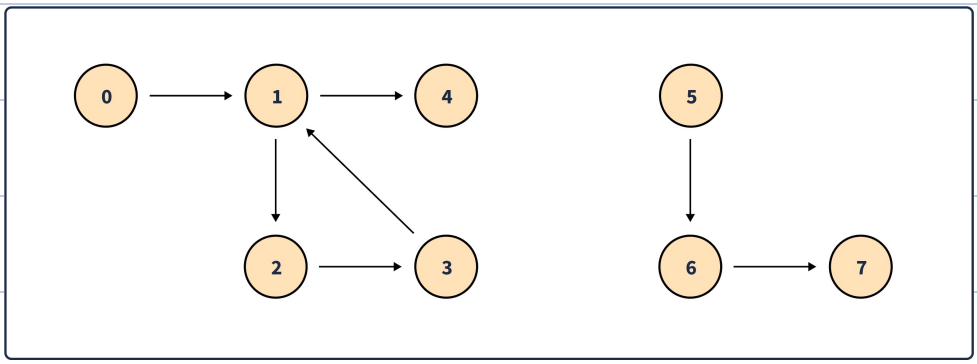
Agenda

1. BFS
2. Rotten Oranges
3. Number of Islands
4. Shortest Distance in a Maze

Breadth First Traversal (BFS)



0 → { 1 }

1 → { 2 , 4 }

2 → { 3 }

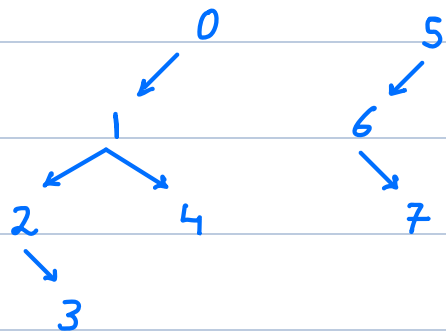
3 → { 1 }

4 → { - }

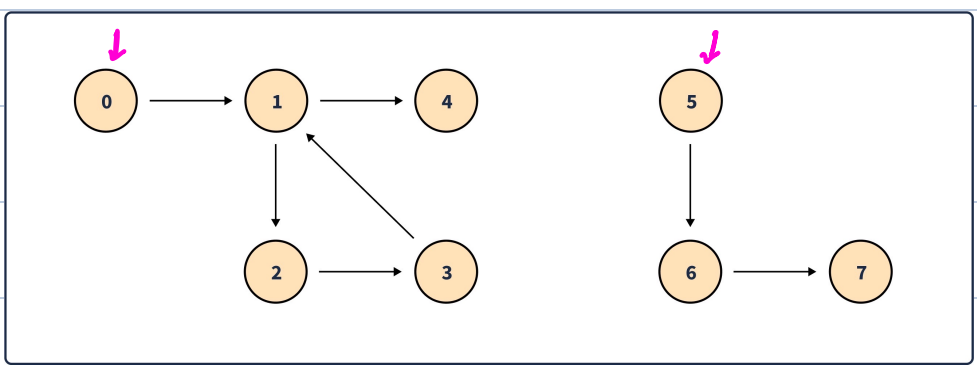
5 → { 6 }

6 → { 7 }

(Level Order Traversal) → Queues



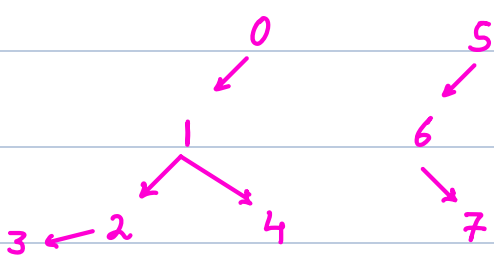
o/p → 0 1 2 4 3 5 6 7



0 1 2 3 4 5 6 7

vst = [T T T T T T T T]

Queue ↙ ~~0~~ ~~1~~ ~~2~~ ~~4~~ ~~3~~ ~~5~~ ~~6~~ ~~7~~



o/p → 0 1 2 4 3 5 6 7

(while inserting or removing)

