UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Practical 10

Release: Monday 28 June 2021, 18:00
Deadline: Friday 02 July 2021, 18:00

PLAGIARISM POLICY

UNIVERSITY OF PRETORIA

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.library.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the *Library* quick link, and then choose the *Plagiarism* option under the *Services* menu). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

Objectives

After completing this practical you would have implemented your own *dynamic* Hash Table, using a *folding* hash function and *chaining* to resolve collisions.

Instructions

Complete the task below. Certain classes have been provided for you in the *files* zip archive of the practical. You have also been given a main file which will test some code functionality, but it is by no means intended to provide extensive test coverage. You are encouraged to edit this file and test your code more thoroughly. Remember to test boundary cases. Upload **only** the given source files with your changes in a compressed archive before the deadline. Please comment your name **and** student number in at the top of each file.

Hashing

In previous practicals searches were performed by comparing keys with each other to determine if a key has been stored and to retrieve it's associated data. Another approach is to use a hash function to derive the index of a key in a hash table. This can ideally reduce the search time to at least O(1) regardless of the number of elements stored. See chapter 10 in the textbook for more information on Hashing.

Hash function

You are required to implement a hash function using the folding technique, as described in section 10.1.2 in the textbook. The key should be interpreted as an ASCII string. The hash function should XOR each of the characters together. The *Modulo operation* should then be applied to make sure the result is in range of the table size.
Example with multiple characters: h(abc) = (a XOR b XOR c) mod TSize.
Example with a single character: h(a) = (a) mod TSize.
The binary representation of the ASCII character should be considered.

Collision resolution

For collision resolution, *chaining* should be used as described in section 10.2.2 in the textbook. Each position in the table will be associated with a linked list data structure. If a key collides with another key, the value associated with the latest key should be appended to the chain.

Rehashing

When the hash table exceeds a certain number of elements, the size of the table should be doubled and all elements rehashed. The *loadFactor* is defined as the average length of the chains in the hash table. After inserting a key, if the average chain length is strictly greater than the specified *loadFactor*, rehashing should take place.

Task 1: Implement a DynamicHashMap [38]

Implement the following methods in the *DynamicHashMap* class according to the given specification. You are not allowed to import Java's built in HashMap or HashTable. Please see the comments in the given source code for additional implementation details.

`int hash(String key)`

Calculate and return the hash of the given key *key*.

`Integer get(String key)`

Return the associated value of the given key *key*. If no value has been associated, return null.

`Integer put(String key, Integer value)`

Associate the given value *value* with the given key *key* in the hash map. Return any previously associated value. If no value has been previously associated, return null.

`Integer remove(String key)`

Remove the given key *key* and the value associated with the key from the hash map. Return the associated value or null if no value has been associated.

Submission

You need to submit your source files on the Assignment website `https://ff.cs.up.ac.za/`. All tasks need to be implemented (or at least stubbed) before submission. Place **ONLY DynamicHashMap.java** file in a zip or tar/gzip archive (you need to compress your tar archive) named uXXXXXXXX.zip or uXXXXXXXX.tar.gz where XXXXXXXX is your student number. You have 4 days to complete this practical, regardless of which practical session you attend. Upload your archive to the *Practical 10* slot on the Assignment website. Submit your work before the deadline. No late submissions will be accepted.