Binary Search Trees - Motivation

- A linked list is a one-dimensional recursive structure – each node has one pointer to the next node.
  - Problem: finding, deleting or inserting items takes a long time because of the need to linearly search for the item of interest.
- A binary tree is constructed from nodes, where each node contains:
  - A "left" pointer (which could be NULL).
  - A "right" pointer (which could be NULL).
  - A "data" value.
- This leads to a multi-branching recursive structure which can speed up finding, deleting and inserting of items considerably.

Binary Tree Structure

We define a self-referential structure similar to a linked list, but with two pointers, to the "left" and "right" branch:

```c
typedef struct tnode Tnode;
struct tnode {
  int data;
  Tnode *left;
  Tnode *right;
};
```

Binary Trees

- A "root" pointer points to the topmost node in the tree.
- The left and right pointers recursively point to smaller "subtrees" on either side.
- A NULL pointer represents a binary tree with no elements – an empty tree (or subtree).

Binary Search Tree

- A binary search tree is a binary tree in which the items are ordered from left to right across the tree.
- To ensure the items remain ordered, new items must be inserted according to these rules:
  - If the new item has a data value less than the data value at this node, it is recursively inserted into the left subtree.
  - If the new item has a data value greater than the data value at this node, it is recursively inserted into the right subtree.
Building a Binary Search Tree

The shape of the tree depends on the order in which the nodes are inserted:

10, 5, 15, 2, 8, 12
10
/ \
5 15
/ \
2 10
/ \
8 12
/ \
5 12
/ \
10

Question: what if the order is 2, 5, 8, 10, 12, 15?

Binary Search Tree Operations

Tnode * makeTnode(int data); // create new node
Tnode * findTnode(int data, Tnode * root);
Tnode * insertTnode(Tnode *newNode, Tnode *root);
void printTree(Tnode *root); // print all items
void freeTree(Tnode *root); // free entire tree
int treeSize(Tnode *root); // number of items
int treeHeight(Tnode *root); // max depth of an item

Finding a Node in the Tree

// Search tree to find node with specified data value, return NULL if not found
Tnode * findTnode(int data, Tnode * root) {
    Tnode * node = root; // start at root of tree
    while (node != NULL && node->data != data) {
        if (data < node->data) {
            node = node->left;
        } else {
            node = node->right;
        }
    }
    return node;
}
Recursive version of findTnode

```c
Tnode *findTnode(int data, Tnode *root) {
    if (root == NULL || root->data == data) {
        return root;
    } else if (data < root->data) {
        return findTnode(data, root->left);
    } else {
        return findTnode(data, root->right);
    }
}
```

Recursive version of insertTnode

```c
Tnode *insertTnode(Tnode *newNode, Tnode *root) {
    if (root == NULL) {
        root = newNode;
    } else if (newNode->data < root->data) {
        root->left = insertTnode(newNode, root->left);
    } else {
        root->right = insertTnode(newNode, root->right);
    }
    return root;
}
```

Insert New Node into Binary Search Tree

```c
Tnode *insertTnode(Tnode *newNode, Tnode *root) {
    Tnode *parent = NULL;
    while (root != NULL) {
        parent = root;
        if (newNode->data < root->data) {
            child = root->left;
        } else {
            child = root->right;
        }
        if (parent == NULL) {
            root = newNode;
        } else if (newNode->data < root->data) {
            parent->left = newNode;
        } else {
            parent->right = newNode;
        }
    }
    return root;
}
```

Printing a Binary Search Tree

```c
void printTree(Tnode *root) {
    if (root != NULL) {
        printf("%c", root->data);
        printTree(root->left);
        printf("%c", root->data);
        printTree(root->right);
    }
}
```
Freeing all items from a Tree

// Recursively free all the items from a Binary Tree

void freeTree(Tnode *root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

Computing the Size of a Tree

// Compute the size of a Binary Tree
// (the number of items stored in the tree)

int treeSize(Tnode *root) {
    if (root == NULL) {
        return 0;
    }
    else {
        return 1 + treeSize(root->left) + treeSize(root->right);
    }
}

Computing the Height of a Tree

int treeHeight(Tnode *root) {
    int leftHeight, rightHeight;
    if (root == NULL) {
        return 0;
    }
    else {
        leftHeight = treeHeight(root->left);
        rightHeight = treeHeight(root->right);
        if (leftHeight > rightHeight) {
            return 1 + leftHeight;
        } else {
            return 1 + rightHeight;
        }
    }
}

Questions

- what is the maximum number of items that could be stored in a BST with height $H$?
- if the number of items in a BST is $N$, what is the minimum height such a tree could have?
- how would you delete an item from a BST (so that the BST structure is preserved)?