SNU CSE: Data Structures Due date: 11 pm Oct 31 2018

Assignment 2

Lecturer: Srinivasa Rao Satti

1) Maze-Solver:

A) Shortest pass

Write a program that finds a path through a "maze". The maze will be represented by a two dimensional array of integers. You will be given a starting point in the array that consists of a pair of integers <x, y>. In this pair, x represents the row coordinate of the starting cell and y represents the column coordinate of the starting cell.

Given a starting coordinate <x, y>, your program should construct a sequence of coordinates (beginning with <x, y>) that represents the path through the maze. As an example, consider the following maze that has a starting coordinate of (2 ,0). An entry of 0 in this maze represents a "solid rock" and you cannot visit a cell with that value. And an entry 1 in this array represents a cell that u can visit.

0	0	0	0	0	0
0	1	1	1	1	0
1	1	1	1	1	0
0	1	1	0	1	0
0	1	1	0	1	1
0	0	0	0	0	0

A successful run of your program with this array as input would thus output the sequence <2, 0>, <2, 1>, <2, 2>, <2, 2>, <2, 3>, <2, 4>, <3, 4>, <4, 4>, <4, 5>
Your program should solve this maze problem by using a stack to keep track of those cells that have already been visited. Your program thus should proceed as follows.

- Change the value of the starting cell from 1 to some other integer value (say [2]). Make an instance of the Coordinate class that corresponds to the starting cell and push it onto the stack. This coordinate represents the starting. It is the place at which you enter the maze.
- Do a clockwise search for the next cell to be visited. For a given cell there are at most three possibilities for new cell to be visited. Once a "new" cell has been found push a coordinate instance with the cell's coordinate onto the stack and change the value in that cell from 1 to [2] (the same value you used above. NB. The square bracket around the number indicates that 2 is optional and you can

change it to any other value). That cell becomes the “current cell”. If no “new” cell can be found, you are at a dead end and must backtrack. Note that, no “new” cell can be found if the only cell(s) found has(have) a value of [2]. Backtracking is accomplished by popping the stack and designating the “current” cell as the one whose coordinate values match the values of the Coordinate instance on the top of the stack. When a search for the next value to be visited after the “current” cell yields coordinate values that are outside the array, you are done. Note that if the entry cell is the first row, your search will automatically take you outside the array. Thus, you must treat the entry cell as a special case.

You are free to use one of the two implementations of stack, i.e., either Linked list based or array based stack. Your stack should be defined for objects. And, then, you must define a class named Coordinate whose instances will get pushed into the stack. The following code shows how you might do this. You are free to add this code if you like.

```
1  class Coordinate {
2      private int row, col;
3      public Coordinate(int row, int col){
4          this.row = row;
5          this.col = col;
6      }
7      public int getRow(){
8          return row;
9      }
10     public int getCol(){
11         return col;
12     }
13     public void setRow(int row){
14         this.row=row;
15     }
16     public void setCol(int col){
17         this.col = col;
18     }
19     public String toString(){
20         return "<"+row + ","+col+">";
21     }
22 }
```

Your program should get an input from a file. It is not the given file it is the file that you can prepare for yourself but its name should be `maze1.txt` The first line in the line will contain the number of rows and columns for the array separated by a space. Each succeeding line, except the last, will contain the values for the rows. Thus, the file for the array shown above, would consist the following lines:

```
6 6
0 0 0 0 0 0
1 1 1 1 1 0
0 0 1 0 1 0
0 0 1 0 0 0
0 0 1 1 1 1
0 0 0 0 0 0
1 0
```

Your program should read in the first line and construct an array dimensioned to the values in that line. Your program should then read in the rest of the lines (except the last line) and populate the array with the values from each line. The last line in the file will contain a value that is the x and y coordinate of the starting cell separated by a space. Once the file has been read into the array, your program should process the maze (the array) and print out the sequence of coordinates that constitute the path from the starting point to the exit point. Note that, the print out must begin with the starting coordinate and must end with the exit coordinate, not the other way around.
Your program should be capable of solving a maze of any size.

In this first problem, an **entry(starting) point(coordinate)** is the point where you start in the maze, and an **exit point (coordinate)** is the end point where the path should end. These points are assumed to be located at one of the edge points of the array (the shaded cells in the following array). Also, the starting and exit points cannot be the same.

0	0	0	0	0	0
1	1	1	1	1	0
0	0	1	0	1	0
0	0	1	0	0	0
0	0	1	1	1	1
0	0	0	0	0	0

B) **Longest path**

As an extension to problem 1_A , write a program that finds the longest path between two points in the maze.your program should construct a sequence of coordinates (beginning with <x, y>) that represents the path through the maze. For example, for the array given above your program output should look like this; <2, 0>, <2, 1>, <3,1>, <4, 1>, <4, 2>,<3,2>,<2,2>,<1,2>,<1,3>,<1,4>,<2,4>,<3,4>,<4,4>,<4,5>.

You should not consider any loop. It means your code should visit each cell just once.

Your program should get an input from a file.It is not the given file it is the file that you can prepare for yourself. Just like what we done for 1_A and its name is `maze2.txt`.

0	0	0	0	0	0
0	1	1	1	1	0
1	1	1	1	1	0
0	1	1	0	1	0
0	1	1	0	1	1
0	0	0	0	0	0

2) **Subarray:**

Given an array of (positive and negative) integers, report all subarrays in the array whose sum is zero. The sum of a subarray is defined as the sum of all the integers in that subarray.

You should write code with hashing which prints out the all possible subarrays which have sum zero in the given array. Your code should read the file from `array.txt` and try to find all combination of possible subarrays and print the output in the format specified below. The input file stores the array with each integer on a separate line (without any delimiters like ",") as shown in the example file (you can look at the example file array.txt first).

The format of your output: Subarray found from Index X to Y

Example if the given array was arr

```
arr = [6, 3, -1, -3, 4, -2, 2, 4, 6, -12, -7]
```

The output should be like this:

```
Subarray found from Index 2 to 4
Subarray found from Index 2 to 6
Subarray found from Index 5 to 6
Subarray found from Index 6 to 9
Subarray found from Index 0 to 10
```

Important Notes:

1. If two codes are found to be the same, you are getting zero.
2. Any submission after the deadline is not accepted. So, please make sure to submit before the deadline.
3. Any Compile error will not be fixed by the TA. If your program is not able to compile and run correctly, you are getting zero. So, please make sure that there is no compile error in your program.
4. When you write your code, you should assume that the input file is stored in the same directory as your source code. Any valid input will be given by the TA and your program should work for any valid input, as described above.
5. Make sure to use a stack implemented by yourself. **Avoid using system built-in stack**
6. In problem 1 there is no specific output.