Information:

NAME: Abdusselam koç

SECTION: cmpe 343- section 2

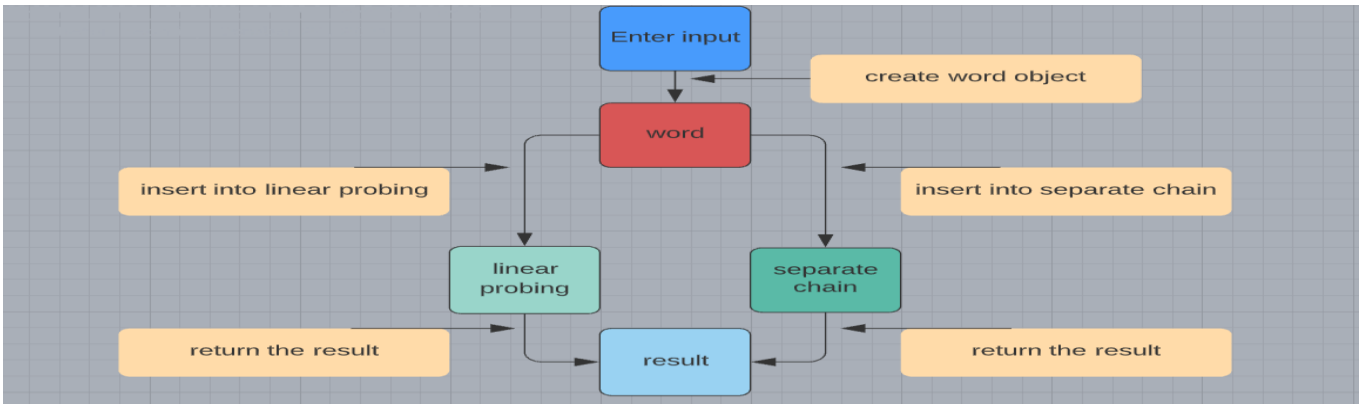
ASSİGNMENT NUMBER: homework 1

Problem Statement and Code Design:

Problem:

In this task the problem is creating a basic dictionary application that calculates the most used three words and their numbers by implementing hash table. The application must store all the distinct words and their number of occurrences. Hash table algorithm must be implemented in both separate chaining and linear approach. The program should start with the table of size 16 in both algorithms. We need a word class to keep the word, word's hashcode and the number of occurrences. In separate chain class and linear probing, we only need two main function which puts keys according to algorithm and return keys. The other functions are helper functions. At the end the program should return the indexes for both tables therefore we are going to need and index for Node class in separate chain, but we are not going need it in linear probing because in linear probing we can easily reach array index.

Structure chart:



Implementation and functionality:

Word class:

Firstly, we need to create a word class to represent the dictionary.

The word class must have three attributes: A string that keeps the word, an integer to keep number of occurrences and an integer to keep the hash code. In this way I will be able to call each attribute whenever needed. For example, we can change number of occurrences when an equality happens. Also, this class has a function named hashCode to calculate hash code and assign it to hashcode attribute.

Hash table implementation for separate chaining:

We need to create a class for separate chain implementation named hashForSC. This class has three attributes: an integer to keep Number of the Nodes, an integer to keep size of the Node array and a Node array to keep the nodes that been created. This class will have A node class to create Node objects for chains.

Node class:

Node class is the most important part for separate chaining. It has four attributes: an integer to keep the value of the key, and Word object to keep the key, an integer to keep the index of the Node in chain and a Node that keeps the next Node address.

Functions:

This class has 6 methods to create the program.

hash: this method simply returns a simplified hashcode according to size of the array.

Check: it takes a word as an input and checks for equality to the word in the chained array. It helps put method.

CreateNode: this method takes a word object as input and creates nodes whenever there is a need for a new node and puts it in the correct chain in the array.

Put: it takes a word object as an input and checks whether the chain has the word or not, then decides to whether the program needs a new node or not. If there is need to create a new chain calls CreateNode method.

Print: searches all array and Nodes and returns the most used three words and their number of occurrences with their Node index.

Expand: expands the array when Number of Nodes increased.

Hash table implementation for linear probing:

We need to create a class for hash table implementation in linear probing named hashFL.

