ECE20010 Data Structures

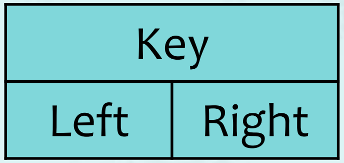
Chapter 5

개인별 숙제 입니다.

- *binary search tree*
  - *Implementation*
- *Homework07 (3 points)*
  - *implement inorderSum().*
  - *submit it in dropbox after clean it up (-1.0 point)*
  - *by Saturday May 3, 11:55 PM*

Chapter 5.7 Binary search trees

Node structure:



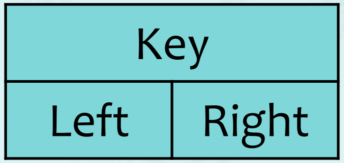
Key

Left

Right

Chapter 5.7 Binary search trees

Node structure:



Key

Left

Right

```
struct node {
   int key;
   struct node *left;
   struct node *right;
};
```

Chapter 5.7 Binary search trees

Node structure:

<table>
<tr><td colspan="2">Key</td></tr>
<tr><td>Left</td><td>Right</td></tr>
</table>

```
struct node {
   int key;
   struct node *left;
   struct node *right;
};
```

Create a new node:

```
struct node *newNode(int item) {
   struct node *aNode = (struct node *)malloc(sizeof(struct node));
   if (aNode == NULL) return NULL;

   aNode->key   = item;
   aNode->left  = NULL;
   aNode->right = NULL;
   return aNode;
}
```

Chapter 5.7 Binary search trees

Insert: insert a new node with given key in BST

```
struct node *insert(struct node *node, int key) {

   if (node == NULL) return newNode(key);    // If empty, return a new node

   if (key < node->key)                      // Otherwise, recur down the tree
      node->left = insert(node->left, key);
   else
      node->right = insert(node->right, key);
   return node;                              // return the (unchanged) node
}
```

Chapter 5.7 Binary search trees

inorder traversal: do inorder traversal of BST.

```
void  inorder(struct node *root) {

   if (root != NULL) {
      inorder(root->left);
      printf("%d ", root->key);
      inorder(root->right);
}
```

Chapter 5.7 Binary search trees

min: find and return the node with minimum key in the given BST.
Note that the entire tree does not need to be searched.

```
struct node *min (struct node *node) {

   struct node *current = node;
   while (current->left != NULL)              // loop down to find the leftmost leaf
      current = current->left;
   return current;
}
```

```
struct node *min(struct node *node) {

   if (node->left == NULL) return(node);
   return min(node->left);
}
```

Chapter 5.7 Binary search trees

deleteNode: delete node with the key and return the new root.

```
struct node *deleteNode(struct node* root, int key) {
   if (root == NULL) return root;                    // base case
   if (key < root->key)                              // then the key to delete lies in left subtree
      root->left = deleteNode(root->left, key);
   else if (key > root->key)                         // then it lies in right subtree
      root->right = deleteNode(root->right, key);
   else {                                            // This is the node to be deleted
      if (root->left == NULL) {                      // node with only one child or no child
         struct node *t = root->right;
         free(root);       return t;
      }
      else if (root->right == NULL) {
         struct node *t = root->left;
         free(root);       return t;
      }
      // implement this case: - node with two children
      // place your code here
   }
   return root;
}
```

Chapter 5.7 Binary search trees

driver: test inorder traversal in BST.

```
void main() {
/****************************
Let us create following BST
          50
         /   \
       40    55
      /  \
    30    45
    / \
  20   35
****************************/
struct node *root = NULL;
  int i;
  int a[] = { 50, 40, 30, 20, 55, 45, 35 };
  int size = sizeof(a) / sizeof(a[0]);

  for (i = 0; i < size; i++)
    root = insert(root, a[i]);

  printf("Inorder traversal\n");
  inorder(root);
  printf("\nsum = %d\n", inorderSum(root));
}
```

Q 1: how many times is inorder() invoked?

Chapter 5.7 Binary search trees

inorderSum: compute the sum of all keys while doing inorder traversal.

Q : trace input argument(root or root->key), push( ) and system stack status every time stack changes, printf( ), sum changes, return status, and pop( ).

```
int  inorderSum(struct node *root) {    // use code tracing purpose only
                                         // use “static” variable(s)


   return sum;
}
```

Call Num	ptr or ptr->key	push()	action printf, sum	return (sum)	pop()
1	50	1.push(50)			
2	40	2.push(40)			

after call 1
system stack
1.push(50)

after call 2
system stack
2.push(40)
1.push(50)