**Joshua Coffey**

**Programming Project**

**CS215-01 - Data Structures**

**Fall 2018**

**Data Structures: Dictionaries and Hashing**

**Introduction**

**Dictionaries**

Dictionaries are data structures who’s entries contain two parts: a search key, and a value. The search key is what allows you to locate the desired entry. A dictionary is different from other data structures in that a dictionary organizes and identifies its entries using solely the search key, rather than some other factor like position. As such, entries can be removed or retrieved from a dictionary given only the search key.

Depending on the implementation, a dictionary may allow duplicate search keys or enforce unique search keys. For example, if you were using a dictionary to store items in a grocery store and their price, you may want to consider enforcing unique search keys since it would be quite silly for a grocery store to sell two identical items for different prices. On the flip side, if you were to use a dictionary to store people’s names and marital status, then you may want to allow duplicate search keys since many people share names with each other.

Dictionaries tend to use iterators to traverse their keys and values. This is done in our book by having separate iterators for keys and values, which allows for one to traverse keys and values independently or together in parallel depending on the situation. In the latter case, the ith key of the key iterator must correspond to the ith value of the value iterator, and in all cases, since each search key must have an associated value, the lengths of the two iterators must be the same.

**Hashing**

Hashing is a technique to determine an entry’s location in an array using only the entry’s search key, using no actual searching. This is useful because, in particularly large databases where locating data is critical, even a search time complexity as low as O(log n) (which is the best worst case efficiency an un-hashed dictionary can have for data retrieval) can still end up being too slow, whereas an ideal hashing technique can get the time complexity down to as low as O(1).

The array in which hashed data is stored is typically called a hash table. A hash function takes the search key of a given entry and converts it to an integer index in the hash table for the entry to be placed/located. Ideally, there would be enough space in the hash table to store everything you could ever want to store in it. In reality, this is rarely the case and this is where the problems lie.

The example used in our text book is storing the phone numbers and addresses of everyone in a small town whose phone numbers all begin with 555, using the numbers as a search key. A perfect hashing function (that is, one that maps each search key into a different integer that is suitable as an index to the hash table) would be to use the last 4 digits of each phone number as the index. So 555-1234 would be placed at hashTable[1234] (if hashTable is our hash table). This algorithm would work for all possible search keys 555-0000 through 555-9999, so the array hashTable would need 10,000 elements to contain every possible index (0-9999). If there are 10,000 people in the town who’s numbers are all stored in this manor, we would end up with a full hash table.

But what if there were only 500 people in the town? In that case a hashTable with 10,000 elements would seem quite excessive and would only end up being very sparsely populated. Assuming these 500 phone numbers are non-sequential, we would end up needing a very different hashing function to reduce the size of our hash table.

**Conclusion**

**References**

**\*Notes: Font must be 10pt, Times New Roman. Headings, titles, etc. should be bolded. 1” margins all around. Should be single-spaced with a setting of 1.15. Subheadings should be denote by a., b., c., etc.**