Today - Lecture 10-11 CS163

Binary Search Trees

1. Traversal Algs
2. Review Recursion with LLL
3. Implement Insert for BST
4 Removal Algorithm ← <u>next</u> <u>time</u>
    - special cases

2) Examine efficiency of a BST
    - Discuss height and shape

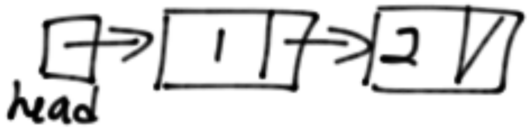
Remember Recursion for a LLL

- Add at the end -

```
void add_at_end(node * & head, student & data)
{
   if (!head) //time to add!
   {
       head = new node;
       head->peer.set(data);
       head->next = NULL;
   }
   else
      add_at_end(head->next, data);
}
```

Alternatively…

```
node * add_at_end(node * head, student & data)
{
  if (!head) //time to add!
  {
     head = new node;
     head->peer.set(data);
     head->next = NULL;
   }
  else
    head->next = add_at_end(head->next, data);
   return head;
}
```



Pass by Pointer - used to simulate pass by reference is some languages

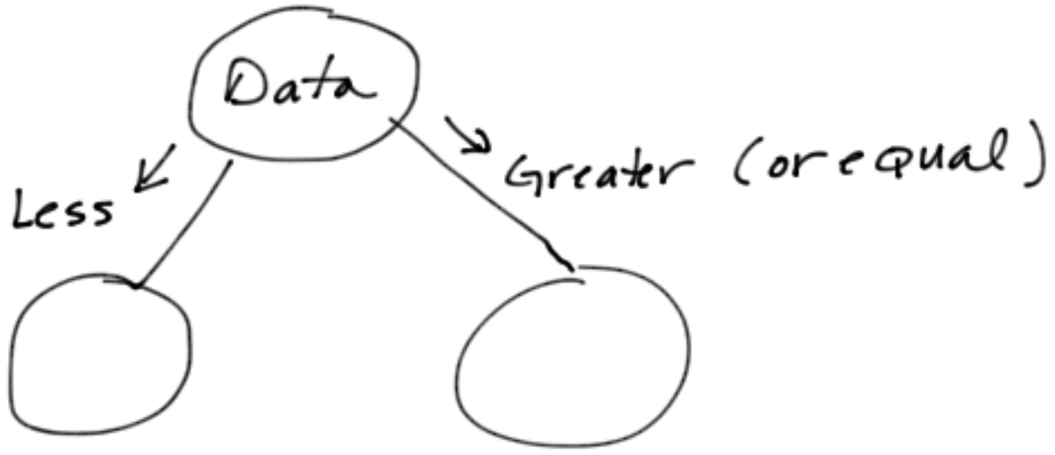
```
void add_at_end(node ** phead, student & data)
{
  if (!(*phead) ) //time to add!
  {
     node * head = new node; //why not head = new node?
     head->peer.set(data);
     head->next = NULL;
    *phead = head;     // this is important if using a temp pointer!
  }
  else
    add_at_end(&(*phead)->next, data);
}
```

Insert for Trees

1. Examine the simple case

```
if (!root)
{
    root= new node;
    root->peer.set(data);
    root->left = root->right = NULL;
}
```

