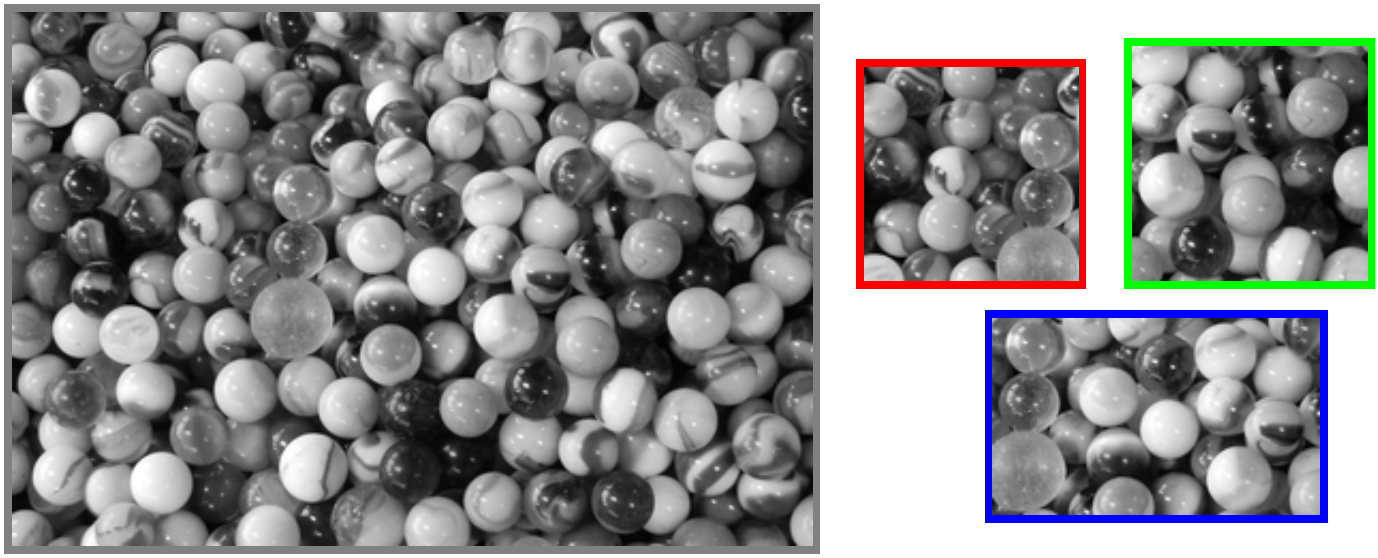
CSCI-1200 Data Structures — Fall 2016
Homework 9 — Hash Tables for Image Comparison

In this assignment you will dramatically improve the performance of a simple image comparison program. This homework is inspired by a real world application of computer vision to the problem of object recognition. For example, in manufacturing, a common task is to locate and identify specific parts as they roll down an assembly line. Another example is scanning the barcode or "QR" code on your boarding pass at the airport.

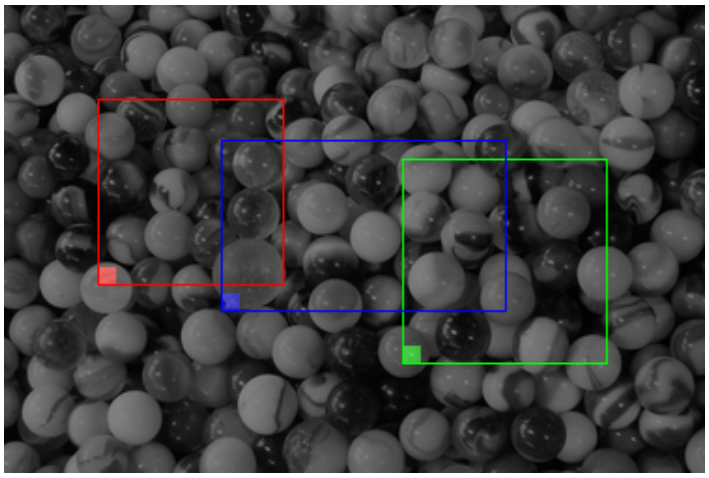
Problem statement: Given a set of greyscale images, compare the pixels in each image to the pixels in every other image in the set and discover if the images have similar subregions. Your program should then output the coordinates of the overlapping regions and the approximate percent of pixels that match. Additionally, to aid in debugging and grading, you will output an image visualization of the overlap.

Let's look at a specific example. Suppose we are given as input the four black and white images below. Can you see the overlapping regions? The smaller images are in fact cropped from the larger image.



