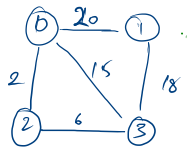
allowed undirected graph

SPANNING TREE :-

Connected Graph

Graph

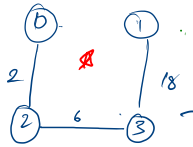


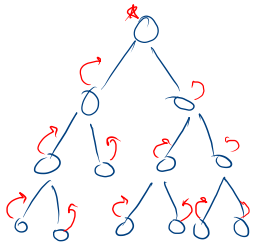
N vtces

↳ Max Edges : $\frac{N(N-1)}{2}$

N-1 Edges

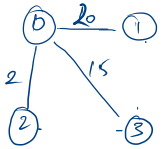
Tree

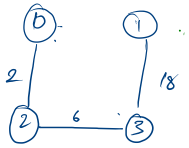


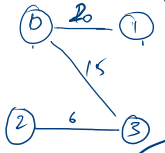


N nodes

↳ n-1 edges







on the basis of Total Edge wt.

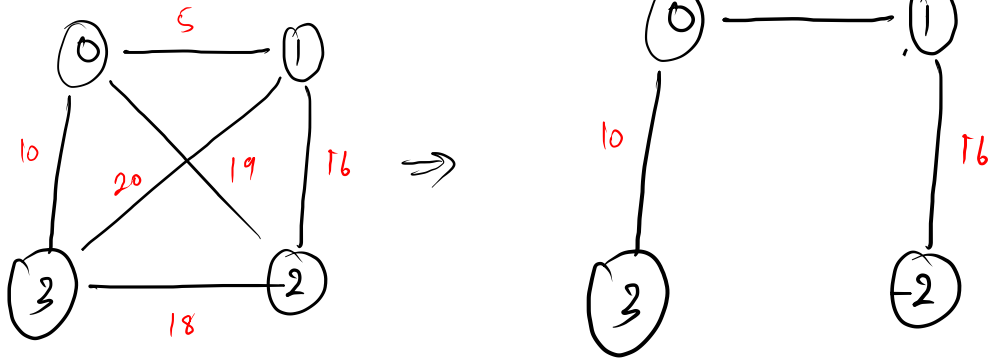
MINIMUM SPANNING TREE

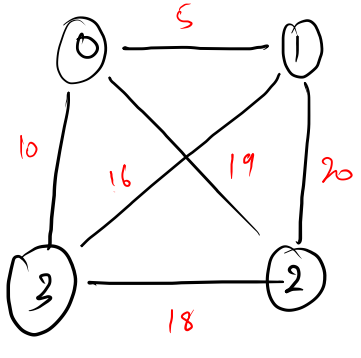
↳ Undirected + weighted edges

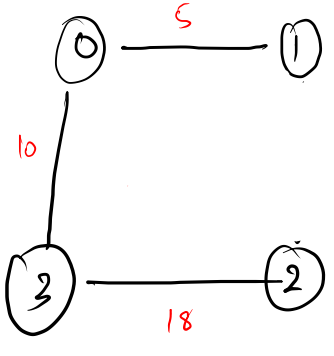
MST
↳ Prim's

; Level 1

1. N-1 Edges
2. Minimum wt. (Total Edges)
3. Connected



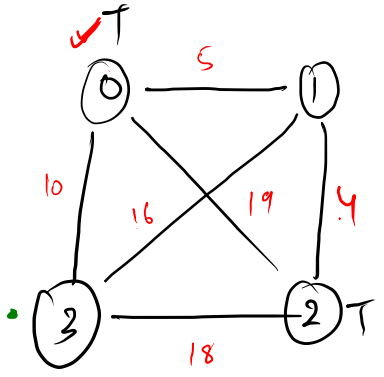




