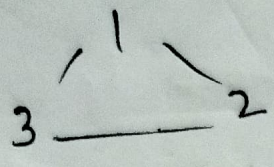
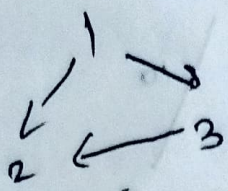
Detect cycle in an undirected graph



There exists a cycle if we hit a visited node which not a parent of current node during bfs.

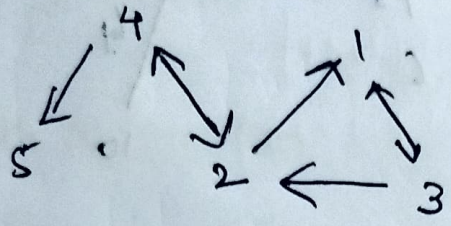
```
bool bfs (graph, src, vis)
{
    vis[src] = True;

    Queue<Pair> q = LinkedList<>();
    q.add(src, 0);
    while (q.size() > 0)
    Pair curr = q.poll();
    int node = curr.first;
    int parent = curr.second;
    for (int i : graph.get(node))
        if (!vis[i]){
            q.add (i, node);
        } vis[i] = True;
    else if (parent != i){
        return True;
    }
return false;
```

Detect cycle in a directed graph.

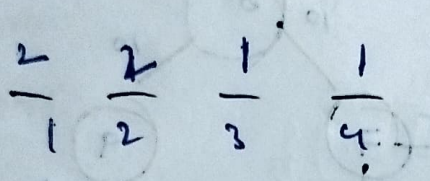


(Not a cycle)

0 2 1
1 2 3

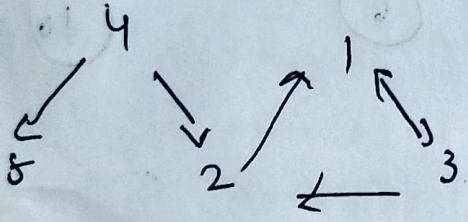


1 3 2



2 2 1 1 1
1 2 3 4 5

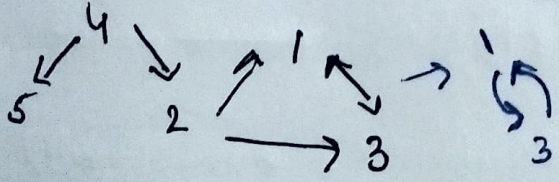
(Topo sort can't be done)



2 2 1 0 1
1 2 3 4 5

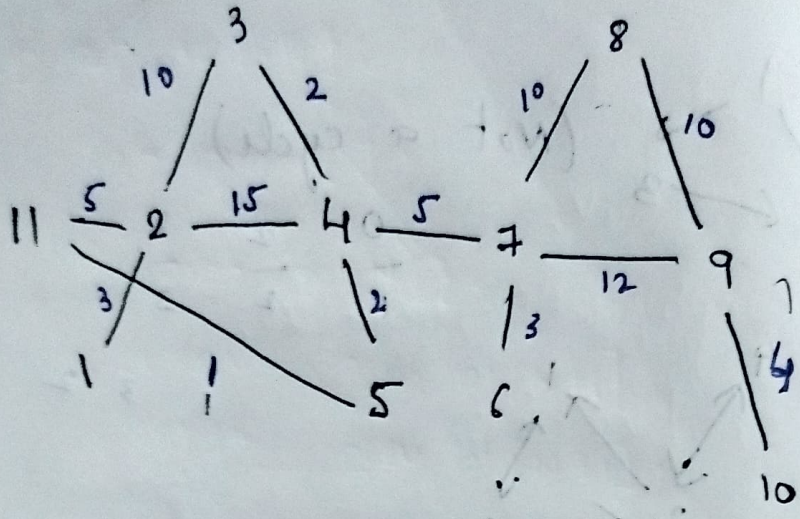
4 5 . . . . (Topo sort stopped)

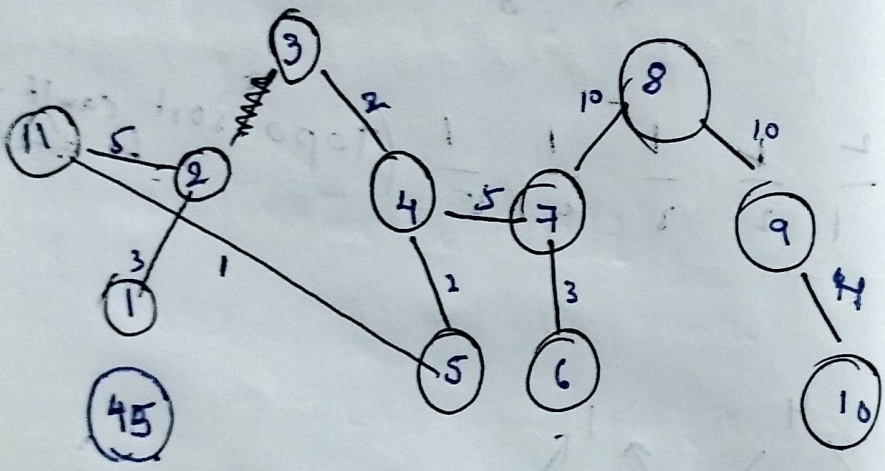
So change Topo sort return stmt as "return res.size() < n;"



→ This is a cycle.

