Joanna Klukowska
joannakl@cs.nyu.edu

Project 3: Image Segmentation by Region Growing

Due date:April 6, 11:55PM EST.

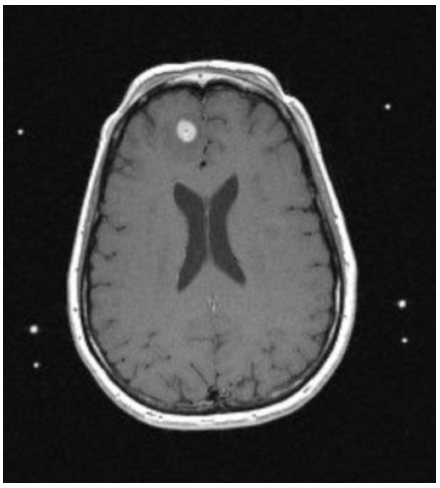
You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

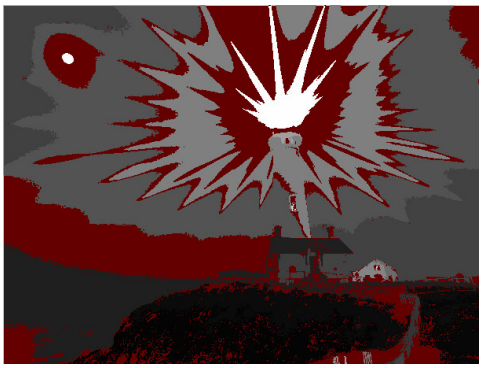
In this assignment you will work on the program that performs image segmentation using region growing algorithms.

Image segmentation is a process of partitioning a digital image into multiple segments. Usually, the segmentation simplifies the image and makes it easier to analyze. The segments supposed to represent meaningful regions of the original image.

Region growing approach is a simple approach to image segmentation. It starts with the user selected seed points and grows each of them into a region based on a region membership criterion. In this project a seed is a single pixel and the membership criterion is based on the gray level values of the pixels.

So, in short, we want to start with an image on the left hand side and obtain a segmented image like the one shown on the right hand side:





Joanna Klukowska
joannakl@cs.nyu.edu

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- class implementation using test driven development approach
- desiging and implementing unit tests
- implementation of a reference based stacks,
- implementation of a reference based queue,
- implementation of an algorithms based on the descriptions,
- working with existing code,
- implementing generic classes.

Start early! This program may not require you to write a lot of code, but debugging may be tricky. You should develop the code using the TDD approach: write the tests for your stack and queue classes and then write the code for the classes.

The Program Input and Output

This program uses input image files that are provided with the assignment. You may use your own image files, but make sure that the code that opens our test files remains unchanged when you submit the final version of the code.

The program writes the output to an output image file when used in test mode. In the interactive mode, the output is only visible in the window.

All input output operations are already implemented in the provided code.

Problem Description

You are given the program that performs image segmentation. The implemented algorithm is based on the depth first search approach to region growing. This solution needs to use a stack to keep track of the pixels that need to be visited and evaluated. Currently it is using the stack implementation from the Java API. **Your first task is to provide your own reference based stack implementation.**

The depth first search algorithm for creation of a single region from a seed is as follows:

Depth first search algorithm for region growing

```
INPUTS:
- seedX, seedY   are the coordinates for a seed
- value   is the gray level value to be used for the region
- outputImage   this is where the region is written to (should be filled with background
  color, may contain other regions)
- imputImage   the gray level image

ALGORITHM:
regionGrowingDFS ( seedX,  seedY,  value,  outputImage,  inputImage )
  create an empty stack S
  determine a threshold to be used

  push seed pixel (seedX,seedY) onto the stack S

  while stack is not empty
    currentPixel = pop the pixel from the stack S
    for each pixel in the 3x3 neighborhood of the currentPixel
      if the pixel is a background pixel in the outputImage
          and the pixel is within threshold of the currentPixel (in the inputImage)
        push pixel onto the stack S
        set pixel's gray level to value (in the outputImage)
```

Joanna Klukowska
joannakl@cs.nyu.edu

The above algorithm is already implemented in the method:

```
void getRegionDFS( int x, int y, int value, int [] pixels, int [] imagePixelsCopy )
```