V + Vs = 4

nEdges allowed ⇒ 3

nEdges = 6



{3,18}
~~{3,10}~~
{2,19}
{3,16}

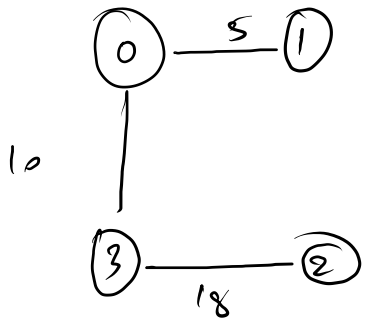
{V,E}

{2, 20}
~~{3, 18}~~
~~{2, 19}~~
~~{3, 10}~~

1. N-1 Edges
2. Minimum Wt
3. Connected

MST

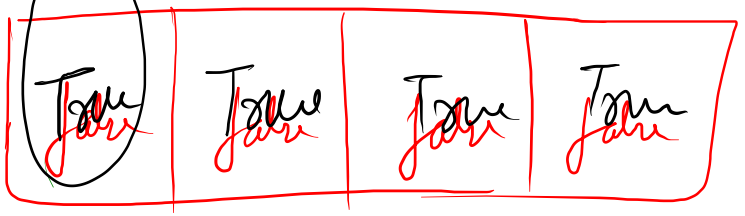
Concept

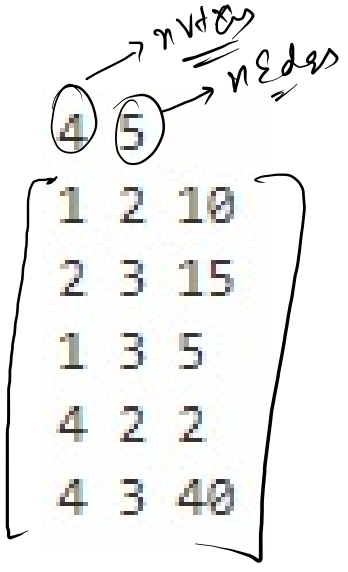


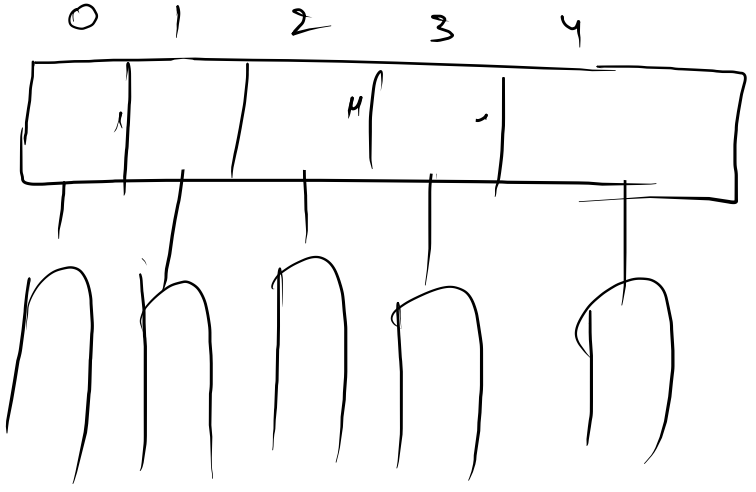
VIS →

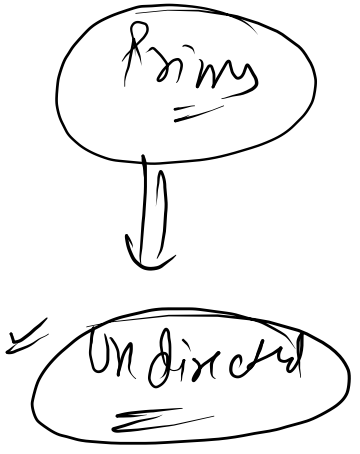
N-1 Edges

0 1 2 3

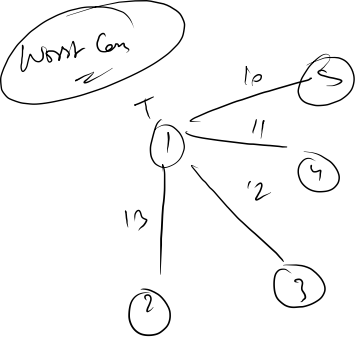


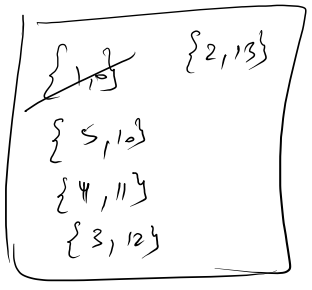


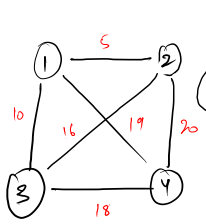




ArrayList<Edge>[] graph = new ArrayList[5];

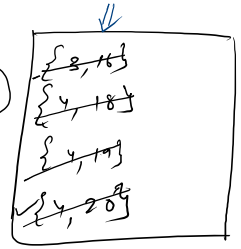






ladd ⇒ log E

Eadd = ElogE



E

Time Complexity ⇒

Time = O(ElogE)