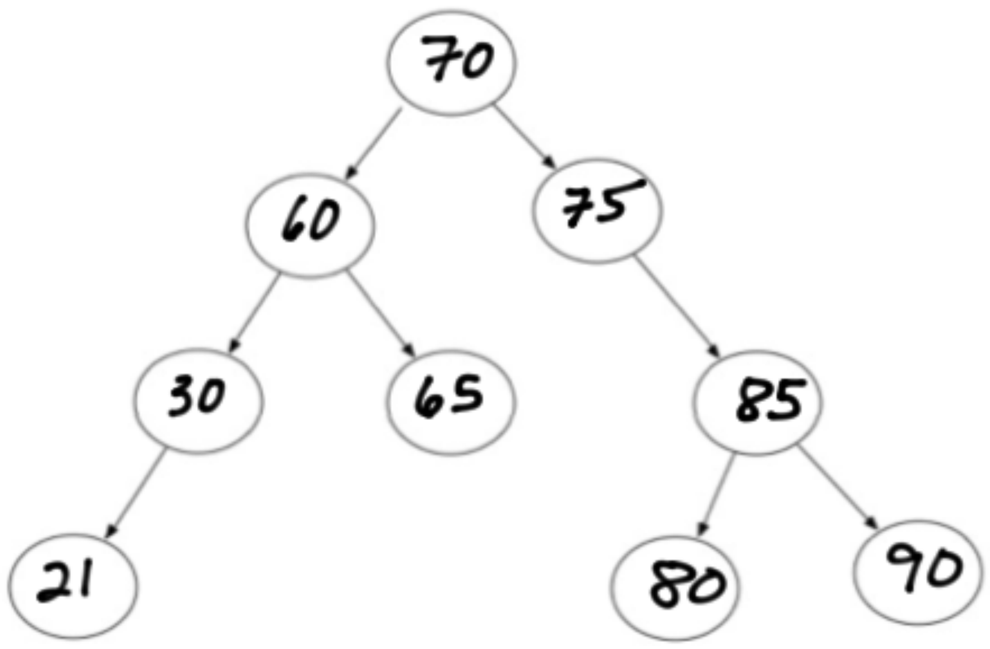
2. Examine which direction to traverse



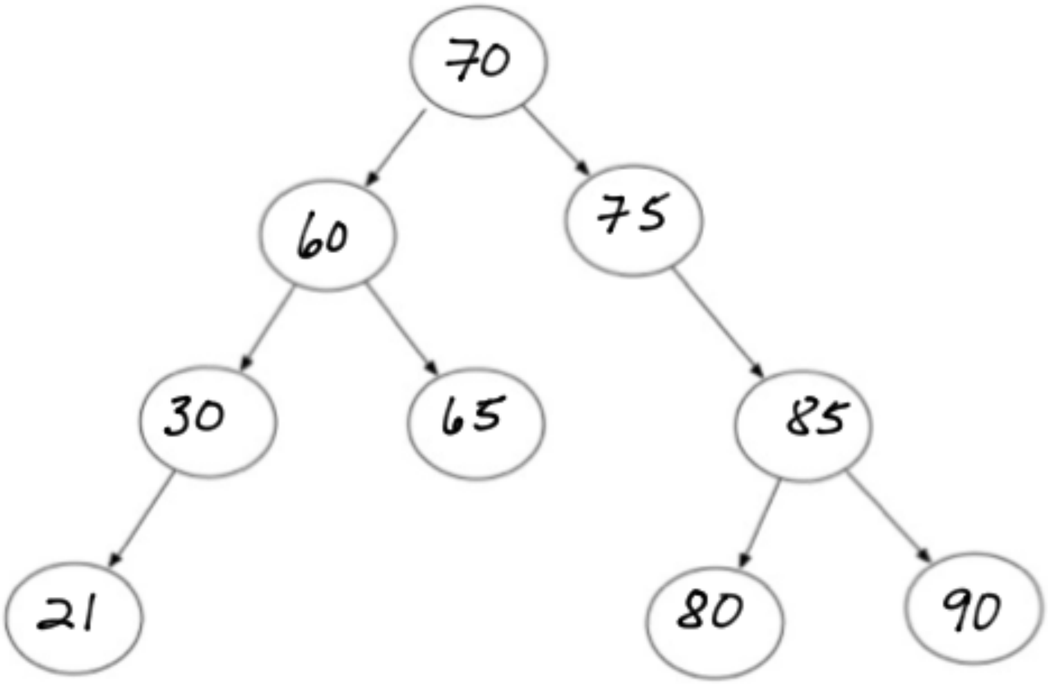
3. Stopping Condition? When Root is NULL!

Insert in a BST:

```
void add_at_end(node * & root, student & data)
{
  if (!root)
  {
     root= new node;
     root->peer.set(data);
     root->left = root->right = NULL;
  }
  else if (root->peer.compare(data) < 0) //LESS
     add_at_end (root->left, data);
  else
     add_at_end (root->right, data);
}
```



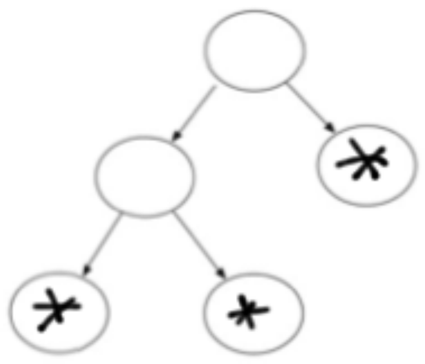
```
node * add_at_end(node * root, student & data)
{
  if (!root)
  {
     root= new node;
     root->peer.set(data);
     root->left = root->right = NULL;
  }
  else if (root->peer.compare(data) < 0) //LESS
     root->left = add_at_end (root->left, data);
  else
     root->right = add_at_end (root->right, data);
   return root
}
```



