Skip List

Anoop S Babu

Faculty Associate

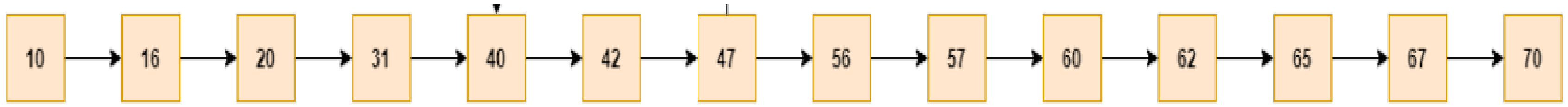
Dept. of Computer Science & Engineering

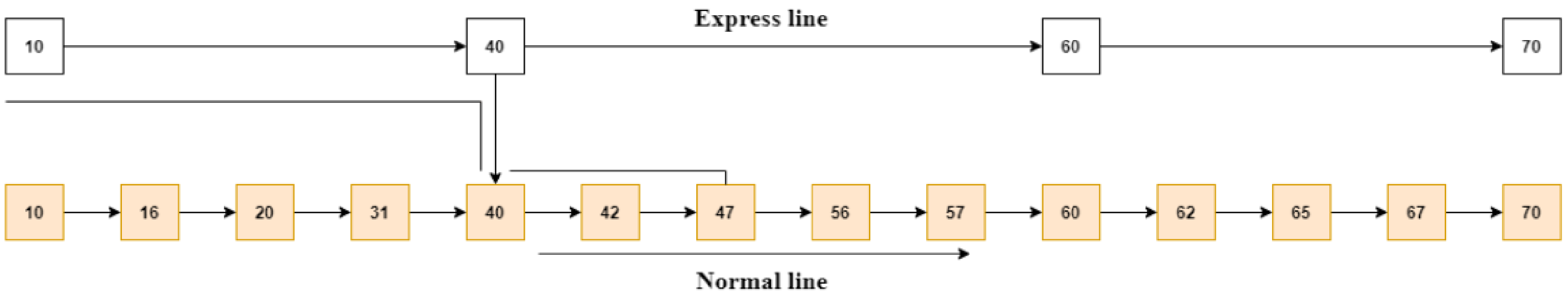
bsanoop@am.amrita.edu

Searching in a Linked List

- Average and worst time complexity for searching and element in unsorted linked list with n elements ?

  Answer: O(n)

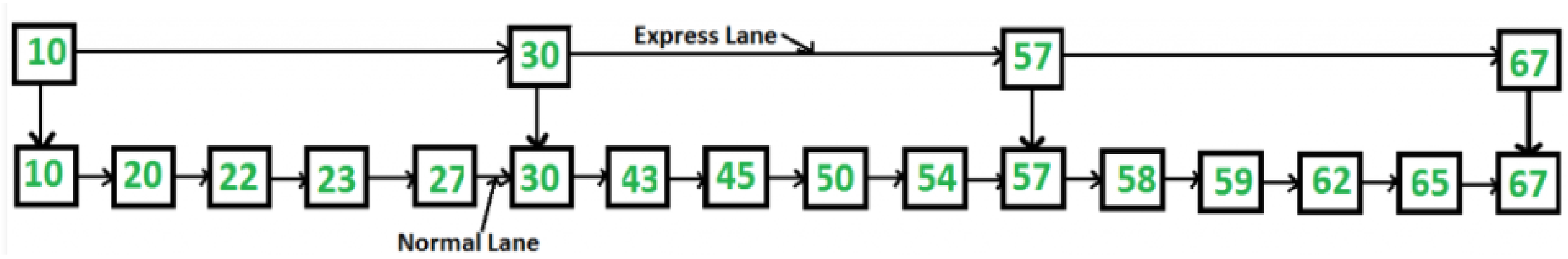




Skip List

- A skip list is a probabilistic data structure.
- The skip list is used to store a sorted list of elements or data with a linked list.
- In one single step, it skips several elements of the entire list.

