# BST Operations



You are given Q queries. Each query can be one of four types:

- 1 X: Insert an integer X into the binary search tree.
- 2 X: Delete X from the tree.
(If the node has both child present, replace it with the smallest value from its right sub-tree)
- 3 X: Print "Yes" if the value exists in the binary search tree, print "No" otherwise.
- 4: Print the Preorder traversal of the binary search tree.

### Input Format

First line of input contains T - number of test cases.
The second line contains Q - the number of queries.
The Q subsequent lines each contain 2 integers A - the type of query and X - the value to be processed in the query (except for query of type 4).

### Constraints

1 <= T <= 100
1 <= Q <= 100
1 <= A <= 4
$-10^5$ <= X <= $10^5$

### Output Format

For each test case, print the result, separated by newline. If no such result exists, print "null".

### Sample Input 0

```
2
8
1 4
1 2
1 6
1 8
2 2
3 3
4
3 1
7
1 5
1 7
1 3
4
3 3
2 3
3 3
```

### Sample Output 0

```
Case #1:
No
4 6 8
No
Case #2:
5 3 7
Yes
No
```

### Explanation 0

Self Explanatory

Java 8

```java
import java.io.*;
import java.util.*;

class Node{
    int data;
    Node left, right;
    Node(int x){
        this.data = x;
        this.left = this.right = null;
    }
}

public class Solution {

    static Node insert(Node root, int d){
        if(root == null){
            Node n = new Node(d);
            return n;
        }

        if(d < root.data){
            root.left = insert(root.left, d);
        }
        else{
            root.right = insert(root.right, d);
        }

        return root;
    }

    static String isPresent(Node root, int x){
        if(root == null){
            return "No";
        }

        if(x == root.data)
            return "Yes";


        if(x < root.data){
            return isPresent(root.left, x);
        }
        else{
            return isPresent(root.right, x);
        }

        // return "No";
    }

    static int findMin(Node root){
        Node temp = root;

        while(temp.left != null){
            temp = temp.left;
        }
        return temp.data;
    }

    static Node delete(Node root, int x){
        if(root == null)
            return root;

        if(x < root.data){
            root.left = delete(root.left, x);
            return root;
        }
        else if(x > root.data){
            root.right = delete(root.right, x);
            return root;
        }
        else{
            if(root.left == null && root.right == null){
                return null;
            }

            // if(root.left == null && root.right != null){
```

```java
            //      return root.right;
            // }
            // else if(root.left != null && root.right == null){
            //      return root.left;
            // }
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
                return root.left;
            }

            root.data = findMin(root.right);
            root.right = delete(root.right, root.data);
            return root;
        }
        // return root;
    }


    private static void preorder(Node root){
        if(root == null){
            return;
        }
        System.out.print(root.data+" ");
        preorder(root.left);
        preorder(root.right);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        for(int i=0; i<t; i++) {
            System.out.println("Case #"+(i+1)+":");
            int queries = sc.nextInt();
            Node root = null;
            while(queries-- > 0){
                int c = sc.nextInt();
                switch(c){
                    case 1:
                        int x = sc.nextInt();
                        root = insert(root, x);
                        break;

                    case 2:
                        int x1 = sc.nextInt();
                        root = delete(root, x1);
                        break;

                    case 3:
                        System.out.println(isPresent(root, sc.nextInt()));
                        break;

                    case 4:
                        preorder(root);
                        System.out.println();

                }
            }
        }
    }
}
```

Copy And Save | Share | Ask Copilot

Line: 15 Col: 42

Upload Code as File | Test against custom input | Run Code | Submit Code