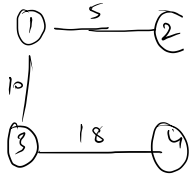
maxEdges = $\frac{V(V-1)}{2}$ ∝ V^2

$O(V^2 \log V^2)$

$O(2V^2 \log V)$

$O(E \log V)$

$b \log a^b$



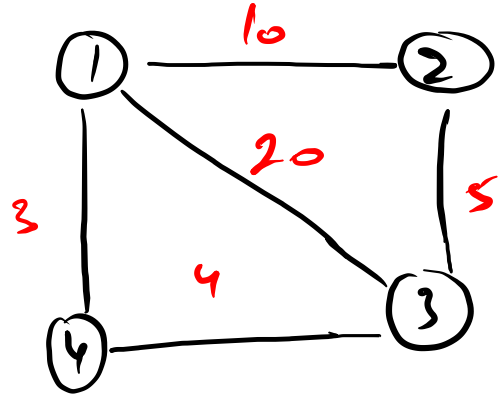
ans = 0 + 5 + 10 + 18 ⇒ 33

```java
public static int solve(ArrayList<Edge>[] graph){
    int nvtces = graph.length;
    PriorityQueue<Pair> pq = new PriorityQueue<>();
    pq.add(new Pair(1,0));
    boolean[] vis = new boolean[nvtces];
    int ans = 0;

    while(pq.size() > 0){
        Pair rem = pq.remove();
        if(vis[rem.vtx] == true){
            continue;
        }

        vis[rem.vtx] = true;
        ans += rem.wt;

        for(Edge e : graph[rem.vtx]){
            if(vis[e.nbr] == false){
                pq.add(new Pair(e.nbr,e.wt));
            }
        }
    }
    return ans;
}
```

```
Input :{{0, 1, 2, 3, 4},
        {1, 0, 5, 0, 7},
        {2, 5, 0, 6, 0},
        {3, 0, 6, 0, 0},
        {4, 7, 0, 0, 0}};
```

for( )

