Practical Assignment

BM40A1500 Data Structures and Algorithms

1. Implementing the Hash Table

1.1 Structure of the hash table

This hash table uses an open hashing structure using linked lists with key-value pairs. The collisions of records are resolved by assigning a linked list to each of the tables slots, which allow multiple values to be stored at the same key value calculated by the hash function. Each index with a value contains the key to the index, inserted value and an address for the next node in the linked list, none if empty. The hash table stores values in each linked list in the order they are inserted.

1.2 Hash function

The hash function used in this table is a string folding function with two folding characters at a time. The hash function takes the insertable value and multiplies every second character's ASCII value with 256 and sums them up. Every other character's ASCII value is just added to the sum. This hash function usually only takes strings as input, so every input is converted to string before calculating the hash key.

The reason this hash function was selected is because it was easy to implement and it yields relatively well distributed results.

1.3 Methods

The hash table has methods for:

1. Calculating the hash key
    - Takes a value and converts it to a string. Cuts the string to two character chunks and calculates the hash key using methods described above.
2. Inserting a new value to the hash table
    - Gets the hash key for the inserted value and saves the value to the given hash key index slot if it is empty. Otherwise iterates through the linked list in the calculated hash key index and saves the value at the end of the list.
3. Searching for an exact value
    - Gets the hash key for the searchable value and checks if the hash key index slot is empty or not. Proceeds to iterate through the list at calculated index and returns the value that was searched.

4. Removing a value from the hash table
   - Gets the hash key for the removable value and checks if the hash key index slot is empty. Iterates through the linked list at given index and removes the value by removing the address for value.

2. Testing and Analyzing the Hash Table

2.1 Running time analysis of the hash table

- What is the running time of adding a new value in your hash table and why?

  Running time of adding a new value to the hash table is Θ(n) because there is a chance that all nodes will be attached to the same linked list due to collision.

- What is the running time of finding a new value in your hash table and why?

  Running time of finding a new value in the hash table is also Θ(n) because the same single linked list can have all the values and thus they need to be checked one by one.

- What is the running time of removing a new value in your hash table and why?

  Removing a new value from the hash table is Θ(n) also for the same reason that the value to remove is the last value of the linked list at index which contains all the nodes.

Of course these running times are worst-case scenarios and this can be solved with better hashing functions.

3. The Pressure Test

Table 1. Results of the pressure test

Step	**Time (s)**
Initializing the hash table	0,039
Adding the words	2,102
Finding the common words	1,023

3.1 Comparison of the data structures

List was a faster option to add the words from the file because the words will be added to the end of the list and we don’t need to calculate the hash key for using a list or resolve any of the possible collisions.

Search was however a whole different story and that is exclusively due to hashing. When the search was executed using a list it took almost three and a half minutes to find the colliding words. That is because searching through a regular list has to be done once for every word where as for hash tables the search takes the time to calculate the hash key and iterate through the indexes.

3.2 Further improvements

By increasing the size of the hash table, the insertion and searching for values was faster but not by a lot. That is because of collisions being less common.

List of references

OpenDSA Project

Antti Laaksonen, **Tietorakenteet ja algoritmit**, 2021

GeeksforGeeks.org