Minimum Spanning Tree





Solutions

① Brute Force $O(N^2)$.

- draw every edge and check if there is cycle

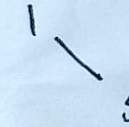
Disjoint set → Union By Rank
→ Union By Size

If graph is dynamic, we hv to use disjoint sets.

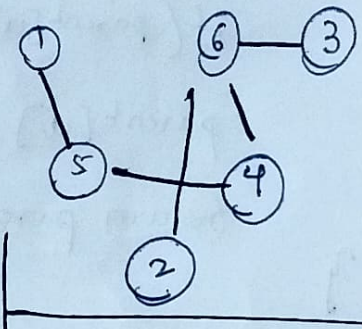
Path compresion

parent	1	2	3	4	5	6
	1	2	3	4	5	6
rank	0	0	0	0	0	0
	1	2	3	4	5	6

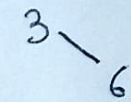
once we draw a path from 1,5



parent[5] = 1
rank[1] = 1

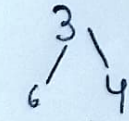


- now 3,6



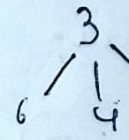
parent[6] = 3
rank[3] = 1

- now 6,4



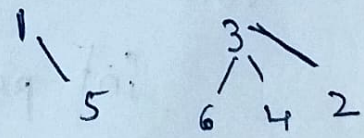
parent[4] = 3
rank[3] = 1

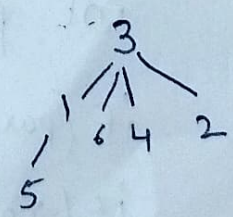
- now 6,2



parent[2] = 3
rank[3] = 1

- now 5,4





parent[5] = 3
rank[3] = 2.

To check if there is a path b/w 1,6 ⇒ par[1] = 3, par[6] = 3 so true

par	3	3	3	3	3	3
	1	2	3	4	5	6
rank	1	0	2	0	0	0
	1	2	3	4	5	6

```
class Disjoint {
    int rank[];
    Int parent[];
    Disjoint (int N) {
        rank = new int[N+1];
        parent = new int[N+1];
        N
        ∀ parent[i] = i;
        i=0
    }
    int findParent (int u) {
        if( parent[u] == u) return u;
        parent[u] = findParent (Parent[u]).
        return parent[u];
    }
    void unionByRank (int u, int v) {
        int paru = findParent(u);
        int parv = findParent(v);
        if (paru == parv) return;
        if (rank[paru] > rank [parV])
            parent[parv] = ~~parent~~ paru;
        else if (rank[parv] > rank[paru])
            parent[paru] = parv;
```

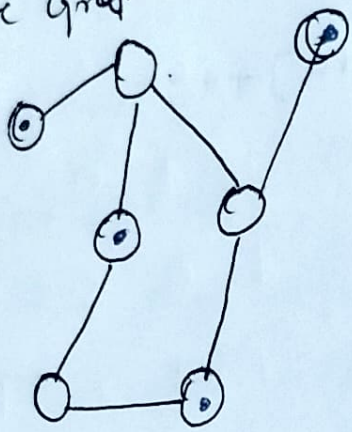
```
    else {
        par[parv] = paru;
        rank[paru]++;
    }
}
```

Lab agenda

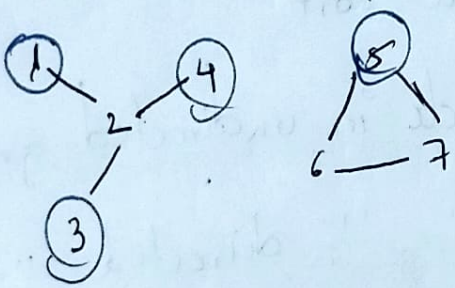
1. Topological sort
2. Detect cycle in undirected graph
3. " " " directed "
4. Minimum spanning tree.