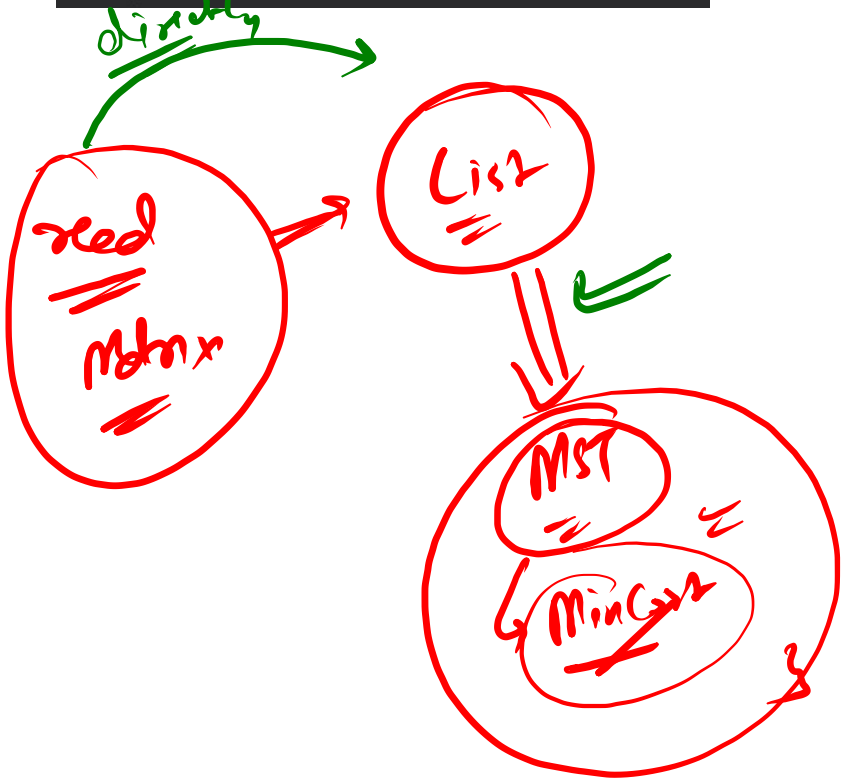
4 → cities



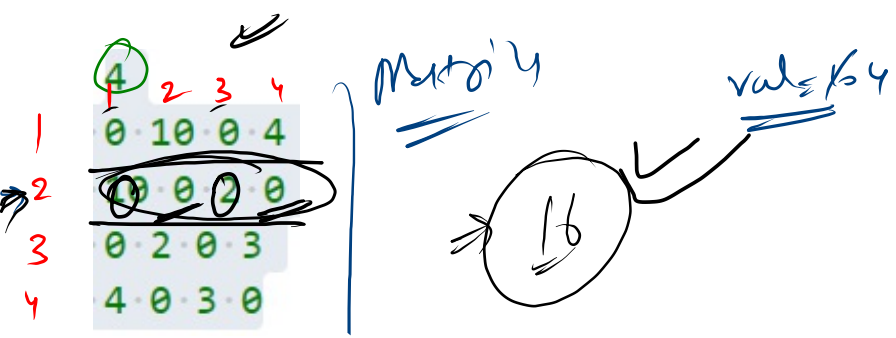
MST

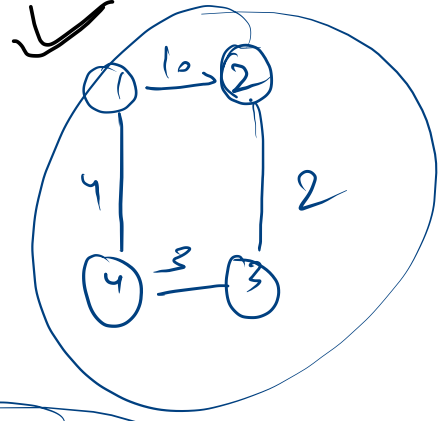
Minimum set of the connected tree

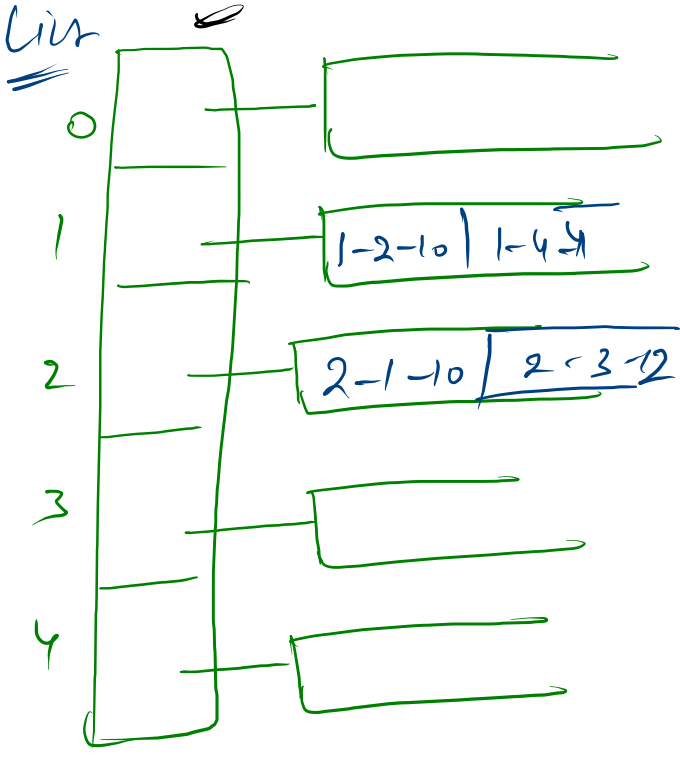
```
Input : {{0, 1, 1, 100, 0, 0},
         {1, 0, 1, 0, 0, 0},
         {1, 1, 0, 0, 0, 0},
         {100, 0, 0, 0, 2, 2},
         {0, 0, 0, 2, 0, 2},
         {0, 0, 0, 2, 2, 0}};
Output : 106
```

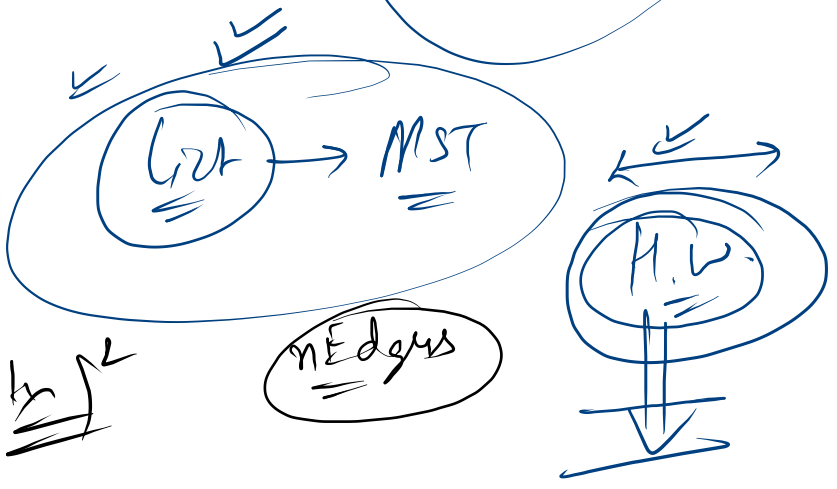



```
for ( int i=0 ; ———— ){
    for (int j = 0 ; ———— ){
        inp ←———
        if(inp >0) {
            ————
        }
    }
```







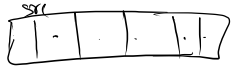



```java
Scanner scn = new Scanner(System.in);
int nvtces = scn.nextInt();

ArrayList<Edge>[] graph = new ArrayList[nvtces+1];
for(int i = 0 ; i <= nvtces ; i++){
    graph[i] = new ArrayList<>();
}

// input
for(int  r = 1; r <= nvtces ; r++){
    for(int c = 1; c <= nvtces ; c++){
        int vl = scn.nextInt();
        if(vl > 0){
            graph[r].add(new Edge(r,c,vl));
        }
    }
}

for(int i = 1; i <= nvtces ; i++){
    System.out.println(i+" --> " +graph[i]);
}
```

```
1 --> [src : 1 nbr : 2 wt : 10, src : 1 nbr : 4 wt : 4]
2 --> [src : 2 nbr : 1 wt : 10, src : 2 nbr : 3 wt : 2]
3 --> [src : 3 nbr : 2 wt : 2, src : 3 nbr : 4 wt : 3]
4 --> [src : 4 nbr : 1 wt : 4, src : 4 nbr : 3 wt : 3]
```

