|  |  |  |
| --- | --- | --- |
| American University of Sharjah  College of Engineering  Department of Computer Science & Engineering  P. O. Box 26666, Sharjah, UAE |  | **Instructors:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-126  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Summer 2017 |

**CMP305L - Data Structures and Algorithms Lab**

**Lab. Assignment 10**

**Binary Search Tree as an Array**

***Objectives***

* Implementation of *Binary Search Tree* ADT using arrays.
* Processing *Binary Search Tree*.
* To check if a *Binary Search Tree* is complete.

***Exercise 1***

Add a function to the *BinarySearchTree* class that transforms a complete tree into an equivalent *array.*

***Exercise 2***

Add a function to the *BinarySearchTree* class that returns *true* if the tree is *complete*, otherwise it should return *false.*

template <class T>

void BinarySearchTree<T>::arrayBST(T \* a){

//TreeNode<T>\*\* nodes = new TreeNode<T>\*[LengthIs()];

//TreeNode<T> \*root = new TreeNode<T>(a[i]);

//nodes[0] = root;

//for (int i = 0; i < LengthIs(); i++){

// TreeNode<T> \*parent = nodes[(i + 1) / 2 - 1];

// TreeNode<T>\*node = new TreeNode<T>(arr[i]);

// if (i % 2 == 0){

// parent->left = node;

// }

// else{

// parent->right = node;

// }

// nodes[i] = node;

//}

}

template <class T>

void BSTarray(TreeNode<T>\* tree, T\* a, int index){

}

template <class T>

bool BinarySearchTree<T>::isComplete( TreeNode<T>\* root, unsigned int index,

unsigned int number\_nodes){

if (root == NULL) return (true);

if (index >= number\_nodes) return (false);

return (isComplete(root->left, 2 \* index + 1, number\_nodes) &&

isComplete(root->right, 2 \* index + 2, number\_nodes));

//int rear, front;

//struct TreeNode \*\*que = createQ(&front, &rear);

//bool flag = false;

//if (root == NULL)return true;

//Enqueue(que, &rear, &front);//enqueue?

//while (!IsEmpty()){

// struct TreeNode \*temp = Dequeue(que, &front);

// if (temp->left){

// if (flag = true)return false;

// Enqueue(que, &rear, temp->left);

// }

// else flag = true;

// if (temp->right){

// if (flag == true) return false;

// Enqueue(que, &rear, temp->right);

// }

// else flag = true;

//}

//return true;

}

//

//bool isComplete(struct Node\* root, unsigned int index,

// unsigned int number\_nodes)

//{

// // An empty tree is complete

// if (root == NULL)

// return (true);

//

// // If index assigned to current node is more than

// // number of nodes in tree, then tree is not complete

// if (index >= number\_nodes)

// return (false);

//

// // Recur for left and right subtrees

// return (isComplete(root->left, 2 \* index + 1, number\_nodes) &&

// isComplete(root->right, 2 \* index + 2, number\_nodes));

//}

***NOTE:*** The array MUST be created dynamically.

***Exercise 3***

Develop and test a recursive *function* that takes an *array* of integers (*array* - represents a binary search tree), its *size*, and an integer *value* as its arguments. The function must return the number of *integers* in the array that are less than the *value.*

***NOTE:*** You *MUST* utilize the fact that this array represents a *binary search tree.*