current PSP 52.44 → 58%

Nov23_PSP_6May
Gobika K
Vijay V A
Piyush Kumar
kameswarreddy Yeddula
Sawan
Sai Sharath
Harshil Dabhoya
Suraj Devraye
Rajeev
Yash Malviya
Manjunatha I
Kevin Theodore E

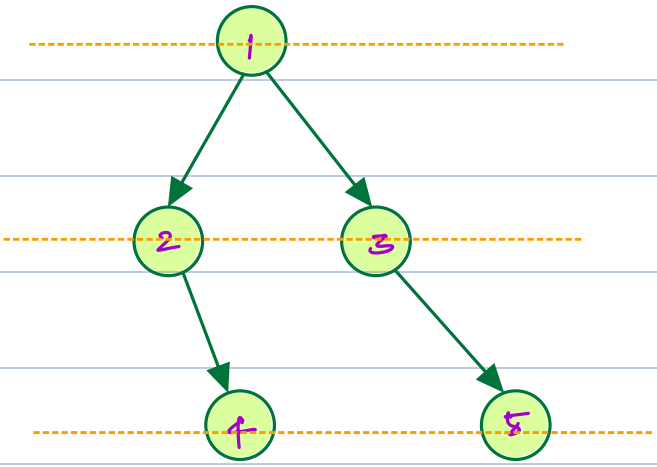
Nov23_PSP_6May
sudhakar venkatachalam
Manoj Vijayakumar
kumkum
Mohammad Mateen
Vigneshwaran K
MD JASHIMUDDIN
manikandan m
Prashant Kumar Soni
Pradeep Kumar Chandra
Shaurya Srivastava
Mohammed Arshad
Mayur Hadawale

Agenda :

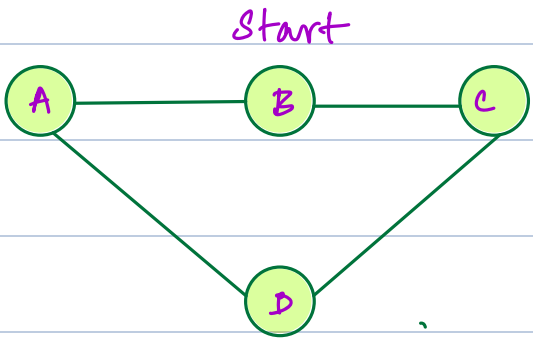
1. BFS
2. Multisource BFS
3. Rotten Oranges
4. Possibility of finishing courses
5. Topological sort

