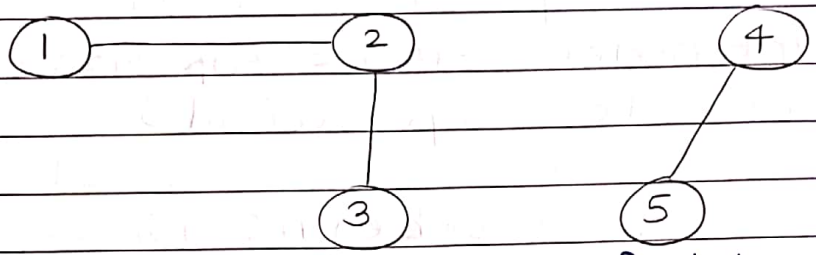
24/06/2023

Q1 Number of provinces (Leetcode 547)

Here we need to return number of components in a graph.



i/p →

1st province 2nd province

o/p → 2

Here we need to make a count variable and whenever the function call goes, simply do count ++.

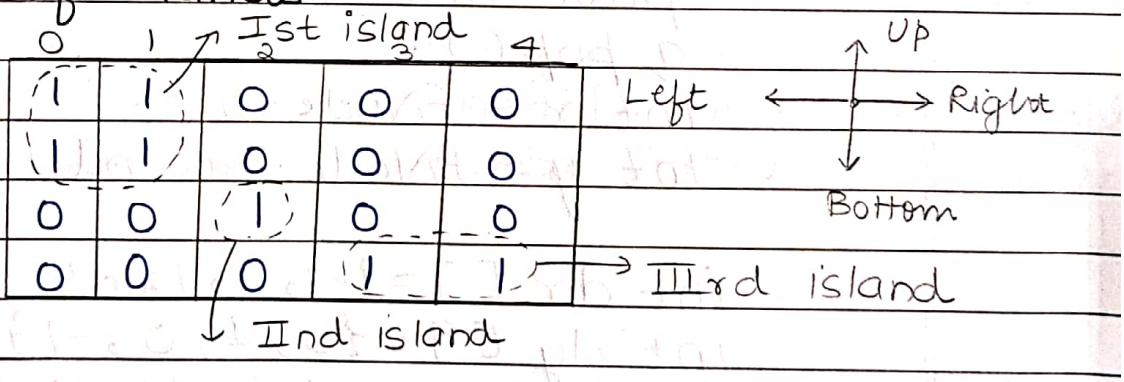
Code

```
void dfs (unordered_map <int, bool> &visited,
int src, vector <vector <int>> & isConnected ) {

    visited [src] = true;
    //row → source & col → loop
    int size = isConnected [src].size();
    for (int col = 0; col < size; col++) {
        //edge present or not
        if (isConnected [src][col] == 1) {
            //col is a neighbour
            if (!visited [col])
                dfs(visited, col, isConnected);
        }
    }
}

int findProvince (vector <vector <int>> &isConnected)
                                    ↳ Adjacency matrix
```

```
{
    unordered_map<int, bool> visited;
    int count = 0;
    int n = isConnected.size();
    for(int i=0; i<n; i++) {
        if(!visited[i]){
            dfs(visited, i, isConnected);
            count++; //Discussed in approach
        }
    }
    return count;
}
```

Time complexity = $O(n^2)$ {Adjacency matrix}

Q2 Number of islands



o/p → 3

queue → {<0,0>} → mark visited
Pop and insert neighbours
queue → {<0,1>, <1,0>} → mark visited
Pop and insert neighbours
queue → {<1,0>, <1,1>}
↳ mark visited
Pop and insert nbr
queue → {<1,1>}
Pop and insert nbr

queue → { 3
Now queue empty. Hence count++.

We need to do this for all components which will be handled via for loop.

Code

```
void bfs (vector <vector <char >>& grid,
map < pair <int, int >, bool > & visited, int
row, int col) {
    queue <pair <int, int>> q;
    q.push ({row, col});
    visited [{row, col}] = true;
    while (!q.empty()) {
        pair <int, int > fNode = q.front();
        q.pop();
        int x = fNode.first;
        int y = fNode.second;

        int dx [] = {-1, 0, 1, 0};
        int dy [] = {0, 1, 0, -1};
        for (int i = 0; i < 4; i++) {
            int newX = x + dx [i];
            int newY = y + dy [i];
            if (newX >= 0 && newX < grid.size()
&& newY >=0 && newY < grid [0].size() &&
!visited [{newX, newY}] && grid [newX]
[newY] == '1') {
            q.push ({newX, newY});
            visited [{newX, newY}] = true;
        }
    }
}
```

```
    }
  }
int numIslands(vector<vector<char>>& grid){
    map<pair<int,int>, bool> visited;
    int count=0;
    for(int row; row<grid.size(); row++){
        int n = grid[row].size();
        for(int col=0; col<n; col++){
            if(!visited[{row,col}] &&
              grid[row][col]=='1'){

                bfs(grid, visited, row, col);
                count++;
            }
        }
    }
    return count;
}
```

