**PROJECT REPORT**

**Project By:**

Name: Swapnil Shah

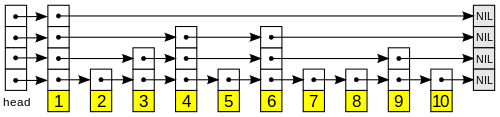
NetId: sjs140230

**1.ABSTRACT**

The project was to implement skip list, a data structure, which allows faster searches. The elements are maintained in an ordered sequence and the speed to the data structure is provided by the way it is implemented. Basic operations were implemented on the skip list. The final part of this project was to compare the performance of skip list with TreeSet in Java.

**2.SkipList Definition**

Skip list is a data structure that allows fast search within an ordered sequence of elements. Fast search is made possible by maintaining a linked hierarchy of sub-sequences, each skipping over fewer elements. Searching starts in the sparsest subsequence until two consecutive elements have been found, one smaller and one larger than the element searched for.



Skip list is implemented with nodes that have a data element and an array of pointers which point to various elements in list. The list has a dummy head and tail node with value as null.

Skip list is similar to linked list, with additional speed. This speed is because of the array of pointers to various nodes, unlike a single pointer to the next node in singly linked list or pointer to previous and next node in doubly linked list.

|  |  |  |
| --- | --- | --- |
| Operation | Time Complexity Skip List | Time Complexity Linked List |
| Insertion | **O(log N)** | **O(N)** |
| Removal | **O(log N)** | **O(N)** |
| Search | **O(log N)** | **O(N)** |
| Enumerate | **O(N)** | **O(N)** |

N is the number of elements in the list.

**3. Problem Statement**

Implement the skip list data structure .Compare its performance with Java's TreeMap.

The interface is given below:

*import java.lang.Comparable;*

*import java.util.Iterator;*

*public interface SkipList<T extends Comparable<? super T>>*

*{*

*void add(T x); // Add an element x to the list*

*T ceiling(T x); // Least element that is >= x, or null if no such element*

*boolean contains(T x); // Is x in the list?*

*T findIndex(int n); // Return the element at index n in the list*

*T first(); // Return the first element of the list*

*T floor(T x); // Greatest element that is <= x, or null if no such element*

*boolean isEmpty(); // Is the list empty?*

*Iterator<T> iterator(); // Returns an iterator for the list*

*T last(); // Return the last element of the list*

*void rebuild(); // Rebuild this list into a perfect skip list*

*boolean remove(T x); // Remove x from this list; returns false if x is not in list*

*int size(); // Number of elements in the list*

*}*

Comparison of Skip lists and tree maps:

Evaluate the performance of skip lists and tree maps on randomly generated

inputs, over the following operations:

Add, Contains, Remove.

**4 . Implementation**

Search :

1. To search an element with data value x, we start with the header node with maximum level.

2. We check the next pointer, if the value pointed by this level’s next pointer is more than x come down by one level on the same node.

3. If value pointed by this level’s next pointer is less than x then jump to that node so that we can skip multiple nodes with value less than x.

4. Search operation returns the search value as well as the previous data nodes that are being used to reach to that position.

Add:

1.We call the search operation and check if the value exists in skiplist.

2. If the value does not exists in the skip list, we can use the previous nodes given by the search operation to find the correct position of the element and adjust the pointers for all the nodes accordingly.

3. Also we need to set the node pointers of all levels of this newly generated nodes.

Remove:

1. First check if the element exists.

2. Search provides if the element exists. If it exists point all the previous pointers to the next pointers of the node, level wise.

Contains :

1. Check whether list contains the given value. Return true is list contains that value or return false.

**5. System Configuration**

Processor: Intel Core I7 (Second Generation)

RAM: 6 GB

HDD: 750 GB

IDE : Eclipse

**6. Test Cases and Comparison of result**

The sample input were run on the program which implements skip list and on java tree maps and the following result were recorded.

|  |  |  |
| --- | --- | --- |
| Input size | SkipList | TreeSet |
| 50 | 24 | 21 |
| 100 | 35 | 31 |
| 200 | 46 | 40 |
| 1000 | 119 | 107 |
| 10000 | 408 | 295 |
| 50000 | 685 | 656 |
| 100000 | 972 | 920 |
| 1000000 | 8135 | 6638 |

All time values indicated are in miliseconds.

**7. Conclusion**

By above result we can conclude that the running time of TreeSet is better than that of skiplist. But the performance is comparable between the two data structures.

Theoretically, skip list have a better running time than that of treeset. This difference in theoretical result and implementation is because of the fact that the skiplist that is constructed during runtime is not a perfect skip list. Plus some functions need to be implemented efficiently. The skip list needs to be periodically rebuilt so that the skip list is a perfect skiplist and this could reduce the running time of skip list and could prove that it is a better data structure than treeset.

**8. References**

http://en.wikipedia.org/wiki/Skip\_list

http://web.stanford.edu/class/cs161/skiplists.pdf

http://openmymind.net/Building-A-Skiplist/