An alternative approach uses a breadth first search approach to region growing algorithm. This solution needs to use a queue to keep track of the pixels that need to be visited and evaluated. **Your task is to implement a reference based queue and to implement the second approach.**

The breadth first search algorithm for creation of a single region from a seed is as follows:

Breadth first search algorithm for region growing

```
INPUTS:
- seedX, seedY   are the coordinates for a seed
- value    is the gray level value to be used for the region
- outputImage     is where the region is written to (should be filled with background
  color, may contain other regions)
- imputImage   the gray level image

ALGORITHM:
regionGrowingBFS ( seedX,  seedY,  value,  outputImage,  inputImage )
  create an empty queue Q
  determine a threshold to be used

  enqueu seed pixel (seedX,seedY) onto the queue Q

  while queue is not empty
    currentPixel = dequeue the pixel from the queue Q
    for each pixel in the 3x3 neighborhood of the currentPixel
      if the pixel is a background pixel in the outputImage
          and the pixel is within threshold of the currentPixel (in the inputImage)
        enqueue pixel onto the queue Q
        set pixel's gray level to value (in the outputImage)
```

This algorithm is implemented in the method:

```
void getRegionBFS( int x, int y, int value, int [] pixels, int [] imagePixelsCopy )
```

Program Modes

The program can be run in an interactive mode or in a test mode.

In the **interactive mode** the user manually selects the seeds using a mouse or a touch pad. The user has the following options to chose from (selectable by single key presses):

r - resets the image to the original

x - select random seed points

d - run the region growing algorithm (using depth first search) and show the results

b - run the region growing algorithm (using breadth first search) and show the results

In the **test mode** the program uses several images (names provided in the `testFileNames` array) with predefined seeds (seeds provided in the `testSeeds` array) to run either or both of the algorithms (the Boolean flags `RUN_TESTS_DFS` and `RUN_TESTS_BFS` determine which of the algorithms should run). The segmented images are saved to files on disk. No window is launched and no user interaction is possible in this mode.

Joanna Klukowska
joannakl@cs.nyu.edu

Unit Tests

Testing of the code that you implement is as crucial as writing it in the first place. For the two classes that you implement, you need to develop a set of unit tests that will demonstrate the class' correctness (or bugs) independent of the program itself.

The objective of unit tests is to test individual methods of the class in such a way that the results are as independent of the correctness of other methods as possible (this is generally hard to achieve, but you should keep that in mind as you are writing the tests). The unit tests should be based on the specification of the class, not on specific implementation.

Here are couple of examples of unit tests for class `MyStack`:

- create an object of the class and assign it to a reference variable; test if the reference variable points to null or not (it should not point to null) - problems here would indicate errors in the constructor
- create an object of the class and assign it to a reference variable; test if the object is empty using the `empty` method (it should be empty at this point) - problems here would indicate errors either in the `empty` method or in the constructor
- create an object of the class and assign it to a reference variable; add an element to the stack; test if the object is empty using the `empty` method (it should NOT be empty at this point) - problems here would indicate errors either in the `empty` method or in the `add` method
- ...

The tests should be short and test one thing at a time.

Computational Task

Implement the unit tests for the stack and queue classes

You need to write two test classes:

- `MyStackTest` - test class for the class `MyStack`
- `MyQueueTest` - test class for the class `MyQueue`

They both should have a lot of methods. The specific number does not matter, but you should convince yourself that 1) those tests are going to tell you when your own code is broken, AND 2) the tests will catch errors in our (i.e. graders') implementation of the two classes.

You do not need to document the test casses using Javadoc format. You do have to specify above each method what the method is testing and you need to follow the same code style conventions (formatting, naming, etc) as for any other code.

The unit tests should only use the public methods listed in the two classes below. If you chose to implement any additional private methods, they should not be used within the tests. (If they are, then your tests cannot be used for testing any other implementation of the two classes.)

Implement the `MyStack` class

Implement a reference based stack. Replace the stack used in getRegionDFS with your own implementation. Your stack must be generic. Your stack must provide a reference based implementation. It must provide the following methods:

`boolean empty()` Tests if this stack is empty.

`E peek()` Returns the element at the top of this stack without removing it from the stack.

`E pop()` Removes the element at the top of this stack and returns that element as the value of this function.

`E push(E item)` Pushes an item onto the top of this stack. Returns the item itself.

