GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 6 Report

Muhammed Sinan Pehlivanoğlu
1901042664

Q12.A

Coalescde hashing method is an hybrid method that use open adressing and similar to chaining technique. It uses the technique of Open Addressing(linear probing) to find first empty place . It uses the chainig technique to link the colliding elements to each other through pointers without create a new node.

Advantages:

Coalesced hashing technique avoids the effects of primary and secondary clustering. If the chains are short , it can perform efficent search operation.

İt does not use waste of memory like other chaining technique. It does not make much effort to find new empty space like other open adressing methods.

Disadvantages:

If the size and chains are long, it becames unefficent. Rehasing and deletion is expensive .

Q1.2.B

Double hashing is a open adressing techniuqe that used to resolve collisions by using a secondary hash of the key as an offset when a collision occurs. It minimizes repeated collisions and the effects of clustering. It performs hash function to find free slot.

Advantages:

Double hashing offers better resistance against clustering due to dobule hash function. Double functions enable double hashing to perform a more uniform distribution of keys in contrast to the chainig method.

Disadvantages:

As the table fills up the performance degrades faster than other open adressing method.

1. SYSTEM REQUIREMENTS

Operating System need to jdk and jre for start the program
Stack capacity should not be less.
You need to enough space to store data's of program according to the how many you have objects.
Initial Capacity of Binary Search Tree array used by Hash Table is 11.
Initial Capacity of Entry array is 101 used by Hybrid Hash Table.
The capacity will be incremented almost twice with respect to put element to hash table.

2.PROBLEM SOLUTION APPROACH

Our first problem is that implementing the *the chaining Hash Table using Binary Search tree.*

To implement the chaining Hash Table using Binary Search tree , we use binary search tree array to chain element. We insert the element to binary search tree with respect to key of value of element's hashcode. Hashcode is determined by java default hashcode method for each key value. The initial Capacity is 11 and load factor is 3.0. If the load factor is exceded , then the hash table is rehashed incrementing the table length more than twice to make odd number.

Our second problem is that *implementing the Hybrid Hash table.*

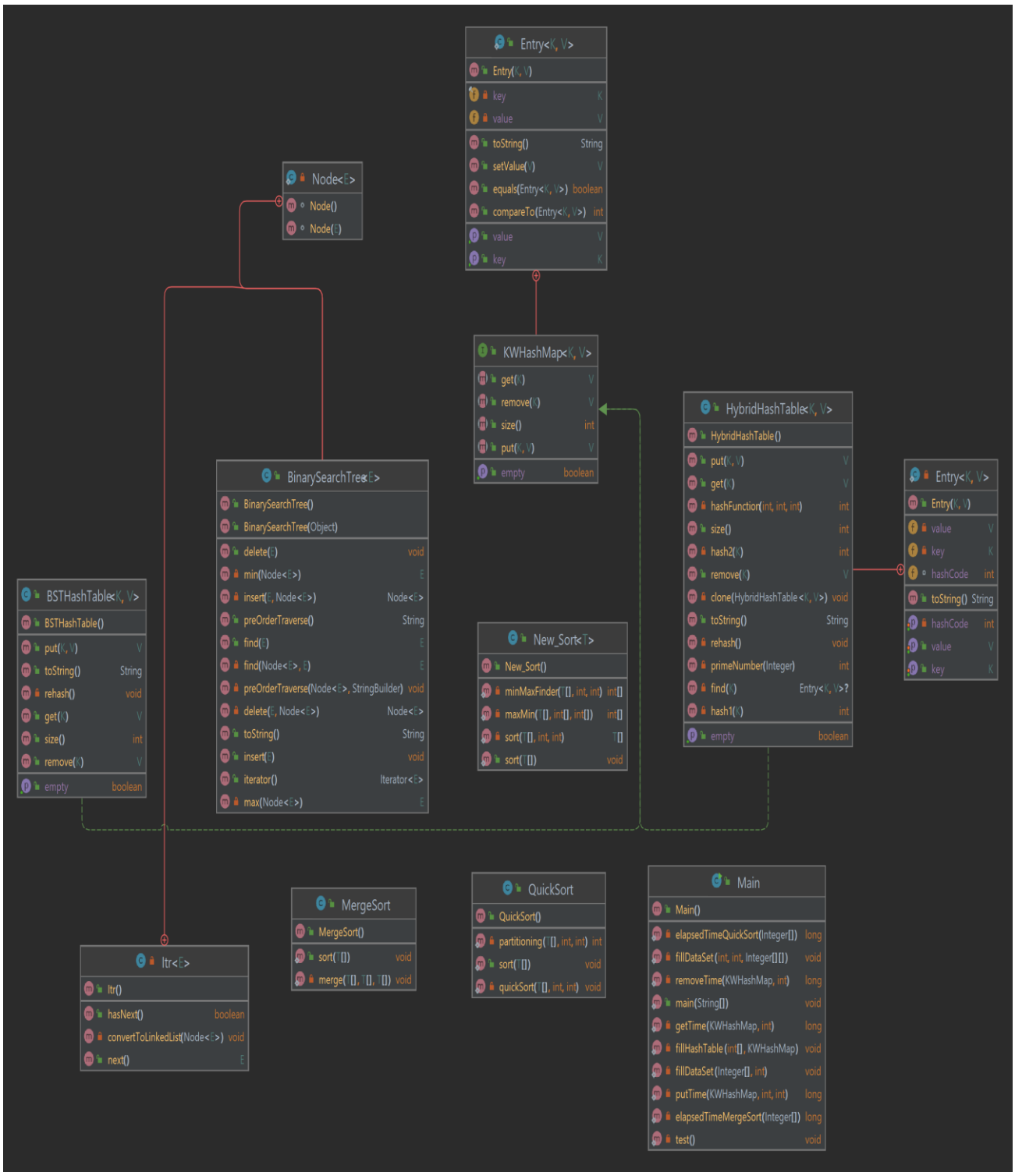
To implement the Hybrid Hash table used Colasced technique and chain technique together. However Colasced tree has different approach to find index of key value. After the key value's index is calculated by formula then inserting the free slot. The new Entry inner class is created with extra field like next pointer. The next pointer mapped to the next key value which original index is that slot. We can find the element using by this next pointer after do operations. Initial Capacity is 101 and load factor is 0.8. If the the load threshold is exceed then the hash table is rehashed incrementing the table length more than twice to make odd number.

