Introduction to Computing

Binary Search Trees

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

November, 2024

Binary search trees

Definition

A binary search tree is principally a binary tree in which the following properties hold for all the nodes:

- key values in left subtree are less than the key value in the node
- key values in right subtree are greater than the key value in the node

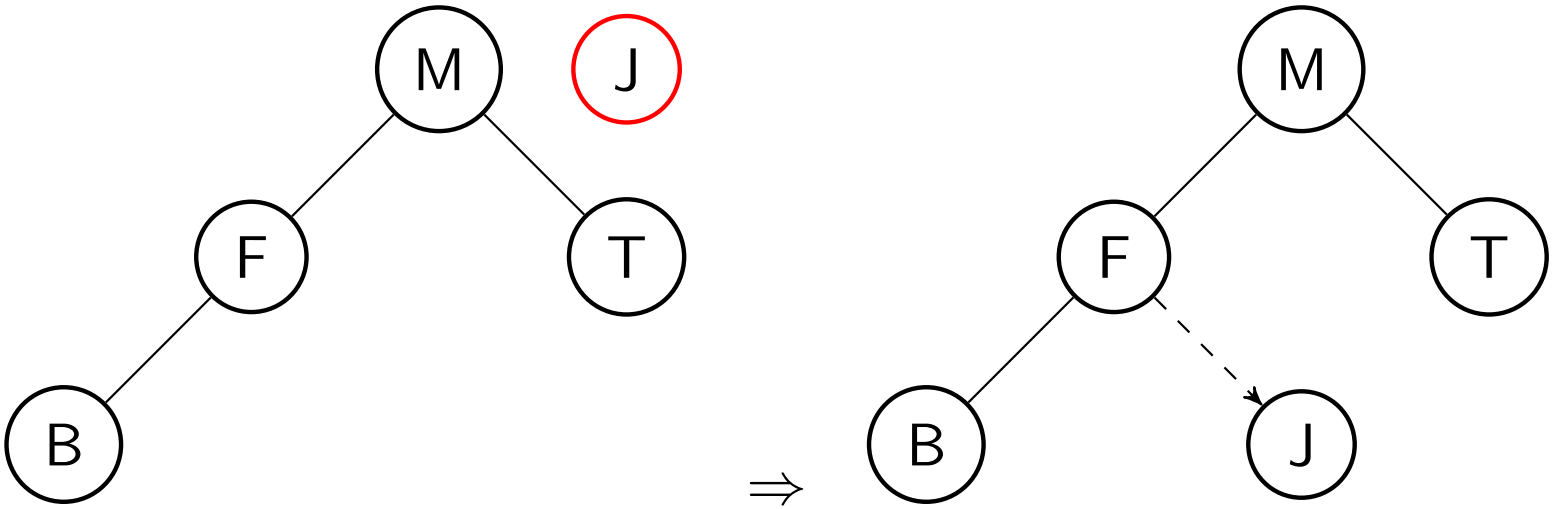
Main operations

- **Insertion**
- **Search**
- **Deletion**

Auxiliary operations

- **Find successor**
- **Find predecessor**

Conventional implementation – Insertion



Conventional implementation – Deletion

Let X be the node to be deleted.

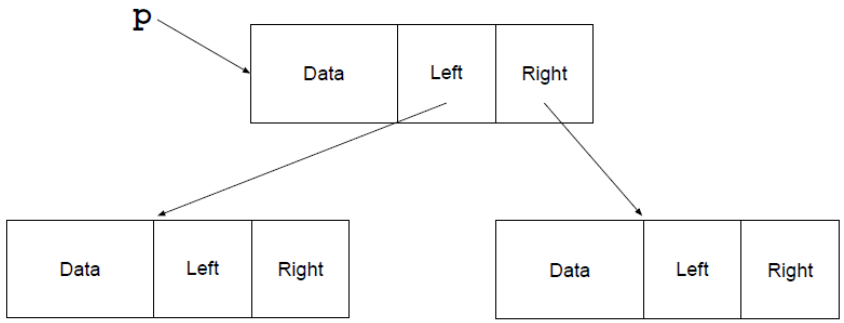
Case I: X is a leaf node.
- Simply delete X.

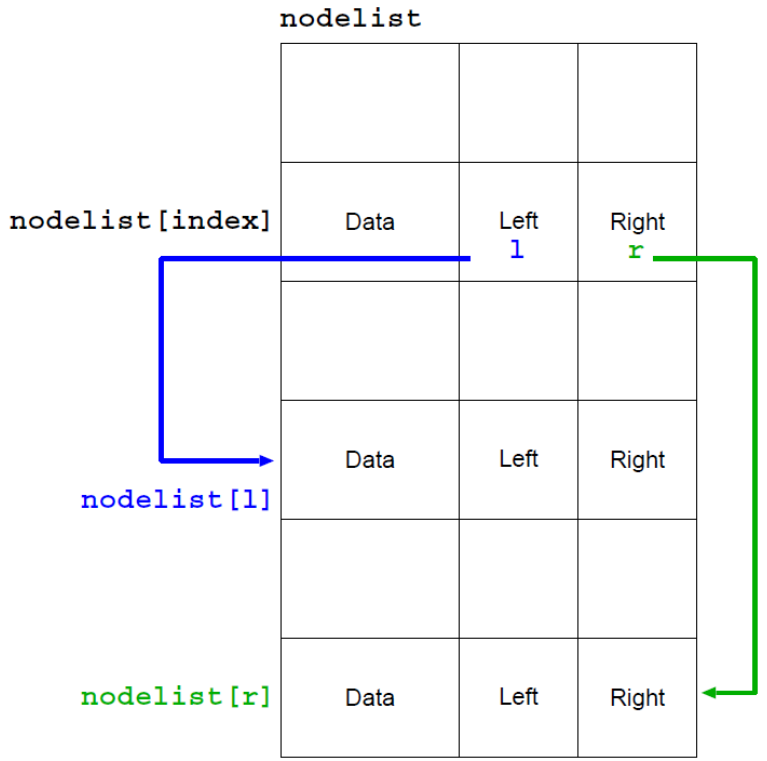
Case II: X has one child.
- Replace the link to X with a link to its only child.

Case III: X has 2 children.

1. Find S, the successor of X (node with the smallest key in the right subtree of X or with the largest key in the left subtree).
2. Replace the value in X by the value in S.
3. Delete node S from the tree (see Cases I and II above).

Conventional versus Alternative implementation





Alternative implementation

```
class BinarySearchTree(BST):
    # If no init is provided, the init inherited from the superclass'
    # will be called. Since that is all we're doing here, we don't
    # really need this __init__.
    # NOTE: If we do provide an __init__, that init will not
    # automatically call super().__init__()
    def __init__(self, capacity):
        super().__init__(capacity)
    def search(self, start_index, d):
        if index == -1:
            return -1
        if d < self.nodelist[index].Data:
            return self.search(self.nodelist[index].Left, d)
        elif d > self.nodelist[index].Data:
            return self.search(self.nodelist[index].Right, d)
        else:
            return index
```