Program 3 Design

Alexander DuPree

July 30, 2018

1 Task and Purpose

Program number 3's focus is in the design, implementation and manipulation of a hash table. The hash table is required to use 'chaining' for its collision resolution technique. This data structure will be used to implement a table ADT. The table ADT will be used in the shopping application program. This assignment tasked our table to perform the following:

- Add a new item to the table
- Retrieve information on a item by entering the item's key
- Remove an item from the table
- Display all items on the table
- Display all items related to a particular website

2 Design Considerations

Design patterns from object oriented design continue to apply for this assignment. However, utilizing a hash table data structure raises new considerations. Because the hash table will be utilizing strings as the search key, we must design a hash function that can hash a string quickly and distribute data evenly through the table. To accomplish this I will be following hashing techniques described in the book. My hash funciton will take the following form:

1. Bit shift the search key, left, by N digits and cast the result into an integer. This will grow the binary representation of the search key by N digits, effectively growing the integer value by a power of 2.
2. Multiply the bit shift result by a large prime number, this mutates and grows the search key again.
3. Modulo the product by the table size, which will also be a prime number. This final operation will result in a hashed search key that represents an index on the table.

This hash function will take on this form, or something similar, to distribute data evenly. However, through experimentation it is likely that this algorithm will be modified to prevent overflows and mutate the search key efficiently.

3 Data Flow

The flow of this assignment is fairly straightforward. On application startup, the program will prompt the user to enter a table size and load in the data set located in 'shopping.txt' onto the hash table. The user will then be able to add/remove/display data based on a search key. The search key will be the name identifier for a given item. Once the user exits the applcation, the program will provide a print out of the hash table's efficiency. This printout will include the table's load factor, the length of the longest chain, the average chain length, and the chain distribution ratio.

The following is a summary of the data structures and ADT's directly related to this assignment. Other classes, such as the interface, will be used but has been summarized in other design documents and is a utility classe not directly related to this assignment.

3.1 hash table

The hash table will be the underlying data structure for the Table ADT. In this program the hash table will be specialized to hold c-strings as the key, and item objects as the value. As such the hash table will contain the following public interface

Function	Summary	Parameters	Return
insert	Hashes the search key, and inserts the item into the hased index	Read only reference to the search key and value objects	A read/write reference to the hash table, this allows chaining of methods.
erase	Hashes the search key, and removes the item with a the matching key	A valid search key object	True if a item with a matching search key was found and removed
find	Hashes the search key, returning a reference to the matched item	A valid search key object	A reference to the object with the matching search key, if found, otherwise a default object reference
clear	Loops through the table recursively clearing each chain	None	None
size	Returns the number of elements in the search table	None	The hash table keeps an internal counter of its size so this is a constant time operation
chains	Calculates the current number of chains in use	None	Returns the number of chains in use
max chains	Max number of chains is the table size	None	Returns the max number of chains
load factor	Calculates the current load factor by taking the number of items / table size	None	Returns the calculated load factor as a float
empty	Tests if the hash table is empty	None	returns true if the hash table is empty
begin	Returns an iterator to the front of the first element in the hash table	None	If the hash table is empty the iterator will be NULL, this can be checked with the iterators null() method
end	Returns an iterator to one past the rear of the list	None	This will be a NULL iterator

Table 1: hash table public interface.

3.2 item table

The item table ADT utilizes a hash table to manage a table of item objects. The table will utilize the item's name identifier as the search key. The client will be able to interact with the item table with the following public interface:

Function	Summary	Parameters	Return
add item	Adds an item object to the table	A Read only reference to the copyable the item object	True if the operation was successful
remove item	Removes the item with the matching name	A string with the name of the item	True if the operation was successful
get item	Returns the item that matches the search key	A string with the name of the item	reference to the item if found, otherwise a reference to a default item
display all	Uses the hash tables iterator methods to loop through the table displaying each item	None	None
empty	Tests if the table is empty	None	Returns true if the table is empty
size	Returns the number of elements in the table	None	Utilizes the internal counter of the hash table so this is a constant time operation

Table 2: item table public interface.

3.3 Item

The Item ADT is a utility class that stores information related to a specific commerical Item. Each Item will contain the following data:

1. Item name
2. Description
3. Color
4. Website to purchase the item

The Item ADT's public interface is fairly straightforward. The Item will contain methods for inspection and mutation of each data member and have the '«' overloaded for displaying the item in a formatted fashion.

A file, 'Shopping.txt' will contain a large data set of these Items attributes. On program startup, 'Shopping.txt' will be parsed, composing each attribute into an Item object and adding that object onto the Hash table. This will allow us to gauge the efficieny of the Hash table with different table sizes, and hash functions, given the same data set.