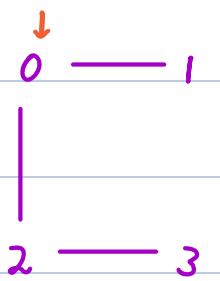
```
∀i, vst[i] = false
for i → 0 to (N-1) {
    if (!vst[i])  bfs(i)
}

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

TC = O(N + E)

SC = O(N) Queue / vst[]

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



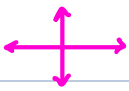
✓ 0 1 2 3 or

0 2 1 3

Rotten Oranges

There is given a matrix and there are 3 values where 0 means empty cell, 1 means fresh orange present and 2 means rotten orange present, we need to find the time when all oranges will become rotten.

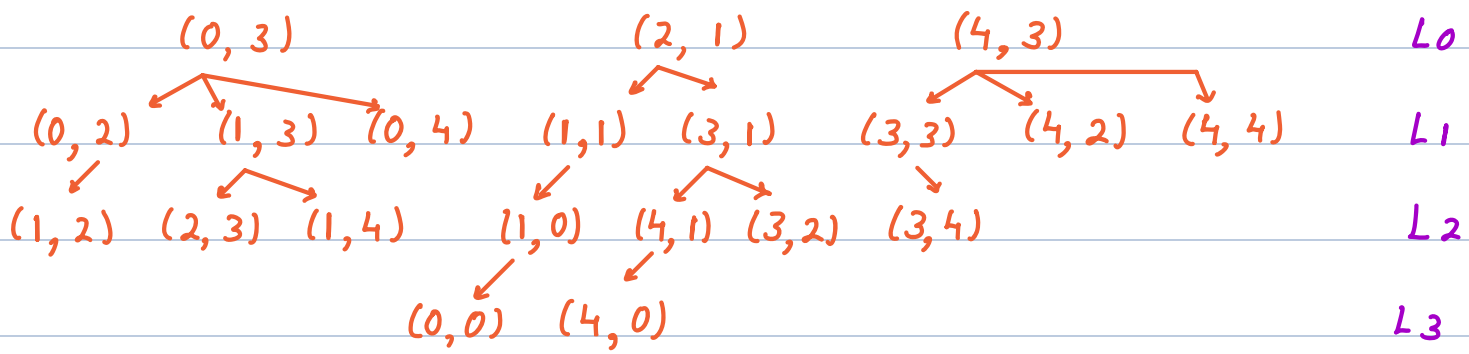
→ if that is not possible, return -1.



→ in every 1 minute, fresh orange adjacent to rotten orange becomes rotten.

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

	0	1	2	3	4
0	3	-1	1	0	1
1	2	1	2	1	2
2	-1	0	-1	2	-1
3	-1	1	2	1	2
4	3	2	1	0	1



Ans = 3 (∀ oranges, time >= 0)

↑ ← → ↓

dx = [-1 0 0 1]

dy = [0 -1 1 0]

∀i, j → T[i][j] = -1

```
for i → 0 to (N-1) {
    for j → 0 to (M-1) {
        if ( A[i][j] == 2 ) {   // Rotten orange
            T[i][j] = 0
            q.enqueue ( {i, j} )
        }
    }
}
```

```
while (! q.isEmpty ()) {
    r , c = q.dequeue ()
    for i → 0 to 3 {   ↔↕
        x = r + dx[i]        y = c + dy[i]
        if ( 0 <= x && x < N && 0 <= y && y < M
            && A[x][y] == 1 && T[x][y] == -1 ) {   // unvisited fresh orange
            T[x][y] = T[r][c] + 1
            q.enqueue (x, y)
        }
    }
}
```

