PROLOG ACADEMY

- **DATA STRUCTURE**

By- MASEERA ALI

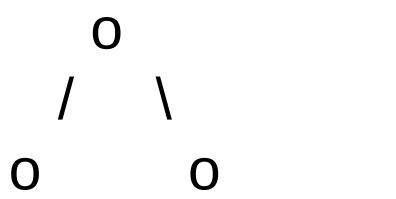
- Book followed - Data structures by Seymour Lipschutz (Schaum Series)

LET’S START!

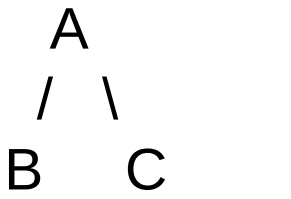
Enumeration of Binary Trees

A Binary Tree is labeled if every node is assigned a label
A Binary Tree is unlabeled if nodes are not assigned any label.

Unlabeled tree



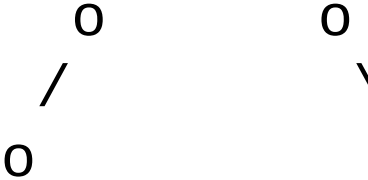
Labeled trees



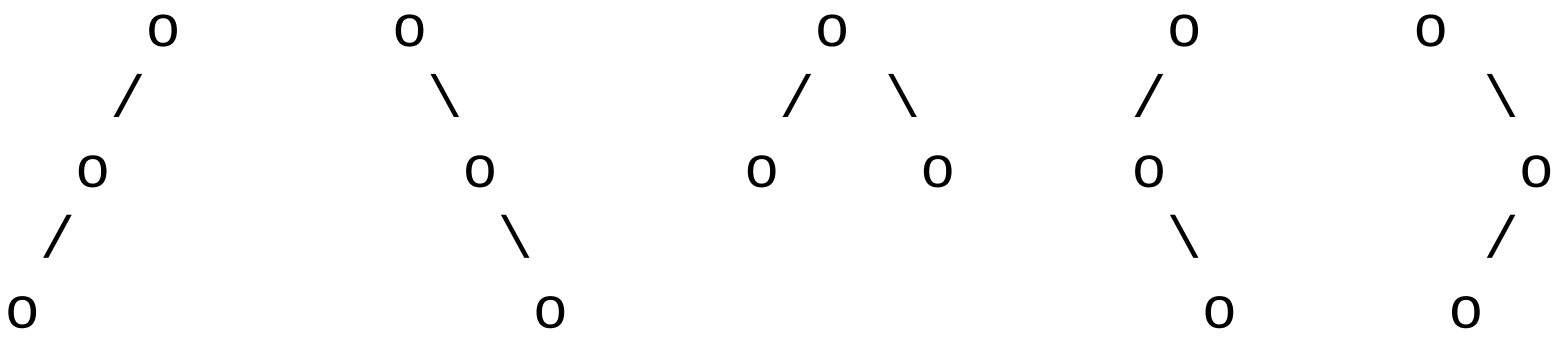
How many different Unlabeled Binary Trees / Binary Search trees can be there with n nodes?

For n = 1, there is only one tree

For n = 2, there are two trees



For n = 3, there are five trees



This can be evaluated by using Catalan number

- n’th Catalan Number can be evaluated using direct formula.

$$T(n) = (2n)! / (n+1)!n!$$

How many labeled Binary Trees / Binary Trees can be there with n nodes?

Number of Labeled Tees
= (Number of unlabeled trees) * n!
= [(2n)! / (n+1)!n!] × n!

