CS300 - Fall 2024-2025 Sabancı University Homework #3
Basic Image Detection With HashTable

1 Introduction

Image detection and recognition systems play a vital role in various modern technologies, from security applications like fingerprint recognition to facial recognition in smartphones and intelligent image search engines. These systems use sophisticated algorithms to identify, store, and search for patterns within images, enabling us to automate complex identification tasks with high accuracy. The demand for effective image detection has increased as we aim to build systems that can recognize objects, people, or even specific patterns within an instant.

In this homework, you will dive into the fundamentals of image detection by implementing a basic program that can store and search for binary images. You will work with 28x28 binary images provided in text files, simulating a mock image detection system. Your task will be to read the binary images from text files, convert them into their **Run Length Encoding** representations, and store these encodings in a hash table.

When a query image is presented, your program will search for it within the hash table. If a match is found, the program will reconstruct the image that is in Run Length Encoded form and print it on the console. If not, it will notify that "No match for the image with encoding: **actual RLE of the image**" and output the Run Length Encoding of the query image for comparison.

Through this assignment, you will explore core concepts in image encoding, hashing, and data retrieval, building a foundation for more complex image detection applications. Although modern image detection systems are highly sophisticated, often dealing with RGB or grayscale images, we will start with a simplified approach. In this assignment, you will work with binary images and use hash tables to demonstrate basic image detection concepts. This focus on binary images is intentional, as handling RGB or grayscale images would require significantly more complex algorithms and is beyond the scope of this introductory task.

Figure 1: Binary image to be worked with

```
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000011111000000000000000
0000000011111111000000000000
0000000011111111000000000000
0000000011111111100000000000
0000000011111111110000000000
0000000000011111111100000000
0000000000011111111110000000
0000000000111111111110000000
0000000011111111111100000000
0000000011111111111000000000
0000000011111000000000000000
0000000011111000000000000000
0000000011110000000000000000
0000000011110000000000000000
0000000011110000000000000000
0000000011110000000000000000
0000000011110000000000000000
0000000011110000000000000000
0000000011110000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
```

Figure 2: text file representation of image (28x28)

In binary images we have 2 values, 0 which represents black and 1 which represents white. Above example demonstrates the image that will be given you in the text file and its actual binary image. 1 represents foreground or white pixels and 0 represents background or black pixels. In order to put those images to hash table and look for them, we will apply one of the most basic lossless data compression algorithms, which is **Run Length Encoding** and turn those images to strings and store them in hash table.

2 Run Length Encoding

Before we can apply **Run Length Encoding (RLE)** to the binary image data, we first need to flatten the image. The binary image is provided as a 28x28 grid in a text file, where each row of the grid represents one line in the file. To flatten the image, read the file line by line and concatenate each line into a single continuous string of 0s and 1s. This flattened string will represent the image in a one-dimensional format, which is required for the encoding process.

Once the image has been flattened, we can apply Run Length Encoding. RLE is a straightforward lossless data compression algorithm that replaces consecutive identical values with a single value followed by the count of repetitions. This approach is particularly efficient for binary images, where long sequences of 0s or 1s are common. For example, a sequence like "0000001111" would be compressed to "6B4W", where "W" stands for white pixels and "B" for black pixels.

The RLE process for our binary images involves scanning through the flattened string from left to right, counting consecutive 0s (black pixels) or 1s (white pixels) and creating an encoding format that alternates between the counts of white and black pixels. This encoded representation will be much shorter than the original data and can easily be stored in a hash table for quick retrieval.

In this assignment, once the binary image is encoded using RLE, you will store this encoding in a hash table. When a query image is provided, you will flatten and encode it as well, then search the hash table to check if a matching encoded image already exists. If a match is found, the program will print the RLE of the stored image; otherwise, it will indicate that "no image was found in the database" and output the RLE of the query image for reference. Here is the overview of the algorithm.

Algorithm 1 Run Length Encoding (RLE)

Require: Flattened string of binary image, `flattened`
Ensure: Encoded RLE string, `rle`
Initialize an empty string `rle`
Initialize `count` as 1
Set `current` to the first character of `flattened`
for each character in `flattened` starting from the second **do**
  if character equals `current` **then**
    Increment `count` by 1
  else
    Append `count` and `current` (as "W" or "B") to `rle`
    Set `current` to the new character
    Reset `count` to 1
  end if
end for
Append `count` and `current` (as "W" or "B") to `rle` (for the last sequence)
return `rle`

Once the Run Length Encoding is ready, we will store these strings in hash table.

3 Hash Table Implementation

In this section, we provide details on the implementation of the hash table for inserting and finding encoded images. The class `HashTable` is designed to store and retrieve Run-Length Encoded strings efficiently by organizing them into a fixed-size array of entries.

