ITS 231: Data Structures and Algorithms Laboratory Semester 1
School of Information, Computer, and Communication Technology
Sirindhorn International Institute of Technology

Lab 6: Binary Trees and their Applications

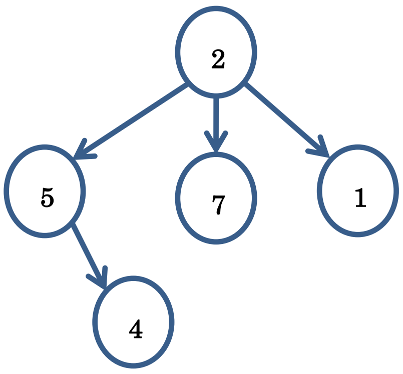
Objectives

- Understand the concept of tree and binary tree data structures and their implementation.
- Understand the basic operations of trees and binary trees.
- Write simple applications that use binary trees.

Review

Trees

- A tree is a hierarchical nonlinear data structure which composes of *nodes* and *branches* where the incoming branch to each node is at most one.



- The ***root*** of the tree is the node in the top of the hierarchy. The indegree of the root is 0.
- *A* ***parent*** is the node of which its outdegree is nonzero.
- *A* ***child*** is the node of which its indegree is nonzero.
- A ***leaf*** node is a node that has no children.
- A ***level (L)*** of a node in a tree is the distance from the root to it.
- A ***height (H)*** of a node in a tree is the number of level of that node + 1Height of the tree

  The height of the tree is the level of the leaf in the longest path from the root. By definition the height of an empty tree is 0. For example the tree that shown below has height equal to 4.

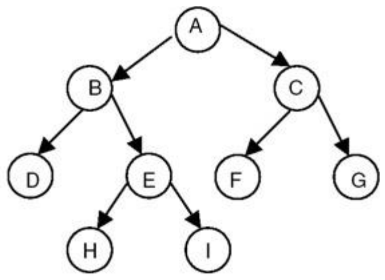
Binary Tree

- A *binary tree (BT)* is a tree in which each node has at most two children.
- A ***balance factor (B) of a node*** is the difference between the height of its left subtree (H_L) and the height of its right subtree (H_R), i.e. $B = H_L – H_R$
- A BT is ***balanced*** if the absolute balance factors (|B|) of all nodes in the tree are either 0 or 1

Binary tree Traversal

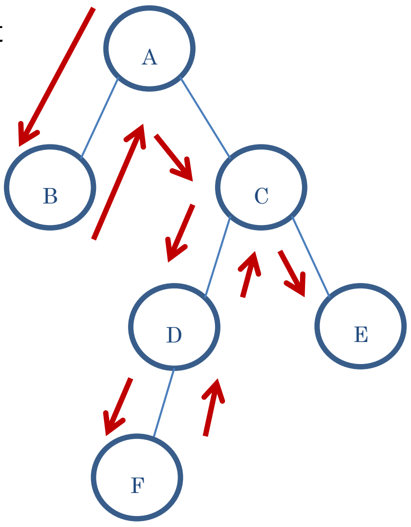
- There are 3 ways to traverse a binary tree according to the order of the root and its left and right node.

- **Preorder** : In the preorder traversal, the root node is visited first, followed by the left subtree and then the right subtree.
- **Inorder** : The inorder traversal visits the left subtree first, then the root, and finally the right subtree
- **Postorder** : The last one is the postorder traversal. It visits the left subtree first, then the right subtree, and finally the root.



Inorder : DBHEIAFCG
Preorder : ABDEHICFG
Postorder : DHIEBFGCA

- There are 2 ways to traverse all nodes according to the orientation of the traversal.
  - **Depth first traversal**: In the **depth-first traversal**, we visit the nodes in the tree along a path from the root through one child to the most distant descendent of that first child before processing a second child. In other words, in the depth-first traversal, we visit all of the descendents of a child before going on to the next child.



