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
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 consecutives 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

![Tree Diagram]
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 implementation

• 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
public class GeneralTreeNode<ElementType> implements Position<ElementType> {
    ElementType element;
    GeneralTreeNode<ElementType> parent;
    ArrayList<GeneralTreeNode<ElementType>> children;
    // ... Constructors, accessors, setters
}
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**

**Input:** Tree $T$

1. **preOrder**($T$, $T$.$root()$)

**Algorithm preOrder**

**Input:** Tree $T$, Position $p$

1. **visit-action**($p$)

2. **for each** Position $c \in T$.$children(p)$ **do**

3. **preOrder**($T$, $c$)

---

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

9. 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**

**Input**: Tree $T$

1. $\text{preOrder}(T, T.\text{root}())$

**Algorithm preOrder**

**Input**: Tree $T$, Position $p$

1. $\text{visit-action}(p)$
2. for each Position $c \in T.\text{children}(p)$ do
3. $\text{preOrder}(T, c)$
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
Input: Tree $T$
1. $postOrder(T, T.root())$

Algorithm postOrder
Input: Tree $T$, Position $p$
1. for each Position $c \in T.children(p)$ do
2. $postOrder(T, c)$
3. $visit-action(p)$
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
Input: Tree T
1. postOrder(T, T.root())
```

```
Algorithm postOrder
Input: Tree T, Position p
1. for each Position c ∈ T.children(p) do
2.   postOrder(T, c)
3.   visit-action(p)
```
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
EXAMPLE NODE CLASS FOR BINARY TREE

```java
public class BinaryTreeNode<ElementType> implements Position<ElementType> {
    ElementType element;
    BinaryTreeNode<ElementType> parent, left, right;
    // ... Constructors, accessors, setters
}
```
ARRAY-BASED REPRESENTATION OF BINARY TREES

- Nodes are stored in an array $A$

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

- In an **inorder 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 inOrder**

**Input:** Tree $T$

1. `inOrder(T, T.root())`

```
Algorithm inOrder
Input: Tree T
1. inOrder(T, T.root())

```

2. if $T$.left($p$)$ \neq \text{null}$ then
3. `inOrder(T, T.left(p))`
4. visit-action($p$)
5. if $T$.right($p$)$ \neq \text{null}$ then
6. `inOrder(T, T.right(p))`

```
3 1
2
4
6
8
7
9
1
3
5
4
6
8
7
9
1
3
5
```
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**
**Input:** Tree $T$
1. $\text{inOrder}(T, T.\text{root}())$

**Algorithm inOrder**
**Input:** Tree $T$, Position $p$
1. **if** $T.\text{left}(p) \neq \text{null}$ **then**
2. $\text{inOrder}(T, T.\text{left}(p))$
3. $\text{visit-action}(p)$
4. **if** $T.\text{right}(p) \neq \text{null}$ **then**
5. $\text{inOrder}(T, T.\text{right}(p))$
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 \textit{printExpr}
\textbf{Input:} Tree $T$
1. $\text{printExpr}(T, T.\text{root}())$

\begin{algorithm}
\textbf{Algorithm} \textit{printExpr}
\textbf{Input:} Tree $T$, Position $p$
1. \textbf{if} $T.\text{left}(p) \neq \text{null}$ \textbf{then}
2. \hspace{1em} \text{print} ("")
3. \hspace{1em} $\text{printExpr}(T, T.\text{left}(p))$
4. \hspace{1em} $\text{print}(p.\text{getElement}())$
5. \textbf{if} $T.\text{right}(p) \neq \text{null}$ \textbf{then}
6. \hspace{1em} $\text{printExpr}(T, T.\text{right}(p))$
7. \hspace{1em} \text{print} (""")
\end{algorithm}

\[(2 \times (a - 1)) + (3 \times b)\]
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
Input: Tree $T$
1. evalExpr($T, T.root()$)

Algorithm evalExpr
Input: Tree $T$, Position $p$
1. if $T.isExternal(p)$ then
2. return $p.getElement()$
3. $x \leftarrow$ evalExpr($T, T.left(p)$)
4. $y \leftarrow$ evalExpr($T, T.right(p)$)
5. $\circ \leftarrow$ operator stored at $p$
6. return $x \circ y$
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**

**Input**: Tree $T$

1. $\text{eulerTour}(T, T.\text{root})$

2. $\text{if } T.\text{left}(p) \neq \text{null} \text{ then}$

3. $\text{eulerTour}(T, T.\text{left}(p))$

4. $\text{bottom-visit-action}(p)$

5. $\text{if } T.\text{right}(p) \neq \text{null} \text{ then}$

6. $\text{eulerTour}(T, T.\text{right}(p))$

7. $\text{right-visit-action}(p)$
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).

EXAM 2 - PROGRAMMING

• Hack sheet — you can have a single 8 ½" x 11" paper with handwritten notes on both sides with you during the exam. You can put anything on it, but simple algorithms with iterators might be useful.

• Can bring blank pieces of paper for scratch work.

• Internet only to access the API. Focus is on programming skills.

• Be sure to follow all naming conventions in the exam.

• Format — 2 questions and a bonus (there will be File IO and unit testing in this test)
  • Q1 – Java Generics
  • Q2 – Java Deques or Java Lists with Iterators
  • Bonus — ?
EXAM 2 - WRITTEN

• Separate Hack sheet – you can have a single 8 ½" x 11" paper with handwritten notes on both sides with you during the exam. You can put anything on it, but summary slides and the generic tree traversal algorithms make great candidates.

• Can bring blank sheets of paper for scratch work.

• Focus on algorithmic skills and programming concepts.

• Format – 5 questions and a bonus (will only have to write a single algorithm)
  • Q1 – T/F (similar to quizzes)
  • Q2 – Fill-in-the-blank questions (similar to quizzes)
  • Q3 – Write and/or analyze algorithm using Stacks, Queues, Deques (similar to homework)
  • Q4 – Write and/or analyze algorithm using Lists (similar to homework)
  • Q5 – Write and/or analyze algorithm using Trees (similar to homework)
  • Bonus – ?