Practical Assignment

BM40A1500 Data Structures and Algorithms

1. Implementing the Hash Table

1.1 Structure of the hash table

I used the same table that we used in Linear Probing. For the linked list I used the code from the exercise without any operations except get/set for the list nodes. So, the element on the list also has an element next in the node.

1.2 Hash function

First, I used the linear hashing for the hash function, but it created too many collisions, so the linked lists were long, and it took more time to go through the hash table. My improvement to the hash_3_1.py was to change the linear hashing to polynomial hashing.

1.3 Methods

getEmbryo: returns the current element in the list

getNext: returns the next element in the list

setEmbryo: sets given data to the current element in the list

setNext: sets given data to the next element in the list

insert: inserts given data to the hashed location in the first available element in the linked list

delete: deletes given data by hashing the location and finding the data on the linked list, deletes it by changing the route and skipping the deleted list node in the linked list.

search: searches the given data by hashing and checks if the data is in the linked list of this location, if found returns True, else return False.

print: prints the hash table by going through every location in the hash table and prints every linked list element on that location. Also prints “ | ” between hash table locations so it’s easier to understand.

hashing: hashing function hashes the given data by polynomial hashing and returns the location in the hash table

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?

The running time is Θ(n), because there aren't any nested loops in this method

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

The running time is Θ(n), because there aren't any nested loops in this method

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

The running time is Θ(n), because there aren't any nested loops in this method.

> Where n is the input size. All methods have Θ(n) running time because the size of the data effects the hashing() function's running time. Also going through the linked list doesn't always have the same running time. Print() method has running time of Θ(n^2) because there are two nested loops.

3. The Pressure Test

Table 1. Results of the pressure test.

Step	**Time (s)**
Initializing the hash table	0.008
Adding the words	5.681
Finding the common words	2.458

3.1 Comparison of the data structures

> Initializing the list: 0.0 seconds
>
> Adding the words: 0.197 seconds
>
> Finding the common words: 33.765 minutes

Adding words to a list is faster because we only have to add the given data to the end of the list by using list.append() -function. We can see that finding the words from the list takes way longer than the hash table, because we must go through the whole list every time the data isn't in the list.

3.2 Further improvements

Making the hash table bigger makes adding the words and finding the words faster, because there is less collision, so the linked lists are shorter.

I also changed linear hashing to polynomial hashing, and this made a big difference to the running time. I use 11111 in the polynomial hashing.

List of references

Add here the references and source that you used.