Skip List Structure



- The **lowest layer** of the skip list is a **common sorted linked list**.
- Top layers of the skip list are like an "**express line**" where the elements are skipped.

Skip List Basic Operations

- There are the following types of operations in the skip list.
  - **Insertion operation:** It is used to add a new node to a particular location in a specific situation.
  - **Deletion operation:** It is used to delete a node in a specific situation.
  - **Search Operation:** The search operation is used to search a particular node in a skip list.

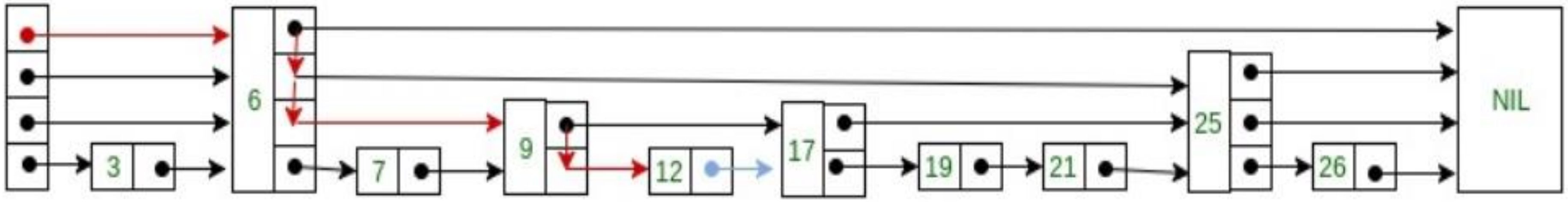
Searching an element in Skip list

Steps

1. Key of next node is less than search key then we keep on moving forward on the same level.
2. Key of next node is greater than the key to be inserted then we store the pointer to current node **i** at **update[i]** and move one level down and continue our search.

- At the lowest level (0), if the element next to the rightmost element (update[0]) has key equal to the search key, then we have found key otherwise failure.

