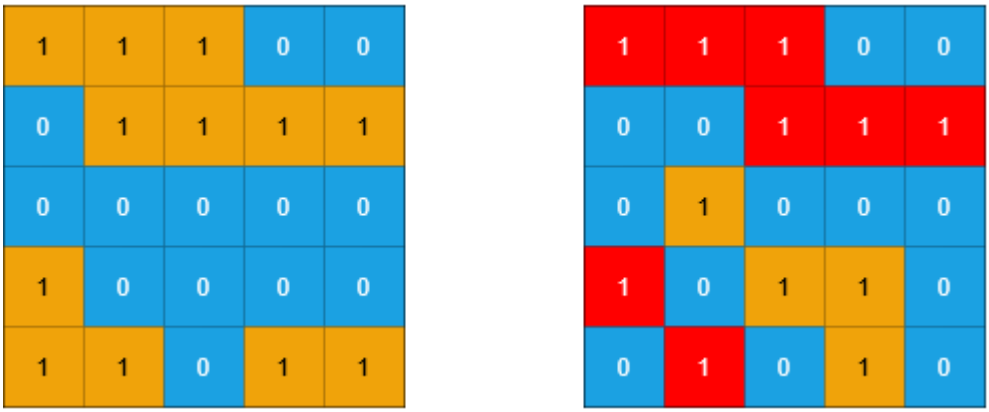
1905. Count Sub Islands

You are given two `m x n` binary matrices `grid1` and `grid2` containing only `0`'s (representing water) and `1`'s (representing land). An **island** is a group of `1`'s connected **4-directionally** (horizontal or vertical). Any cells outside of the grid are considered water cells.

An island in `grid2` is considered a **sub-island** if there is an island in `grid1` that contains **all** the cells that make up **this** island in `grid2`.

Return the ***number*** *of islands in* `grid2` *that are considered* ***sub-islands***.

Example 1:

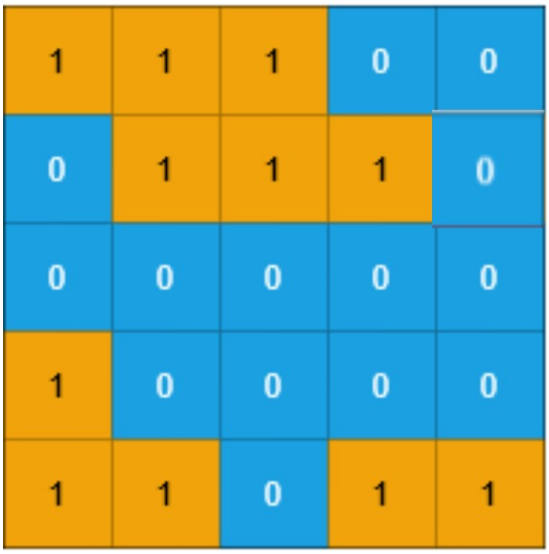


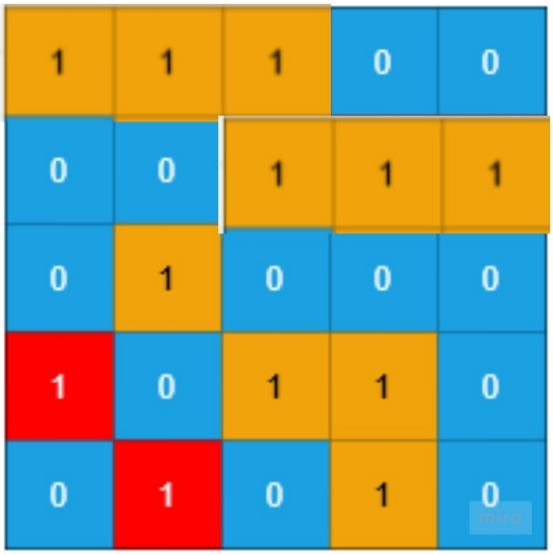
Input:
grid1 = [[1,1,1,0,0],[0,1,1,1,1],[0,0,0,0,0],[1,0,0,0,0],[1,1,0,1,1]]
grid2 = [[1,1,1,0,0],[0,0,1,1,1],[0,1,0,0,0],[1,0,1,1,0],[0,1,0,1,0]]

Output: 3

Explanation: In the picture above, the grid on the left is grid1 and the grid on the right is grid2.
The 1s colored red in grid2 are those considered to be part of a sub-island. There are three sub-islands.

