CITS2200 Project 2020

You must work individually for this project.

Problem Specification

You must implement the following four functions, and provide a report on your implementations. Each function is worth a different number of marks for a total of 20 marks. Details for what the mark breakdowns mean are provided below.

All questions work with a greyscale image specified as a 2D `int` array. The array is indexed first by row, then by column. Every row in the array will be the same length. Every element in the array will be non-negative and no greater than 255. A value of 0 represents a black pixel, and a value of 255 represents white, with shades of grey in between.

Time complexity specifications use `R` for number of rows, `C` for number of columns, and `P = R*C` for number of pixels.

`public int floodFillCount(int[][] image, int row, int col)` (4 marks)

Compute the number of pixels that change when performing a black flood-fill from the pixel at (`row`, `col`) in the given image. A flood-fill operation changes the selected pixel and all contiguous pixels of the same colour to the specified colour. A pixel is considered part of a contiguous region of the same colour if it is exactly one pixel up/down/left/right of another pixel in the region.

Marks (4 total):

- Correctness: +2 marks
- Complexity:
  - `O(P)`: +1 mark
- Quality: +1 mark

`public int brightestSquare(int[][] image, int k)` (5 marks)

Compute the total brightness of the brightest exactly `k*k` square that appears in the given image. The total brightness of a square is defined as the sum of its pixel values. You may assume that `k` is positive, no greater than `R` or `C`, and no greater than 2048.

Marks (5 total):

- Correctness: +2 marks
- Complexity:
  - `O(Pk)`: +1 mark
  - `O(P)`: +1 mark
- Quality: +1 mark

`public int darkestPath(int[][] image, int ur, int uc, int vr, int vc)` (5 marks)

Compute the maximum brightness that MUST be encountered when drawing a path from the pixel at (`ur`, `uc`) to the pixel at (`vr`, `vc`). The path must start at (`ur`, `uc`) and end at (`vr`, `vc`), and may only move one pixel up/down/left/right at a time in between. The brightness of a path is considered to be the value of the brightest pixel that the path ever touches. This includes the start and end pixels of the path.

Marks (5 total):

- Correctness: +2 marks
- Complexity:
  - `O(P log P)`: +1 mark
- Quality: +2 mark

`public int[] brightestPixelsInRowSegments(int[][] image, int[][] queries)` (6 marks)

Compute the results of a list of queries on the given image. Each query will be a three-element int array `{r, l, u}` defining a row segment. You may assume `l < u`. A row segment is a set of pixels (`r`, `c`) such that `r` is as defined, `l <= c`, and `c < u`. For each query, find the value of the brightest pixel in the specified row segment. Return the query results in the same order as the queries are given.

Marks (6 total):

- Correctness: +2 marks
- Complexity: (where `Q` is the number of queries)
    - `O(PC + Q)`: +1 mark
    - `O(P log C + Q log C)`: +1 mark
    - Faster is possible but will not receive additional marks
- Quality: +2 marks

Required Work

You must write a public class called `MyProject` that implements the provided `Project` interface (see `Project.java`)

- Your class must have a zero-argument constructor, which will be the only one used when marking
- Your code must be in a file named `MyProject.java`
- `MyProject.java` must include your full name and student number as a comment as the first line of the file
    - For example: `// Ada Lovelace (21234567)`
- DO NOT modify `Project.java` in any way
- You may use anything from the Java standard library
- You may **NOT** use anything from the CITS2200 lab package or any other sources

You must write a report explaining your implementation

- Your report must be provided as a PDF named `report.pdf`
- Your report must for each problem:
    - Explain why your chosen algorithm will give the correct answer (that is, a logical argument for why it is correct)
    - Provide an analysis explaining the time complexity of your implementation (and memory complexity if relevant)

Testing

Your code must compile correctly when compiled with:

```
javac Project.java MyProject.java SampleProjectUnitTest.java
```

After compiling, you can test your code using the provided sample unit tests with:

```
java SampleProjectUnitTest
```

You are encouraged to copy `SampleProjectUnitTest.java` and extend it to add your own tests.

Assessment

Each problem will be marked separately and the results combined for a total of 20 marks (mark distributions provided above). For each problem, you will receive marks based on:

- Correctness
    - Compiles without error
    - Gives correct results
- Complexity
    - Efficient code
    - Complexity targets are provided above
- Quality
    - Convincing and well presented arguments in report
    - Code readable and easy to follow, including sufficient commenting

For all components, you will be assessed on how well the evidence demonstrates your understanding of the content.

- Your submitted work will serve as evidence of understanding of the project content.
- Any work that is not your own original work does not constitute evidence of understanding.

- You are permitted to research prior work that others have done on these problems, but any work you reference must be cited in your report.
- Your code must be wholly your own original work.

Submission

Submit only `MyProject.java` and `report.pdf` via cssubmit.