DIJKSTRA's Algo

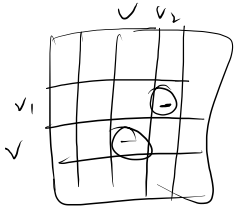
Single src/point → Minimum path to all other vtcs



directed + undirected

Shortest path algorithm

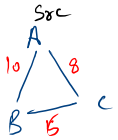
- Dijkstra's algo (single point) ⇒ $E \log E$, (+ve, 0) → Edge wts
- Bellmanford ⇒ $(E \cdot V)$, (-ve, +ve, 0) → Edge wts
- floyd ⇒ V^3 (multiple src)

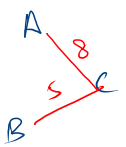
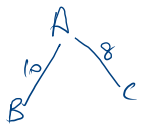


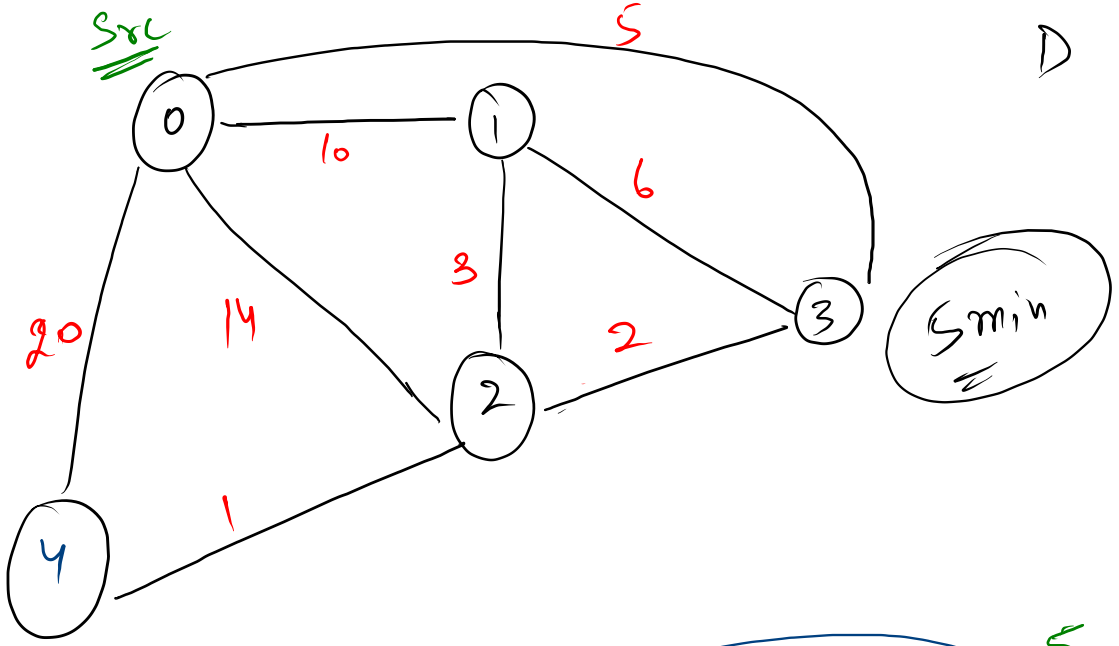
Prims :→ Undirected + weighted

Edge wt. allowed +ve, -ve, 0

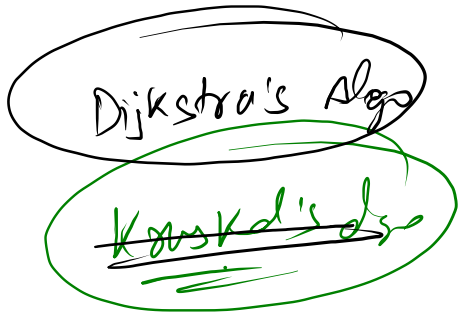
MST