Example 2:

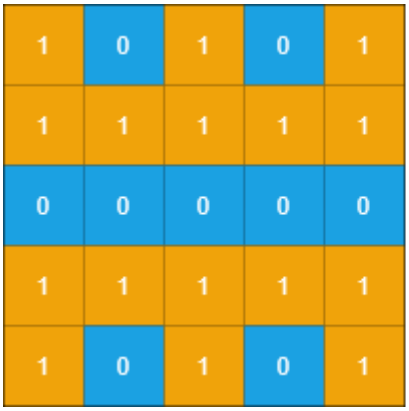


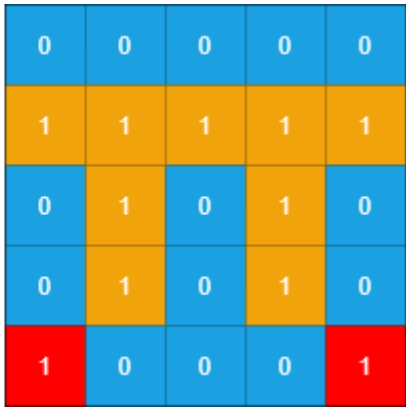


Input:
grid1 = [[1,1,1,0,0],[0,1,1,1,0],[0,0,0,0,0],[1,0,0,0,0],[1,1,0,1,1]]
grid2 = [[1,1,1,0,0],[0,0,1,1,1],[0,1,0,0,0],[1,0,1,1,0],[0,1,0,1,0]]

Output: 2

Example 3:





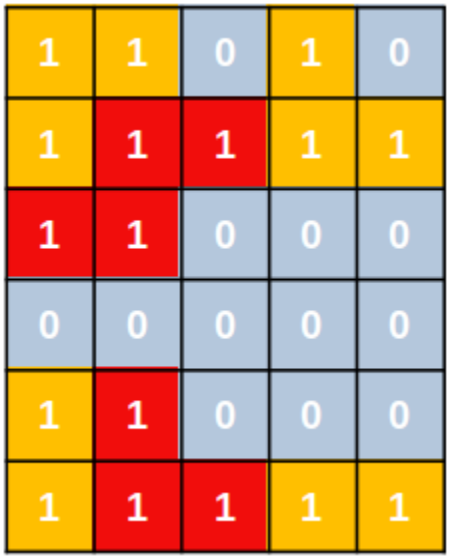
Input:
grid1 = [[1,0,1,0,1],[1,1,1,1,1],[0,0,0,0,0],[1,1,1,1,1],[1,0,1,0,1]]

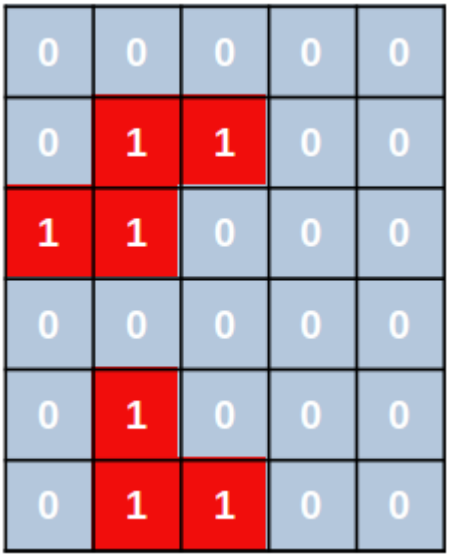
grid2 = [[0,0,0,0,0],[1,1,1,1,1],[0,1,0,1,0],[0,1,0,1,0],[1,0,0,0,1]]

Output: 2

Explanation: In the picture above, the grid on the left is grid1 and the grid on the right is grid2.
The 1s colored red in grid2 are those considered to be part of a sub-island. There are two sub-islands.

