Lab Exercise 7: Hashing [30 Marks]

SE2205: Data Structures and Algorithms using Java – Fall 2023

Open Day: November 18, 2023; **Cut off time**: November 25, 2023, Saturday @11.55pm

---

Prepared by Dr. Quazi Rahman (qrahman3@uwo.ca); see your instructor if you need any clarification on the lab handout.

A. Rationale and Background

In this lab Exercise, we will demonstrate our understanding on hashing with open-addressing collision-resolving techniques. We will be using Integer type dataset to accomplish this task. Before working on this lab, reviewing Unit 5 will be a very important task we are highly recommended to complete. This lab should prepare you with all the background info on hashing and for the final exam.

B. Evaluation and Submission Instructions

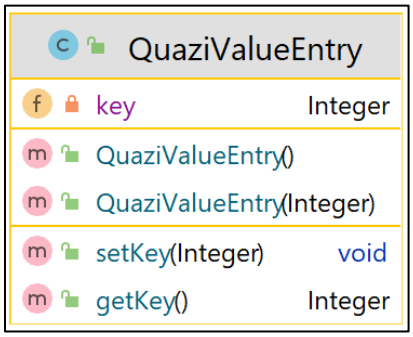
Submit your Lab-Exercise online by carrying out the following instructions:

1. Create a Project with the following name: *username_LabExercise7*
2. For this question create a package for each Question (LE7Q1, LE7Q2)
3. Use meaningful names for each class and the associated variables by following the general naming conventions.
4. For this question, use the static header and footer methods you created before.
5. Comments: **Writing short but clear comments for Lab Exercise is <u>mandatory</u>**. If the comments are clear, full credit will be given to the written comments.
6. Once the Exercise is completed, go to your Lab folder. Select the project folder (e.g. *username_LabExercise7*). Right-click to select 'Send to' 'Compressed zipped folder'. Make sure it has the same naming convention (e.g., *username_LabExercise7*.zip). Upload this file to OWL as your submission.
7. You will be graded on this lab Exercise based on the comments and running code.
8. Please replace the name *Quazi* in your code with your firstName wherever the name *Quazi* is used.

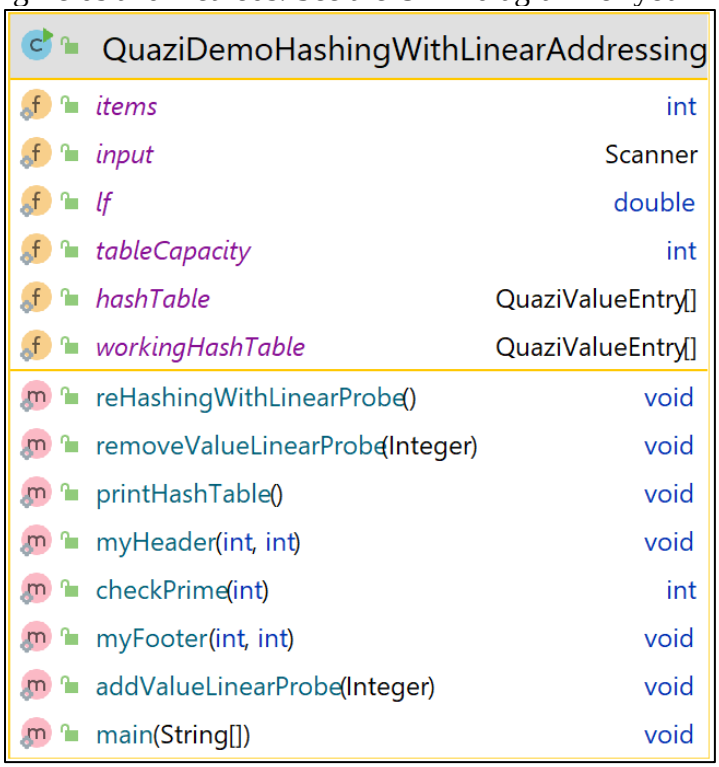
C. Lab Questions

Question 1 [20 Marks]

1. Define a class called ***YourFirstnameValueEntry***. The class should contain the following field and methods. See the UML diagram for your information:
   - An Integer type private field.
   - No argument constructor: This should assign -1 for the field. This value (-1) will be considered as the '**null**' flag for the hash table. (Option: you may choose to assign null here)
   - Constructor with argument.
   - Getter and Setter methods.



2. Now define a class called ***YourFirstnameDemoHashingWithLinearProbing***. The class should contain the following fields and methods. See the UML diagram for your information:



(a) Declare the following public static fields:
   i) items – a int type field that will hold the value of the total number of data items to be added to the hash table.
   ii) input – a Scanner type object reference
   iii) lf – double type field to record the load-factor.
   iv) tableCapacity – will store the value of the table capacity.
   v) hashTable – a QuaziValueEntry type array reference
   vi) workingHashTable – a second QuaziValueEntry type array reference, which will be used to store the copy of the hashTable reference for rehashing operation.

