Lecture : Graph - 2

Agenda

- Topological sort
- DSU
- Path compression.

Qu1 Given n courses with pre-requisites of each course. Check if it is possible to finish all courses.

Input: $n = 5$

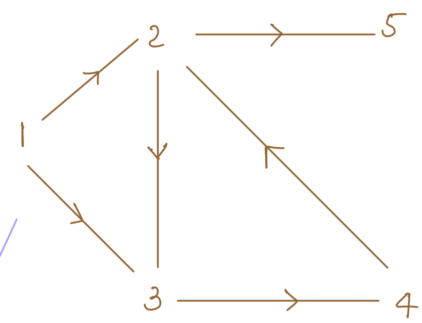
x is a prerequisite of y.

$1 \longrightarrow 2$ & 3

$2 \longrightarrow 3$ & 5

$3 \longrightarrow 4$

$4 \longrightarrow 2$

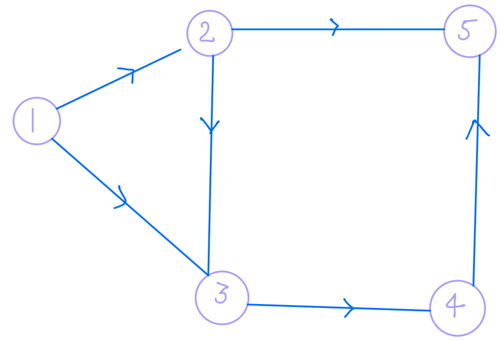



```
if ( graph has a cycle) {
    ans = false;
} else {
    ans = true;
}
```

DAG (Directed acyclic graph)

1 2 3 4 5 ✓

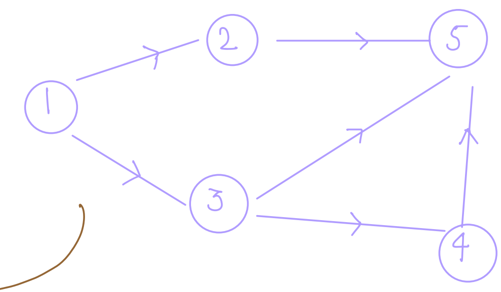
5 4 3 2 1 ✗



1 2 3 4 5 ✓

1 3 2 4 5 ✓

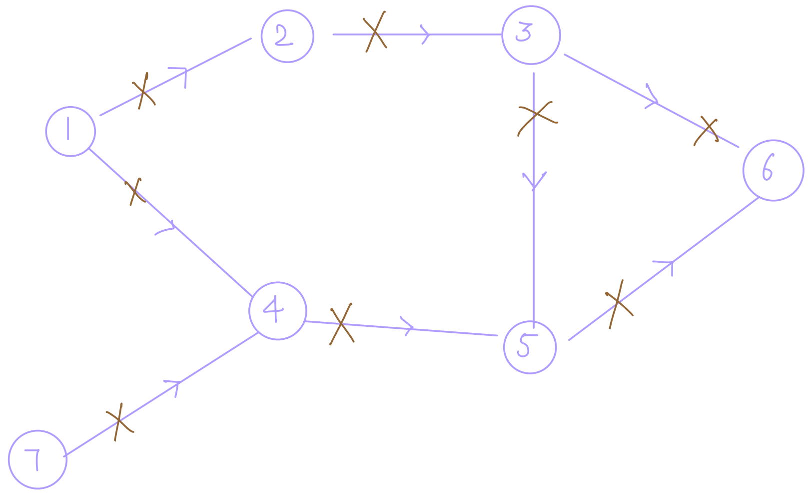
1 3 4 2 5 ✓



Topological sort: / order

Linear ordering of nodes such that if there is an edge b/w u and v, u will always come to left of v.

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

Code:

```
void topologicalsort( n, edges[][] ) {
        List<Integer> graph[n+1];
        indegree[n+1];
        m = edges.length;
        for (i=0; i<m; i++) {                 →  Build graph &
                 u = edges[i][0];                  indegree
O(e)   ←         v = edges[i][1];
                 graph[u].add(v);
                 indegree[v]++;
        }

       Queue<Integer> q;
       for (i=1; i<=n; i++) {
O(n) ←         if (indegree[i]==0) {
                      q.add(i);
               }
       }

       while(!q.isEmpty()) {
             curr = q.poll();
             print(curr);
             List<Integer> nghbrs = graph[curr];
O(n) ←       for(int v: nghbrs) {
                   indegree[v]--;
                   if( indegree[v]==0) {
                          q.add(v);
                   }
             }
       }
}
```

TC: $O(n+e)$
SC: $O(n)$

Qu: Detect if graph is cyclic.

indegree[ ]

all 0

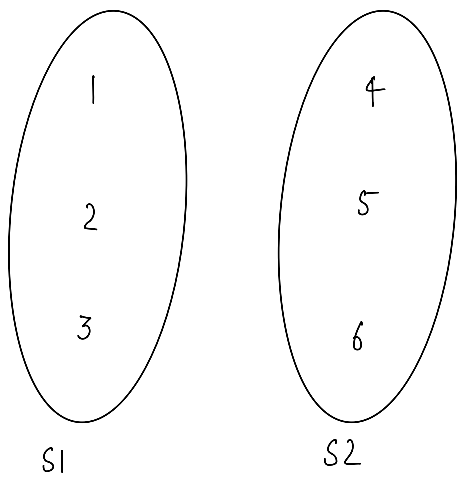
DAG (Acyclic)

not all are 0

DCG (Cyclic)