Hash Function

The hash function is a critical part of the hash table, responsible for mapping each encoded string (our "key") to a unique position in the array. It is your responsibility to find a good hash function for this homework. In the end your hash table should accurately handle collisions and print back the correct image if it exists in the table and should print the Run length encoding of the image if it does not.

Collision Handling with Linear Probing

Despite our efforts to avoid collisions, some keys may still map to the same position. In this implementation, we handle collisions using **linear probing**. When a collision occurs, the algorithm checks the next position in the array. If this position is also occupied, it moves forward one position at a time until it finds an empty slot.

Linear probing is simple and effective, especially in situations where the hash table is not densely populated, making it easy to find an open spot.

Class Overview

The `HashTable` class includes functions for inserting, finding, and removing encoded strings:

- `insert`: Finds an appropriate position using the hash function and inserts the string if it is not already present.

- `find`: Searches for a string by computing its position and using linear probing if necessary. If found, it returns the string; otherwise, it returns a special value indicating that the item was not found.

By leveraging this hash function and linear probing, the hash table provides an efficient way to store and search for encoded images, enabling rapid insertion and retrieval based on the Run-Length Encoded keys.

You may add additional structures to your class implementation if you wish, but these structures or functions should not come from an external module.

Printing Format

Once you have found the image in the Hash Table, you will print its corresponding Run-Length Encoding. The specific output format for this operation is provided in the Sample Runs section.

Important Remark: You can be confident that no two different images will produce the same RLE, so each image's RLE will be unique. However, even with unique RLE strings, it is still possible for two different RLEs to yield the same hash value due to the nature of the hash function. The hash function calculates the position in the Hash Table based on the encoded string, but in rare cases, two unique RLEs may map to the same hash value.

If such a collision occurs, the implementation will handle it through **linear probing**. When two RLEs share the same hash value, the algorithm will check subsequent slots in the array until it finds an empty or matching slot. This ensures that each RLE can be correctly inserted and retrieved from the Hash Table, even in cases of hash collisions. Thus, the system is robust against rare hash collisions and ensures accurate retrieval of each encoded image.

Your class does not have to be templated, but if you want to do it template based, there is no restriction.

4 Program Input and Output

You will be given a folder of images for the hash table to store, and then query images. All images in the folder will be inserted into the hash table sequentially, named in the format `imageX.txt`, where X is a number.

After inserting, you may input a `queryX.txt` file to the program, where X is the query image number. For each query image, if the encoded string is found in the hash table, the program will decode and display the 28x28 binary image on the console. If no match is found, the program will output the run-length encoded (RLE) string of the queried image and notify the user that no match was found.

You cannot assume anything about the number of images or number of queries to be inputted to the system but your program should work with any number of files. You may assume that each image file is named as imageX and each query file is named as queryX where X is a number. You can also assume that X cannot take any negative value.

To end input taking process, you should type in `"query"` and to end the querying process, input `"exit"` to exit the program.

Your output shoulde exactly match CodeRunner.Any deviation from CodeRunner's output will receive zero from that test case.

5 Program Flow

This assignment can be thought as having two parts: image insertion and query processing. In the image insertion part, your program must ask for the numbers of the image files, and at the query processing part your program must ask for numbers of the query files. You can assume that these files always exist.

5.1 Image Insertion

When your program starts executing it will be at the image insertion part. In this stage, your program must print `"Enter image number to insert into the hash`

`table (or 'query' to continue): \n` " to ask for input from the user. You can assume that the input given from the user will be either "query" or a number. You can also assume that if a number N is given, the file "image$N$.txt" exists at the same directory.

If the user types a number, your program must open the related file and process the image inside, as explained earlier. After your program is done with that file, it must ask for a new input by typing "`"Enter image number to insert into the hash table (or 'query' to continue): \n`", and repeat until it takes "query" as the input. If the user types "query", your program must move to the next part.

5.2 Query Processing

At this part, your program will process the query images and check whether that image was inserted or not. First, your program must print "`Enter image number to query (or 'exit' to quit): \n`" to ask for input from the user. This time, the input will either be "quit" or a number. Similarly, if the input is anything other than "quit", you can assume that the file "query$N$.txt" exists for input N in the same folder.

If the user types a number, your program must process the relevant file as explained earlier. Then, it will print "`Enter image number to query (or 'exit' to quit): \n`" and repeat until it takes "exit" as an input. If the user types "exit", your program must terminate.

6 Sample Runs

The following section provides an example of a sample run, showcasing the insertion of images into the hash table and querying images by filename.