Example 4:





Input:
grid1 = [[1,1,0,1,0],[1,1,1,1,1],[1,1,0,0,0],[0,0,0,0,0],[1,1,0,0,0],[1,1,1,1,1]]

grid2 = [[0,0,0,0,0],[0,1,1,0,0],[1,1,0,0,0],[0,0,0,0,0],[0,1,0,0,0],[0,1,1,0,0]]

Output: 2

Constraints:

- `m == grid1.length == grid2.length`
- `n == grid1[i].length == grid2[i].length`
- `1 <= m, n <= 500`
- `grid1[i][j]` and `grid2[i][j]` are either `0` or `1`.

1905. Count Sub Islands

```
/*
    DFS
    Time complexity:
        nested for loops: O(mn)
        DFS: O(V+E)=O(mn+V)
        Overall: O(mn*(mn+V))
        If all cells are 1:
        nested for loops: O(1)
        DFS: O(V+E)=O(mn+V)
        Overall: O(mn+V)=O(mn)
    Space complexity:
        Recurion stack calls: O(mn)
*/
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;
typedef vector<int> vi;
typedef vector<vi> vvi;
class Solution {
    public:
        vii moves={
            {-1,0}, // Up
            {1,0}, // Down
            {0,1}, // Right
            {0,-1} // Left
        };

        int m; // Number of rows
        int n; // Number of columns

    public:
        bool is_in_grid(int x,int y){
            return x>=0 && x<=m-1 && y>=0 && y<=n-1;
        }

        bool dfs(int i, int j,vvi& grid1, vvi& grid2){
            // If out of grid or visited or cell in grid2 is water
            if(!is_in_grid(i,j) || grid2[i][j]==0) return true;

            grid2[i][j]=0; // Mark cell as visited

            bool is_sub_island=grid1[i][j]==1;

            for(auto& move: moves){
                int x=i+move.first;
                int y=j+move.second;
                is_sub_island = dfs(x,y,grid1,grid2) && is_sub_island ;
            }
            return is_sub_island;
        }
```

```
        int countSubIslands(vvi& grid1, vvi& grid2) {
            m=grid1.size();
            n=grid1[0].size();

            int ans=0;
            for(int i=0;i<m;++i){
                for(int j=0;j<n;++j){
                    if(grid1[i][j]==0 || grid2[i][j]==0) continue;
                    ans+=dfs(i,j,grid1,grid2);
                }
            }
            return ans;
        }
};
```

1905. Count Sub Islands

```
/*
    Union Find
    Time complexity:
        nested for loops: O(mn)
        DSU: O(α(mn))
        Overall: O(α(mn)mn),
        where α is the inverse Ackermann function
    Space complexity:
        Recurion stack calls: O(mn)
*/

typedef vector<int> vi;
typedef vector<vi> vvi;

class DSU{
    public:
        vi parent;
    public:
        DSU(int _N){
            parent.resize(_N);
            for(int i=0;i<_N;++i) parent[i]=i;
        }

        int find(int p){
            if(p==parent[p]) return p;
            return parent[p]=find(parent[p]); // Path compression
        }

        int unify(int p,int q){
            int parent_p=find(p);
            int parent_q=find(q);
            if(parent_p==parent_q) return 0;
            parent[parent_q]=parent_p;
            return 1;
        }
};
```

```
class Solution {
    public:
        int m; // Number of rows
        int n; // Number of columns

    public:
        int cell_id(int i,int j){return i*n+j;}

        int countSubIslands(vvi& grid1, vvi& grid2) {
            m=grid1.size();
            n=grid1[0].size();

            int _N=m*n;

            //+1, water source to flood non valid sub-island in grid2
            DSU dsu=DSU(_N+1);

            int ans=0;
            for(int i=0;i<m;++i){
                for(int j=0;j<n;++j){

                    if(grid2[i][j]==0) continue;

                    int cur=cell_id(i,j);
                    int down=cell_id(i+1,j);
                    int right=cell_id(i,j+1);

                    ans++;

                    // Flood the current island in grid2 by water
                    if(grid1[i][j]==0) ans-=dsu.unify(cur,_N);

                     // Count the number of islands in grid2
                    if(i+1<m && grid2[i+1][j]==1) ans-=dsu.unify(cur,down);
                    if(j+1<n && grid2[i][j+1]==1) ans-=dsu.unify(cur,right);
                }
            }

            return ans;
        }
};
```