ASSIGNMENT 2

Subject : Hash Tables
TAs: Selman Bozkır, Merve Ozdes
Programming Lanuage: Java
Due Date: 03.05.2020 23:59

1 Introduction

Searching and finding data on a data structure is a painful process. Various search algorithms have been created and various data structures have been used. For example, the hashing data structure which is designed to use a special function called the Hash function is one of them. Hash function maps a given value with a particular key and it allows faster access to elements. The efficiency of mapping depends of the efficiency of the hash function used. Finding the appropriate hash function which will always produce a unique key is very difficult or even impossible in some cases. If the same key is generated from two different values, this is called collision. A hash function should be able to take any length value and it should be able to generate a key of the specified length. It should not allow collision. Hash table is data structure that holds data and keys obtained by hashing function together. Assume that we have the set of integer items 54, 26, 93, 17, 77, and 31. The size of hash table is m=11. Hash function takes the set of integer items and return a key as its hash value between 0 and m-1 (h(item)=item%11). A hash table with six items is shown in Figure 1.

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

Figure 1: Hash Table with six items

If, when an element is inserted, it hashes to the same value as an already inserted element, then we have a collision and need to resolve it. There are several methods for dealing with this:

- Separate chaining
- Open addressing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

The load factor α of a hash table with n elements is given by the following formula:

$$\alpha = n/table.length \tag{1}$$

Thus, 0 $<\alpha<$ 1 for linear probing. (α can be greater than 1 for other collision resolution methods. With separate chaining, it is possible to have $\alpha > 1$. In separate chaining, table size equals to the number of linked lists, so $\alpha >$ is the average length of the linked lists. Load factor refers to the

size of the array and the density of the filled cells and it depends upon the hashing algorithm and collision algorithms selected. Some algorithms require a load factor of about .5 or .6; others allow a load factor of > 1.
In this assignment only three of the above-mentioned methods which are Separate Chaining ,Linear Probing and Double Hashing will be exemplified.

Separate Chaining is defined as a method by which linked lists of values are built in association with each location within the hash table when a collision occurs

If we want to add values 44, 55 and 20 respectively to the hash table, there will be a collision due to the same hash value. In Figure 2, separate chaining method is used for collision resolution. A list of all elements that hash to the same value is kept. The array elements are pointers to the first nodes of the lists. A new item is inserted to the front of the list or end of the list.

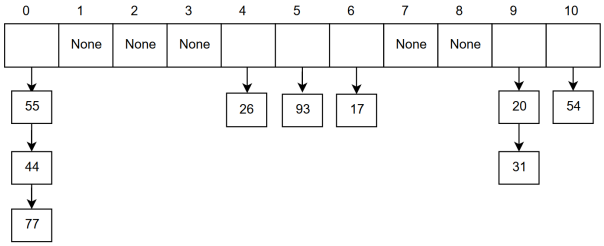

Figure 2: Collision resolution with separate chaining

Linear Probing During insert of key k to position p: If position p contains a different key, then examine positions p+1, p+2, etc.* until an empty position is found and insert k there. During a search for key k at position p: If position p contains a different key, then examine positions p+1, p+2, etc.* until either the key is found or an unused position is encountered.

Double Hashing is used to reduce secondary clustering. This method uses a second hash function to generate different probe sequences for different keys to drive the collision resolution.

Assume that we will add to hash table integer items 14, 17, 25, 37, 34, 16, 26 respectively and the first hash function is h_1(item)=item%11 and the second hash function is h_2(item) = item % 7 + 1. These functions are taken as examples and double hashing can be done by different functions. Double hashing can be done using :

$$(h_1(item) + i * h_2(item))\%TABLE\_SIZE \tag{2}$$

Here $h_1()$ and $h_2()$ are hash functions and TABLE_SIZE is size of hash table. (We repeat by increasing i when collision occurs)
14 mod 11 = 3
17 mod 11 = 6
25 mod 11 = 3 (Collision) —> (h_1(25) + 1 * h_2(25)) % 11 =8
37 mod 11 = 4
34 mod 11 = 1
16 mod 11 = 5
26 mod 11 = 4 (Collision) —> (h_1(26) + 1 * h_2(26)) % 11 =10

0	1	2	3	4	5	6	7	8	9	10
None	34	None	14	37	16	17	None	25	None	26

Figure 3: Collision resolution with double hashing

2 Problem Definition

In this assignment, you are expected read a data file and create a hash table for the data and apply different collision resolution methods. The data set for this assignment is a collection of employees information data. Included in this data are as below:

- employeeCode: A unique Identification code for each employee.
- NRIC: The National Registration Identity Card number.
- phone: Phone number of the employees.

Note: In this assignment you are NOT allowed to use Java's (or other languages') default hash table/map/sorting/linked list/etc utilities.

3 Experiment

3.1 Part A: Separate Chaining

1. Each line of the dataset represents one employee's information. Create a class Employee whose instance can contain all the data from an employee. It should contain:
   - The employeeCode
   - The NRIC
   - The phone
2. Create a basic linked list class.
3. Create a class MyHashTable. This will be your hash table class you will use to look up test data later in the assignment. It should minimally contain:

   • A private table member. This should be an array of linked lists. The linked lists will hold EmployeeData objects. The size of the linked lists depends on load factor. You will handle collisions by inserting at the end of the linked list contained in the slot calculated by the hash function. (see below)

   • A private hash() method. This method will take a key (a phoneNumber) and generate a number (hashvalue) between 0 and TABLE_SIZE (the size of the table array) based on the given key. Generate the key value using the following hash function:

```
int hash1(int key)
{
  return (key % TABLE_SIZE);
}
```

• A public put() method. This method will take a key (a phoneNumber) and an Employee object, and insert it into the hash table. It should make use of hash() to decide which table index to insert the object into.