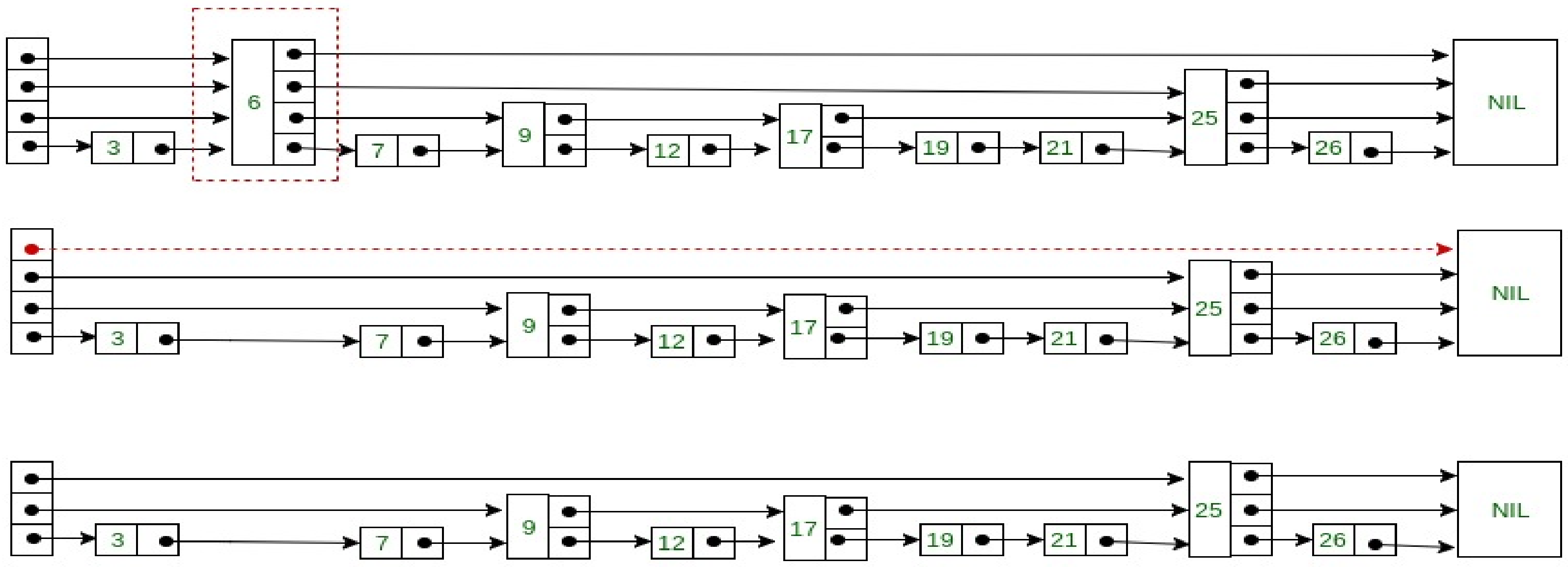
Example: Searching for 17



Deleting an element from the Skip list

- Deletion of an element k is preceded by locating element in the Skip list.
- Once the element is located, rearrangement of pointers.
- After deletion of element there could be levels with no elements; if so remove that levels also.

Example: Deleting 6



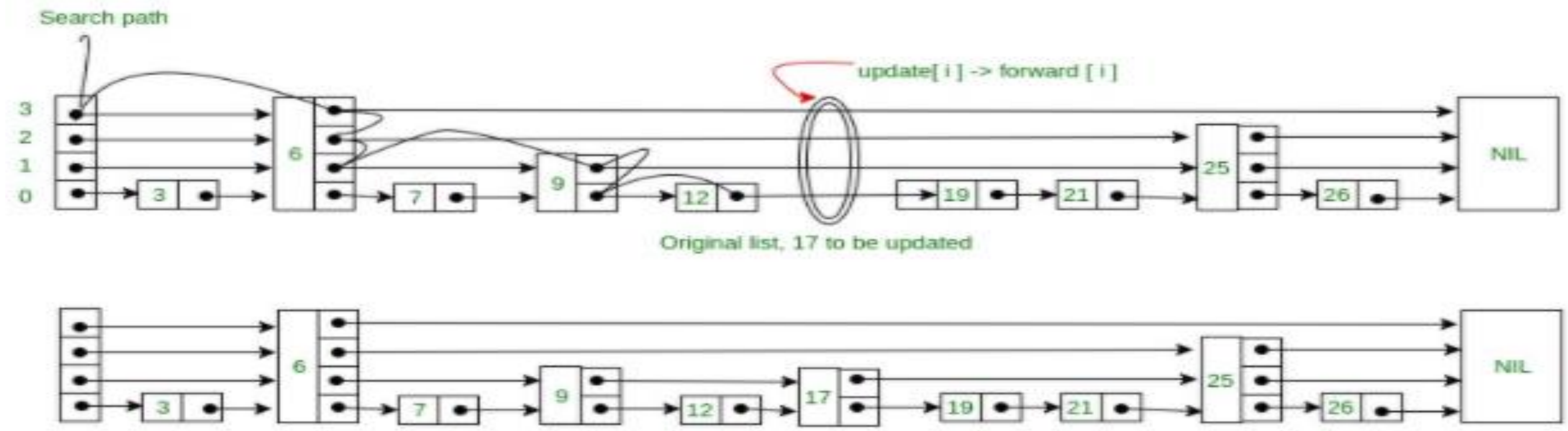
Insertion in Skip List

- Start from highest level in the list and compare key of next node of the current node with the key to be inserted.

Steps

1. Key of next node is less than key to be inserted then we keep on moving forward on the same level
2. Key of next node is greater than the key to be inserted then we store the pointer to current node **i** at **update[i]** and move one level down and continue our search.

At the level 0, we will definitely find a position to insert given key.


Search path
update[i] -> forward [i]
3
2
1
0
3
6
7
9
12
19
21
25
26
NIL
Original list, 17 to be updated
17