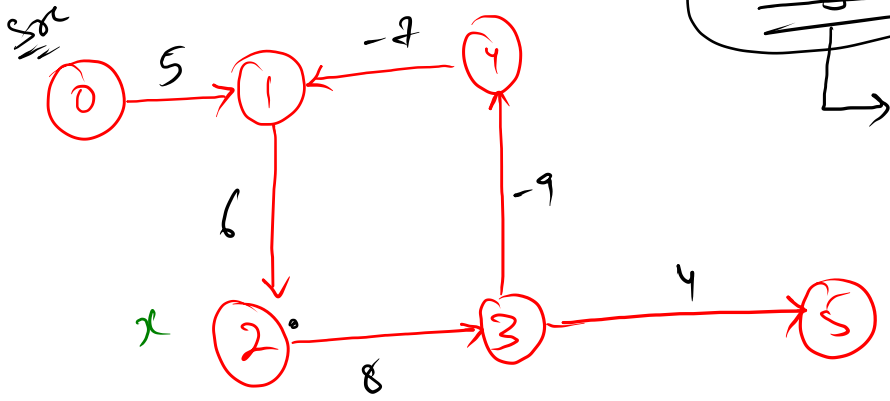
Graph ⇒ Cycle (+ve wt)

↳ shortest ✗

0-1-2 ⇒ 11

0-1-2-3-4-1-2 ⇒ 10



Dijkstra ⇒ $(E \log E) \Rightarrow (E \log V)$

↳ Edge wt ⇒ +ve, 0

Cyclic

Acyclic

-ve wt cycle

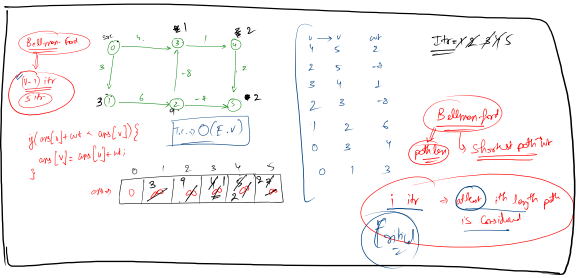
shortest path wt ✗

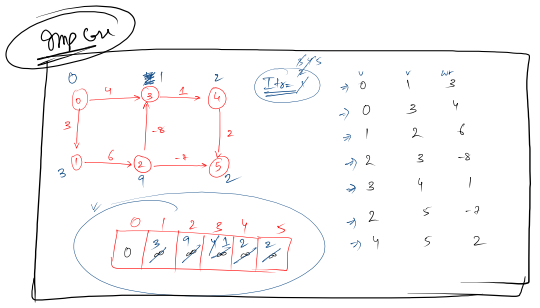
+ve wt cycle

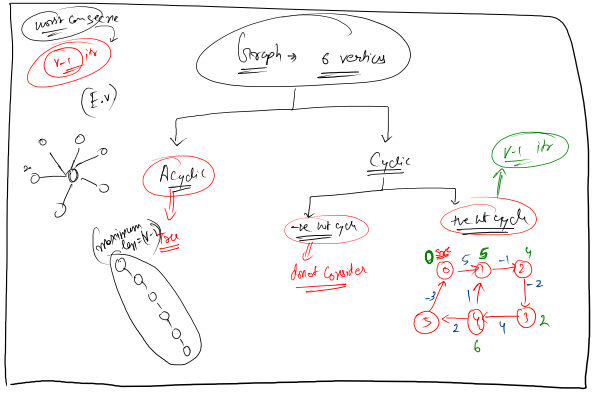
sp wt ✓

Tree

shortest path wt ✓







Graph: Acyclic / +ve wt cycle

↳ V-1 iterations

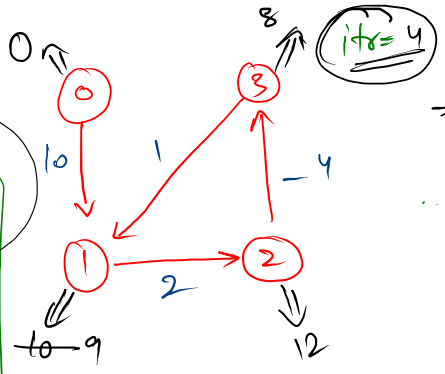
↳ shortest/minimum path in terms of wt for all vertices

detection of -ve wt cycle

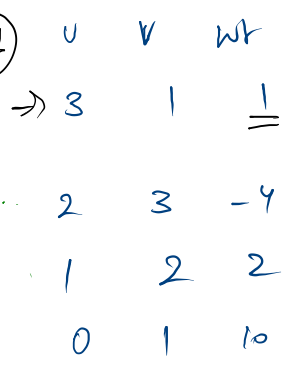
V^{th} → iteration → if shortest path wt of any vertex gets updated