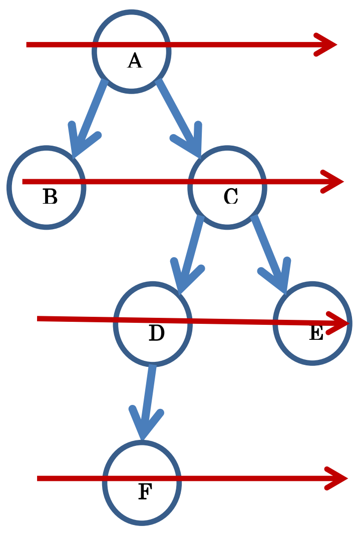
DFT using Stack

- Create a stack S
- Insert the root to S
- Until S is empty
  - Pop and print the top element in S
  - If right child of the popped element exists, push it to S
  - If left child of the popped element exists, push it to S

  - **Breadth first traversal**: In the **breadth-first traversal**, we visit the nodes horizontally from the root to all of its children from left to right, layer by layer from top to bottom.

BFT using Queue

- Create a queue Q
- Insert the root to Q
- Until Q is empty
  - dequeue and print the front element of Q
  - If a left child of the popped element exists, enqueue it to Q
  - If a right child of the popped element exists, enqueue it to Q

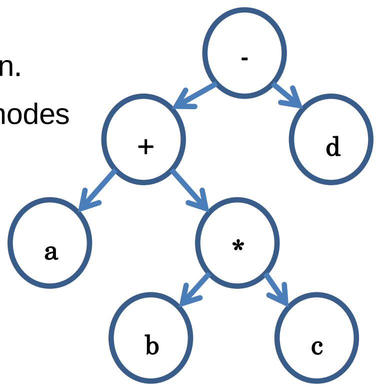


Binary tree Application

Expression tree

- **The expression tree** is the tree that represents an infix expression.
- The leaf nodes of the expression tree are all operand the parent nodes are operators.

  Ex. (a + b*c) -d



Constructing an expression tree

Step 1: create a tree stack and operator stack

Step2: read data in expression

Step3: If data is an operand, push it to the tree stack

Step4: If data is an operator

If it has a higher priority than the top of the stack, push data to the operator stack.

Otherwise, pop an element from an operator stack and pop two elements from an operand stack, then built an expression tree from popped elements. Push this tree to the tree Stack. Try pushing an operator to the operator stack again by repeat step 4

Step 5 If the data in the expression is not done, repeat steps 2-4

<u>Exercise 1</u> Implement the method findHeight(). This method returns the height of the tree. Test the result in the main() method. Use the given template source code to complete this exercise. You may use the given method isLeaf, which takes a BTNode as an input and returns true if the input node is a leaf node (i.e. it has no children). This method is a recursive method.

$$findHeight(root) = \begin{cases} 0, & if\ tree\ is\ empty. \\ 1 + \max\big(findHeight(root\ of\ left\ subtree), \quad findHeight(root\ of\ right\ subtree)\big), & otherwise \end{cases}$$

Exercise 2 Mimic the provided method preorder(BTNode<T> root) provided in the template to implement methods inorder and postorder.

<u>Exercise 3</u> Implement printDFT(). This method prints all nodes in the tree by the depth-first traversal. Use the given template source code to complete this exercise.

Exercise 4 Implement printBFT(). This method prints all nodes in the tree by the breadth-first traversal. Use the given template source code to complete this exercise. You may need to use a queue for this method.

<u>Exercise 5</u> Implement the method makeExpressionTree<T> (Queue<String> Q). In this method the input parameter of the constructor is a queue of String representing an infix expression. There is also a given method hasHigherPriority(String sign1, String sign2) that takes two operators as input strings and returns true if sign1 has higher priority than sign2 and returns false otherwise. The template already provides some part of the code for you. Fill in the missing codes in the constructor. The pseudocode for the constructor is given on the next page.