FIT1008 Computer Science
Semester 1, 2013
Prac 9: Hash Tables

What's in this prac

This prac covers material from lectures L25 and L26. You will also be required to consult the web regarding the `String` class. The aim of this prac is to implement a simple dictionary using Hash Tables.

Background

The idea behind the program is to be able to (a) read a dictionary into a Hash Table, and (b) check whether a given word file is in the Hash Table. If not there, it will be assumed to be misspelt and reported. The dictionary is provided in the `dict.txt` file which contains, in each line, a word (string of lowercase alphabetic characters), and an integer number indicating the frequency of the word, i.e., whether the word is rare (1), uncommon (2), common (3) or unknown (0).

In order to do this we will create five main classes:

1. `Dictionary`: will provide the user with a menu designed to read dictionaries from files (such as the `dict.txt` file), determine whether a word given by the user appears in the dictionary, and print the dictionary (skeleton provided).
2. `Frequency`: stores a given frequency (provided).
3. `HashTable`: includes the required functionality of a hash table. It contain an inner class to store an entry including a key and value (provided).
4. `LinearProbe`: Extends `HashTable` implenting Linear Probing (provided).
5. `SeparateChaining`: Extends `HashTable` implementing Separate Chaining for Question 5.

Additionally you'll need to define an inner class in your Dictionary to store each word (Question 1). This word will be used as the key for the hash function and thus implements the `Hashable` interface.

The advanced questions implement a resizable hash table as well as comparing the frequency of rare, uncommon, common and unknown words in two nominated files, seeing how similar these two files are.

The frequency counts of the words were established by analysing 30MB of text from the Gutenberg project. Words that appeared more than 400 times were classed as common, between 101 and 400 occurrences of a word (inclusive) was classed as uncommon. Less than 101 occurrences of a word resulted in a rare classification. Note that the text analysed may not be representative of common English writing or speech, and may be biased towards scientific or historical usage.

Question 1: (1 mark)

Define an inner class in `Dictionary` called `Word` class that implements the `Hashable` interface for a key of type `String` (words that will be used as part of the dictionary).

Add the following methods to the class:

- `public int hash()`. You can use any of the hash functions defined during the lectures.
- `public String toString()`
- `public boolean equals(Object otherObject)`. Remember that the input argument is only known to be an `Object`, not a `Word`.

Important: If your hash function requires a prime and the table size, you will need to create these as variables in the `Dictionary` class, since the `hash` method does not have any parameters. Would it be a good idea to make them static (i.e., their value will be shared by all words)? Would it be a good idea to make them final (i.e., once assigned, you cannot change them)? You choose, but make sure you have a reason for the choice.

Have a look at the `Frequency` class, which stores the average frequency of a word as represented in the file `dict.txt`: unknown is indicated by 0, rare is indicated with a value of 1, uncommon words have the value 2, and common words use the value 3.

The class contains the following methods:

- `public Frequency(int value)`. Note that this method performs error checking on the values of the input.
- `public int getFrequency()`
- `public String toString()`

The class is implemented with enumerated types, which you have not seen in FIT1008. You might want to Google it or ignore it.

Question 2: (2.5 marks)

Modify the `Dictionary` class to create a Hash Table using Linear Probing and implement the following options in `public void menu()`:

1. `Read:` which asks the user for a filename, opens the file, and stores all the character strings in the Hash Table.
2. `Search:` which asks the user for a word, looks up the word in the hashtable and either prints the frequency of that word, or reports if it is not found.
3. `List:` which prints out all the strings stored in the Hash Table with their associated frequencies.

Important: You must use (as in call, not cut and paste) the linear probing implementation implemented by the files `LinearProbe.java` and `HashTable.java` which are provided in the `prac10.zip` file. You must try the workings of your menu with the also provided file `dict.txt`, where each of its lines contains only two things: a word (alphabetic string) and an integer in the range 0 to 3. Make sure that the Hash Table you create has the appropriate size for storing all words in `dict.txt`. And make sure the string you use to create a word is in lowercase. Remember, you can use Java methods like `boolean isAlpha(String str)` and `String toLowerCase()` to make your task easier.

