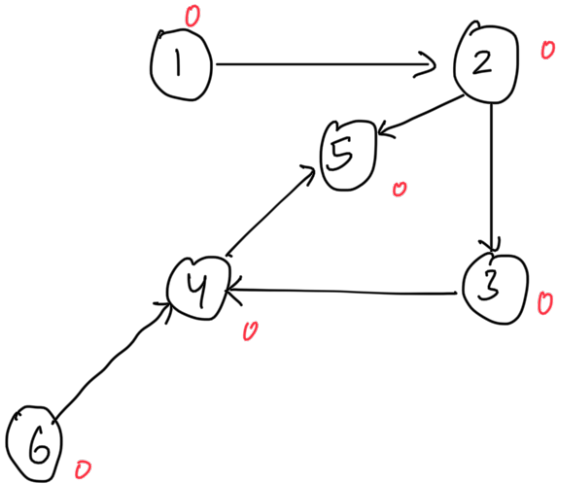
17 April 2022

Khan's Algorithm.



Q

TS : 1 6 2 3 4 5

> for each node i whose indegree = 0 it can come to first place in the ordering of Topological Sort.

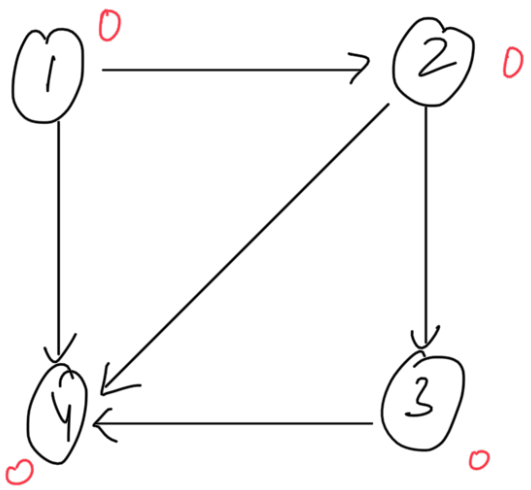
Code

```
vector<int> KhanAlgo(V, G) {
    vector<int> InDegree(V, 0);
    for(i=0; i<V; i++) {
        for(to : G[i]) {
            InDegree[to]++;
        }
    }
```

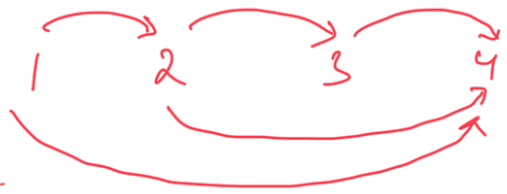
```
>

Queue<int> Q;

for(i=0; i<v; i++) {

    if(InDegree[i]==0){

        Q.push(i);

    }

}

vector<int> res;

int cnt=0;

while(!Q.empty()) {

    int at = Q.front();

    Q.pop();

    res.push_back(at);

    for(to: G[at]) {

        InDegree[to]--;

        if(InDegree[to]==0){

            Q.push(to);

        }

    }

    cnt++;

}

if(cnt != v) {

    Cycle Detected;

} else {

    return res;
```

3 3

Ex



valid

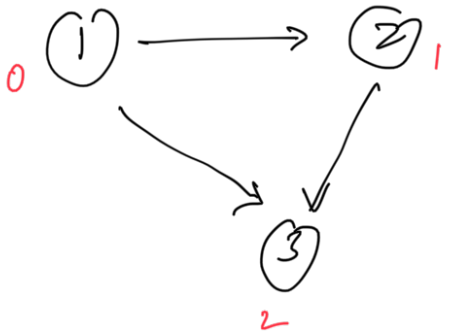


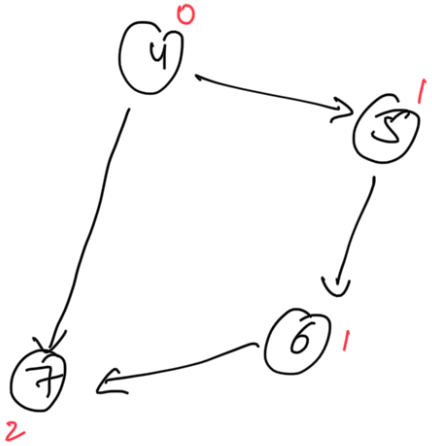
Cnt = 4

Q :

TS : | 1 2 3 4

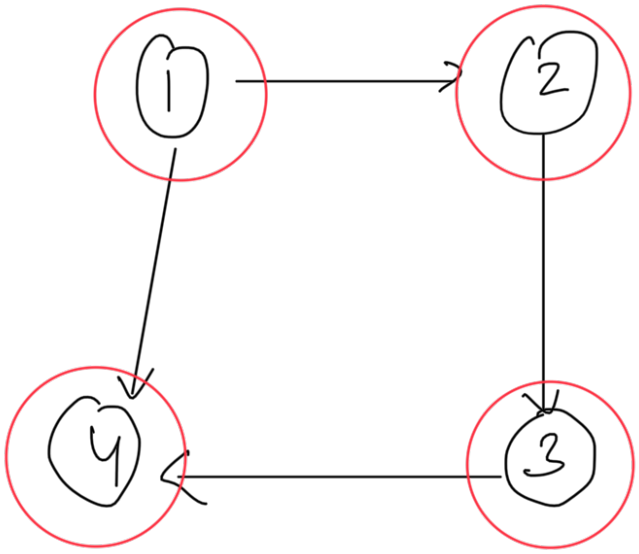
Ex:





* khan's Algorithm also works for disconnected graphs.

