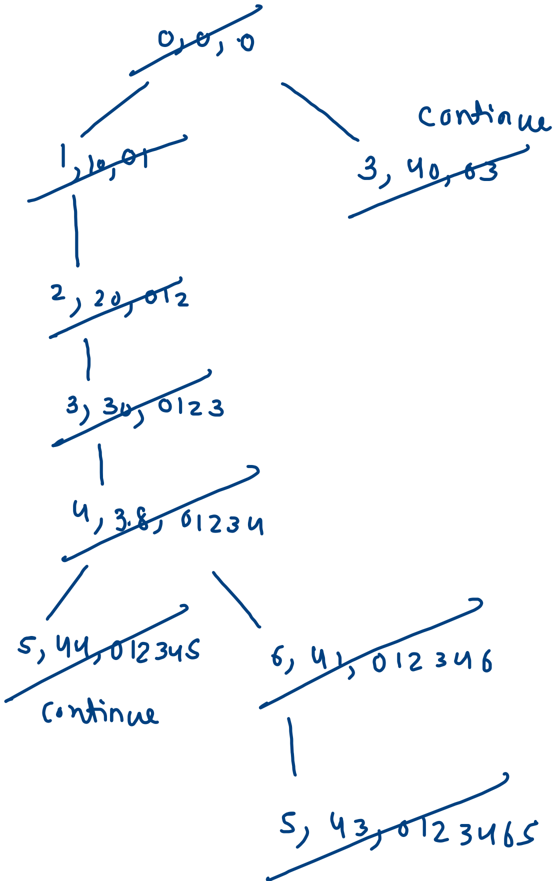




```java
public static void dijkstra(ArrayList < Edge > [] graph, int src) {
    //single src to all dest - shortest path(wt)
    boolean[] vis = new boolean[graph.length];

    PriorityQueue < Pair > pq = new PriorityQueue < > ();
    pq.add(new Pair(src, 0, "" + src));

    while (pq.size() > 0) {
        //remove
        Pair rem = pq.remove();

        //mark*
        if (vis[rem.v] == true) {
            continue;
        }
        vis[rem.v] = true;

        //work
        System.out.println(rem.v + " via " + rem.psf + " @ " + rem.wsf);

        //add nbr*
        for (Edge edge: graph[rem.v]) {
            int nbr = edge.nbr;
            int wt = edge.wt;

            if (vis[nbr] == false) {
                pq.add(new Pair(nbr, rem.wsf + wt, rem.psf + nbr));
            }
        }
    }
}
```



