

Topic 4: More Stacks, Depth-First Search, and FloodFill

Competitive Programming I (Fall 2019)

Ninghui Li (Purdue University)

Kattis: Working at the Restaurant (Optional)

Problem from Southwestern Europe Regional Contest (SWERC) 2009

Tom is asked to take plates from waiters and, when the dishwasher requests plates, pass the plates on. The plates must be passed in the same order as they are brought in. Tom can put plates in two piles, P1 and P2.

Input is a sequence of commands: each "DROP x" or "TAKE y", where x and y are integers. Output is a sequence of commands

- "DROP 1 x" or "DROP 2 x" Repeat x times putting a plate on P1 (or P2)
- "TAKE 1 x" or "DROP 2 x" Repeat x times taking a plate from P1 (or P2)
- "MOVE 1->2 x" Repeat x times taking a plate from P1 and putting it on P2
- "MOVE 2->1 x" Repeat x times taking a plate from P2 and putting it on P1

Working at the Restaurant: An Example

Sample Input: 5 **DROP 3** **TAKE 2** **DROP 2** **DROP 2** **TAKE 5**

Sample Output: **DROP 2 3** **MOVE 2->1 3** **TAKE 1 2** **DROP 2 2**
DROP 2 2 **TAKE 1 1** **MOVE 2->1 4** **TAKE 1 4**

				7	7	4	
3	1			6	6	5	
2	2		5	5	5	6	
1	3	3	3 4	3 4	4	7	

DROP 2 3 **TAKE 1 2** **DROP 2 2** **MOVE 2->1 4**
MOVE 2->1 3 **DROP 2 2** **TAKE 1 1** **TAKE 1 4**

Working at the Restaurant: The Concepts

Tom is asked to implement a Queue (First-in-first-out)

Tom is given two piles, which act as two Stacks (Last-in-first-out)

Streaming elements through one stack reverses the order, so streaming them through two stacks gives the correct order.

Thus using two stacks can implement a queue.

Using Two Stacks to Implement a Queue

- When an item enters (the tail of) a queue, it is pushed to Stack 2.
- When we want to get an element from (the head of) a queue, we try to pop from Stack 1. If Stack 1 is empty, we pop each element in Stack 2, and push it to Stack 1 until Stack 2 is empty.
 - At this point, the original bottom item of Stack 2 (first element) becomes the top of Stack 1 (and the next to go out).
- What is the complexity? Assuming constant push/pop time, what is the complexity of enqueue and dequeue?

A single removeFirst() may have to wait until all N items moved from Stack 2 to 1.

However, the next N-1 removeFirst each will take one pop.

For each item, entering the queue and then leaving, takes 4 stack operations (2 pushes, 2 pops).

The **amortized complexity** is constant time.

LeetCode: Max Area of Island (Required)

Given a non-empty 2D array `grid` of 0's and 1's, an island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water. Find the maximum area of an island in the given 2D array.

Note: The length of each dimension in the given `grid` does not exceed 50.

Max Area of Island Example


```
[ [0,0,1,0,0,0,0,1,0,0,0,0,0],  
  [0,0,0,0,0,0,0,1,1,1,0,0,0],  
  [0,1,1,0,1,0,0,0,0,0,0,0,0],  
  [0,1,0,0,1,1,0,0,1,0,1,0,0],  
  [0,1,0,0,1,1,0,0,1,1,1,0,0],  
  [0,0,0,0,0,0,0,0,0,0,1,0,0],  
  [0,0,0,0,0,0,0,1,1,1,0,0,0],  
  [0,0,0,0,0,0,0,1,1,0,0,0,0]]
```

- What should be the output for the map to the left?
- 6. Not 11, because the two bottom parts are not connected.
- How to compute the answer?
- Look at each cell, if it is 1, we find out how big is the island this cell is 1, by recursively visiting all its neighbors.

Max Area of Island: Code in C++ (Part 1)

```
1 class Solution {  
    ...  
2 public:  
3     int maxAreaOfIsland(vector<vector<int>>& grid) {  
4         m = grid.size();    n = grid[0].size();  
5         int ans = 0;  
6         for (int i=0; i<m; i++) for (int j=0; j<n; j++) {  
7             if (grid[i][j] == 1)    ans = max(ans, dfs(grid,i,j));  
8         }  
9         return ans;  
10    }  
11};
```

This call returns the area of the island including cell(i,j).



DFS Code

```
1 class Solution {
2     int m, n;
3     int dfs(vector<vector<int>>& g, int i, int j) {
4         if (i<0 || i>=m || j<0 || j>=n || g[i][j]!=1)
5             return 0;
6         g[i][j] = 2;
7         return 1+dfs(g,i-1,j)+dfs(g,i+1,j)+dfs(g,i,j-1)+dfs(g,i,j+1);
8     }
9 }
```

Here dfs returns an integer, which is the number of cells newly visited in the search. Note that dfs returns 0 when the indices are out of the boundary, when the cell contains 1, or when it has already been visited.

