

# Project 2 Report: Best Hashing Function

## Team Members

Saurabh Mishra  
CWID: 887579779  
Email: [saurabhm2906@csu.fullerton.edu](mailto:saurabhm2906@csu.fullerton.edu)

Maziar Bastani  
CWID: 890312985  
Email: [maziarbastani@csu.fullerton.edu](mailto:maziarbastani@csu.fullerton.edu)

CPSC 535 (Spring 2020)

Dr. Doina Bein  
Department of Computer Science  
California State University, Fullerton  
May 15, 2020

## Table of Contents:

### [Problem Definition](#)

[Summary](#)

[Input](#)

[Output](#)

### [Project Methodology](#)

[Steps](#)

[File Info Retrieval](#)

[Hash Functions](#)

[Adding Item to Hashtables](#)

[Removing Item From Hashtable:](#)

[BestHashing Method](#)

### [Project Implementation](#)

[File Info Retrieval](#)

[Hash Functions](#)

[Adding Item to Hashtables](#)

[Removing Item From Hashtable](#)

[BestHashing Method](#)

[How to Run the code](#)

[Output Screenshots](#)

## Problem Definition

### Summary

This project is designed to help a store in our case a glass store in a balanced way. The concept is each pair of glasses has a 7-digit barcode, we want to use this barcode as a reference for our hashing function. We can use the first through the seventh digit of these hash functions. The glasses could be placed in a cubby based on the value returned from their hash function. For example, if the first digit is 1 it's in the first location of the cubby. It has 10 locations.

This algorithm finds which digit should be used in our hashtable to have the most balanced hashtable.

The definition of a balanced hashtable is the table in which the difference between the minimum bucket size and the maximum one is less than the other tables. The project definition in the following link clarifies the details.

<https://docs.google.com/document/d/16PF-B2-PZtCX0wWbYAGrSabUyJ-GHR1AB9C-tP98lmg/edit>

### Input

Files with an extension of .txt which include the information for many glasses including their color, shape, brand name, and barcode.

### Output

The algorithm computes the best hashing function based on the position of the barcode digit and prints that hash function to the user so all the glass pairs could be sorted based on that hash function. The Algorithm returns a hash function which will put the items in the hashtables here (containers) in the best-balanced way (minimum disbalance in the heaviest bucket and lightest one)

## Project Methodology

### Steps

1. Reading the input file items per line
2. Adding the items to our hash table using the hash function
3. Handling any collision by using a chaining concept
4. Retrieving each hashtable balance by using the bucket size of each

### File Info Retrieval

Opening the .txt file and reading each line of code. There is a function add to the hash function which takes the input of each line and the barcode and adds it to the hash table by using hash functions

### Hash Functions

The hash function we used are simply mod of the corresponding digit for example if we hash based on the first digit we have the following:

Barcode: 1234567

If we use the first digit: the value is 1 ( $1 \% 10 = 1$ ) and is placed to the first slot of the table

This can be working for all seven tables and we can fill the slots of each table based on the value returned from that function.

### Adding Item to Hashtables

This is a function used to add the data read from the .txt file per line to the hashtable. It utilizes the hash function for each digit and saves the value in all seven hash tables.

### Removing Item From Hashtable:

Removing an item based on their barcode, we use a key-value idea to retrieve the item and remove it.

### BestHashing Method

In the best hashing method, we compare the balance of each table by using their `bucket_size()` function and determines the best hashtable with the best balance, in case there are two hashtables with equal balance. This function returns the hashtable with lower hash function value, for example, if the best are HT1 and HT5 it return HT1.

