Report - assignment 5

Camilla K. Birkelund IDATT2101 September 29th, 2022

Assignment description

The first part of the assignment was to implement a HashTable that could store strings. The program should be able to read names from a file and save them to the table, handle collisions with linked lists, find names using the table and print results. Additionally, the number of collisions per person (name) should be less than 0.4.

The second part of the assignment was to implement a HashTable with enough space for 10 million integers, where collisions should be handled using double hashing. The time used to insert the numbers into the HashTable should then be compared to the time used by c++ own methods (in this case, unordered_map was used). Lastly, the program should include a print with both times, the load factor for the HashTable implementation and number of collisions.

Implementation

Implementation of the first task

The source code for the first task can be viewed in “task1.cpp”. To solve the first task, a HashTable class was implemented to handle all methods and variables related to the HashTable. The class contains methods to create a hash from a string, insert into the HashTable, to check if the HashTable contains a certain name, get the load factor and calculate the collisions per element in the table. Outside of the class, a method was implemented to read from a file to allow the program to read from the file given in the task description using the library “fstream”.

Implementation of the second task

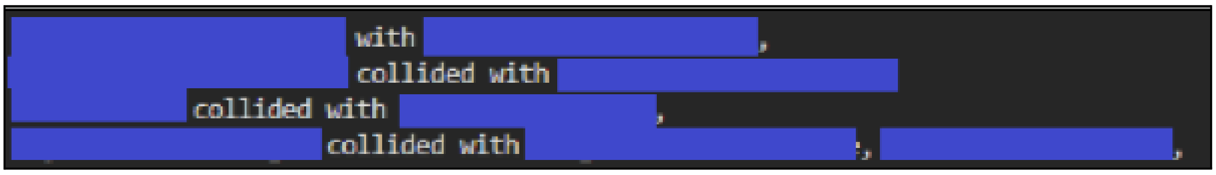
The source code for the second task can be viewed in “task2.cpp”. A lot of code concerning the HashTable was reused from the first task, except the hash-methods were split into two parts. The first hash-method (hash_first) was to suggest a position for the number. If the position is vacant, the number is placed there. If there is already a number in the suggested position, the next hash-method will be used (hash_next), which is described in the insert_to_table method. The next hash-method suggests a “jump” to a new position, where the length of the jump is decided by the value of (number % (the tables capacity)) +1.

The library chrono was used to measure the time used by the HashTable implementation and the predefined method in c++. All values, vectors and tables were defined before starting the time measurements to ensure that the resulting times represented both methods as accurately as possible.

Result

Result for the first tasks implementation

The first task's implementation is a hashtable with string keys that handles collisions by using double linked lists, as described in the assignment. The wanted results from the print from the program was a list of all names in the file, including the collisions between the names, a check whether or not a certain name is in the table, the load factor and collisions per insertion, which should be below 0.4, but higher than 0.



```
[redacted] with [redacted],
[redacted] collided with [redacted]
[redacted] collided with [redacted],
[redacted] collided with [redacted], [redacted],
```

Figure 1: A small portion of the first part of the print, which is all the names from the file including the collisions that occurred.

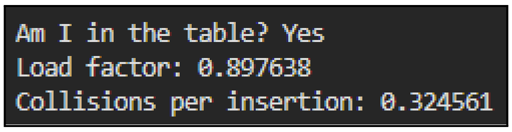

```
Am I in the table? Yes
Load factor: 0.897638
Collisions per insertion: 0.324561
```

Figure 2: Second part of the print, which is the result of the check of whether or not a name is in the list (in this case it checked for “Camilla Kristiansen Birkelund”), the load factor, and the collisions per insertion, which is below 0.4.

In the process of writing the program, checks were also performed to confirm that names like “Ole” and “Leo” would not get the same values, as well as checks to see that the amount of names in the file were the same amount as the names in the HashTable when reading from the file.

Result for the second tasks implementation

The second task’s implementation is a HashTable of integers, that handles collisions using double hashing as described in the assignment. The time used by the HashTable implementation and the methods from the unordered_map library in c++ were compared while inserting 10 000 000 numbers into the tables.

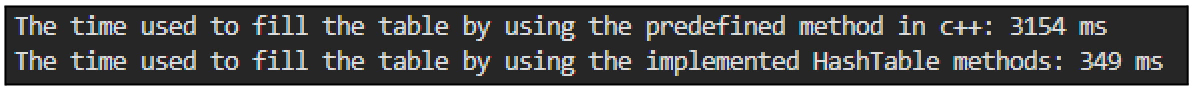

```
The time used to fill the table by using the predefined method in c++: 3154 ms
The time used to fill the table by using the implemented HashTable methods: 349 ms
```

Figure 3: Result of the times used by the HashTable implementation and the predefined c++ methods. The HashTable implementation is faster, although both methods are fairly quick to fill the tables with numbers.

Another part of the assignment description was to include the number of collisions that occured in the HashTable, as well as the load factor. The number of collisions is about 9 000 000, and the load factor is about 0.77 which is an acceptable value considering the requirement of the load factor being 0.75 or higher.

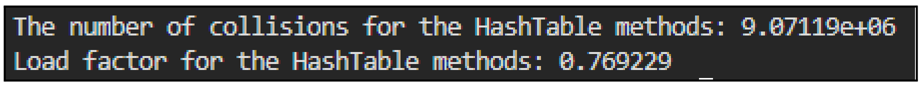

```
The number of collisions for the HashTable methods: 9.07119e+06
Load factor for the HashTable methods: 0.769229  _
```

Figure 4: The result of the HashTable implementation’s number of collisions and load factor. Both values are acceptable values.