ECE250: Lab Project 2

Due Date: February 14th, 2020, 11:00 pm

1. Project Description

In this project, you implement a solution for the example in the lecture on hash table. R&T is a large phone company that provides caller ID capabilities. Given a phone number, your program will help R&T Company by returning the caller name associated to that number.

Your solution will be a C++ implementation for a hash table data structure. In this data structure, values are mapped to a position in a table using a hash function. For this project, you will implement a hash table in which collisions resolve using two different techniques: (i) open addressing using double hashing and (ii) separate chaining where the chains are ordered. Both primary and secondary hash functions, provided in the next section, use the Division Method.

We ask that you write a C++ class for each of these collision techniques. Each class should provide several services to: (i) initialize/create a hash table (ii) insert a value into a hash table, (iii) search for a value in a hash table, and (iv) delete a value from the hash table.

You may use the vector class of the STL C++ library to store the hash table. For the hash table handling collision using chaining technique, you have to implement your own linked list.

2. Hash functions

Primary hash function
$h_1(k) = k \mod m$, in which k is the key, and m is the size of the hash table.

Secondary hash function
For open addressing, you will implement the hash table using the double hashing technique to resolve collisions. The secondary hash function is $h_2(k) = \lfloor k/m \rfloor \mod m$. Since $h_2(k)$ must be an odd number, you will add 1 to the resulting value if this value is even.

3. Program Design

Write a short description of your design. You will submit this document along with your C++ solution files for marking. This document must include your design decisions. Please refer to the course website for “Programming Guidelines” and the expected content for your design document.

4. Input and Output Requirements

Write a test program for each technique handling collision: (i) **openhttest.cpp** for open addressing using double hashing and (ii) **orderedhttest.cpp** for chaining. The test programs will read commands from standard input and write the output to standard output. These programs will respond to the commands described in this section.

The phone numbers (keys), as input, are limited to North American phone numbers (ten digits), with the following format: *xxxyyyyyyy*, where *xxx* is the area code and *yyyyyyy* is the local number. For example: 5198884567.

Command	**Parameters**	**Description**	**Output**
n	*m*	Defines size of the hash table	**success**
i	*k;caller*	Inserts the key *k* and the associated *caller*	**success:** if the insertion was successful **failure:** if the insertion was unable to complete since the table was full or the key was already there
s	*k*	Searches for the key *k* in the table	**found** *caller* **in p:** if the desired key was found in the position *p* of the hash table **not found**: if the desired key was not found
d	*k*	Deletes the key *k* from the table	**success:** if the deletion was successful **failure:** if the deletion was unable to complete since the value was not found in the table
p	*i*	This command is only for *separate chaining* Prints the chain of keys that starts at position *i;* Keys in the chain are separated by one space If chain is empty, the command does not print anything	**Example:** For m=10 5197225111 5197225131 6473621111

Assuming uniform hashing, the expected average runtime for each insert (*i*), search (*s*), and delete (*d*) operation is constant. In your design document, you should also describe how you have achieved this in your implementation.

- **Test Files**

The course website contains example input files for each technique handling collisions with the corresponding output files. The files are named *test01.in*, *test02.in* and so on with the output files named *test01-chain.out*, *test01-open.out*, *test02-chain.out*, *test02-open.out*, and so on.

5. How to Submit Your Program

Once you have completed your solution and tested it comprehensively in your computer or on the lab computers, you have to transfer your files to the *eceUbuntu server* and test there since we perform the automated testing using this environment.

Once you finish testing in the *eceUbuntu server*, you will create a compressed file (tar.gz) that should contain:

- A typed document (maximum two pages) describing your design. A document beyond 2 pages will not be marked. Submit this document in PDF format. The name of this file should be:

  xxxxxxxx _design_ p***n***.pdf

  in which ***xxxxxxxx*** is your UW user id (e.g., jsmith) and ***n*** is the project number that is 2 (two) for this submission.
- Two tests program, one for each techniques handling collision (**openhttest.cpp** and **orderedhttest.cpp***)*
- Required header files and classes (ending in *.h .cpp*)
- A make file (named Makefile), with instructions on how to compile your solutions and create two executable files named **openhtdriver** and **orderedhtdriver**

The name of your compressed file should be ***xxxxxxxx***_p***n***.tar.gz, where ***xxxxxxxx*** is your UW user id (e.g., jsmith) and ***n*** is the project number that is 2 (two) for this submission.