Initially each cell is either 0 or 1. After all dfs calls, each is either 0 or 2.

DFS Code: Why does it terminate?

```
1 class Solution {  
2     int m, n;  
3     int dfs(vector<vector<int>>& g, int i, int j) {  
4         if (i<0 || i>=m || j<0 || j>=n || g[i][j]!=1)  
5             return 0;  
6         g[i][j] = 2;  
7         return 1+dfs(g,i-1,j)+dfs(g,i+1,j)+dfs(g,i,j-1)+dfs(g,i,j+1);  
8     }
```

At most how many times would dfs(i,j) be called for a pair (i,j)?
Once? Twice? Some other number of times? Unbounded?

At most once from line 7 on previous slide. At most 4 times from line 7 on this slide, once (**why?**) from each of its neighboring cells.

Line 6 ensures that next time dfs(g,i,j) is called; it will return on line 5.

DFS Pseudo Code

```
int or void dfs(...) {  
    // 1. Check if out of bound, wrong type of cell, or  
    // already visited  
    // 2. Update current cell to indicate that it is visited  
    // 3. Recursively call dfs on all neighbors  
    // 4. Do whatever else needs to be done  
}
```

DFS uses recursive function calls, which uses stacks. If the connected portion gets too big, and the recursion goes too deep, we will be out of stack. (Seen out of stack in Java with around 7000 nested calls.)

LeetCode: Making a Large Island (Required)

In a 2D grid of 0s and 1s, what is the size of the largest island if one is allowed to change at most one 0 to 1? (An island is a 4-directionally connected group of 1s).

$1 \leq \text{grid.length} = \text{grid}[0].\text{length} \leq 50.$

$0 \leq \text{grid}[i][j] \leq 1.$

- One bruteforce method is: for each 0 cell, changes it to 1, then find the max island area. (Then change 1 back to 0.)
- Let N be size of grid (no more than 2500). What is the complexity?
- Complexity of brute-force is $O(N^2)$. Can we avoid this? Are we repeating computations?
- Bruteforce does repeated DFS search. Can we use DFS just once?

Making a Large Island: Flood Fill

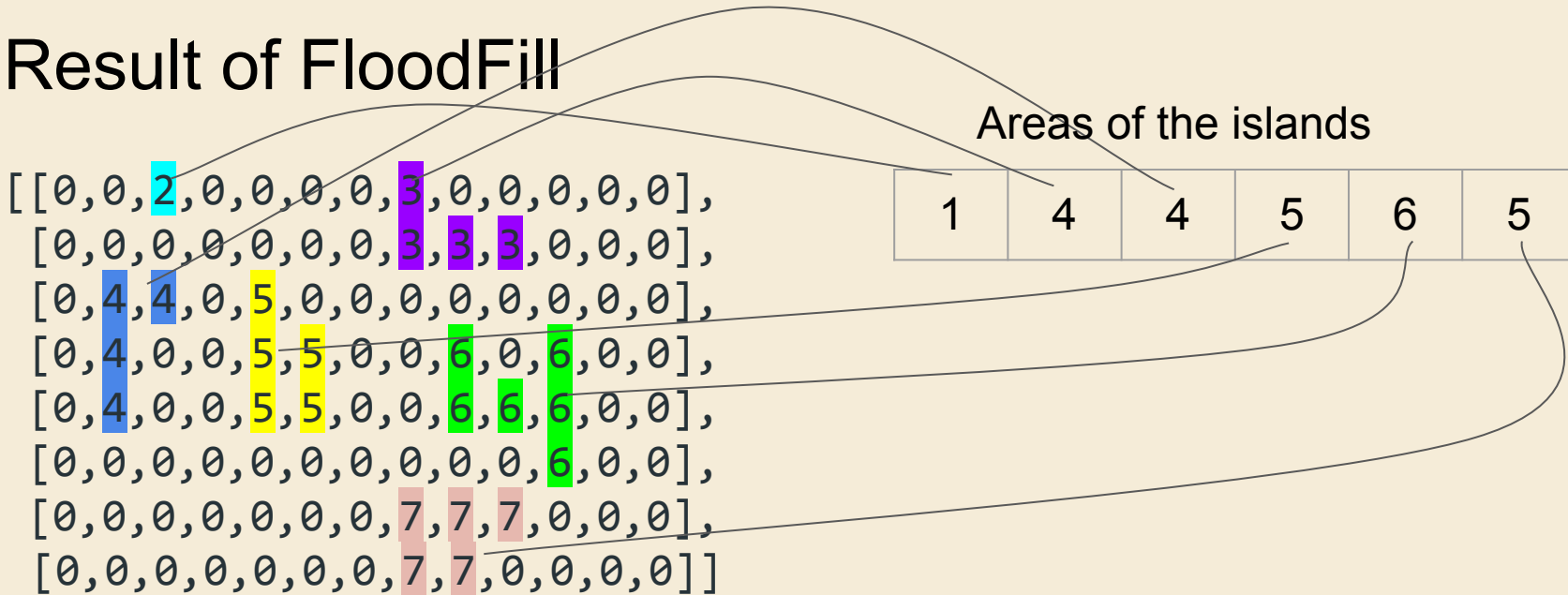
- First use DFS to find all islands.
- Store areas of all islands.
- Mark each island differently (overwrite a different value for each island).
- Also, for each new island, store its area.
- (See next slide for illustration.)
- Then loops through all 0's, for each 0, computes the sum of the area of all neighboring islands.

