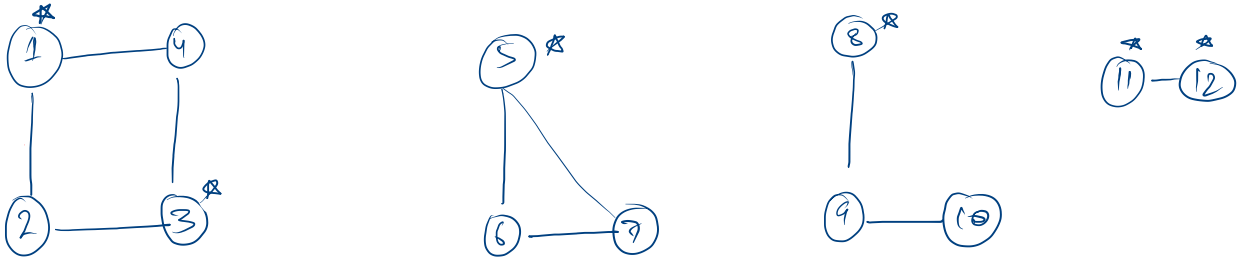
Minimize Malware SPREAD

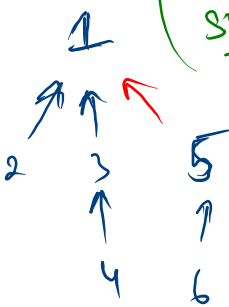


~~Infected~~ → {1, 3, 5, 6, 11, 12}

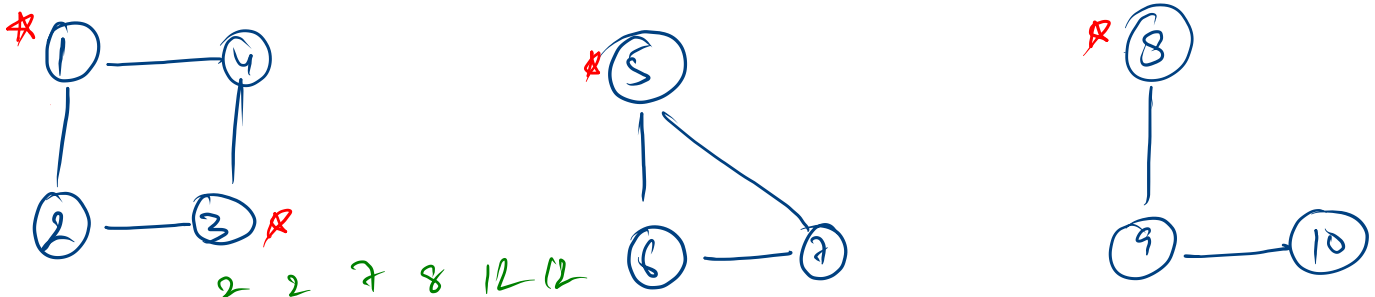
Power → to disinfect a Person

(Union by Rank (int) ⟺ Union by Size)

par
size → default = 1




```
public static void Union(int x, int y){
    int lx = find(x);
    int ly = find(y);
    if(lx != ly){
        if(size[lx] >= size[ly]){
            par[ly] = lx;
            size[lx] += size[ly];
        } else if(size[lx] < size[ly]){
            par[lx] = ly;
            size[ly] += size[lx];
        }
    }
}
```



