**Lab Assignment 2**

**Alice’s flat planet**

|  |  |
| --- | --- |
| **Due Date**  **(a two-week LA)** | |
| **Sections(540,543,544,545)** | **2/11/22 @ 11:59pm** |

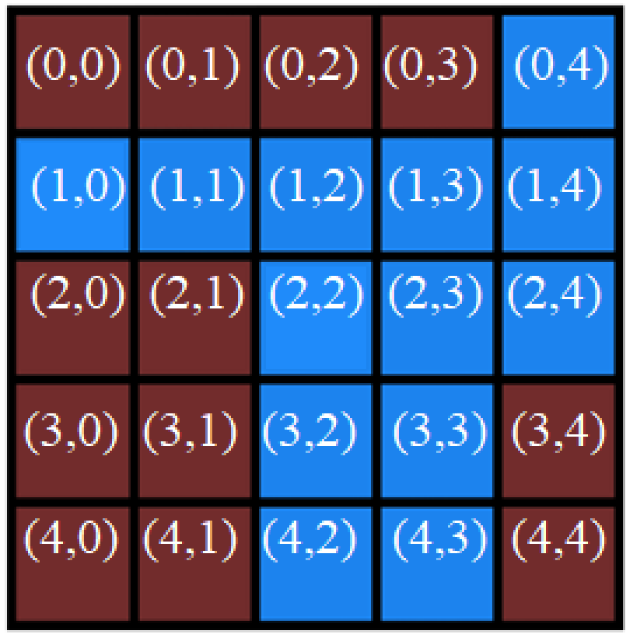
**Concepts**

* Recursion
* Base and General Cases
* Divide, Conquer and Glue

**Problem Specification**

Alice lives on a flat planet that can be modeled as a square grid of size n×n, with rows and columns enumerated from 0 to n-1. We represent the cell at the intersection of row r and column c with ordered pair (r,c). Each cell in the grid is either *land* or *water*.

Alice resides in land cell (r0,c0). She wishes to travel to land cell (r1,c1). At any moment, she may move to one of the cells adjacent to where she is—in one of the four directions (i.e., N, E, S, or W).



An example planet with n=5. It also appears in the first two sample tests.

Unfortunately, Alice cannot swim, and there are no viable transportation means other than by foot (i.e., she can walk only on land). As a result, Alice's trip may be impossible.

For now, your task is to output a path as a list so that Alice could travel from (r0,c0) to (r1,c1). If Alice’s trip is impossible, the output path should be an empty list, [].

**Input**

The first line contains one integer n(1≤n≤50) — the width of the square grid.

The second line contains two space-separated integers r0 and c0 (0≤r0,c0≤n-1) — denoting the cell where Alice resides.

The third line contains two space-separated integers r1 and c1 (0≤r1,c1≤n-1) — denoting the cell to which Alice wishes to travel.

Each of the following n lines contains a string of n characters. The j-th character of the i-th such line (1≤i,j≤n) is 0 if (i,j) is *land* or 1 if (i,j) is *water*.

It is guaranteed that (r0,c0) and (r1,c1), i.e. starting and ending position are land.

**Examples:**

**Input**:

5

0 0

0 3

00001

11111

00111

00110

00110

**Output:**

[(0,0),(0,1),(0,2),(0,3)]

**Input:**

5

0 0

4 4

00001

11111

00111

00110

00110

**Output:**

[]

**Input:**

8

7 3

1 3

00000000

00000000

00111000

00101000

00101000

00000000

00000000

00000000

**Output:**

[(7, 3), (6, 3), (5, 3), (5, 4), (5, 5), (4, 5), (3, 5), (2, 5), (1, 5), (0, 5), (0, 4), (1, 4), (1, 3)]

**Design Requirements**

**Basic Structure**

You will write two functions. One is named create\_map\_find\_path and other is find\_path.

**Function create\_map\_find\_path(file\_name) does the following tasks:**

* parse the input file given by parameter file\_name to create a land/water grid map stored in a global variable,
* Call the find\_path(…) function with the appropriate arguments based on the input file and the above function specification, AND
* Simply return the path returned by function find\_path(…).

**Function** **find\_path(curr\_pos, dest\_pos) is a recursive function doing the following tasks:**

* Find a path from parameter curr\_pos to parameter dest\_pos based on the previously created land/water grid map stored in a global variable in function create\_map\_find\_path(…), explained as follows. Return a path in a list if found; otherwise, return []
* When traversing from a location in the map, the traversal order ***MUST BE clockwise***, starting with North, followed by East, South, and West, AND
* The Design/Implementation of this function **MUST follow the Divide-** Conquer **-Glue strategy**. Namely, you need to show all the base cases and the recursive case with explicitly showing all subproblems, their (sub-)solutions and finally the glue of all the sub-solutions in pseudo code.

**Function print\_path(pathList)** prints out the path given by parameter pathList. This function is provided by us and please feel free to call it.

A template main.py has been given to you.

**Pseudocode**

You must **MUST follow the Divide-** Conquer **-Glue strategy** for function find\_path(…) and design a pseudo code based on that.

**Implementation Phase**

Using the pseudocode developed, write the Python code for your assignment. This is a two-week assignment.

**Testing Phase**

* You have given plenty example inputs with desired outputs. The output should match exactly.
* Build your program incrementally, carefully testing each function as you go.

**Assignment Submission**

* Generate a .zip file that contains all your files including:
  + Program Files
  + Any input or output files
  + Application of the DCG strategy for function find\_path(…).
* Submit the .zip file to the appropriate folder on E-Learning.