BFS

Still need visited array



{1, 2, 3, 4, 5}



~~B~~ ~~A~~ ~~C~~ ~~D~~

Queue

{B, A, C, D}

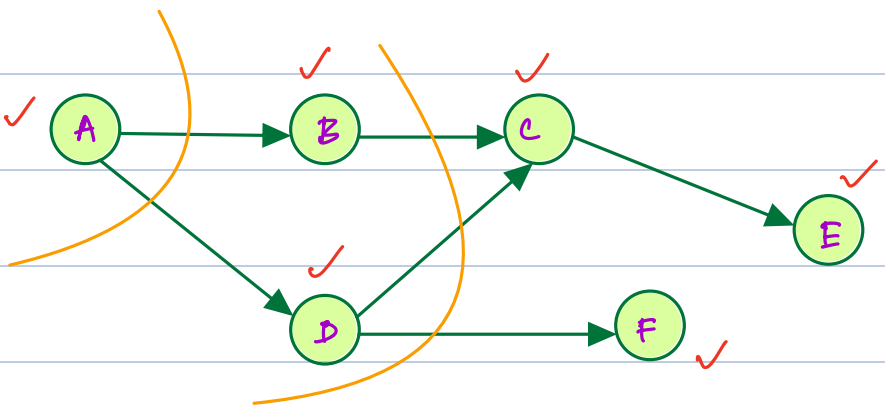
Steps

① Initialize an empty Queue & Insert the start node

② while Q is not empty → Dequeue front

→ add neighbours of front if ! visited

→ mark nei as visited



~~A~~ ~~B~~ ~~D~~ ~~C~~ ~~F~~ ~~E~~

pseudo code

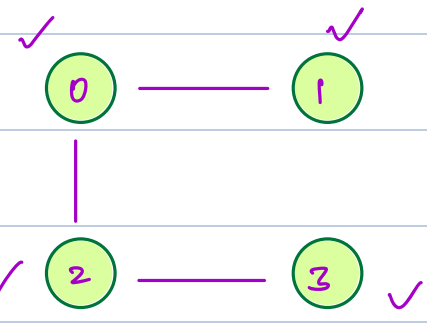
```
// Initialize empty Q. get start node
Queue.enqueue(node);
visited[node] = True
while(! queue.isEmpty()){
    n = queue.dequeue;
    print(n);
    for(nei : graph[n]){
        if(!visited[nei]){
            queue.enqueue(nei);
            visited[nei] = true;
        }
    }
}
```

T.C = O(n+e)

S.C = O(n)

Quiz 1

	0	1	2	3
0	0	1	1	0
1	1	0	0	0
2	1	0	0	1
3	0	0	1	0

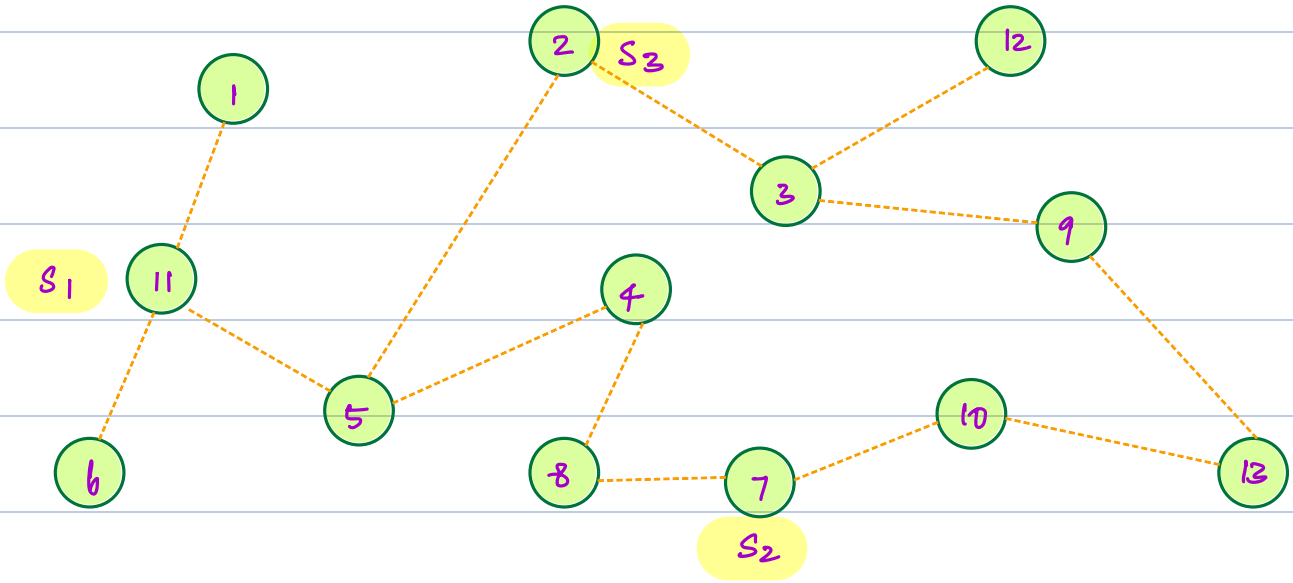


{0, 1, 2, 3}

{0, 2, 1, 3}

Multisource BFS * * * * *

There are N no. of nodes and multi source {S1, S2, S3}
Find the length of shortest path for given destination to any one of the sources



