T.C.

MANISA CELAL BAYAR UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

DATA STRUCTURES PROJECT

Name-SURNAME

HASAN ALAY – 200315007

ADVISOR:

DİDEM ABİDİN

EMRE ŞATIR

MANISA, 2021

DATA STRUCTURES PROJECT REPORT

Note: I explained some topics in the main class again.

I used my own data structures except hash structure. First, I wrote a student class, which has definition of student. Also, in student class I have getters and setters. I implemented my student class in main class. I wrote a to string method in the student class.

toString()

helps to print all elements in the structures with string format.

```
public String toString() {
    return "[" + name.toUpperCase() + " " + surname.toUpperCase() + " " +
studentId + "] ";
}
```

STRUCTURES AND SIMPLE OVERVIEW OF OPERATIONS

LIST:

I decided to use array list as list structure because, I had student name, surname and student id to implement. Each student is an array and these arrays are parts of an array list. I wrote my own generic codes to add and remove student in array list class. This array list can increase size itself. Array list's base capacity is 10 if it reaches ten nodes, it increases size 2 time. Whenever you want to add a new student, it will ask you student name, surname and id. On the other hand, you want to remove a student it will ask you a student id. Remove method uses your input as removeId. Whenever removeId is equal to a student id, it will remove this student in array list. In the find operation, program controls are there a student in the list. If list contains at least one student program asks a student id to find. I used a for loop to search student. When it finds the student, it returns all information about student and hops. For the 4th operation I used toString for printing all astudents in the list structure.

OVERVIEW ON ARRAY LIST METHODS:

add (int index, AnyType x)

It always checks our list’s size. If it is full it set new capacity: the size*2.

And it adds the new student to the next index on the list.

remove (AnyType x)

It roams on the list with a for loop. When our removeId equals to the student id information in the i th index node on the list. It removes the student with all information

toString()

helps to print all elements in the structure with string format. When I try to print my list with this method my to string method from student class and this method work together.

HASH:

For the hash structure I used my own hash table that contains insert, remove, contains, get, print hash Table and my hash functions. Int first version of the project I used liked hash table structure from the java library. Later I learnt this usage of the hash is wrong, so I changed my structure and somewhere I used lab source codes. My hash structure can insert, remove and find the students that user inputs. To add a student, I use user’s id input as key and other information as value. And find a place with the hash function for the student that added before. I used hash function from the lab codes. Because these both work with integers and I can store my students in the array which’s length is 10. I didn’t need write a new hash function for this. To remove a student from hash table I created a new object from the student class. If this object can find the equal of the id, that inputted as remove id in the table, it takes the all student information in the object. If it can’t find the equal of the id, that inputted as remove id it prints "This student isn't in the structure" message as error. When it finds, my remove method removes the new object from the student class, that I created for the remove a student from the table. To find a student in the table I used same logic with remove method. I created another object from the Student class. When user decide to find a student a student my object try to get information in the hast table with the find id input from the user. If it can’t equalize find id with an id in the hash table, it prints "THIS STUDENT CANNOT BE FOUND" message as error. If it can equalize it get the all information of that student to my object, that created from the Student class. Then prints my object with “toString” method. I tried to print number of the hops, but I couldn't. I want to explain something about myself. I changed my department this year. When this semester started, I started to study java language. My java knowledge is not good as my friends. Then this project declared. But I still have issues object-oriented programming and java. In 25 days, I studied subjects of three semesters and tried to write this project. Maybe I can have some problems in my code. I'm already sorry for them. To print hash table, I used print method in the hash table class. It prints all students in the hash table with any problem.

OVERVIEW ON HASH METHODS:

Insert (Student student)

Firstly, this method creates a linked list which's type is Student. When you try to insert a student, it sends the student id to the hash function, then find an index for it. After that it checks linked list contains this student or not. If it does not contain it adds the student in the linked list(whichlist).

```
public void insert(Student student) {
    LinkedList<Student> whichList =
theLists[myHash(student.getStudentId())];

    if (!whichList.contains(student))
        whichList.add(student);
}
```

remove (int key)

In this method I created a linked list whose name is whichlist. It takes the student (This student is an object that you create in the operation 2) id and send it to the hash function and find the index of this student id. If our linked list, that in this index contains a student with this id, it removes the student in the hash table.

```
public void remove(Student student) {
    LinkedList<Student> whichList =
theLists[myHash(student.getStudentId())];

    if (whichList.contains(student))
        whichList.remove(student);
}
```

public Student get (Integer id) {}

I use this method to helps me when I try to find and delete a student in the hash table. It contains a nested loop that can get a student in the hash structure. First for is roam between the theLists(the lists that created top of the hash table class. They are linked lists that handle our other linked lists.) Second for roam between the whichlists and try to get a student with an id. If it can get it returns that student.

```java
public Student get(Integer id) {
    LinkedList<Student> whichlist;

    for (int i = 0; i < theLists.length; i++) {
        whichlist = theLists[i];
        for (int j = 0; j < whichlist.size(); j++) {
            Student ogrenci = whichlist.get(j);
            if (ogrenci.getStudentId() == (id)) {
                return ogrenci;
            }
        }
    }
    return null;
}
```

myHash(Integer x)

