Assignment: Project Hash Table

Zinedin Bautista

Professor Alex Lai

University of Advancing Technology

Data Structures and Algorithms

July 20, 2020

Assignment: Project Hash Table

Introduction: A hash table is a data structure that is based on mapping keys so some values using the hash function which is calculated using the module (%). A problem with the hash table data structure is that it might encounter a collision which is when a data value is going to enter a node that already has a data value in that node. There are two solutions to avoid this collision, the first solution is by either double hashing which is creating a second hash function. The other solution is by chain hashing which would make each cell in the hash table point to a linked list of records. Also another important point of creating a hash table is that the size of the hash table should be a prime number. The reason for this is to avoid clustering values which as mentioned before is an issue that hash tables encounter unless you use the two collision resolving techniques mentioned before.

Programmer's Guide:

Project Hash Table.h:

(This file contains the size of the hash table)

Class PhoneBook: This class is in charge of representing the entry information which are the names and phone numbers for the phone book. All attributes are placed within the public access specifier. The class contains the string variable name which will store the name correlated to the phone number and will also represent the key in the hash table. There is another string variable called phoneNumber which will store the phone number information. There's a PhoneBook pointer representing the next node that has the same key. And finally there is a constructor for PhoneBook which will initialize a node with the parameters string name and string phoneNumber.Within the constructor there are three this points, the first this pointer points to the name object and holds its address and sets it to the name string. The second this pointer points to the phoneNumber object and holds that address and sets it to the phoneNumber string. And finally there is a this pointer that will point to the next node and hold that address while initializing the next node as NULL.

<u>Class PhoneBookHashTable:</u> This class is in charge of storing all the phone and phone number entries into itself. All the attributes will be placed into the public access specifier and under that public access specifier. A pointer-to-pointer that will be used to represent the hash table. Then there is the constructor for PhoneBookHashTable which will initialize the PhoneBookHashTable to its initial state. Within the constructor it will create the initial hash table which has a size of eleven. There's a for loop that will keep running until i is no longer less than the table size while incrementing by one. And the line of code within that for loop will initialize the hash table to NULL. After the constructor is the deconstructor for the PhoneBookHashTable which has the responsibility of deleting the memory space that is taken up by the hash table. Within the deconstructor is a for loop that will keep running until it is no longer less than the table size while incrementing by one. And within that for loop there is a PhoneBook pointer representing the entry that will be equal to the hash table. There's a while loop that will execute the code below when the entry of the hash table does not equal NULL. In that while loop is a PhoneBook pointer that will represent the previous node and will be equal to the entry of the hash table. Set the entry node to have access to the next node and then finally for the while loop deallocate the memory space on the previous node and call the destructor for the single object. And then the final thing within the deconstructor it will deallocate the memory space on the hashTable and call the destructor for the array object. The final segment of the

PhoneBookHashTable class are the function prototypes for hashFunction, insertHash, and retrieveHash.

Project Hash Table.cpp:

hashFunction(): This function will obtain the key value by adding the ASCII values of the characters in the string name and it will also obtain the hash value from the key. Within the function definition is an int variable for the key and the key is the sum of ASCII of all the characters present in the string name. There is a for loop that will keep running until it is no longer less than the length of the name by incrementing one. And within the for loop is key += name.at(i) which is like saying key is equal to key plus the name.at(i). The at() method will extract characters from the name string. And then finally outside of the for loop the hashFunction function definition will return the value of the key module TABLE_SIZE using a module (%) which will return the remainder.

insertHash(): This function will allow user to insert a name and phone number into the phone book.Within the function definition it will assure if the name given by the user already has a phone number linked to it then it will add the new phone number at the end in the list.The first line of code in the function definition it will set the hash value to the value of the string name from the hashFunction. After that there is a PhoneBook pointer representing the previous node and will be equal to NULL and another PhoneBook pointer representing the entry and setting it to the hash value from the hash table. After that is a while loop that will run the code below when the entry does not equal NULL. The purpose of the while loop is to first find the node after which this number needs to be inserted. Within the while loop it will set the previous node and the entry equal to each other and give entry access to the next node. After the while loop is a if, else statement. The if statement will run the nested if,else statement below when the hash entry is equal to NULL. Within the if statement the first thing it runs before the nested if, else statement it will insert the phone number at the proper position and initialize and create a new entry into the phone book. Now into the nested if, else statement the if statement will run the code under when the previous node is equal to NULL. Within the if statement it will set the entry into the hash value within the hash table. Then if the else statement is to be executed then it will give the previous node to the next node and set to the entry. The purpose of the else statement that is for the first if statement is for when the name given from the user does not have a number linked hence link this number with the given name. And then finally there are two cout statements to inform the user that the name they added has been added to the phone book and a line of dashes acting as a dividing line for menu organization.

retrieveHash(): This function will allow the user to retrieve back phone number information depending on the name the user inputted. There is a boolean variable checking if the name provided by the user has a phone number linked to it. And another int variable for hashValue which is set to the value of the string name from the hashFunction. There is a PhoneBook pointer representing the entry and setting it to the hash value from the hash table. After that is a while loop that will run the code below when the entry does not equal NULL. In the while loop is a if statement to see if name the user inputted has multiple numbers linked to it then display all the numbers. If the condition of the if statement is met then it will display all the phone numbers and set the boolean value of occupied to true. Outside of the if statement but still in the while loop it will also set the entry to whatever data the entry gets from accessing the next node. Then finally outside of the while loop is another if statement to see if the occupied boolean variable is false then the name does not have any number linked to it. If the condition is met then return the value of negative one.