Result of FloodFill

```
[ [0,0,2,0,0,0,0,3,0,0,0,0,0],  
  [0,0,0,0,0,0,0,3,3,3,0,0,0],  
  [0,4,4,0,5,0,0,0,0,0,0,0,0],  
  [0,4,0,0,5,5,0,0,6,0,6,0,0],  
  [0,4,0,0,5,5,0,0,6,6,6,0,0],  
  [0,0,0,0,0,0,0,0,0,0,6,0,0],  
  [0,0,0,0,0,0,0,7,7,7,0,0,0],  
  [0,0,0,0,0,0,0,7,7,0,0,0,0]]
```

Areas of the islands

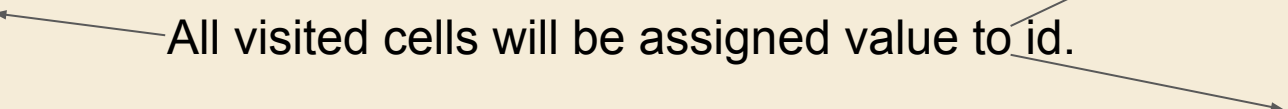
1	4	4	5	6	5
---	---	---	---	---	---



Making a Large Island: Floodfill Code

```
int m, n;  
    // Using two arrays to help enumerate all neighbors.  
int dx[4] = {-1,1,0,0}, dy[4] = {0,0,-1,1};  
  
int floodfill(vector<vector<int>>& A, const int i, const int j, int id) {  
    if (i<0 || i>=m || j<0 || j>=n || A[i][j] !=1) return 0;  
    A[i][j] = id;  
    int a = 1;  
    for (int k=0; k<4; k++)    a += floodfill(A, i+dx[k], j+dy[k], id);  
    return a;  
}
```

All visited cells will be assigned value to id.



Making a Large Island: Code Fragments

```
int largestIsland(vector<vector<int>>& A) {
    m = A.size();
    if (m == 0)    return 0;
    n = A[0].size();
    If (n == 0)    return 0;
    int id = START_ID;          // #define START_ID 2
    vector<int>  sizes;
    for (int i=0; i<m; i++) for (int j=0; j<n; j++) {
        if (A[i][j] == 1)      sizes.push_back(floodfill(A, i, j, id++));
    }
    if (id == START_ID)        return 1;    // all 0 in input
    else if (sizes[0] == m*n)   return sizes[0]; // all 1 in input
}
```


Making a Large Island: Code Fragments

```
for (int i=0; i<m; i++) for (int j=0; j<n; j++) {
    if (A[i][j] == 0) {        // find a 0 cell
        set<int> s;            // set of all neighboring island ids
        for (int k=0; k<4; k++) {
            int x = island_id(A, i+dx[k], j+dy[k]);
            if (x != 0)        s.insert(x - START_ID); // add to set
        }
        int sum = 1;          // area if this 0 turns to 1
        for (auto it = s.begin(); it!=s.end(); it++)    sum+=sizes[*it];
        max_size = max(max_size, sum);
    }
}
return max_size;
```

Making a Large Island: Code Fragments

```
int island_id(vector<vector<int>>& A, const int i, const int j) {  
    if (i<0 || i>=m || j<0 || j>=n)    return 0;  
    return A[i][j];  
}
```

Concepts in Making a Large Island

- Flood fill: overwrite each cell in an island (connected component) with a different number
- Using two arrays dx, dy to help enumerate through all neighbors
 - Helps even more when considering 8 directions
- Neighbors of a 0 cell may belong to the same island

Kattis: Island (Required)

Given a image, which is a grid where each cell is either land (denoted as 'L'), water (denoted as 'W'), or covered by clouds (denoted as 'C'). Clouds mean that the square could either be land or water. Determine the minimum number of islands that is consistent with the given image.

Input: Output

3 2 1

LW

CC

WL

Input:

3 4

CCCC

CCCC

CCCC

Output:

0

How to deal with cloud?

Kattis: Island

- Start DFS when encountering a land cell
- During DFS, each encountered could cell is treated as a land cell