T-201-GSKI, GAGNASKIPAN

Vor 2017

CONTACT LIST using HASH TABLES

Assignment grading. For full marks your solution needs to be accepted by Mooshak. Points may be deducted for

- not implementing the solution according to the specifications,
- redundant or repeated code.
- messy code,
- code that is difficult to understand and not explained in comments.

If your solution is not accepted by Mooshak, it will receive a maximum grade of 7.

INTRODUCTION

In this project, we will reiterate on the previous project (V6) and create a contact manager.
But instead of using a binary search tree to implement *StringContactMap* we will now use a hash table.
We make use of the fact that all our classes define abstract data types, so changing the implementation of *StringContactMap* does not require changing other sections of the program.
Note that completing V6 is not a preliminary for the current assignment and that you can use your own implementation of ContactManager, or use the one supplied.

1: CONTACT LIST (35%)

We will use open hashing (seperate chaining) with linked lists to resolve hash collisions in our hash map.
A preliminary step before we re-implement *StringContactMap* is then to implement a linked list of contacts, *ContactList*.
ContactList is a singly linked list of (key, value) pairs, more specifically of *StringContactPair* objects, with a string as a key and a Contact object as value. Each pair is stored in a node, *ContactNode*.
StringContactPair is defined in *StringContactPair.h*.

Implementation. The supplied interface of *ContactList* gives the implementation specifications. You are supplied with a template for the implementation in *ContactList.cpp*.

Note that you do ***not*** need to implement the function *get_contacts_by_prefix* in this part of the assignment.

Submitting. To submit your solution to Mooshak you should create a zip file containing at least the following

- Contact.cpp,
- Contact.h,
- ContactList.cpp,
- ContactList.h,
- ContactNode.cpp,
- ContactNode.h,
- KeyException.h,
- StringContactPair.cpp,
- StringContactPair.h.

The public interface of *StringContactMap* is the same as in the previous assignment—although their implementation will be different. In this section, you will implement *StringContactMap* using a hash map as specified in

the following section.

Implementation. As previously stated, *StringContactMap* will use seperate chaining with linked lists to resolve hash collisions. Use the variable ListPtr* map as a dynamic array of such lists.

You should implement the hash function used by the hash map in the file Hash.cpp. You are free to use any hash function you like, e.g., ***djb2*** or ***MurmurHash2***.

Besides the specification provided in the supplied interface of *StringContactMap*, there is one thing of note.

When adding a key-value pair, a check must be performed to test whether the length of each list in the hash map is on average of reasonable length. The test is performed according to the formula:

```
(count / capacity) < MAX_LOAD
```

2: STRING CONTACT MAP (45%)

Where count is the number of entries in the map, and capacity the current number of buckets (the size of the dynamic array). If the test fails, the number of items in the hash map is considered to exceed the acceptable amount, *MAX_LOAD*, and we must rebuild the hash map to facilitate the addition of more entries. You rebuild the hash map by creating a new hash map containing double the number of buckets as the current hash map, and adding all the values of the old hash map to the new hash map. The test should be performed in the method load_check and the rebuilding should be implemented in the method rebuild. You are free (and encouraged) to tweak the constants *MAX_LOAD* and *INITIAL_CAPACITY* to improve the performance of your hash map.

You are supplied with a template for the implementation in *StringContactMap.cpp*. Note that you do not need to implement the function prefix_search in the current part of the assignment.

Submitting. To submit your solution to Mooshak you should create a zip file containing every C++ file supplied with the project, modified as specified.

2B: CONTACT MANAGER

Because ContactManager does not care about the implementation of StringContactMap you do not need to re-implement ContactManager. Whoo-hoo!

3: PREFIX SEARCH (20%)

Implement the function `prefix_search(string prefix)` in *StringContactMap*.

Implementation. Your implementation of the function should use

`get_contacts_by_prefix (string prefix , vector<Contact>& contacts )`

from *ContactList* as a helper function. Further details about this function are provided in the interface.

Submitting. To submit your solution to Mooshak you should create a zip file containing every C++ file supplied with the project, modified as specified.