BLG335E, Analysis of Algorithms I, Fall 2016 Project 4

Handed out: 28.11.2016

Due: 12.12.2016, until 23:59

PURPOSE:

The purpose of this project is to develop a better understanding of hash tables. This is accomplished by writing a hash table class and methods.

Part A. Questions on Hash Tables (20 points)

1) Why do we use Hash Tables as a data structure in our problems? Please explain briefly. **(5 points)**

2) Consider a hash table consisting of M = 11 slots, and suppose nonnegative integer key values are hashed into the table using the hash function `h1()` :

```
int h1 (int key) {
int x = (key + 7) * (key + 7);
x = x / 16;
x = x + key;
x = x % 11;
return x;
}
```

Suppose that collisions are resolved by using linear probing. The integer key values listed below are to be inserted, in the order given. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (if any) for each key, and the final contents of the hash table after the following key values have been inserted in the given order: **(10+5 points)**

Key Value	Home Slot	Probe Sequence
43		
23		
1		
0		
15		
31		
4		
7		
11		
3		

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents											

Part B. Implementation and Report (80 points)

1) Implementation (70 points)

Problem: In this assignment, you will create a simple spell checker. The program will read some words from a file that is given as the first argument (words.txt), and apply the appropriate method to the word. The structure of the file can be seen below:

```
insert:cat
delete:dog
retrieve:door
```

While reading the dictionary, the spelling checker might run for the lines which starts with "retrieve". Each word (with the *retrieve* prefix) will be looked up in the dictionary. If it is not in the dictionary, it will be written to the standard output together with a list of suggested corrections.

In your implementation, there will be three different operations:

- **insert (key)**
  Inserts **key** into the appropriate slot in the hash table. The location (**index**) in the hash table is determined by the key and the hash function.
- **remove (key)**
  Searches the hash table for the given **key** and delete it from the table. In order to delete a key from a hash table slot, you should just mark the corresponding slot as **deleted** (Lazy deletion). Since your hash table will be `string` type, you can use any alphanumeric (except a…z) character to mark as "deleted" (e.g. `hashTable[deleteIndex]="*"`). **Remember that, during insertion, slots marked as deleted are treated as empty.** So, you have to use two different markers both for deleted and emptiness.
- **retrieve (key)**
  Finds the given **key** in the hash table and return its **index**. If the key is not in the first slot, you have to apply linear probing from the current slot to the next empty slot. Note that the deleted slots are NOT treated as empty in this function. If the key is not in the table, it should start for correction and finally serves possible suggestions.
  Note: You may implement your spell_checker() method separately if you wish.

The Hash Function:

Let the word to be hashed be $w_1w_2...w_n$ where each w_i $1 \leq i \leq n$, is a single character. Also, let ascii(x) be the ascii value of the character x.

$f(w_1w_2...w_n) = [acsii(w_1)*ascii(w_2)*...*ascii(w_n)] \bmod n$

where, n is the table size which will be given as the second argument (**Use n=337** for this project but still you have to give as command line argument).

Since the result of the multiplication might be very large number, **use long long int data type for the result.** However, keep in mind that when you do this computation, you may **still** encounter with **overflow** issues. You'll have to think of a way to prevent it.

Here is a small example for n=59999:

Consider inserting the word "dog" into the hash table:

ascii('d')*ascii('o')*ascii('g') mod 59999 = 100*111*103 mod 59999

= 1143300 mod 59999

= 3319

Use the linear probing strategy for your hash table.

Generating Corrections

The easiest way to generate corrections in a spell checker is a trial-and-error method. If we assume that the misspelled word contains only a single error, we can try all possible corrections and look each up in the dictionary.

Traditionally, spelling checkers have looked for four possible errors: a wrong letter ("wird"), an inserted letter ("woprd"), a deleted letter ("wrd"), or a pair of adjacent transposed letters ("wrod").

