UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Practical 10

Release: Monday 20 May 2019, 18:00
Deadline: Tuesday 21 May 2019, 18:00

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 zip archive before the deadline. Please comment your name **and** student number in at the top of each file.

The following files will be overwritten by Fitchfork:

- *Main.java*
- *makefile*

Your source will be compiled using *javac *.java* to make provision for your own data structures. If your code writes to STDOUT or STDERR during execution of the test cases you may be awarded a mark of 0.

Introduction

In previous practicals searches were performed by comparing keys with each 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. The implementation of this data structure is left up to you to implement. 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, if the average chain length is strictly greater than the specified *loadFactor*, rehashing should take place.

Task: [47]

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 more implementation details.

The number of marks related to testing a specific function is indicated to the left of each function. Due to the nature of black box testing it may not be possible to test each function in isolation. For example if there is an error in your *hash* function, other functions will also return incorrect results. Best effort is made to test functions in a bottom-up fashion.

[0] `DynamicHashMap(int tSize, Double loadFactor)`

Create a new empty hash map of size *tSize* and a *loadFactor* threshold.

[6] `int hash(String key)`

Return the hash of the given key.

[4] `Integer get(String key)`

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

[26] `Integer put(String key, Integer value)`

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

[8] `Integer remove(String key)`

Remove the associated value with the given key from the hash map. Return the value associated prior to removal or null if no value has been associated.

[**0**] `int tableSize()`

Return the size of the table, as set in the constructor. This function is only used for converting the hash map to a string.

[**3**] `Integer[] chain(int index)`

Return the chain with the given index. Return an empty array if the chain does not contain any values.

Submission

You need to submit your source files on the Assignment website (assignments.cs.up.ac.za). All methods need to be implemented (or at least stubbed) before submission. Place all the source files including a makefile in a zip or tar/gzip archive named uXXXXXXXX.zip or uXXXXXXXX.tar.gz where XXXXXXXX is your student number. There should be no folders in your archive. You have 24 hours to finish this practical, regardless of which practical session you attend. Upload your archive to the *2019 Prac 10 - Tuesday* slot on the Assignment website. Submit your work before the deadline. **No late submissions will be accepted!**