1829E - The Lakes

Idea: **mesanu**

Tutorial

1829E - The Lakes

We can approach this problem using Depth First Search (DFS) or Breadth First Search (BFS) on the given grid.

The idea is to consider each cell of the grid as a potential starting point for a lake, and explore all the cells reachable from it by only moving up, down, left or right, without stepping on any cell with depth 0. If we reach a dead end, or a cell with depth 0, we backtrack and try another direction.

During this exploration, we keep track of the sum of depths of all the cells that we have visited. This sum gives us the volume of the current lake. When we have explored all the reachable cells from a starting point, we compare the volume of this lake with the maximum volume found so far, and update the maximum if necessary.

To implement this approach, we can use a nested loop to iterate through all the cells of the grid. For each cell, we check if its depth is greater than 0, and if it has not already been visited in a previous lake. If these conditions are satisfied, we start a DFS/BFS from this cell, and update the maximum volume found so far. See the implementation for more details.

The time complexity is $O(mn)$.

Solution

```cpp
#include <bits/stdc++.h>
#define startt ios_base::sync_with_stdio(false);cin.tie(0);
typedef long long  ll;
using namespace std;
#define vint vector<int>
#define all(v) v.begin(), v.end()
#define MOD 1000000007
#define MOD2 998244353
#define MX 1000000000
#define MXL 1000000000000000000
#define PI (ld)2*acos(0.0)
#define pb push_back
#define sc second
#define fr first
#define endl '\n'
#define ld long double
#define NO cout << "NO" << endl
#define YES cout << "YES" << endl

int n, m;
bool vis[1005][1005];
int a[1005][1005];

int dfs(int i, int j)
{
    vis[i][j] = true;
    int ans = a[i][j];
    if(i != 0 && a[i-1][j] != 0 && !vis[i-1][j])
    {
        ans+=dfs(i-1, j);
    }
    if(i != n-1 && a[i+1][j] != 0 && !vis[i+1][j])
    {
        ans+=dfs(i+1, j);
    }
    if(j != 0 && a[i][j-1] != 0 && !vis[i][j-1])
    {
        ans+=dfs(i, j-1);
    }
    if(j != m-1 && a[i][j+1] != 0 && !vis[i][j+1])
    {
        ans+=dfs(i, j+1);
    }
    return ans;
}

void solve()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            vis[i][j] = false;
            cin >> a[i][j];
        }
    }
    int ans = 0;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(!vis[i][j] && a[i][j] != 0)
            {
                ans = max(ans, dfs(i, j));
            }
        }
    }
    cout << ans << endl;
}

int32_t main(){
    startt
    int t = 1;
    cin >> t;
    while (t--) {
        solve();
    }
}
```