CS 130: Algorithms and Data Structures I
Fall 2019
Professor Reeves
Assignment 3: MyTunes Artist Search
Released: 10/31/2019
Due: 11/10/2019, 11:59 PM

1 Introduction

After previously building the queue for the MyTunes music service, you have decided that you want to build an artist search feature. MyTunes' users will have the ability to search for their favorite artist and get the list of songs for the artist which are accessible on the platform. After learning about hash tables in CS 130, you are attracted to the efficiency of insertions, deletions, and search using this data structure. You decide that this would be a good data structure for implementing the MyTunes artist search feature.

2 Instructions

Given the HASHTABLE_OA data structure below:

```
enum STATUS { EMPTY_SINCE_START, EMPTY_AFTER_REMOVAL, OCCUPIED };
struct BUCKET {
    ARTIST *art;
    STATUS st;
};
typedef BUCKET *HASHTABLE_OA;
```

artist records will be stored in this hash table based on hashing of the artist's name. The artist record is defined by the data structure below:

```
struct ARTIST {
    std::string name;
    std::set<std::string> songs;
};
```

Using this representation of an artist, every song of the artist available on MyTunes will be stored in a STL *set* which keeps the songs in sorted order for easy display to the MyTune's users. The STL *set* is implemented using red-black trees (fyi).

You would like to use the space for the hash table efficiently and avoid clustering of elements, so you choose to implement your hash table using an open addressing approach. In particular, you find double hashing to be an attractive approach to collision resolution. These components will serve as the foundation of your artist search implementation.

You are responsible for implementing the following functions for using this hash table:

Initialize - This function takes two parameters: *table* and *table_size*. This function will initialize the *HASHTABLE_& table* so that each *BUCKET** in the table has memory allocated for it. The *ARTIST* art* of each *BUCKET* should be initialized to a *NULL* pointer. The *STATUS st* of each BUCKET should be initialized to *EMPTY_SINCE_START*.

Clear - This function takes two parameters: *table* and *table_size*. This function will delete the memory allocated for each *BUCKET** in the table. If the *ARTIST* art* of the *BUCKET* is not *NULL*, the memory for *art* should be deleted as well.

NextBucket - Two hash functions (*HashFunc1* and *HashFunc2* that take a *string* as a parameter and return an ***unsigned long***) are provided for you to use. These hash functions will be used to calculate the *BUCKET* to check in the functions below (*Insert*, *Remove*, and *Search*) using double hashing as described in lecture and your zyBook.. You should use the hash value of the *ARTIST**'s *name* returned by *HashFunc1* and *HashFunc2* as the parameter values passed to the *NextBucket* function for *hash_val1* and *hash_val2*, respectively. As a reminder, the formula for calculating a bucket using double hashing is

$h_1(x) + i * h_2(x) \bmod N$

where

x - the key to hash
$h_1(x)$ - the **unsigned long** hash value of key x produced by hash function h_1
i - the current probing sequence value
$h_2(x)$ - the **unsigned long** hash value of key x produced by hash function h_2
mod - the modulus operator
N - the size of the hash table

Insert - This function takes three parameters: a *HASHTABLE_OA*& table*, an *int& table_size*, and an *ARTIST* art*. When this function is executed *art* is inserted into *table* at the first non-*OCCUPIED* bucket encountered using the double hashing algorithm. The *key* to hash for inserting *art* should be the *name* of *art*. If an *ARTIST** already exists in the *table*, *art* should replace the *ARTIST** currently stored in *table*. When inserting an *ARTIST** into the hash table, the *STATUS* of the *BUCKET* in which it is entered should be updated to be be *OCCUPIED* and the function will return *true*.

For Extra Credit: If *table* is full after inserting this *ARTIST**, you double the capacity of the *table*, re-hash each *key*, and insert the *ARTIST** in the proper bucket in the newly allocated hash table. The *table_size* value should be updated to reflect that the size of the hash table has increased. This is why *table_size* is a *int*& because we are able to change this value within the function.

Remove - This function takes three parameters: a *HASHTABLE_OA**& *table*, an *int table_size*, and a *std::string key.* This function will search *table* for an *ARTIST** stored in *table* which has a *name* equal to *key.* The function will use double hashing to search the *table* for an *ARTIST** with a name equal to key until one of 3 conditions are met:

1) An *ARTIST** with *name* equal to *key* is found. In this case, the memory for the *ARTIST** is deleted, the *ARTIST* art* data member of the *BUCKET* is set to *NULL* and the *BUCKET*'s *st* data member is set to *EMPTY_AFTER_REMOVAL*. The function should return *true* in this case.