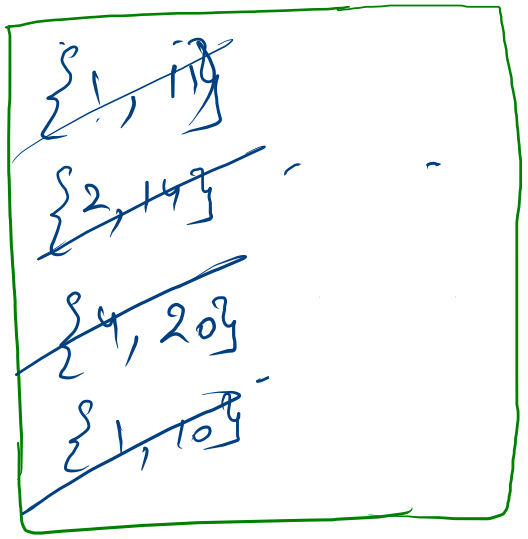


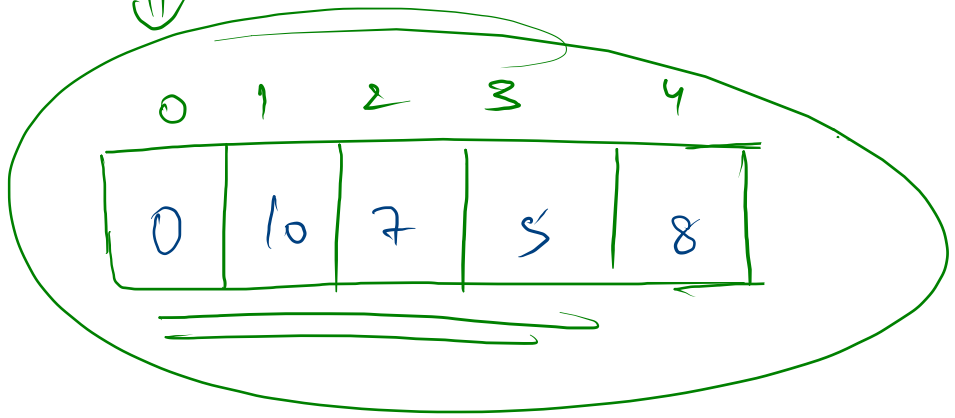
Prims	Dijkstra's
MST ⇓ 	

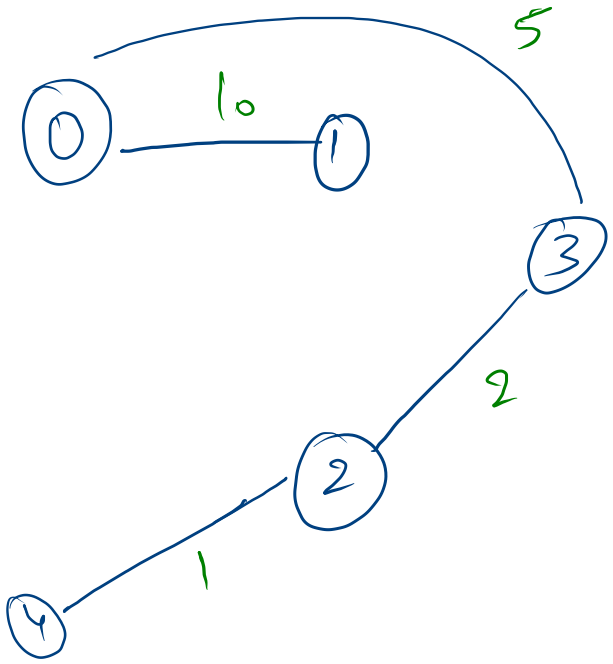

Src
5
0
10
1
6
3
20
14
2
3
2
4
1
Smin

D


Dijkstra's Algo
Kruskal's algo


{1, 10}
{2, 14}
{4, 20}
{1, 10}


0 1 2 3 4
0 10 7 5 8


5
0
10
1
3
2
2
1
4

For Input:

```
2 1
0 1 9
0
```

Your Output is:

```
[[[1, 9]], [[0, 9]]]
0 0
```

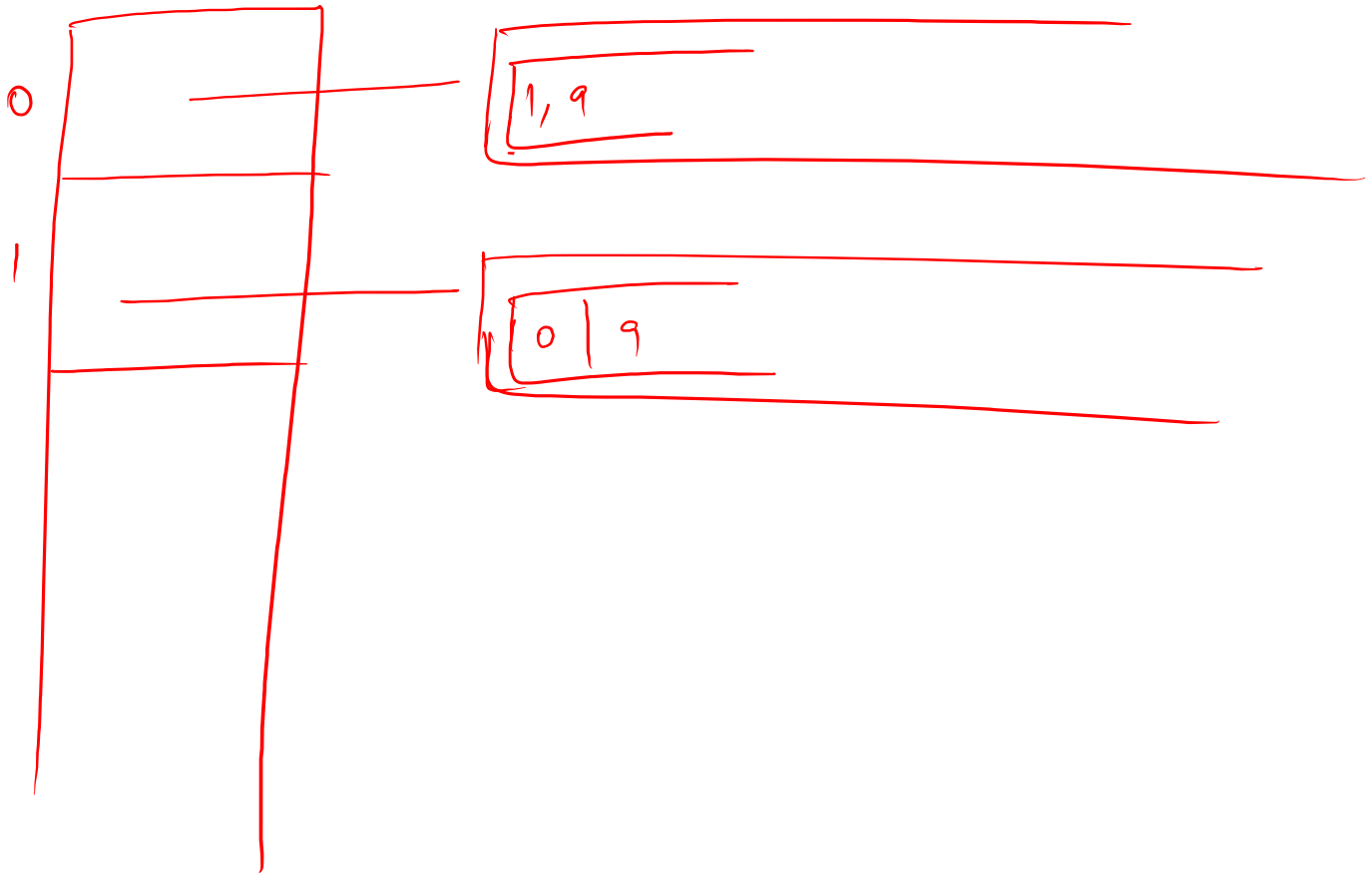
adj

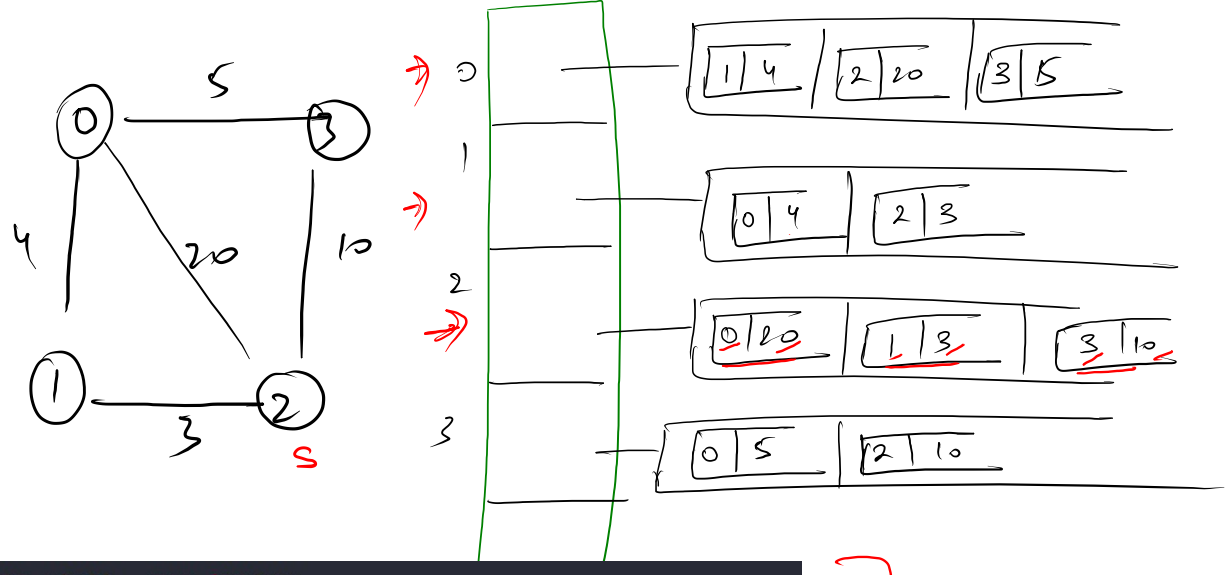
For Input:

V E
2 1
0 1 9
0

Your Output is:

[[[1, 9]], [[0, 9]]]
0 0





vis =

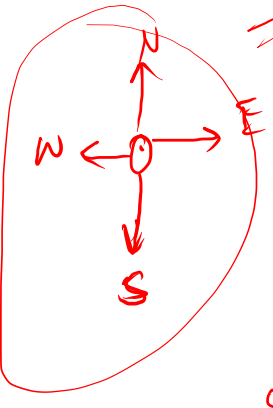
0	1	2	3
~~7~~	~~3~~	~~0~~	-1

~~{0,4}~~

{0,20}

{3,10

```
int vis[] = new int[V];
Arrays.fill(vis,-1);
PriorityQueue<Pair> pq = new PriorityQueue<>();

pq.add(new Pair(S,0));

while(pq.size() > 0){
    Pair rem = pq.remove();
    if(vis[rem.vtx] != -1){
        continue;
    }

    vis[rem.vtx] = rem.wsf;

    ArrayList<ArrayList<Integer>> edges = adj.get(rem.vtx);
    for(ArrayList<Integer> edge : edges){
        int nbr = edge.get(0) , wt = edge.get(1);
        if(vis[nbr] == -1){
            pq.add(new Pair(nbr,rem.wsf + wt));
        }
    }
}
return vis;
```

