



Floodfill

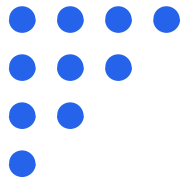
<https://usaco.guide/silver/ff>



CP Initiative
joincpi.org

Introduction to Floodfill

- Floodfill is similar to dfs as it traverses a connected component while prioritizing depth
- Applied on grids where adjacent cells are connected
- Grids are implicit graphs since each cell can be thought of as a node and the edges are between adjacent cells

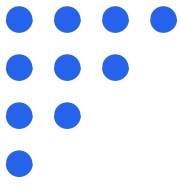


Implementation (C++)

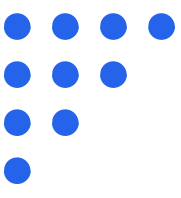
```
int grid[MAXN][MAXM]; // the grid itself
int n, m; // grid dimensions, rows and columns
bool visited[MAXN][MAXM]; // keeps track of which nodes have been visited

void floodfill(int r, int c, int color){
    if(r < 0 || r >= n || c < 0 || c >= m) return; // if outside grid
    if(grid[r][c] != color) return; // wrong color
    if(visited[r][c]) return; // already visited this square
    visited[r][c] = true; // mark current square as visited
    // recursively call floodfill for neighboring squares
    floodfill(r, c+1, color);
    floodfill(r, c-1, color);
    floodfill(r-1, c, color);
    floodfill(r+1, c, color);
}

int main(){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            if(!visited[i][j]){ //floodfill in every unvisited cell
                floodfill(i, j, grid[i][j]);
            }
        }
    }
}
```



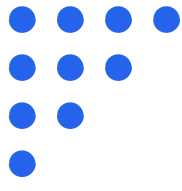
Example Problem



- You are given a map of a building, and your task is to count the number of its rooms. The size of the map is $n \times m$ squares, and each square is either floor or wall. You can walk left, right, up, and down through the floor squares.
 - Input
 - The first input line has two integers n and m : the height and width of the map.
 - Then there are n lines of m characters describing the map. Each character is either `.` (floor) or `#` (wall).
 - Output
 - Print one integer: the number of rooms.
 - Constraints
 - $1 \leq n, m \leq 1000$
 - Example
 - Input:
 - 5 8
 - #####
 - #...#
 - #####.#
 - #...#
 - #####
 - Output:
 - 3



Example Solution



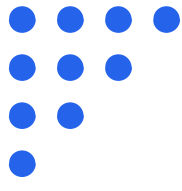
```
#include <bits/stdc++.h>

using namespace std;

const int MAX_N = 250;

int n;
int grid[MAX_N][MAX_N];
bool visited[MAX_N][MAX_N];
```

Example Solution



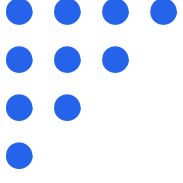
```
int floodfill(int size, int r, int c, int color1, int color2) {
    if((r < 0 || r >= n || c < 0 || c >= n) || (grid[r][c] != color1 && grid[r][c] != color2) ||
visited[r][c]) {
        return size;
    }
    visited[r][c] = 1;

    size++;
    size = floodfill(size, r, c + 1, color1, color2);
    size = floodfill(size, r, c - 1, color1, color2);
    size = floodfill(size, r - 1, c, color1, color2);
    size = floodfill(size, r + 1, c, color1, color2);
    return size;
}
```



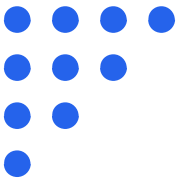
Example Solution

```
int main() {
    cin >> n;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }
    //largest individual region
    int ans = 0;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(!visited[i][j]) {
                ans = max(ans, floodfill(0, i, j, grid[i][j], grid[i][j])); //note that color1
                //and color2 are the same
            }
        }
    }
    cout << ans << '\n';
    //largest team region
    ans = 0;
```



Example Solution

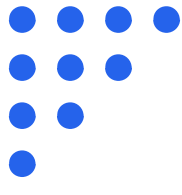
```
//adjacent y IDs
for(int i = 0; i < n; i++) {
    for(int j = 1; j < n; j++) {
        if(ans >= n*n/2) {
            /*
            thinking back on this part
            idk why i wrote this code
            but it made it pass, so mistakes are helpful sometimes
            programmer mindset
            */
            cout << ans << '\n';
            return 0;
        }
        if(grid[i][j] != grid[i][j-1]) {
            memset(visited, 0, sizeof(visited));
            ans = max(ans, floodfill(0, i, j, grid[i][j], grid[i][j-1]));
        }
    }
}
```



Example Solution

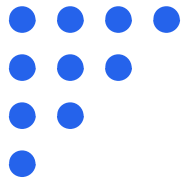
```
//adjacent x IDs
for(int i = 1; i < n; i++) {
    for(int j = 0; j < n; j++) {
        if(ans >= n*n/2) {
            cout << ans << '\n';
            return 0;
        }
        if(grid[i][j] != grid[i-1][j]) {
            memset(visited, 0, sizeof(visited));
            ans = max(ans, floodfill(0, i, j, grid[i][j], grid[i-1][j]));
        }
    }
}

cout << ans << '\n';
return 0;
}
```



