Mandatory Project 2: Skip List

Project Report

Niveditha Varadha Chandrasekaran
10/23/2016

CONTENTS

1. ABSTRACT

The aim of this project is to implement SkipList and compare its performance with JAVA's TreeMap.

2. PROBLEM STATEMENT

This project aims at implementing Skip List with following operations:

a. Add – Add new items to the skip list
b. Ceiling – Find least element from the list that is greater than or equal to the given element
c. FindIndex – Find the element at the given index position
d. First – Find the first element of the list
e. Last – Find the last element of the list
f. Floor – Find the greatest element from the list that is lesser than or equal to the given element
g. Remove – Remove the given element from the list if it exists in the list
h. Contains – Check if the given element is present in the list
i. Rebuild – Rebuild the list back into a perfect Skip List

The performance of the implemented Skip List is compared with JAVA's TreeMap.

3. METHODOLOGY

The elements in a Skip List are stored in sorted order, as a linked list of nodes. Each skip list entry has an array of next pointers, where next[i] points to an element that is roughly 2i nodes away from it. The next array at each entry has random size between 1 and maxLevel, the maximum number of levels in the current skip list. Ideally, maxLevel ≈ log n. Each skip list has dummy head and tail nodes, both of maxLevel height, storing null value. Iterating through the list using next[0] will go through the nodes in sorted order. It also has width[i] which stores the number of crossed by each next[i] pointers.

a. Add Operation:
A new item is added to the list in the position based on its value in sorted order. A random level is chosen for this entry and is added to the list and the corresponding next pointers and widths of the entries are updated. If the given item is already present in the list then it is replaced.

b. Remove Operation:
The given item is removed from the list if it is present by finding the position of the item in the list. Once an item is removed the corresponding next pointers and widths of the list entries are updated.

c. FindIndex Operation:
Given an index (starts from 0) the element at the corresponding index position is returned. This is done by maintaining the length of the gaps for each next[i] pointer at each node of the skip list in width[i].

d. Rebuild Operation:

This operation is required to rebuild a perfect Skip List whenever the size of the list exceeds or is less than 2^maxlevel by a certain percentage (ex: 50 %). An array of empty Skip list entries with perfect levels is created first. Then this array is populated with the actual values of the list and the next pointers pointing to the corresponding next elements with that level. The widths of the nodes (no of elements that can be crossed by the next pointers are also calculated. This is done by maintaining a global pointer (an array of entries with size of maxLevel). The pointer has the nodes whose next pointers have to be linked with the current node. Using this pointer the links are formed.

4. DEVELOPMENT PLATFORM

IDE: Eclipse. Version: Mars.1 Release (4.5.1)
Java Version: 1.8
Operating System: Windows 10

5. SAMPLE INPUT/OUTPUT

Sample input:

Add 1
Add 2
Add 13
First
Last
Ceiling 10
Remove 1
Remove 22
End

Sample output:

31
Time: 17 msec.
Memory: 10 MB / 981 MB.

6. ANALYSIS OF RESULT

Running Time analysis of Skip List on the inputs:

Given Skip List			
Input	**Running time (ms)**	**Memory Used(MB)**	**Memory Total(MB)**
a1.txt	31	10	981
a2.txt	42	10	981
a3.txt	66	10	981
a4.txt	144	10	981
a5.txt	296	20	981
a6.txt	1160	66	981
a7.txt	1514	122	981
a8.txt	9258	199	1237
a9.txt	10413	248	1237
b1.txt	25	10	981
b2.txt	44	10	981
b3.txt	67	10	981
b4.txt	257	10	981
b5.txt	468	20	981
b6.txt	1135	71	981
b7.txt	1696	138	981
b8.txt	10974	289	1237

Performance analysis of SkipList versus TreeMap:

A set of randomly generated inputs are first added to TreeMap and SkipList and then the different input sizes of elements are added to the Skip List and Tree Map and the following running times are achieved.

- put( ) – For adding elements to TreeMap
- add( ) – For adding elements to SkipList

Add Operation		
	Running time (ms)	
Input	**SkipList**	**TreeMap**
1000	2	4
5000	10	6
10000	150	90
50000	91	174
100000	269	215

First a set of randomly generated inputs are added to the SkipList and TreeMap and then the operations add, remove and contains are performed for an element and the following results are obtained.

Performance analysis		
Operation	**Running time (ms)**	
	SkipList	**TreeMap**
First run		
add	1	1
remove	1	1
contains	2	1
Second run		
add	0	1
remove	0	0
contains	0	0
Third run		
add	1	1
remove	1	1
contains	1	1

The above performance analysis shows that Skip List is as efficient as a JAVA's implementation of TreeMap (just a very marginal difference in running time).

7. CONCLUSION

A Skip List can provide all the operations that are provided by a TreeMap and is a very efficient Data Structure. The performance of Skip list is as good and equivalent to the performance of JAVA's TreeMap just with very low marginal differences.

8. REFERENCES

- https://en.wikipedia.org/wiki/Skip_list