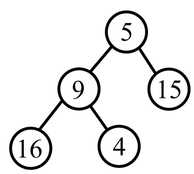
Computer Science 2530
April 6, 2020

Happy Monday, April 6.

Today's topic is binary trees, including what a binary tree is, terminology for binary trees, and how to define elementary functions

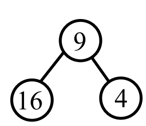
Binary trees

Read page **38A**, which describes binary trees. Here is a sample binary tree, which we refer to as Tree 1.



Terminology is as follows.

- Each circle in the diagram is called a ***node***.
- A ***binary tree*** is either an ***empty tree***, having no nodes, or a ***nonempty tree***, having one or more nodes. An empty tree is represented by a ***null pointer***.
- The ***root*** of nonempty tree is the node at the top, which is the one holding 5 in Tree 1. A nonempty tree is represented by a pointer to its root.
- Each node has two ***subtrees***, its ***left subtree*** and its ***right subtree***. The left subtree of the root of Tree 1 is

  and the right subtree has just one node, containing 15.
- Each node holds three things: an integer called the node's ***item***, a ***pointer to a left subtree*** and a ***pointer to a right subtree***.

- Pointers point downwards in the tree. By convention, we don't show arrows in tree diagrams. Also, by convention, an empty subtree (a null pointer) is not shown in a tree diagram unless the whole tree is empty.
- Suppose v is a node in a binary tree. If the left subtree of v is nonempty, then the root of v's left subtree is called the ***left child*** of v. If the right subtree of v is nonempty, then the root of v's right subtree is called the ***right child*** of v.

  For example, in Tree 1, the node holding 9 is the left child of the root and the node holding 15 is the right child of the root.
- If u is the left or right child of node v, then v is called the ***parent*** of u. For example, the node holding 9 is the parent of the node holding 4 in Tree 1.
- A node that has two empty subtrees is called a ***leaf***.

Exercises

Do the exercises at the bottom of page **38A**.

Trees in C++

Page **38B** shows a definition of type Node; a binary tree is a pointer to a Node. Here is the definition of Node.

```
struct Node
{
  int   item;      // Information at this node
  Node* left;      // The left subtree
  Node* right;     // The right subtree

  Node(int it, Node* lft, Node* rgt)
  {
    item  = it;
    left  = lft;
    right = rgt;
  }
};
```

Notice that a node contains an integer **item** and two pointers, pointing to the left and right subtrees of the node.

Nondestructive functions on binary trees

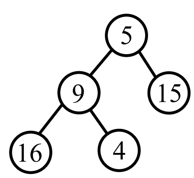
When defining a function on a binary tree, keep these facts in mind.

1. There are two kinds of binary tree: an empty tree and a nonempty tree.
2. An empty tree is a NULL pointer.
3. A nonempty tree is a pointer to a node that has three parts: an item, a left subtree and a right subtree.

A function that works on a tree usually has a case to handle an empty tree and one or more cases to handle nonempty trees.

Example: numNodes(T)

Here is a simple example: function numNodes(T) returns the number of nodes in tree T. An empty tree has no nodes. Look at an example of a nonempty tree, Tree 1 from above.



The left subtree of Tree 1 has 3 nodes. The right subtree has 1 node. Tree 1 has $3 + 1 + 1 = 5$ nodes, counting

(a) the nodes in the left subtree,

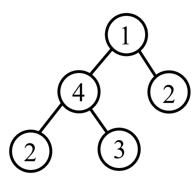
(b) the nodes in the right sutree,

(c) the root.

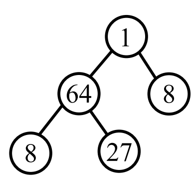
```
int numNodes(const Node* T)
{
  if(T == NULL)
  {
    return 0;
  }
  else
  {
    return 1 + numNodes(T->left) + numNodes(T->right);
  }
}
```

cubes(T)

Function cubes(T) returns a tree that you get by replacing each item x by x^3. For example, if T is tree



then cubes(T) should return the following tree.



Since it cannot change tree T, cubes(T) needs to build new nodes. For that, it uses **new Node(**i**,** L**,** R**)** where i is the desired item in the new Node, L is the desired left subtree and R is the desired right subtree.

Suppose T is a nonempty tree, with item x, left subtree L and right subtree R. Then cubes(T) is a tree whose root is a node with item x^3, whose left subtree is the tree returned by cubes(L) and whose right subtree is the tree returned by cubes(R). Look at the trees above to see that.

Here is a definition of cubes(T) that follows those observations.

```
int cube(int x)
  {
    return x*x*x;
  }

  Node* cubes(const Node* T)
  {
    if(T == NULL)
    {
      return NULL;
    }
    else
    {
      return new Node(cube(T->item), cubes(T->left), cubes(T->right));
    }
  }
```

Reading and exercises

Read page **38C** in the notes and work the exercises at the bottom of the page. Here are some hints.

1. To define numLeaves(T), use top-down design. Create a function isLeaf(T), which returns true if T is a leaf.

   - How many leaves does an empty tree have?
   - How many leaves does a tree have if its root is a leaf?
   - How can you find the number of leaves in a nonempty tree if the root is not a leaf?

   Look at a small example, and use it to guide you in the case of a nonempty tree.

2. Have a case for an empty tree and a case for a nonempty tree.

3. Nonneg(T) returns a tree (a pointer to a Node). What tree should nonneg(NULL) return? If T is nonempty, then nonneg(T) returns a pointer to a node that is constructed using **new Node(**i**,** L**,** R**)** for three particular values i, L and R. What should they be? **Work from an example.**