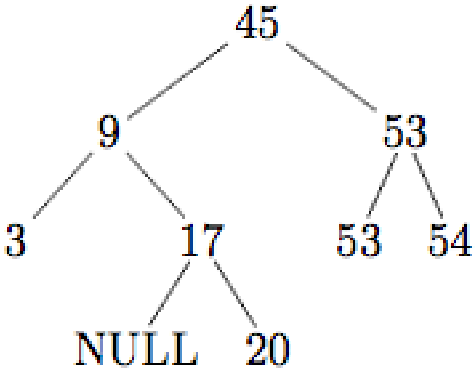
Binary Search Tree Insertion

March 26, 2020

Binary Search Tree Storage Rules

1. Every entry in *n*'s left subtree is less than or equal to the entry in node *n*.
2. Every entry in *n*'s right subtree is greater than (or equal to) the entry in node *n*.


45
9
53
3
17
53
54
NULL
20

Sample Binary Search Tree

```
static DATA_TYPE A[] = { 45, 9, 53, 3, 17, 53, 17, 20, 54 };
static int nA = sizeof(A)/sizeof(DATA_TYPE);

for( int i = 0 ; i < nA ; i++ )
   t1.AddNode( A[i] );

t1:
        54
    53
        53
45
            20
        17
     9
         3
```

Q: Does order of insertion matter?

Binary Search Trees—Code

```
/*  BSTree2.h
 *
 *  Binary Search Tree class Interface WITH deletion.
 */

#ifndef _BSTREE_H_
#define _BSTREE_H_

typedef int  DATA_TYPE;  // Type of node's data

class BinarySearchTree
{
  private:
    typedef struct  BSTreeNode
    {
       DATA_TYPE     data;
       BSTreeNode   *leftPtr;
       BSTreeNode   *rightPtr;
    } *TreePtr;

    TreePtr rootPtr;   //  root of the BST
```

Private Methods

```
void     InitBSTree()
             { rootPtr = NULL; }

void     AddNodeR( TreePtr& t, DATA_TYPE newData );

//  Delete methods

bool     IsLeaf( TreePtr treePtr );

TreePtr  SearchNodeInBST( TreePtr treePtr,
                          DATA_TYPE searchKey );
```

Public Methods

```
 public:
    BinarySearchTree()    { InitBSTree(); }
    ~BinarySearchTree();

    bool     IsEmpty()
                 { return (rootPtr == NULL); }

    void     AddNode( DATA_TYPE newData );
    void     AddNodeR( DATA_TYPE newData );

    void     SearchNode( DATA_TYPE searchKey );
    void     DeleteNode( DATA_TYPE val );

    //  Print methods
};
#endif
```

AddNodeR() – public

```
void BinarySearchTree::AddNodeR( DATA_TYPE newData )
{
    AddNodeR( rootPtr, newData );
}
```

AddNodeR() – private

```
void BinarySearchTree::AddNodeR( TreePtr &t, DATA_TYPE newData )
{
    if( t == NULL )
    {
        TreePtr newPtr = new BSTreeNode;
           // Add new data in the new node's data field
        newPtr->data     = newData;
        newPtr->leftPtr  = NULL;
        newPtr->rightPtr = NULL;

        t = newPtr;
    }
    else if( newData <= t->data )
        AddNodeR( t->leftPtr, newData );
    else
        AddNodeR( t->rightPtr, newData );
}
```

AddNodeR() – private

```
void BinarySearchTree::AddNodeR( TreePtr &t, DATA_TYPE newData )
{
    if( t != NULL )
    {
       if( newData <= t->data )
          AddNodeR( t->leftPtr, newData );
       else
          AddNodeR( t->rightPtr, newData );
    }
    else
    {
        TreePtr newPtr = new BSTreeNode;
           // Add new data in the new node's data field
        newPtr->data     = newData;
        newPtr->leftPtr  = NULL;
        newPtr->rightPtr = NULL;

        t = newPtr;
    }
}
```

AddNode() – Non-recursive 1

```
// AddNode()
//   Add (insert) new item into the BST, whose
//   root node is pointed to by "rootPtr".  If
//   the data already exists, it is ignored.

void BinarySearchTree::AddNode( DATA_TYPE newData )
{
   TreePtr newPtr;

   newPtr = new BSTreeNode;
      // Add new data in the new node's data field
   newPtr->data     = newData;
   newPtr->leftPtr  = NULL;
   newPtr->rightPtr = NULL;

      //  If the BST is empty, insert the new data in root
   if( rootPtr == NULL )
   {
      rootPtr = newPtr;
   }
```

AddNode() – Non-recursive 2

```
else   //  Look for the insertion location
{
   TreePtr   treePtr = rootPtr;
   TreePtr   targetNodePtr;

   while( treePtr != NULL )
   {
     targetNodePtr = treePtr;
     if( newData == treePtr->data )
        // Found same data; ignore it.
        return;
     else if( newData < treePtr->data )
        // Search left subtree for insertion location
        treePtr = treePtr->leftPtr;
     else   //  newData > treePtr->data
        // Search right subtree for insertion location
        treePtr = treePtr->rightPtr;
   }
```

AddNode() – Non-recursive 3

```
         // "targetNodePtr" is the pointer to the parent of
         // the new node.  Decide where it will be inserted.
      if( newData < targetNodePtr->data )
         targetNodePtr->leftPtr = newPtr;
      else  // insert it as its right child
         targetNodePtr->rightPtr = newPtr;
    }
}
```

AddNode() – Differences

- What are the differences between the `AddNode()` and `AddNodeR()` methods?
- Why is this important?

AddNode() – Differences

- What are the differences between the `AddNode()` and `AddNodeR()` methods?

  Traversal: iterative vs. recursive

  `AddNodeR()` is overloaded

  Access: public vs. private

- Why is this important?

  Non-recursive less likely to blow the stack if tree grossly unbalanced.