In this assignment, you will only need to deal with the first possibility; a wrong letter. When a word isn't found in the dictionary, you will need to look up **all variants** that can be generated by changing one letter. For example, given "wird," you should look up "aird", "bird", "cird", etc. through "zird", then "ward", "wbrd", "wcrd" through "wzrd", and so forth. Whenever you find a match in the dictionary, you should add it to your output line. Example output line for the correction / suggestion is like that:

```
SUGGESTIONs for wird: bird gird ward word wild wind wire wiry
```

Important Notes:

1) In your implementation, put the following messages at the appropriate lines:

```
cout<<"INSERT: The word '"<<s<<"' is put in the cell number "<<index<<endl;
cout<<"WARNING: The word '"<<s<<"' is already in the dictionary!"<<endl;
cout<<"WARNING: There are no empty space in the hash table for the word: "<<s<<endl;
cout<<"RETRIEVE: The word '"<<s<<"' found in the dictionary with index: "<<index<<endl;
cout<<"REMOVE: The word '"<<s<<"' is removed from the dictionary."<<endl;
cout<<"WARNING: The word '"<<s<<"' couldn't be found in the dictionary"<<endl;
```

2) All your code must be written in C++ **using object oriented approach** and able to compile and run on Linux using g++. Here is the definition for the HashTable class:

```
class myHash {

    public:
       int _tableSize;
       int collision=0;
       myHash(int tableSize);   // Creates a hash table that can store up to tableSize entries.
                                  (Constructor) 5 POINTS
       void insert(string s);   // Inserts w into the hash table. 15 POINTS
       int hashFunc(string s);  // Computes the hash value of w. 5 POINTS
       bool retrieve(string s); // Finds index of a given word in the hash table, if it isn't in
                                   the table, starts for spell checking and suggests similiar
                                   words if any found. 30 POINTS
       bool remove(string s);   // Removes given word from the dictionary. 15 POINTS
    private:
        string *myHashTable;    // Stores words.
        int totalEntries=0;     // Stores the current number of entries in the table.
};
```

3) Do not use external libraries such as STL. We are going to compile your codes at ITU SSH server, so you **MUST** test your code at ITU SSH server as well. The codes which are not tested at ITU SSH server, will **NOT** be evaluated.

4) Do NOT send your own project files such as .sln, .cbp or any other IDE file. ONLY send your .cpp file, .h (if you have one) and words.txt file.

5) In your report, explain your classes and methods briefly. (10 POINTS)

6) You are given an example report.doc file with the Question 1, please use it for your complete report and save as **StudentID.PDF**

7) Save your .cpp and .h files with your student ID (Ex: StudentID.cpp/h)

8) During the execution, remove, retrieve and insert methods should give explanatory messages to the screen (see Note 1). At the end of the execution, total number of collisions should be printed out. An example output can be seen below: (Note that the numbers in the output are imaginary).

```
INSERT: The word 'above' is put in the cell number 315
COLLISIONS :0
-------------------------------------
h(x) result is 79 for: wild
INSERT: The word 'wild' is put in the cell number 86
COLLISIONS :7
-------------------------------------
h(x) result is 319 for: written
WARNING: There are no empty space in the hash table for the word: written
COLLISIONS :337
-------------------------------------
h(x) result is 140 for: which
WARNING: The word 'which' is already in the dictionary!
COLLISIONS :99
-------------------------------------
RETRIEVE: The word 'options' found in the dictionary with index: 282
COLLISIONS :97
-------------------------------------
The word 'wilb' couldn't be found in the dictionary.
Looking for possible suggestions.
SUGGESTIONS for wilb: wild, will,
COLLISIONS :78
-------------------------------------
REMOVE: The word 'these' is removed from the dictionary.
COLLISIONS :45
-------------------------------------
WARNING: The word 'tyx' couldn't be found in the dictionary.
COLLISIONS :25
-------------------------------------
INSERT: The word 'the' is put in the cell number 209
COLLISIONS :0
-------------------------------------
REMOVE: The word 'about' is removed from the dictionary.
COLLISIONS :14
-------------------------------------
RETRIEVE: The word 'by' found in the dictionary with index: 64
COLLISIONS :15
====================================
TOTAL COLLISIONS: 105000
Final form of hash table is saved as StudentID_output.txt
```

9) As you see in the example output, you should save your final form of hash table as StudentID_output.txt. Example file content is:

```
0: kuenning
1: look
2: match
3: mode
4: possibility
5: program
```

10) All your work will be tested with a **different** data set. For this reason, make sure that your work is applicable to any data set in this format.

11) Your program should be run from the command line with the following format:

```
$ ./studentID words.txt N
Example: $ ./15005985 words.txt 337
```

If you have any questions, please feel free to contact Res. Asst. Emrullah GAZİOĞLU via e-mail: egazioglu@itu.edu.tr