Dr. S. Alper SERT
alper.sert@ceng.metu.edu.tr
Cihad TEKİNBAŞ
ctekinbas@ceng.metu.edu.tr

Assignment 01

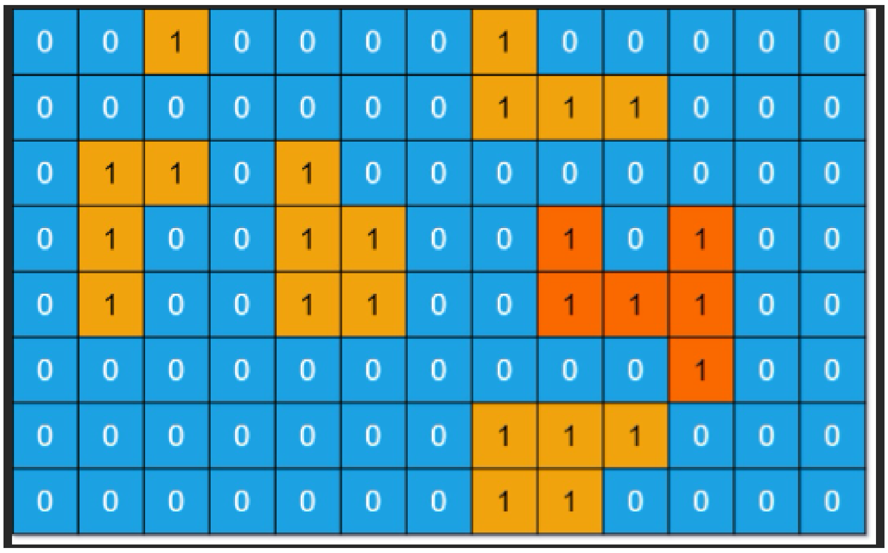
METU CENG310 Fall 2023-2024
Data Structures and Algorithms with Python

Start Date: December 8th, 2023
Due Date: December 15th, 2023

1 Finding Island (10 pts)

In this assignment you are expected to develop a Python module, namely `land`, that comprises classes and functions that collectively finds a island that is represented in a text file. In doing so, you need to follow the steps that are described in the following sections.

In the context of a 2D grid representation, an island is defined as a cohesive group of cells containing the value ’1’, indicating land. The connectivity between land cells occurs in a 4-directional manner, allowing for horizontal or vertical connections. As part of the assumptions, it is considered that all four edges of the grid are enveloped by water, designated by the value ’0’. To illustrate, envision a grid where ’1’ represents land, ’0’ signifies water, and the arrangement of these values forms distinct islands in 1.



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

Figure 1: An example Land

1.1 `IslandFinder` Class

`class IslandFinder()`

The module `land` should include an implementation for the `IslandFinder` class. `IslandFinder` should have the following data members and methods.

`__init__(path)`

The constructor will accept a string `path` and create a `Land` instance by utilizing `text_to_array` method, and the new `Land` instance will be assigned to class member `_land`.

`_land`

The class member `_land` will identify the `Land` object for which the solution will be performed.`_cellstack`

The class member `_cellstack` will identify a `Stack` object onto which land cell coordinates will be pushed as tuples of `row_index` and `column_index`. As the land is being explored, the cells to be explored will be pushed onto the stack. At each exploration iteration, the next cell to be investigated will be read from the top of the `_cellstack` with a `top` operation. If the cell being investigated does not have a valid neighbor, it is popped out from the `_cellstack`.

`_explored`

`_explored` will hold the list of the explored cells. For this class member, you may prefer to use a `list` object. During the execution of the solution algorithm, this data member will be accessed frequently to check whether a given cell is visited before.

[Bonus (2.5pts)] As you have already learned, containment checking with a `list` object takes $O(n)$ time in the worst case. In order to make these checks, maintain an ordered `_explored` list and implement a custom containment check operation that performs a binary search on the list so that this frequent operation can be done in $O(logn)$.