Dest	Min L
9	2
13	2
5	1

Observation

Add the sources to the queue with distance = 0

~~(0,11)~~ ~~(0,7)~~ ~~(0,2)~~ ~~(1,1)~~ ~~(1,5)~~ ~~(1,6)~~ ~~(1,8)~~ ~~(1,10)~~ (1,3) (2,4) (2,13) (2,9) (2,12)

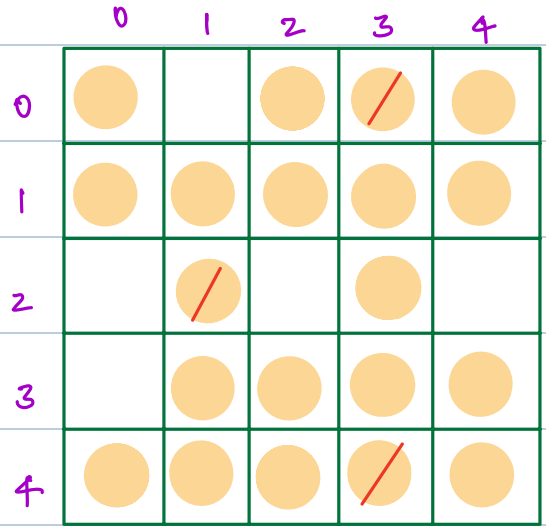
BFS will give the shortest path with a Big condition

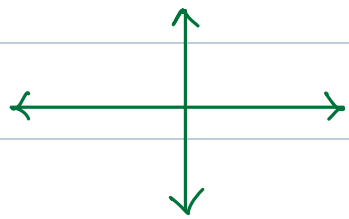
end of notes

Rotten Oranges ***** (Amazon)

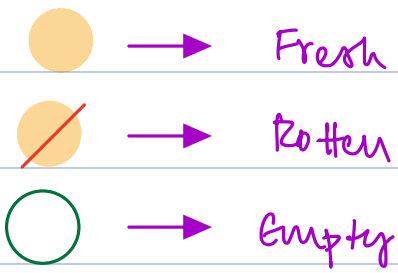
Given a matrix [R][C] with 3 values 0, 1, 2 {0 - Empty, 1 - Fresh, 2 - Rotten} Find the earliest time when all become rotten

