Gebze Technical University

CSE222 – Data Structures and Algorithms

HW6 – Part4

Subject : Hashtable

Nevzat Seferoglu

171044024

*Problem Description :

Implement KWHashMap interface in the book using the following hashing methods to organize hash tables:

a) In chaining is a technique, each table element references a linked list that contains all the items that hash to the same table index. Use chaining with binary tree instead of linked list.

b) In open addressing, if the table entry is full, subsequent probe locations are calculated using linear probing (hash(x) +i) or quadratic probing (hash(x) + i^2) methods. Use an alternative probing method called double hashing where a second hash function is used to calculate subsequent probe locations (hash(x) + i*second_hash(x)).

Compare the performance of your implementations with HashTableOpen and HashTableChain classes in the book. Note that you should generate large maps of several entries and perform several random operations to compare the performance of the methods. Report your experimental evaluation using tables and graphs.

*Solution Approach :

I only implement Hashtable with chaining technique. There was binary search tree in it. There were some operations that I use in BinaryTree and BinarySearchTree.

*Each duplicate hash belongs to same tree.

Hash table. Collision resolution by chaining (closed addressing)

Chaining is a possible way to resolve collisions. Each slot of the array contains a link to a singly – linked list containing key-value pairs with the same hash. New key-value pairs are added to the end of the list. Lookup algorithm searches through the list to find matching key. Initially table slots contain nulls. List is being created, when value with the certain hash is added for the first time.

***Note** = In this explanation , linked list was used. But it can be any data structure to chain the hashes. Like this assignment , binary tree.

****Note = Graphs that attached to the prepared with random index for different sizes. Linearity can should not be expected.**

Test Cases :

Scenario	Test Data	Expected Result	Actual Result	Pass	Fail

put(1 , "A");
put(1 , "B");
put(1 , "A");
put(1 , "C");
	1, “A”
	1, “B”
	1, “C”
		Size : 1
		Current hashtable :
		{1=C}
			Size : 1
			Current hashtable :
			{1=C}
				Pass	

put(2 , "A");
put(2 , "B");
put(2 , "A");
put(2 , "C");
	2, “A”
	2, “B”
	2, “C”
		Size : 2
		Current hashtable :
		{1=C, 2=C}
			Size : 2
			Current hashtable :
			{1=C, 2=C}
				Pass	

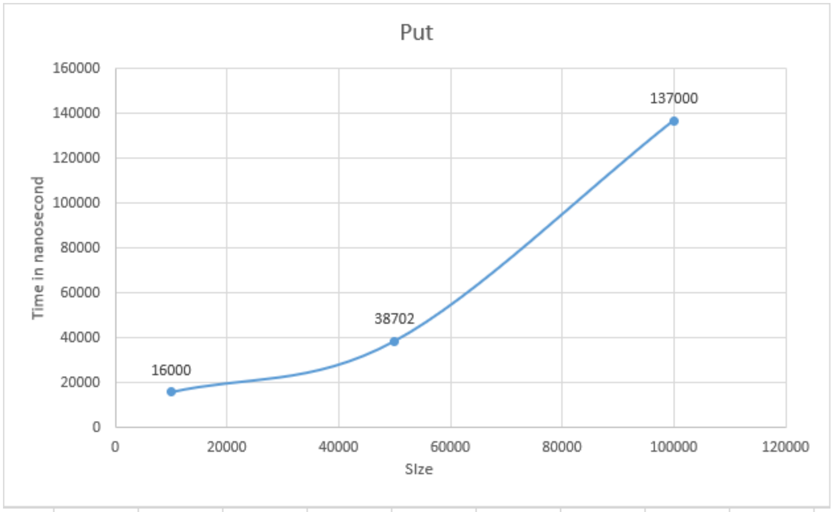
put(3 , "A");
put(3 , "B");
put(3 , "A");
put(3 , "C");
	3, “A”
	3, “B”
	3, “C”
		Size : 3
		Current hashtable :
		{1=C, 2=C, 3=C}
			Size : 3
			Current hashtable :
			{1=C, 2=C, 3=C}
				Pass	

