Seoul National University

Data Structure

Spring 2025, Kang

Programming Assignment 4: Searching (Chapter 9)

Due: June 17, 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.: Jaehyeon Choi (snuds.ta@gmail.com)
- Write a program in Java **(JDK 21)**.
- **Do not use** Java Collection Framework or third-party implementation from the Internet.

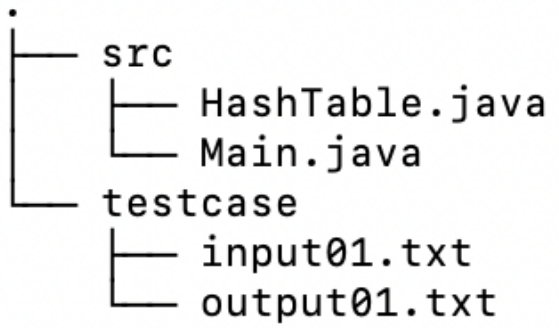
Importance

- Use JDK 21.0.7, as this is the version used for evaluation. If your code is compiled or executed under a different JDK version, run errors may result in 0 points.

1. How to submit the programming assignment

1) Fill in the provided skeleton code with your own answer.

2) Compress your codes as 'PA04_(studentID).zip' file. The structure of your 'PA04_(studentID)' file should look like follows:
   - `(studentID)` must follow the format "2025-12345".



3) Your code will be compiled and executed in a Linux environment using these commands. Make sure your code runs correctly with the following commands.

```
cd src && javac *.java

java Main < ../testcase/input01.txt > ../testcase/output01.txt
```

4) Submit the zip file to the eTL (http://etl.snu.ac.kr).

2. How to grade your programming assignment

1) We made a grading machine to grade your programming assignment automatically. The machine will run your program and compare the 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 reason, 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***.

2) We will generate 20 input files and assign 5 points for each input file. For example, if your program gets 17 accepts and 3 wrong answers by the machine, the total points will be 85 points. Hence, before submitting your programming assignment, please be sure that your program generates correct answers in reasonable time for any input cases.

3. Problem

In this assignment, you have to implement a Hash Table with quadratic probing for collision resolution.

a. Hashing

The hash table has 127 slots, and the hash function $h(\cdot)$ is defined for any positive integer key k as:

$$h(k) = k \mod 127.$$

b. Collision resolution policy

The probe function is defined as $P(k,i) = c_1 i^2 + c_2 i + c_3$ for constants c_1, c_2, and c_3, hence the i^{th} value in the probe sequence is computed as follows:

$$h(k) + P(k,i) \mod (\text{current\_table\_size}).$$

For example, consider $P(k,i) = i^2$ and current_table_size=127. If $h(k_1) = 10$, then the probe sequence for k_1 is 10, 11, 14, 19, and so on. c_1, c_2, and c_3 should be given when you create a hash table. For the sake of simplicity, the collision resolution policy is always applied; in particular, $i = 0$ when there is no collision.

c. Deletion

Consider the following example: assume that $h(k_1) = h(k_2) = h(k_3) = index$, and k_1 was inserted first, then k_2, then k_3. After that, you deleted k_1 from the hash table setting it back to $null$. Later when you search for k_2 or k_3, you will find that $h(k_2) = h(k_3) = index$ and $table[index] = null$, which leads to the exception "k_2 *or* k_3 *is not in the table*." To solve the problem, you need to set $table[index]$ with a special marker (*tombstone*) when the key is deleted from the table. In this assignment, you have to use -1 as the tombstone.

The operations you need to implement are as follows:

- **create**: Create a new hash table that has 127 slots, all initialized to 0 (empty). Also the quadratic probing collision resolution policy has to be defined with given parameters.
- **insert**: Insert a 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 key" has to be printed and nothing happens to the table. You have to use the integer -1 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 key".
- **rehash**: Resize the table to the smallest prime number $\geq$ 2×(current table size), and re-insert all active keys (ignore tombstones).
- **cleanup**: Remove all tombstones by resetting every slot to 0 and re-inserting all active keys.

Here is the clarification:

- All keys are positive integers, and no duplicate key will ever be inserted. Therefore, do not implement any extra duplication-check step that would lengthen probe sequences.
- You may ignore the case when the probe sequence never ends.
- You may ignore the case when $c_1 = c_2 = 0 \ mod\ 127$.
- During *rehash* or *cleanup*, active keys should be re-inserted in the same order as they were originally inserted.