src = 0



v, wsf, psf

0 via 0 @ 0

1 via 01 @ 10

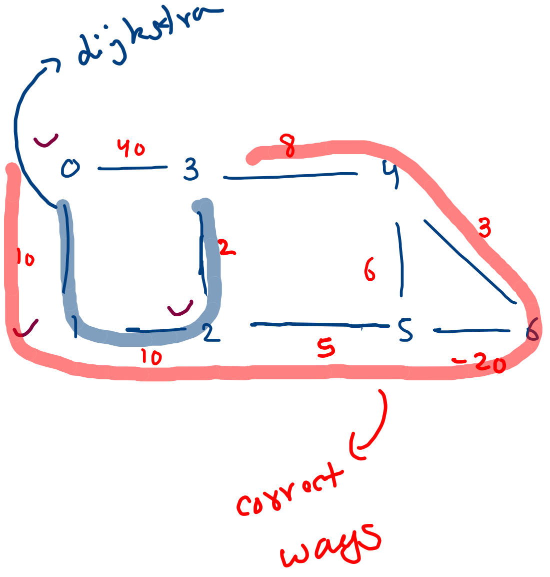
2 via 012 @ 20

3 via 0123 @ 30

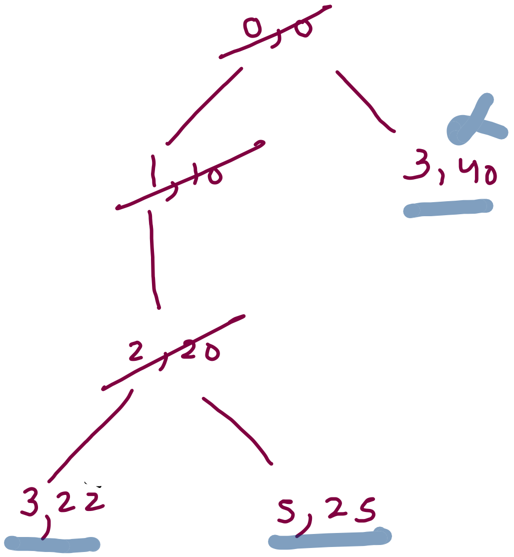
4 via 01234 @ 38

6 via 012346 @ 41

5 via 0123465 @ 43



src = 0



v, wsf

22.

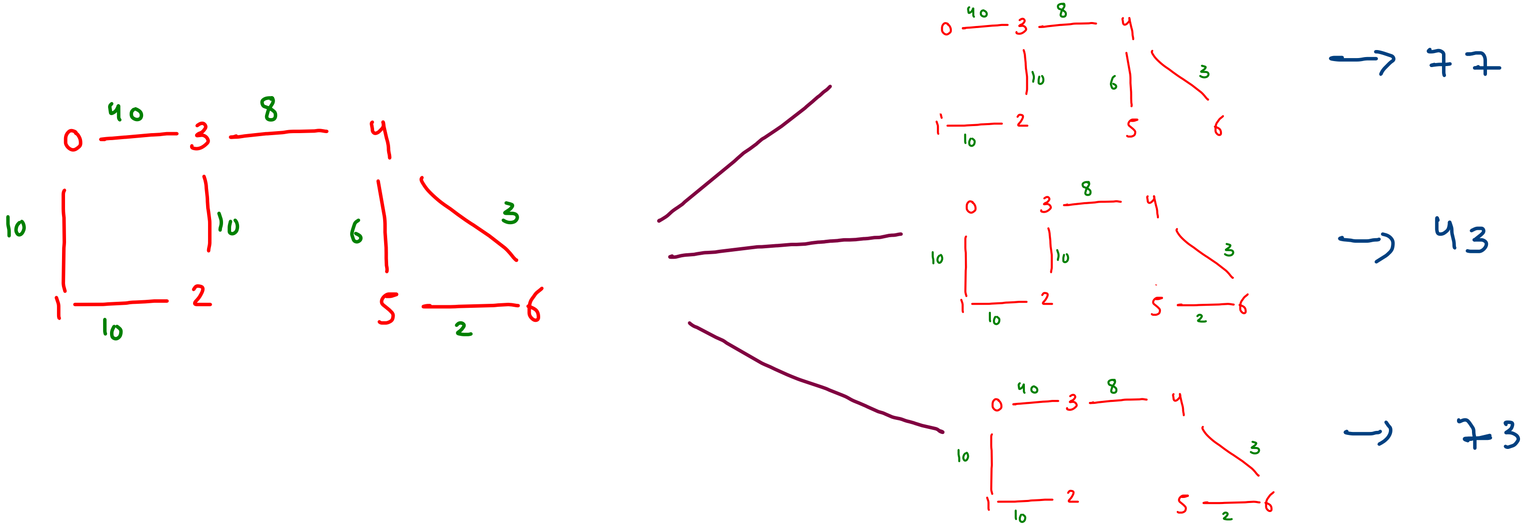
25 + ...

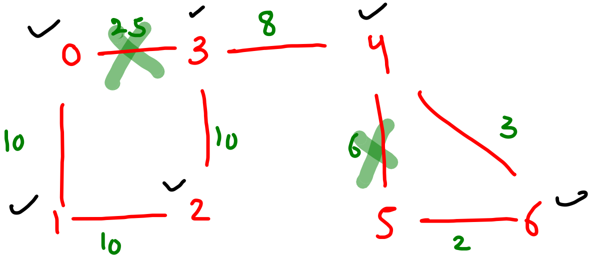
Note: limitation of dijkstra, fails on -ve edge wt.