This is my hash function. It is from the lab source codes and it can help me as well. I didn’t need to change the function.

```java
private int myHash(Integer x) {
    return (x % theLists.length);
}
```

printHashTable ()

This method uses a nested loop to print all students in the hash structure. whichLists are connected to the theLists.

```java
public void printHashTable() {
    LinkedList<Student> whichList;

    for (int i = 0; i < theLists.length; i++) {
        whichList = theLists[i];

        System.out.print("|" + i + "|" + " --> ");

        for (int j = 0; j < whichList.size(); j++)
            System.out.print(whichList.get(j) + " --> ");

        System.out.println();
    }
}
```

BINARY SEARCH TREE:

For binary search tree structure, I wrote recursive add and delete methods. I made little bit change on the standard node class. First, my node class is inner class. My nodes use student information as node data. Add method can find exact location to add node in the tree. Delete method can delete nodes that has no children, one child, two children. My first delete method couldn’t delete nodes with two children. Then I wrote a minimum value method and used it in the delete method. To add a student in the binary search tree I used insertRecursively method and send the student object in that. It finds the exact location for the student with using its student id as key. For delete method I use deleteRec method and send the id that user input as the student id which asked before. For the 3rd operation I used findNode method that is in the binary search tree class. This method has a control and while loop. In the loop it compares the id that inputted before. If it equals the root it prints the root node and control. If it is smaller than root, it goes to the left. And if is bigger than root node it goes to the right node. It is typical binary search tree property. It makes it when the temp node isn’t equal to the null. For the 4th operation, I used a switch statement to ask user for print type (inorder, preorder, postorder). My binary search tree class has three print methods. They can print inorder, print preorder, print postorder.

OVERVIEW ON BINARY SEARCH TREE METHODS:

Node<AnyType> insertRecursively (AnyType newValue, Node<AnyType> tempNode)

That method takes the studentInfo from the student class as new value. Then compares the student id in the studentInfo with temp node’s student id. If temp node is null it creates a new node. If studentInfo is smaller than the temp node’s student id it goes to the left child. If studentInfo is bigger than the temp node’s student id it goes to the right child. It works until find the correct position for our new student.

deleteRec (Node<AnyType> root, int data)

this method compares user’s input(data) with root node in the binary search tree. If input is smaller than student id which is in the root node it gets to the left child. On the other hand, if it is bigger than student id which is in the root node it gets to the right child. This method makes it until it finds the correct student id. Then delete the student from our tree.

Student minValue (Node root)

This method finds the smallest value in the binary search tree. It helps me to delete nodes with one or two children. It goes to the leftmost node while left node isn’t null.

```
Student minValue(Node root)
{
    int minv = root.data.getStudentId();
    while (root.left != null)
    {
        minv = root.left.data.getStudentId();
        root = root.left;
    }
    return root.data;
}
```

findNode (int x)

This method starts from the root node and works until temp node is null. It gets an input from user and search for the input on the binary search tree. It uses binary search tree logic to find a student. When it finds the student, it prints all student information and that node’s level on our tree with using a counter.

printPreorder ()

It prints nodes with this order: root-left-right.

printPostorder ()

It prints nodes with this order: left-right-root.

printInorder ()

It prints nodes with this order: left-root-right.

5TH OPERATION: PRINT ALL DISTINCT NAMES

I used hash set for 5th operation. I wrote all names in the array list to hash set with a for loop. As I know hash set contains unique elements only. I decided to use hash set for this operation. Then I wrote I foreach loop to print all names in the hash set. If user press 5, it will print all distinct names in the hash set.

```
if (myList.size() == 0){
    System.out.println("THE STRUCTURE IS EMPTY!!");
}
```

```java
// adds distinct names in arraylist to hashset. Then print all names in
hashset.
for (int i = 0; i < myList.size(); i++) {
    names.add(myList.get(i).getName().toUpperCase());

}
for (String item : names)
    System.out.println(item);
```

6TH OPERATION: PRINT FREQUENCIES OF EACH NAME

For the 6th operation I used tree map. I wrote a for loop to roam on the array list. Then I copied every single name with this loop. After that I created a string object that hold the name in the array list. If a name comes again from the array list to my tree map structure, my tree map structure(nameCount) replace the old frequency with new and add 1 to its frequency. If the name is not in my tree map structure before it puts the student in the tree map and set its frequency as 1. Finally, I wrote a foreach loop to print all names and their frequencies.

```java
// Create a treemap object and counts the all names in arraylist then print
all names and their frequencies.
if (myList.size() == 0){
    System.out.println("THE STRUCTURE IS EMPTY!!");
}
TreeMap<String, Integer> nameCount = new TreeMap<>();
for (int i = 0; i < myList.size(); i++) {
    String counts = myList.get(i).getName().toUpperCase();

    if (nameCount.containsKey(counts)) {
        nameCount.replace(counts, nameCount.get(counts) + 1);
    } else {
        nameCount.put(counts, 1);
    }
}
for (Map.Entry<String, Integer> entry : nameCount.entrySet()) {
    System.out.println(entry.getKey() + ":" + entry.getValue());
}
```

In the 7th operation it prints author’s information.

