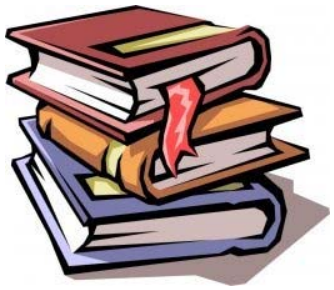
CS 202-Data Structures

Dr. Ihsan Ayyub Qazi



Assignment 3

Hash Tables

(Due: 11pm on Monday, April 3rd, 2017)

In this assignment, you are required to implement different types of hash tables. You will then store a dictionary of words and compare the lookup times for the different types of hash tables. The dictionary file is provided to you.

The course policy about plagiarism is as follows:

1. Students must not share actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students cannot copy code from the Internet.
4. Students must indicate any assistance they received.
5. All submissions are subject to automated plagiarism detection.

Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee

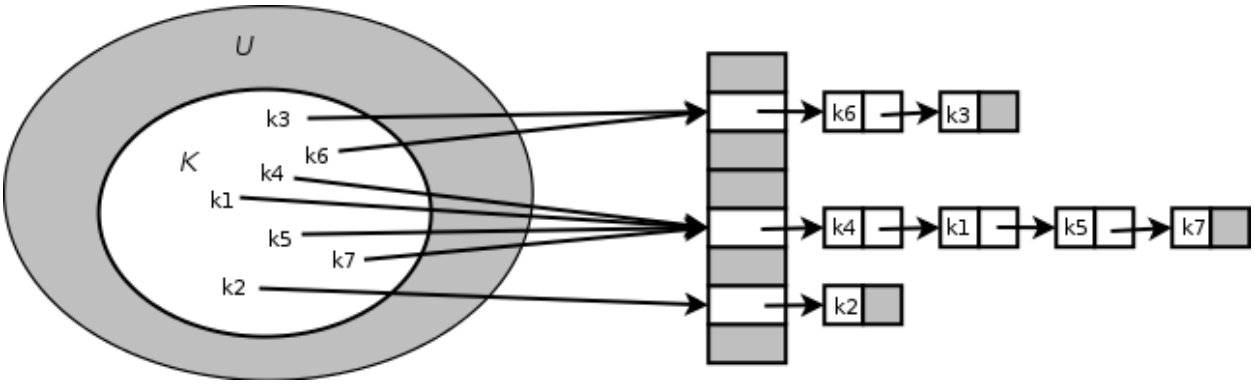
Task 1:

In the first task of this assignment, you have to implement the polynomial hash code from the textbook (Section 9.2.3) along with the division method compression function. The value of the parameter a should be configurable.

Task 2:

The specifics of the first hash table are as follows:

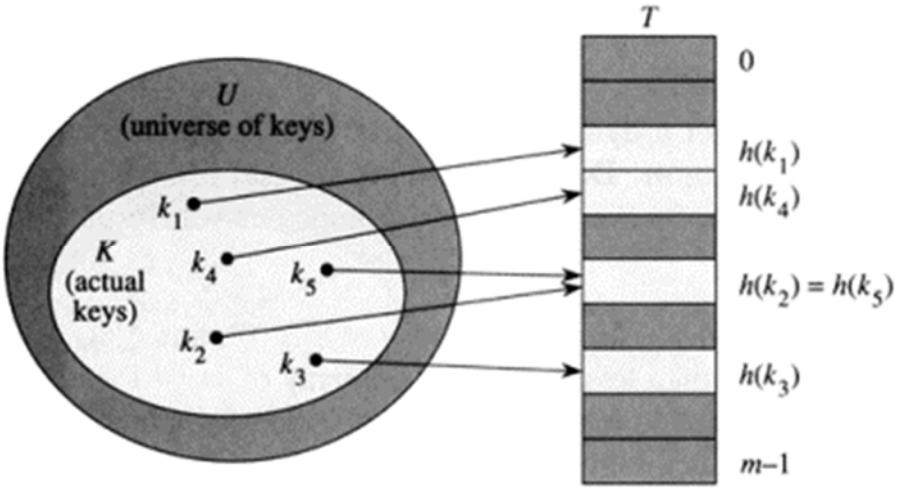
The first hash table will use chaining, where you will be required to use the LinkedList from previous assignments. This HashTable will be created with a fixed size. It should support the insert and lookup commands. The constructor should take, as a parameter, the value of a.



Counting collisions: As far as collisions are concerned, if you hash a value, and the index it hashes to has a non-empty linkedlist, the collision is counted only once. Do not think of the length of the linked list at that slot as the number of collisions for this assignment.

Task 3:

Now you will try out the same hash function with a different hash table, which should use open addressing with linear probing. This HashTable will initially be created with a small size; it must support resizing along with insert and lookup. Similar to the previous hash table, the constructor should take, as a parameter, the value of the parameter a.



***Counting collisions*:** As far as collisions are concerned, if you hash a value and the index the value hashes to is occupied, count that as one collision. When you start linear probing to find the next free slot, however many occupied slots you encounter, each of them will also count as collisions.

Task 4:

As you have seen in the implementations of linear probing and chaining, the issue of collisions was addressed by storing both the colliding values, but these techniques increase the look up time. So, in order to improve this, in this task you will be implementing *double hashing* as discussed in the class using the following functions:

str = $a_1a_2a_3\ldots a_{n-1}a_n$
e.g., (in case of str = "Hello", 'H' is a_1, 'e' is a_2 … 'o' is a_5)

Initialize Hash1 = 0
For every a_i in str
Hash1 ^= (Hash1 << 5) + (Hash1 >> 2) + a_i

Hash2(str) = Hash function used in Task 1

Note: Double hashing cannot completely eliminate collisions. To obtain full credit in this task, you will have to devise and implement a method to handle the case when both functions result in a collision.

One method to adopt, for example, would be the following:

Index = h(key) + i*d(key), where **h()** is the first hash function, **d()** is the second hash function and **i** is an integer zero onwards (0,1,2,3……)

Hence to compute the index to insert the value at, use the above formula but keep the value of **i** as 0. If you get a collision then use 1 as the value of **i**. If you get a collision again, use 2 as the value for **i** and so on.

***Counting collisions*:** As far as collisions are concerned, if you hash a value and the index the value hashes to is occupied, count that as one collision. Every time you hash it again, and you get a collision, it will be counted as well.

Task 5:

Now you just have to write a program to tie it all together. The program should read dict.txt, and then load all the words into a vector upon startup. The same should be done for queries.txt, for which the words should be loaded into another vector. This time should not be accounted for later on.

The program you write should compare the runtime of all hash tables. Varying the value of parameter a between 1 and 30 (in increments of 1), measure the number of collisions and the average lookup time for all implementations of hash table.

Task 6:

You will have to devise and implement an **efficient** function to tell whether a certain word is present in the given text file or not using any hash table technique (e.g., chaining or open addressing) you like but you must use the hash function *Hash1* listed in Task 4.

(*This is an open-ended question and may have multiple solutions*)

Deliverables:

You are required to submit the following:

1. Implementation of the hash table with chaining
2. Implementation of the hash table with open addressing (linear probing)
3. Implementation of the hash table with open addressing (double hashing)
4. Graphs comparing the lookup time and number of collisions for various values of a, for each type of hash table
5. Function implemented in Task 6