300157147 sunny pak comp 359 skip list report
https://github.com/sunnybutrainy/comp359-assignment2

Skip list data structure

Abstract

This paper examines the design, implementation, and analysis of skip lists, a probabilistic data structure that optimizes dynamic set operations. We discuss key functions such as insertion, deletion, and search, highlighting the roles of pointers and randomization.

Introduction

A skip list is a data structure improved from linked list by incorporating indexing on insertion, deletion, and search operations to $O(log_n)$ time complexity. The structure of a skip list consists of organized elements and multiple layers. Similarly to a linked list, a skip list contains pointers that point to the next element. The core difference is that these multiple pointers in skip list travel across different layers, where each layer can skip elements for efficient search process. Another characteristic of skip list is that it maintains stochastic data structure.

Algorithm

Each node in a skip list contains a key and an associated element. The key is used to identify the element, and the element holds the values. Each layer has infinite plus and infinite minus as a key value conceptually, and the lowest layer, S_0 contains all the values in ascending order. Additionally, the lower layer encompasses the elements found in the upper layers such as S_1, S_2, etc. The higher layers have fewer elements and allows the skip list to skip over multiple nodes when searching, while still maintaining references to the elements in the lower layers.

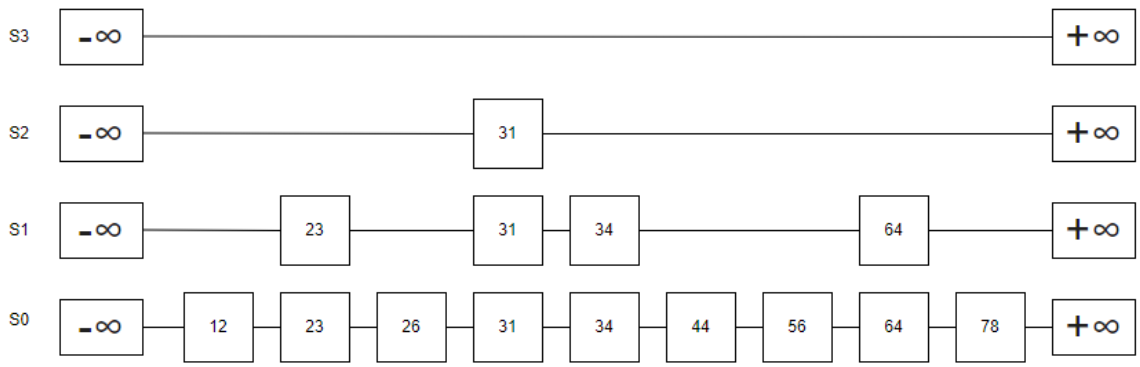

Figure 1. Skip list data representation

300157147 sunny pak comp 359 skip list report

https://github.com/sunnybutrainy/comp359-assignment2

Search Operation in Skip List

During the search operation in a skip list, the search begins at the topmost level. If the current element is larger than the next element, the algorithm descends to the lower level and continues the process until the desired element is found. This behavior could be found in a binary search tree, when it searches for a desired value.

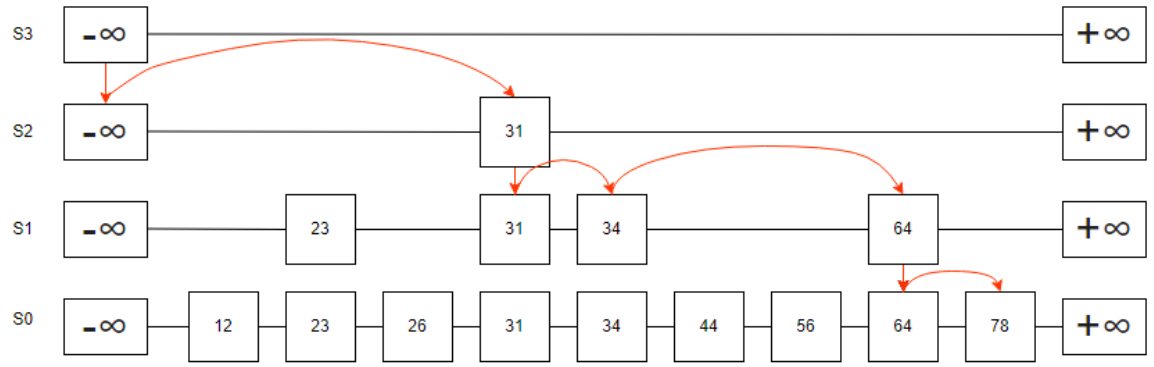

Figure 2. Skip list data format during search

Insertion Operation in Skip List

Skip list is a randomized algorithm. The randomness in a skip list is achieved through the probability of a coin toss, where each "heads" result determines the height of the node being inserted. Similarly, quicksort is another example of a randomized algorithm, as it selects the pivot element randomly during the sorting process. This process facilitates maintaining a reasonable performance.

