Lecture : Graphs 2

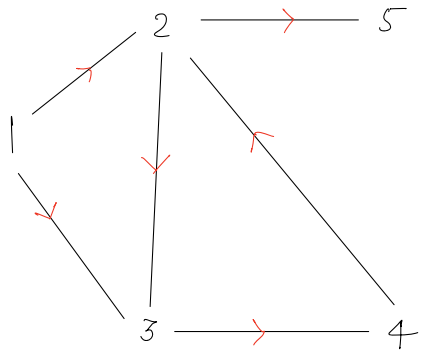
Agenda

- Topological sort
- DSU
- Path compression
- No of islands [8 dir^c — h/w]

Qu Given n courses with pre-requisites of each course. Check if it is possible to finish all courses

ip n = 5

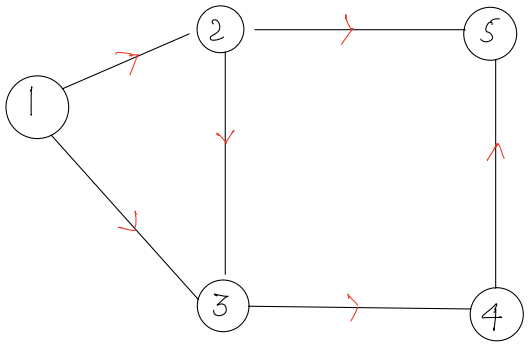
x is a prerequisite of y.




```
if ( graph has a cycle) {
        ans = false;
} else {
        ans = true;
}
```

1 2 3 4 5 ✓

5 4 3 2 1 ✗

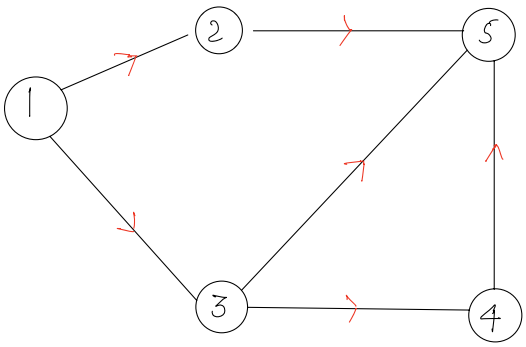


true

1 2 3 4 5 ✓

1 3 2 4 5 ✓

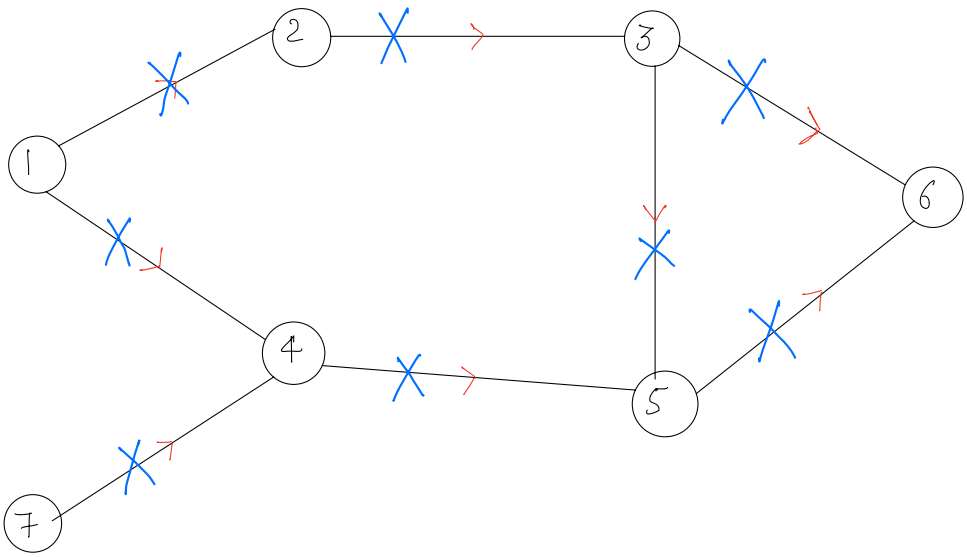
1 3 4 2 5 ✓



Topological sort / order.