## Project Implementation

The project has been implemented in c++ upon the given skeleton from CPSC 535 course description. The project has the following methods and implemented (some parts are already implemented in the given project description)

### File Info Retrieval

The code below iterates through the .txt file and save the glasses pair information as below the first step is to open the file and if the file does not exist an exception with a customized message is displayed to the user and if the file is a valid file, the function gets all the items as below:

```
void ItemCollection::readTextfile(string filename) {
    // load information from the text file passed as argument
    ifstream myfile(filename);
    if (myfile.is_open()) {
        cout << "Successfully opened file " << filename << endl;
        string itemColor;
        string itemShape;
        string itemBrand;
        unsigned int barcode;
        while (myfile >> itemColor >> itemShape >> itemBrand >> barcode) {
            if (itemColor.size() > 0)
                addItem(itemColor, itemShape, itemBrand, barcode);
        }
        myfile.close();
    }
    else
        throw std::invalid_argument("Could not open file " + filename);
}
```

### Hash Functions

Seven hash function each one takes the corresponding digit of the barcode (the barcode is a 7-digit integer) and gets the % 10 value for placement in the hashtable as below:

```
unsigned int hash_function1(unsigned int barcode) {
    // function to return the hash value based on the first digit of a unique 7-digit key
    unsigned int d;
    d = barcode / 1000000;
    d = d % 10;
    return d;
}
```

```

unsigned int hash_function2(unsigned int barcode) {
    // function to return the hash value based on the second digit of a unique 7-digit key
    unsigned int d;
    d = barcode / 100000;
    d = d % 10;
    return d;
}

unsigned int hash_function3(unsigned int barcode) {
    // function to return the hash value based on the third digit of a unique 7-digit key
    unsigned int d;
    d = barcode / 10000;
    d = d % 10;
    return d;
}

unsigned int hash_function4(unsigned int barcode) {
    // function to return the hash value based on the fourth digit of a unique 7-digit key
    unsigned int d;
    d = barcode / 1000;
    d = d % 10;
    return d;
}

unsigned int hash_function5(unsigned int barcode) {
    // function to return the hash value based on the fifth digit of a unique 7-digit key
    unsigned int d;
    d = barcode / 100;
    d = d % 10;
    return d;
}

unsigned int hash_function6(unsigned int barcode) {
    // function to return the hash value based on the sixth digit of a unique 7-digit key
    unsigned int d;
    d = barcode / 10;
    d = d % 10;
    return d;
}

unsigned int hash_function7(unsigned int barcode) {
    // function to return the hash value based on the seventh digit of a unique 7-digit key
    unsigned int d;
    d = barcode;
    d = d % 10;
    return d;
}

```

## Adding Item to Hashtables

This function is used to add the items saved from the .txt file to the hashtables one to the seven based on the position of the barcode digit index

```

void ItemCollection::addItem(string itemColor, string itemShape, string itemBrand, unsigned int barcode) {
    // add specific glass and it's details to main display (i.e., to all hash tables)
    Item item(itemColor, itemShape, itemBrand, barcode);
    hash_table1[barcode] = item;
    hash_table2[barcode] = item;
    hash_table3[barcode] = item;
    hash_table4[barcode] = item;
    hash_table5[barcode] = item;
    hash_table6[barcode] = item;
    hash_table7[barcode] = item;
}

```

## Removing Item From Hashtable

This function removes any item by barcode as below:

```
bool ItemCollection::removeItem(unsigned int barcode) {
    /*
     * removes glass (and its details) specified by the barcode from the display
     * (i.e., from all hash tables). If barcode is found, return true, else return false
     */
    if (hash_table1.erase(barcode) != 1)
        return false;
    if (hash_table2.erase(barcode) != 1)
        return false;
    if (hash_table3.erase(barcode) != 1)
        return false;
    if (hash_table4.erase(barcode) != 1)
        return false;
    if (hash_table5.erase(barcode) != 1)
        return false;
    if (hash_table6.erase(barcode) != 1)
        return false;
    if (hash_table7.erase(barcode) != 1)
        return false;
    else
        return true;
}
```

