Middle East Technical University | Department of Computer Engineering

CENG 213

Data Structures

Fall '2018-2019

Homework 3

Due date: 29 December 2018, Saturday, 23:55

1 Introduction

This assignment aims to get you familiar with hashing.

Keywords: Hashing, Double Hashing, Open Addressing, Templates

2 Problem Definition

In this homework you will implement a simple access control system, where you can add/delete users, change passwords of the users, and log users in and out, list all registered and active users. Registered users are the ones added with adduser function, and active users are the ones that are logged in. A user must be registered to be added to the active users list with the login function.
You must use open addressing with double hashing as your method to implement a hash table. Operations that your program should support are inserting to hash table, and removing and getting data from hash table. You will have two hash tables. Table1 maps string to std::pair<string, string>, and should contain the usernames and passwords of the registered users. Here old passwords and new password of the users will be present. table2 maps string to string, and it should contain usernames of the active(logged in) users.
For example, if 4 users are registered and 2 of them are active(logged in), tables might look like the following. Order of the items depend on the hash functions:

Table 1

- <"John", password of John >
- <"Alice", password of Alice >
- <"Bob" , password of Bob >
- <"John", old password of John >
- <"Eve", password of Eve >
- <"Bob" , old password of Bob >
- <"John", another old password of John >

Table 2

- Alice
- John

3 Implementation

You are given a header file(hw3.h) which includes a class definition of AccessControl, and you must create a "hw3.cpp" file and complete the class. You can add private methods and variables, but dont modify the public methods and variables. You must use hash tables. You can create other classes. You will have two hash tables in this homework, one is to hold <username, password> pairs(registered users), and one is to hold usernames(active users). Note that, usernames will contain only alphanumeric values.

Upon insertion(addUser, addUsers, changePass, login), if the load factor of your table is greater than MAX_LOAD_FACTOR, you should expand the table. New table size becomes the smallest prime number greater than $2 * (tablesize)$. Don't rehash the "DELETED" and "EMPTY" entries.

The hash function is given in header file. You can modify the arguments of this function, or move it to another class as long as the algorithm does not change. For each query variable, "i" starts from 0. Each time a collision occurs you should increment "i" and call this hash function again to calculate new index.

If you cant find an empty slot after a number of probes that is equal to the table size for a new entry, do not insert that entry.

While expanding and rehashing, you should do the following:
Traverse the old table from top to bottom, for each entry, if it is not empty or deleted, add ALL passwords of that user to the new table starting with the oldest one. Doing so you will preserve the order of passwords, so that you will guarantee that the index of an older password will be calculated with the hash function with a smaller i value than the value i that will be used to calculate the index of a newer password. For example, for the table below:
hash("Sarah", i1) = index of <"Sarah", sarahsoldpass >
hash("Sarah", i2) = index of <"Sarah", sarahsnewpass >
i1 is always smaller than i2, but we can not guarantee anything about the order of indexes.

——————Old Table——————

- <"Chuck", chuckspass >
- <"Sarah", sarahsnewpass >
- <"Morgan", morgansoldpass >
- <"Sarah", sarahsoldpass >
- <"Morgan", morgansnewpass >
- ...

———————————————————

The order of inserting to new table from the old table should be: Chuck chuckspass, Sarah sarahsoldpass, Sarah sarahsnewpass, Morgan morgansoldpass, Morgan morgansnewpass ...

3.1 AccessControl Class

- AccessControl(int table1Size, int table2Size)
  Initialize tables. Set entries of table1 to <"EMPTY", "EMPTY">. Set entries of table2 to "EMPTY".

- int addUser(string username, string pass)
  Register the given user with username and password to registered users table if not registered. Return 1 if successful, 0 if not.

- int addUsers(string filePath)
  Given a file, register all users in file to the registered users if not already registered. return the number of users registered from file. File will look like this:

  Alice pass123
  Bob P4ssw0rd
  Eve Suchalongpassword

- int delUser(string username, vector<string>& oldPasswords)
  Remove a user from registered users and put all passwords of given user to oldPasswords variable. Return 1 if the user is registered else just return 0. You should delete all entries of that user. Put <"DELETED", "DELETED"> as deleted entry. The vector, oldPasswords, should be sorted from oldest password to the newest password.

- int changePass(username, oldpass, newpass)
  Change the password of the given user if user is registered and oldpass is equal the current password. While changing the password keep the old entries <username, oldpassword> in the table, and add new entry <username, newpass> to the next empty cell that is found with the hash function. Last entry added is the current password of that user. If operation is successful return 1, else return 0.

- int login(username, pass)
  If the user is registered and pass is the current password of given user log the user in and return 1, if not do nothing and return 0.

- int logout(username)
  If the given user is logged in log that user out and return 1, if not return 0. Put "DELETED" for logged out users.

- float printActiveUsers()
  Print all logged in users. Print "DELETED" for deleted entries, "EMPTY" for empty entries. Returns load factor.
  Example:
  Alice
  DELETED
  Bob
  EMPTY
  Eve

- float printPasswords()
  Print all users and passwords. print "DELETED DELETED" for deleted entries. Print "EMPTY EMPTY" for empty entries. Returns load factor.
  Example:
  Alice pass123

DELETED DELETED
Eve Anoldpassword
Bob P4ssw0rd
EMPTY EMPTY
Eve Suchalongpassword
EMPTY EMPTY
EMPTY EMPTY

4 Regulations

1. You will use C++.
2. You may want to use templates for your hash implementation.
3. You are not allowed to use any libraries/includes except for the ones included in header file.
4. You must use hash tables as specified. If your submission fails to follow the specifications or does not compile, there will be a "significant" penalty of grade reduction.
5. **Evaluation** will be glass-box. Implementation details may cause a penalty, or may be awarded.
6. **Late Submission:** Refer to the syllabus for late submission policy.
7. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
8. **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.

5 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.