Counting Rooms Solution Sketch

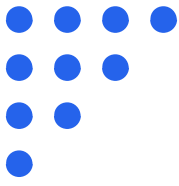
- Whenever we come across an unvisited cell, call the flood-fill method and increment the number of rooms
- Mark all cells we visited from floodfill as “visited”
- Can be thought of counting number of connected components



FloodFill: Challenge Problem

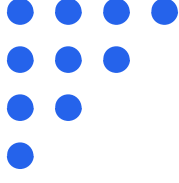
[USACO - CountCross](#)

[USACO - MultiMoo](#)



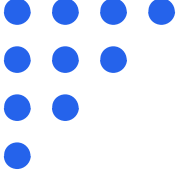
FloodFill: CountCross Solutions

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int h, w;
5 vector<vector<int>> heights, flags;
6
7 vector<vector<int>> dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
8
9 void dfs(vector<vector<bool>>& vis, int r, int c, int x) {
10     vis[r][c] = true;
11     for (auto& d : dirs) {
12         int dr = r + d[0], dc = c + d[1];
13         bool out = dr < 0 || dc < 0 || dr >= h || dc >= w;
14         if (!out && !vis[dr][dc] && abs(heights[r][c] - heights[dr][dc]) <= x) {
15             dfs(vis, dr, dc, x);
16         }
17     }
18 }
19
```



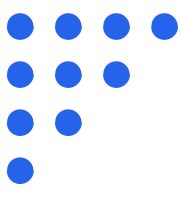
FloodFill: CountCross Solutions

```
20 // Return true if using x as rating can make all 1's reachable
21 bool ok(int x) {
22     vector<vector<bool>> vis(h, vector<bool>(w));
23     for (int r = 0; r < h; r++) {
24         bool found = false;
25         for (int c = 0; c < w; c++) {
26             if (flags[r][c] == 1) {
27                 found = true;
28                 dfs(vis, r, c, x);
29                 break;
30             }
31         }
32         if (found) break;
33     }
34     for (int r = 0; r < h; r++) {
35         for (int c = 0; c < w; c++) {
36             if (flags[r][c] == 1 && !vis[r][c]) return false;
37         }
38     }
39     return true;
40 }
41
```



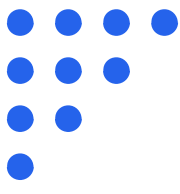
FloodFill: CountCross Solutions

```
42 void solve() {
43     cin >> h >> w;
44     heights = flags = vector<vector<int>>>(h, vector<int>(w));
45     for (auto& row : heights) {
46         for (int& x : row) {
47             cin >> x;
48         }
49     }
50     for (auto& row : flags) {
51         for (int& x : row) {
52             cin >> x;
53         }
54     }
55     int lo = 0, hi = 1e9, ans = 1e9;
56     while (lo <= hi) {
57         int mid = lo + (hi - lo) / 2;
58         if (ok(mid)) {
59             ans = mid;
60             hi = mid - 1;
61         } else {
62             lo = mid + 1;
63         }
64     }
65     cout << ans;
66 }
67
```

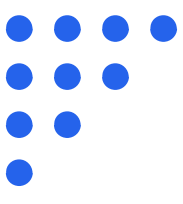


FloodFill: CountCross Solutions

```
68  /* I/O Template */
69  string task = "ccski";
70  bool multiple_testcases = false;
71
72  signed main() {
73      int T = 1;
74      cin.tie(0)->sync_with_stdio(0);
75      if (!task.empty()) {
76          freopen((task + ".in").c_str(), "r", stdin);
77          freopen((task + ".out").c_str(), "w", stdout);
78      }
79      if (multiple_testcases) cin >> T;
80      for (int t = 1; t <= T; t++) {
81          solve();
82      }
83  }
```



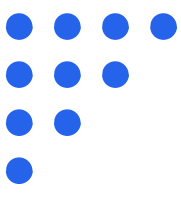
FloodFill: Multiplayer Moo Solution