***Our third problem** is that Implementing the Merge Sort and Quick Sort Algorithm.*

*We use divide and conquer method to implement **Merge Sort**. The array is divided into left sub array and left sub and array until the left and right sub array size reach 1 recursively. Then two sub array is merged as sorted.*

We use quite similar approach to implement **Quick Sort**. There is pivot element. The pivot element is the first element of array in this algorithm. There are elements at the left side of pivot that less than pivot number and at the right side of pivot that larger than pivot number. Then left side and right side are sorted recursively.

3.CLASS DIAGRAMS



4.TEST CASES

- **Putting the element to BinarySearch Tree Hash Table and Hybrid Hash Table.**
- **Removing the Existing element to BinarySearch Tree Hash Table and Hybrid Hash Table.**
- **Removing the Non-Existing element to BinarySearch Tree Hash Table and Hybrid Hash Table.**
- **Getting the value of Existing key value of BinarySearch Tree Hash Table and Hybrid Hash Table.**
- **Getting the value of non-Existing key value of BinarySearch Tree Hash Table and Hybrid Hash Table.**

- **Is Empty Query**

- **Size of the Hash Table**

- **Sort the array using MergeSort**

- **Sort the array using QuickSort**

- **Sort the array using New Sort**

5.RUNNING COMMANDS AND RESULT

ADD THE ELEMENTS TO BİNARY SEARCH TREE HASH TABLE

```
Adding Elements to Binary Search Tree Hash Table
Adding the (3,9)
Adding the (12,10)
Adding the (22,11)
Adding the (13,12)
Adding the (25,13)
Adding the (23,14)
Adding the (51,15)
Hash Table
{22  , 11 } ->
{12  , 10 } -> {23  , 14 } ->
{13  , 12 } ->
{3  , 10 } -> {25  , 13 } ->
{51  , 15 } ->
```

REMOVE THE EXİSTİNG ELEMENTS TO BİNARY SEARCH TREE HASH TABLE

```
Removing the (25)
Value is :  13
New Hash Table

{22  , 11 } ->
{12  , 10 } -> {23  , 14 } ->
{13  , 12 } ->
{3  , 10 } ->
{51  , 15 } ->
```

REMOVE THE NON- EXİSTİNG ELEMENTS TO BİNARY SEARCH TREE HASH TABLE

```
Removing Non Existed Element (15)
Value is : null

Hash Table{22  , 11 } ->
{12  , 10 } -> {23  , 14 } ->
{13  , 12 } ->
{3  , 10 } ->
{51  , 15 } ->
```

GET THE VALUE OF EXİSTİNG AND NON-EXSTİNG KEY

```
Getting the value of 51 : 15
Getting the value of non Existing Item 101 :  null
```

SIZE OF THE HASH TABLE AND IS EMPTY QUERY

```
Size of BST Hash Table :  6
Is empty ?   false
```

PUT ELEMENT TO THE HYBRID HASH TABLE

```
Adding Elements to HYBRID Hash Table
Adding the (3,9)
Adding the (12,10)
Adding the (22,11)
Adding the (13,12)
Adding the (25,13)
Adding the (23,14)
Adding the (51,15)
Hash Table

Hash Value          Key                 Next
6                   51                  null
32                  25                  34
34                  23                  6
35                  22                  44
44                  13                  32
45                  12                  35
79                  3                   45
```

REMOVE EXSTING ELEMENT FROM HYBRID HASH TABLE

```
Removing the (25)
Value is :  13
New Hash Table

Hash Value          Key                 Next
6                   51                  null
32                  23                  6
34                  23                  6
35                  22                  44
44                  13                  32
45                  12                  35
79                  3                   45
```

REMOVE NON-EXISTING ELEMENT FROM HYBRID HASH TABLE

```
Removing Non Existed Element (15)
Value is : null

Hash Table
Hash Value          Key                 Next
6                   51                  null
32                  23                  6
34                  23                  6
35                  22                  44
44                  13                  32
45                  12                  35
79                  3                   45
```

GET EXISTING AND NON- EXISTNG ELEMENT FROM HYBRID HASH TABLE

```
Getting the value of 51 : 15
Getting the value of non Existing Item 101 :  null
```

SIZE OF THE HASH TABLE AND IS EMPTY QUERY

```
Size of Hybrid Hash Table :  6
Is empty ?   false
```

SORT USING MERGE SORT

```
Sort Using Merge Sort
Array
45 48 55 12 15 66 45 78 78 99 25 16 19
Sorted Array
12 15 16 19 25 45 45 48 55 66 78 78 99
```

SORT USING QUICK SORT

```
Sort Using Quick Sort
Array
45 48 55 12 15 66 45 78 78 99 25 16 19
Sorted Array
12 15 16 19 25 45 45 48 55 66 78 78 99
```

SORT USING NEW SORT

```
Sort Using New Sort
Array
45 48 55 12 15 66 45 78 78 99 25 16 19
Sorted Array
12 15 16 19 25 45 45 48 55 66 78 78 99
Process finished with exit code 0
```