Hash Tables Project 7   240 points

Due Monday December 11, 11:30-1:00
in the Engineering College Library, PL 2600

With Thanks and Credit to: John Edgar, Simon Fraser University, who devised the original version of this assignment.

Extra credit. A significant amount of extra credit is allowed on this project (see below). Use a cover sheet to state what you did. Any extra credit that isn't explicitly stated on the cover sheet may be overlooked and thus not counted, or graded at a reduced value because we do not realize the full impact.

Assignment

In this assignment you are to implement a hash table for use in a simple spell checker. The spell checker should be set up with a GUI for the interface. The hash table MUST be built as specified in this document. The GUI can be built with Java Swing or JavaFX. There is extra credit of 20 points for using JavaFX.

Partial credit will be awarded to non-working projects. The assignment is worth 220 points, allocated as follows:

Correctness and completeness - 180
Having a minimum reasonable Program Style - 20
Human Engineering of the GUI - 20

There is an automatic deduction of 70 points for any code longer than 10 lines copied from the Internet.

The project can be done in pairs, and we encourage this. There is no penalty or extra credit for a pair submission.

Submission

A printed version of you project must be submitted to one of the graders, who will be in the Engineering College Library from 11:30-1:00 on Monday December 11. If you need to make other arrangements or your project is late, contact Naren via the email below.

Naren's email: Naren.Thanguturi@rockets.utoledo.edu

Projects may be submitted up to 2 days late, and will be charged a 60 point penalty, even if one hour late. Later submissions will not be accepted. Email submissions are NOT allowed. As in previous projects, see the posted instructions for

Printed Submission of Projects.pdf

You submission must include

a) A printout of the results from running your program using the two example test sets given at the end of this document. The printout should highlight or mark any words that do not appear in the dictionary.
b) The results of any other document given as input
c) The number of collisions in the hash table and the length of the longest chain
d) A "ReadMe file" explicitly stating any grounds for extra credit.

There should be multiple sample runs to demonstrate all required functionality.

Hash Table Class

Implement a hash table to store strings (i.e. objects of the Java String class). The Hash Table must be implemented as a data type with the class name "HashTable". The HashTable class will have the following methods:

HashTable ( )		Constructor for a hash table of default size: 1000
HashTable (int n)	-	Constructor for a hash table of size n.
void Insert (String S)		Inserts S into the hash table.
boolean Contains (String S)		Return true if S is in the table, false otherwise
int Count ( )	-	Returns the number of strings stored in the table

Hash Function

The insertion of a word requires the computation of an array index

array index = HashValue **mod** array size

The insert function must thus generate a hash value from a given word w. The hash value should be calculated by assigning the values 1 to 26 to the letters **a** to **z** (regardless of letter case) and calculating a numerical value based on the letters in the word as follows:

Group the letters of w in chunks of 4. For each four letters, calculate

$ch_1 * 31^3 + ch_2 * 31^2 + ch_3 * 31^1 + ch_4 * 31^0$

and add the total value of each chunk to get the hash value. For the actual index of the word, compute

HashValue **mod** array size

For example the string "cat" has the hash value 90,954:

$3*31^3 + 1*31^2 + 20*31^1 = 90,344$

and the string “table” has the hash value 745,810, calculated as follows

tabl	$= 20*31^3 + 1*31^2 + 2*31^1 + 12*31^0$	= 596,855
e	$= 5*31^3$	= 148,955
596,855 + 148,955		= 745,810

For a hash table of size 101, “cat” maps into the index

90,954 mod 101 = 54

and “table” maps into the index

745810 mod 101 = 26.

Note: In Java, there is a built-in hashCode function which can be applied to a string object S

S.hashCode()

This built in function computes the hash code for a string S as

```
S[0]*31^(n-1) + S[1]*31^(n-2) + ... + S[n-1]*31^(n-n)
    = ch1*31^(n-1) + ch2*31^(n-2)  + ... + chn
```

where `S[i]` is the `i`th character of the string, `n` is the length of the string, and `^` indicates exponentiation. This function may **NOT** be used.

Insertion and Collisions

Your insert function must deal with collisions (a collision occurs when two different strings have the same hash value). You **must** use **linear probing** to deal with collisions. You should test your insertion method carefully. In particular you should ensure that insertion still works correctly when the hash table is nearly full.

For one test, consider setting your hash table's size to be smaller than the size of your word list (see below) and attempting to insert the entire word list into the hash table. This may cause errors that you might not encounter with smaller test data sets. When you get your hash function working correctly, and your hash table is an appropriate size you should be able to successfully enter all the words in the word list (see below) into it.