(b) Define a public static void method addValueLinearProbe(Integer), which will accept an integer key and add this to the hash table based on linear probing technique.

(c) Define a public static checkPrime(int) method that will accept the initial table capacity and return the nearest prime number which will be assigned to the tableCapacity field. Here is the code that you may copy-paste if you want to.

```
//since in hashing, the prime number has to be greater than 2, we can
discard 2; 0 and 1 are not prime numbers by definition
public static int checkPrime(int n) {
    int m = n / 2;//we just need to check half of the n factors
    for (int i = 3; i <= m; i++) {
        if (n % i == 0) {//if n is not a prime number
            i = 2; //reset i to 2 so that it is incremented to 3 in the for-
header
            //System.out.printf("i = %d\n",i);
            n++;//next n value
            m = n / 2;//we just need to check half of the n factors
        }
    }
    return n;
}//end of checkPrime()
```

(d) Define a public static void method removeValueLinearProbe(Integer), which will accept an integer key, search through the hash table using linear probing technique and if it finds the key, it will remove it by replacing the key with -111 value as a flag for '**available**' location, otherwise it will print a message that the key is not found (see the sample output 2).

(e) Define a public static void method printHashTable(), which will print the content of the hash table (see the sample output). In printing the keys, if the key is -1, this method will print ***null***, and if the key is -111, it will print '***available***'.

(f) Define a public static void method called rehashingWithLinearProbe() which will rehash the table according to rehashing technique discussed in the class.

(g) Copy-paste the header and footer methods from the previous labs and modify accordingly.

(h) Define the driver method with the following specifications:

i) Call your header method.

ii) Prompt the user for the number of items and load factor (see the sample output; use the same values as shown in the sample output to cross check your result).

iii) Find the table capacity from the above two values (number of items and load factor), and then make sure that you use the nearest prime number by calling the checkPrime() method and get the required capacity value.

iv) Print the minimum required table-capacity value on the screen.

v) Create a QuaziValueEntry type array of size of the table-capacity (from iv) and assign that to hashTable field.

vi) Instantiate the table contents with no-argument constructor.

vii) Enter each of the key items using the keyboard and pass that to addValueLinearProbe(Integer) method which will add each key to the hashTable using linear probing technique.

viii) Once done print the hash table by calling the printHashTable() method.

ix) Now remove two items from the hashTable by prompting the user and subsequently calling the removeValueLinearProbe(Integer) method. Every time after removing the key, print the hashTable content (see the sample output).

x) Now add a new key to the hashTable and print it again.

xi) Call the rehashing method and then print the table again.

xii) Call your footer method.

Note: To get a good understanding of the hashing concept use a second set of data during second run as shown in the sample output 2. Since you are working with integer keys, your results should be identical for identical set of data with identical order. This may not be the case

for keys generated from different instantiated objects which are neither numerical nor String type. Use different data sets based on your choice; do the work on paper, and then cross check your result by running your code.

Sample output 1 (With load factor 0.75):

```
=======================================================
Lab Exercise 7, Q1
Prepared By: Quazi Rahman
Student Number: 999999999
Goal of this Exercise: ........!
=======================================================
Let's decide on the initial table capacity based on the load factor and dataset size
How many data items: 5
What is the load factor (Recommended: <= 0.5): 0.75
The minimum required table capacity would be: 7
Enter item 1: 13
Enter item 2: 23
Enter item 3: -12
Enter item 4: 26
Enter item 5: -43
The Current Hash-Table: [-43, null, 23, -12, null, 26, 13]

Let’s remove two values from the table and then add one…….

Enter a value you want to remove: -12
-12 is Found and removed! The Current Hash-Table: [-43, null, 23, available, null, 26, 13]
Enter a value you want to remove: 26
26 is Found and removed! The Current Hash-Table: [-43, null, 23, available, null, available, 13]
Enter a value to add to the table: 9
The Current Hash-Table: [-43, null, 23, 9, null, available, 13]

Rehashing the table...
The rehashed table capacity is: 17
The Current Hash-Table: [null, null, null, null, null, null, 23, null, -43, 9, null, null, null, 13, null, null, null]
=======================================================
Completion of Lab Exercise 7, Q1 is successful!
Signing off - Quazi
=======================================================
```

Sample output 2 (With load factor 0.5):

```
=======================================================
Lab Exercise 7, Q1
Prepared By: Quazi Rahman
Student Number: 999999999
Goal of this Exercise: ........!
=======================================================
Let's decide on the initial table capacity based on the load factor and dataset size
How many data items: 5
What is the load factor (Recommended: <= 0.5): 0.5
The minimum required table capacity would be: 11
```

