COMP122/20 - Data Structures and Algorithms

12 Trees

Instructor : Ke Wei 〖柯韋〗

A319 ✆ Ext. 6452 ✉ wke@ipm.edu.mo

http://brouwer.ipm.edu.mo/COMP122/20/

Bachelor of Science in Computing, School of Applied Sciences, Macao Polytechnic Institute

March 9, 2020

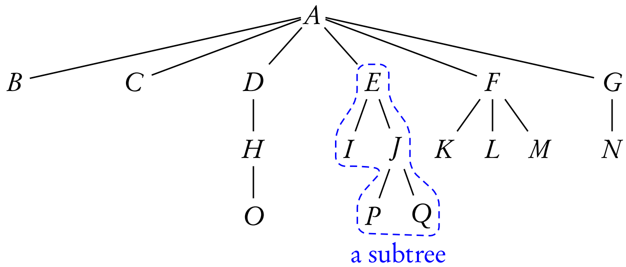
Outline

Textbook §8.1 – 8.2, 8.3.1, 8.3.3, 8.4.1 – 8.4.2, 8.4.6.

General Trees

A tree is

- either empty, or
- a *root* node r which contains an element, and zero or more non-empty *sub-trees*, and there is an edge, which is directional, going from node r to the root node of each subtree.



The tree is a recursive data type.

Parents, Children and Siblings

- The root of each subtree is called a *child* of r, and r is the *parent* of each child. The number of children that a node has is called its *degree*.
- Nodes with the same parent are *siblings*.
- A node with no children (0-degree) is called a *leaf*, or an *external node*; otherwise it is called an *internal node*.
- If the order of the siblings are significant, then the tree is called an *ordered tree*.
- An *unordered tree* can be specified by the set of its edges: $\{parent \to child\}$.

$$\{A\to B,\ A\to C,\ A\to D,\ A\to E,\ A\to F,\ A\to G,\ D\to H,\ E\to I,$$
$$E\to J,\ F\to K,\ F\to L,\ F\to M,\ G\to N,\ H\to O,\ J\to P,\ J\to Q\}$$

Paths and Depths

- A path from node n_1 to n_k is a sequence of nodes $n_1, n_2, \ldots, n_k$ such that

$$n_i \text{ is the parent of } n_{i+1}, \quad \text{for } 1 \leqslant i < k.$$

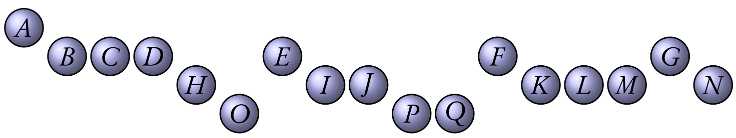
  The number of edges on the path is called its length, that is, $k-1$.
- The *depth* (*level*) of a node is the length of the unique path from the root to the node. The depth of a tree is the depth of the deepest leaf.

  The depth of the root node is 0.
- The *height* of a node is the length of the longest path starting from the node. The height of a tree is the height of its root.

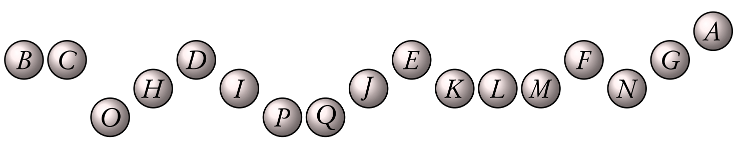
  The depth of a tree equals the height of the tree.
- If there exists a path from node x to node y, then x is the *ancestor* of y and y is a *descendant* of x. If $x \neq y$, then they are called *proper* ancestor and descendant.
- The number of nodes in a tree is called the *size* of the tree.

