One-Day Assignment 8 – algorithmic solution

Islands

One Day Assignment 8 – General

We keep the graph as is (in grid form)

We use an additional (2D) boolean array (visited_arr) to keep track of whether a cell in the grid has been visited or not. Initially, all cells are marked as unvisited

This problem can be solved through either BFS or DFS; if BFS is chosen, we will also need a Queue<IntegerPair> to keep track of cells in the BFS queue. We use an IntegerPair here as we represent a cell by (cell_row, cell_col) in the queue, as opposed to just a single integer.

If using DFS, you can use an explicit stack similarly to BFS, or just use recursion directly.

One Day Assignment 8 – BFS (loop in main)

```
answer = 0
for curr_cell in grid[1]:
    if curr_cell is 'L' && curr_cell is unvisited:
        answer = answer + 1
        add curr_cell to queue
        mark curr_cell as visited in visited_arr
        BFS(queue, grid, visited_arr)
output answer
```

1. In actual implementation, you will likely need to use 2 for loops to do this, instead of 1

One Day Assignment 8 – BFS Algorithm

Body of BFS(queue, grid, visited_arr[2]):

```
while queue is not empty:
    dequeue cell_to_check from queue
    for each of the four directions (up, down, left, right) from cell_to_check:
        next_cell = cell in the direction from cell_to_check
        if next_cell is not out of bounds, and is a ‘C’ or ‘L’, and is unvisited:
            add next_cell to queue
            mark next_cell as visited
```

2. It may help to pass in the dimensions of the grid as additional parameters as well, as opposed to calling .length on the grid to get the dimensions

One Day Assignment 8 – DFS (loop in main)

```
answer = 0
for curr_cell in grid[1]:
    if curr_cell is ‘L’ && curr_cell is unvisited:
        answer = answer + 1
        DFS(curr_cell, grid, visited_arr)
output answer
```

1. In actual implementation, you will likely need to use 2 for loops to do this, instead of 1

One Day Assignment 8 – DFS Algorithm

```
Body of DFS(curr_cell, grid, visited_arr[2]):
mark curr_cell as visited in visited_arr
for each of the four directions (up, down, left, right) from curr_cell:
    next_cell = cell in the direction from curr_cell
    if next_cell is not out of bounds, and is a ‘C’ or ‘L’, and is unvisited:
        DFS(next_cell, grid, visited_arr)
```

2. It may help to pass in the dimensions of the grid as additional parameters as well, as opposed to calling .length on the grid to get the dimensions

One Day Assignment 8 – English Description

We iterate through every cell in the grid. When we encounter a cell that is an ‘L’, and has not been visited yet, we increase a variable called answer (initialised to 0) by 1, and begin graph traversal (either BFS or DFS) on it.

During graph traversal, when looking through the 4 different directions from a cell, we first check:

- If the new position is out of bounds of the grid
- If not, we then check if the new position contains a ‘C’ or an ‘L’
- If it does, we then check that the new position has not been marked as visited yet

Only when the new position passes all 3 checks do we continue graph traversal on that cell, and mark it as visited

When done, output answer as the final result

Looping over directions

When trying to loop over the “directions” (up, down, left, right), it can be helpful to define direction arrays:

```
    int dx[] = {0, 0, 1, -1};
    int dy[] = {1, -1, 0, 0};
```

So now to loop over directions, we can loop over the indices 0,1,2,3 to get the x and y direction.

You can even extend the “directions” to include diagonals!