• A public get() method. This method will take a key (a phoneNumber) and retrieve (return) the full Employee object that has the given phoneNumber. It should make use of hash() to decide the table slot to look for the object in.

- When you use hash() to get a table index, the linked list you find there will probably have multiple objects in it. Simply linear search through this linked list when you get there.
- Make sure your get() function can handle cases where the full object doesn't exist in the table!

3.2 Part B: Linear Probing and Double Hashing

In this part, you should read dataset as in the Part A and you should create the hash table for both Linear Probing and Double Hashing.

- You should use hash function which is given above for first hash function
- Second hash function for double hashing is given below:

```
public int doubleHashFunction(int key) {
          return 1 + (key % (TableSize - 1));
       }
```

4 Input and Output Format

Sample Input

The collection of employee information data will be used for this assignment. The format of the file given to you as input is txt. This data includes EmployeeCode, NRIC, and PhoneNumber.A part of the dataset for creating a hash table is shown below in Figure 4.The input consists of some instances. The first line of the file is column names separated by a space. This is followed by lines containing three columns about employee information separated by a space.

Sample output

The representation of the output file is shown in Figure 5. The entire output file will be shared with you. First line of your output file should contain test case arguments as shown below:
<input_file_name>,<load_factor1>,<load_factor2><search_key>
And it should be followed by hash tables of each part. And lastly, you should print comparisons and CPU time taken to search key given as a parameter for each collision method which are separate chaining, linear probing, and double hashing respectively. For testing your program, command-line parameters should be as shown below.

Command-line parameters:
>javac main.java
>java main <input_file> <load factor of Part1> <load factor of Part2> <search key>

```
E_Code NRIC Phone
TFTBYS H22779228 85670570
XDVSTY W14372211 96841548
ZLOTNS L13126691 87129676
XYOQDE U62261805 82247950
ELJMNX Y63331209 82001809
JNBAWS J53583275 90979102
DHAOJG X25469023 82714659
EVTXDF E55926940 83476729
VXZFTH V23776523 99672106
QGKXSN M98243414 83537365
XVEWRT B54062674 85783856
UHBOPB V47790511 90161268
FIYTLF W79734515 86759716
XMBZFY V81952780 80721160
ZWJUBS J72079032 85305979
DNDOHO U03927006 92779900
LZOMJS K22484920 81785172
EGZMIA K56527550 88208777
JEGYUC H36565825 84237493
NFVLMW S63626408 81810304
```

Figure 4: Sample Input

5 Grading and Evaluation

- Your work will be graded over a maximum of 100 points.
- There are four items you have to do in this experiment and the contribution of each item to your total score will be partial according to the grading policy stated below.

Correct Hash tables	25p
Separate Chaining	25p
Linear Hashing	25p
Double Hashing	25p

- Your code will be tested with different load factors and search keys. And one output file will be expected as output file that consists the result of each part.
- Your programs are going to be tested on the dev machine.

Usage example:
>javac main.java
>java main d20.txt 1 0.5 86759716

Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- The output of your program will be graded automatically. So create your outputs according to the shared sample output file.
- Write READABLE SOURCE CODE block
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/fall/bbm203) and you are supposed to be aware of everything discussed in Piazza.

```
d20,LF=1,LF2=0.5,86759716
PART1
[Chain 0]: 80721160---->92779900
[Chain 1]: Null
[Chain 2]: 90979102
...
...
[Chain 14]: Null
[Chain 15]: Null
[Chain 16]: 87129676---->85783856---->86759716
[Chain 17]: 88208777
[Chain 18]: Null
[Chain 19]: 82714659---->85305979
PART2
Hashtable for Linear Probing
[0]--->80721160
[1]--->null
...
...
[36]--->87129676
[37]--->86759716
[38]--->null
[39]--->null
Hashtable for Double Hashing
[0]--->80721160
[1]--->86759716
[2]--->null
[3]--->null
...
...
[36]--->87129676
[37]--->null
[38]--->null
[39]--->null
SEPARATE CHAINING:
Key found with 3 comparisons
CPU time taken to search = 3500.0 ns
LINEAR PROBING:
Key found with 2 comparisons
CPU time taken to search = 9300.0 ns
DOUBLE HASHING:
Key found with 2 comparisons
CPU time taken to search = 7300.0 ns
```

Figure 5: Sample Output

- You will use online submission system to submit your experiments. https://submit.cs.hacettepe.edu.tr/ Deadline is: 23:59 pm. No other submission method (email or etc.) will be accepted. Do not submit any file via e-mail related with this assignment.
- File hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system). You must submit your work with the file hierarchy stated below:

  <Assignment2.zip>/(Required)

  →src/(Required)

  →src/main.java(Required)

  →*.java(optional)

Policy

All work on assignments must be done **individually** unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an **abstract** way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) **will not be tolerated**. In short, turning in someone else's work (from internet), in whole or in part, as your own will be considered **as a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.