Today's agenda

↳ BFS +!

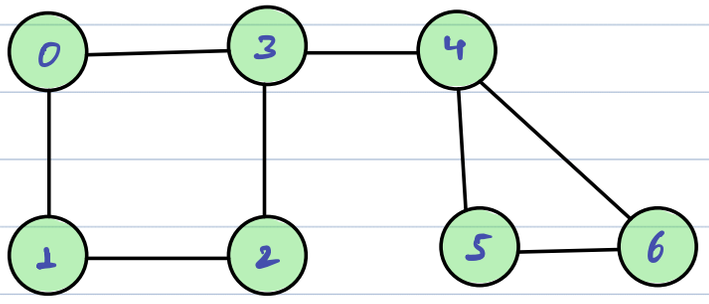
↳ Rotting oranges

Tree → Pre, Post & in

&

level order → BFS (Breadth first search)

BFS traversal



i=0

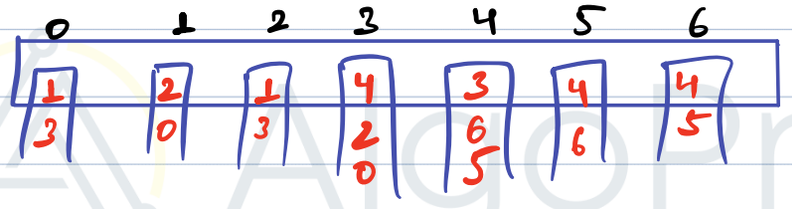
N = 7 → No. of nodes

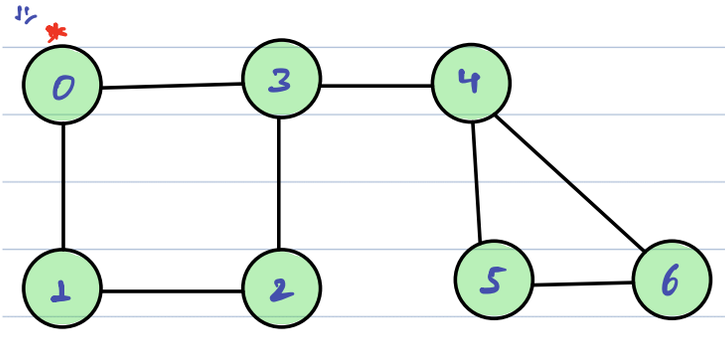
M = 8 → No. of edges

	0	1
0	3	4
1	1	2
2	2	3
3	4	6
4	0	1
	4	5
	5	6
	0	3

```
List<List<Integer>> graph = new ArrayList<>();
```

graph

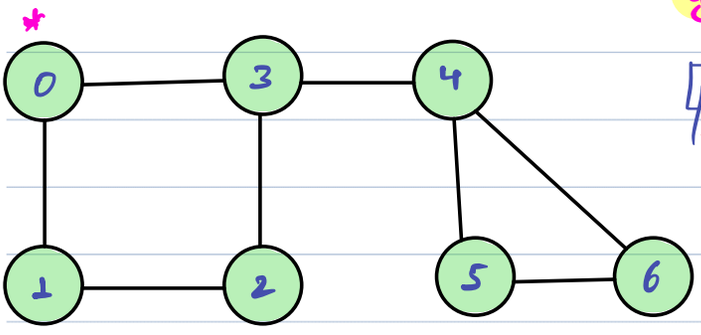




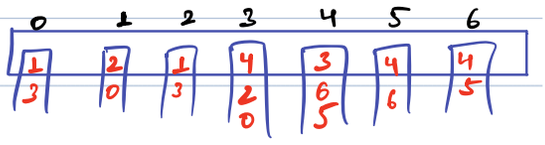
→ In graph traversal, you will have decide the start point.

b o (1 3) (2 4) (5 6)

BFS → level order in graph



graph



(0)(1 3)(2 4)(6 5)

queue: ~~0 1 3 2 4 6 5~~

vis :