Enter item 1: -31
Enter item 2: 13
Enter item 3: -23
Enter item 4: -21
Enter item 5: 11
The Current Hash-Table: [11, -21, -31, 13, null, null, null, null, null, null, -23]

Let's remove two values from the table and then add one……

Enter a value you want to remove: -31
-31 is Found and removed! The Current Hash-Table: [11, -21, available, 13, null, null, null, null, null, null, -23]
Enter a value you want to remove: 17
17 is not found! The Current Hash-Table: [11, -21, available, 13, null, null, null, null, null, null, -23]
Enter a value to add to the table: 21
The Current Hash-Table: [11, -21, 21, 13, null, null, null, null, null, null, -23]

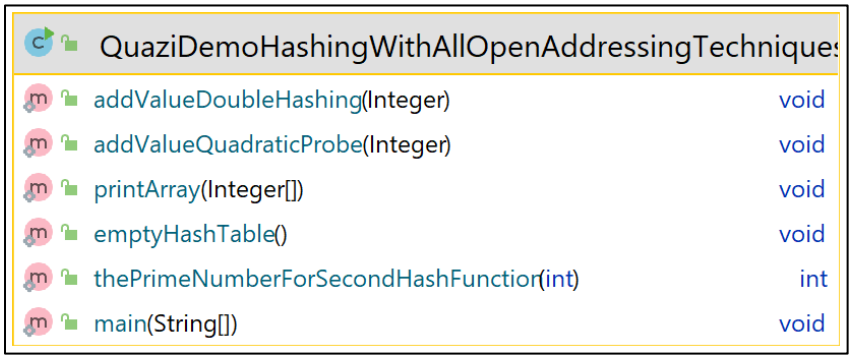
Rehashing the table...
The rehashed table capacity is: 23
The Current Hash-Table: [-23, null, -21, null, null, null, null, null, null, null, null, 11, null, 13, null, null, null, null, null, null, null, 21, null]
==========
Completion of Lab Exercise 7, Q1 is successful!
Signing off - Quazi
==========

Question 2 [10 Marks]

1. Define a class called ***YourFirstnameDemoHashingWithAllOpenAddressingTechniques***. Here you will reuse most of the codes you wrote for question 1. To be able to reuse these codes, import the classes and static fields/methods from the other package you created for question 1 as follows:

```
import PackageNameForQ1.*; //this will import classes from the Q1 package
import static PackageNameForQ1.TheDriverClassName.*; //this will import all the
static fields and methods from the driver class of the Q1 package
```

2. This class should contain the following methods. See the UML diagram for your information:



(a) Define a public static void method addValueQuadraticProbe (Integer), which will accept an integer key and add this to the hash table based on quadratic probing technique.
(b) Define a public static void method addValueDoubleHashing(Integer), which will accept an integer key and add this to the hash table based on double hashing probing technique. In this

case, it will call the method called thePrimeNumberForSecondHashFunction(int) method that will return the prime number q for the second hash function.

(c) Define a public static method called thePrimeNumberForSecondHashFunction(int) method that will return the prime number q for the second hash function by accepting the current table capacity. The goal here is to get a second prime number which is immediately smaller than the table capacity (if the table capacity is 23, the second prime number should be 19). To define this one, you can modify the code provided to you for checkPrime(int) method in question 1.

(d) Define a public static void method called emptyHashTable() method, which will reset the hashTable with values -1 (null-flag). The goal would be to call this method before calling addValueQuadraticProbe (Integer) and addValueDoubleHashing(Integer) methods from the main().

(e) Define a public static void method called printArray(Integer[]) which will print the contents of a given Integer array.

(f) Define the driver method with the following specifications:

i) Call your header method.
ii) Prompt the user for the number of items (use 5) and load factor (use 0.5) (see the sample output).
iii) Find the table capacity from the above two values (number of items and load factor), and then make sure that you use the nearest prime number by calling the checkPrime() method (it has already been imported for you with your import statement in the top of your code).
iv) Print the minimum required table-capacity value on the screen.
v) Create a QuaziValueEntry type array of size of the table-capacity (from iv) and assign that to hashTable field.
vi) Instantiate the table contents with no-argument constructor.
vii) Create an Integer type array with the following values: 7, 14, -21, -28, 35.
viii) By calling the printArray() method print those values on the screen as shown in the sample output.
ix) Add each of the key items from the above Integer array to the hashTable by calling addValueLinearProbe(Integer) method which will add each key to the hashTable using linear probing technique.
x) Once done print the hash table by calling the printHashTable() method.
xi) Now reset the hashTable by calling the emptyHashTable() method.
xii) Now add each of the key items from the above Integer array to the hashTable by calling addValueQuadraticProbe(Integer) method which will add each key to the hashTable using quadratic probing technique.
xiii) Once done, print the hash table by calling the printHashTable() method.
xiv) Now reset the hashTable by calling the emptyHashTable() method.
xv) Now add each of the key items from the above Integer array to the hashTable by calling addValueDoubleHashing(Integer) method which will add each key to the hashTable using double hashing probing technique.
xvi) Once done, print the hash table by calling the printHashTable() method.
xvii) Call your footer method.