Topological Sort Using DFS & Departure Time.



Time = 0

vector<int> res(V);

0	1	2	3
4	3	2	1

time = 4

```
void DFS ( at ) {
```

== Arrival

```
visited[at] = true
for ( to : g[at] ) {
    if ( !visited[to] ) {
        DFS (to)
    }
}
res[time] = at
time++ ;
```

== Departure

```
}
```

Reverse order

res . reverse

Q: Given a DAG G. Find the lexographically smallest topological sorting ordering.

#

(1) DFS + stack

(2) Khan's Algorithm ✓

(3) DFS + DepartureTime

We can use min-heap with Khan's Algorithm to solve the above problem.

Union-Find (Disjoint-Set)

[1, n] → numbers

R[n]

R[i]

i = R[i]

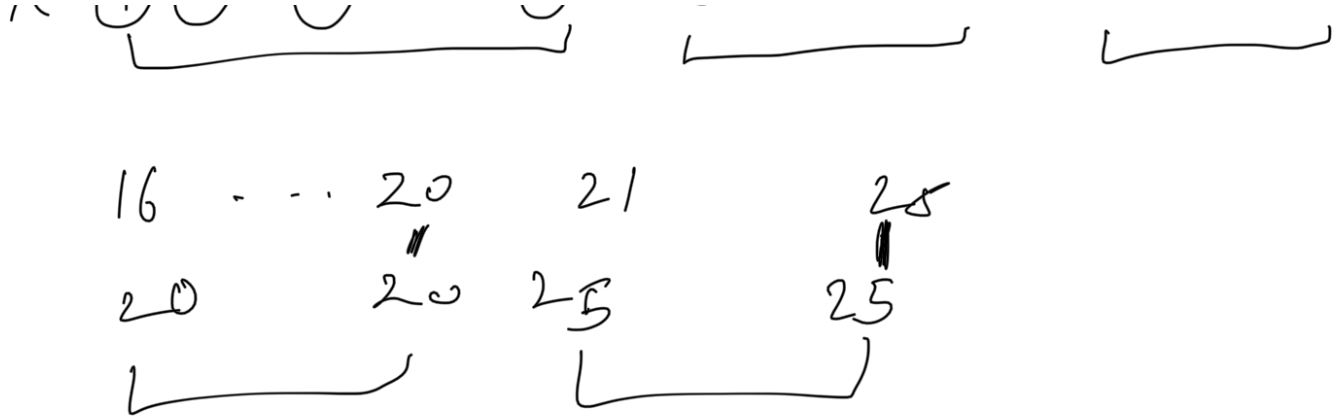
R[i] is the group that contains ith number.

i ≠ R[i]

ith number belong to the group in which R[i]th number is present.

R[i]

i	1	2	3	4	5	6	...	10	11	...	15
R	1	1	2	3	2	6		6	11		11

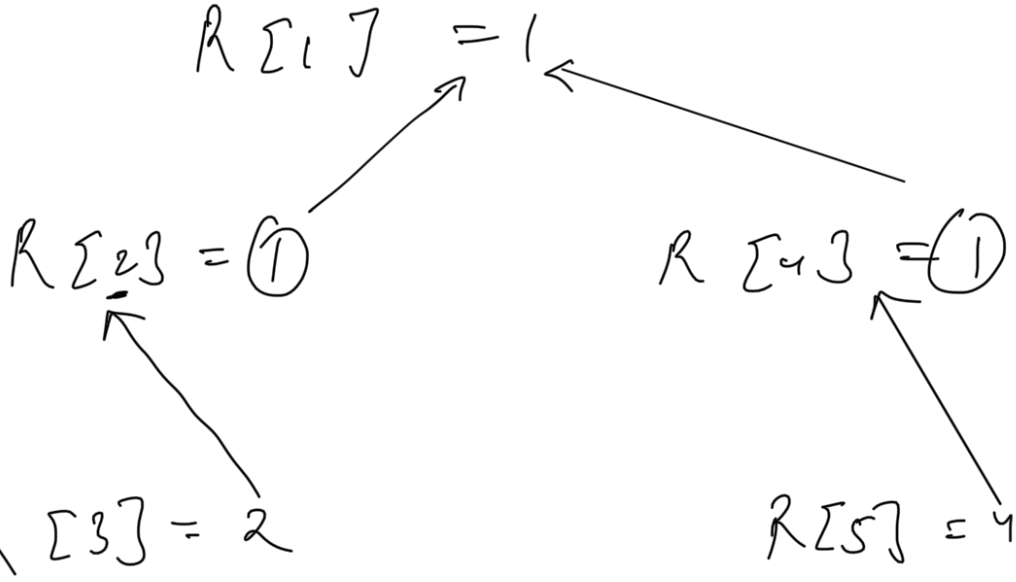


$X \Rightarrow$ $X = R[x]$

for all i's such that

$R[i] = x$

$\Rightarrow$ i'th number belongs to Xth group



Implementation

```
int N;
vector<int> R;

void initialize(int n) {
    N = n;
    R.resize(n+1);
    for(i=1; i<=n; i++){
        R[i] = i;
    }
}
int findRoot(int n) {
    if(n = R[n]) {
        return n;
    }
    return findRoot(R[n]);
}

void UnionRoots(int x, int y)
{
    int g1 = findRoot(x);
    int g2 = findRoot(y);
```

```
if(g1 = g2) {
        return;
}
R[g1] = g2;
}
```