Seoul National University

Data Structure

Spring 2023, Kang

Programming Assignment 4: Searching (Chapter 9)

Due: June 7th, 23:59, submit at eTL

**Reminders**

* The points of this homework add up to 100.
* Like all homework, this has to be done individually.
* Lead T.A.: Jeongyoung Lee ([jklist91@gmail.com](mailto:jklist91@gmail.com))
* Write a program in Java **(JDK 17)**.
* **Do not use** Java Collection Framework or any third-party implementation from the Internet.

# How to submit the programming assignment

## You should fill in “HashTable.java” and “Main.java” of “PA04” java project.

## Create a ***JAR*** file including ‘src’ folder that contains your sources files.

### We will run your ***Main*** class in the JAR file to grade your programming assignments. Before submitting the JAR file, make sure that your Main class in the JAR file works correctly.

### You must obey the I/O specification of the programming assignment, and rules for the submission of the programming assignment.

### Before submitting, check if your JAR file runs properly in your terminal with the following commands:

### “java -classpath PA\_04\_(studentID).jar Main”.

## (studentID) must be in the following format:

### 2023-12345

# How to grade your programming assignment

## We made a grading machine to automatically grade your programming assignment. The machine will run your program and compare answers and outputs that your program generates for given inputs. If your program cannot generate correct answers for an input file, it will not give you the point corresponding to the input. Our machine will consider the following scenarios:

### (***Accept***) When your program generates exact outputs for an input file, the machine will give you the point of the input.

### (***Wrong* *Answer***) When your program runs normally, but generates incorrect outputs for an input file, including typos, the machine will not give you the point of the input.

### (***Run Error***) When your program does not run, or is terminated suddenly for some reasons, the machine will not give you the point of an input file because it cannot generate any outputs.

### (***Time Limit***) When your program runs over a predefined execution time for an input file, our machine will stop your program, and it will not give you the point of the input. The time limit of the execution is ***5 seconds***.

### 

## We will generate 10 input files and assign 10 points for each input file. For example, if your program gets 9 accepts, and 1 wrong answer by the machine, the total point will be 90 points. Hence, before submitting your programming assignment, please be sure that your program generates correct answers in reasonable time for any input cases.

# Problem

In this assignment, you have to implement a Hash Table with quadratic probing for collision resolution. The tasks are described as follows:

* 1. **Hashing**

The hash table has 523 slots and the hash function is defined by

.

* 1. **Collision resolution policy**

The probe function is defined as for constants and , hence the value in the probe sequence is computed as follows:

.

For example, consider . If , then the probe sequence for is 10, 11, 14, 19, and so on. , , and should be given when you create a hash table. Note that the collision resolution policy is always applied; in particular, when there is no collision.

* 1. **Deletion**

Assume that 1) , and 2) , , and were inserted in that order. Suppose you deleted from the hash table setting it back to . When you search for or , you will find that and , which leads to the exception “ *or is not in the table*.” To solve the problem, you need to set with a special marker (*tombstone*) when the key is deleted from the table. In this assignment, you have to use as the tombstone.

The operations you need to implement are the following:

* **create**: Create a new hash table that has 523 slots. Also the quadratic probing collision resolution policy has to be defined with given parameters.
* **insert**: Insert the key into the hash table using the hash function and the collision resolution policy.
* **delete**: Delete the key from the hash table. If the key is not in the table, the message “Failed to find ” has to be printed and nothing happens to the table. You have to use the integer as a tombstone when you delete the key.
* **search**: Find the index of the key in the hash table. If the key is not in the table, print the message “Failed to find ”.
* **maxProbe**: Return the maximum number of probes for an insertion. For example, if the hash function is , the probe function is , and the hash table is as below, then the maximum number of probes for an insertion is .

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

Here are some clarifications:

* The hash table has 523 slots, and the hash function is .
* All keys are positive integers, and we never insert any duplicated keys. Therefore, you should not implement any duplication-check process which would lengthen the overall probe sequences.
* The probe function is defined as .
* The tombstone for deletion is the integer .
* You may ignore the case when the probe sequence never ends.
* You may ignore the case when .