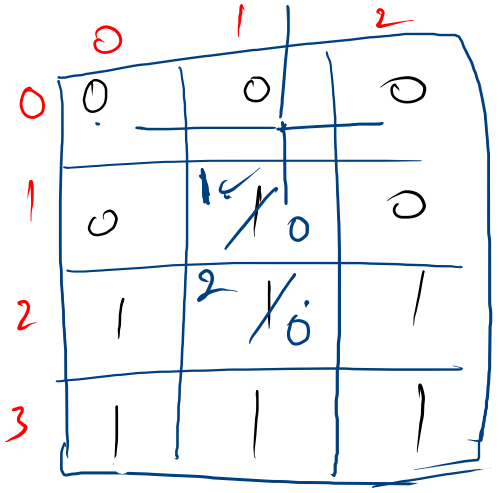


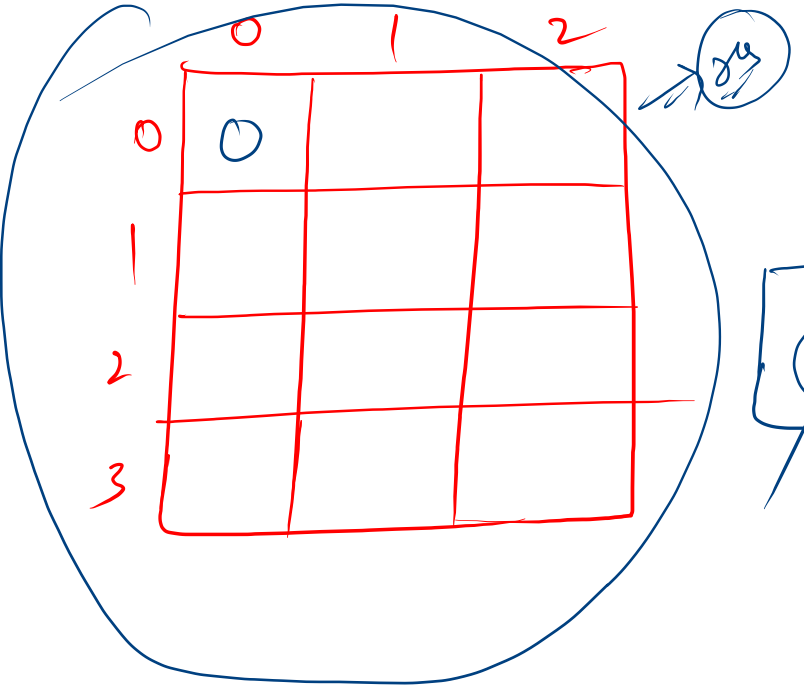
(r, c, d)

	0	1	2
0	0	0	0
1	0	~~1~~ 0	0
2	~~1~~ 0	~~1~~ 0	~~1~~ 0
3	~~1~~ 0	1	1

- ~~{0,0,0}~~
- ~~{0,1,0}~~
- ~~{0,2,0}~~
- ~~{1,0,0}~~
- ~~{1,2,0}~~
- ~~{1,1,1}~~
- ~~{2,0,1}~~
- {2,2,1}
- {2,1,2}
- {3,0,2}

	0	1	2
0	0	0	0
1	0	1	0
2	1		
3			





```java
public int[][] updateMatrix(int[][] mat) {
    Queue<Pair> queue = new ArrayDeque<>();

    int res[][] = new int[mat.length][mat[0].length];

    for(int r = 0 ; r < mat.length ; r++){
        for(int c = 0 ; c < mat[0].length ; c++){
            if(mat[r][c] == 0){
                queue.add(new Pair(r,c,0));
            }
        }
    }

    int dir[][] = {{-1,0},{0,1},{+1,0},{0,-1}};
    while(queue.size() > 0){
        Pair rem = queue.remove();

        res[rem.r][rem.c] = rem.d;

        for(int d = 0 ; d < 4 ; d++){
            int rdash = rem.r + dir[d][0] , cdash = rem.c + dir[d][1];
            if(rdash < 0 || cdash < 0 || rdash >= mat.length || cdash >= mat[0].length || mat[rdash][cdash] != 1){
                continue;
            }
            mat[rdash][cdash] = 0;
            queue.add(new Pair(rdash,cdash,rem.d+1));
        }
    }
    return res;
}
```