ECE 2400 Computer Systems Programming
Fall 2017

Topic 17: Trees

School of Electrical and Computer Engineering
Cornell University

revision: 2017-11-19-23-43

1 Tree Concepts 2
2 Binary Trees 3
3 Binary Search Trees 7
1. Tree Concepts
2. Binary Trees

- Object-oriented binary tree which stores ints
- Could also use dynamic or static polymorphism to store any type
- Could add iterators to improve data encapsulation

```cpp
class BinaryTreeInt {
    public:
        BinaryTreeInt();
        ~BinaryTreeInt();

        void insert_root(int v);
        void insert_left(Node* node_p, int v);
        void insert_right(Node* node_p, int v);

    private:
        struct Node {
            Node(int v);
            int value;
            Node* left_p;
            Node* right_p;
        };

        Node* m_root_p;
};
```
- Implementation of member functions
- Let’s defer implementing the destructor for now

```cpp
BinaryTreeInt::Node::Node( int v )
  : value(v), left_p(nullptr), right_p(nullptr)
{
}

BinaryTreeInt::BinaryTreeInt()
  : m_root_p(nullptr)
{
}

void BinaryTreeInt::insert_root( int v )
{
    m_root_p = new Node(v);
}

void BinaryTreeInt::insert_left( Node* node_p, int v )
{
    node_p->left_p = new Node(v);
}

void BinaryTreeInt::insert_right( Node* node_p, int v )
{
    node_p->right_p = new Node(v);
}
```
2. Binary Trees

```c
int main( void )
{
    BinaryTreeInt bt;
    bt.insert_root( 10 );

    typedef BinaryTreeInt::Node Node;
    Node* r = bt.m_root_p;
    bt.insert_left ( r, 11 );
    bt.insert_right( r, 12 );
    bt.insert_left ( r->left_p, 13 );

    return 0;
}
```
Recursive member function to print tree

```cpp
void BinaryTreeInt::print_h( Node* node_p ) {
```

Recursive function to delete tree

```cpp
void BinaryTreeInt::clear_h( Node* node_p ) {
```
3. Binary Search Trees

- Recall that set ADTs provide add and contains member functions
- Consider implementing a set ADT with a linked list vs. vector

<table>
<thead>
<tr>
<th></th>
<th>add</th>
<th>contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list (sorted)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vector (sorted)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>binary search tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- A binary search tree is a binary tree with the following invariant:

  For any node in the tree with value v, all values to the left of that node are less than v and all values to the right of that node are greater than v.

- We can use a binary search tree to achieve $O(\log_2(N))$ time complexity for both add and contains
- This time complexity bound Assumes binary tree is balanced which may or may not be a reasonable assumption
Let’s begin by implementing a recursive member function to find which node contains a given value in the tree.

Function should return a pointer to the node with the given value.

For now assume given value is always in the tree.
3. Binary Search Trees

Recursive member function to find node with given value in tree

1 Node* BinaryTreeInt::find_h( Node* node_p, int v ) {

• Now assume given value is not in the tree

• Modify your algorithm to return a pointer to the node which
  would be the parent of where we could insert a new node with the
  new value
Member function to search for value in tree

```cpp
bool BinaryTreeInt::contains( int v ) {
```

Member function to add value to tree

```cpp
void BinaryTreeInt::add( int v ) {
```