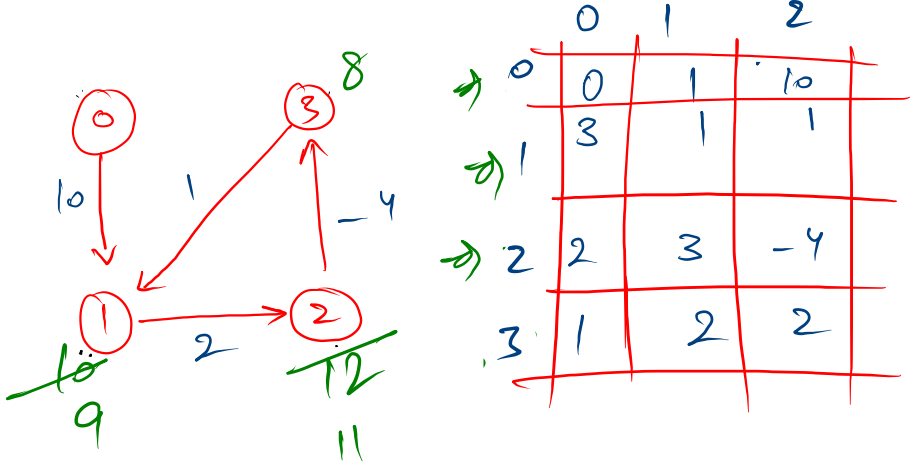
⇓

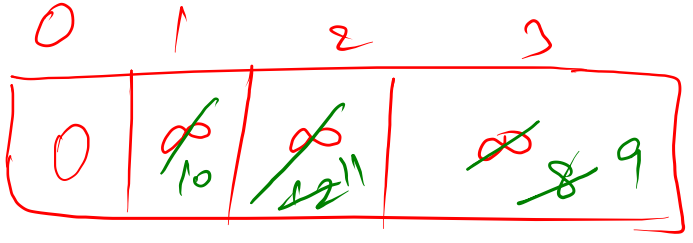
-ve wt. cycle.



0	1	2	3
0	~~10~~ 9	12	8



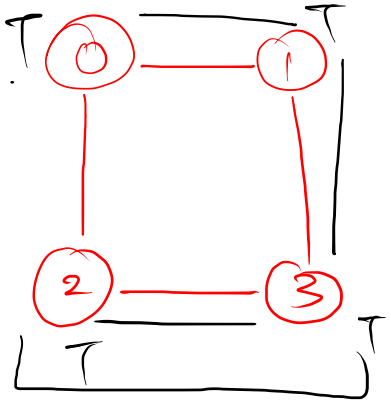


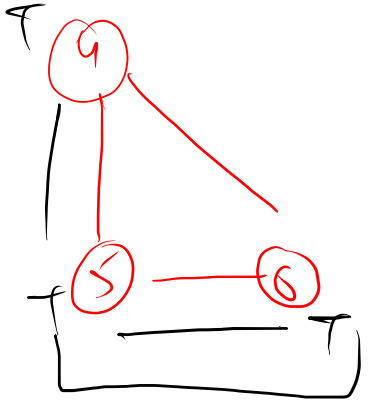



```java
class Solution
{
    public int isNegativeWeightCycle(int n, int[][] edges)
    {
        int ans[] = new int[n];

        Arrays.fill(ans,Integer.MAX_VALUE);
        ans[0] = 0;

        for(int i = 0 ; i < n-1 ; i++){ // v-1 iterations
            for(int idx = 0 ; idx < edges.length ; idx++){
                int edge[] = edges[idx];

                int u = edge[0] , v = edge[1] , wt = edge[2];

                if(ans[u] != Integer.MAX_VALUE && ans[u] + wt < ans[v]){
                    ans[v] = ans[u]+wt;
                }
            }
        }

        // vth iteration to detect negative loop
        for(int idx = 0 ; idx < edges.length ; idx++){
            int edge[] = edges[idx];

            int u = edge[0] , v = edge[1] , wt = edge[2];

            if(ans[u] + wt < ans[v]){
                return 1;
            }
        }

        return 0;
    }
}
```