2) A *BUCKET* is reached in the search which has an *st* value of *EMPTY_SINCE_START*. The function should return *false* in this case.

3) All *BUCKET*s have been checked without finding an *ARTIST** having *name* equal to *key.* The function should return *false* in this case.

Search - This function takes three parameters: a *HASHTABLE_OA**& *table*, an *int table_size*, and a *std::string key.* This function will search *table* for an *ARTIST** stored in *table* which has a *name* equal to *key.* The function will use the double hashing technique to search the *table* for an *ARTIST** with a name equal to key until one of 3 conditions are met:

1) An *ARTIST** with *name* equal to *key* is found. The function should return the matching *ARTIST** in this case.

2) A *BUCKET* is reached in the search which has an *st* value of *EMPTY_SINCE_START*. The function should return *NULL* in this case.

3) All *BUCKET*s have been checked without finding an *ARTIST** having *name* equal to *key.* The function should return *NULL* in this case.

3 Files

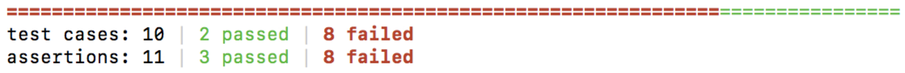
You are being provided with a set of files for this assignment that are available on Blackboard. The files are located in a ZIP file named *my_tunes_search.zip*. The directory structure provided when you uncompress *my_tunes_search.zip* is as follows:

```
my_tunes_search
├── src
│   ├── artist.hpp
│   ├── artist.cpp
│   └── Makefile
├── writeup
│   └── search_instructions.pdf
└── test
    ├── artist_song.txt
    ├── catch.hpp
    ├── tests.cpp
    └── Makefile
```

Starter files for your implementation have been provided for you. These files are *artist.hpp* and *artist.cpp*. Implement your solution in *artist.cpp*. The *test* directory stores files for testing that your code works as expected. You should make sure that all tests are passed before you submit your assignment. More details on testing are located in the *Testing* section of this document.

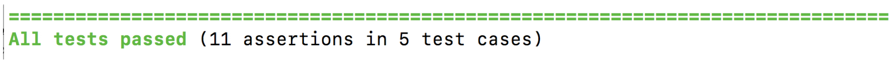
4 Testing

Your implementation is being tested using a unit-testing framework called Catch2. The functionality of this library is provided in the file *test/catch.hpp*. Feel free to view this file but **do not** modify it. A Makefile has been provided for you in the *test* directory. A file of test data is also present in this file *artist_song.txt*. To build the tests, type *make* inside the *test* directory from your computer's command line. To run, the tests after they have been successfully built, type *./runtests* in the *test* directory from your computers command line. You can build and run your tests with a single line by typing *make && ./runtests*. When you initially run your tests (before implementing your solution), you will see the following at the end of the output from the *./runtests* command:



```
===============================================================================
test cases: 10 | 2 passed | 8 failed
assertions: 11 | 3 passed | 8 failed
```

After successfully implementing your solutions, you will see a message such as the following in the console after executing the *./runtests* command:



```
===============================================================================
All tests passed (11 assertions in 5 test cases)
```

You may need to install the GNU g++ and GNU make on your computers in order to properly compile your code and run the tests. Instructions for downloading and installing these programs are included below for Windows:

4.1 g++

`http://www.codebind.com/cprogramming/install-mingw-windows-10-gcc/`

4.2 Make

`http://gnuwin32.sourceforge.net/packages/make.htm` (Windows)

Things are a bit simpler for Mac OS X users. You need to install Xcode and the command line tools (both are free). Instructions can be found here:

`https://www.embarcadero.com/starthere/xe5/mobdevsetup/ios/en/installing_the_commandline_tools.html`

If you are having trouble setting up these programs, please come talk to me about it as soon as possible.

5 Evaluation

The only file that you need to submit is your version of *artist.cpp* on Blackboard for this assignment. If your code passes all of the tests for this assignment **and** satisfies the requirements for the functionality of each function above, you will receive all available points.