```
1 #include <fstream>
2 #include <iostream>
3 #include <map>
4 #include <set>
5 #include <vector>
6
7 using std::cout;
8 using std::endl;
9 using std::pair;
10 using std::vector;
11
12 /** @return the 4 cardinal neighbors of a position */
13 vector<pair<int, int>> neighbors(int r, int c) {
14     return {{r + 1, c}, {r - 1, c}, {r, c + 1}, {r, c - 1}};
15 }
```

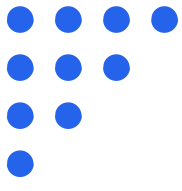


FloodFill: Multiplayer Moo Solution



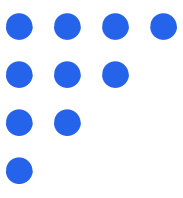
```
17 int main() {
18     std::ifstream read("multimoo.in");
19
20     int side_len;
21     read >> side_len;
22     vector<vector<int>> grid(side_len, vector<int>(side_len));
23     for (int r = 0; r < side_len; r++) {
24         for (int c = 0; c < side_len; c++) { read >> grid[r][c]; }
25     }
26
27     // contains the region ids of each cell- those with the same id are
28     // connected
29     vector<vector<int>> regions(side_len, vector<int>(side_len));
30     // region_cells[r] contains the positions of the cells with region id r
31     vector<vector<pair<int, int>>> region_cells;
32
33     int one_biggest = 0;
34     vector<vector<bool>> visited(side_len, vector<bool>(side_len));
35     // floodfill the regions to see which cells are connected
```

FloodFill: Multiplayer Moo Solution



```
35 // floodfill the regions to see which cells are connected
36 for (int r = 0; r < side_len; r++) {
37     for (int c = 0; c < side_len; c++) {
38         if (visited[r][c]) { continue; }
39
40         int curr_region = region_cells.size();
41
42         vector<pair<int, int>> contained;
43
44         vector<pair<int, int>> frontier{{r, c}};
45         visited[r][c] = true;
46         // floodfill to find all cells connected to the current one
47         while (!frontier.empty()) {
48             pair<int, int> curr = frontier.back();
49             frontier.pop_back();
50
51             contained.push_back(curr);
52             regions[curr.first][curr.second] = curr_region;
```

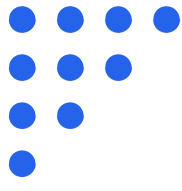
FloodFill: Multiplayer Moo Solution



```
53     for (const auto &[nr, nc] :
54         neighbors(curr.first, curr.second)) {
55         if (0 <= nr && 0 <= nc && nr < side_len && nc < side_len &&
56             !visited[nr][nc] && grid[nr][nc] == grid[r][c]) {
57             visited[nr][nc] = true;
58             frontier.push_back({nr, nc});
59         }
60     }
61 }
62 }
63 one_biggest = std::max(one_biggest, (int)contained.size());
64 region_cells.push_back(contained);
65 }
66 }
```



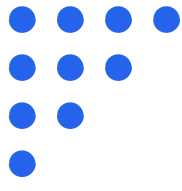
FloodFill: Multiplayer Moo Solution



```
68 // get the regions that are adjacent to other regions
69 vector<std::set<int>> adj_regions(region_cells.size());
70 for (const vector<pair<int, int>> &reg : region_cells) {
71     for (const auto &[r, c] : reg) {
72         for (const auto &[nr, nc] : neighbors(r, c)) {
73             if (0 <= nr && 0 <= nc && nr < side_len && nc < side_len &&
74                 regions[nr][nc] != regions[r][c]) {
75                 adj_regions[regions[r][c]].insert(regions[nr][nc]);
76             }
77         }
78     }
79 }
```



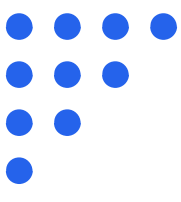
FloodFill: Multiplayer Moo Solution



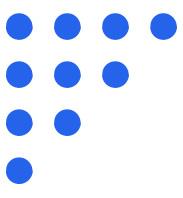
```
81 /** @return the cow id of a region */
82 auto region_id = [&](int r) {
83     return grid[region_cells[r][0].first][region_cells[r][0].second];
84 };
85 // record of pairs of regions' areas that've been processed already
86 std::map<pair<int, int>, std::set<int>>> seen;
87 int two_biggest = one_biggest;
88 for (int r1 = 0; r1 < region_cells.size(); r1++) {
89     for (int r2 : adj_regions[r1]) {
90         pair<int, int> valid{region_id(r1), region_id(r2)};
91         if (valid.first > valid.second) {
92             std::swap(valid.first, valid.second);
93         }
94     }
```

FloodFill: Multiplayer Moo Solution

```
98 // floodfill across whole regions this time, not just cells
99 int two_size = 0;
100 vector<int> frontier{r1};
101 // regions we've currently visited
102 vector<bool> curr_vis(region_cells.size());
103 curr_vis[r1] = true;
104 while (!frontier.empty()) {
105     int curr = frontier.back();
106     frontier.pop_back();
107     two_size += region_cells[curr].size();
108     seen[valid].insert(curr);
109     for (int nr : adj_regions[curr]) {
110         int nid = region_id(nr);
111         if (!curr_vis[nr] &&
112             (valid.first == nid || valid.second == nid)) {
113             curr_vis[nr] = true;
114             frontier.push_back(nr);
115         }
116     }
117 }
118 two_biggest = std::max(two_biggest, two_size);
119 }
120 }
```



FloodFill: Multiplayer Moo Solution



```
121
122     std::ofstream("multimoo.out") << one_biggest << '\n' << two_biggest << endl;
123 }
```