Trees

Ritankar Mandal

Binary Tree

- **Tree:** A connected acyclic graph.
- **Binary Tree:** A Tree where every node has at most two children.

Binary Search Tree

The values in a binary search tree are always stored in such a way as to satisfy the **binary-search-tree property**.

Let x be a node in a binary search tree. If y is a node in

- the left subtree of x, then $value(y) \leq value(x)$.
- the right subtree of x, then $value(x) < value(y)$.

Binary Search Tree

```
 1  typedef struct node
 2  {
 3      float val;
 4      struct node *lchild;
 5      struct node *rchild;
 6  } Node;
 7
 8  int getMenu();
 9  Node* addNode(Node*, float);
10  void inorder_r(Node*);
```

Recursively add a new Node into a BST

```
 1  Node* addNode_r(Node *curr, float newval)
 2  {
 3      if(curr == NULL)
 4      {
 5          curr = (Node *)malloc(sizeof(Node));
 6          curr->val = newval;
 7          curr->lchild = curr->rchild = NULL;
 8      }
 9      else if(newval <= curr->val)
10          curr->lchild = addNode_r(curr->lchild, newval);
11      else
12          curr->rchild = addNode_r(curr->rchild, newval);
13
14      return curr;
15  }
```

Recursive Inorder traversal

$$\langle \text{ Left subtree } \rangle \text{ root } \langle \text{ Right subtree } \rangle$$

```
1  void inorder_r(Node *curr)
2  {
3      if(curr != NULL)
4      {
5          inorder_r(curr->lchild);
6          printf("%f, ", curr->val);
7          inorder_r(curr->rchild);
8      }
9  }
```

Inorder traversal of BST always produces sorted output.

Recursive Preorder traversal

root $\langle$ Left subtree $\rangle$ $\langle$ Right subtree $\rangle$

```
1  void preorder_r(Node *curr)
2  {
3      if(curr != NULL)
4      {
5          printf("%f, ", curr->val);
6          preorder_r(curr->lchild);
7          preorder_r(curr->rchild);
8      }
9  }
```

Recursive Postorder traversal

⟨ Left subtree ⟩ ⟨ Right subtree ⟩ root

```
1  void postorder_r(Node *curr)
2  {
3      if(curr != NULL)
4      {
5          postorder_r(curr->lchild);
6          postorder_r(curr->rchild);
7          printf("%f, ", curr->val);
8      }
9  }
```

Recursive Delete a node from a BST

```
1  Node* delNode(Node* curr, float val)
2  {
3      Node* temp;
4      // base case
5      if(curr == NULL)
6          return curr;
7      // Search in the left subtree
8      if(val < curr->val)
9          curr->lchild = delNode(curr->lchild, val);
10     // Search in the right subtree
11     else if(val > curr->val)
12         curr->rchild = delNode(curr->rchild, val);
13     // If val is found
14     else
15     {
16         <See the next slide for the else part>
17     }
18     return curr;
19 }
```

Recursive Delete a node from a BST

```
1  // node with only one child or no child
2  if(curr->lchild == NULL)
3      return curr->rchild;
4  else if(curr->rchild == NULL)
5      return curr->lchild;
6  // node with two children: replace with inorder successor
7  // find the inorder successor
8  temp = curr->rchild;
9  while((temp != NULL) && (temp->lchild != NULL))
10     temp = temp->lchild;
11 // Copy the inorder successor's content to this node
12 curr->val = temp->val;
13 // Delete the inorder successor
14 curr->rchild = delNode(curr->rchild, temp->val);
```