CS 1181  
Programming Assignment 7

There are many applications that require counting connected regions in a two-dimensional grid. For example, suppose that you want to estimate damage to agricultural crops in Ohio or the extent of flooding in Louisiana after tropical storm Issac. One approach to solving this problem is to take a satellite photograph of the area, superimpose a grid over the photograph and count the number of grid cells filled with specific colors. The same approach could be used to measure many different types of information including the area of a tumor in a medical image, the amount of bacteria growth in a Petri dish, or the number and area of craters on the moon.

In this programming assignment you will measure the total area of a forest from a “simulated” image. The input consists of a pattern of zeros and ones that represent a binary image. A “one” indicates a forest cell and a “zero” represents open ground. The following example illustrates the layout of the input for this problem. Assume the input consists of a 10x20 (10 rows by 20 columns) grid of zeros and ones. Your task is to locate all **eight-connected regions** in the grid. An eight-connected region consists of a set of marked cells (value 1) such that each cell in the region can be reached by moving up, down, left or right or diagonally from another marked cell in the region. In the example shown below there are 6 eight-connected regions of various sizes. Your tasks are as follows:

1. Prompt the user for an input file name.

2. Read the contents of the simulated image from the input file.

The form of an input file is as follows:

10 20 Line 1: Size of the grid (rows by columns)

0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 Line 2: First row

0 1 1 1 0 1 1 0 0 1 0 0 1 1 1 0 0 0 1 1 Line 3: Second row

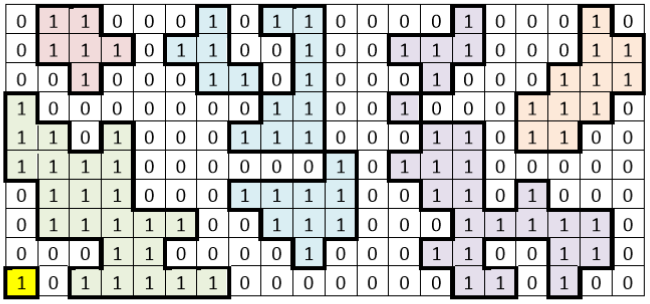
... ... etc.

3. Process the grid **using a recursive function**.

4. Prompt the user for an output file name.

5. Write the number of connected regions, size of each region and a grid showing the regions labeled in the order they were identified by your program to the output file. Make sure to label and align output to improve readability.

The following example illustrates how connected regions are defined. This is a sample input with the connected regions highlighted. Note that the actual grid lines are not represented in the input or output.



For full credit, handle exceptions associated file I/O (i.e. errors related to opening and closing files). Make sure to **implement a grid class that incorporates all the methods needed to input the data, process the data and output the results to a file.** You may add public and private methods and fields to the class as needed, but your class must include at least three public member functions of the following form:

class grid {

public void load( String inputFilename ); // loads the initial data from a text file

public void findRegions (); // generate and store the results in the object

public void save( String outputFilename ); // saves the results to a text file

// other public and private methods

// It might be beneficial for you to make a display method that outputs the results to the console first.

// After the class is working, then convert the display method to write to a file, or copy the code from

// display method into the save method and modify the save method so it writes to a file.

}

**WARNING:** The input files use the integer 1 to indicate that a cell in the grid is part of a region and that cell has not been processed. This WILL cause you a problem in your recursive solution because you are supposed to mark each region with a region number starting with the number 1! The easiest way to deal with this issue is when you read the file into your 2-D array, if the value you read from the file for a cell is 1, store a -1 into that cell in your array. Setup your algorithm so that it recognizes any cell which contains a -1 as part of a region that has not yet been labeled.

**The format of input and output files are shown below.**

Sample Input File:

10 20

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

Sample Output File:

Region Size   
--------- -----   
 1 6   
 2 23   
 3 26   
 4 11   
 5 23   
 6 1

Labeled Regions   
----------------------

0 1 1 0 0 0 2 0 2 2 0 0 0 0 3 0 0 0 4 0   
0 1 1 1 0 2 2 0 0 2 0 0 3 3 3 0 0 0 4 4   
0 0 1 0 0 0 2 2 0 2 0 0 0 3 0 0 0 4 4 4   
5 0 0 0 0 0 0 0 2 2 0 0 3 0 0 0 4 4 4 0   
5 5 0 5 0 0 0 2 2 2 0 0 0 3 3 0 4 4 0 0   
5 5 5 5 0 0 0 0 0 0 2 0 3 3 3 0 0 0 0 0   
0 5 5 5 0 0 0 2 2 2 2 0 0 3 3 0 3 0 0 0   
0 5 5 5 5 5 0 0 2 2 2 0 0 0 3 3 3 3 3 0   
0 0 0 5 5 0 0 0 0 2 0 0 0 3 3 0 0 3 3 0   
6 0 5 5 5 5 5 0 0 0 0 0 0 0 3 3 0 3 0 0

**There are 4 data files provided with this lab. You are to run your program on the 4 data files. Only one output file is produced that contains the region size and labeled region output for all 4 input data files. You are to turn in your Netbeans project all in ONE zip file, your ONE output file, and ONE text file that contains a copy of YOUR source code.**

**There are NO static member variables AND there are NO static methods in the project other than main! (-20 pts if this is violated in any way!)**

**RUBRIC: 100 pts**

* **Load and save methods work properly. (15)**
* **The size of all regions for all 4 data files is properly reported in the output file. (20)**
* **All cells belonging to each region are properly marked with region numbers in the output file for all 4 data files (20)**
* **The findRegions method *calls a recursive method* for any cell marked as part of a region that has not been processed yet. (15)**
* **The recursive method finds and marks each cell that is part of a region with that region’s number. (20)**
* **Proper commenting and coding style: ALL classes have a description comment. ALL methods have proper Javadoc comments including any preconditions, parameter, return value, and exception comments. The Javadoc tool should run with no errors while generating the HTML files. (10)**