Let N be the number of strings to be stored in the table. Choose an array size that will have a load factor around 1/3. That is, make the size of the array larger than 3*N. You **must** choose a modulus that is prime (i.e. choose an actual array size that is a prime number just over 3*N) and have your program **automatically** choose this prime number. Automatically means that your program needs to use the size of the word file as a starting point in your search for a prime; do not hard-code a value that happens to work for this data set.

Spell Checker

Use your hash table to implement a simple spell checker. The spell checker should load a dictionary into a hash table, and then test each word in a given document to see if it is contained in the dictionary. For the dictionary you must use the list of frequently used words given in the text file "***`SpellCheckDictionary.txt`***" posted on the website. Store this file in the project for access by your program in the project.

There are two documents to be spell-checked given at the end of this assignment. Place the first document in a text file called TestFileDogs.txt. Place the second in a file called BillOfRights.txt. Your sample runs should include checking both of these files.

Your spell checker should thus be implemented as a program that reads in the dictionary file and the name of the file to be checked, and then checks the file for possible spelling errors.

Extra Credit

Extra credit may not be counted if it is not explicitly stated in the readme document. *It is your resposibility to document in the Readme file what you think counts as extra credit.* It is not our job so somehow deduce what you did for extra credit.

There are many options for extra credit.

a. Use excellent program style (see the posted programming guidelines). Pay special attention to the use of alignment, comments, and naming.
b. Develop a good graphical user interface.
c. Use JavaFX for the graphical user interface
d. Present good documentation of the project.
e. Augment your spell checker so that it tries to change one letter in an incorrect word to find some valid word to suggest.
f. Check a misspelled word against a list of common misspellings.
g. Use a special dictionary (there are some online).

h. Show that your hash table works correctly when the table is almost full or totally full.
i. Allow the user of the GUI to enter his or her own text to be checked.
j. Demonstrating the kind of test used to ensure that the program is reliable, without any kind of error.
k. Other options are also fine, just try to be reasonable.

There is no extra credit for the use of pointers, e.g. to achieve chaining of collisions.

Notes

The following incomplete code fragments may be helpful. The first reads a dictionary into a hash table:

```
File F = new File("Dictionary1.txt");
Scanner dictFile = new Scanner(F);
String line;
while (dictFile.hasNextLine()) {
  line = dictFile.nextLine();
  // insert the words in line into the hash table
}
```

For reading a document into an array of single words, consider:

```
public static String readFile(String fileName) {
  // Reads the text file fileName into a single string of words
  File F = new File(docName);
  Scanner docFile = new Scanner(F);
  String result, line;

  result = "";
  while (docFile.hasNextLine()) {
    line = docFile.nextLine();
    result += line + "\n";
  }
  return result;
}

public static String[] getWords(String str){
    // Breaks str into an array of words
    // Splits str on the basis of blanks
    // Returns an array of words
  }
```

For testing if an integer is prime:

```
boolean isPrime(int n) {
  if (n%2==0)          //check if n is a multiple of 2
    return false;
  else
    for(int i=3; i*i<=n; i+=2) {
```

```
            if(n%i==0)
              return false;
          }
        return true;
    }
```

Test Set 1 – Simple text

This is a test document.
How many dogs can fly?
Do you think this realy works?

Test Set 2 – The Bill of Rights

Article the first -- Not Ratified

After the first enumeration required by the first article of the Constitution, there shall be one Representative for every thirty thousand, until the number shall amount to one hundred, after which the proportion shall be so regulated by Congress, that there shall be not less than one hundred Representatives, nor less than one Representative for every forty thousand persons, until the number of Representatives shall amount to two hundred; after which the proportion shall be so regulated by Congress, that there shall not be less than two hundred Representatives, nor more than one Representative for every fifty thousand persons.

Article the second -- Amendment XXVII -- Ratified 1992

No law, varying the compensation for the services of the Senators and Representatives, shall take effect, until an election of Representatives shall have intervened.

Article the third -- Amendment I

Congress shall make no law respecting an establishment of religion, or prohibiting the free exercise thereof; or abridging the freedom of speech, or of the press; or the right of the people peaceably to assemble, and to petition the Government for a redress of grievances.

Article the fourth -- Amendment II

A well regulated Militia, being necessary to the security of a free State, the right of the people to keep and bear Arms, shall not be infringed.

Article the fifth -- Amendment III

No Soldier shall, in time of peace be quartered in any house, without the consent of the Owner, nor in time of war, but in a manner to be prescribed by law.

Article the sixth -- Amendment IV

The right of the people to be secure in their persons, houses, papers, and effects, against unreasonable searches and seizures, shall not be violated, and no Warrants shall issue, but upon probable cause, supported by Oath or affirmation, and particularly describing the place to be searched, and the persons or things to be seized.