02/05/2025

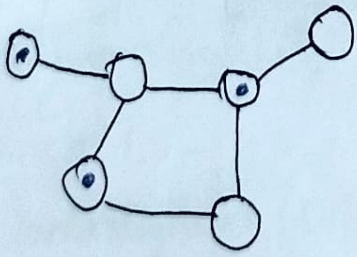
Bipartite Graph



NO



NO



Yes

```
bool bipartite(- graph, int[] col, int src)
{
    Queue<int> q= new linkedlist<>();
    col[src] = 0;
    q.add(src);
    while(q. size()>0){
        int curr = q.pop();
        int cc = col[curr];
        int nc=col[curr]==0?1:0;//1-col[curr];
        for(int i: graph.get(curr))
        {
            if(col[i]==-1){
                col[i]= nc;
                q.add(i);
            }
            else if(nc== col[i]){
                return False;
            }
        }
    return True;
}
```

Rat in a Maze

N-8 connectivity

0	1	1	0	0	1
0	1	0	1	1	0
0	1	0	0	1	0
0	1	0	1	1	0
0	R	0	1	C	0
1	0	0	0	1	0

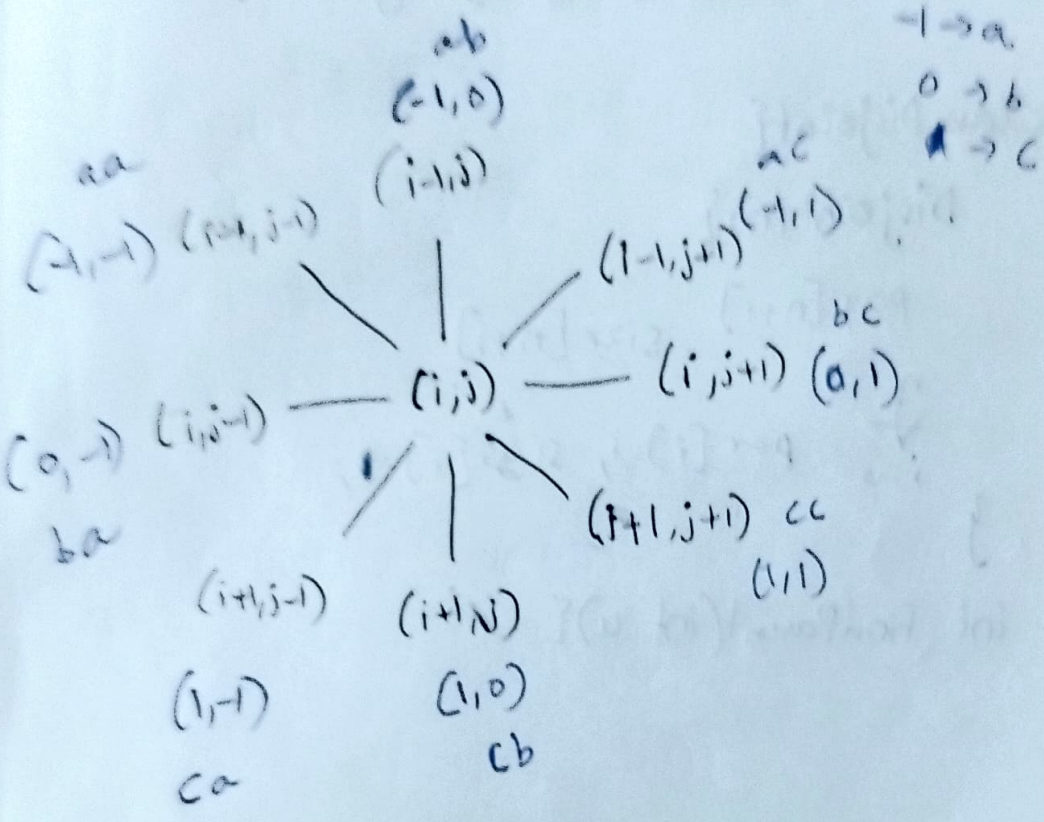
C → cheese
R → Rat
0 → water
1 → land

(i) Is there a path b/w R & C

(ii) Min steps b/w src & dest

```
void bfs(mat, n, m, dist, si, sj){
{   dx =
    dy =
    Queue<Pair> q = new LinkedList<>();
    q.add(new Pair(si, sj));
    dist[si][sj] = 0;
    while(q.size() > 0){
        Pair curr = q.pop();   int x = curr.x;
                               int y = curr.y;
        for(int i=0; i<8; i++){
            if(isvalid(x+dx[i], y+dy[i], n, m)){
                dist[x+dx[i]][y+dy[i]] = dist[x][y]+1;
                q.add(new pair(x+dx[i], y+dy[i]));
            }
        }
}
```

(iii) lexicographically shortest path



- We have to consider the cell with distance $n-1$ where n is dist of src in lexicographic order.

```
Class Disjoint{
    Disjoint(n){
        par[n+1], size[n+1]
        for i=0  par[i]=i, size[i]=1;
    }
    int findParent(int u){


    }
    void unionBySize(int u, int v){
        paru = FindParent(u);
        parv = findParent(v);
        if(paru == parv) ret;
        if( size[paru] > size[parv])
            size[paru] += size[parv];
        par[parv] = paru;
    }
```

```
else if (size[parv] > size[parw]{
    size[parv]
```