Linear ordering of nodes such that if there is an edge b/w u and v, u will always come before v in order

Approach



Indegree : no of incoming edges

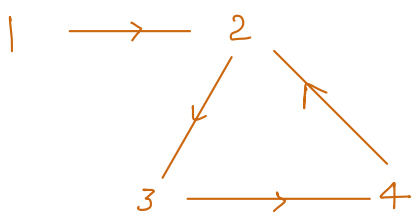
indegree[] =

0	1	2	3	4	5	6	7
	0	~~1~~ 0	~~1~~ 0	~~2~~ ~~1~~ 0	~~2~~ ~~1~~ 0	~~2~~ ~~1~~ 0	0

Queue | ~~1~~ ~~7~~ ~~2~~ ~~4~~ ~~3~~ ~~5~~ ~~6~~ |

Order 1 7 2 4 3 5 6

Ex 2



indegree[] =

0	1	2	3	4
	0	2	1	1

Pseudocode

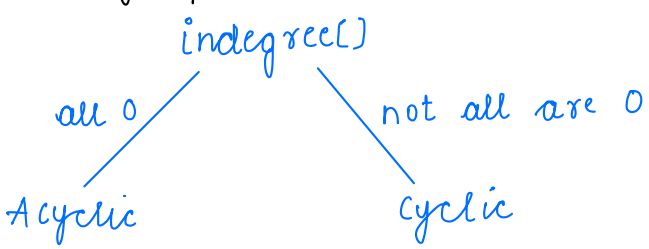
```
void topologicalSort(n, edges[][]) {
    List<Integer> graph[n+1];
    indegree[n+1];
    m = edges.length;
    for(i=0; i<m; i++) {
        u = edges[i][0];
        v = edges[i][1];
        graph[u].add(v);
        indegree[v]++;
    }
    Queue<Integer> q;
    for(i=1; i<=n; i++) {
        if(indegree[i]==0) {
            q.add(i);
        }
    }
    while(!q.isEmpty()) {
        curr = q.poll();
        print(curr);
        List<Integer> nghbrs = graph[curr];
        for(int v: nghbrs) {
            indegree[v]--;
            if(indegree[v]==0) {
                q.add(v);
            }
        }
    }
}
```

Build graph & indegree

TC: $O(n+e)$

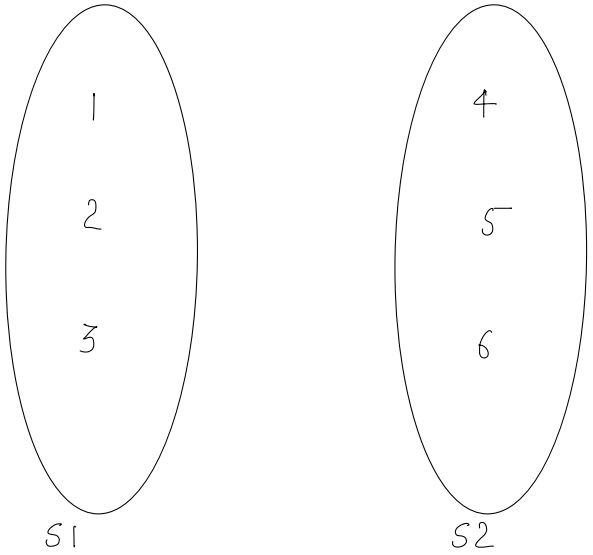
SC: $O(n)$

Qu Detect if graph has cycle?



