Gabriel Brehm

CS260

Dr. Breeann Flesch

15 June 2018

Lab 5 Report

For this lab, I wrote the appropriate functions to finish the hash set class. This class uses the doubly linked list class that was written earlier in the term.

The first function I wrote was the inset function. This function took in a T type data and returned true if the data was inserted into the hash table, and false if was not. First, a bool data type is created and initially set to false. Then an index is created and set to the result of the hash function that was provided. If the data is in the hash table already, based on the find function described later, then the function just skips to the end. If the data is not in the table, and the index of the table is NULL, make a new doubly linked list, put the data in it, and set the pointer in the table to it. Then increment the size and set the bool value to true. Otherwise, just add the data to the doubly linked list, increment the size and set the result to true. Then the result is returned.

```
template<typename T>
bool HashSet<T>::insert(T data)
{
    bool result = false;
    double index = hashFunction(data);
    if(find(data) == true)
    {
    }
    else if(hashTable[static_cast<int>(index)] == NULL)
    {
        DLList<T>* list = new DLList<T>();
        list->add(data);
        hashTable[static_cast<int>(index)] = list;
        size++;
        result = true;
    }
    else
    {
        hashTable[static_cast<int>(index)]->add(data);
        size++;
        result = true;
    }
    return result;
}
```

The second function I wrote was the clear function. This one has no return type and no parameters. It walks through the hash table using a for loop, and at each index that is not NULL, it runs the clear function form the doubly linked list and then sets the index to NULL. Once the loop is done, the size is set to 0.

```
template<typename T>
void HashSet<T>::clear()
{
    for(int i = 0; i < tableSize; i++)
    {
        if(hashTable[i] != NULL)
        {
            hashTable[i]->clear();
            hashTable[i] = NULL;
        }
    }
    size = 0;
}
```

The third function is the find function, which returns a bool and takes in a T type data as a parameter. It creates a bool and sets it to false, hashes the data, and checks if the index in the table is NULL. If it is not NULL, the result is set to the result of the doubly linked list contains function. The result is then returned.

```
template<typename T>
bool HashSet<T>::find(T data)
{
    bool result = false;
    double index = hashFunction(data);
    if(hashTable[static_cast<int>(index)] != NULL)
    {
        result = hashTable[static_cast<int>(index)]->contains(data);
    }
    return result;
}
```

The fourth function is the is empty function, which returns a bool but takes in no parameters. It just checks if the size is 0 or not and returns the appropriate result.

```
template<typename T>
bool HashSet<T>::isEmpty()
{
    if(size == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
```

```
}
```

The fifth function is the erase function which returns a bool and takes in a T type data as a parameter. It hashes the data, sets a bool value to the result of the find function. If the result is true, the doubly linked list remove function is called on the index, and the size is decremented. The result is then returned.

```
template<typename T>
bool HashSet<T>::erase(T data)
{
    double index = hashFunction(data);
    bool result = find(data);
    if(result == true)
    {
        hashTable[static_cast<int>(index)]->remove(data);
        size--;
    }
    return result;
}
```

The get size function is the get size function. Enough said.

Work Cited

Peer Collaboration:

I worked with Jacob Malmstadt, Megan T, Walker M, Mike D on this project.

Websites:

http://www.cplusplus.com/

https://stackoverflow.com

http://en.cppreference.com/w/