`text_to_array(self, path)`

`text_to_array` should read the file whose path is `path` and return a `Land` object. In order to create a `Land` object, the content of the input text file should be read and a `list` of `lists` should be constructed such that it can be indexed as a two-dimensional array. For example, for a text file such as the one shown below, it should create and return a list like this: `[['1','1','1']['0','0','1']['0','1','0']]` so that it should be able to be indexed such that, for example, the character at `(2,1)` would be `'1'`.

[Bonus (1 pt)] Write a single line of code that converts the content of the input file to `list` of `lists`. Please note that obtaining the file handle object should not be included in this one-liner code.

```
110
010
000
```

`get_a_neighbor(a_tuple)`

Given a tuple of row and column id, `get_a_neighbor(a_tuple)` computes the location of a **valid** adjacent up, right, down, or left cell and returns a coordinate tuple. Valid cells have character `1` and does not exist in `_explored` list.

`find_island()`

`find_island` will provide a solution to the land which is identified by `_land` by adopting the following algorithm.

```
* Clear and initialize _cellstack and _explored.
* Initialize _maxArea to 0.
* Iterate through each cell in the grid:
      + If the current cell is part of an island and not in _explored:
            * Push the current cell to _cellstack.
            * Initialize _area to 0.
            * While _cellstack is not empty, loop:
                  + Get the top cell c from the _cellstack.
                  + If c is not in _explored:
                        * Mark c as visited .
                        * Increment _area.
                        * Add c to _explored.
                        * Add unvisited neighbors of c to _cellstack.
                  + If there are no unvisited neighbors, pop c from _cellstack.
            * Update _maxArea with the maximum of _maxArea and _area.
* Return _maxArea, which represents the maximum area of an island in the grid.
```

1.1.1 Usage of the `land` Module

A typical usage of your `land` module would be as follows.

```
import land
fi = land.IslandFinder('testcase.txt')
fi.find_island()
# Example Output:
# 6
# 6 is maximum island
```

1.2 `Land` Class

Build a `Land` class that internally maintains a two-dimensional array of characters which represents land structure. In such land representation, there exists two distinct cell types: island cells,water cells

`Land` class should have a constructor accepting a list object satisfying land structure rules:

- List object has to be comprised of `list`s of characters.
- It should represent a mxn array where $m > 3$ and $n > 3$, and each of the internal `list`s should be of length n, and each cell of lists has to hold a single character (technically speaking, they have to be strings of length one).
- Land structure have to include only `'0'` (water), `'1'` (island), and characters. No other characters should be allowed.

Constructor should accept only those `list` objects that satisfy these rules, otherwise it should raise an `InvalidLandException`.

Indexing of the land should start from the upper left corner of the array, in other words, the cell (0,0) should be the upper left corner of the two-dimensional array.

`Land` class should have the following accessor functions.

`get_start()`

`get_start()` should return the starting cell (i.e., a tuple) of the land. Each cell should be represented as a tuple `(row_index, column_index)`.

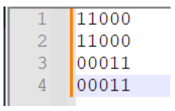
1.3 `Stack` Class

Develop `Stack` class that implements the Stack ADT that we covered in the lecture. The implementation provided in the textbook could be adopted.

1.4 Example Test Cases

Below are some of the example text cases that might be helpful to you during the development. Text file including these examples is available on ODTUClass page. Please note that there will be many more test cases that we will use during the evaluation of your module.

1.4.1 Test Case 1 (Solution Exists)



```
11000
11000
00011
00011
```

1.4.2 Error Cases

2 Delivery Instructions

Please hand in your module as a single file named as `land.py` over ODTUClass by 11:59pm on due date. An Assignment-01 page will be generated soon after the start date of this assignment. Should you have any questions pertaining to this assignment, please ask them in advance (rather than on the due date) for your own convenience. Whatever IDE you use, you have to make sure that your module could be run on a Python interpreter ($> `python.exe land.py`) or iPython (%`run land.py`).