Project Hash Table Main.cpp:

The main file starts off by initializing and creating a hash table by naming it myHashTable. There are two string variables that will store the name and phone number information that the user inputs. There is also the int variable called choice which will collect user input on deciding what function the user wants to execute from the menu. The menu for the hash table phone book will inform the user on what numbers they would need to input and how to exit out of the program. The hash table phone book is run by a do while loop which will keep running the hash table phone book with all of its functionality until the user inputs three into the menu which will exit out of the program. If the user inputs one into the menu then it will execute the insertHash function which will then ask the user to input a name and phone number. Then the program will store that user input into the phone book. And finally if the user types two into the menu then it will execute the retrieveHash function which will ask the user to enter the name they want to search. Once the name has been inputted the case will check if that name exists in the phone book. If the name does not exist then the program will inform the user that there are no phone number(s) associated with the name they inputted. If the name does exist it will display all the phone numbers associated with the name.

Analysis:

Insert:

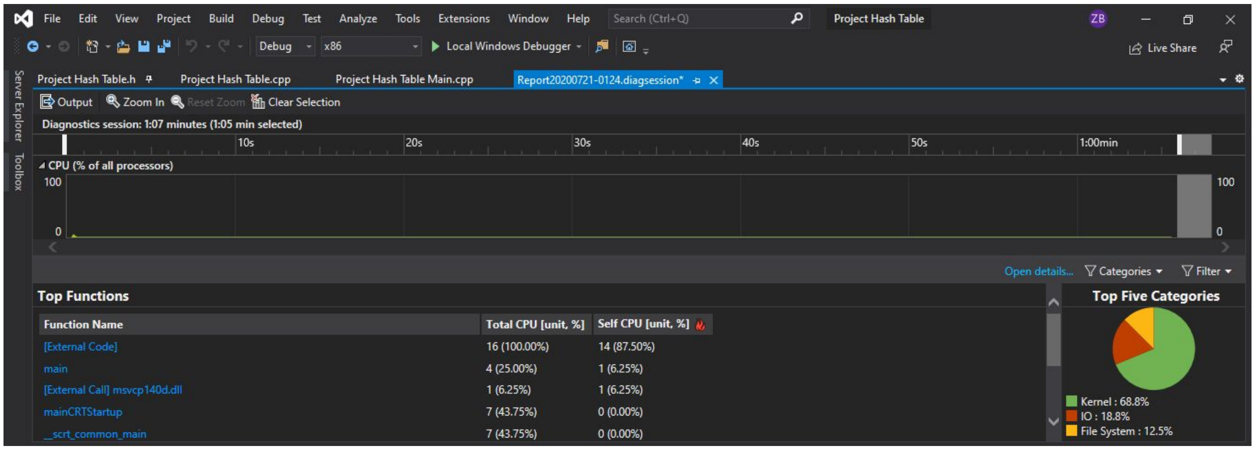
- Average Case - O(1)
- Worst Case - O(n)

Search:

- Average Case - O(1)
- Worst Case - O(n)

Space:

- Average Case - O(n)
- Worst Case - O(n)

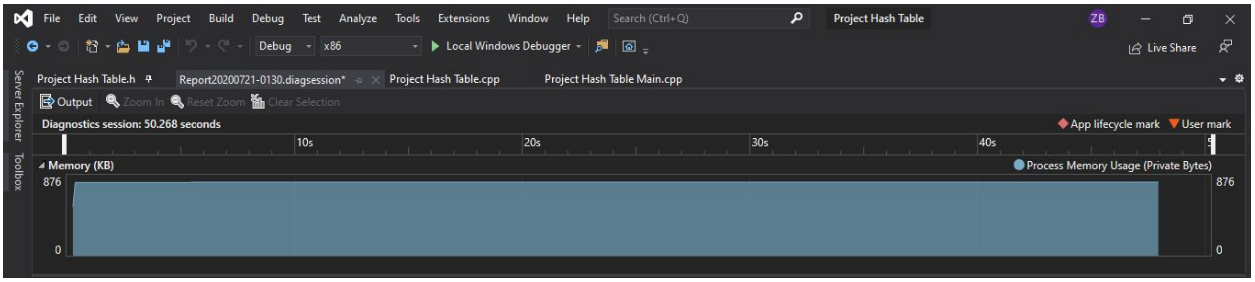


Diagnostics session: 1:07 minutes (1:05 min selected)

Top Functions

Function Name	Total CPU [unit, %]	Self CPU [unit, %]
[External Code]	16 (100.00%)	14 (87.50%)
main	4 (25.00%)	1 (6.25%)
[External Call] msvcp140d.dll	1 (6.25%)	1 (6.25%)
mainCRTStartup	7 (43.75%)	0 (0.00%)
__scrt_common_main	7 (43.75%)	0 (0.00%)

🟩 Kernel : 68.8%
🟥 IO : 18.8%
🟨 File System : 12.5%



Diagnostics session: 50.268 seconds