**SkipList Implementation Report**

The **SkipListImpl** class includes following classes-

Node – Holds the node value, array that points to the next nodes at each level and width at each level.

PrevNode – Holds the distance traveled at each level to find an element and an array of previous nodes

**Following functions are implemented –**

**Add** – Insert the node at its respective position in the list and accordingly update the next node links and width for previous and new nodes.

**Remove** – Remove the node with given value and update the links and width of previous nodes acoordingly.

**Find** – find the array of nodes that point to the previous nodes at each level for a given search element.

**FindIndex** – find the element at a given index. To find this width has to be maintained at each level for all nodes. Whenever a node is added or deleted from the list, width is updated.

Following cases are handled as part of **CalculateWidth** method-

Below are the cases I am handling:

1. When the newly added node has level <= the level of previous node returned from find. Here from level 0 to new node's level I assign the previous node width to new node width and updating the previous node width to 1.

2. When the new node level > previous node level, from level 0 to previous node level, I assign its width to new node and updating its width to 1 (same as above). For the levels greater than previous node level,   using the distance array returned from find method above to calculate the previous node's width and accordingly update the new node's width.

3. After the above cases, from the levels greater than new node level to maxlevel i am incrementing the previous node's width by 1.

Similar cases are handled for remove function as well.

**Rebuild** – Whenever the size of skiplist exceeds 2^maxlength, it needs to be resized and rebuilt. Rebuilt includes a function ‘RebuildStructure’ to create a perfect skiplist with the nodes in the middle should have a level 1 less than the maxlevel and so on. In rebuild after the structure is built, the elements are added in sequence and previous nodes widths are updated.

**First** – returns the first element in the list

**Last** – returns the last element in the list

**Floor** – returns the element that is <= given element

**Ceiling** – returns the element that is >= given element

**Iterator** – Returns the Skiplist Iterator which has next(), hasnext() functions that checks for the node’s next element at level 0 and returns it.

**Default values-**

Max level – 10(can be changed)

Rebuild – maxlevel is incremented by 5

**Output and run times for the mp2a and mp2b files using skiplist implementation** –

|  |  |  |  |
| --- | --- | --- | --- |
| Input File | Output Value | Run-Time(msec) | Memory |
| A1 | 117 | 6 | 2 MB / 245 MB |
| A2 | 584 | 8 | 2 MB / 245 MB |
| A3 | 2583 | 10 | 2 MB / 245 MB |
| A4 | 54501 | 22 | 3 MB / 245 MB |
| A5 | 539108 | 71 | 15 MB / 245 MB |
| A6 | 343528 | 187 | 58 MB / 245 MB |
| A7 | 858099 | 266 | 64 MB / 245 MB |
| A8 | 130122 | 1992 | 260 MB / 707 MB |
| A9 | 626281 | 2062 | 309 MB / 706 MB |
| B1 | 36 | 7 | 2 MB / 245 MB |
| B2 | 73 | 8 | 2 MB / 245 MB |
| B3 | 150 | 11 | 2 MB / 245 MB |
| B4 | 721 | 21 | 3 MB / 245 MB |
| B5 | 7097 | 71 | 16 MB / 245 MB |
| B6 | 35313 | 175 | 62 MB / 245 MB |
| B7 | 70918 | 275 | 14 MB / 309 MB |
| B8 | 709447 | 2164 | 350 MB / 706 MB |

**Output and run times for the mp2b files using java TreeMap –**

|  |  |  |  |
| --- | --- | --- | --- |
| Input File | Output Value | Run-Time(msec) | Memory |
| B1 | 36 | 6 | 2 MB / 245 MB |
| B2 | 73 | 8 | 2 MB / 245 MB. |
| B3 | 150 | 10 | 2 MB / 245 MB |
| B4 | 721 | 20 | 3 MB / 245 MB |
| B5 | 7097 | 62 | 14 MB / 245 MB |
| B6 | 35313 | 165 | 56 MB / 245 MB |
| B7 | 70918 | 218 | 41 MB / 245 MB |
| B8 | 709447 | 1517 | 120 MB / 703 MB |

Skiplist implementation functions add, remove and contains run as fast as java’s treemap functions.