Data Structures and Algorithms II
COMP 2631 (Winter 2015)

Lab 2 — Hacking Bitmap (BMP) Files

Total points: 20

Note: The Bitmap specification + sample image files are posted on Moodle.

Overview

The goal of this lab is to hack into uncompressed Bitmap (BMP) files. Bitmap files are simple image files that are common on the Windows platform (e.g., for wallpaper). You have been given a document describing the BMP format, written by Wim Wouters and available through `www.wotsit.org` (which has information about a large number of file formats).

Basic Setup

Create a class named `BitmapHacker`, and include a `main` method that repeatedly presents the user with a menu like this (in the console output area or in a dialog window):

```
0. Exit
1. Flip image top/bottom
2. Enhance color
3. Blur image
4. Combine two images

Enter your choice ---->
```

For each choice other than 0, call a method that performs the indicated operation (details below). Note that you can make all of these methods `static`. Each method should prompt the user for the full path name of an input BMP file (two input files for Choice 4) as well as the full path name of an output BMP file. It should then read in the input file(s) and create the appropriate output file. Note: ***The input file(s) should not be modified.*** After the method is complete, the user should be returned to the main menu.

Use `RandomAccessFile` for reading and writing the BMP files. Make sure you *close* all files that you open in order to avoid OS-level problems (e.g., an image viewer may not display the image if the file is still open; also, changes to a file may not take effect until the file is closed).

Sample Files

Several sample BMP files are included, with a variety of widths and heights. *Make sure your methods work with all the sample image sizes.*

2-D Arrays of Pixels

Write a simple class called `Pixel` that has three `int` instance variables named `red`, `green`, and `blue`. Include a constructor with three parameters that will be assigned to the three instance variables. The constructor should enforce the restriction that each pixel color component must lie in the range $0 \ldots 255$ (use a class constant for the value 255). Your class should also have an accessor and a mutator method for each instance variable, and each mutator method should ensure that only appropriate values are assigned to its respective color component in the same manner as the constructor. (One way to set things up is to put the checks in the mutator methods and then have the constructor call these.)

Whenever a method in `BitmapHacker` reads in the pixel data from a Bitmap image, place this information in a 2-D array of `Pixel` objects (the "source"). Then when applying a manipulation to this image, create a second 2-D array of `Pixel` objects (the "target") and assign RGB color values to the target `Pixel` objects based on the appropriate manipulation of the source (which should remain untouched). Finally use the data in the target 2-D array to create the new BMP file. If you need to work with a larger number of Bitmap files for a given manipulation, construct the appropriate number of 2-D arrays.

Image Manipulation Methods in `BitmapHacker`

1. This method should flip the input image upside down, i.e., the order of the pixel rows should be reversed.

2. When this method is called, it should also prompt for a preferred primary color (red, green, or blue). For each pixel, do something to enhance the preferred primary color (leave the others unchanged). You can experiment here to see what is effective. For example, you could add a fixed integer to the preferred color (maxing out at 255, of course). Or you could increase the preferred color by a fixed percentage, say 50%. *Include a comment in your code explaining your ehancement.*

3. This is a bit more sophisticated. For each pixel in the original input image, the corresponding pixel in the output image should be the average of the original pixel and its 8 neighbors (unless the pixel is on the edge, in which case form the average of itself and whatever neighbors it has). Perform this averaging component-wise, i.e., average each of the three primary colors in turn.

   This manipulation should produce a softening (blurring) of the image. See what happens if you call this method multiple times, producing a sequence of images each of which is the blurred version of the one before it.

4. This method should prompt the user for *two* input files and an output file. If the two input images are *not* the same size (in terms of numbers of rows and columns), print an error message (and return to the menu). Otherwise, the output image should be the pixel-by-pixel average of the two input images (again, average each of the primary colors in turn). This should produce some visually interesting results.

Remarks / Things to Remember

- You can assume that all input files are uncompressed.
- You can also assume that all images use the 24-bit color standard, i.e., each pixel is stored using 1 byte for each of the red, green, and blue components. This has a couple of implications:
  - the *Bits Per Pixel* field in the BMP header will be equal to 24
  - the *Palette* field, which consists of $N \times 4$ bytes, will be empty, so $N = 0$
- ***Achtung!*** The 3 bytes storing the color components of each pixel are in the order blue, green, red.
- The rows of the image in a BMP file are stored *bottom to top,* i.e., the bottom row is given first.
- The number of bytes used to store each row must be a multiple of 4 (the end is padded with zeros if necessary). This is not a problem if the number of pixels per row is divisible by 4 (e.g., 640), but might be an issue for other image sizes. Keep this in mind so that your code will be able to handle any BMP file (e.g., `rovers.bmp`).
- Before you work on the main lab exercises, see if you can first correctly extract a 2-byte ("word") or 4-byte ("dword") value from the BMP header and interpret it as an unsigned integer. (It might be useful to write a small helper method.) For example, can you extract the width and height of the image?
- Use Javadoc style comments.

Submitting Your Work

Submit `Pixel.java` and `BitmapHacker.java` to Moodle.