`Also important`: You are not allowed to simply declare a precondition every time you want to avoid checking something. For example, you cannot simply add a precondition (to the reading method) indicating the file must contain in each line two things: a lower case word and an integer in range 0..3. The point of pracs is to make sure you perform those checks and appropriately handle the cases in which the checks return `false`.

Question 3: (1.5 marks)

Modify the `HashTable` class provided to include two counters: `collisions` and `probes`.

When searching, a collision is defined by hashing to a non-empty entry that doesn't contain the key you're searching for (i.e., we have to keep searching). When inserting, a collision is when you hash to a non-empty entry.

Probes are incremented after a collision occurs. When searching, if you don't find the value straight away, each iteration after that collision is considered a probe. When inserting, the probe will keep being incremented until you find an empty entry.

Add two methods to the `HashTable` class `resetCounters()` and `displayCounters()` to allow us to reset the counters and display their values. Modify the `Dictionary` class above to print the value of these counters every time the menu option for search or insert is given by the user. Also, modify the `main` method in the `Dictionary` class to use:

- a table size of 97, a prime of 53
- a table size of 20089, a prime of 12289
- a table size of 20089, a prime of 1
- a table size of 20080, a (non)prime of 2
- a table size of 40283, a prime of 3
- a table size of 40283, a prime of 20089

and, for each of the above values, use the menu developed in the previous question to (a) read the file `dict.txt` and (b) search for the word `zaproater`. Explain what happens in terms of correct behaviour and in terms of the above counters. Explain why it happens.

Important: if you have used the universal hash function given in page 24 of lecture L25, you might not see much difference. Try the hash function given in page 18.

Question 4: (3.5 marks)

Create a `SeparateChaining` class that is similar to `LinearProbe` but follows the separate chaining method, i.e., it stores its data in a linked list of nodes. You should not need to modify the `Dictionary` class, except for making the new object.

Note: You are welcome to use the LinkedList class from the Java Standard Library.

Question 5: (1.5 marks)

Repeat the experiment of Question 3 for the separate chaining hash table.

Note: Probing is not applicable for separate chaining. However, for the purpose of the exercise, we'll consider iterating over the chain as probing.

Advanced Questions: (2 marks)

In the above implementation of a Hash Table the size of the table was fixed. In this part of the prac we extend the implementation so that the size of the table is dynamic. To implement a dynamically resizable hashtable, the function `resize()`, which changes the size of the Hash Table needs to be implemented. The hashtable should start with a small size in case a small dictionary is being used in order not to waste space. Thus, you should set the initial size of the Hash Table to 11, and the prime number to 7. When the hashtable is full, a new larger one needs to be created and used.

You will need to change the hash function when you change the size of the Hash Table. Furthermore, you will also need to insert all the items from the old Hash Table into the new Hash Table. To do this, use the following list of prime numbers (but don't use the same values at the same time for both the table size and the prime value).

7, 11, 23, 53, 97, 193, 389, 769, 1543, 3079, 6151, 12289, 24593, 49193

Hall of Fame

Change your program to read and compare two input files specified at the command prompt, for example:

```
q2 dict.txt infile1 infile2 outfile
```

You are not to prompt or read anything from the user at any time. Instead of printing the misspelt words to the outfile, print the frequencies of unknown, common, uncommon and rare words in both files to the output file, in percentages limited to two decimal places. To find how similar two files are, you can add up the absolute differences between all the pairs of frequencies (or you can determine your own, more interesting measure). The larger the number, the more different the two files are. Comparing a file to itself should give the value 0. Print this value to the file as well.

Example output:

```
File: infile1
 unknown: 37.46
 rare: 13.07
 uncommon: 7.07
 common: 42.40

File: infile2
 unknown: 28.39
 rare: 15.58
 uncommon: 5.53
 common: 50.50

Similarity indicator: 21.21
```