CH8. TREES

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN JAVA, GOODRICH, TAMASSIA AND GOLDWASSER (WILEY 2016)
WHAT IS A TREE

• In computer science, a tree is an abstract model of a hierarchical structure

• A tree consists of nodes with a parent-child relation

• Applications:
  • Organization charts
  • File systems
  • Programming environments

![Diagram of a tree structure with nodes and relationships]
FORMAL DEFINITION

• A tree $T$ is a set of nodes storing elements in a parent-child relationship with the following properties:
  • If $T$ is nonempty, it has a special node called the root of $T$, that has no parent
  • Each node $v$ of $T$ different from the root has a unique parent node $w$; every node with parent $w$ is a child of $w$

• Note that trees can be empty and can be defined recursively!
• Note each node can have zero or more children
**TREE TERMINOLOGY**

- **Root**: node without parent (A)
- **Internal node**: node with at least one child (A, B, C, F)
- **Leaf** (aka External node): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, great-grandparent, etc.
- **Siblings** of a node: Any node which shares a parent
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Descendant** of a node: child, grandchild, great-grandchild, etc.

- **Subtree**: tree consisting of a node and its descendants
- **Edge**: a pair of nodes \((u, v)\) such that \(u\) is a parent of \(v\) \(((C, H))\)
- **Path**: A sequence of nodes such that any two consecutive nodes form an edge\((A, B, F, J)\)
- **A tree is ordered** when there is a linear ordering defined for the children of each node

![Diagram of a tree with labeled nodes and edges](subtee)
EXERCISE

• Answer the following questions about the tree shown on the right:
  • What is the size of the tree (number of nodes)?
  • Classify each node of the tree as a root, leaf, or internal node
  • List the ancestors of nodes B, F, G, and A. Which are the parents?
  • List the descendants of nodes B, F, G, and A. Which are the children?
  • List the depths of nodes B, F, G, and A.
  • What is the height of the tree?
  • Draw the subtrees that are rooted at node F and at node K.
TREE ADT

- We use positions to abstract nodes, as we don’t want to expose the internals of our structure

- Position functions:
  - $p$.parent() — return parent
  - $p$.children() — list of children positions
  - $p$.isRoot()
  - $p$.isLeaf()

- Tree functions:
  - size()
  - empty()
  - root() — return position for root
  - positions() — return list of all positions

- Additional functions may be defined by data structures implementing the Tree ADT, e.g., begin() and end()
TREE ADT

- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - Integer numChildren(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Additional update methods may be defined by data structures implementing the Tree ADT
A LINKED STRUCTURE FOR GENERAL TREES

- A node is represented by an object storing:
  - Element
  - Parent node
  - Sequence of children nodes

- Node objects implement the Position ADT
PREORDER TRAVERSAL

- A *traversal* visits the nodes of a tree in a systematic manner.
- In a *preorder traversal*, a node is visited before its descendants.
- Application: print a structured document.

**Algorithm** `preOrder(v)`

1. **Input**: Node `v`
2. `visit(v)`
3. for each child `w` of `v`
   3. `preOrder(w)`

---

**Make Money Fast!**

1. Motivations
   - 1.1 Greed
   - 1.2 Avidity
2. Methods
   - 2.1 Stock Fraud
   - 2.2 Ponzi Scheme
   - 2.3 Bank Robbery
3. References
EXERCISE: PREORDER TRAVERSAL

• In a *preorder traversal*, a node is visited before its descendants

• List the nodes of this tree in preorder traversal order.

**Algorithm** `preOrder(v)`

**Input:** Node `v`

1. `visit(v)`
2. *for each* child `w` of `v`
3. `preOrder(w)`
POSTORDER TRAVERSAL

- In a postorder traversal, a node is visited after its descendants.
- Application: compute space used by files in a directory and its subdirectories.

Algorithm postOrder(v)
Input: Node v

1. for each child w of v
2. postOrder(w)
3. visit(v)
EXERCISE: POSTORDER TRAVERSAL

• In a **postorder traversal**, a node is visited after its descendants

• List the nodes of this tree in postorder traversal order.

**Algorithm** `postOrder(v)`

**Input:** Node `v`

1. **for each** child `w` of `v`
2. `postOrder(w)`
3. `visit(v)`
A binary tree is a tree with the following properties:
- Each internal node has two children
- The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- If a child has only one child, the tree is improper
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

Applications:
- Arithmetic expressions
- Decision processes
- Searching
ARITHMETIC EXPRESSION TREE

• Binary tree associated with an arithmetic expression
  • Internal nodes: operators
  • Leaves: operands

• Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$
DECISION TREE

• Binary tree associated with a decision process
  • Internal nodes: questions with yes/no answer
  • Leaves: decisions

• Example: dining decision

Want a fast meal?
- Yes
  - How about coffee?
    - Yes
      - Starbucks
    - No
      - Spike's
- No
  - On expense account?
    - Yes
      - Al Forno
    - No
      - Café Paragon
PROPERTIES OF BINARY TREES