This class has 3 attributes: an integer to keep the number of the words that are in the array, an integer to keeps the size of array and an array to keep the word objects.

Functions:

This class has 5 methods to create the program:

hash: this method simply returns a simplified hashcode according to size of the array.

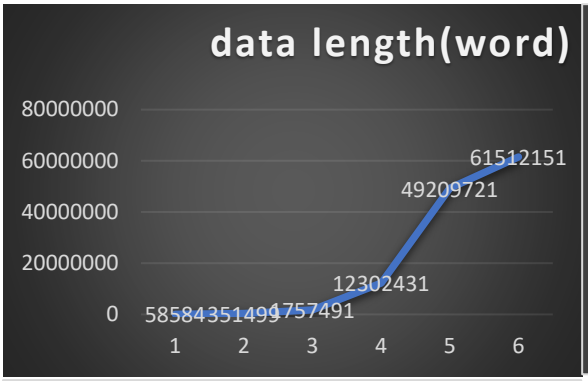
Put: It takes a word object as an input and checks whether is it already in the array or not if it is in the array increases the counter of the same object in the array if not in the array puts it into array.

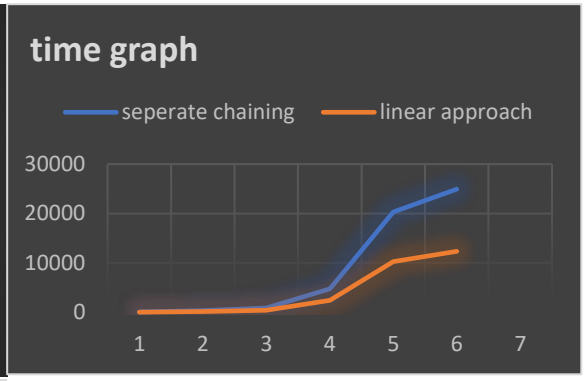
Rehash: rehashes all values in the array in case of resizing the array.

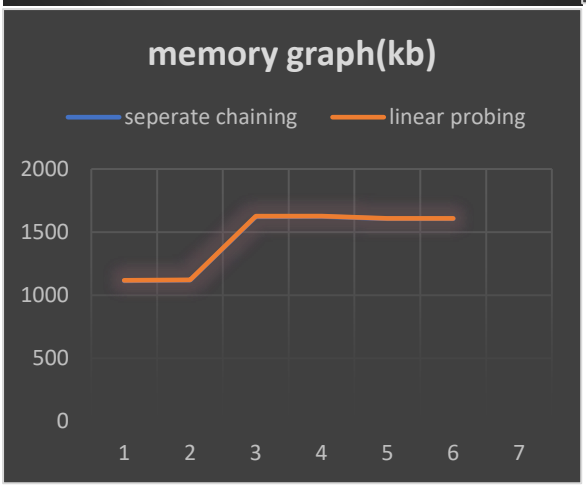
Expand: expands the array when array become half full.

Print: searches the word array and returns the most three most occurred word, their occurrence numbers, and their indexes.

Performance Comparison:







when data length increases the time and memory usage is also increase. According to my test results the memory usage is a bit higher in linear probing since the difference is too small it doesn't show in the graph. According to my test result the time consuming of separate chain implementation is higher than linear probing. The time consuming in separate chain is higher because in separate chain on searches we must search for array and nodes therefore it takes more time than linear probing also, we create new Node object during the insertion to table it also takes more time. The memory is less in separate chain because whether we full the array or not, the array occupies memory but in Nodes or LinkedList when the node is not created it doesn't occupy any memory. Since we always have a bigger array in linear probing it is normal to have a bit higher memory usage.

Testing:

I created two testing class one is named test class and one is named Runner class in part two. The test class created for different inputs and Runner class created for time and memory measurement. While testing the code you must change directory in Runner class to measure different inputs. İt is because the data that i used for testing was too big to keep it in IDE. The test and Runner class working as expected and both gives correct results for my input statements.

Final Assessments:

- The report section was my biggest trouble point in this assignment because it really a huge time-consuming part and I know we need it, but it just takes too much time.
- The most challenging part for me was put method in both algorithms it just wasn't working correctly or working correctly but didn't do the correct counting but at the end I dealt with it.
- I like to back to coding algorithms again and I learned hash tables how have more control on Nodes their connections.