547. Number of Provinces

This is the problem of calculating number of connected components. We maintain a counter, an array that records whether the node has been visited and a queue that records neighbours of a node. We are given adjacency matrix and not list. This affects time complexity.

Algorithm

(i) Traverse for i from 1 to n. If i has not been visited, increase the counter. set visited[i] = true. Push i to queue.

(ii) Now, while queue is not empty (BFS), keep visiting the neighbours and add them to queue using adjacency matrix.

(iii) An empty queue means one connected component is traversed.

(iv) Return the final counter value.

Time complexity → worst case occurs when all the components are disconnected. complexity for that case becomes $O(n^2)$.

space complexity → O(n), visited array and stack / queue.

we can use DFS / BFS both for this problem, below implementation uses BFS

CODE

```
int numProvinces(vector<vector<int>>& isConnected){
    int n = isConnected.size();
    vector<bool> visited(n, false);
    queue<int> q;    int counter = 0;

    for(int i=0; i<n; i++){
        if(!visited[i]){
            q.push(i);
            visited[i] = true;
            counter++;
        }
        while(!q.empty()){
            int node = q.front();
            q.pop()
            for(int j=0; j<n; j++){
                if(node != j && isConnected[node][j]
                    && visited[j] == false){

                    q.push(j);
                    visited[j] = true;

                }
            }
        }
    }
    return counter;
}
```