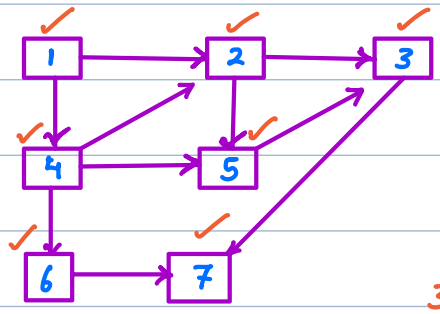
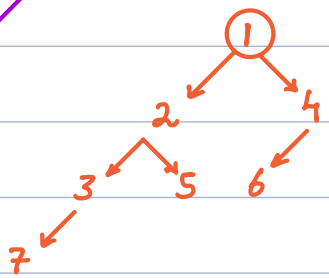
29 → Graphs 3

31 → Interview Problems

5 Aug → Final Exam (Mon)

BFS → Breadth First Search / Level Order Traversal





DS used → Queue

1 2 4 3 5 6 7

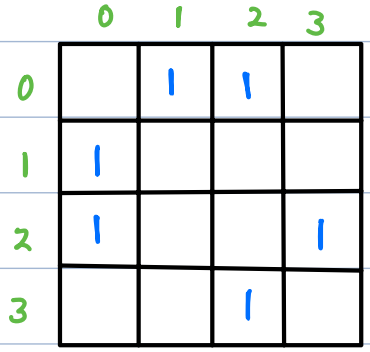
o/p → 1 2 4 3 5 6 7

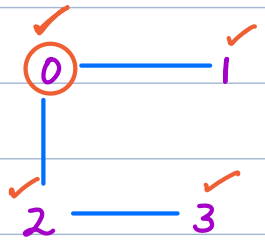
```
void bfs (s) {
    // Queue → q
    q.enqueue(s)
    vst[s] = true
    while (!q.isEmpty()) {
        u = q.dequeue()
        print(u)
        for (v : Adj[u]) {
            if (!vst[v]) {
                q.enqueue(v)
                vst[v] = true
            }
        }
    }
}
```

```
∀i, vst[i] = false
for i → 1 to N {
    if (!vst[i]) bfs(i)
}
```

TC = O(N + E)

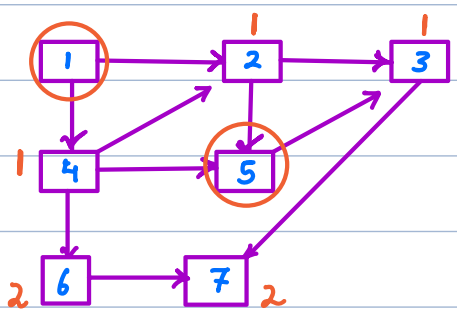
SC = O(N)



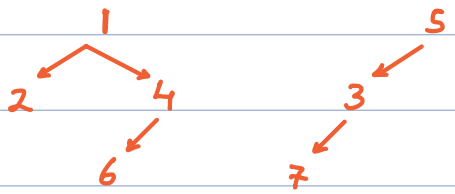


o/p → 0 1 2 3

Multisource BFS



Find min edges to travel to reach every node starting from anyone starting point.

