Chapter 4: Trees

Section 4.1: Typical anary tree representation

Typically we represent an anary tree (one with potentially unlimited children per node) as a binary tree, (one with exactly two children per node). The "next" child is regarded as a sibling. Note that if a tree is binary, this representation creates extra nodes.

We then iterate over the siblings and recurse down the children. As most trees are relatively shallow - lots of children but only a few levels of hierarchy, this gives rise to efficient code. Note human genealogies are an exception (lots of levels of ancestors, only a few children per level).

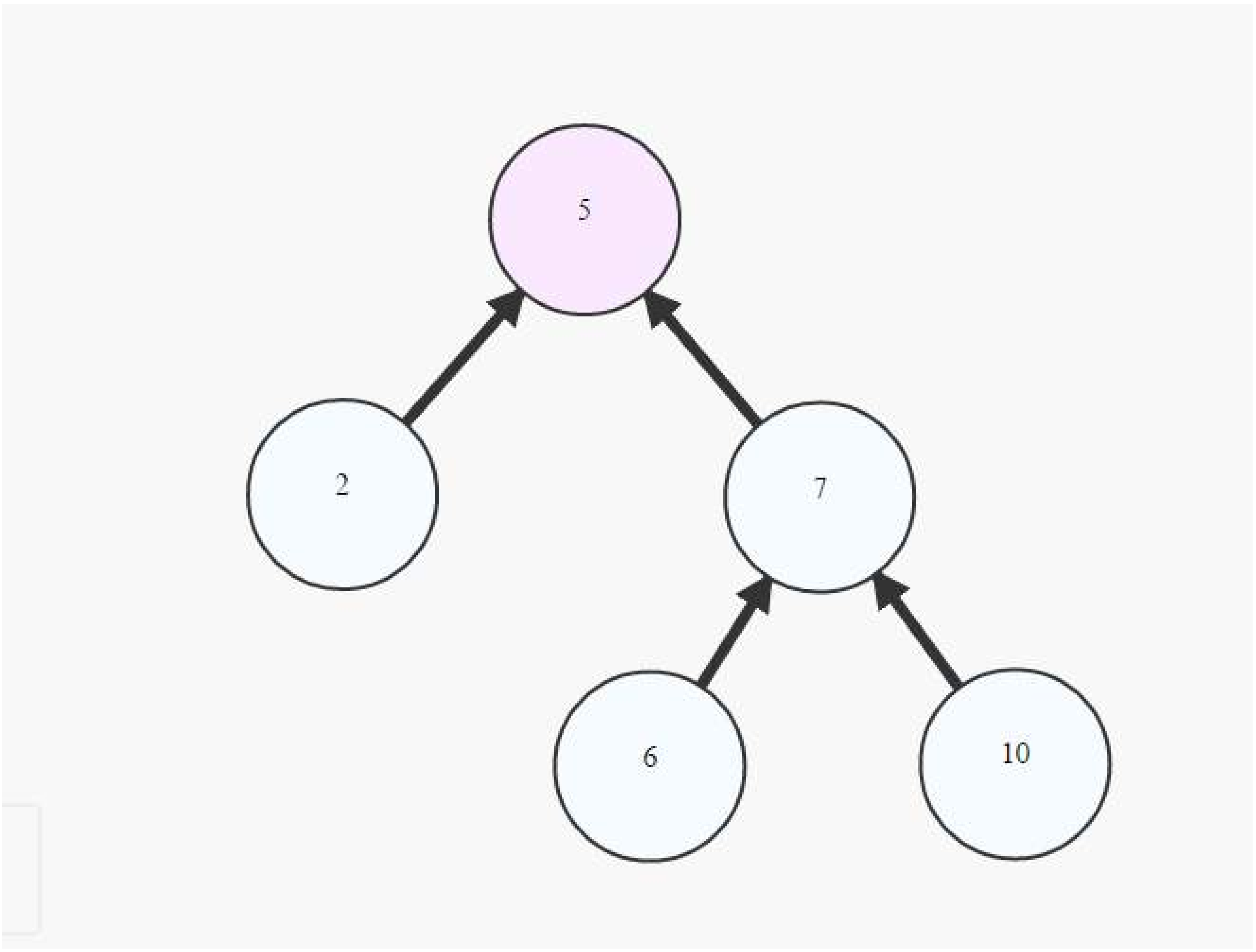
If necessary back pointers can be kept to allow the tree to be ascended. These are more difficult to maintain.