```
ans = 0
for i → 0 to (N-1) {
    for j → 0 to (M-1) {
        if (A[i][j] == 1) {   // fresh orange
            if (T[i][j] == -1)  return -1  // unvisited
            else  ans = max(ans, T[i][j])
        }
    }
}
return ans
```

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

Flipkart Delivery Optimisation

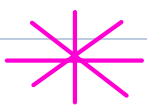
Flipkart Grocery has several warehouses spread across the country and in order to minimise the delivery cost, whenever an order is placed they try to deliver the order from the nearest warehouse.

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

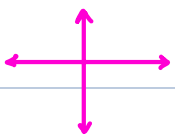
In this new matrix, the value of each cell will represent the minimum distance to the nearest warehouse. For the purpose of distance calculation, you are allowed to move to any of the eight adjacent cells directly surrounding a given cell.



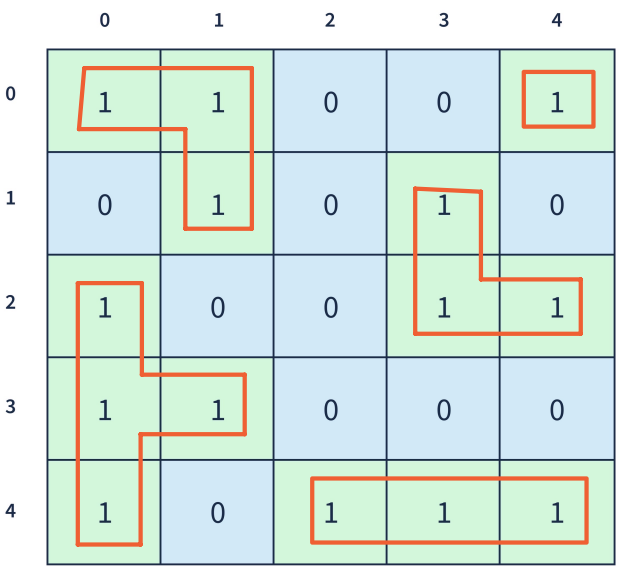
0 0 0 0 1 0 0 1 0
0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1

Number of Islands

You are given a 2D grid of '1's (land) and 'O's (water). Your task is to determine the number of islands in the grid. An island is formed by connecting adjacent (horizontally or vertically) land cells. Diagonal connections are not considered. Given here if the cell values has 1 then there is land and 0 if it is water, and you may assume all four edges of the grid are all surrounded by water.



arr[N][N]



Ans = 5

↑ ← → ↓

dx = [-1 0 0 1]

dy = [0 -1 1 0]

∀i, j → vst[i][j] = false

ans = 0

```
for i → 0 to (N-1) {
    for j → 0 to (M-1) {
        if ( A[i][j] == 1 && !vst[i][j]) { // unvisited land
```

