Assignment overview

This week we will investigate the effects of different hash table sizes and hash functions on the number of collisions that occur while inserting data into a hash table using simple hash techniques.

For an explanation of hashing, see Lecture 7, and also Chapter 7.3 in the textbook.

Your code will:

- Create a hash table (array) of a particular size
- Read an input file containing a list of names
- Store each name in the hash table
- Count the number of *collisions* that occur (not *probes*, see note in Details section)

Then you will use your program to perform enough testing to complete the table below:

- For 3 different data files
- For 4 different hash table sizes (per file)
- For 3 different hash functions

NOTE: You do not need to construct your program to do all of that at one time. It is OK to run the program repeatedly with different data files and hash table sizes.

Input file	**Declared size of hash table array**	**Hash function H1 #collisions**	**Hash function H2 #collisions**	**Hash function H3 #collisions**
37_names.txt	37			
37_names.txt	74			
37_names.txt	185			
37_names.txt	370			
333_names.txt	333			
333_names.txt	666			
333_names.txt	1665			
333_names.txt	3330			
5163_names.txt	5163			
5163_names.txt	10326			
5163_names.txt	25815			

5163_names.txt	51630			

Detailed requirements

Implement your hash table as a plain array (of Strings) in Java. (ArrayList is OK.) *Do not use* any of the Hash-related data types that are available in Java (HashMap, HashSet, etc.).

For collision resolution, your program should use *closed hashing* with *linear probing*. Linear probing is the *closed hashing* technique described on slides #68-71 in Lecture 7, and page 272 of the textbook.

A *collision* occurs when an item's hash function result is a bucket that already contains another item. This is different from the number of *probes*, which is how many more buckets you have to try before you find a place for that item. *One collision* could lead to *any number* of probes.

In this lab you are counting *collisions, not probes.*

Your code does not need to produce all of the table data in a single execution; you can run it multiple times with different inputs and settings. However, it's certainly nice if you want to take the extra effort.

Input files

Unlike on an earlier lab, these filenames show the exact number of items in the file.

- 37_names.txt
- 333_names.txt
- 5163_names.txt

Tip: There are no newlines in these data files; the names are separated by commas. Read the contents as a single string and then use str.split() to divide it into an array of strings.

Hash functions

A hash function (for our purposes in this lab) takes two arguments: 1) a string; 2) N, the size of the hash table, and returns an integer in the range of [0..N-1].

H1 – Let A=1, B=2, C=3, etc. Then the hash function H1 is the sum of the values of the letters in the string, mod N. For example, if the string is BENNY, the sum of the letters is 2+5+14+14+25 = 60 (and then you would take 60 mod N).

H2 – For the ith letter in the string (counting from 0), multiply the character value (A=1, B=2, C=3) times 26^i. Add up these values, and take the result mod N. For BENNY the partial result would be 2*1 + 5*26 + 14*676 + 14*17576 + 25*456976 = 11680060. For the final answer you will take this partial result mod N. **WARNING: This function might cause integer overflow if you use ints!**

H3 – *Invent your own hash function!* Pull one right out of your imagination, or Google around. **Write good and clear comments in your Java code describing how your hash function works. If you found it online, give the source.** Your goal should be to find a hash function that results in very few collisions.

Submission and marking

Due date: As shown on Learning Hub. Your last submission will be the one that counts. Late assignments will not be graded.

What to submit:

- Your Java source code (file name not important, please not zipped)
- The completed table of results (Word doc, Excel, plain text – whatever makes you happy)

Marking: This lab is worth 15 marks.

- Implementation of the hash table + collision counting – 4 marks
- H1 hash function – 1 mark
- H2 hash function – 1 mark
- H3 hash function – 3 marks
- Completed table of results – 3 marks
- Main program and coding style – 3 marks