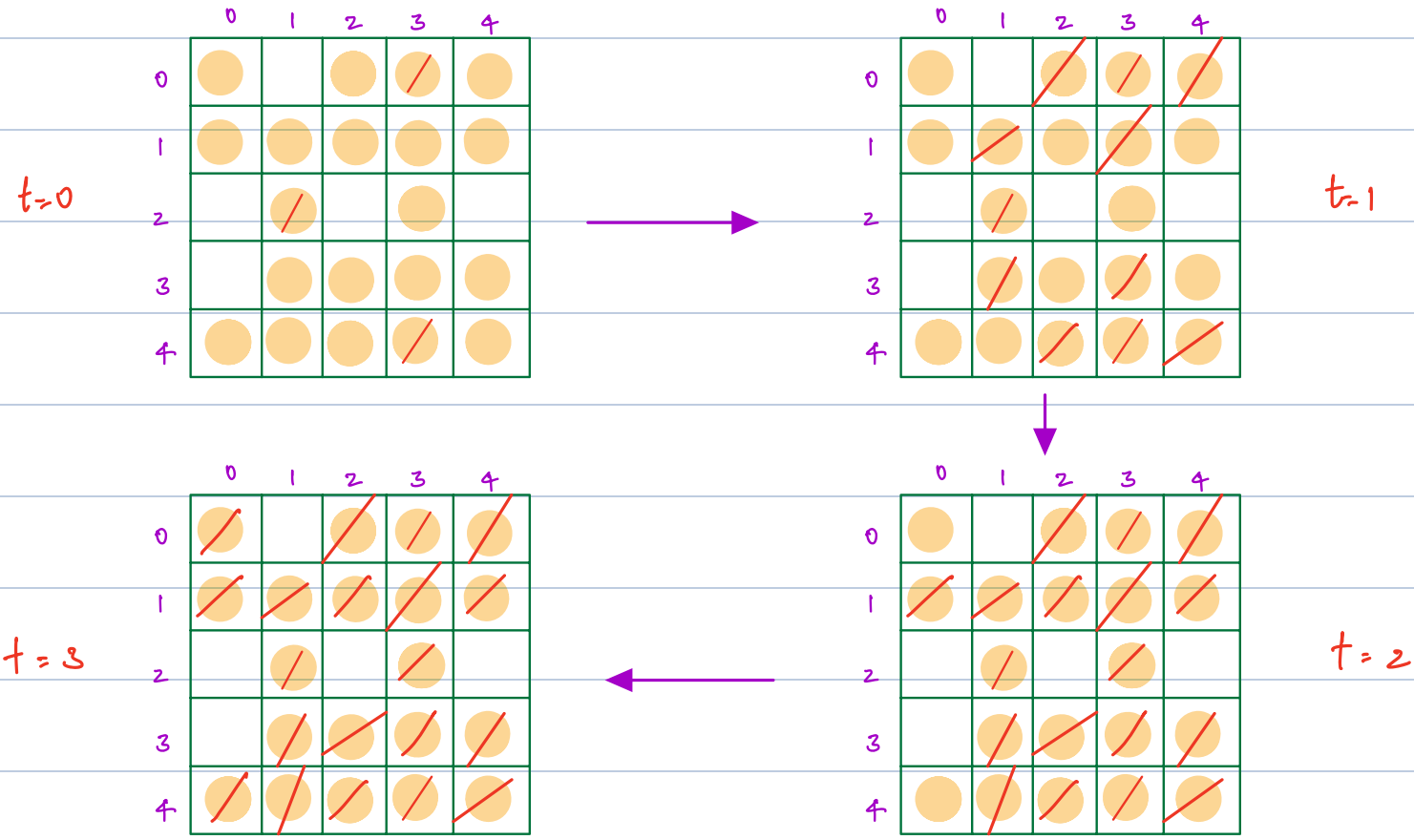
Note: Return -1 if not possible



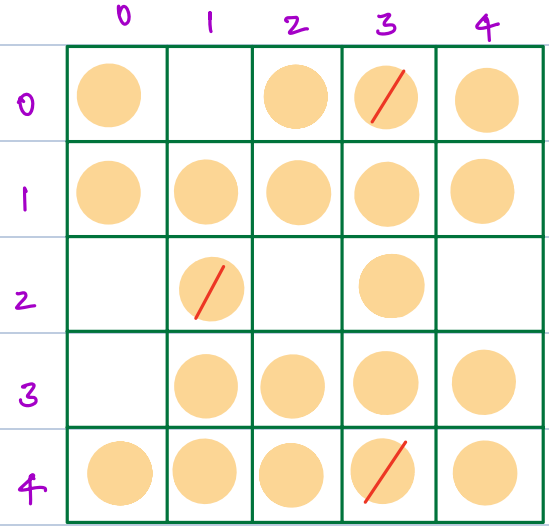


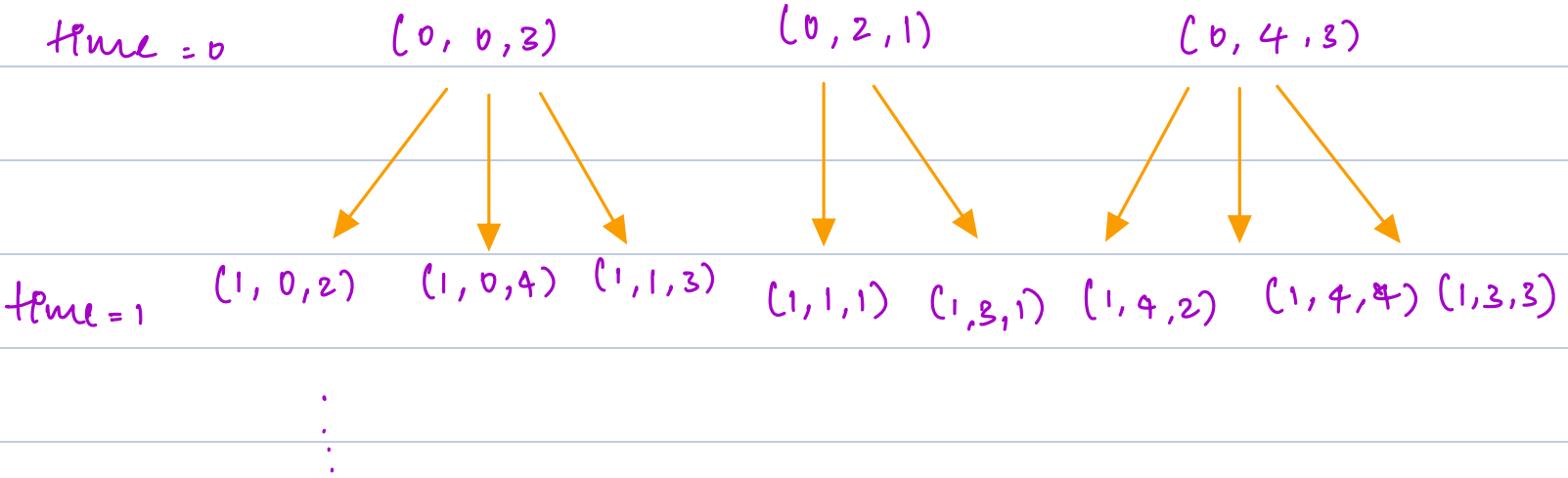
valid directions





Visualize





pseudo code

```
// Initialize queue
min time = 0;
for (r=0; r<R; r++)
    for (c=0; c<C; c++)
        if (A[r][c] == ROTTEN)
            queue.append ((0, r, c))
// Initialize directions
DX = [0, -1, 0, 1]
```

```
Dy = [1, 0, -1, 0]

while (! queue.isEmpty()) {
    time, r, c = queue.dequeue()
    min_time = time;
    for (i=0; i<4; i++) {
        nr = r + Dx[i];
        nc = c + Dy[i]
        // check if nr, nc is in range
        // 0 <= nr <= R   0 <= nc <= C
        if (A[nr][nc] == FRESH) {
            A[nr][nc] = ROTTEN;
            queue.enqueue((time+1, nr, nc))
        }
    }
}

// check for fresh oranges
for (r=0; r<R; r++)
    for (c=0; c<C; c++)
        if (A[r][c] == FRESH)
            return -1;

return min_time
```

T.C = R*C

S.C = R*C

Break : 10:30 pm

Flipkart Grocery has several warehouses spread across the country and in order to minimize the delivery cost, whenever an order is placed we try to deliver the order from the nearest warehouse.

Therefore, each Warehouse is responsible for a certain number of localities which are closest to it for deliveries, this **minimizes the overall cost for deliveries**, effectively managing the distribution workload and minimizing the overall delivery expenses.

Problem statement:-

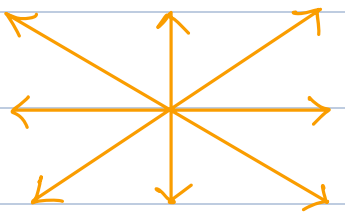
You are given a 2D matrix **A** of size **NxM** representing the map, where each cell is marked with either a **0 or a 1**. Here, a **0** denotes a locality, and a **1** signifies a warehouse. The objective is to calculate a new **2D matrix** of the same dimensions as **A**.