```
            ans++
            dfs(i, j)
        }
    }
} return ans
```

```
void dfs(r, c) {
    vst[r][c] = true
    for i → 0 to 3 {
        x = r + dx[i]      y = c + dy[i]
        if (0 <= x && x < N && 0 <= y && y < M
            && A[x][y] == 1 && !vst[x][y])
                dfs(x)(y)
    }
}
```

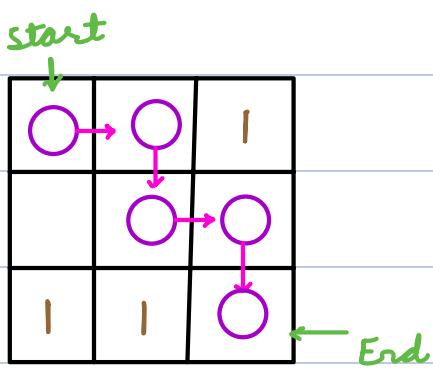
TC = $O(N * M)$ SC = $O(N * M)$

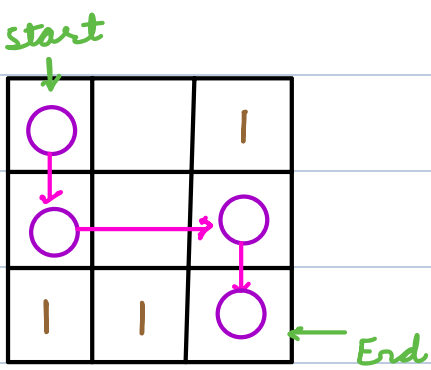
Shortest Distance in a maze

Given a matrix of integers A of size N x M describing a maze. The maze consists of empty locations and walls. 1 represents a wall in a matrix and 0 represents an empty location in a wall.

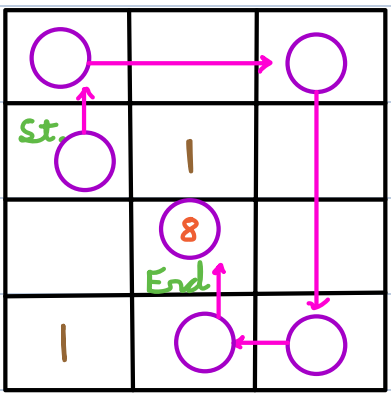
There is a ball trapped in a maze. The ball can go through empty spaces by rolling up, down, left or right, but it won't stop rolling until hitting a wall (maze boundary is also considered as a wall). When the ball stops, it could choose the next direction.

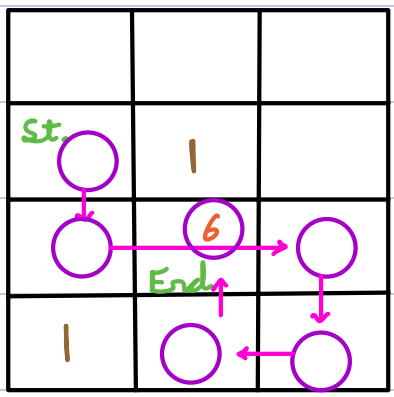
Given two array of integers of size B and C of size 2 denoting the starting and destination position of the ball. Find the shortest distance for the ball to stop at the destination. The distance is defined by the number of empty spaces traveled by the ball from the starting position (excluded) to the destination (included). If the ball cannot stop at the destination, return -1.



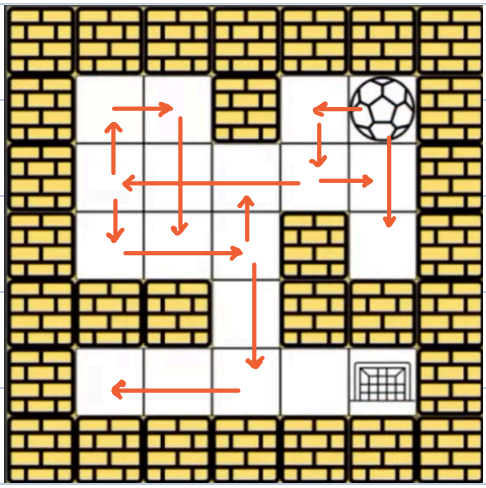


Ans = 4





Ans = 6



6	7		1	0 START
5	-1	9	2	3
6	9	8		2
		-1		
12	-1	10	-1	12 END

Distance

Ans = 12

Distance → BFS ✓

Path → DFS

↑ ← → ↓

dx = [-1 0 0 1]

dy = [0 -1 1 0]

∀ i, j → d[i][j] = -1 // Int Max

q.enqueue (start_x , start_y)

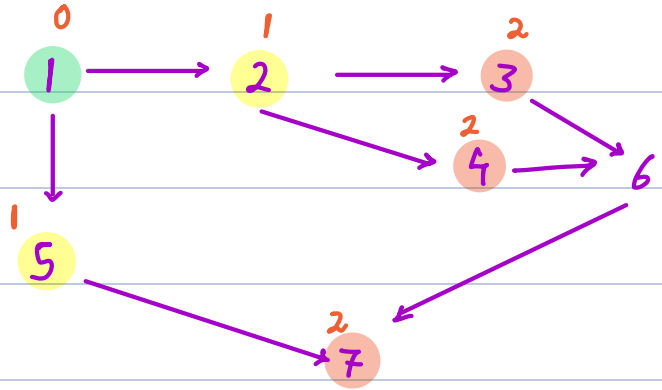
d [start_x] [start_y] = 0

```
while (! q.isEmpty ()) {
    r, c = q.dequeue ()
    for i → 0 to 3 {
        x = r   y = c   cnt = 0
        while ( 0 <= x && x < N && 0 <= y && y < M
                && A[x][y] == 0) { // Empty cell
```

```
            x += dx[i]    y += dy[i]
            crt++
        }
        x -= dx[i]   y -= dy[i]   crt--  // undo last step
        dist = d[r][c] + crt
        if (d[x][y] == -1) {  // unvisited
            d[x][y] = dist
            q.enqueue({x, y})
        }
    }
}
return d[end_x][end_y]
```

blocked

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



Find min #edges to travel from 1 to 7.

⇒ if root = 1, find level of 7

value

weight

index

dp[index][wt] = value

find min wt to make every profit.