Implementation Project

COMP 5408- Advanced Data Structures

Winter 2014

Submitted To

Professor Pat Morin

By

Ferhan Jamal (100 953 487)

Carleton University

Overview

This benchmark compares the memory usage and performance of various data structures. Data Structures which we have tested are –

- HashMap
- LinkedHashMap
- TreeMap
- Trie

Problem Statement:-

We have a large text file which has string phrases along with its frequencies in each line. We will read this file line by line and store each string and its frequencies in a various data structures as proposed above.

Metrics being evaluated:

- Memory requirements
- Runtime for search, addition

What I did as part of this project was - I am generating a large text file which has around 10 million strings and its frequencies in the below format -

hello 100
world 200
abcde 300
jamal 500

(Above data means – hello is a string phrase and 100 is the frequency of that string)

After that I am reading this file line by line and storing it in all the above data structures and measuring how much time it takes to insert the data in those data structures, how much time it takes to randomly lookup strings from those data structures and what is the size in memory after insertion.

So the program which I wrote is configurable and it accepts various parameters and "DataStructureProject" is my main class. Below are the lists of parameters --

- Location: This is the location of file where you want to generate your text file which will have strings and its frequencies in each line.
- Number of String: This parameter means how many strings you want to have in your file. In the above example it has only "4" strings.
- Length: This parameter means, what is the length of each string you want to have. In the above example, all the strings are of length "5".

- Number of Searches: This parameter means how many random searches you want to do on a particular data structure for this test.

There are couples of ways to run this program.

Running as Runnable Jar:-

If you are running it as a runnable jar from the command line, then use below command where the jar file is there -

```
java -XX:-UseTLAB -jar datastructureproject.jar C:/DataStructureProject/words.txt 10000 6 5000
```

What the above command will do is - It will create the "words.txt" file in this folder for you "C:/DataStructureProject" automatically in which total number of strings will be "10000" and each string length will be "6". And total number of random searches you are planning to do is "5000" on various data structures.

If you want to generate a new file, then use this -

```
java -XX:-UseTLAB -jar datastructureproject.jar C:/DataStructureProject/words1.txt 1000 6 500
```

Now what the above command will do is - It will create a new "words1.txt" file in this folder for you "C:/DataStructureProject" automatically in which total number of strings will be "1000" and each string length will be "6". And total number of random searches you are planning to do is "500" on various data structures.

Running via Eclipse:-

Just import my project as it is in the eclipse. Now right click on the project and click Run Configurations for "DataStructureProject". And in the Program Arguments text box, type this -

```
C:/DataStructureProject/words.txt 10000 6 5000
```

It means exactly same as I have described above. And in the VM Arguments section, type this -

```
-XX:-UseTLAB
```

NOTE: PLEASE SPECIFY THE INPUTS IN THE ORDER IT IS SHOWN ABOVE.

Result

For input parameters

Number of Strings we have generated: 3000000

Length of each Strings/Words: 6

Number of random Searches we did: 2000000

	Insertion Time**	**Read Time****	**Size in Memory after insertion****
HashMap	1347 ms	~ 1 ms	509737632
LinkedHashMap	2268 ms	~ 2 ms	320021008
TreeMap	119 ms	~ 41 ms	545339976
Trie	3255 ms	~ 10696 ms	437807160

**Times are given in milliseconds and memory usage is given in bytes.

So with this test what I noticed was –

- HashMap lookup is pretty fast but the memory usage is slightly high.
- LinkedHashMap lookup and memory usage is similar to HashMap but insertion time is pretty high as compared to HashMap.
- TreeMap seems to be best for this problem, Insertion time is pretty fast and memory usage is not very high as well.
- But Trie is not good solution for this problem as Trie can save memory if the keys are long and share many common prefixes but in this particular problem, strings do not share the same prefixes so insertion and read time is pretty high. But most importantly it is good in saving the memory which means Trie are much more space efficient as compared to other data structures. Insertion and Read time is pretty high.

In short HashMap is the fastest lookup structure as compared to other data structures.