EAST WEST UNIVERSITY Department of Computer Science & Engineering

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

Course Code: CSE207

Course Name: Data Structure

Section: 01

Project Group: 10

Submitted To:

Dr. Maheen Islam

Chairperson, Associate Professor

Department of Computer Scienceand Engineering

Submitted By:

Partho Sarker	2021-3-60-202
Dipta Das	2022-1-60-275

Date Of Submission: 28 May 2024

Introduction:

A skip list stands as a probabilistic data structure designed to facilitate swift search, insertion, and deletion operations within a sorted list. The evaluation of its average time complexity involves a meticulous analysis rooted in probability, highlighting its prowess in managing sorted data with efficiency.

Properties:

In a skip list, elements are organized in some levels using linked list and each level has 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 for search, insertion, and deletion with the advantage of simpler implementation.

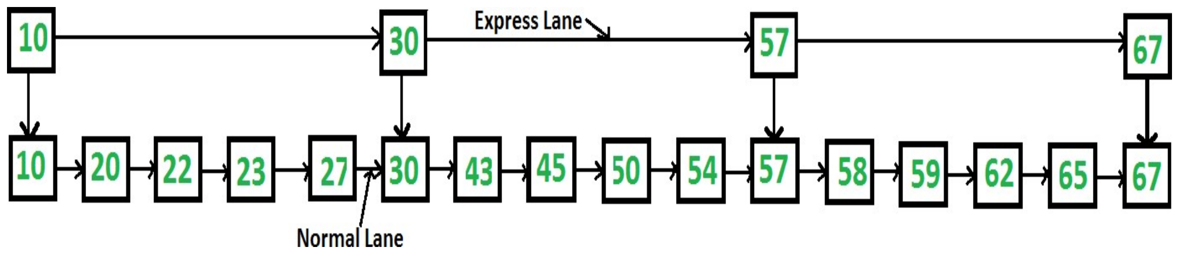

Figure: Skip List

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

Conclusion:

To conclude, skip lists offer a straightforward and effective substitute for balanced trees in specific scenarios, especially when dealing with a considerable average number of elements in the list. Through the completion of this project, we have gained valuable insights into a novel approach for implementing and organizing data. This newfound knowledge not only enhances our understanding of skip lists but also equips us with a valuable tool for addressing challenges in our future projects.