Data Structures and Algorithms

Lab 3 on Trees in C++

ENSIA 2023-2024

Objectives

- Implement Binary Trees, Binary Search Trees, AVL Trees, and B-Trees
- Implement traversal of trees
- Implement operations on trees: insert, delete, search, etc.
- Calculate the complexity of the different operations on trees
- Select the appropriate tree for a given problem
- Familiarize with the data structures `Set` and `Map` in STL (C++ Standard Template Library)

Prerequisites

C++ Classes (1.4), C++ Details (1.5), and Template (1.6) from the course textbook[1].

Refresher (at home)

For the binary tree in Figure 1:

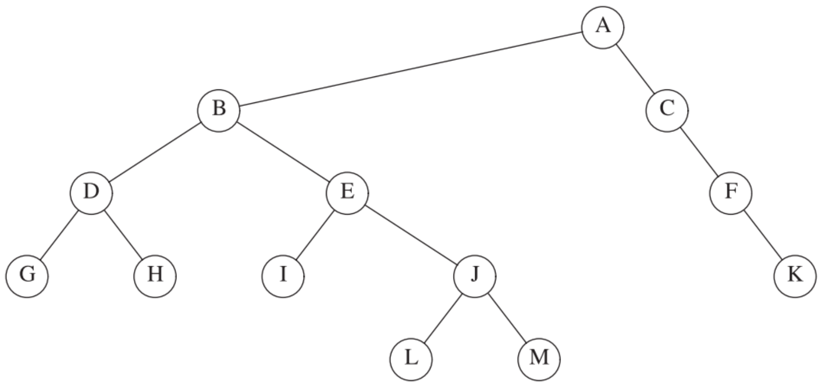
- Which node is the root?
- Which nodes are leaves?
- What is the depth of the tree?
- Give the output of the prefix, infix, and postfix traversals of the tree.

For each node in the tree of Figure 1:

- Name the parent node.
- List the children.
- List the siblings.

[1] Data Structures and Algorithm Analysis in C++, Fourth Edition, Mark Allen Weiss

Figure 1:



- Compute the depth.
- Compute the height.

Exercise 1

Write efficient functions that take only a pointer to the root of a binary tree T and compute:

- The number of nodes in T
- The number of leaves in T
- The number of full nodes in T (nodes which has non-empty left and right children)
- The depth of T
- The printing of the elements in T

What is the running time of your functions?

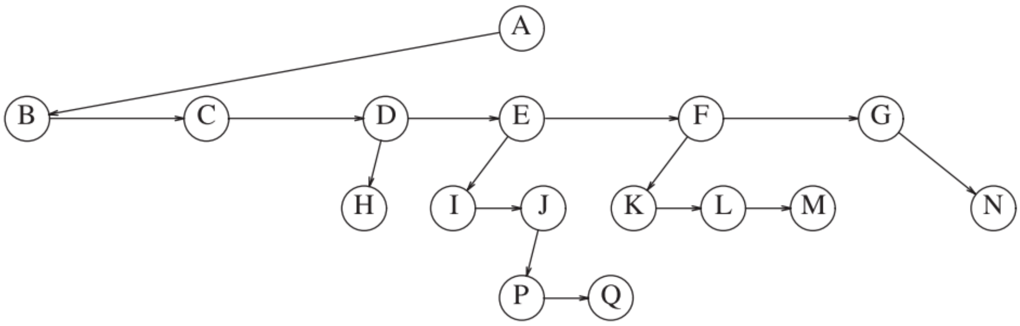
Exercise 2

Write a recursive function that takes a pointer to the root node of a tree T and returns a pointer to the root node of the tree that results from removing all leaves from T.

Exercise 3

Design a recursive linear-time algorithm that tests whether a binary tree satisfies the search tree order property at every node. A binary tree has the ordering property if, for every parent node in the tree, its left child has a smaller value and its right child has a larger value.

Figure 2: child/sibling representation of a tree



Exercise 4

Write a function to generate an N-node random binary search tree with distinct keys 1 through N. What is the running time of your routine?

Exercise 5

Write a function to traverse a tree stored with child/sibling links. In this representation, we keep the children of each node in a linked list of tree nodes. The following declaration is typical:

```
struct TreeNode {
   Object element;
   TreeNode *firstChild;
   TreeNode *nextSibling;
};
```

Figure 2 shows how a tree might be represented in this implementation. The horizontal arrows that point downward are `firstChild` links. The arrows that go left to right are `nextSibling` links. The `Null` links are not drawn because there are too many. For example, in the tree of Figure 2, the node E has both a link to a sibling F and a link to a child I, while some nodes have neither.

Exercise 6

Write a function to generate an AVL tree of height H with fewest nodes. What is its running time?

Exercise 7

Write a non-recursive function to insert a node into an AVL tree.

Exercise 8

1. Write a function to perform insertion into a B-tree.

2. Write a function to perform deletion from a B-tree. When an item is deleted, is it necessary to update information in the internal nodes?
3. Modify the insert function so that if an attempt is made to add a node that already has M entries, a search is performed for a sibling with less than M children before the node is split.

Exercise 9

Let's suppose a B*-tree of order M is a B-tree in which each interior node has between $2M/3$ and M children. Describe and implement a method to perform insertion into a B*-tree.