In the provided code we tackle this problem by looping over every point (x_A, y_A) in image A and every point (x_B, y_B) in image B. Then we compare pixel-by-pixel a small *seed* (of size s x s) surrounding these two points. Once we find a matching seed between the images, we break out of the nested loops. Next, starting from this small seed, we grow the matched region in width and then in height until we hit the edge of one of the images or a mismatched pixel.

A visualization of this algorithm is shown on the right. The initial small square seed match for each pairwise image comparison is highlighted and the larger colored outline shows the extended region of the final match. This simple algorithm will find a subregion match between two images if one exists; however, it will not necessarily find the largest matching subregion. Furthermore, if the image contains small differences or "noisy" pixels, the algorithm will return a smaller region. Finally, this algorithm is inefficient, especially when processing a bigger set of high-resolution images. We'll use hashtables to do better!



Expected Output

Here is expected output for the image comparison of this set of four images. Your program should compare each image specified on the command line to every other image and report the *approximate* percentage of matching pixels between the images and the bounding boxes of similar regions of pixels. Your program will also produce the visualization images of these regions shown on the previous page.

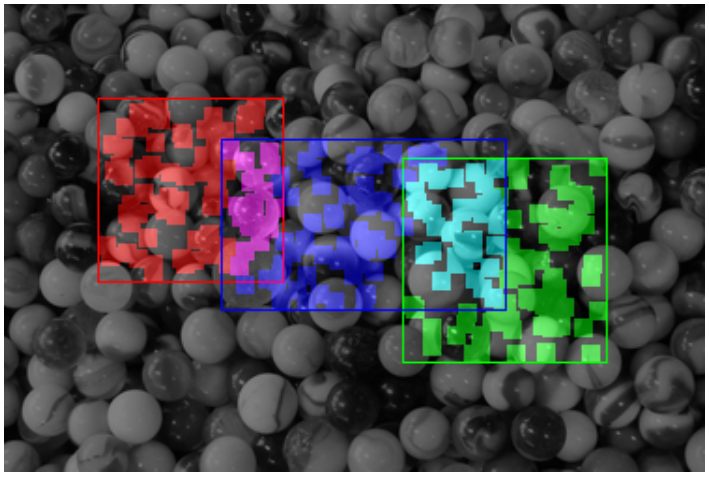
```
marbles.pgm
   10.7% match with marbles_1.pgm          (50,100):(149,199) similar to (0,0):(99,99)
   12.9% match with marbles_2.pgm          (213,58):(322,167) similar to (0,0):(109,109)
   15.0% match with marbles_3.pgm          (116,86):(268,177) similar to (0,0):(152,91)
marbles_1.pgm
  100.0% match with marbles.pgm            (0,0):(99,99) similar to (50,100):(149,199)
    0.0% match with marbles_2.pgm
   26.5% match with marbles_3.pgm          (66,0):(99,77) similar to (0,14):(33,91)
marbles_2.pgm
  100.0% match with marbles.pgm            (0,0):(109,109) similar to (213,58):(322,167)
    0.0% match with marbles_1.pgm
   38.0% match with marbles_3.pgm          (0,28):(55,109) similar to (97,0):(152,81)
marbles_3.pgm
  100.0% match with marbles.pgm            (0,0):(152,91) similar to (116,86):(268,177)
   18.8% match with marbles_1.pgm          (0,14):(33,91) similar to (66,0):(99,77)
   32.6% match with marbles_2.pgm          (97,0):(152,81) similar to (0,28):(55,109)
```

Performance Analysis

First, let's analyze the simple algorithm. Let's assume the program is given i images, each of average size width=w x height=h, using a seed of size s, with a typical final subregion match of size r x r. What is the Big 'O' order notation for the expected running time and memory use of this algorithm? Write a short justification of your answer in your `README.txt` file.

Image Comparison with a Hash Table

To improve the performance of the nested loop simple algorithm, we will instead hash all s x s seed blocks from each image (shifting 1 pixel up and/or right at a time), and store these hashed values in a large array (a.k.a. hash table) for each image. Then, to compare two images, we simply walk over the hash tables for these two images and collect the matching hashed seeds.



A visualization of the hashing version of the algorithm is shown on the left. The highlighted blocks are the pairwise matching seeds between the large image and each of the three smaller images. Note that because we have a huge number of overlapping seeds and perfect hashed value matches between these images, we do not need to walk through the entire hash table. Instead, it is sufficient to examine a tiny fraction of the seeds. In this case we look at just 1 percent of the hash table (`compare` = 0.01). By simply taking the bounding box of these matching seeds we can get a very good approximation of the matching image subregions.

After you complete the implementation and testing of the improved improved hash table algorithm, compare the two versions of the program. What is the Big 'O' notation for running time and memory use using the variables above, plus t = table size and c = fraction of hash table to compare? Test your implementation on a variety of images and command line arguments and summarize and discuss the results (both running time and quality of results) in your `README.txt` file.