Removal – Special Cases

1. Empty Tree
2. Item to remove is <u>not</u> found
3. Item is found and it is located at:
   - 3a) Leaf
   - 3b) Internal Node with only 1 child (left)
   - 3c) Internal Node with only 1 child (right)
   - 3d) Internal Node with 2 children – but the right child has no <u>LEFT</u> children
   - 3e) Internal Node with 2 children

Case 1 & 2: Root

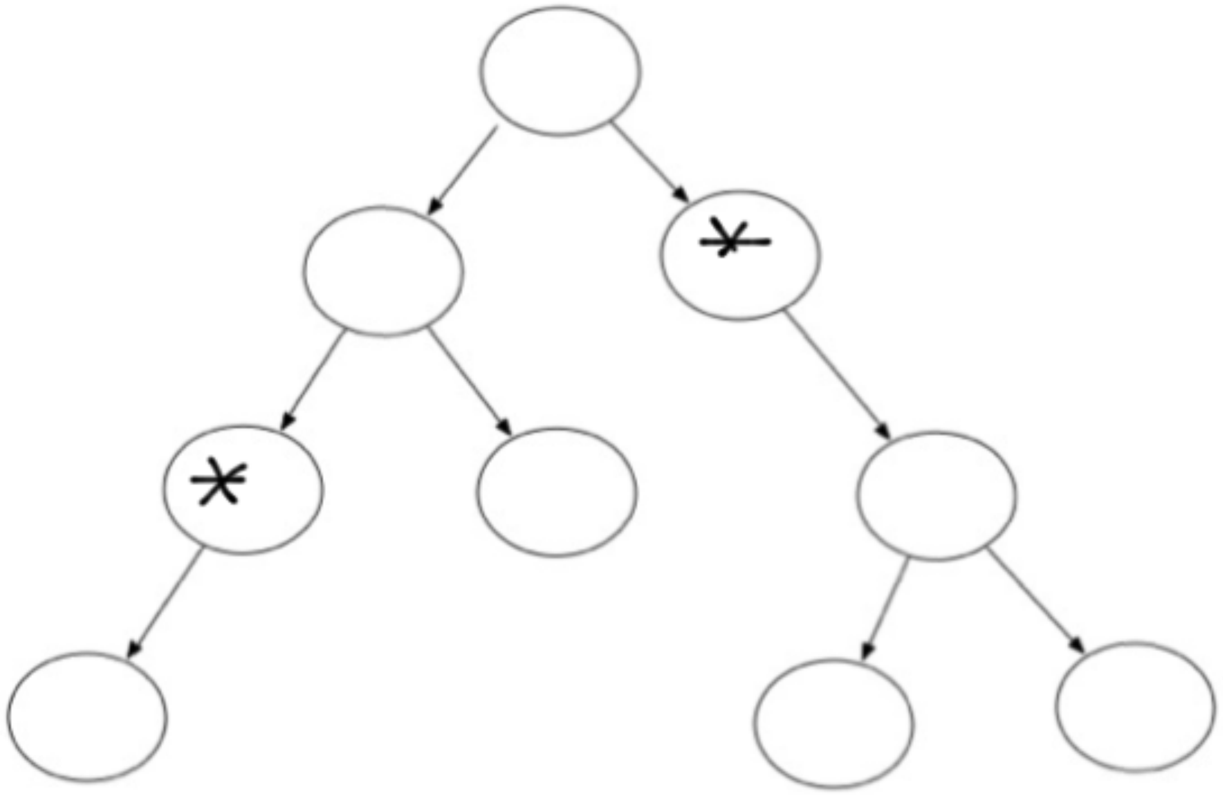


case 3a):

Root→left <u>and</u>
Root→right are NULL

Case 3b & 3c) Internal node w/ 1 child

1) Root→left is <u>not</u> NULL <u>OR</u> Root→right is not NULL — but the other $\boxed{\text{IS}}$ NULL



Case 3d & 3e) Internal Node with 2 children

Root→Left and Root→Right are $\boxed{\text{NOT}}$ NULL

Find the inorder successor