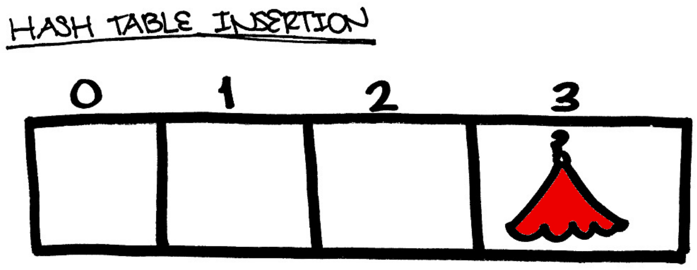
CS 202-Data Structures

Dr. Ihsan Ayyub Qazi

Assignment 3

Hash Tables

(Due: 11pm on Friday, April 6th, 2018)



In this assignment, you are required to implement different types of hash tables.

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 are expected to implement (a) the polynomial hash code from the textbook (Section 9.2.3) and (b) bitwise hash code function as shown below:

str = $s_1 s_2 s_3 \ldots s_{n-1} s_n$
e.g., (in case of str = “Hello”, ‘H’ is s_1, ‘e’ is s_2 … ‘o’ is s_5)

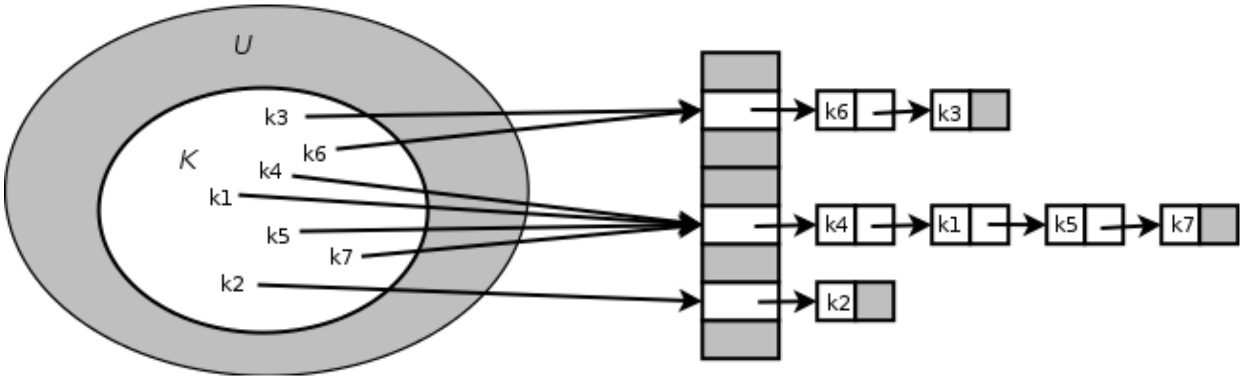
```
Initialize bitwise_hash = 0
For every s_i in str
    bitwise_hash ^= (bitwise_hash << 5) + (bitwise_hash >> 2) + s_i
```

along with the division method compression function for both (a) and (b). The value of the parameter `a` in the polynomial hash code should be configurable.

Task 2:

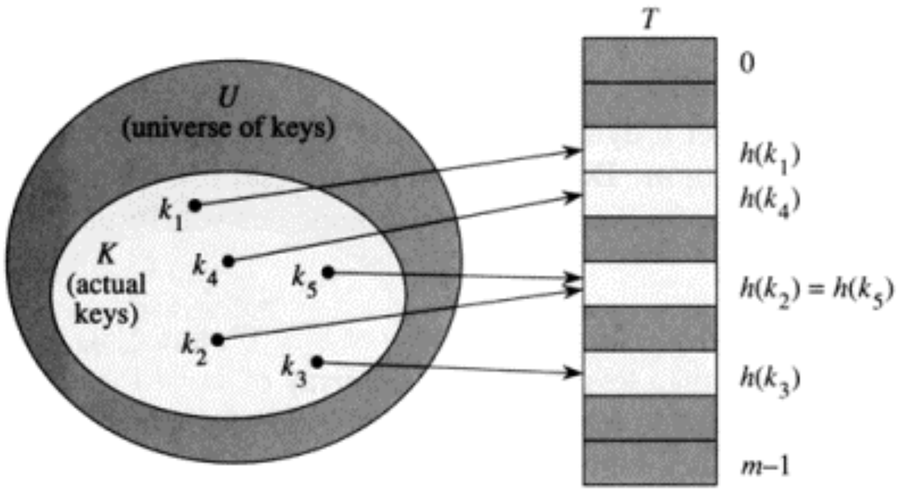
The specifics of the first hash table are as follows:

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, deletion and lookup commands. The constructor should take the size of the table as a parameter.



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 Hash Table will initially be created with a small size; it must support resizing along with insert, deletion and lookup.



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 two functions you made in Task1:

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.

Task 5 (Bonus):

In this task you are going to analyze the performance of different methods of chaining. You are expected to implement Hash Tables with different data structures to resolve collisions.

1. Arrays (vectors)
2. BST
3. Hash Table

You are expected to take into account their memory footprints, load time and access time.

(This is an open-ended question)

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)

Note: In order to compile the test cases, you will be required to give the following flags:

```
-pthread –std=c++11
```

As shown in the following example:

```
g++ test_chaining.cpp -pthread –std=c++11
```