CSci 1933 Lab 11

December 01, 2015

1. Introduction

In class you have been learning about hash table and how they are implemented. In this lab you will be applying your knowledge to write a hash table `lookup` method, a `remove` method and implement the `rehashing` logics.

2. Your task

To complete this lab, you will do the following:

- In IntelliJ, create a new Java project named lab11.
- Import the following files into your lab11 project.
  - Dictionary.java :A simple dictionary interface. Place this in the src/ directory.
  - MyHashTable.java :A hash table implementation of the dictionary interface. Place this in the src/ directory.
  - MyKey.java A class which wraps ints, allowing them to serve as keys in a hash table. Place this in the src/ directory.
  - TestHashTable.java A set of JUnit tests for this lab. Place this in the src/ directory.
- Add JUnit to your project in the same way as you have done in your previous projects.
- Complete the `lookup()` method in `MyHashTable`.
  - The `lookup` method takes one key as its parameter. Your task is to search through the entries in the hash table and return this key's corresponding value if it is found. If the key is not found, return null.

- ➢ Recall what you have learned about maps and hash tables in class and decide the most appropriate method for searching through the keys before proceeding.
- ➢ You may find it helpful to look at the `insert` method for inspiration. This method must check for the existence of a key in the hash table before deciding whether to add it as a new entry.

❖ Implement the `remove(key)` method in `MyHashTable`.

- ➢ The `remove` method takes one key as its parameter. Your task is to search through the entries in the hash table and return this key's corresponding value and remove the key from the hash table if it is found. If the key is not found, return null.

❖ Implement the rehashing logics in `MyHashTable`:

- ➢ declare and implement a `needRehash` method, which returns true when the load factor of the hash table is greater than or equal to 1 (false otherwise). You may need to modify `insert` and `remove` to keep track of current entries in `MyHashTable` in order to efficiently calculate load factor (otherwise you may need to iterate over complete `MyHashTable`).
- ➢ at the beginning of the `insert` method, call the `needRehash` method to check if the load factor is so big that the size of the hash table needs to be enlarged. If rehashing is needed, call the function `rehash`.
- ➢ declare and implement the `rehash` method. It should increase the size of the hash table to the smallest prime number that is at least twice bigger than the current size (take advantage of the `getNextPrime` method of the `MyHashTable`). Then it needs to add all of the existing entries to the new hash table by calling `insert` on each of the entries.

❖ Run the unit tests in TestHashTable.java to check whether you have implemented your methods correctly.

------Solution will be available in last 15 mins------