Tree Traversals

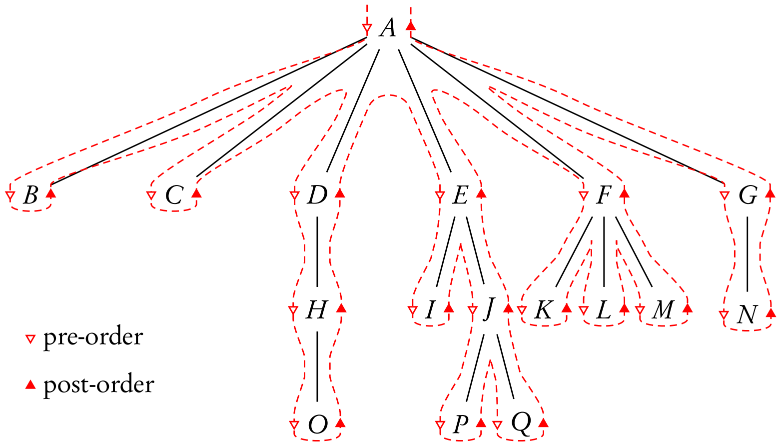
Pre-order traversal: 1) visit the root node;
2) recursively traverse each subtree of the root.



Post-order traversal: 1) recursively traverse each subtree of the root;
2) visit the root node.



Euler Tour Traversal

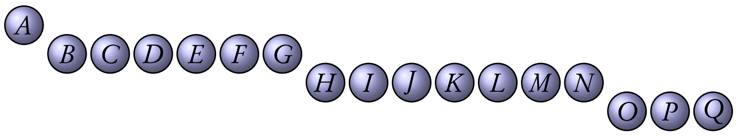


Tree Traversals — Depth First and Breadth First

- Pre-order and post-order traversals are cases of *depth first search*, which can be performed recursively.
- Alternatively, we usually use FIFOs (queues) to perform *breadth first search*. A *BFS* visits the tree nodes in increasing depths.
  - Push-back the root node into an empty queue;
  - While the queue is not empty, do

    Pop a node from the queue, and visit it;

    Push-back all of its children (if any) into the queue.

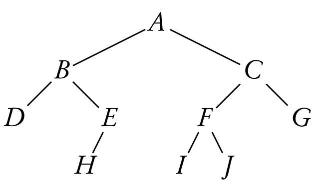


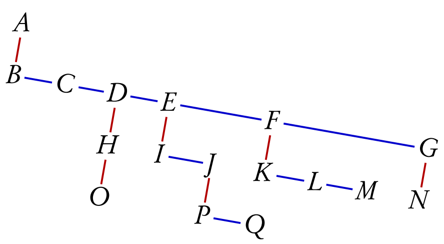
Binary Trees

A binary tree is

- either *empty*, or
- a node with exactly two sub (binary) trees (may all be empty). The two subtrees are called *left* subtree and *right* subtree. A binary tree is an ordered tree.

Binary trees are special cases of trees, however, we can encode general trees as binary trees.

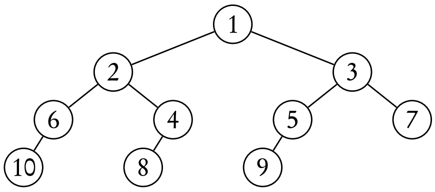




Perfectly Balanced (Binary) Trees

If for every node in a tree, the size difference of its left and right subtrees is at most 1, then the tree is a perfectly balanced tree.

A perfectly balanced tree of size n has depth/height $\lfloor \log n \rfloor$.



To build a perfectly balanced tree from scratch, we insert new nodes to the tree as follows:

- If it is an empty tree, we make the new node the root;
- Otherwise, we recursively insert the node to the right, and then swap the left and right subtrees on the way back.

Representing Binary Trees

- A binary tree can be represented as a reference to a tree node, and we can use `None` to represent an empty tree.

```
class Node:
    def __init__(self, elm, left = None, right = None):
        self.elm = elm
        self.left, self.right = left, right
```

- The preorder generator function of such a binary tree can be recursively defined as follows.

```
def preorder(root):
    if root is not None:
        yield root.elm
        yield from preorder(root.left)
        yield from preorder(root.right)
```


Node Insertion of Perfectly Balanced Trees

- The function *insert_bal* returns the new root of the tree after the insertion of element x.

```
def insert_bal(root, x):
    if root is None:
        return Node(x)
    else:
        root.left, root.right = insert_bal(root.right, x), root.left
        return root
```

- We can then construct the tree on Slide 10 by repeatedly inserting elements beginning from an empty tree.

```
r = None
for x in range(1,13): r = insert_bal(r, x)
```

- We may verify the result by the preorder traversal `list(preorder(r))`. We get `[1, 2, 4, 12, 8, 6, 10, 3, 7, 11, 5, 9]`.