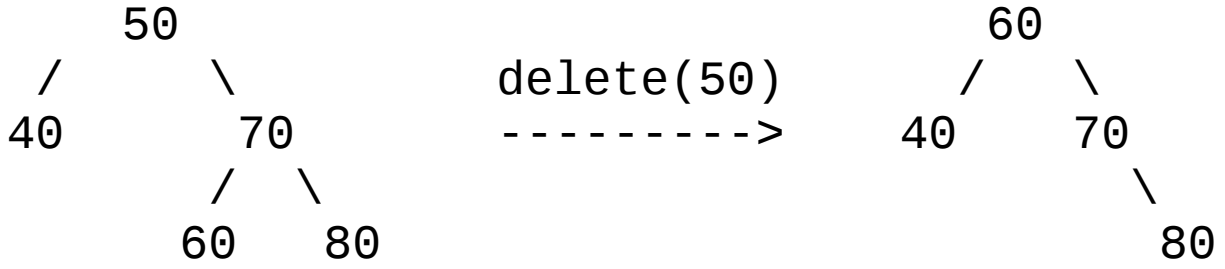
Deletion from BST

When we delete a node, there possibilities arise.

1) Node to be deleted is leaf: Simply remove from the tree.

2) Node to be deleted has only one child: Copy the child to the node and delete the child

3) Node to be deleted has two children: Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.



The important thing to note is, inorder successor is needed only when right child is not empty. In this particular case, inorder successor can be obtained by finding the minimum value in right subtree of the node.

Deletion in BST

```
#include <stdio.h>
#include <stdlib.h>

struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL;

void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))
    search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
    t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))
    search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
    t->l = temp;
}
```

```
void insert()
{

    int data,ch=1;
        while(ch==1){
    printf("Enter data: ");
    scanf("%d", &data);

   temp = (struct btnode *)malloc(sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
    if (root == NULL)
    root = temp;
    else
    search(root);

   printf("To continue press 1 else 0");
   scanf("%d",&ch);

        }
}
```

```
struct btnode * minValueNode(struct btnode* node)
{
    struct btnode* current = node;
    while (current->l != NULL)
        current = current->l;

    return current;
}

struct btnode* deleteNode(struct btnode* root, int key)
{

    if (root == NULL) return root;

    if (key < root->value)
        root->l = deleteNode(root->l, key);

    else if (key > root->value)
        root->r= deleteNode(root->r, key);

    else
    {

                if (root->l == NULL)
                {
                    temp = root->r;
                    free(root);
                    return temp;
                }
                else if (root->r == NULL)
                {
                    temp = root->l;
                    free(root);
                    return temp;
                }

                temp = minValueNode(root->r);

                root->value = temp->value;

                root->r = deleteNode(root->r, temp->value);
        }
    return root;
}
```

```
int main()
{
        insert();
    int data;
    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("\nInorder traversal is\n");
    inorder(root);

        printf("\nEnter the data to be deleted : ");
    scanf("%d", &data);
        deleteNode(root, data);
        printf("Node deleted successfully!\n Inorder Traversal is-\n");
    inorder(root);

        return 0;
}
int inorder(struct btnode *t)
 {
     if (t == NULL)
         return;
     inorder(t->l);
     printf("%d\t", t->value);
     inorder(t->r);
 }
```

Smallest and Largest node of BST

Smallest node – Smallest node will be the left most node of the BST

CODE -

```
int smallest(struct btnode *t)
{
    if (t->l != NULL)
        return(smallest(t->l));
    else
        return (t->value);
}
```

Largest Node – Largest Node will be the right most node of the BST

CODE -

```
int largest(struct btnode *t)
{
    if (t->r != NULL)
        return(largest(t->r));
    else
        return(t->value);
}
```

Kth min and Kth max of BST

- As we know that inorder traversal of a BST is the ascending order of the nodes in BST
- So by printing kth node in inorder traversal(L Rt R) we actually print the kth minimum node of the BST
- And by printing kth node in reverse inorder traversal (R Rt L) we actually print the kth maximum node of the BST

CODE -

```
int kmin(struct btnode *t,int k)
{
	if (t == NULL)
		return;
	kmin(t->l,k);
	count++;
	if(count == k)
		printf("%d",t->value);
	kmin(t->r,k);
}

int kmax(struct btnode *t,int k)
{
	if (t == NULL)
		return;
	kmax(t->r,k);
	count++;
	if(count == k)
		printf("%d",t->value);
	kmax(t->l,k);
}
```