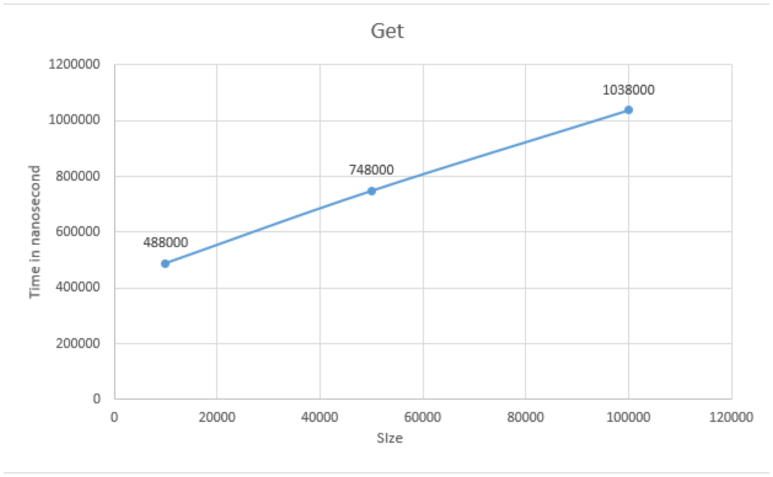
-> get(1) -> get(2) -> get(3)	get(1) = C get(2) = C get(3) = C	get(1) = C get(2) = C get(3) = C	get(1) = C get(2) = C get(3) = C	Pass	
remove(1)	1	Size : 2 Current hashtable : {, 2=C3=C}	Size : 2 Current hashtable : {, 2=C , 3=C}	Pass	
remove(2)	2	Size : 1 Current hashtable : {3=C}	Size : 1 Current hashtable : {3=C}	Pass	
remove(2)	3	Size : 0 Current hashtable : {}	Size : 0 Current hashtable : {}	Pass	

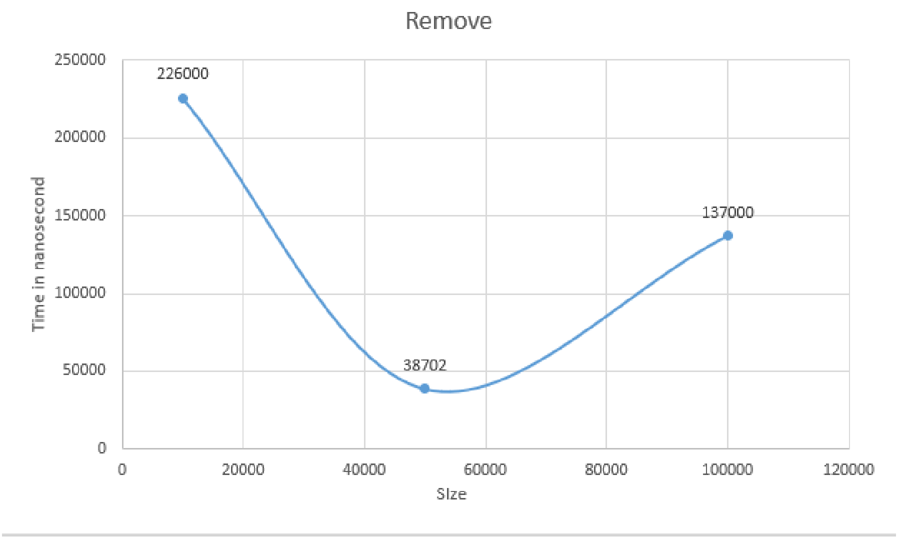
Time issues :

Experiment has been for each method in Hashtable. Calculation has been made for different size such as 10.000 , 50.000 , 100.000.

T01	10.000 hash table analyzing result : -> get(randomKey) = 1671575112 ------------------------- 10K size analyzing... Creating runtime = 56672000 ns get( ) runTime = 488000 ns remove( ) runTime = 368000 ns put( ) runTime = 16000 ns				Pass	
T02	-> get(randomKey) = -749392258 ------------------------- 50K size analyzing... Creating runtime = 152715000 ns get( ) runTime = 748000 ns remove( ) runTime = 54000 ns put( ) runTime = 38702 ns				Pass	
T03	-> get(randomKey) = 1124844911 ------------------------- 100K size analyzing... Creating runtime = 183173000 ns get( ) runTime = 1038000 ns remove( ) runTime = 226000 ns put( ) runTime = 137000 ns				Pass	


Put
Time in nanosecond
Size
16000
38702
137000
0
20000
40000
60000
80000
100000
120000
140000
160000


Get
Time in nanosecond
Size
488000
748000
1038000
0
200000
400000
600000
800000
1000000
1200000


Remove
Time in nanosecond
Size
226000
38702
137000
0
50000
100000
150000
200000
250000

*Class Diagram:

*Attached to the assignment file as png.

*Running Command and Result:

*Attached to the assignment file as txt.