In this new matrix, the value of each cell will represent the minimum distance to the nearest warehouse. For the purpose of distance calculation, you are allowed to move to any of the **eight adjacent cells** directly surrounding a given cell.

Input :

1	0	0
0	0	0
0	0	0

0	1	2
1	1	2
2	2	2



8 Directions

Input :

1	0	0
0	0	0
0	0	1

0	1	2
1	1	1
	1	0

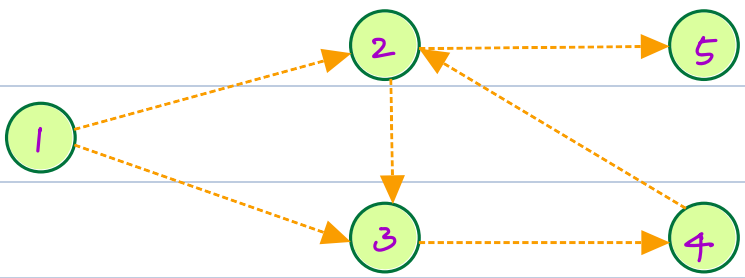
Possibility of finishing courses

Given N courses with pre-requisites

Check if it is possible to finish all the courses

Course	P.R for
1	{2, 3}
2	{3, 5}
3	{4}
4	{2}

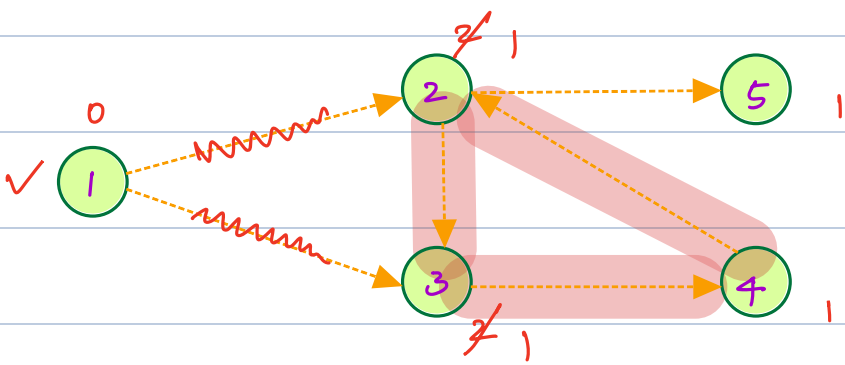
→ {1 is pre-req for 2 & 3}



Where will you begin from?

