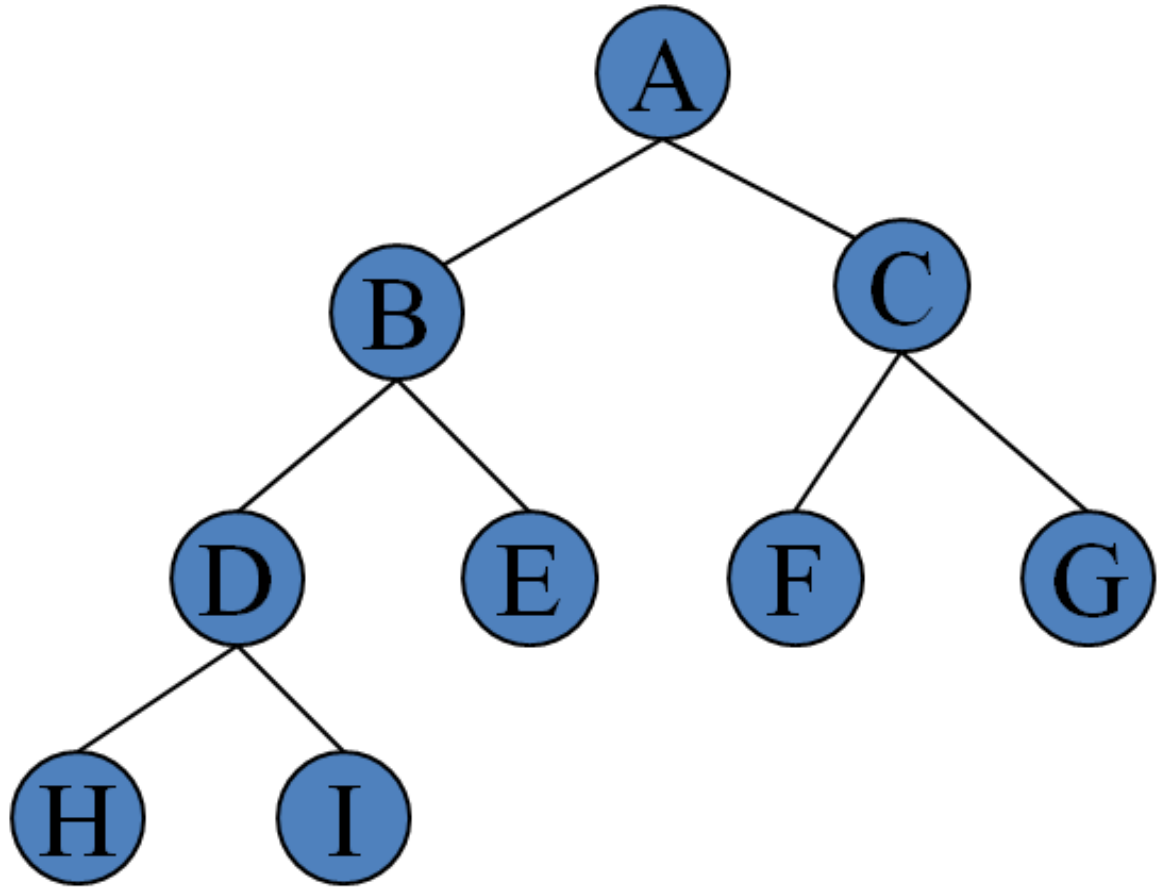
자료구조 실습
BST

http://visualgo.net



Inorder : H – D – I – B – E – A – F – C – G

Preorder : A – B – D – H – I – E – C – F – G

postorder : H – I – D – E – B – F – G – C – A

GetLength

```
// Tree의 node개수를 알려줌
template<class ItemType>
int BinarySearchTree<ItemType>::GetLength()const
{
    return CountNodes(root);              // node 개수를 세는 함수 호출
}
```

```
// Tree의 node 개수를 세는 함수
template<class ItemType>
int CountNodes(Node<ItemType>* root)
{
    if (root == NULL)       // root 노드가 null일경우 0을 리턴
        return 0;
    else
        return CountNodes(root->left) + CountNodes(root->right) + 1;
}
```

Insert

```
// BinarySearchTree에 새로운 노드 추가
template<class ItemType>
void Insert(Node<ItemType>*& root, ItemType item)
{
    if (root == NULL)                // root가 null일 경우
    {
        root = new Node<ItemType>;   // root 노드 생성
        root->left = NULL;           // root 노드이므로 left와 right는 NULL로 설정
        root->right = NULL;
        root->data = item;           // root 노드의 값
    }
    else if (root->data > item)      // root가 존재하고, 그 값이 새로운 item 값보다 클 때
        Insert(root->left, item);    // root의 왼쪽으로 Insert 함수 다시 호출
    else if (root->data < item)      // root가 존재하고, 그 값이 새로운 item 값보다 작을 때
        Insert(root->right, item);   // root의 오른쪽으로 Insert 함수 다시 호출
}
```

Delete

```
// 지우려는 노드를 찾으면 실제로 트리에서 그 노드를 지우는 함수
template<class ItemType>
void DeleteNode(Node<ItemType> *&root)
{
    ItemType item;
    Node<ItemType>* tempPtr;            // 임시 노드를 생성하고 root 노드를 임시 노드에 복사
    tempPtr = root;

    if (root->left == NULL)             // 왼쪽노드가 없을 때
    {
        root = root->right;             // 오른쪽 노드가 root가 되도록 복사하고 임시노드를 지움
        delete tempPtr;
    }
    else if (root->right == NULL)       // 오른쪽노드가 없을 때
    {
        root = root->left;              // 왼쪽 노드가 root가 되도록 복사하고 임시노드를 지움
        delete tempPtr;
    }
    else
    {
        GetPredecessor(root->left, item);   // 중간에 있는 노드를 지우고 싶을 때 (left, right, child 노드 있을 경우)
        root->data = item;                  // 지우려는 노드보다 작은 노드들 중에 가장 큰 노드를 찾음
        Delete(root->left, item);           // 그 값을 지울 노드에 복사를 해서 지운 것처럼 눈속임
    }
}

// 내가 지우려고 하는 노드를 찾는 recursive 함수
template<class ItemType>
void Delete(Node<ItemType> *&root, ItemType item)
{
    if (item < root->data)              // root노드값보다 item노드가 작을 때
        Delete(root->left, item);       // 왼쪽노드로 가서 delete함수 호출
    else if (item > root->data)         // root노드값보다 item노드가 클 때
        Delete(root->right, item);      // 오른쪽노드로 가서 delete함수 호출
    else
        DeleteNode(root);               // 찾고자 하는 값이 일치하는 경우 deletenode 함수 호출
}
```

Search

```
// Tree에서 node를 검색하는 함수
template<class ItemType>
void Retrieve(Node<ItemType> *root, ItemType& item, bool &found)
{
    if (root == NULL)                          // root가 NULL인 경우 found는 false
        found = false;
    else if (item < root->data)                // 찾고자 하는 아이템값이 root값보다 작을 때
        Retrieve(root->left, item, found);     // 왼쪽 노드로 가서 retrieve 함수 호출
    else if (item > root->data)                // 찾고자 하는 아이템값이 root값보다 클 때
        Retrieve(root->right, item, found);    // 오른쪽 노드로 가서 retrieve 함수 호출
    else
    {                                          // 찾고자 하는 값과 일치할 때
        item = root->data;                     // item에 노드 정보를 복사
        found = true;                          // found값을 true로 해서 찾는 과정을 멈춤
    }
}
```