4. Specification

1) create

Function
`void create(int c1, int c2, int c3)`
Description
- This function creates a new hash table with quadratic probing for collision resolution ($\boldsymbol{P(k,i) = c_1 i^2 + c_2 i + c_3}$). - $\boldsymbol{c_1}$, $\boldsymbol{c_2}$, and $\boldsymbol{c_3}$ are non-negative integers. - You may ignore the case when $\boldsymbol{c_1 = c_2 = 0 \ (\mathrm{mod}\ 127)}$.

2) 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. - If the slot is empty (0) or marked with a tombstone (-1), store the *key* in that slot and print "INSERT: *key*, INDEX: *index*". - After inserting *key*, compute *load factor* = (number of slots holding a positive key) / (current table size) and, if *load factor* >= 0.7, perform rehash.

3) delete

Function
`void delete(int key)`
Description
- This function deletes *key* from the hash table using the same quadratic-probing policy. - If *key* is not in the hash table, the message "Failed to find *key*" has to be printed and nothing happens to the table. - The tombstone for deletion is -1. - After deleting *key*, compute the *tombstone ratio* = (number of slots holding *-1*) / (current table size), and if *tombstone ratio*>=0.2, perform cleanup.

4) 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.

5) rehash

Function
`void rehash()`
Description
- Compute *newSize* = smallest prime number $\geq 2\times$(current size). - Gather all active keys (>0), ignoring slots with 0 and -1. - Allocate a new table array of length *newSize*, and keep the same c_1, c_2, c_3. - Re-insert each key from the temporary list into the new table, preserving the original insertion order. - Replace the old table with the newly built array and print “Rehashed to <newSize>”.

6) cleanup

Function
`void cleanup()`
Description
- Collect all active keys (>0), ignoring slots with 0 or -1. - Reset every slot in the current table array to 0. - Re-insert each key into the table in the original insertion order and print “Cleanup Done”.

5. I/O Specification

1) create

Input Form	Output Form
`create (c1) (c2) (c3)`	`(No output)`
Description	
- (c1), (c2), and (c3) are non-negative integers for the collision resolution policy. - There is no output for this input.	
Example Input	**Example Output**
`create 3 7 0`	

2) insert

Input Form	Output Form
`insert (`*`key`*`)`	`INSERT: (`*`key`*`), INDEX: (`*`index`*`)`
Description	
- (*key*) is a positive integer. - (*index*) is the slot index where the *key* is inserted.	
Example Input	**Example Output**
`insert 127`	`INSERT: 127, INDEX: 0`

3) delete

Input Form	Output Form
`delete (`*`key`*`)`	`DELETE: (`*`key`*`), INDEX: (`*`index`*`)`
Description	
- (*key*) is a positive integer. - If (*key*) exists in the table, set its slot to -1 (tombstone) and print "DELETE: (*key*), INDEX: (*index*)", where (*index*) is the slot from which (*key*) was removed. - If (*key*) is not in the table, print "Failed to find (*key*)" and make no changes to the table.	

Example Input	Example Output
`delete 127`	`DELETE: 127, INDEX: 0`
`delete 127`	`Failed to find 127`

4) 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)" has to be printed.

Example Input	Example Output
`search 1`	`SEARCH: 1, INDEX: 1`
`search 4`	`Failed to find 4`

5) rehash

Input Form	Output Form
`rehash`	`Rehashed to (`*`newSize`*`)`
Description	

- There is no input for this function.
- Print "Rehached to (*newSize*)", where (*newSize*) is the smallest prime number >= 2 x (current size).

Example Input	Example Output
`rehash`	`Rehashed to 257`

6) cleanup

Input Form	Output Form
`cleanup ()`	`Cleanup Done`
Description	
- There is no input for this function. - Print “Cleanup Done.”	
Example Input	**Example Output**
`cleanup`	`Cleanup Done.`

6. Sample Input and Output

The grading script expects the sample output for the given sample input. Hence, for the sample input, your program should print the same lines in the sample output shown below. 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 0 0`
`insert 10`
`insert 137`
`insert 254`
`search 10`
`search 137`
`search 255`
`delete 10`
`search 137`
`quit`
	`INSERT: 10, INDEX: 10`
	`INSERT: 137, INDEX: 11`
	`INSERT: 254, INDEX: 0`
	`SEARCH: 10, INDEX: 10`
	`SEARCH: 137, INDEX: 11`
	`Failed to find 255`
	`DELETE: 10, INDEX: 10`
	`SEARCH: 137, INDEX: 11`