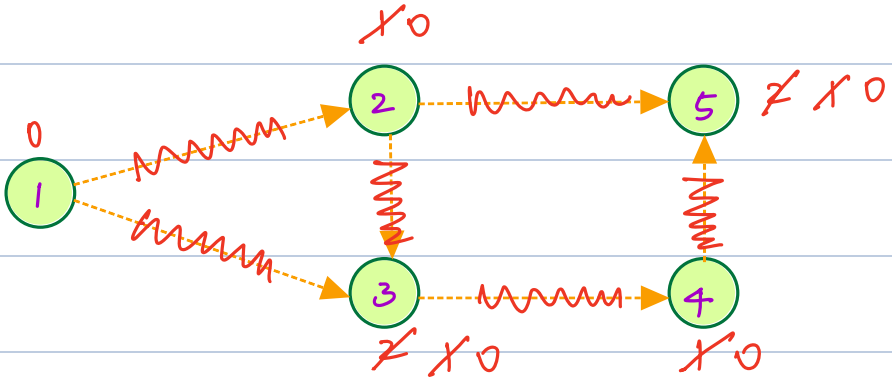
Course without pre-requisite

Indegree = 0



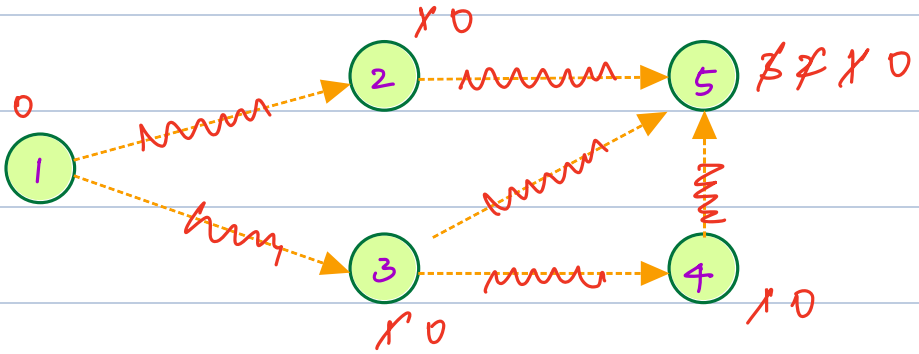
Deadlock because we have a cycle

No



1 2 3 4 5

Topological order



1 2 3 4 5

Is the order {1, 2, 3, 4, 5} valid? Yes

{1, 3, 2, 4, 5} valid? Yes.