Note that it is typical to have one function to call on the root and a recursive function with extra parameters, in this case tree depth.

```
struct node
{
   struct node *next;
   struct node *child;
   std::string data;
}

void printtree_r(struct node *node, int depth)
{
   int i;

   while(node)
   {
       if(node->child)
       {
          for(i=0;i<depth*3;i++)
              printf(" ");
          printf("{\n"):
          printtree_r(node->child, depth +1);
          for(i=0;i<depth*3;i++)
              printf(" ");
          printf("{\n"):

          for(i=0;i<depth*3;i++)
             printf(" ");
           printf("%s\n", node->data.c_str());

           node = node->next;
         }
     }
}

void printtree(node *root)
{
   printree_r(root, 0);
}
```

Section 4.2: Introduction

Trees are a sub-type of the more general node-edge graph data structure.



To be a tree, a graph must satisfy two requirements:

- **It is acyclic.** It contains no cycles (or "loops").
- **It is connected.** For any given node in the graph, every node is reachable. All nodes are reachable through one path in the graph.

The tree data structure is quite common within computer science. Trees are used to model many different algorithmic data structures, such as ordinary binary trees, red-black trees, B-trees, AB-trees, 23-trees, Heap, and tries.

it is common to refer to a Tree as a `Rooted Tree` by:

```
choosing 1 cell to be called `Root`
painting the `Root` at the top
creating lower layer for each cell in the graph depending on their distance from the root -the
bigger the distance, the lower the cells (example above)
```

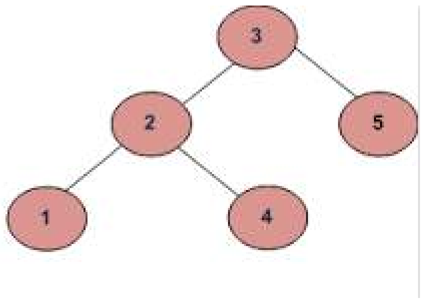
common symbol for trees: `T`

Section 4.3: To check if two Binary trees are same or not

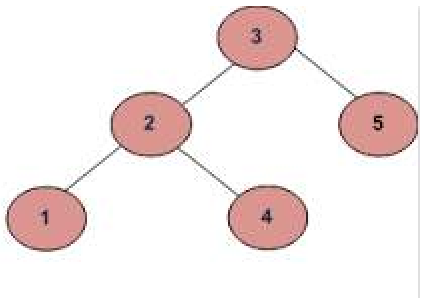
1. For example if the inputs are:

Example:1

a)



b)

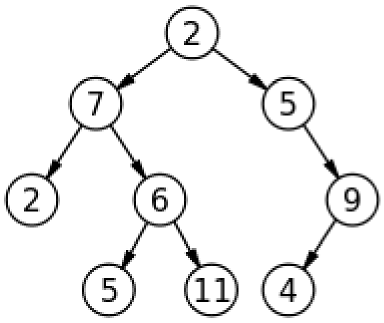


Output should be true.

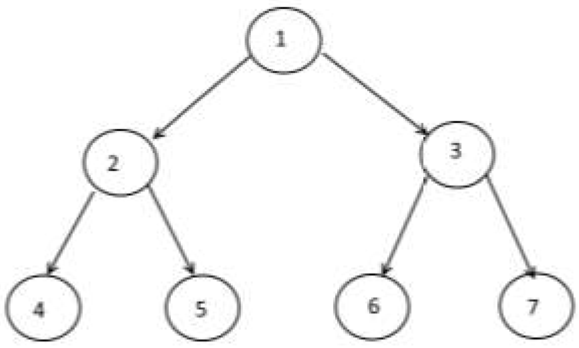
Example:2

If the inputs are:

a)



b)



Output should be false.

Pseudo code for the same:

```
boolean sameTree(node root1, node root2){
```

```
if(root1 == NULL && root2 == NULL)
return true;

if(root1 == NULL || root2 == NULL)
return false;

if(root1->data == root2->data
     && sameTree(root1->left,root2->left)
        && sameTree(root1->right, root2->right))
return true;

}
```