## BestHashing Method

The best hashing function compares the hash tables balances and returns the best choice:

```
unsigned int ItemCollection::bestHashing() {
    /*
     * Balance is computed by subtracting the maximum element from the minimum element.
     * This process is done for every hash_function, and the one with the lowest balance is returned.
     */
    unsigned int min_loc, max_loc;
    unsigned int balance[8];
    unsigned int bucket[10];

    balance[0] = 0;

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table1.bucket_size(i);
    }
    min_loc = *std::min_element(bucket, bucket + 10);
    max_loc = *std::max_element(bucket, bucket + 10);
    balance[1] = max_loc - min_loc;

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table2.bucket_size(i);
    }
    min_loc = *std::min_element(bucket, bucket + 10);
    max_loc = *std::max_element(bucket, bucket + 10);
    balance[2] = max_loc - min_loc;

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table3.bucket_size(i);
    }
    min_loc = *std::min_element(bucket, bucket + 10);
    max_loc = *std::max_element(bucket, bucket + 10);
    balance[3] = max_loc - min_loc;
}
```

```

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table4.bucket_size(i);
    }
    min_loc =* std::min_element(bucket, bucket + 10);
    max_loc =* std::max_element(bucket, bucket + 10);
    balance[4] = max_loc - min_loc;

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table5.bucket_size(i);
    }
    min_loc =* std::min_element(bucket, bucket + 10);
    max_loc =* std::max_element(bucket, bucket + 10);
    balance[5] = max_loc - min_loc;

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table6.bucket_size(i);
    }
    min_loc =* std::min_element(bucket, bucket + 10);
    max_loc =* std::max_element(bucket, bucket + 10);
    balance[6] = max_loc - min_loc;

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table7.bucket_size(i);
    }
    min_loc =* std::min_element(bucket, bucket + 10);
    max_loc =* std::max_element(bucket, bucket + 10);
    balance[7] = max_loc - min_loc;

    for (unsigned int i = 0; i < 10; ++i) {
        bucket[i] = hash_table7.bucket_size(i);
    }
    min_loc =* std::min_element(bucket, bucket + 10);
    max_loc =* std::max_element(bucket, bucket + 10);
    balance[7] = max_loc - min_loc;

    unsigned int result = balance[1];
    result =* std::min_element(balance + 1, balance + 8);
    for (unsigned int pos = 1; pos <= 7; pos++) {
        if (result == balance[pos]) {
            min_loc = pos;
            break;
        }
    }
    return min_loc;
}

```

## How to Run the code

In order to run the code on the Mac or Linux environment use the following command after opening the terminal and going to the project directory:

