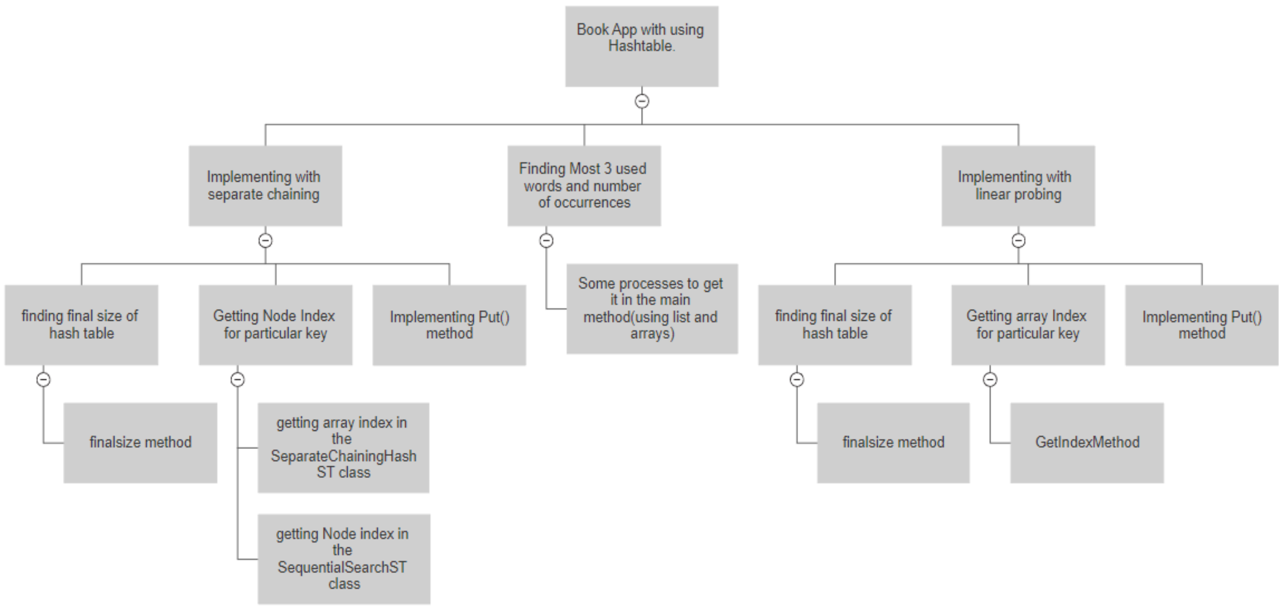
Cmpe 343 HW 1 Report

4865115490,
Fatih Mehmet Yiğitel,
Section 02,
HW1

Problem Statement and Code Design (%30)
Small Book App using Hashtable.


Book App with using Hashtable.
Implementing with separate chaining
Finding Most 3 used words and number of occurrences
Implementing with linear probing
finding final size of hash table
Getting Node Index for particular key
Implementing Put() method
Some processes to get it in the main method(using list and arrays)
finding final size of hash table
Getting array Index for particular key
Implementing Put() method
finalsize method
getting array index in the SeparateChainingHashST class
getting Node index in the SequentialSearchST class
finalsize method
GetIndexMethod

Implementing with two approaches: Linear Probing and Separate Chaining.
To do that I used Sedgewick books. I implemented codes from the book.
I rearranged the implementations according to homework's requirements.
I rearrange the code for linear probing according to if half of the table is full make the size double rule in the Hw.
I rearrange the code in separate chaining according to (N/M >= 8) rule in the Hw.
With some methods I found that most 3 used words and their number of occurrences.
I found the final table size for a hash table with separate chaining and linear probing using final size methods.
I found the Key's Indexes that from given Key value with GetIndex methods for separate chaining and linear probing.
I add finalsize, getIndex methods for both Linear Probing and separate chaining solutions.

Implementation, Functionality, Performance Comparison (%40):
Linear probing and functions:
In linear probing we have int n for number of key value pairs, int m for size of linear probing table, Arrays for Key and Value.
Our INIT_CAPACİTY is 16.

int finalsize();
This method returns the final size of the linear probing table.

int size();
This method returns the number of key value pair of the linear probing table.

boolean isEmpty();
This method returns the true or false. If the size of the table is equal to 0 it returns true.

boolean contains(Key key);
This method returns the true or false. If the table contains the given key it returns true.
It includes a get method that I will explain later.

int hashTextbook(Key key);
It computes the hash value of the key. First computes the hashcode and with the dividing number of capacity of the table (int m) reaches the hash value of the key. It determines the key's index in the table.