If the coin lands heads consecutively, say twice, it results in a value of $i = 2$. If value of i is exceeds the current skip lists' height h, it will add $i + 1$ of the height on top of the list.

Case where value of the insert key is 15, $i = 2,\ h = 3$.

List will begin at S_2, and the current list do not have value of 15. It will insert 15, and will move down to S_1. The program will perform a sequential search, and if it encounters a value larger than 15, it will stop and insert 15. At S_0, the algorithm will skip values lesser than 15 until it encounters a larger value, at which point it will also insert 15. Furthermore, the skip list structure allows connections not only between adjacent nodes on the same level but also connects the newly inserted value, 15, with its corresponding nodes across the levels.

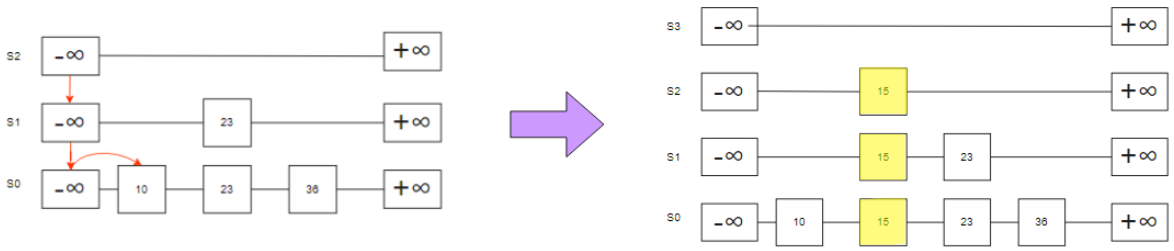

Figure 3. Skip list data representation of insertion process

300157147 sunny pak comp 359 skip list report

https://github.com/sunnybutrainy/comp359-assignment2

Deletion Operation in Skip List

Deletion of an element can easily happen since all the same key values have been connected in insertion process. When deleting a key, the algorithm traverses the layers, identifying all nodes associated with that key. This process helps to eliminate unnecessary high layers that were created from multiple consecutive heads, resulting from coin tosses during insertion phase.

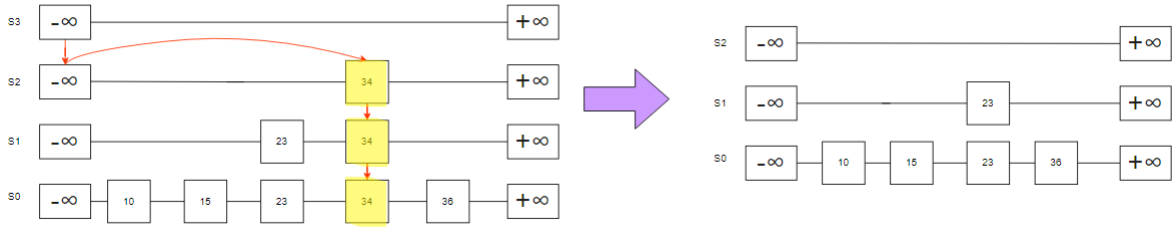

Figure 4. Skip list data representation of deletion process

Implementation

The multiple pointers in skip list allows algorithm to search efficiently. Each node containing a vector of pointers corresponds to the next node at that level. The size of the vector is determined by the level of the node(h), and the node points specific node present in different levels.

```
struct SkipListNode {
    int key;
    std::vector<SkipListNode*> forward;

    SkipListNode(int key, int level) : key(key), forward(level, nullptr) {}
};
```

Figure 5: Structure of Nodes (C++)