Command Line Arguments

The program will take several *optional* command line arguments (method, seed size, table size, and fraction of the hashtable to compare) on the command line, followed by two or more greyscale images in `.pgm` format. For example:

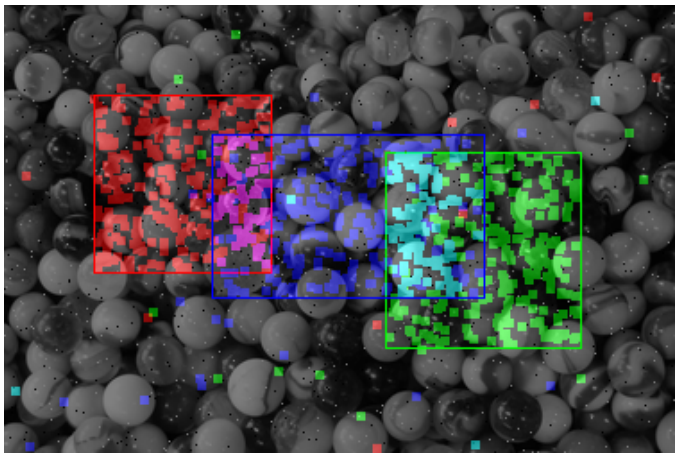
```
./image_search.exe -method simple -seed 10 marbles.pgm marbles_1.pgm marbles_2.pgm marbles_3.pgm

./image_search.exe -method hashtable -seed 5 -table 1000000 -compare 0.01 \
        marbles.pgm marbles_1.pgm marbles_2.pgm marbles_3.pgm > cout_hashtable.txt
```

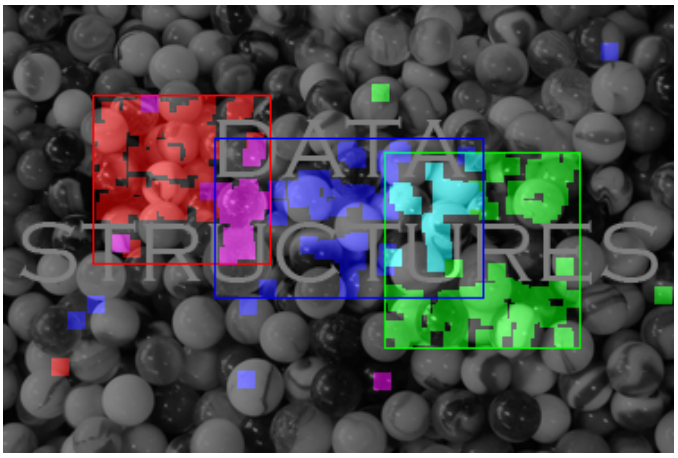
Viewing `.pgm` and `.ppm` Images

Your program will work with image data using the standard "Portable Pixel Map" format. The input will be greyscale `.pgm` images (each pixel is an integer ranging from 0=black through 255=white). The visualization output of your program will be color `.ppm` images, with 3 integers per pixel representing the red, green, and blue components of color, each in the range of 0-255. Many standard image viewing programs (Photoshop, GIMP, `xv`) will load and display `.pgm` and `.ppm` images. On UNIX/Cygwin, ImageMagick can be used to convert between image formats, such as the popular "Portable Network Graphics" format, `.png`. For example: `convert.exe tmp.ppm tmp.png`

Noisy Images, Image Edits, Multiple Region Matches, and Rotation

On the left we see hash table matching results when a small number of the pixels in the images are corrupted by "salt & pepper" noise. On the right, we see results when the large image in the set is edited to add a text overlay. Note that it may be necessary to edit the seed size and compare parameters to get reasonable quality results. Note in both cases we have incorrect seed matches caused by coincidental collisions in the hash table. How will you ignore false positive matches and still detect large subregions of near similarity?





For extra credit, extend your program to find multiple subregion matches per image pair. Additionally, you can modify your program to handle cases where the matching regions are rotated by 90°, 180°, or 270°, or flipped horizontally or vertically. Note that even though our hash table algorithm is much faster, the method is still quite primitive. Active research in computer vision aims to solve the general problem of object detection and recognition in more challenging situations, including arbitrary image scale and rotation, skew or distortion, illumination changes, and camera position changes.

Provided Code and Homework Submission

Please study the provided code for parsing command line arguments and loading and saving image files with the templated `Image` class. You may modify the provided code. **You must do this assignment on your own, as described in the "Academic Integrity for Homework" handout. If you did discuss the problem or error messages, etc. with anyone, please list their names in your `README.txt` file.**