Joanna Klukowska
joannakl@cs.nyu.edu

The stack should also contain a default constructor that creates an empty stack object.

For details of each method you should see http://docs.oracle.com/javase/8/docs/api/java/util/Stack.html. Your own class has to implement the four methods according to the specification for the Java's **Stack** class (this means same bahavior, same parameter list, same return types/values, same exceptions thrown, ...). It does not need to implement the **search** method.

Any additional methods should be private.

Implement the MyQueue class

Implement a reference based queue. Your queue must be generic. Your queue must provide a reference based implementation. It must provide the following methods:

E peek() Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

E poll() Retrieves and removes the head of this queue, or returns null if this queue is empty.

boolean offer(E item) Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.

The queue should also contain a default constructor that creates an empty queue object.

For details of each method you should see https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html. Your own class has to implement the three methods according to the specification for the Java's **Queue** interface (this means same bahavior, same parameter list, same return types/values, same exceptions thrown, ...). It does not need to implement the other methods required by the **Queue** interface.

Any additional methods should be private.

Note1: The names of the methods in the interface are different than the names of the methods that we discussed in class. Their behavior, though, is the same.

Note2: The queue that you are implementing is NOT capacity-restricted. The **offer** operation should not fail due to limited capacity!

Using Provided Code

This program uses Processing package for the graphical interface and for working with images. You will need to configure your Eclipse to work with Processing. If you are running from the command line, the Processing package need to be included in the class path when the program is compiled and run. See below for details.

Working with Eclipse

You should download the **proj3_Eclipse.zip** file from the course website. Remember the location to which the file was saved.

Extract the zip file so that you can access individual files from within the **proj3_Eclipse** directory. Open this directory and Eclipse side by side.

Create a new project or select a project to which you want to import the files for this program.

You should see a file named **core.jar** in the extracted folder. Drag and drop that file over the name of the project in Eclipse (make sure it is placed directly into the project (not in **src** directory within the project). Right click on the file **core.jar** in Eclipse and select **Add to build path**.

Open the src directory in the extracted folder. Highlight all the image files and the **project3** directory and drag and drop them on the **src** directory in your project in Eclipse (all the files should end up inside the **src** folder).

Goto Run menu, select Run As and then Java Applet.

You can run the program RegionGrowing as any other program. You may need to select to run it as Java Applet, if prompted the first time you run it. Eclipse should have placed all image files in the correct location so that the program can find them.

Joanna Klukowska
joannakl@cs.nyu.edu

Working from a Terminal

You should download the `proj3_terminal.zip` file from the course website. Remember the location to which the file was saved.

Extract the zip file. Lets assume that the content of the zip file is in directory called `proj3_terminal`. You will be running all the commands from within the `proj3_terminal` directory. You should have a file `RegionGrowing.java`, a file `core.jar` and several image files. To compile the program execute

```
javac -cp core.jar *.java
```

To run the program execute:

```
java -cp ".:core.jar" RegionGrowing
```

To run the program in the test mode (with both tests included) execute

```
java -cp ".:core.jar" RegionGrowing test
```

Working on This Assignment

You should start right away!

You should modularize your design so that you can test it regularly. Make sure that at all times you have a working program. You should start by implementing the two test classes (or at least create the first draft of them). This will require a careful reading through the documentation and force you to think about the two data structures and their design before you write the code. You can then implement your own stack class, then implement the region growing algorithm that uses one of the classes that implement the Java's queue interface, and finally implement your own queue class. This way after each stage you have a working and testable program.

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens. (A second copy of the files on the same computer is a good idea to keep multiple versions, but it is NOT a good backup since you do not have access to it if there are problems with your computer.)

Grading

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

10 points efficient design of the data structures

50 points program and class correctness: the correct values and format of output, correct behavior of methods, handling of invalid arguments

10 points design and implementation of the two required classes

10 points implementation and quality of the unit tests

20 points proper documentation, program style and format of submission

The program will not receive any correctness and design credit for implementations of stack and queue that are NOT reference based implementation (DO NOT use the array based implementation that we discussed prior to spring break).

Joanna Klukowska
joannakl@cs.nyu.edu

How and What to Submit

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to NYU Classes.

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select "Save in zip format", "Compress the contents of the file" and "Create only selected directories"
- click Finish

Do not submit any image files in your zip file!