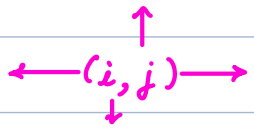


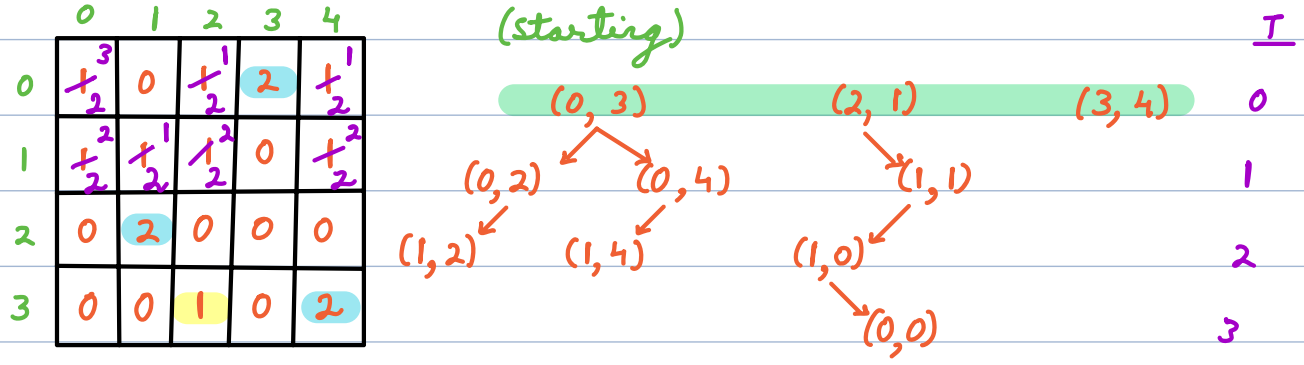
1) Insert all starting points in the queue.
2) Apply BFS

Q → **Rotten Oranges**

Given a matrix s.t A[i][j] → 0 empty
→ 1 fresh orange
→ 2 rotten orange

Every hour a fresh orange adjacent to a rotten orange becomes rotten. Find the time when all oranges become rotten, if not possible return -1.





Ans = -1

```
for i → 0 to (N-1) {
    for j → 0 to (M-1) {
        if ( A[i][j] == 2) {
            q.enqueue ( {i, j, 0} )   // cell index, timer
        }
    }
}   T = 0
dx[] = { 0  -1  1  0 }
dy[] = {-1  0  0  1 }
while (! q.isEmpty ()) {
    i, j, h = q.dequeue ()
    T = max (T, h)
    for k → 0 to 3 {
        x = i + dx[k]
        y = j + dy[k]
        if ( 0 <= x && x <= N-1  &&
            0 <= y && y <= M-1 && A[x][y] == 1) {
                A[x][y] = 2   // any integer > 1
                q.enqueue ( {x, y, h+1} )
        }
    }
}
```

```
    }
    for i → 0 to (N-1) {
        for j → 0 to (M-1) {
            if (A[i][j] == 1)  return -1
        }
    }
    return T
```

TC = O(N*M) SC = O(N*M)

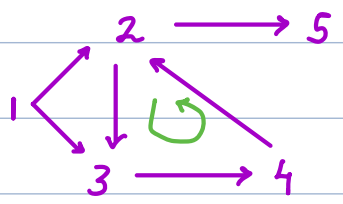
Q→ Given N cources with pre-requisites.
Check if it is possible to finish all courses.

N = 5

x is pre-requisite of y

x	y
1	2 & 3
2	3 & 5
3	4
4	2
5	X

Ans = false



if (cyclic graph) ⇒ Ans = false ✓
else ⇒ Ans = true

Q→ If it is possible to finish all cources,
find any one order of completing the cources