Connected Component

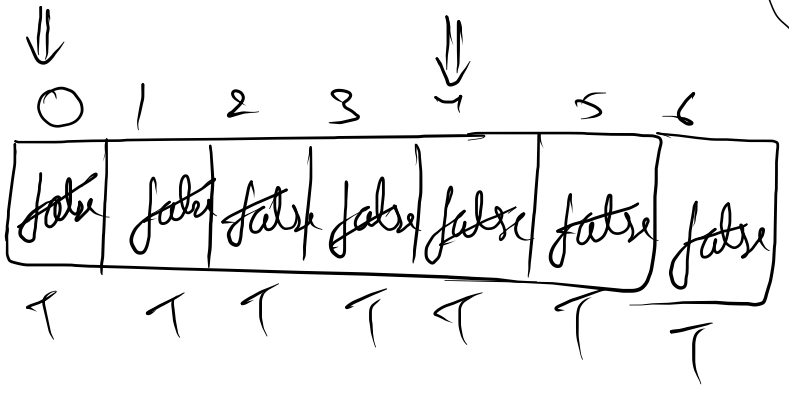
Undirected Graph



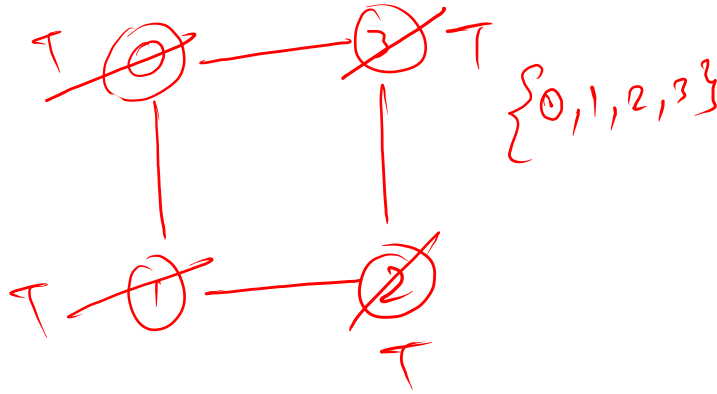


{0, 1, 2, 3}

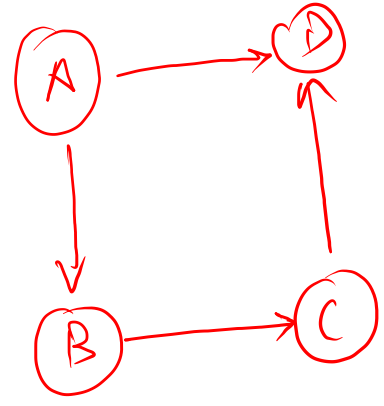
{4, 5, 6}

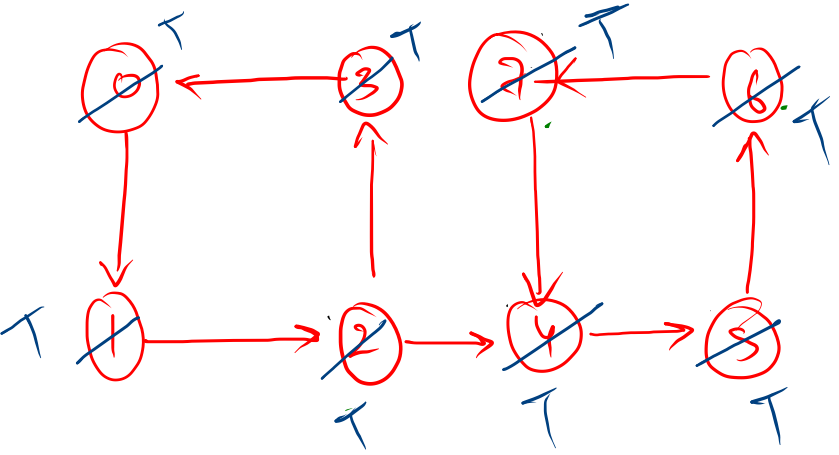


undirected Graph ⇒ Connected Components



Directed Graph ⇒ Strongly Connected Components





DFS

1. DFS → stack
2. Edges → dir → rev
3. Dfs → revGraph → Count

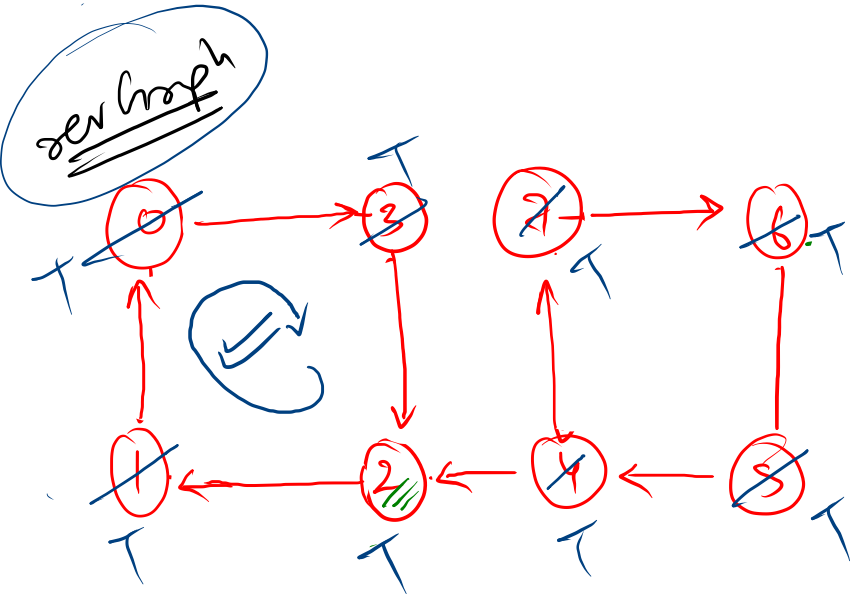
SCC1

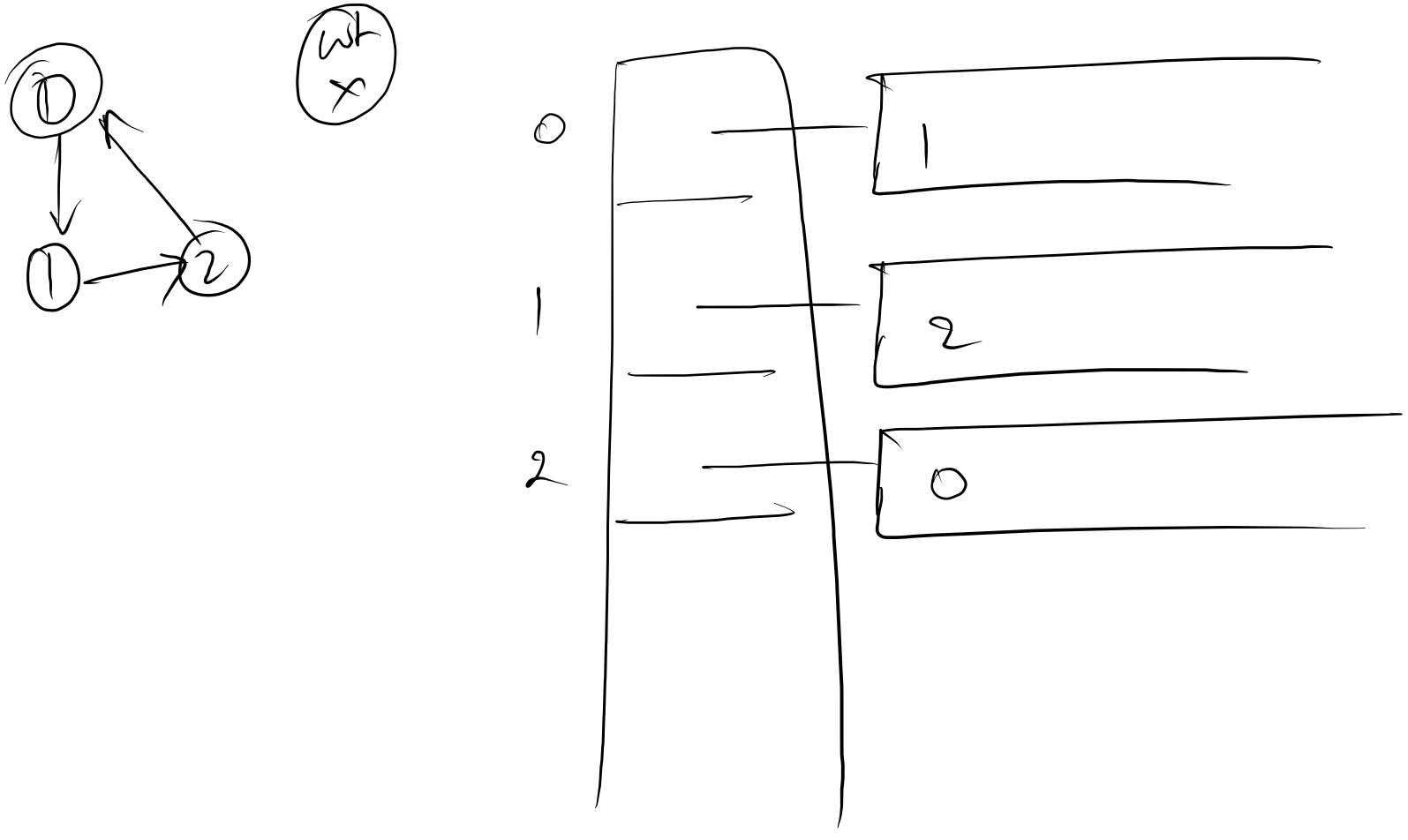
SCC2

{0, 1, 2, 3} ← { 4, 5, 6, 7 }

Count : 1 X 2

revGraph



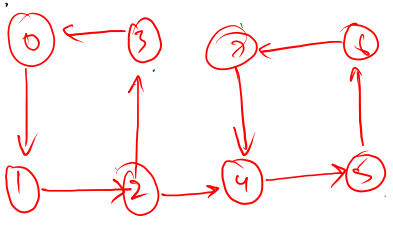

0
1
2
wt
x
0
1
1
2
2
0

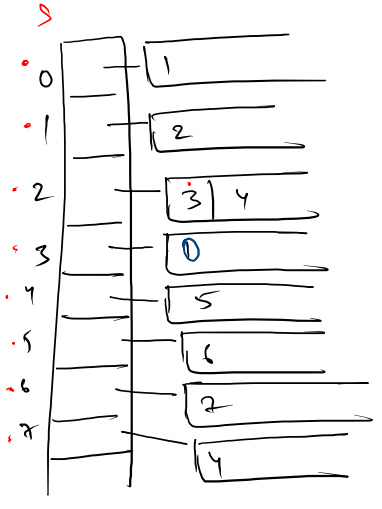
1

```
// dfs --> stack fill