Prims

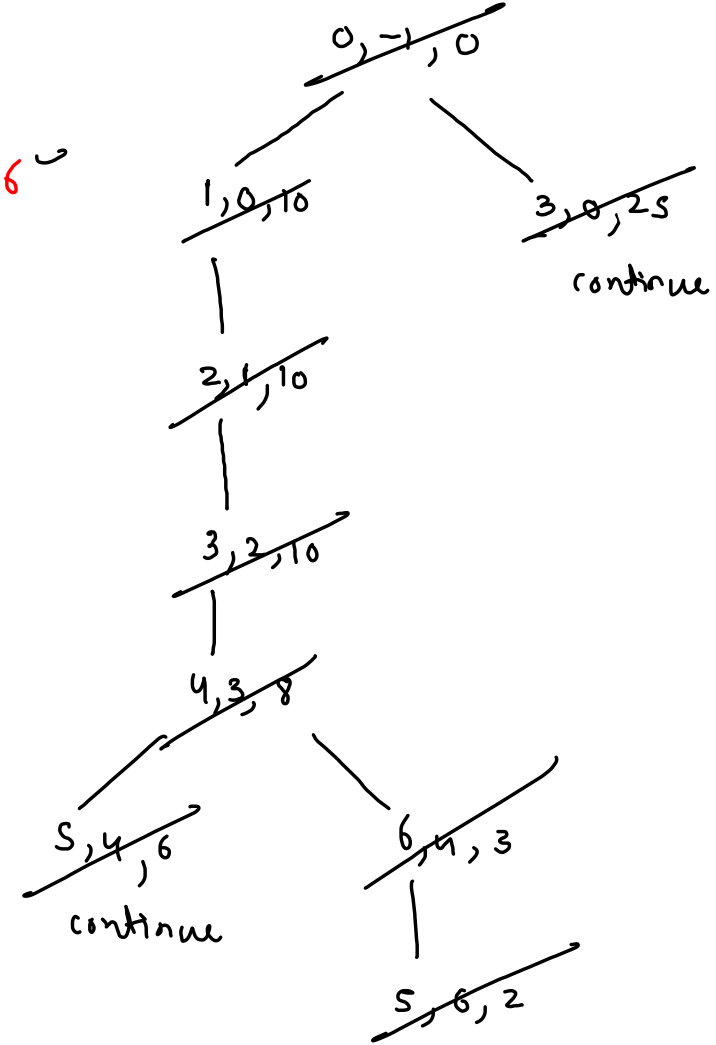
MST : min spanning tree

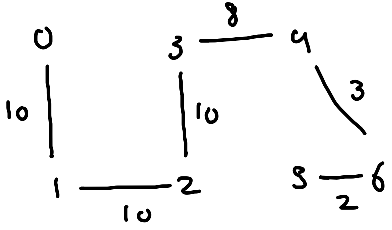
tree : acyclic, connected graph

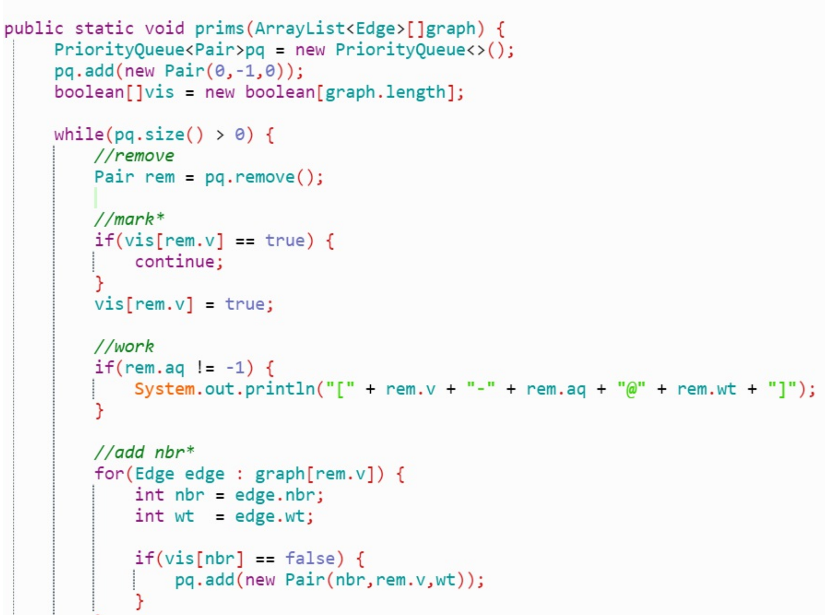




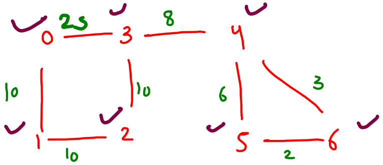
```
Pair {

    int  v;
    int  av;
    int  wt;

}
```

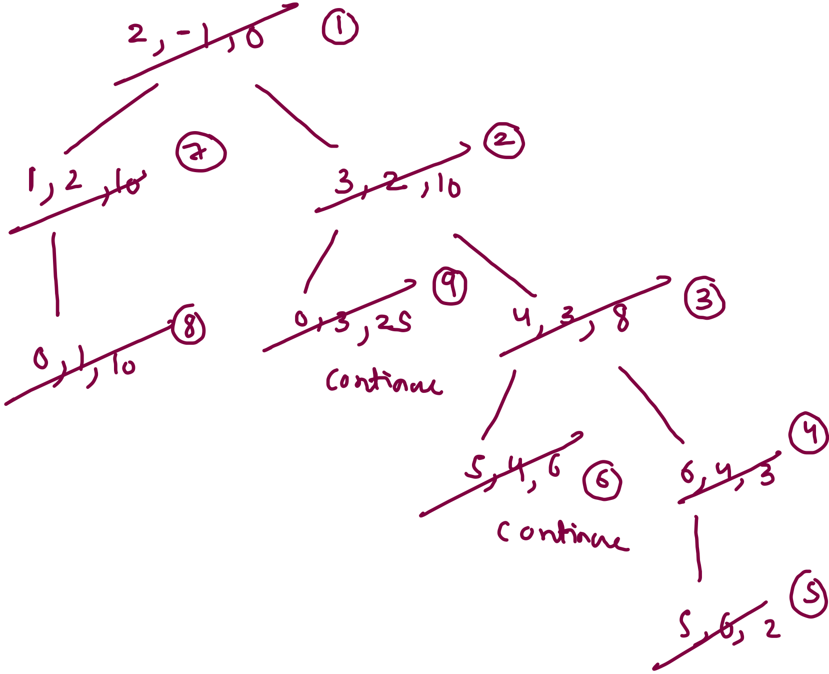


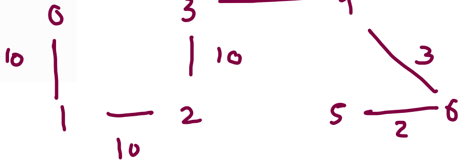




```
public static void prims(ArrayList<Edge>[]graph) {
    PriorityQueue<Pair>pq = new PriorityQueue<>();
    pq.add(new Pair(0,-1,0));
    boolean[]vis = new boolean[graph.length];

    while(pq.size() > 0) {
        //remove
        Pair rem = pq.remove();

        //mark*
        if(vis[rem.v] == true) {
            continue;
        }
        vis[rem.v] = true;

        //work
        if(rem.aq != -1) {
            System.out.println("[" + rem.v + "-" + rem.aq + "@" + rem.wt + "]");
        }

        //add nbr*
        for(Edge edge : graph[rem.v]) {
            int nbr = edge.nbr;
            int wt  = edge.wt;

            if(vis[nbr] == false) {
                pq.add(new Pair(nbr,rem.v,wt));
            }
        }
    }
}
```


0
25
3
8
4
10
10
6
3
1
10
2
5
2
6


~~2, -1, 0~~
1
~~1, 2, 10~~
7
~~3, 2, 10~~
2
~~0, 1, 10~~
8
~~0, 3, 25~~
9
continue
~~4, 3, 8~~
3
~~5, 4, 6~~
6
continue
~~6, 4, 3~~
4
~~5, 6, 2~~
5


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