Data Structures – Excersice 4

Due date: 1.4.2014 until 18:00. Put solutions in "Hadar Avodot", and submit a softcopy via Moodle.

The Assignment

In this assignment:

1. You will write a Java implementation of the Dynamic Disjoint Sets ADT (Union-Find).
2. You will load a bitmap representing an archipelago, and use the Union-Find structure to answer efficiently whether:
    a. Any two given land coordinates are connected by land (i.e. are on the same island)
    b. Any two given water coordinates are connected by water.
3. You will change the archipelago by efficiently adding bridges between land coordinates and canals between water coordinates.

Disjoint Sets

Write a Java class called UpTreeForest that provides the Union-Find functionality, using up-trees. The class should contain the following public methods:

- public UpTreeForest (int size);
- public void union (int i, int j);
- public int find (int i);
- public int getNumDisjointSets();

The constructor initializes the ADT with size sets, each containing a single element. The elements are numbered sequentially from 0 to size-1.

The union(i, j) method unites the sets that contain i and j. We assume the method is called with $i \neq j$, and i and j are the representatives of their sets. The method should use the *weighted union* methodology.

The find(i) method returns the representative of the set that contains i, and applied *path compression* to the traversed path.

The getNumDisjointSets () method returns the current number of disjoint sets.

You should also write a main() method which performs tests on the UnionFind class, but this should not be submitted as part of your solution.

Segmentation

Write a Java class called IslandsConnectivityChecker that reads a bitmap image that reads a bitmap image that represents the archipelago (black pixels are land, white pixels are water). Each pixel will be associated with a pair of coordinates (x,y): x is the column number starting from the left (zero-based), and y is the row number starting from the top (zero-based).
Example: In map with width 3 and height 4:

- The top left pixel is (0,0)
- The bottom left pixel is (0,3)
- The top right pixel is (2,0)
- The bottom right pixel is (2,3)

After reading the map, IslandsConnectivityChecker will segment the map into connected components. Two coordinates (x1,y1) and (x2,y2) will be in the same connected component if both of the following condition apply:

- (x1,y1) and (x2,y2) have the same land type (i.e. are both water, or are both land).
- If t is the land type of (x1,y1) and (x2,y2), there exists a path from (x1,y1) to (x2,y2) crossing only coordinates of land type t. The path goes through neighboring points. Each point has **4** neighbors (up\down\left\right) and **not 8** neighbors.

The class will have the following methods:

- public IslandsConnectivityChecker (String bmpPath);
- public void connectCoords (int x1, int y1, int x2, int y2);
- public boolean areConnected (int x1, int y1, int x2, int y2);
- public int getNumComponents();

The constructor accepts the name of a file containing a .bmp image. It reads the image using the provided ImageReader class (see below), and then creates an instance of UpTreeForest(see above), and uses it to segment the image into connected components.

The connectCoords (int x1, int y1, int x2, int y2) connects the connected components of (x1,y1) and (x2,y2). The method does so only if (x1,y1) and (x2,y2) are of the same land type, and not yet in the same connected component. Otherwise, the method does nothing.

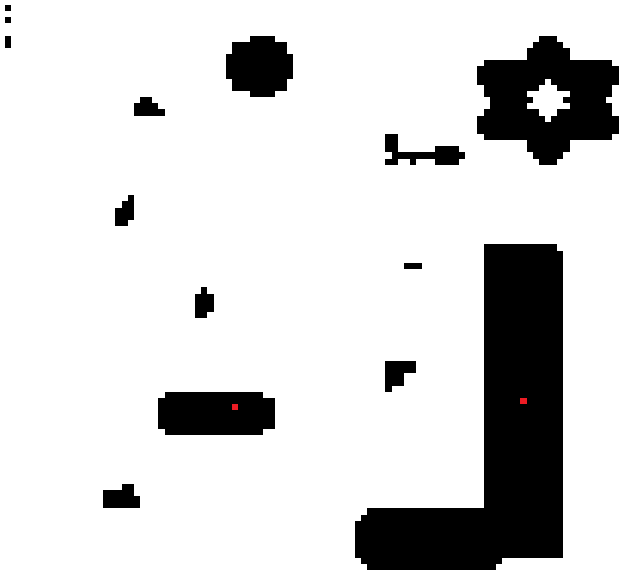
The boolean areConnected (int x1, int y1, int x2, int y2) method checks whether or not the two given pixels belong to the same component.

The getNumComponents() method returns the current number of components.

You should also write a main() method which performs tests on the Segmentation class, but this should not be submitted as part of your solution.

Example

This following image is given to you to test your code (Archipelago.bmp):



The image contains 16 connected components – 14 islands and 2 water regions.
If we connect the points marked in red – (37,65) and (84,64), we shall have 15 connected components – 13 islands and 2 water regions.
You are also given a very small archipelago (archipelago2.bmp) of size (3x3) to simplify debugging.

Implementation Details

- Each pixel is either white or black. Use ImageReader.readImage(string bmpPath) to read it. ImageReader.readImage will return an object of type Archipelago. The Archipelago has the following methods:
  getWidth() – returns the width of the map (= the width of the bitmap image).
  getHeight() – returns the width of the map (= the width of the bitmap image).
  getLandType(x,y) – return the landType of coordinate (x,y) (LAND\WATER)
- We use 4-connectivity, so each coordinate has 4 neighbors, not 8 (only the ones to the left/right or above/below are considered).
- When traversing the image pixels, we connect each pixel with its neighbor if they are of the same land type (both WATER or both LAND). An efficient implementation will process **each pair of neighboring pixels exactly once**.

General Guidelines

- You are not allowed to change any of the given *.java files.
- All methods should be readable and well documented.
- All your methods should be efficient.
- You may add classes and methods that were not defined in the given API, as long as they conform to proper Object Oriented Design.
- You are responsible for testing your code.
- You should provide full documentation of all classes and methods.

Grading

- Correctness: 50%
- Efficiency: 10%
- Robustness: 10%
- Architecture & Design: 10%
- Documentation & Readability: 10%
- Testing 10%

Submission Instructions

- Programming exercises can be submitted in pairs.
- You are given a main function (in UpTreeProject.java) to test your code.
  The main expects as the first (and only) argument the full path the Archipelago.bmp.
  (To add arguments in eclipse go to Run->Run Configurations->Arguments)
  The main function will test that your code runs correctly - **Make sure it runs correctly for your code** (**No errors).**
  You are not allowed to change the main function in any way.
- Do not add package declarations to java files
- Submit only *.java files (no class files, project files or JavaDoc files).
- The submitted zip file should include all files required for running – i.e., the following command set should work in the directory where the submitted zip file is opened:
  - javac *.java
  - java UpTreeProject Archipelago.bmp

    The result should be:

    expected 16 and got: 16
    expected true and got true
    expected true and got true
    expected false and got false
    expected false and got false
    expected true and got true
    expected false and got false
    expected true and got true
    expected 15 and got: 15
    expected false and got false
    expected true and got true
    expected 14 and got: 14
    Num of Errors: 0
- Hard-copy: printed works should be submitted to Hadar Avodot.
  - The printed material should include all the source code (*.java files), plus a cover page with your personal details.
- Soft-copy: all the source files (no need for a cover page) should be uploaded using the moodle.
  - All of the *.java files should be compressed into a single .zip file – everything required for running the given main function!
  - No folders in the zip file!

- The name of the zip file should include your name/s and the number of the exercise, in the following format:
  - (Two students) ID1_ID2_Ex04
  - (A single student) ID_Ex04
  - Example: 987654321_123456789_Ex04

The body of the email should also include your id number/s.