void resize(int capacity);
It resize the table becomes half full. We create a new hashtable named temp with new capacity. With the for loop we transfer the Key and Value parameters to our new table using put method. And after that we assign temp.keys to keys and temp.values to values. We will use this method in the put method.

void put(Key key, Value value);
At first we check the number of key pair values bigger or equal to capacity/2. If it is true we use resize method. After that using hash value of the key we search a proper index to put in the table to avoid Index mistake at the end of the table instead of using normal search we use (i = (i+1) %m). it provides after we reach the end of the table it goes beginning of the table and put the key proper index. we look until there is no null value in the key array. After the finding index we put the value in the array and increment the int n (number of the key value pair).

```
keys[i] = key;
vals[i] = val;
        n++;
```

Value get(Key key);
We search table just like put method for getting the particular Key's value. With for loop we search the table and when key in the array is equal to our parameter key. We return Value array index that refer to our key array index (vals[i])

```
if (keys[i].equals(key))
```

int getIndex(Key key);
This method returns the index of the particular key in the hash table. We assign the keys hashcode value an temporary Integer. Using While loop we look the keys int he array. Is it equal to our key or not.

```
while (!key.equals(keys[i]))
```

If it is not equal to our key we increment the value of the hash.

```
i = (i + 1) % m;
```

Loop will continue until we find the key. After finding it returns the Index.

Separate chaining functions:
In the separate chaining I use Sequential Search class for the nodes that related to array. So it will be some additional function from the that Sequential Search.

resize(int chains);
We resize the hashtable to have more number of chains and rehash all the keys.
At first we create new SeparateChainingSt named temp and with for loop we put the our all keys in to the our new hashtable. And we assign the our new capacity and size of the key pair value our old variable.

```
 temp.put(key, st[i].get(key));
```

int hashTextbook(Key key);
It computes the hash value of the key. First computes the hashcode and with dividing number of capacity of the table (int m) reaches the hash value of the key. It determine key's index in the table.

int finalsize();
This method returns the final size of the linear probing table.

int size();
This method returns the number of key value pair of linear probing table.

boolean contains(Key key);
This method returns the true or false. If table contain the given key it returns true.
It includes get method that I explain later.

Value get(Key key);
We assign our key hash value as an index and in the using st[i](our array) we reach our key's value.

```
 return st[i].get(key);
```

void put(Key key, Value value);
At the first we check if the key or value null or not.
After that we check the n (number of the chains) is bigger than 8*m(m is array capacity).
if it is true we use the resize method. We do it for better performance of separate chaining.
We check every the array st[i] and if it does not contain the key we put the key and value this proper index.

```
if (!st[i].contains(key)) n++;
```

int GetNodeIndex(Key key);
In this method we get the node index for particular key. At first we get hash value of key and check the value in the array (st[i]). If it contains the key we use I understood we are in the correct index and we called getIndex method from the class SequenitalSearchST.

```
if (st[i].contains(key)){
```

int GetIndex(Key key);
This method came from the SequentialSearchST Class and it represents chains in the SeparateChainingST class. In this method we create counter and get the search the all the nodes in the linked list and if our key are equals the some key in the linked list we stop the counter and so we can find the index number that key we search for.

```
if (key.equals(x.key))
```

Getting the most 3 used world
In the main method using Collectors I collect them in the Map according to worlds counting and to get list of most 3 used worlds I use reversOrder and limit methods from the java.util.comparator library. And I take them different arrays according to String and their number of occurrences as Integer. After that I use the arrays to display these informations in the main method.

Performance of Separate Chaining and Linear Probing
I observe that linear probing is faster than separate chaining. Because these two has different collision solutions. In the linear probing if there is no collision we can find the proper index for key. But in the separate chaining after the calculating hash method we need to get the first index in the linked list. We have to reach pointer and after that we can put the our key. It consist of two operation.
So we can say, linear probing is faster than separate chaining.

Testing (%15)

To test the program I use different input files.All the test gives the most used words correctly.
I thought the Index place for linear probing and separate chaining is correct
I test it for resize methods also. Some specific values make the hash tables resized correctly.
I couldn't find any mistake except if there is no world in the input file it gives java.lang.NullPointerException and if there is not enough world to get most 3 used worlds I take the same error.
For the memory part we know the separate chaining has Linked lists and it uses more memory than linear probing because it use just one array to keep the values.

Final Assessments (%10)
What were the trouble points in completing this assignment?
At first I couldn't find the index number for the given key in the separate chaining.
I missed some simple steps and trying to find them was tiring.

Which parts were the most challenging for you?
Testing part was the most challenging because I had to use some different inputs and checking results all the time was hard.

What did you like about the assignment? What did you learn from it?
I liked making non abstract , basic app. It was useful. I learned the distinctions between separate chaining and linear probing hash tables and their performance differences.