Stack<Integer> st = new Stack<>();
boolean[]vis = new boolean[V];
for(int vtx = 0 ; vtx < V ; vtx++){
    if(vis[vtx] == false){
        dfs1(graph,vtx,vis,st);
    }
}
```

```
public static void dfs1(ArrayList<ArrayList<Integer>> graph , int src , boolean[]vis, Stack<Integer> st){
    vis[src] = true;
    ArrayList<Integer> nbrs = graph.get(src);
    for(int nbr : nbrs){
        if(vis[nbr] == false){
            dfs1(graph,nbr,vis,st);
        }
    }
    st.push(src);
}
```

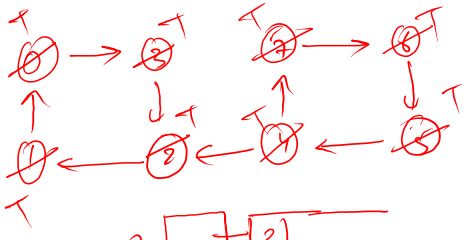
2

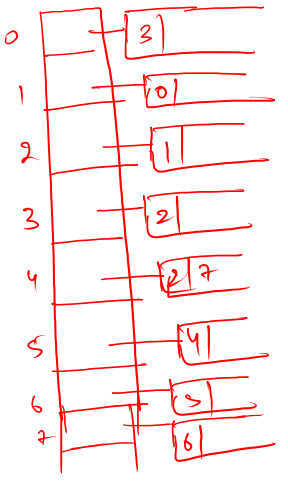


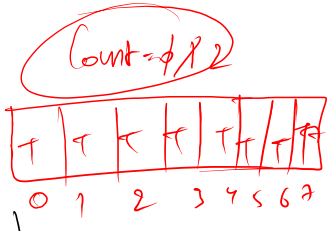


```
ArrayList<ArrayList<Integer>> revGraph = new ArrayList<>();
for(int vtx = 0 ; vtx < V ; vtx++){
    revGraph.add(new ArrayList<Integer>());
}

for(int vtx = 0 ; vtx < V ; vtx++){
    ArrayList<Integer> nbrs = graph.get(vtx);
    for(int nbr : nbrs){
        revGraph.get(nbr).add(vtx);
    }
}
```

3







```
// dfs & count SCC
int count = 0;
vis = new boolean[V];
while(st.size() > 0){
    int vtx = st.pop();
    if(vis[vtx] == false){
        count++;
        dfs2(revGraph,vtx,vis);
    }
}

return count;
```

4

```
public static void dfs2(ArrayList<ArrayList<Integer>> graph , int src , boolean[]vis){
    vis[src] = true;
    ArrayList<Integer> nbrs = graph.get(src);
    for(int nbr : nbrs){
        if(vis[nbr] == false){
            dfs2(graph,nbr,vis);
        }
    }
}
```