• Notation
  • \( n \) number of nodes
  • \( e \) number of external nodes
  • \( i \) number of internal nodes
  • \( h \) height

• Properties:
  • \( e = i + 1 \)
  • \( n = 2e - 1 \)
  • \( h \leq i \)
  • \( h \leq \frac{n-1}{2} \)
  • \( e \leq 2^h \)
  • \( h \geq \log_2 e \)
  • \( h \geq \log_2 (n+1) - 1 \)
BINARY TREE ADT

• The Binary Tree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT

• Additional position methods:
  • position left(p)
  • position right(p)
  • position sibling(p)

• The above methods return null when there is no left, right, or sibling of p, respectively

• Update methods may also be defined by data structures implementing the Binary Tree ADT
A LINKED STRUCTURE FOR BINARY TREES

• A node is represented by an object storing
  • Element
  • Parent node
  • Left child node
  • Right child node
ARRAY-BASED REPRESENTATION OF BINARY TREES

- Nodes are stored in an array $A$

- Node $v$ is stored at $A[\text{rank}(V)]$
  - $\text{rank}(\text{root}) = 0$
  - if node is the left child of parent(node),
    $\text{rank}(\text{node}) = 2 \times \text{rank}(\text{parent}(\text{node})) + 1$
  - if node is the right child of parent(node),
    $\text{rank}(\text{node}) = 2 \times \text{rank}(\text{parent}(\text{node})) + 2$
INORDER TRAVERSAL

• In an in-order traversal a node is visited after its left subtree and before its right subtree

• Application: draw a binary tree
  • \( x(v) = \) inorder rank of \( v \)
  • \( y(v) = \) depth of \( v \)

Algorithm \( \text{inOrder}(v) \)

Input: Node \( v \)

1. if \( v.\text{left}() \neq \text{null} \) then
2. \( \text{inOrder}(v.\text{left}()) \)
3. visit(v)
4. if \( v.\text{right}() \neq \text{null} \) then
5. \( \text{inOrder}(v.\text{right}()) \)
EXERCISE: INORDER TRAVERSAL

• In an inorder traversal a node is visited after its left subtree and before its right subtree
• List the nodes of this tree in inorder traversal order.

**Algorithm** `inOrder(v)`

**Input:** Node `v`

1. if `v.left() ≠ null` then
2. `inOrder(v.left())`
3. `visit(v)`
4. if `v.right() ≠ null` then
5. `inOrder(v.right())`
EXERCISE: PREORDER & INORDER TRAVERSAL

• Draw a (single) binary tree $T$, such that
  • Each internal node of $T$ stores a single character
  • A preorder traversal of $T$ yields EXAMFUN
  • An inorder traversal of $T$ yields MAFXUEN
APPLICATION
PRINT ARITHMETIC EXPRESSIONS

• Specialization of an inorder traversal
  • print operand or operator when visiting node
  • print “(" before traversing left subtree
  • print ")" after traversing right subtree

Algorithm printExpr(v)
Input: Node v
1. if v.left() ≠ null then
2. print("(")
3. printExpr(v.left())
4. print(v.element())
5. if v.right() ≠ null then
6. printExpr(v.right())
7. print("")

((2 × (a − 1)) + (3 × b))
APPLICATION
EVALUATE ARITHMETIC EXPRESSIONS

- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

Algorithm `evalExpr(v)`

Input: Node `v`

1. if `v.isExternal()` then
2. return `v.element()`
3. `x ← evalExpr(v.left())`
4. `y ← evalExpr(v.right())`
5. `o ← operator stored at v`
6. return `x o y`
EXERCISE
ARITHMETIC EXPRESSIONS

• Draw an expression tree that has
  • Four leaves, storing the values 1, 5, 6, and 7
  • 3 internal nodes, storing operations +, -, *, /
    operators can be used more than once, but each internal node stores only one
  • The value of the root is 21
EULER TOUR TRAVERSAL

- Generic traversal of a binary tree
- Includes as special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)
EULER TOUR TRAVERSAL

**Algorithm eulerTour(v)**

**Input:** Node v

1. leftVisit(v)
2. if v.left() ≠ null then
3.   eulerTour(v.left())
4. bottomVisit(v)
5. if v.right() ≠ null then
6.   eulerTour(v.right())
7. rightVisit(v)
APPLICATION
PRINT ARITHMETIC EXPRESSIONS

• Specialization of an Euler Tour traversal
  • Left-visit: if node is internal, print “(”
  • Bottom-visit: print value or operator stored at node
  • Right-visit: if node is internal, print “)”

\[(2 \times (a - 1)) + (3 \times b)\]
INTERVIEW QUESTION 1

• Implement a function to check if a binary tree is balanced. For the purposes of this question, a balanced tree is defined to be a tree such that the heights of the two subtrees of any node never differ by more than one.

INTERVIEW QUESTION 2

• Given a binary tree, design an algorithm which creates a linked list of all the nodes at each depth (e.g., if you have a tree with depth D, you'll have D linked lists).