Note: To get a good understanding of the hashing concept use a second set of data during second run as shown in the sample output 2 with a load factor of more than 0.5 (use the same values as shown in the sample output). Observe that with a load factor of more than 0.5 the quadratic probe will fail (high lighted below for your information).

Sample output 1 (With load factor: 0.5):

```
------------------------------------------------------
======================================================
Lab Exercise 7, Q2
Prepared By: Quazi Rahman
Student Number: 999999999
Goal of this Exercise: .....!
======================================================
Let's demonstrate our understanding on all the open addressing techniques...
How many data items: 5
What is the load factor (Recommended: <= 0.5): 0.5
The minimum required table capacity would be: 11
The given dataset is: [-11, 22, -33, -44, 55]
Let's enter each data item from the above to the hash table:

Adding data - linear probing resolves collision
The Current Hash-Table: [-11, 22, -33, -44, 55, null, null, null, null, null, null]

Adding data - quadratic probing resolves collision
The Current Hash-Table: [-11, 22, null, null, -33, 55, null, null, null, -44, null]

Adding data - double-hashing resolves collision
The q value for double hashing is: 7
The Current Hash-Table: [-11, -33, 55, null, null, null, 22, null, null, -44, null]
======================================================
Completion of Lab Exercise 3, Q2 is successful!
Signing off - Quazi
======================================================
```

Sample output 2 (With load factor: 0.75 and a different data set):

```
-----------------------------------------------------------------------------------------------
======================================================
Lab Exercise 3, Q2
Prepared By: Quazi Rahman
Student Number: 999999999
Goal of this Exercise: .....!
======================================================
Let's demonstrate our understanding on all the open addressing techniques...
How many data items: 5
What is the load factor (Recommended: <= 0.5): 0.75
The minimum required table capacity would be: 7
The given dataset is: [7, 14, -21, -28, 35]
Let's enter each data item from the above to the hash table:

Adding data - linear probing resolves collision
The Current Hash-Table: [7, 14, -21, -28, 35, null, null]
```

Adding data - quadratic probing resolves collision
Probing failed! Use a load factor of 0.5 or less. Goodbye!
The Current Hash-Table: [7, 14, -28, null, -21, null, null]

Adding data - double-hashing resolves collision
The q value for double hashing is: 5
The Current Hash-Table: [7, 14, -28, null, null, 35, -21]
==========
Completion of Lab Exercise 7, Q2 is successful!
Signing off - Quazi
==========

Here are some sample questions for your exam-prep on hashing.

1. *What is the goal of using hashing technique in Data Structure?*
2. *Explain linear, quadratic, double hashing probing techniques?*
3. *What are the advantages of using open addressing over linear chaining? What is the price you pay to enjoy those advantages?*
4. *Best-case/worst-case time/space complexity of hashing technique in inserting/searching/removing data items when open addressing/linear chaining resolve collision. What is the rationale of your answer.*
5. *What is the load factor. How does it play a role in hashing?*
6. *What is the use of 'available' flag in hashing?*
7. *Based on the background provided in the lecture hand out, explain/show how (by drawing on the screen) for each run in both questions, a linear chain can be constructed.*
8. *Explain/demonstrate how in q-2, run-2 the quadratic probing failed due to a load factor of greater than 0.5. Will this always happen? Demonstrate it using the following data set with a load factor of 0.75 in the order as shown: -7, 13, -29 40, 54.*
9. *Demonstrate by hand what would be content of a hash-table when the data set from above (Q7) is used with load factor 0.75 for each of the following cases: linear probing resolve collision, quadratic probing resolve collision, double-hashing resolve collision, linear-chaining resolve collision.*
10. *For the open addressing techniques, run your code and make sure that your code for open addressing is working fine.*
11. *What is rehashing? Why is it required? How to accomplish this task.*
12. *Review the method checkPrime()?*
13. *It seems to be much easier approach if we overwrite the hash table with no argument constructors every time, instead of defining a public static void method called emptyHashTable() method, which will reset the hashTable with values -1 (null-flag). What is the rationale behind defining this method?*