Break: 7:53 – 8:03 AM

DSU [Disjoint set union | Union-find]



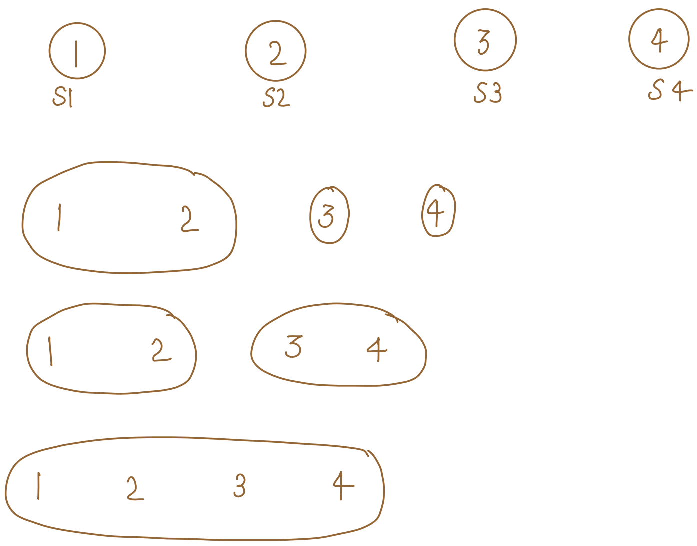
$S1 \cup S2 = \{1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6\}$

$S1 \cap S2 = \phi$ {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 3 4 5

(1, 2)	parent[1] = 1 parent[2] = 2	1 2 · 3 · 4 · 5 — true
(3, 4)	parent[3] = 3 parent[4] = 4	1 2 · 3 4 · 5 — true
(1, 2)	parent[1] = 1 parent [2] = 1	⟶ false
(1, 4)	parent[1] = 1 parent[4] = 3 ↓ parent[3] = 3	1 2 3 4 · 5 — true
(2, 4)	parent[2] = 1 ⟶ parent[1] = 1 parent[4] = 3 ⟶ parent[3] = 1	false
(1, 3)	parent[1] = 1 parent[3] = 1 ⟶ parent[1] = 1	false
(4, 5)	parent[4] = 3 ⟶ parent[3] = 1 ⟶ parent[1] = 1 parent[5] = 5 1 2 3 4 5	true
(2, 5)	parent[2] = 1 ⟶ parent[1] = 1 parent [5] = 1 ⟶ parent[1] = 1	false

Code:

```
        boolean union( x, y, parent[] )    {
O(n)  ——  rootX = root(x);
O(n)  ——  rootY = root(y);
              if(rootX == rootY) {
                  return false;
              }
O(1)  ——      if( rootX < rootY) {
                  parent[rootY] = rootx;
              } else {
                  parent[rootx] = rooty;
              }
              return true;
        }
```

```
        int root( x, parent[]) {
            if(x == parent[x]) {
                return x;
TC: O(n) <——  }
            int ans = root( parent[x], parent);
            return ans;
        }
```

root(4) 1

ans = root(3) 1

1

ans = root(1)

parent[] =

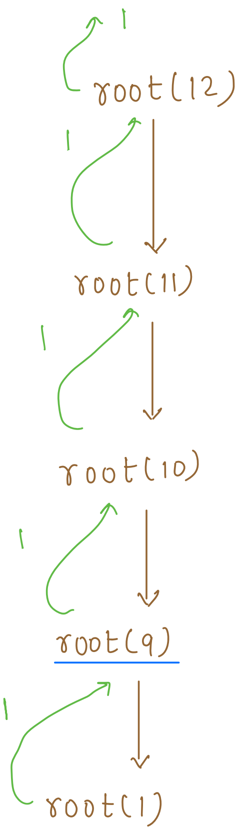
0	1	2	3	4	5
	1	~~2~~ 1	~~3~~ 1	~~4~~ 3	~~5~~ 1

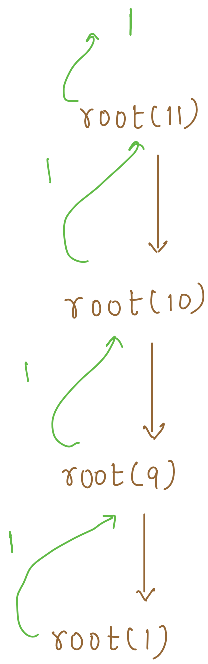
TC: O(n)

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

Path compression

0	1	2	3	4	5	6	7	8	9	10	11	12
	1	1	1	3	1	6	2	4	1	~~9~~ 1	~~10~~ 1	~~11~~ 1

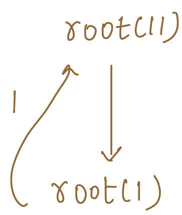





```
int root(x, parent[]) {
    if (x == parent[x]) {
        return x;
    }
    int ans = root(parent[x], parent);
    parent[x] = ans;
    return ans;
}
```

TC: O(1) amortized

Because of 1 expensive operation, other op^n become easy.



Thankyou :)