***G++ -o output ItemCollection.cpp main.cpp***

Then run

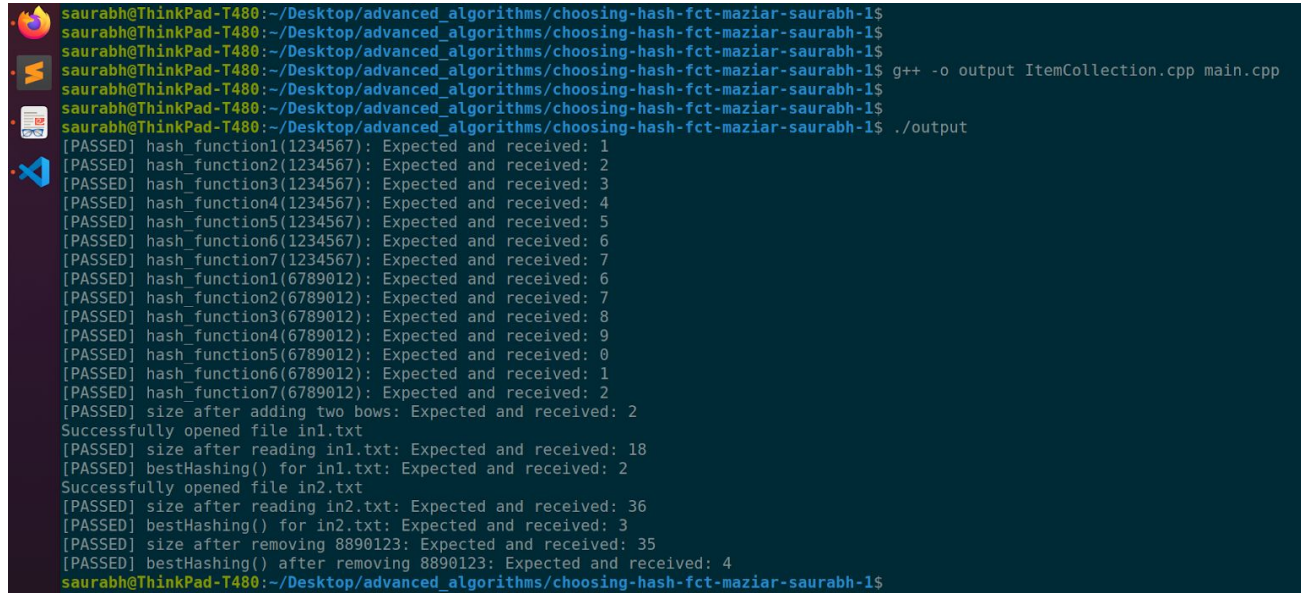
***./ output***

(note: the code iterate In1.txt for other txt files please hardcode the .txt file name)



## Output Screenshots

Below is the screenshot of the output with successfully passing all the tests:

A terminal window with a dark blue background and light blue text. The prompt is 'saurabh@ThinkPad-T480:~/Desktop/advanced\_algorithms/choosing-hash-fct-maziar-saurabh-1\$'. The user enters 'g++ -o output ItemCollection.cpp main.cpp' and then './output'. The output shows a series of 14 test cases, all passing. The tests include hash functions for two different inputs (1234567 and 6789012), file operations (opening in1.txt and in2.txt), and bestHashing() calls. The final prompt is 'saurabh@ThinkPad-T480:~/Desktop/advanced\_algorithms/choosing-hash-fct-maziar-saurabh-1\$'.

```
saurabh@ThinkPad-T480:~/Desktop/advanced_algorithms/choosing-hash-fct-maziar-saurabh-1$
saurabh@ThinkPad-T480:~/Desktop/advanced_algorithms/choosing-hash-fct-maziar-saurabh-1$
saurabh@ThinkPad-T480:~/Desktop/advanced_algorithms/choosing-hash-fct-maziar-saurabh-1$
saurabh@ThinkPad-T480:~/Desktop/advanced_algorithms/choosing-hash-fct-maziar-saurabh-1$ g++ -o output ItemCollection.cpp main.cpp
saurabh@ThinkPad-T480:~/Desktop/advanced_algorithms/choosing-hash-fct-maziar-saurabh-1$
saurabh@ThinkPad-T480:~/Desktop/advanced_algorithms/choosing-hash-fct-maziar-saurabh-1$ ./output
[PASSED] hash_function1(1234567): Expected and received: 1
[PASSED] hash_function2(1234567): Expected and received: 2
[PASSED] hash_function3(1234567): Expected and received: 3
[PASSED] hash_function4(1234567): Expected and received: 4
[PASSED] hash_function5(1234567): Expected and received: 5
[PASSED] hash_function6(1234567): Expected and received: 6
[PASSED] hash_function7(1234567): Expected and received: 7
[PASSED] hash_function1(6789012): Expected and received: 6
[PASSED] hash_function2(6789012): Expected and received: 7
[PASSED] hash_function3(6789012): Expected and received: 8
[PASSED] hash_function4(6789012): Expected and received: 9
[PASSED] hash_function5(6789012): Expected and received: 0
[PASSED] hash_function6(6789012): Expected and received: 1
[PASSED] hash_function7(6789012): Expected and received: 2
[PASSED] size after adding two bows: Expected and received: 2
Successfully opened file in1.txt
[PASSED] size after reading in1.txt: Expected and received: 18
[PASSED] bestHashing() for in1.txt: Expected and received: 2
Successfully opened file in2.txt
[PASSED] size after reading in2.txt: Expected and received: 36
[PASSED] bestHashing() for in2.txt: Expected and received: 3
[PASSED] size after removing 8890123: Expected and received: 35
[PASSED] bestHashing() after removing 8890123: Expected and received: 4
saurabh@ThinkPad-T480:~/Desktop/advanced_algorithms/choosing-hash-fct-maziar-saurabh-1$
```