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 (last week's slides, in the part we didn't cover yet), and also Chapter 7.3 in the textbook.

Your code will:

- Declare 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

Do the above procedure:

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

…and report all of the results in the table below.

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 in Java. Do not use any of the Hash-related data types that are included in Java libraries.

For collision resolution, use *closed hashing* with *linear probing*. (Linear probing is the technique described on slide #68 in Lecture 7, and page 272 of the textbook.)

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.

Input files

These are (some of) the same data files that we used for Lab 3. These filenames now show the exact number of items in the file.

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

Hash functions

A hash function 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 i^{th} 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 result mod N.

H3 – *Invent your own hash function!* Pull one 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)
- 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 – 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