In this loop below, the algorithm starts at the highest level and moves horizontally using the forward pointers. If the current key is less than the target key, it moves to the next node at the same level. If the current key is greater than or equal to the target key, it drops down a level to continue searching. Traversal logic is used in insertion, deletion, and search methods.

```
    for (int i = level; i >= 0; i--) {
        while (current->forward[i] != nullptr && current->forward[i]->key < key) {
            current = current->forward[i];
        }
        update[i] = current;
    }
```

Figure 6: Pointer Traverse Behaviour (C++)

300157147 sunny pak comp 359 skip list report

https://github.com/sunnybutrainy/comp359-assignment2

Results

Example on left, level 0 has full organized list of elements in ascending order. Level 1 contains two elements of 7 and 26, which have been selected randomly from level 0 during insertion process. Level 2, 7 was selected randomly from level 1. Level 3 is empty level. This is because of probability, which is set as 0.5 in this example. The probability affects the number of nodes that might occupy in different levels. The higher the probability is, the upper layer will have more elements in it. Thus, output of the algorithm will differ every time it is executed.

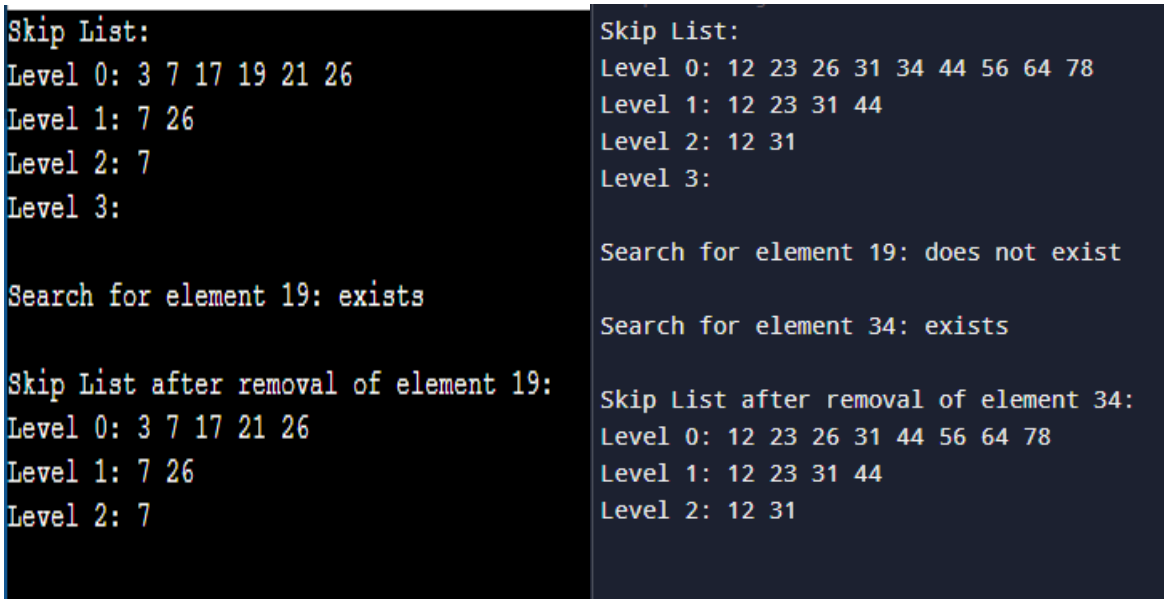

Figure : Results of Implementation of Skip List (C++)

Conclusion

Skip lists present a compelling alternative to linked lists in terms of efficiency. The unique design of pointers, randomization, and layered structures enhances the performance of operations such as insertion and deletion. Therefore, reaching a O(log n) in time complexity. While skip lists demonstrate significant advantages, there is still potential for further improvements. Future research could focus on visual representation of the nodes for insights of the algorithm, and more optimization in memory usage to fasten the process.

Reference

Kumar, S. (2020, July 14). Skip list data structure - explained!. Sumit's Diary. https://blog.reachsumit.com/posts/2020/07/skip-list/

Lecture 7: Randomization: Skip lists | Design and analysis of algorithms | Electrical engineering and computer science | MIT OpenCourseWare. (n.d.). MIT OpenCourseWare. https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/lecture-7-randomization-skip-lists/

(n.d.). YouTube. https://www.youtube.com/watch?v=kBwUoWpeH_Q