CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 7, Oct 2018

Hash Tables

OBJECTIVES

1. **Use a hash**
2. **Store data in a chained hash table**
3. **Search for data in a hash table**

Background

This assignment will recreate assignment 2, but using hash tables instead of vectors. You are welcome to use assignment 2 code wherever possible.

Assignment

- Use the text files from homework 7, **not** those from homework 2. Like before, you will be calculating the top N words and the total number of unique and non-unique words.
- Your program must take **4** command-line arguments:
    1. The number of most common words to print out
    2. The name of the text file to process
    3. The name of the stop words file
    4. The size of your hash table
- You must implement the functions declared in the header file on Moodle: HashTable.hpp. **Do not** modify this header file. You will also need to write a main function. We will assume your main function is written in a separate file while autograding. If you would like to write all of your code in one file, you will have to split it up when submitting it.
- Similar to assignment 2, your program should do the following:
    - Read in a list of *stop words* from the file specified by the *third* command line argument. There will always be exactly *50* stop words.
    - Build a hash table with N boxes, where N is the *fourth* command line argument. Use the hash function detailed below in the description of the *getHash* function.
    - Read in every word from the file specified by the *second* command line argument. Store all *non-stop words* in the hash table. Do not store words multiple times - instead, each word is stored with a *count* variable that indicates how many times it appears.
    - Print out the top N words, where N is the *first* command line argument, along with some other information (detailed below).

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 7, Oct 2018

- The header file and example text files are available on Moodle under the names *HashTable.hpp*, *HW7-HungerGames_edit.txt*, and *HW7-stopWords.txt*.

- Your hash table should be stored in the private *hashTable* variable, which should be a dynamically allocated array of pointers to *wordItem* structs. Each of those *wordItem* structs stores a word and how many times that words has appeared, as well as a pointer to to the next *wordItem* struct with the same hash, in the event of a hash collision.
- Your output should be formatted like this:

```
<Count1> - <Word1>
<Count 2> - <Word 2>
…
<Count N> - <Word N>
#
Number of collisions: <Number of collisions>
#
Unique non-stop words: <Number of unique non-stop words>
#
Total non-stop words: <Total number of non-stop words>
```

This output can be generated using the following commands:

- In the *printTopN* function:

```
cout << ” - “ << wordItem.count << ” - “ << wordItem.word << endl;
```

- In the *main* function:

```
cout << ”#” << endl;
cout << ”Number of collisions: “ <<
     hashTable.getNumCollisions() << endl;
cout << ”#”<<endl;
cout << ”Unique non-stop words: “ <<
     hashTable.getNumUniqueWords() << endl;
cout << “#” << endl;
cout << “Total non-stop words: “ <<
     hashTable.getTotalNumberNonStopWords() << endl;
```

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 7, Oct 2018

- You will have to implement the following functions in the hash table class:
  - `HashTable::HashTable(int hashTableSize):`
    The constructor should set the *hashTable* variable to be a dynamically allocated array of *wordItem* structs, of size *hashTableSize*, as well as set the *hashTableSize* member variable so that other functions can use it.

  - `HashTable::~HashTable():`
    The deconstructor should free all the dynamic memory allocated by the table.

  - `int HashTable::getHash(std::string word):`
    The *getHash* function should return the hash of word using the DJB2 algorithm. You can look up that algorithm online for more information on how and why it works well, but to save you some time, here is the pseudocode for the algorithm:

```
int hash(string word):
    int hash = 5381;
    for each character c in word:
        hash = hash*33 + c
    hash = hash % hashTableSize
    if(hash < 0) hash+=hashTableSize
    return hash;
```

  - `wordItem* HashTable::searchTable(std::string word):`
    The *searchTable* function should search the table and return a pointer to the wordItem struct containing word, or NULL if the word doesn't exist in the table.

  - `void HashTable::getStopWords(char *ignoreWordFileName):`
    The *getStopWords* function should read stop words from *ignoreWordFileName*, which should be formatted with exactly 50 words, one per line. It should store these words in the *vecIgnoreWords* vector, which is a member of HashTable.

  - `bool HashTable::isStopWord(std::string word):`
    The *isStopWord* function should whether or not the given word is a stop word.

  - `bool HashTable::isInTable(std::string word):`
    The *isInTable* function should return whether or not the given word is in the table.

  - `void HashTable::incrementCount(std::string word):`
    The *incrementCount* function should add 1 to the count of the given word.

- `void HashTable::addWord(std::string word):`
  The *addWord* function should insert word into the table with a count of 1. If the word collides with another that already exists in the table, add the new word to the end of the linked list chain.

- `int getTotalNumberNonStopWords():`
  The *getTotalNumberNonStopWords* function should return the total number of non-stop words in the entire tree, counting duplicates.

- `void HashTable::printTopN(int n):`
  The *printTopN* function should print the top *n* most common words in the table.

  Hint: You may need to declare a separate array to store the contents of the hash table in such a way that you can sort out the top N words.

- `int HashTable::getNumUniqueWords():`
  The *getNumUniqueWords* function should return the number of words in the table, not counting duplicates of a word. This is **not** suppose to count the number of words that only appear once in the text!

- `int HashTable::getNumCollisions():`
  The *getNumCollisions* function should return the number of collisions in the table.

● You will also need to write a main function to create an instance of the hash table, read in the text file, and print out information to the user as was described above.

Submission

Submit your code on Moodle by following the directions on Assignment 7 Submit. You must submit an answer to be eligible for interview grading!