**SKIP-LISTS (list structure, not a tree)**

A skip-list is another data structure that can be used as the basis for the NavigableSet or NavigableMap and as a substitute for a balanced tree

It provides O(logn) search, insert, and remove

* In balanced tree structures (AVL, red-black) logarithmic time is worst case logarithmic time.
* In skip-list logarithmic time is average case logarithmic time.

It has the advantage over a Red-Black tree-based TreeSet in that concurrent references resulting from multiple threads are easier to achieve

The concurrency features are beyond the scope of the course, and deal with multiple threads making modifications on a data set

Skip-list is a list of lists

* Each node contains a data element with a key
* The elements in each list are in increasing order by key
* The nodes can contain a varying number of forward links determined by the level of the node
* A level-m node has m forward links
* The level of a new node is chosen randomly in such a way that, for a 4-level skip-list, approximately 50% are level 1 (one forward link), 25% are level 2 (2 forward links), 12.5% are level 3 (3 forward links), and so on

A picture containing text, clock

Description automatically generated

The level of a skip-list is defined as its highest node level, or 4 in this list

The list above is an ideal skip-list (oddth number of elements are level 1); most skip-lists will not have exactly this structure, but will behave similarly

Order of 10 is 2, order of 25 is 1, order of 40 is 4. Elements are part of “order” number of linked lists.

A search always begins in the highest level list (the list with the fewest element)

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Chart, box and whisker chart

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

A picture containing text, outdoor, clock

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

41’i arasaydık yine level 4’ten başlardık. 40’tan büyük o yüzden ileri gidicez. Üste çıktık (40’ın 3. listesi) devamı yok. Üste çıktık, oku takip ettik, 50’ye geldik. Üste çıktık (lowest leveldeyiz), 45 büyük, yani eleman listede yok.

Searching Algorithm

Text

Description automatically generated

At each level we get rid of half of the elements. So running time should be logarithmic.

Because the 1st list searched has the fewest elements, and each subsequent-level list has approximately half as many elements as the current list, the search performance is O(logn), which is similar to that of a binary search

If level of the skip list is 1, performance for the search is linear. This is the worst case.

Insertion

If the search algorithm fails to find the target, it will find its predecessor in the level-1 list, which is the target’s insertion point (we keep predecessor for each level)

While we know the insertion point, we need to determine the level of the new node

The level is chosen at random based on the number of items currently in the skip-list

The random number is chosen with a logarithmic distribution; for a level-4 skip-list

* half the time a level-1 node is chosen
* a quarter of the time a level-2 node is chosen
* and, generally 1/ of the time a level-m node is chosen

Lets say we want to add 41 and its level is 3. So 3 links should be there for 41.

Diagram

Description automatically generated

Diagram

Description automatically generated



Red ones are predecessors. We determine these predecessors during the search.

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Increasing the Height of a Skip-List

A level-m skip-list can hold between and - 1 items

A level-4 skip-list as in our example:

* can efficiently hold up to 15 items ()
* When a 16th item is inserted, the level is increased by 1

If the 36 (item we inserted above) is the 16th item, then its level will be determined between 1 - 5, not between 1 - 4.

**IMPLEMENTING A SKIP-LIST**

Graphical user interface, text

Description automatically generated

m is determined randomly.

Search

Method search() will return an array pred which holds references to the predecessors at each level.

Diagram

Description automatically generated

Text, letter

Description automatically generated

pred[0] is the predecessor at the first level.

for loop iterates 4 times for our example, maxLevel () times for general.

* n <= ---> n+1 <= ---> log(n+1) <= m

while loop iterates not many for each level. It could be as bad as linear or as good as constant. On average, it is constant.

So overall running time is logarithmic.

We make 3 comparisons for 40 when we search for 35 (we compared with 20 1 time, with 30 1 time, with 35 1 time).

Tw(n) = (n) 🡪 all of the elements are at level 1

Tb(n) = (logn) 🡪 for example searching for 40, we have to fill pred so it is not constant. We have to continue for each level

Tavg(n) = (logn) 🡪 since we eliminate almost half of the nodes after each comparison like in bst

On average we obtain good results. This kind of algortihms are randomized algorithms.

Skip-list is a randomized data structure.

Insertion

Diagram, schematic

Description automatically generated

This 2 lines of code is required for all levels of the inserted item (from 0 to level of inserted node).

Determine level of the inserted node:

Text

Description automatically generated

We divide by log2 to have base 2 logarithm.

maxCap =

Completing the Insertion Process

Text

Description automatically generated

If there are 15 elements and we are inserting 16th element.

Performance of a Skip-List

In an ideal skip-list (every element is at the lowest level, every 2nd element is at the upper level, every 4th element is at the 1 upper level, every 8th element is at the 1 upper level…), every other node is at level 1, and every node is at least level m

With this ideal structure, performance matches that of a binary search at O(logn)

By randomly choosing the levels of inserted nodes to have an exponential distribution, the skip-list will have the desired distribution of nodes

However, the nodes are randomly positioned throughout the skip-list --- making the average for search and distribution O(logn)

Worst skip list is all elements are at the highest level or all are only at the first level.

Number of nodes with 1 element is almost n/2 for average. Worst case for the memory is all of them are at the highest level. There are n nodes, number of links is logn. So S(n) = (nlogn) for the worst case.

Best case for the memory is all elements are only at level 1. So S(n) = (n) for the best case.

Average case is almost n/2 elements are at level 1, n/4 of them are at level 2, n/8 of them are at level 3, etc. So (1\*(n/2) + 2\*(n/4) + 3\*(n/8)) :

S(n) = = (n)

Number of nodes at lowest level are very few compared to other level’s elements.