**Insert Operation**

To insert key k with value v, we follow these steps:

1. Find the predecessor and success or k
2. Create a new leaf node between predecessor and successor of K and point it to v.
3. Start traversing the tree from the root to leaf and create the missing internal nodes and adding the descendant pointers wherever necessary.

**Delete Operation**

To delete the key k, we follow these steps:

1. Remove the k from the hash table indexing the leaf nodes.
2. Delete the k node and link its predecessor and successor.
3. Traverse the tree from the root and delete the internal nodes that have k as only nodes within its subtrees and update the descendant pointers appropriately.

## 11.6 Y-FAST TRIES

Y-fast tries are bitwise tries that store integers from bounded context. They are the improved version of X-fast tries that optimize the memory. Similar to the X-fast tries, all subtrees will have a common prefix of its parent.

The complexity of successor operation is O(log log M) where M is the domain's maximum value; find operation is O(1); insert and delete operation is O(log M) where M is the domain's maximum value.

The Y-fast trie consists of two data structures – X-fast trie at the top and the balanced binary search tree in the bottom. We have depicted the Y-fast Trie in Figure 6.

We chose a representative r from the binary search tree and store it in the X-fast trie. The r value should be smaller than its successor in the X-fast trie.
One of the main differences between X-fast trie and Y-fast trie is that in the Y-fast trie, the X-fast trie operates on the representatives of the binary search tree. The leaf node point to the binary search tree.

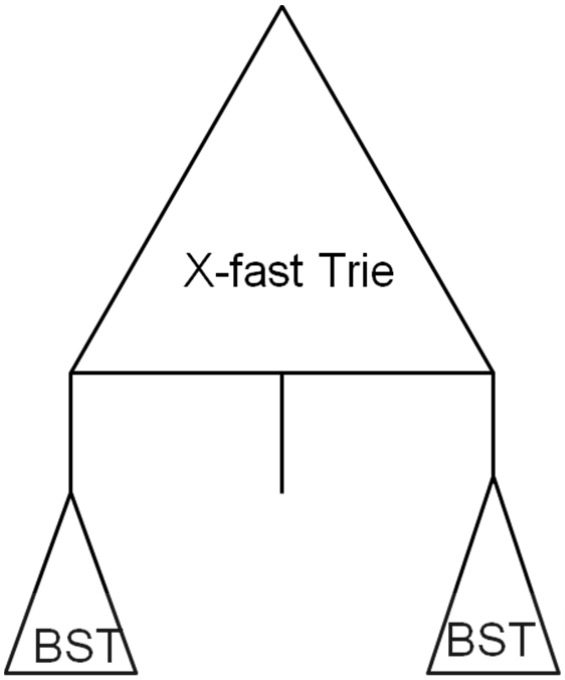


**Figure 7 Y-fast Trie**

### 11.6.1 Operations on Y-fast tries

In this section we shall look at the main operations on Y-fast tries.

**Find operation**

Given below are the steps to find the key k:

1. Find the successor and predecessor of k in the X-fast trie. We can do this by finding the smallest representative r that is greater than k and largest representative $r^1$ that is smaller than k in the X-fast trie.
2. We carry out the binary search for k in the binary search trees represented by r and $r^1$

**Successor and Predecessor**

1. To find successor of k, search for smallest representative r in the X-fast trie.
2. Find the predecessor of r in X-fast trie.
3. Search for the binary trees pointed by these two representatives for getting the successor of k.

**Insert Operation**

To insert the key k, given below are the steps:

1. Insert the key to the tree that has the successor of k.
2. Point the node to v
3. If the tree contains more than 2 log U elements (where U is the maximum value of the domain), then remove it representative from the X-fast trie. Split the binary search tree into two and add the representatives from each of the binary search tree into the X-fast trie.

**Delete operation**

Given below are the steps for deleting the key k:

1. Delete the key k from the tree

2. If the elements in the tree drop below (log U)/4 (where U is the maximum value of the domain), merge the tree with its predecessor or successor. Remove the representatives of the merged trees from the X-fast trie.
3. Add the representative of the new tree into X-fast trie.
4. If the newly-formed tree contains more than 2 log U elements (where U is the maximum value of the domain), then remove it representative from the X-fast trie. Split the binary search tree into two and add the representatives from each of the binary search tree into the X-fast trie.

### ☞ Check Your Progress – 2

1. The child nodes of a node in a Trie are associated with a _____
2. The average time complexity for search, insert and delete operations for trie data structure is _____
3. The binary tries use ____ as keys that can be used to represent integer
4. X-fast tries are bitwise tries that stores the integers from a _____
5. Each level of the x-fast trie is implemented as _____
6. Y-fast tries are the improved version of _____

## 11.7 SUMMARY

In this unit, we started discussing the key aspects of binary search tree. Binary search tree is a sorted tree that stores the keys in ordered way. The scapegoat tries are self-balancing binary search tree. When the weight of the left nodes matches the weight of the right nodes then we say that the binary search tree is balanced. To insert the data we find the scapegoat node in scapegoat trees and balance the tree around it. Tries are tree-based data structure that are used to search for string-based keys in a given set. The average time complexity for search, insert and delete operations for trie data structure is O(n). The binary tries use bits as keys that can be used to represent integer. X-fast tries are bitwise tries that stores the integers from a bounded domain.

## 11.8 SOLUTIONS/ANSWERS

### ☞ Check Your Progress – 1

1. Self-balancing
2. scapegoat
3. O(logn)
4. Lesser
5. O(n)

### ☞ Check Your Progress – 2

1. common prefix
2. O(n)
3. bits
4. bounded domain
5. hash table

6. X-fast tries

## 11.9 FURTHER READINGS

Horowitz, Ellis, Sartaj Sahni, and Susan Anderson-Freed. *Fundamentals of data structures*. Vol. 20. Potomac, MD: Computer science press, 1976.

Cormen, Thomas H., et al. *Introduction to algorithms*. MIT press, 2022.

Lafore, Robert. *Data structures and algorithms in Java*. Sams publishing, 2017.

Karumanchi, Narasimha. *Data structures and algorithms made easy: data structure and algorithmic puzzles*. Narasimha Karumanchi, 2011.

West, Douglas Brent. *Introduction to graph theory*. Vol. 2. Upper Saddle River: Prentice hall, 2001.