# Specification

## create

|  |
| --- |
| Function |
| void create(int c1, int c2, int c3) |
| Description |
| * This function creates a new hash table with quadratic probing for collision resolution (). * (c1), (c2), and (c3) are non-negative integers. * You may ignore the case when (c1)=(c2)=0 mod 523. * This function is called only once at the beginning. |

## insert

|  |
| --- |
| Function |
| void insert(int key) |
| Description |
| * This function inserts (key) into the hash table according to the defined collision resolution policy. * Duplicated keys are not allowed for insertion. |

## delete

|  |
| --- |
| Function |
| void delete(int key) |
| Description |
| * This function deletes (key) from the hash table. * If (key) is not in the table, the message “Failed to find (key)” has to be printed and nothing happens to the table. * The tombstone for deletion is -1. |

## search

|  |
| --- |
| Function |
| void search(int key) |
| Description |
| * This function finds the index of the (key). * If (key) is not in the table, the message “Failed to find (key)” has to be printed. |

## maxProbe

|  |
| --- |
| Function |
| void maxProbe() |
| Description |
| * This function returns the maximum number of probes for an insertion in the table. |

# I/O Specification

## create

|  |  |
| --- | --- |
| Input Form | Output Form |
| create (c1) (c2) (c3) | (No output) |
| Description | |
| * (c1), (c2), and (c3) are constants for the collision resolution policy. * There is no output for this input. * “create” is given only once at the beginning. | |
| Example Input | **Example Output** |
| create 3 7 0 |  |

## insert

|  |  |
| --- | --- |
| Input Form | Output Form |
| insert (key) | INSERT: (key), INDEX: (index) |
| Description | |
| * (key) is a positive integer. * (index) is an index of the table where the (key) is inserted. | |
| Example Input | **Example Output** |
| insert 523 | INSERT: 523, INDEX: 0 |

## delete

|  |  |
| --- | --- |
| Input Form | Output Form |
| delete (key) | DELETE: (key), INDEX: (index) |
| Description | |
| * (key) is a positive integer. * (index) is an index of the table where the (key) is deleted. * If there is no such (key) in the table, the message “Failed to find (key)” has to be printed and nothing happens to the table. | |
| Example Input | **Example Output** |
| delete 523 | DELETE: 523, INDEX: 0 |
| delete 523 | Failed to find 523 |

## search

|  |  |
| --- | --- |
| Input Form | Output Form |
| search (key) | SEARCH: (key), INDEX: (index) |
| Description | |
| * (key) is a positive integer. * (index) is an index of the table where the (key) was found. * If there is no such (key) in the table, the message “Failed to find (key)” must be printed. | |
| Example Input | **Example Output** |
| search 1 | SEARCH: 1, INDEX: 1 |
| search 4 | Failed to find 4 |

# maxProbe

|  |  |
| --- | --- |
| Input Form | Output Form |
| maxProbe () | Maximum number of probes: (int) |
| Description | |
| * There is no input for this function. * Output number should be an int type. | |
| Example Input | **Example Output** |
| maxProbe | Maximum number of probes: 1 |

# Sample Input and Output

The grading machine expects the sample output for the given sample input. Hence, for the sample input shown below, your program should print the same lines in the sample output. If your program prints differently from the sample output, the grading script will mark as wrong for the respective input line.

|  |  |
| --- | --- |
| **Sample Input** | **Sample Output** |
| create 1 1 1  insert 100  insert 200  insert 1000  insert 2000  search 100  search 200  search 1000  search 2000  delete 100  search 100  search 200  maxProbe | INSERT: 100, INDEX: 101  INSERT: 200, INDEX: 201  INSERT: 1000, INDEX: 478  INSERT: 2000, INDEX: 432  SEARCH: 100, INDEX: 101  SEARCH: 200, INDEX: 201  SEARCH: 1000, INDEX: 478  SEARCH: 2000, INDEX: 432  DELETE: 100, INDEX: 101  Failed to find 100  SEARCH: 200, INDEX: 201  Maximum number of probes: 1 |