2 2 7 8 12 12

Injected ⇒ { 1, 3, 5, 6, 11, 12 }

Rep + Size

DSU

Size: 4

Size ⇒ 3

Size ⇒ 3

Size ⇒ 2

Leader	Injected Count
~~2~~	~~2~~
7	1
8	1
~~12~~	~~2~~

Injected vtx = 5

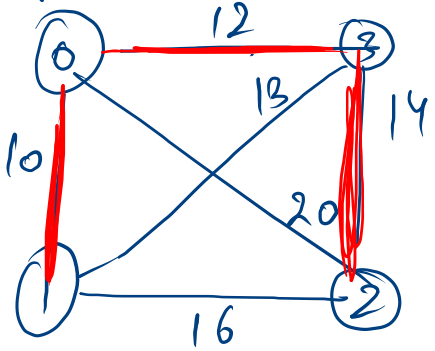
Count = 3

5 ✓

MST → Prim's → MST USING PQ

Src → Graph

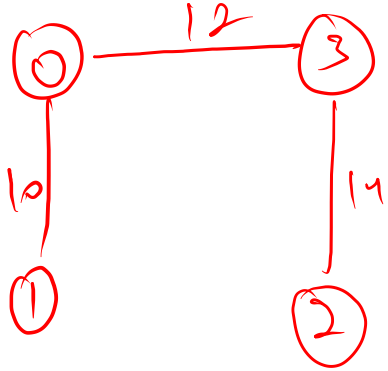
KRUSKAL → MST using DSU



u	v	wt
0	3	12
1	3	13
0	2	20
3	2	14
1	2	16
0	1	10

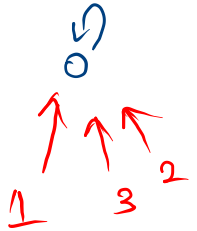
⇒

u	v	wt
0	1	10
0	3	12
1	3	13 ✗
3	2	14
1	2	16 ✗
0	2	20



Cost ⇒ 10 + 12 + 14

⇒ 36



~~0 - 1 = 10~~ ✓

~~3 - 2 = 14~~ ✓

~~0 - 3 = 12~~ ✓

0 - 2 = 20

~~1 - 3 = 13~~ ✗

1 - 2 = 16

u v wt

0 - 1 - 10 ✓

0 - 3 - 12 ✓

1 - 3 - 13 ✗

3 - 2 - 14 ✓

1 - 2 - 16 ✗

0 - 2 - 20 ✗

0 - 3 - 12

1 - 3 - 13

0 - 2 - 20

3 - 2 - 14

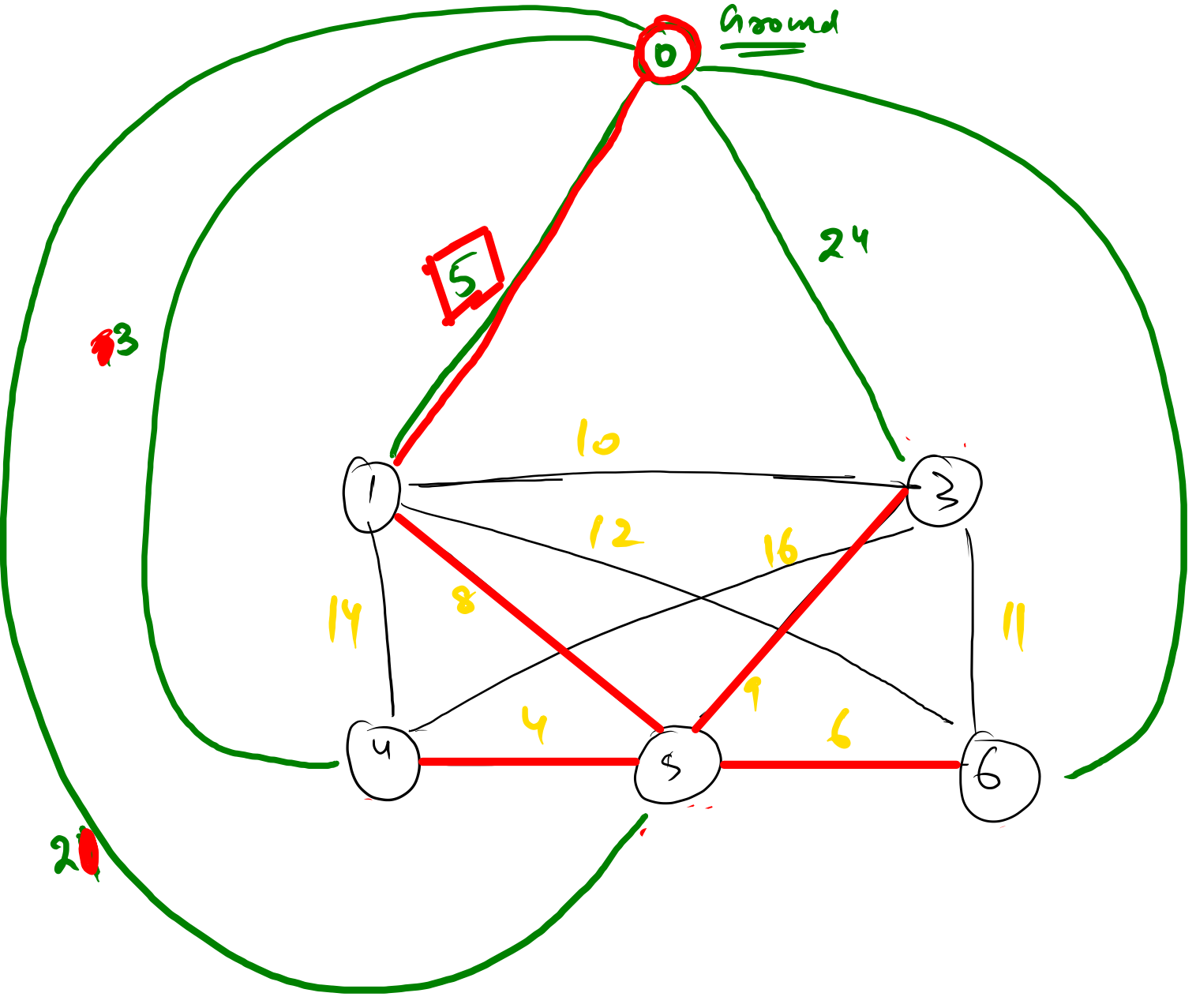
1 - 2 - 16

0 - 1 - 10

Cost: 0 + 10 + 12 + 14

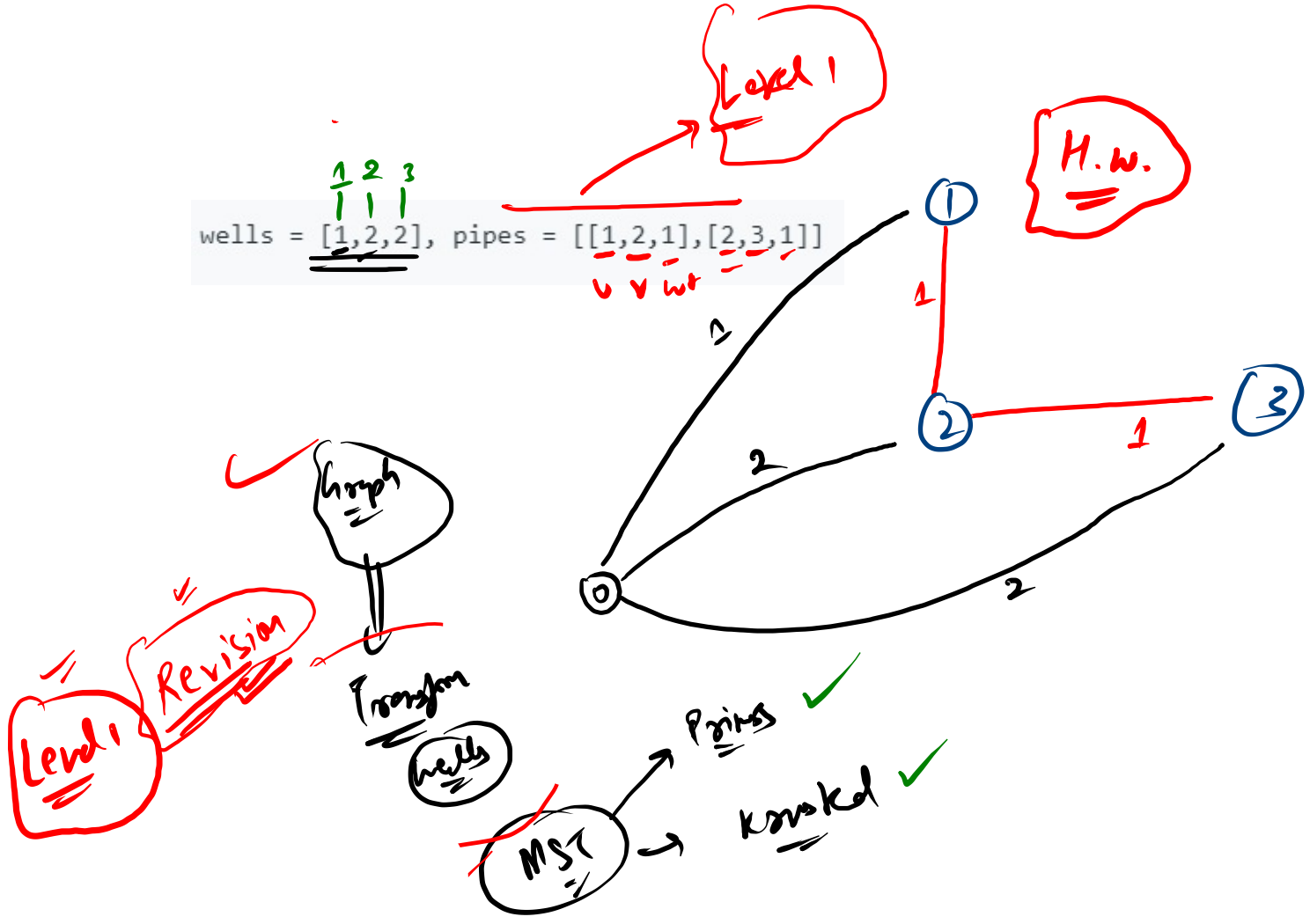
```java
public static void main (String[] args) throws java.lang.Exception
{
    Scanner scn = new Scanner(System.in);

    int vtces = scn.nextInt();
    int n = scn.nextInt();
    Edge edges[] = new Edge[n];
    for(int i = 0 ; i < n ; i++){
        int u = scn.nextInt();
        int v = scn.nextInt();
        int wt = scn.nextInt();
        edges[i] = new Edge(u,v,wt);
    }

    Arrays.sort(edges);

    UnionFind uf = new UnionFind(vtces+1);
    long cost = 0;
    for(Edge e : edges){
        if(uf.union(e.u,e.v)){
            cost += e.wt;
        }
    }

    System.out.println(cost);
}
```

```java
public boolean union(int vtx1,int vtx2){
    int rootv1 = find(vtx1);
    int rootv2 = find(vtx2);

    if(rootv1 != rootv2){
        int rankv1 = rank[vtx1];
        int rankv2 = rank[vtx2];
        if(rankv1 > rankv2){
            par[rootv2] = rootv1;
        }else if(rankv1 < rankv2){
            par[rootv1] = rootv2;
        }else{
            par[rootv2] = rootv1;
            rank[rootv1]++;
        }
        return true;
    }
    return false;
}
```



