Middle East Technical University | Department of Computer Engineering

CENG 213

Data Structures

Spring 2018-2019

Homework 3

Due date: 09.05.2019, Thursday, 23:55

1 Introduction

In this assignment you are going to practice on hash table implementation for an employee information system of a global company, along with its accompanying hash function using C++. Remember that insertion, deletion, and retrieval operations are to run in expected constant time for hash tables.

Keywords: *Hash table, Separate chaining, C++*

2 Problem Definition

In this homework you will implement a hash table for an employee information system of a global company. The application must store the ssn(social security number) of each employee in the hash table. Note that, each employee has a unique ssn. Your application need to provide the following requirements:

- Inserting an employee into the hash table
- Deleting an employee from the hash table
- Finding employee with the given ssn
- Finding the number of employees in the given bucket

3 Specifications

- You will be implementing a hash table class, named as EmployeeTable, all by yourself. The bare header file, "EmployeeTable.h", is given below. You are free to add any private variables/functions to it. However, your hash table should support at least the given functions.

```
class ItemNotFoundException : public exception{
  public:
      const char * what() const throw(){
          return "Item Not Found in the Hash Table!";
      }
};
```

```
class EmployeeTable{

private:
// ... members, methods

public:
// Constructors & Destructor, be careful about memory leaks.
    EmployeeTable(); // Default table size is 151 buckets.
    EmployeeTable(int numberOfBuckets); // numberOfBuckets is a prime number.
    EmployeeTable(const EmployeeTable& empTable);
    ~EmployeeTable();
    EmployeeTable& operator=(const EmployeeTable& rhs);

// Hash an employee's ssn to a non-negative integer ( return hash value, not
// bucket number). Use hashValue(mod numberOfBuckets) as bucket number.
// Return -1 on invalid ssn.
  int hashEmployee(string ssn);

// Put an employee to the hash table.
// Do nothing on invalid parameters.
  void addEntry(Employee& employee);

// Remove the entry from the hash table.
// Do nothing on invalid parameters.
  void removeEntry(Employee& employee);

// Get the total number of employees in the given bucket.
// Return -1 on invalid bucket number.
  int getNumberOfEmployeeInBucket(int bucket);

// Find employee with the given ssn.
// Important: To add an extra flavor to your homework, your code
// will be evaluated by how much time it takes to access an employee
// on the average. You will get a small portion of your grade
// depending on the average access time of your code.
// For invalid cases, throw ItemNotFoundException.
  Employee findEmployee(string ssn);
};
```

- An employee's attributes will be represented with the Employee class. "Employee.h" is given below. It should not be modified in any case.

```
class Employee{

private:
  string name; // name of the employee. Invalid: ""
  int experience; // experience of the employee in terms of months. Invalid: -1
  string city; // current working place of the employee. Invalid: ""
  string ssn; // ssn of the employee. Invalid: ""

public:
// Constructors
  Employee();
  Employee(string name, int experience, string city, string ssn);

// Getters and setters
    string getName();
    void setName(string name);
    int getExperience();
    void setExperience(int experience);
    string getCity();
    void setCity(string city);
    string getSsn();
    void setSsn(string ssn);

// checks whether given two Employees are same or not.
    bool compare(Employee employee);
};
```

- You should try to find a hash function that distributes the employees as uniformly as possible to the hash table (hash table's buckets). The ultimate goal - which may not possible - is to create a hashing that aims at most 1 employee per bucket. You will be graded on how close you are to this uniformity. You are strongly encouraged to use "ssn" when designing your hash function.
- You will implement "EmployeeTable.cpp", and will use **separate chaining** as collision resolution strategy. Employees will be "chained" to co-exist in the same bucket.
- Note that, employee with same name, experience or city may exist in test cases. However, you can assume that ssn of each employee will be unique.
- Sample datasets will be given to you, each including name, experience and city for every employee, as well as their ssn. Test datasets will resemble the given datasets, however, they will also include new employees. So, you simply can not design a perfect hash function that assumes all keys and values are known a priori. During testing, after inserting all values, we'll request a number of employees from the hash table.
- If you have a high number of collisions, the time it takes for your code to fetch an employee increases. Try to reduce the number of collisions as much as possible for the given datasets. Your responsibility in this context includes providing the correct number of employees in a bucket, and of course, returning them correctly.
- You will analyze the success of your implementation by evaluating it for the given datasets, and submit a report including the analysis. Please read "Report" section for details.
- Dynamic resizing/rehashing is not allowed. Your work will be inspected to ensure fairness.
- 80% of your grade will be based on the correctness of your solution, while the remaining 20% will be related to the report and performance of your code.

4 Dataset

- You will be provided a number of datasets (Company1.txt - ... - CompanyM.txt), each file representing a new employee table used for several companies. The file format is given below:

```
167 // Size of employee table.
--Employees--
Alice // Name of first entry (employee).
30 // Experience of first entry.
New York // City of first entry.
0905552321 // SSN of first entry.
...
...
Belle // Name of Nth entry.
35 // Experience of Nth entry.
Oslo // City of Nth entry.
0119120121 // SSN of Nth entry.
--SSNs--
00048203525 // SSN 1.
00798448113 // SSN 2.
...
...
06498095975 // SSN K.
```

- You will test your code yourselves by reading the input and performing the required operations in your main function. For each entry in "Employees" list, you will call "AddEntry(...)" function after forming the "Employee" object with required parameters. Similarly, each entry in "SSNs" list requires a call of "FindEmployee(...)" function. Please test all of your functions to verify that they work correctly. The time performance test will only take "FindEmployee()" calls into account.

5 Report

- You should create a new hash table with the specified size in each dataset, and hash all entries in "Employees" list. Then, you will make the specified calls. Your report should include your table's performance on each dataset.
- The report you will submit should have the following format:

1. **Time Performance:** Report average access time for all sample datasets (Only take "FindEmployee()" calls into account). To measure time, form a loop and do all access calls successively. Make comments on why certain datasets's performance are different than others. If you have made any optimizations in your code, you can mention them here. To measure time, you can use "clock()" function in "time.h".
2. **Hash Function Quality:** Count average number of collisions for all given datasets in "FindEmployee()" calls. Explain the nature of the data, and how you improved your hash function.
3. **Bucket Statistics:** List average number of entries in non-empty buckets (along with total number of non-empty buckets) for each dataset. Compare them with the ideal values. Comment on the results.

6 Regulations

1. You will use C++.
2. If your submission fails to follow the specifications or does not compile, there will be a "significant" penalty of grade reduction.
3. **Late Submission:** Late submission policy is stated in the course syllabus.
4. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations. Remember that students of this course are bounded to code of honor and its violation is subject to severe punishment.
5. **Newsgroup:** You must follow the Forum (in Odtüclass) for discussions and possible updates on a daily basis.

7 Submission

1. Submission will be done via Moodle (cengclass.ceng.metu.edu.tr).
2. Do not write a main function in any of your source files.
3. A test environment will be ready in Moodle.
   - You can submit your source files to Moodle and test your work with a subset of evaluation inputs and outputs.
   - Additional test cases will be used for evaluation of your final grade. So, your actual grades may be different than the ones you get in Moodle.
   - Only the last submission before the deadline will be graded.