Sample Run 1

```
Enter image number to insert into the hash table (or 'query' to continue): 1
Image 1 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 2
Image 2 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 3
Image 3 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 4
Image 4 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 5
Image 5 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): query
Enter image number to query (or 'exit' to quit): 2
No match for the image with encoding: 784W
Enter image number to query (or 'exit' to quit): 1
No match for the image with encoding:
```

```
12B2W25B3W24B3W5B1W1B1W16B3W5B2W1B2W15B3W4B2W2B3W13B3W5B1W3B3W12B3W5
B1W4B3W12B3W4B2W5B2W12B2W5B1W6B2W12B2W4B2W6B2W12B2W3B2W7B2W12B2W2B3W
7B2W12B2W2B2W8B1W12B2W2B2W9B1W12B6W8B2W12B5W9B2W12B4W10B2W11B4W11B2W
11B4W11B2W10B5W11B3W9B5W11B4W7B6W12B4W6B6W13B3W6B6W13B3W6B1W2B3W12B3
W10B2W12B2W12B2W11B2W13B1W12B1W8B
Enter image number to query (or 'exit' to quit): exit
Exiting the program!
```

Sample Run 2

```
Enter image number to insert into the hash table (or 'query' to continue): 1
Image 1 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 2
Image 2 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 3
Image 3 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 4
Image 4 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 5
Image 5 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): query
Enter image number to query (or 'exit' to quit): 6
RLE String for query6.txt found in hash table.
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
```

```
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
10101010101010101010101010
01010101010101010101010101
Enter image number to query (or 'exit' to quit): exit
Exiting the program!
```

Sample Run 3

```
Enter image number to insert into the hash table (or 'query' to continue): 1
Image 1 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 2
Image 2 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 3
Image 3 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 4
Image 4 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 5
Image 5 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 6
Image 6 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 7
Image 7 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 8
Image 8 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 9
Image 9 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): 10
Image 10 inserted into the hash table.
Enter image number to insert into the hash table (or 'query' to continue): query
Enter image number to query (or 'exit' to quit): 4
No match for the image with encoding:
2B1W1B1W3B1W2B1W1B1W3B4W3B3W4B1W2B1W1B1W1B5W3B1W1B1W1B1W1B2W1B1W1B3W5B1W2B1W1B2W1
B6W1B1W1B1W1B1W1B1W3B1W2B2W1B1W1B1W1B2W1B2W1B5W2B1W4B2W2B1W2B4W1B1W2B1W1B1W1B1W1B
2W1B1W1B3W2B1W3B1W1B1W5B4W1B2W2B1W1B1W2B1W1B1W1B2W1B4W4B1W3B4W1B2W1B2W5B1W1B1W1B2
W1B6W1B3W1B2W4B2W1B1W3B7W1B3W1B1W1B1W1B4W3B3W3B4W1B1W1B1W3B1W3B5W2B2W1B2W2B2W1B2W
3B1W1B4W1B2W5B1W2B1W2B1W1B7W1B1W2B4W2B1W2B4W1B4W2B1W3B2W1B1W3B1W1B2W2B1W1B1W2B2W5
B1W1B2W3B1W2B2W4B1W5B3W1B3W6B1W2B1W9B1W2B1W3B1W2B1W1B1W1B1W1B1W1B2W2B1W2B3W1B3W3B
5W2B1W1B1W1B1W1B1W2B1W1B1W3B1W1B1W1B1W1B2W3B2W3B2W1B2W1B2W1B4W2B2W3B2W1B2W1B1W5B3
W2B3W1B2W1B1W3B1W1B2W1B6W4B2W3B2W4B3W2B1W3B3W3B1W3B1W1B2W3B2W1B2W1B3W2B1W1B2W1B3W
2B1W1B1W2B1W1B4W1B2W1B1W2B1W1B2W2B1W2B4W1B1W8B1W2B1W2B2W1B1W4B1W1B1W2B1W3B4W1B4W1
B4W1B8W1B1W2B2W5B1W1B1W2B1W1B1W2B1W2B1W1B1W4B4W2B1W2B1W2B1W4B2W1B3W4B1W1B1W1B3W1B1W
Enter image number to query (or 'exit' to quit): 7
RLE String for query7.txt found in hash table.
00000000000000000000000000
```

```
00000000000000000000000000
00000000000000000000000000
00000000000000000011000000000
00000000011111111111111111100
00000000011111000000000011100
00000000000000000000001111000
00000000000000000000111100000
00000000000000011111000000000
00000000000001111100000000000
00000011111110000000000000000
00000111111111110000000000000
00000000000011111111111100000
00000000000000000000000111000
00000000000000000000000001100
00000000000000000000000111100
00000000000000000000111110000
00000000000000001111100000000
00000000000111111000000000000
00000011111110000000000000000
00001111100000000000000000000
00111000000000000000000000000
00100000000000000000000000000
00100000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
Enter image number to query (or 'exit' to quit): exit
Exiting the program!
```

7 Submission and Grading Guidelines

- Submissions will be done via SUcourse with CodeRunner in order to prevent compilation errors
- Ensure that this file is the latest version of your homework program.
- Submit via `SUCourse` ONLY.
- Submissions through e-mail or other means (e.g., paper) will receive no credit.
- Submit your own work, even if it is not fully functional. Submitting similar programs is easily detectable.

Good Luck!
CS300 Team