

**Project Report on:** Implementation of Skip List

**Course Code**: CSE207

**Course Name:** Data Structure

**Section:** 02

**Project Group:** 07

**Submitted By**: **Submitted To:**

**Name: ID:**

Sami Al Zabid 2022-1-60-103

Md. Saymon Hosen Polash 2022-1-60-171

**Dr. Maheen Islam**

Chairperson, Associate Professor

Department of Computer Science and Engineering

**Date Of Submission:** 17th September, 2023

**Introduction:**

A skip list is a probalistic data structure that allows for efficient search, insertion and deletion of elements in a sorted list. Its average time complexity is determined through a probalistic analysis.

**Properties:**

In a skip list, elements are organized in some levels using linked list and each level have a smaller number of elements than the one below it.

The bottom level is a regular linked list, while the others above it skip links which allows quick traversal to the desired element and reduced number of steps to reach it.

Skip lists have an average time complexity of O(logn) for search, insertion and deletion with the advantage of simpler implementation.

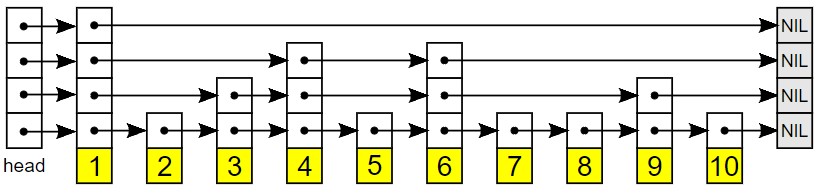


Figure: Skip List

**Algorithm:**

Step 1: Create a Skip List

* Initialize the skip list with a head node (usually with -1) and a maximum level.
* Each level should have a sentinel node with a value of positive infinity.
* Initially, all sentinel nodes are connected to the head node.

Step 2: Insert an Element

* + Start from the top level (the head).
  + Traverse the list horizontally at the current level, moving right until you find a node with a value greater than or equal to the element to be inserted.
  + Keep track of the nodes you traverse at each level.
  + Determine the level for the new element
  + Create a new node for the element and insert it into the list.
  + Update pointers to include the new node at each level based on the tracked nodes from the previous step.

Step 3: Search for an Element

* + Start from the top level (the head).
  + Traverse the list horizontally at the current level, moving right as long as the next element is less than the target.
  + If the next element is equal to the target, the element is found.
  + If the next element is greater than the target, move down one level and repeat the process.
  + Continue until you reach the bottom level or find the target element.

Step 4: Delete an Element

* + Search for the element to be deleted as described in Step 3.
  + Once found, update pointers to bypass the node to be deleted at each level.
  + Free the memory associated with the deleted node.

Step 5: Display the Skip List

* + Start from the top-left (head) node and traverse each level horizontally.
  + Print the elements at each level, separated by arrows or other symbols, to visualize the skip list structure.

Step 6: End

**Implementation process:**

To implement a skip list we used C language. First we defined maximum 5 levels for the skip list. There are two structures for skip list and skip list node. The main functions are search, insert, delete and display operation in our code. Additionally there are createNode() which helps to create a linked list, createSkipList() which initialize a skip list, randomLevel() which generate a random number.

**Insert():** When a new element is inserted, from the highest level we will start comparing it to the key of the next node. If the key of next node is less than new element then we keep on moving forward on the same level. Otherwise, we move one level down and continue until bottom level reached where we can store the given key in its sorted position. After that we use randomLevel() function to generate a random number to determine the number of levels the new element will occupy. If the random number is less than the threshold, the element is inserted into the next level. This process is repeated until the random number is greater than the threshold. It is called the coin flipping technique.

**Search():** The search operation involves traversing the levels from the top to bottom, comparing the nodes data with the key elements and skipping over elements that are not of interest until the key element is found or determined not to be there in the list.

**Delete():** In the delete function we search for the key element to be deleted and then removing it from all the levels in which it appears.

**Display():** In this function we simply traverse from top to bottom levels without skipping any nodes and print each levels node data.

Lastly, in main function we take user input to create a skip list and test either each functionality works properly or not.

**Testing:**

We have tested each operation with various input to verify the code. We have got the expected output which ensures that the skip list properties are maintained.

**Advantages:**

* The skip list is easy to implement comapared to hashtable and binary search tree.
* Inserting a new node is extremely quickly.
* Required only O(logn) time in the average case for all operations.
* Finding a node in the list is relatively straightforward.

**Disadvantages:**

* A skip list needs a greater amount of memory than the balanced tree.
* Reverse search is not permitted.
* Searching is slower than a linked list.
* Skip lists are not cache-friendly because they don’t optimize the locality of reference.

**Difficulties:** While implementing we have faced some problems related to inserting elements and generating random number differently everytime. For that we used srand library function to seed the random-number generator with current time so that the numbers will be different every time we run and also needed to use time.h library.

**Conclusion:**

In summary, skip list provide a simple and efficient alternative to balanced trees for certain use cases, particularly when the average number of elements in the list is large. By doing this project, we learned a new way of implementing and structuring data which can be helpful for our future projects.