Bill Manaris : Spring 2016 / CSCI 230 Homework 2

Assigned Date: Thursday, Jan. 28, 2016
Due Date: Tuesday, Feb. 9, 2016
Due Time: 5 mins before beginning of class

Last modified on February 02, 2016, at 11:49 AM (see updates)

Purpose

This assignment focuses on:

- deriving both recursive and iterative solutions
- creating and manipulating 2D arrays in Java
- deriving the run-time complexity of an algorithm

This is a **solo** assignment. **You may discuss the assignment only with the TA or the instructor.**

Assignment

Create a Java program, `Gerrymander.java`, which solves the following problem:

Consider a 2D grid consisting of cells, some of which are empty and some of which contain an asterisk.

Define two asterisks as *contiguous* if they are adjacent to each other in the same row or in the same column.

Also, define a *district* as follows:

a. A *district* contains at least one asterisk.
b. If an asterisk is in a *district*, then so is any asterisk that is contiguous to it.
c. If a *district* has more than two asterisks, then each asterisk in it is contiguous to at least one other asterisk in the *district*.

For example, there are four districts in the following grid:

*			*	*			*		*	*
							*		*	*

seven districts in this grid:

*		*		*				*	*	*
					*				*	
*				*						

and only one in this grid:

		*	*	*		*	*	*		
				*		*		*		
				*	*	*				

Your program should contain the following two methods (it may also contain additional, auxiliary ones - up to you):

1. a recursive method, `int countDistrictsRecursive()` (parameter list up to you), and
2. **(Bonus)** an iterative method, `int countDistrictsIterative()` (parameter list up to you).

Each method should return the number of districts in a 2D grid.

Input

The program should accept several lines of input, as follows:

an `int` (e.g., 1), which specifies which method to run (e.g., the first method above)
two `ints` depicting the number of rows and columns (0 or more) in the grid, followed by the locations of asterisks within the grid (in increasing order, starting with 0, and terminated by -1).

You may assume that the input is formatted properly and is valid.

Your program should create a 2D array to store the above information.

Output

The program should output the value returned by the method called. Nothing more.

I/O Sample 1

For example, given this input:

```
1
3 11
0 3 4 7 9 10 18 20 21 -1
```

which corresponds to the following grid:

*			*	*			*		*	*
							*		*	*

the output of your program should be:

```
4
```

I/O Sample 2

For example, given this input:

```
2
3 11
0 2 4 8 9 10 16 19 20 22 25 -1
```

the output of your program should be:

```
7
```

I/O Sample 3

For example, given this input:

```
2
3 11
2 3 4 6 7 8 15 17 19 26 27 28 -1
```

the output of your program should be:

```
1
```

and so on.

Run-Time Analysis

In a separate PDF document, `RunTimeAnalysis.pdf`, state the run-time complexity of each of the two methods above, and explain fully why this is the case (see textbook for examples).

You should include all relevant code (just the methods, not everything else).

Note that there are many possible run-time complexities, based on the algorithms you derive to solve the problem.

This document should include similar identification information (name, class, date, etc.) at the top, as your program.

Documentation

Follow the Golden Rule of Style: "A program should be as easy for a human being to read and understand as it is for a computer to execute." [1]

In general, you should comment any variable, obscure statement, block of code, method, and class you create.

Your comments should express **why** something is being done, as opposed to **how** – the how is shown by the code.

Also see the course documentation standards in homework #1.

NOTE: You may use Javadoc formatting, if you prefer. If so, do it consistently.

Submissions

Submit via OAKS as follows:

1. Create a folder named 'First.Last', where 'First' is your first name, and 'Last' is your last name.

2. Place your Java program inside that folder.

3. Place your PDF document inside that folder.

4. Compress the folder as a .zip file.

5. Upload on OAKS.

Grading

Your grade will be based on how well you followed the above instructions, and the depth/quality of your work.

Reference

1. Cooper, D. and Clancy, M. (1985) "Oh! Pascal", 2nd ed., W.W. Norton & Company, New York, p. 42.