N = 5

x is pre-requisite of y

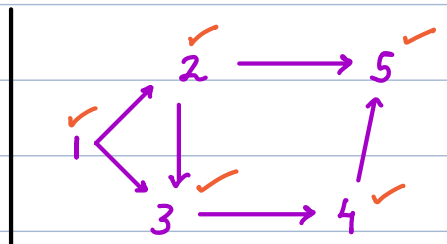
1 → 2 & 3

2 → 3 & 5

3 → 4

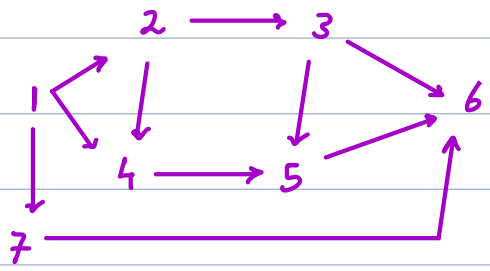
4 → 5

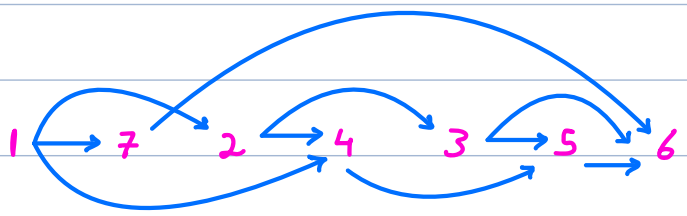
5 → X



o/p → 1 2 3 4 5

Topological Sort → linear ordering of vertices (nodes) in a Directed Acyclic Graph (DAG) s.t ∀ edge (u → v), u is on the left side of v.

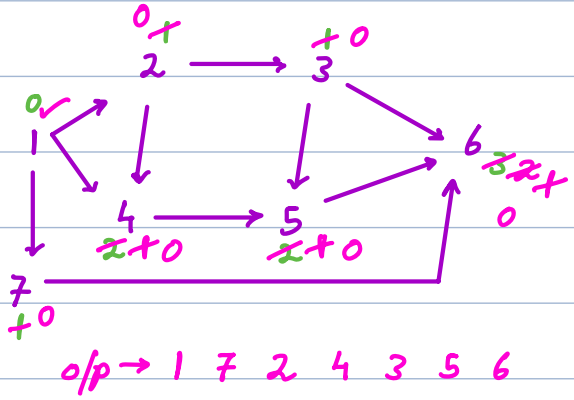




Left to Right

1) Find indegree ∀ nodes.

```
for u → 1 to N {
    for ( v : Adj[u] ) {
        in[v] ++
    }
}
```



o/p → 1 7 2 4 3 5 6

2) Store nodes with indegree 0 in set / array / list.

3) Select any element from the list, print that data & update indegree of

all adjacent nodes by -1.

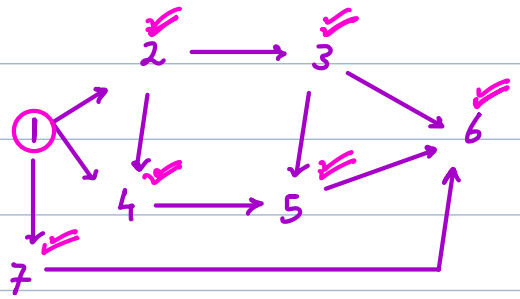
4) If any new node updates indegree to 0, insert that in the list.

5) Repeat from step 3 tell all nodes are travelled.

TC = O(N+E) SC = O(N)

Right to Left (DFS)

Using DFS, once all adjacent nodes are travelled, print the current node.



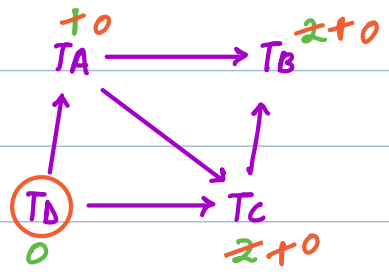
o/p → 6 5 3 4 2 7 1

```
∀i, vst[i] = false
for i → 1 to N {
   if (!vst[i]) dfs(i)
}
```

```
void dfs(u) {
   vst[u] = true
   for (v : Adj[u]) {
      if (!vst[v]) dfs(v)
   }
   print(u)
}
```

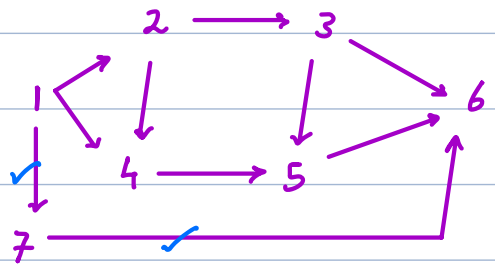
TC = O(N+E)

SC = O(N)



TD TA TC TB

Q → Find min # edges to travel from a source u to a destination v.



u = 1 v = 6 Ans = 2

start BFS from 'u'

Ans = level of 'v'