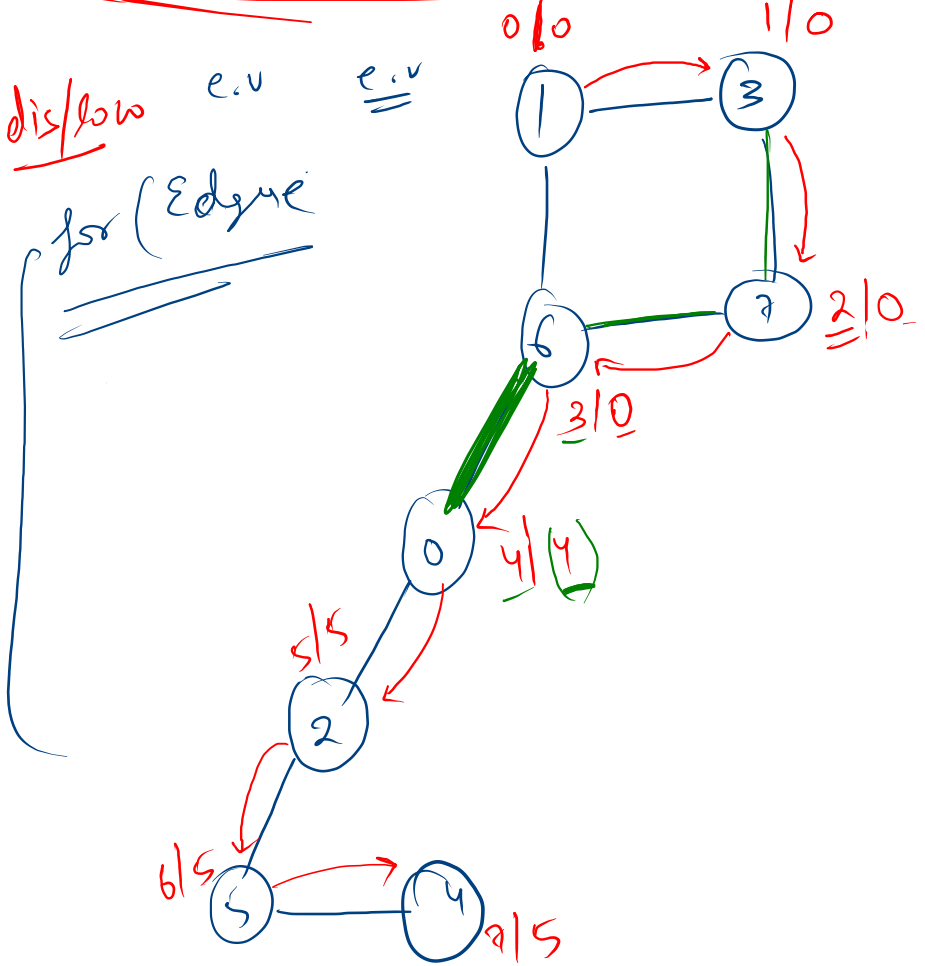
MST + choice
Multiple choices

MST =


Level 1
H.W.
1 2 3
wells = [1,2,2], pipes = [[1,2,1],[2,3,1]]
u v wt
1
2
3
1
2
1
2
0
Graph
Revision
Level 1
Transform
wells
MST
Prims
Kruskal

ARTICULATION POINT :

A point which generates disconnected graph if removed



(1) Low, disc (2)

AP

Parent

Continue

visited

Non Parent

low[u] = Min(low[u], dis[v])

unvisited

dfs call()

```
if ( low[v]  >=  disc[u]){
    ap[u] = true
}
low[u] = Min(low[u], low[v])
```