CPSC 535 Advanced Algorithms
Project 2: Choosing the Hash Function

Prof. Doina Bein, CSU Fullerton

dbein@fullerton.edu

Introduction

In this project you will design and implement one algorithm related to hash tables. You will describe the algorithm using clear pseudocode and implement it using C/C++/C#/Java/Python, compile, test it, and submit BOTH the report (as a PDF file) and the files. Your project is about reading a large number of distinct product numbers (7-digit each) and deciding which digit among the seven gives the best balanced storage of the items. Project is due Friday, May 15, 11:59 pm on GitHub.

Choosing the hash function

At an optometrist store, glasses of various sizes and shapes are displayed in a glass display that has 10 cubbies or sections. Each pair of glasses is unique and has a 7-digit barcode. You have too many pairs of glasses to store them in the same cubby, plus only few would be visible. You are tasked with organizing the pairs of glasses into cubbies such that most of them are visible. You came up with an idea of storing the pairs of glasses based on a single digit of their product number. So you try to group them based on the first digit, second digit, .., seventh (and last digit). (The digits are read from left to right; e.g. for the number 8976565, the first digit is 8, the second is 9, the third is 7, the fourth is 6, the fifth is 5, the sixth is 6 and the seventh and last is 5.) You settle on the digit that gives the best "balancing" of storing the pairs of glasses, i.e., the difference between the number of pairs of glasses in the most populated cubby and the least populated cubby is minimized among all possible five options.

Your project is about reading a large number of distinct product numbers (7-digit each) and deciding which digit among the seven gives the best balanced storage of the pairs of glasses.

For example, let's assume that we have the following product numbers:

1234567, 2345678, 3456789, 4567890, 5678901, 6789012, 7890123, 8901234, 9012345, 5432109, 6543210, 7654321, 8765432, 9876543, 1098765, 2109876, 3210987, 4321098

If we choose storing the bows based on the first digit that we have:

Cubby 0: no pair of glasses

Cubby 1: 1234567, 1098765

Cubby 2: 2345678, 2109876

….

Cubby 9: 9012345, 9876543

The difference in the cubbies' load is 2.

If we choose storing based on the second digit:

Cubby 0: 9012345, 1098765

Cubby 1: 2109876

Cubby 2: 1234567

…

Cubby 9: 8901234,

The difference in the cubbies' load is 1 so clearly this is a more balanced organization of the pair of glasses.

The Skeleton Code

You are provided with the following files.

1. README.md contains a brief description of the project, and a place to write the names and CSUF email addresses of the group members. You need to modify this file to identify your group members.
2. LICENSE contains a description of the MIT license.
3. main.cpp contains tests to check on the correctness of the function members
4. ItemCollection.cpp and ItemCollection.hpp contain the skeleton of the classes and function members
5. in1.txt contains 18 types of glasses
6. in2.txt contains 36 types of glasses

The code to decide which digit leads to the most balanced hashtable is to be implemented in class ItemCollection. Since you will be comparing seven hashtables (which differ only in their hash function), class ItemCollection only has seven hashtables as its member variables. The other member functions are:

- addItem(): Given information about one pair of glasses, create an Item object and insert into each of the seven hashtables. Note that each hashtable has the product number as the key and a Item object as the value. To be completed.
- removeItem(): Given product number, remove the corresponding pair of glasses from each of the seven hashtables. To be completed.
- bestHashing(): The logic to calculate the balance for each of the seven hashtables, and then identifying the hashtable with the best balance should go into this method. Here, balance is defined as the difference between the sizes of the largest bucket and smallest bucket. Only check the first 10 buckets! (If the lowest balance factor is shared by more than one hash table, then return the first hash table with that lowest balance factor. For example, if both hT5 and hT7 have the lowest balance factor, then return the number 3. If hT2, hT4, and ht6 all share the lowest balance factor, then return the number 2.) Some hints on how to get the number of items in each bucket are included. To be completed.
- readTextfile(): The list of pairs of glasses are in a text file. This method calls addItem() for each line. The code to read from the text file is already given.

The seven hashtables will differ in only the hash function that they will use. You are to provide code for these hash functions. Each hash function will take a 7-digit number and return either the first, second, …, seventh (last) digit.

- hashfct1(number): return the first digit of number. To be completed.
- hashfct2(number): return the second digit of number. To be completed.
- hashfct3(number): return the third digit of number. To be completed.
- hashfct4(number): return the fourth digit of number. To be completed.
- hashfct5(number): return the fifth digit of number. To be completed.
- hashfct6(number): return the fourth digit of number. To be completed.
- hashfct7(number): return the fifth digit of number. To be completed.
- main.cpp: This is provided for you to use to test your software as you are writing it. You may change this file to add helpful functions for your own testing. I will test your project with a different but similar file.
- README.md: You must edit this file to include your name and CSUF email. This information will be used so that we can enter your grades into Titanium. To be completed.

What Output is expected

The output should display the result of passing all the tests included in the test file main.cpp.

Grading rubric

The total grade is 35 points. Your grade will be comprised of three parts, Form, Function, and Report:

- Function refers to whether your code works properly and passes all the tests (21 points, 1 point per test).
- Form refers to the design, organization, and presentation of your code. The instructor will read your code and evaluate these aspects of your submission (6 points):
    - README.md completed clearly = 2 points
    - Style (whitespace, variable names, comments, helper functions, etc.) = 2 points
    - C++ Craftsmanship (appropriate handling of encapsulation, memory management, avoids gross inefficiency and taboo coding practices, etc.) = 2 points
- Report (8 points) divided as follows:
    - Summary of report document (2 points)
    - Pseudocode of overall algorithm and description of all 7 hash functions (4 points)
    - Two screenshots: one for the group members and one of the test that includes the two sample input files (1 point each, total 2 points)

Obtaining and Submitting Code

This document explains how to obtain and submit your work:

GitHub Education / Tuffix Instructions

Here is the invitation link for this project:

https://classroom.github.com/g/WkvCj1UX

What to Do

First, add your group members' names to README.md. Then write a clear pseudocode for the algorithm, describe each of the seven hash functions and submit it as a PDF report. Your report should include the following:

1. Your names, CSUF-supplied email address(es), and an indication that the submission is for project 1.
2. A full-screen screenshot with your group member names shown clearly. One way to make your names appear in Atom is to simply open your README.md.
3. The pseudocode for the algorithm and description of all 7 hash functions.
4. A brief description on how to run the code, if needed
5. One snapshot of code executing the main.cpp file that includes also the two given input files.

Then implement your algorithm in C/C++/C#/Java/Python. Submit your PDF by committing it to your GitHub repository along with your code.