0	1	2	3	4	5	6
T	T	T	T	T	T	T

rem = 5

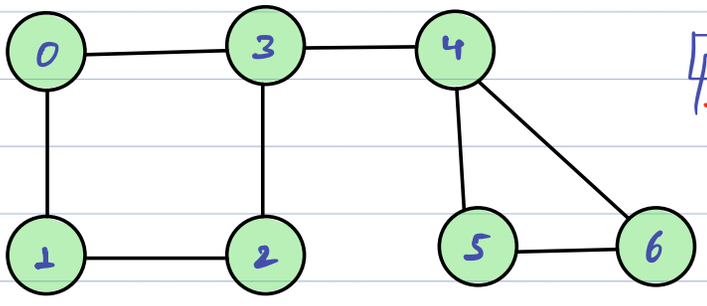
```
while (q.size()>0) {
    int rem = q.remove();
    S.O.p (rem);

    // add all unvisited nbrs

    List<Integer> nbrs = graph.get(rem);
    for (int v: nbrs) {
        if (vis[v] == false) {
            q.add(v);
            vis[v] = true;
        }
    }
}
```

// Pseudo code

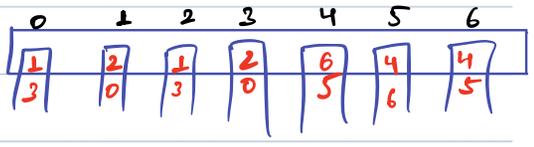
```
void BFS (int N, int m, int [][] edges){

    list<list<Integer>> graph = Construction(n,m,edges);

    Queue<Integer> q = new LinkedList<>();
    boolean[] vis = new boolean[N];

    q.add(0);
    vis[0] = true;

    while (q.size()>0) {
        int rem = q.remove();
        S.O.P (rem);

        // add all unvisited nbrs

        list<Integer> nbrs = graph.get(rem);
        for (int v: nbrs) {
            if (vis[v] == false) {
                q.add(v);
                vis[v] = true;
            }
        }
    }
}
```

Q) Given undirected graph, source node and destination node. Check if destination node can be visited from source node or not?

SN: 0 DN: 2



graph



↳ 1 disconnected graph

Idea

↳ Do BFS and Check if vis[dn] is true or not.

// Pseudo code

```
boolean BFS (int N, int m, int [][] edges, int SN, int DN){

    List<List<Integer>> graph = Construction(n,m,edges);

    Queue<Integer> q = new LinkedList<>();
    boolean[] vis = new boolean[N];

    q.add(SN);
    vis[0] = true;

    while (q.size()>0){
        int rem = q.remove();
        S.O.P(rem);

        // add all unvisited nbrs

        List<Integer> nbrs = graph.get(rem);
        for (int v: nbrs){
            if (vis[v] == false){
                q.add(v);
                vis[v] = true;
            }
        }
    }
    if (vis[DN] == true) { return true; }
    else { return false; }
}
```

T.C: O(N+m) (N: no. of nodes, m: no. of edges)

S.C: O(N)

Break till 9:10 AM

Q) Rotting oranges

mat[N][M]
- 0 {empty}
- 1 {fresh orange}
- 2 {rotten orange}

↳ Every minute any fresh orange adjacent to rotten orange becomes rotten. Find min time when all oranges become rotten. if not Possible to rott all oranges, return -1

Ex1:

	0	1	2	3	4
0	1²	0	1²	0	1²
1	1²	1²	1²	1²	1²
2	0	2	0	1²	0
3	0	1²	1²	1²	1²
4	1²	1²	1²	0	1²

T=0 ↓ T=1 ↓ T=2 ↓ T=3 ↓ T=4 ↓ T=5

Ex2:

