Tony Diep
4-6-17
CS 2420
Design Document

Design Document: Hashing

The purpose of this assignment is to understand and apply the hash function and implementation. The assignment is divided into two parts: the first part deals with a simple password attack program with given text files containing a list of words and a list of hashes that correspond to those words. We will be using the MD5 algorithm for that part to encrypt common passwords and not so common passwords. The second part deals with implementing different variations of hashing functions regarding distinct ways of probing.

The assignment requires several classes to be implemented: for the first part, we are required to create a static class called “Crack.java” that contains a list of library methods used for analyzing different algorithms (i.e. brute force, dictionary, and multi-threading attacks). For implementing the brute force algorithm, we are to choose to provide the implementation of the method using recursion or the other methods demonstrated in lecture (that is, to compute the sum of all of the characters in a word and that will be the hash that corresponds to the word). For my implementation, I will first choose the implementation that computes the sum of all of the characters in the word as the hashing function. Later on, if I know how to implement the recursive way, then that will be the hashing function to use. Of course, it is strongly advised to utilize the pseudocode shown in lecture.

For the second part of the assignment, we are to create two separate classes in addition to a given class called Hash_Table_Linear_Probing. We recall that a hash table utilizes a technique called probing to find empty spots in which hash numbers can be placed. There are times where more than one hash number corresponds to the same “spot” which is called a “collision.” With linear probing, the hash table checks every possible spot one by one for an empty spot to place a new key. However, there are two other ways of probing that may or may not resolve the collision issue. The two classes that represent these other ways of probing are the Hash_Table_Quadratic_Probing and Hash_Table_Chaining.

After implementing the two parts, we are to create statistical analyses that performs timing experiments on the hash table and the hashing functions. There are three ideas we are interested in testing: 1) the number of times a hash table performs add, remove, and find operations, 2) the number of collisions occurring while the hash table performs those operations, and 3) the time it takes for the hash table to perform the hashing functions. We are also to compare the functionalities of using an array versus a hash set to see which of the two performs better on hashing.

My initial strategy when approaching this assignment: complete the Crack class first. The methods do not look challenging as long as I consider the pseudocode demonstrated in lecture. Translating pseudocode to Java code is an essential skill and is the key to completing the implementation. Additionally, it also important to understand the hashing functions and the hash table data structure so that the implementation will not be challenging. As the assignment specifications advises, some of the implementation “should only be 6 lines of code,” which implies that the implementation is simpler than one could think. This makes sense because we learned in lecture that the hash table has a Big O Complexity of O(1) in add, remove, and find operations. Thus, the implementation of the hash tables should not contain difficult code.