CSE 503 Week 4 Assignment

Alex Myers

Mon Jan 5th, 2024

The provided starter file contains a HashMap class with a private member of type HashEntry, and public methods for inserting and retrieving values. The assignment requires writing a new main function and an improved insert method to handle a sequence of integers, as well as implementing and testing a rehash method with a specified table size.

I began by defining the HashEntry class, based on what was included in the starter file. In the provided code snippet, a HashEntry class is defined with private members for key and value, with the key being the hash value and value being the value being stored. Additionally, public member functions getKey and getValue are implemented to retrieve the key and value, respectively. The HashMap class is then introduced with a private member table declared as an array of pointers to HashEntry. The class also includes private members’ currentSize to track the number of entries in the array and tableSize to store the size of the array.

A white background with black and white clouds

Description automatically generatedA computer code with colorful text

Description automatically generated with medium confidence

Next, I updated both the insert method to not only use my newly defined HashEntry class but also implement linear probing as a collision resolution strategy. In the revised public get method, the function begins by obtaining the initial hash value, calculated as the value itself mod the tableSize. The code then loops through the table, handling collisions using linear probing. The while loop checks whether the entry at the calculated hash is NULL and if the key (aka hash value) already exists in the array. If a collision occurs, it increments the hash value by 1 and recalculates the hash until an empty slot or a matching value is found. If the loop finds an empty location in the array (marked as NULL), it returns the hash value, indicating a potential location where a new hash entry could be inserted. If the loop concludes without finding a free location or confirming the presence of the value in the array, -1 is returned. This modified get method facilitates linear probing and allows the insert function to use the hash value in its logic. A computer code with text

Description automatically generatedBelow A black text on a white background

Description automatically generatedis a before and after picture of the get method.

A white background with black text

Description automatically generated Next, I applied the refined get method within the insert method to determine the hash value of the value being inserted. If the hash value is anything other than -1 and the location in the array is NULL, the value is added to a new hash entry within the array. This is accomplished by invoking the HashEntry constructor and placing the new entry in the array at the calculated hash value location, then increasing the currentSize of the array by 1. If the location is not NULL the value is updated. Below is a before and after picture of the insert method.

A computer screen shot of a code

Description automatically generated

A screenshot of a computer

Description automatically generatedFinally, to meet the first requirement of the assignment, I updated the main function to incorporate the values {138, 99, 16, 134, 42, 0, 6, 9, 4, 53, 47, 66} into the hash map. This is the output with the table size set to 17.

A screen shot of a computer code

Description automatically generatedFor the second half of the assignment the assignment I implemented a hash function which can be seen below. The rehash method is responsible for resizing and rehashing the hash table when the current size exceeds half the table size. This logic can be found in the insert method where the rehash method is called if the number of entries over the size of the table exceeds half. If it does the rehash function is called. It first stores the old table size and table in temporary variables, then calculates a new table size using the nextPrime function, and then allocates a new table of HashEntry pointers. The current size is reset to 0, and the new table is initialized with NULL entries. It then iterates through the table, inserts each non-null entry into the new table using the insert method, and deletes the old entry. Finally, it frees the memory where the old table was allocated. This function all in all increases the table size and then redistributes the entries within it. An example of what an original table and a rehashed table look like with a table size of 7 can be seen below.

A computer code with text

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedThis assignment shows how resizing and redistributing values in a hash table is one of several ways of dealing with collisions along with linear probing and other techniques. This is a good example of using these techniques while demonstrating how to build out a hash table.