{1, 3, 4, 2, 5} valid? Yes

Note: there can be multiple ways to complete the course

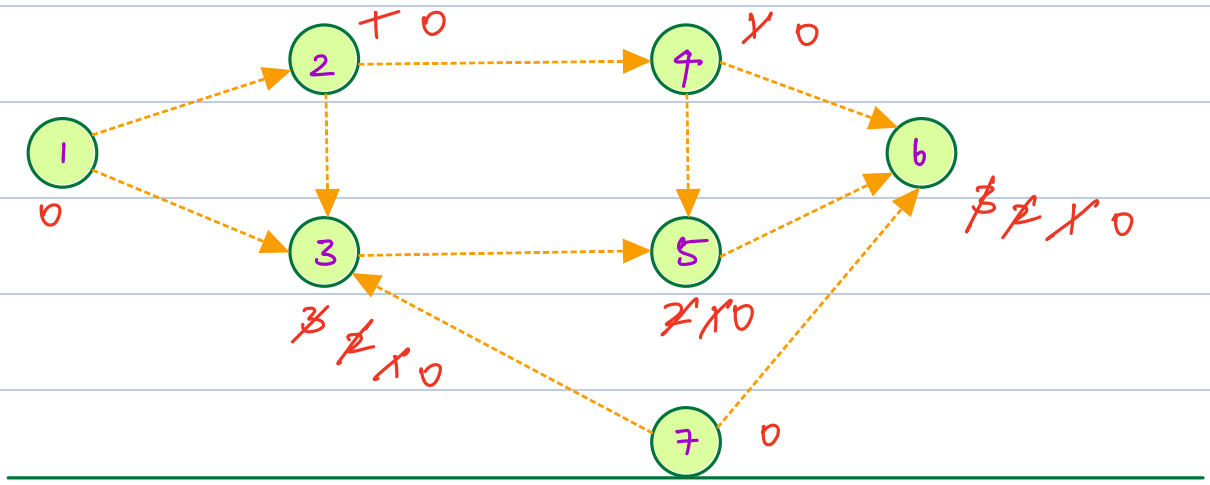
Topological sort

Works on directed acyclic graphs (DAG) only

If we have an edge $u \longrightarrow v$

u will come before v in topological order

Topological sort for below graph



~~(0,1)~~ ~~(0,7)~~ ~~(0,2)~~ ~~(0,3)~~ ~~(0,4)~~ ~~(0,5)~~ ~~(0,6)~~

Step 1: Calculate indegree for all nodes graph : al

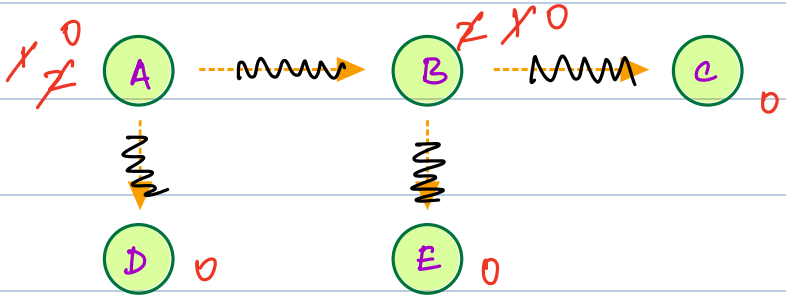
```
in [N] = {0, 0, 0 ...}

for (i=0; i<n; i++)
    for (nei : graph(i))
        in [nei] += 1 ;
```

Step 2: Put all nodes with indegree == 0 into queue

Step 3: Remove front of queue

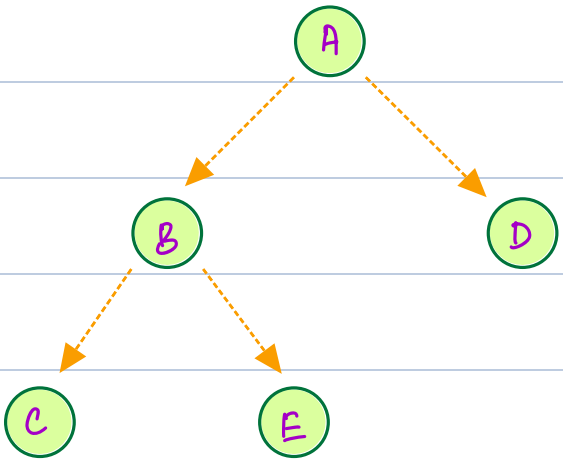
Go to all neighbours of node & mark in [nei] -= 1 ; if in [nei] == 0 → add to queue

Reverse topological sort (Right to left)



outdegree = 0

C E D B A



output

C E B D A

post order

To get the reverse of topological sort implement post order traversal

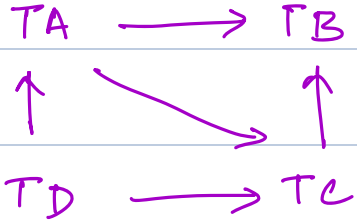
↓

DFS

pseudo code

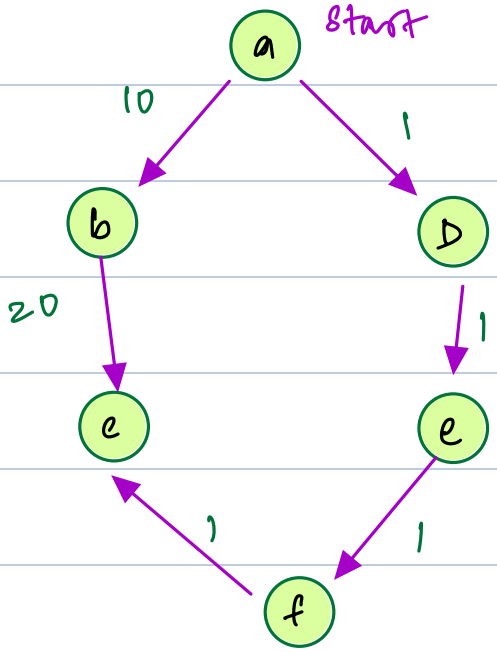
```
for (i = 0; i < n; i++)
    visited[i] = False;
for (j = 0; j < n; j++)
    if (! visited[j]) dfs(j)

void dfs(node) {
    // mark node as visited
    visited[node] = True
    // traverse all unvisited neighbours of node
    for (nei : graph.get(node)) {
        if (! visited[nei])
            dfs(nei);
    }
    print(node)
}
```

→ reverse topological sort



TD, TA, TC, TB

BFS [Additional Information]



shortest path to reach c

If we do not have any weights then BFS gives shortest path