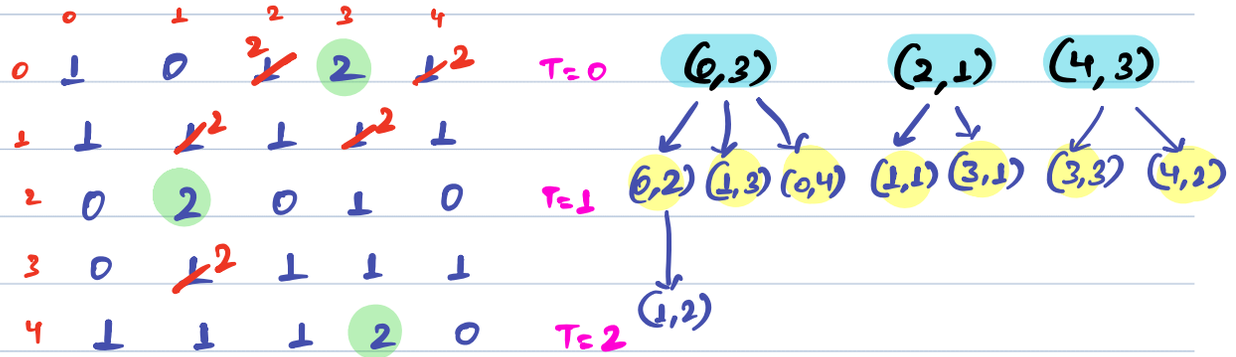
	0	1	2	3	4
0	1²	0	1²	2	1²
1	1²	1²	1²	1²	1²
2	0	2	0	1²	0
3	0	1²	1²	1²	1²
4	1²	1²	1²	2	0

T=0 ↓ T=1 ↓ T=2 ↓ T=3

Ex3: 1 0 2 → -1

0 0 2

Multisource BFS



q: {0,3,0} {2,1,0} {4,3,0} {0,2,1} {4,3,1} {0,4,1} {1,1,1} {3,1,1}

elm = {2,1,0}

```
Class Pair {
    int i;   → row
    int j;   → col
    int t;
    Pair(int x, int y, int z){
        i = x;
        j = y;
        t = z;
    }
}
```

//Pseudo code

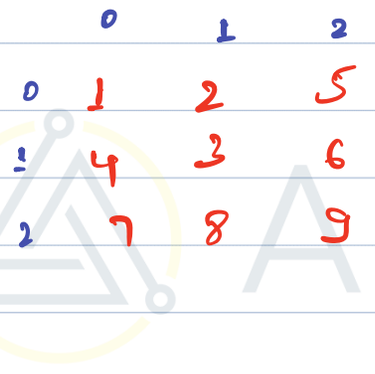
```
int rottingoranges(int mat[n][m]){

Queue<Pair> q = new LinkedList<>();

    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(mat[i][j]==2){
                Pair p = new Pair(i,j,0);
                q.add(p);
            }
        }
    }

int ans=-1;
    while(q.size()>0){
        Pair rem = q.remove();
        int crow = rem.i;
        int ccol = rem.j;
        int ctime = rem.t;
        ans = ctime;
        // crow-1, ccol
        if(crow-1>=0 && mat[crow-1][ccol]==1){
            q.add(new Pair(crow-1, ccol, ctime+1));
            mat[crow-1][ccol]=2;
        }
```

```
// cRow, cCol-1
if (cCol-1 >= 0 && mat[cRow][cCol-1] == 1){
    q.add(new Pair(cRow, cCol-1, cTime+1));
    mat[cRow][cCol-1] = 2;
}

// cRow+1, cCol
if (cRow+1 < N && mat[cRow+1][cCol] == 1){
    q.add(new Pair(cRow+1, cCol, cTime+1));
    mat[cRow+1][cCol] = 2;
}

// cRow, cCol+1
if (cCol+1 < m && mat[cRow][cCol+1] == 1){
    q.add(new Pair(cRow, cCol+1, cTime+1));
    mat[cRow][cCol+1] = 2;
}
}

for(int i=0; i<N; i++){
    for(int j=0; j<m; j++){
        if(mat[i][j] == 1){
            return -1;
        }
    }
}
```

T.C: O(N*m)

S.C: O(1) + O(N*m)

```
}
return ans;
}
```




```
ans=0
row=0  col=0
if(mat[row+1][col]<mat[row][col+1]){
    row++;
    ans += mat[row][col];
}
else{
    col++;
    ans += mat[row][col];
}
```