Generic Indexable Skiplist

Karan Panjabi
Computer Science and Engineering
PES University

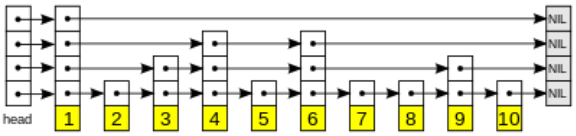
Mayank Agarwal
Computer Science and Engineering
PES University

***Abstract*—**The worst case search time for a sorted linked list is O(n) as we can only linearly traverse the list and cannot skip nodes while searching. Can we augment linked lists to support. O(logn) operations? The solution is Skip lists

Keywords—skip-list, nodes, logn, generic, container, indexable

I. Introduction

Skip lists were invented by William Pugh in 1989. Skip list is an ordered data structure that allows O(log n) search complexity as well as O(log n) insertion complexity in average cases. It's able to do this by 'skipping' over elements. It consists of a base linked list, along with additional lists at different levels.



A perfect skip list with say 16 nodes in base would have 4 nodes one level above, 2 above that and 1 above that. This allows us to search in O(logn) time. But if insertion or deletion takes place, the whole skip lists to be rearranged. The solution to this is randomization. We give 0.5 probability to each node to jump one level more. It can be proved that on average all operations are O(log2n).

To leverage the advantage of quick access as in case of arrays, the list has to be made indexable. The idea is to store for each link the width like below:

```
  1                                10
o---> o-------------------------------------------------------> o    Top level
  1           3              2                    5
o---> o---------------> o---------> o--------------------------> o    Level 3
  1        2        1        2              3              2
o---> o---------> o---> o---------> o---------------> o---------> o    Level 2
  1     1     1     1     1     1     1     1     1     1     1
o---> o---> o---> o---> o---> o---> o---> o---> o---> o---> o---> o    Bottom level

Head  1st   2nd   3rd   4th   5th   6th   7th   8th   9th   10th  NIL
      Node  Node  Node  Node  Node  Node  Node  Node  Node  Node
```

The generic data structure supports a constant bidirectional iterator as in case of STL set, and the indexing operator [] as in case of STL vectors. The value type needs to support less than operator < and equality operator == in order to work with the generic container.

II. Algorithms

A. Insert

`void insert_node(const T& key);`

Inserting a key requires the following steps:

1) Finding nodes which will point to the new node at each level.

2) Using a randomized approach to find the level of the node to be inserted.

3) Assigning appropriate links and widths

Complexity: $O(\log_2 n)$ on average

B. Search

`typename SkipList<T>::iterator search(const T& key);`

For searching, we initialize a pointer with head. At each level, traverse till the next is greater than key you are searching for. Repeating this for all levels top to bottom will result in the key required if present or a node past that.

Complexity: $O(\log_2 n)$ on average

C. Delete

`void delete_node(const T& key);`

Deleting a key requires the following steps:
1) Finding nodes which will point to the new node at each level.
2) Assigning appropriate links and widths

Complexity: $O(\log_2 n)$ on average

D. Indexing

`const T& operator[](int index);`

Indexing uses widths maintained by insertion and deletion. The width is defined as the number of bottom layer links being traversed by each of the higher layer "express lane" links. To index the skip list and find the i'th value, traverse the skip list while counting down the widths of each traversed link. Descend a level whenever the upcoming width would be too large.

Complexity: $O(\log_2 n)$ on average

III. Advantages and Caveats

Skip-lists are easier to implement compared to self balancing binary search trees and perform very well on rapid insertions as there are no rotations or reallocations. However, being a node-link based structure, it still has cache performance issues.

IV. Conclusions

We have implemented a generic data structure for generic skip lists. The implementation involved implementing a constant bidirectional iterator, copy constructor and copy assignment, as well as the algorithms: insert, search, delete and indexing. We'd like to thank Prof NS Kumar for guiding us throughout the course.

V. References

1) https://en.wikipedia.org/wiki/Skip_list
2) Prof NS Kumar's class notes