MAIN CLASS

I wrote my code in while (true) loop. I wanted to print all operations every single time user made an operation.

I have a scanner object to get user’s input.

I used a switch statement for our 8 operations.

CASE 0:

When user wants to exit the program it presses the 0 and program asks “Are you sure? Y/N”

When user presses y or Y the program closes itself and prints "the program is closing!"

When user presses n or N the program continues to work and asks "Enter your selection: "

When user inputs something unexpected the program prints "UNEXPECTED INPUT"

CASE 1:

When user presses 1, the program ask for a name and take the user’s input as scanner object’s value. Them make this for surname and id.

When user inputs all information about student, the program adds these information in the array list, hash table and the binary search tree objects. After that, program prints "STUDENT HAS BEEN ADDED SUCCESSFULLY."

CASE 2:

When user presses 2, the program checks the size of array list. If size of the array list is 0 the program prints "THERE IS NO STUDENT IN THE STRUCTURES. PLEASE ADD A STUDENT." Then continue to work and asks for user’s input.

If size of the array list is not 0. The program asks for an id to remove from our structures.

There is a for loop for check user’s id is equals or not with a student id in the array list. When is equal it removes the student from array list.

To remove a student from hash table I created a new object from the student class. If this object can find the equal of the id, that inputted as remove id in the table, it takes the all student information in the object. If it can’t find the equal of the id, that inputted as remove id it prints "This student isn't in the structure" message as error. When it finds, my remove method removes the new object from the student class, that I created for the remove a student from the table.

My binary search tree can delete a node recursively. It uses it’s delete method. This method finds the node and delete that node without any problem.it doesn’t lose a child node.

CASE 3:

In this case first, program checks is array list empty or not. If is empty it gives an error, then ask an input for an operation. If it is not empty it asks for an id to find student in the structures.

I wrote a for loop to roam on the array list. When it equalizes findbyid with the any student id it prints all information about this student. Also, it prints index number + 1 of this student for number of hoop that finder make.

For binary search tree I used findNode method, which is in the binary search class. This method roam on the binary search tree and tries to equalize find by id with a student id that is in the binary search tree. It can compare which number is bigger which is smaller.

For hash structure. To find a student in the table I used same logic with remove method that is in the hash. I created another object from the Student class. When user decide to find a student a student my object try to get information in the hast table with the find id input from the user. If it can’t equalize find id with an id in the hash table, it prints "THIS STUDENT CANNOT BE FOUND" message as error. If it can equalize it get the all information of that student to my object, that created from the Student class. Then prints my object with “toString” method. I tried to print number of the hops, but I couldn’t.

CASE 4:

First, program checks is array list is empty or not. If is empty it throws and error and asks a input for an operation. If it is not, it asks “Please select a data type”. Here I used another switch block. First case is for list second is for hash and third one is for tree.

In Case 1, program prints array list’s object with toString method, that written before. Also, I could write array list wit for loop, but toSting is more effective.

In Case 2, the program prints hash structure with printHashTable method. This method roams every element in the hash structure and prints.

In Case3, I used another switch block. Because I ask user to print binary search tree print format. User can select inorder, preorder and postorder. When user wants to print inorder, the program uses printInorder method. This method prints nodes in left-root-right order. When user wants to print preorder, the program uses printPreorder method. This method prints nodes in root-left-right order. And finally, when user wants to print postorder, the program uses printPostorder method. This method prints nodes in left-right-root order.

The case 4 (list all elements in the structures) is done here.

Case 5:

In this case, again I check the array list is empty or not. If it is empty the program prints "THE STRUCTURE IS EMPTY!!". If it is not empty, the program starts to roam on the array list. here I copy every name in the array list to the hash set. The rule of set is all elements are unique. This rule helps me to the copy all distinct names in the array list to another object. Then wrote a foreach loop to print all names in the hash set(names).

Case 6:

In this case, again I check the array list is empty or not. If it is empty the program prints "THE STRUCTURE IS EMPTY!!". I created my tree map object in the case because I wanted to reset the map every time when user presses 6 or make any other operation. I wrote a for loop to roam on the array list. Then I copied every single name with this loop. After that I created a string object that hold the name in the array list. If a name comes again from the array list to my tree map structure, my tree map structure(nameCount) replace the old frequency with new and add 1 to its frequency. If the name is not in my tree map structure before it puts the student in the tree map and set its frequency as 1. Finally, I wrote a foreach loop to print all names and their frequencies.

CASE 7:

In the 7th operation the program prints author’s information.