How dx[] and dy[] are created? → bottom

dx[] = {-1, 0, 1, 0} dy[] = {0, 1, 0, -1}

top ↓ right ↓ → left

(x-1, y)

(x, y-1) ← (x, y) → (x, y+1)

↓

(x+1, y)

Q3 Flood fill

i/p →		0	1	2
	0	1	1	1
	1	1	(1)	0
	2	1	0	1

sr = 1

sc = 1

color = 2

12	12 ↑2	12
12	12	0
12	0	1

We have changed 1 to 2, then only 1 will be changed to 2. We can go in 4 direxn and only to that node which is not visited.

Code

```
void dfs (int x, int y, int oldColor, int
newColor, map <pair <int, int>, bool > &
visited, vector <vector <int>> & ans) {

    visited [{x, y}] = true;
    ans [x] [y] = newColor;
    int dx [] = {-1, 0, 1, 0};
    int dy [] = {0, 1, 0, -1};
    for (int i = 0; i < 4; i++) {
        int newX = x + dx [i];
        int newY = y + dy [i];
        if (newX >= 0 && newX < ans.size()
    && newY >= 0 && newY < ans [0].size()
    && ! visited [{newX, newY}] && ans
    [newX] [newY] == oldColor) {

        dfs (newX, newY, oldColor, newColor,
        visited, ans);
    }
}
}
vector <vector <int>> floodfill (vector <vector
```

```
<int>> & image, int sr, int sc, int color) {
    int oldColor = image[sr][sc];
    map <pair <int, int>, bool> visited;
    vector < vector <int>> ans = image;
    dfs (sr, sc, oldColor, color, visited, ans);
    return ans;
}
```

Q4 Rotten oranges

	0	1	2
0	R	F	F
1	F	F	0
2	0	F	F

A single orange (rotten) will rot oranges in 4 direxn in 1 min

* orange at [0][0] will rot [1][0] and [0][1] in 1 min (ans = 1)

R	R	F
R	F	0
0	F	F

* orange at [1][0] rots orange at [1][1] and orange at [0][1] will rot orange at [0][2]. (ans = 2)

R	R	R
R	R	0
0	F	F

* orange at [2][1] will be rot by [1][1] in 1 min. (ans = 3 min)

R	R	R
R	R	0
0	R	F

* orange at [2][1] rots orange at [2][2] in 1 min.

(ans = 4 min)

All oranges are rot in 4 min.

Code

```
int orangesRotting (vector <vector <int>> & grid){
    queue <pair <pair <int, int >, int >> q;
    vector <vector <int>> ans = grid;
    int ansTime = 0;

    for (int row = 0 ; row < grid.size() ; row++){
      for (int col = 0 ; col < grid[row].size() ; col++){
          if (grid[row][col] == 2){
              pair <int, int> coordinate = make_pair
              (row, col);
              q.push ({coordinate, 0});
          }
      }
}
```

```
while (!q.empty())
    auto fNode = q.front();
    q.pop();
    int x = fNode.first.first;
    int y = fNode.          .second;
```

```
int time = fNode.second;
int dx[] = {-1, 0, 1, 0};
int dy[] = {0, 1, 0, -1};
for(int i=0; i<4; i++) {
    int newX = x + dx[i];
    int newY = y + dy[i];
    if(newX >=0 && newX < ans.size() &&
    newY >=0 && newY < ans[0].size() &&
    ans[newX][newY] == 1) {
        pair<int, int> newCoordinate = make_pair
        (newX, newY);
        ansTime = max(ansTime, time+1);
        q.push({newCoordinate, time+1});
        ans[newX][newY] = 2; //mark rotten
    }
  }
}

for(int i=0; i<ans.size(); i++) {
    for(int j=0; j<ans[i].size(); j++) {
        if(ans[i][j] == 1)
            return -1;
    }
}
return ansTime;
}
```