Lec_26.md

Lecture 26

Nov. 17/2020

Note-takers Note: Hi, I hope you enjoyed your reading week!

Announcements

Quiz 5

- Quiz 5 is on November 20th (This Friday)
- Covers Lab 1 to 3

Binary Trees

Trees are a class of **data structure** used in many applications

- **Dynamic** Structures like **linked lists**
- Offer advantages over linear structures like arrays and lists
  - More efficient in *addition* and *deletion* of elements

First, will define a **Binary Tree**

- Then will look at
  - Traversals
  - Ordering (BST - **Binary Search Trees**)
  - Insertion and Deletions on BST's
  - Object-oriented implementation

Binary Tree Description

- A **binary tree** is a structure that is either
  - Empty
  - Consists of one node connected to two disjoint structures
    - Each of which is a binary tree

Any node in a **binary tree** can only have a *minimum* of 0 connections and a *maximum* of 3 connections

- One supernode
  - Nodes above it
  - **Ancestor**
- Two subnodes
  - Nodes below it
  - **Descendent**

Given a node with two subnodes:

- The node is considered a **parent** node
- The two subnodes are considered **children**

- **Left child** and **right child**
    - The children are considered **siblings**
- Think of a family tree
    - Immediate connections in the up/down direction correspond to parents and children

The top (first node) in the tree is called the **root** node

- The tree with **root** node of **left child** of another tree is called the **left subtree**
- Similarly, The tree with **root** node of **right child** of another tree is called the **right subtree**

A node with no subnodes (no children) is called a **leaf** node

Anything that is not a **leaf** node is an **interior node**

- **leaf** nodes are also known as **exterior nodes**

Binary Tree Properties

Unique Parent

There is a unique parent for each node in the tree, except for the root

Prove by Contradiction

- Let `A` be a Node with two parents, `B` and `C`
    - Then `A` has two **supernodes**
    - Then the nodes are *not* disjointed
        - And this is no longer a **binary tree**
- Thus, `A` can only have 1 parent (unique)

Path

The set of nodes `n1,n2,n3,...,nk` is said to be the path from `n1` to `nk` iff (if and only if) `ni` is the parent of `n_(i+1)` for `1 <= i < k`

- Length of a path is `k-1`, or *one less* than the number of nodes in the path

Unique Path

The path between the root and each node in the tree is unique

Prove by Contradiction

- Let `A` be a tree where there are two paths from `A` to a **descendent** `B`
    - Then `B` has two **supernodes**
        - Since there are two paths that reach `B`
    - Then some two nodes are *not* disjointed
        - And this is no longer a **binary tree**
- Thus, the path from **root** node `A` to a **descendent** node `B` ***must be unique***

Binary Tree Implmenetation

TreeNode

```
class treenode{
  public:
    char data;
```

```
    treenode *left;
    treenode *right;
    :
};
```

Notes

1. Tree **node** definition
2. `char data;`

- Value of a given **node**

3. `treenode* left;` and `treenode* right;`

- **children** of a given node

Tree Traversal

Given a binary tree, we would like to visit each node *once and only once*

With a **linked list**:

- There is only *one option* for traversing the **data structure**
  - `->next`
  - **linked lists** are **linear data structures**

With **binary trees**:

- At any node, there are *three options* for traversing **data structure**
  - Visit Node
  - Traverse Left
  - Traverse Right

There are then `3!` traversal options. Examining the most useful 3:

- **In-Order Traversal (LNR)**
  i. Traverse Left
  ii. Visit Node
  iii. Traverse Right
- **Pre-order Traversal (NLR)**
  i. Visit Node
  ii. Traverse Left
  iii. Traverse Right
- **Post-order Traversal (LRN)**
  i. Traverse Left
  ii. Traverse Right
  iii. Visit Node

Preorder Implementation

```
void preorder(treenode* root){
  if(root!=NULL){
    cout << root->data;
    preorder(root->left);
    preorder(root->right);
```

```
    }
  }
```

Notes:

1. You can tell this is a NLR implementation

- `cout << root->data;`
  - Prints **N** data
- `preorder(root->left);`
  - Traverses left
- `preorder(root->right);`
  - Traverses right

2. `if(root!=NULL)`

- **Basis**

Binary Search Tree

Binary Trees are useful, but become more useful when *order* is applied to them

A **Binary Search Tree** (**BST**) is a binary tree with special properties:

- Each node has a *key*
- The **key** of any node is greater than the keys of the nodes in the **left subtree**
- The **key** of any node is less than the keys of the nodes in the **right subtree**

The left and right subtree are also **BST**'s

In general, for a **BST**:

L < N < R