Break: 7:55 – 8:05

DSU



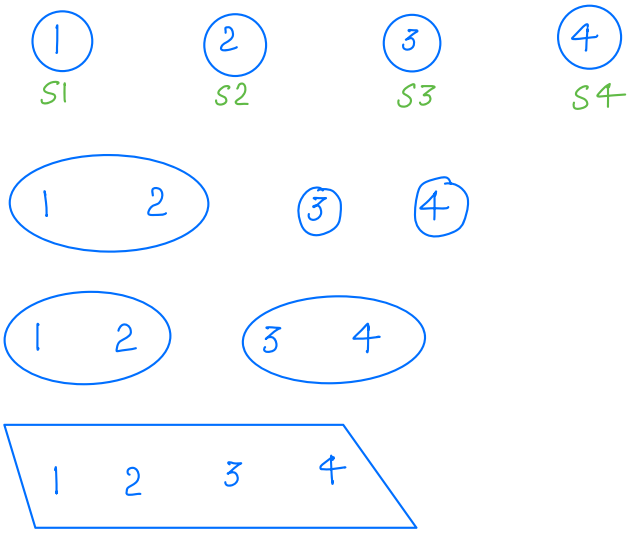
$S1 \cup S2 = \{1, 2, 3, 4, 5, 6\}$

$S1 \cap S2 = \phi$ { disjoint sets }

Qu Given n elements. Consider each element as a unique set and perform multiple queries.

In each query if (u, v) belong to different sets, we do their union and return true, else return false.

n = 4



Queries

(1, 2) — true

(3, 4) — true

(1, 2) — false

(1, 4) — true

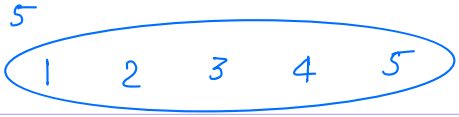
(2, 3) — false

Approach

parent[] =

0	1	2	3	4	5
	1	~~2~~ 1	~~3~~ 1	~~4~~ 3	~~5~~ 1

n = 5

Queries		
(1, 2)	p[1] = 1 p[2] = 2 	true
(3, 4)	p[3] = 3 p[4] = 4 	true
(1, 2)	p[1] = 1 p[2] = 1	false.
(1, 4)	p[1] = 1 p[4] = 3 ↓ p[3] = 3 	true
(2, 4)	p[2] = 1 p[4] = 3 ↓ p[3] = 1	false
(1, 3)	p[1] = 1 p[3] = 1	false
(4, 5)	p[4] = 3 ⟶ p[3] = 1 ⟶ p[1] = 1 p[5] = 5 	true
(2, 5)	p[2] = 1 ⟶ p[1] = 1 p[5] = 1 ⟶ p[1] = 1	false.

Pseudocode

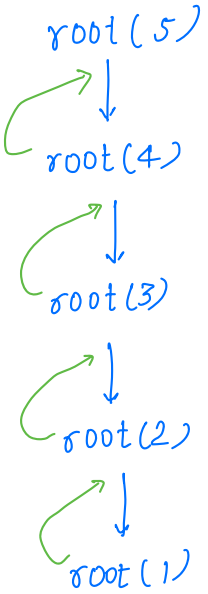
```
boolean union(x, y, parent[]) {
        rootX = getRoot(x);
        rootY = getRoot(y);
        if (rootX == rootY) {
            return false;
        }
        if (rootX < rootY) {
            parent[rootY] = rootX;
        } else {
            parent[rootX] = rootY;
        }
        return true;
}

int getRoot(x, parent[]) {
        if (x == parent[x]) {
            return x;
        }
    int ans = getRoot(parent[x], parent);
        return ans;
}
```

union

TC: O(n)
SC: O(n) + stack size

Path compression




```
int getRoot(x, parent[]) {
    if (x == parent[x]) {
        return x;
    }
    ans = getRoot(parent[x], parent);
    return ans;
}
```

App of DSU

- Check if graph is connected ?
- Detecting cycle in undirected graph
- MST

h/w

Thankyou :)