Lecture 5: Trees

Nature is Structured in Layers

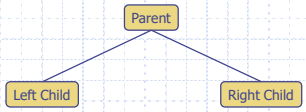
1

Wholeness Statement

Trees are hierarchical data structures that provide wide ranging capabilities and a highly flexible perspective on a set of element objects. *Science of Consciousness*: The whole range of space and time is open to individuals with fully developed awareness. Through the regular twice daily practice of the TM technique, alternated with dynamic activity, we develop more and more of our full potential as demonstrated by 100's of published scientific studies.

2

Binary Trees



3

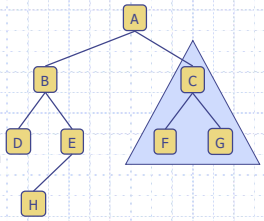
Outline

- BinaryTree ADT
- Preorder and postorder traversals
- Inorder traversal
- Data structures for trees
  - Linked nodes
  - Array based

4

Tree Terminology

- ***Root***: only node without parent (A)
- ***Internal node***: node with at least one child (A, B, C, E)
- ***External node*** (a.k.a. leaf): node without children (D, H, F, G)
- ***Ancestors of a node***: parent, grandparent, grand-grandparent, etc.
- ***Depth of a node***: number of ancestors
- ***Height of a tree***: maximum depth of any node (3 in tree to right)
- ***Descendant of a node***: child, grandchild, grand-grandchild, etc.
- ***Subtree***: tree consisting of a node and its descendants (C, F, G)

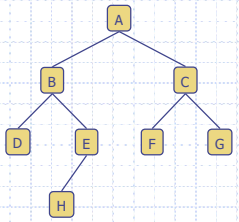


5

Binary Tree

- A (**proper**) binary tree is a tree with the following properties:
  - Each internal node has exactly two children (all nodes in tree to right)
  - Each external node is a null reference (children of D, H, F, G and right child of E which are not shown)
  - The children of an internal node are either internal or external
- **We assume that all binary trees are proper**
- We call the children of an internal node left child and right child
- A binary tree is either
  - a tree consisting of a single external node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree
- What is the height of tree?

- Applications:
  - arithmetic expressions
  - decision processes
  - searching



6

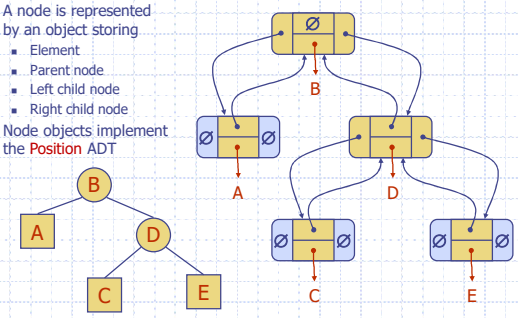
Position and Tree ADT

- A Position or node of a tree is represented by an object storing
  - Element
  - Reference to the Parent node
  - Reference to the two Child nodes
- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - position leftChild(p)
  - position rightChild(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update methods:
  - swapElements(p, q)
  - object replaceElement(p, o)
- Additional update methods would also be defined by data structures implementing the Tree ADT

7

Data Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT



8

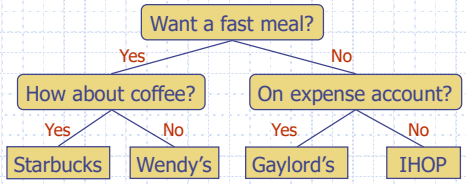
JavaScript Position as used in Binary Trees

```
class TPos {
    constructor(elem, parent, left, right) {
        this._parent = parent;
        this._left = left;
        this._right = right;
        this._elem = elem;
    }
    element() {
        return this._elem;
    }
}
```


9

Decision Tree

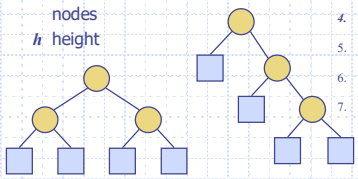
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - internal node with external children: the decisions
- Example: dining decision



10

Properties of Binary Trees

- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height
- Properties:
  1. $n = i + e$
  2. $e = i + 1$
  3. $h \leq i$
  4. $e \leq 2^h$
  5. $\log_2 e \leq h$
  6. $\log_2 (i+1) \leq h$
  7. $\log_2 (i+1) \leq h \leq i$



11

Main Point

1. Each internal node of a Binary Tree has two children and each external node has no children. Thus the height, h, of a binary tree ranges as follows: $\log_2 e \leq h \leq i$, that is, $O(\log_2 n) \leq h \leq O(n)$.

Science of Consciousness: Pure consciousness spans the full range of life, from smaller than the smallest to larger than the largest.

12

BinaryTree ADT

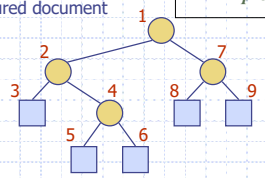
- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)
- Update methods would be defined by data structures implementing the BinaryTree ADT

13

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(T, v)
    if T.isExternal (v) then
        visitExternal()
    else
        visit(v)
        preOrder (T, T.leftChild (v))
        preOrder (T, T.rightChild (v))
```

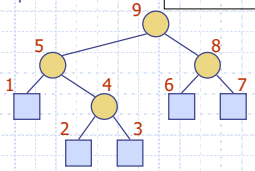


14

Postorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: evaluate a binary expression or type check an expression and its sub-expressions

```
Algorithm postOrder(T, v)
    if T.isExternal (v) then
        visitExternal()
    else
        postOrder (T, T.leftChild (v))
        postOrder (T, T.rightChild (v))
        visit(v)
```

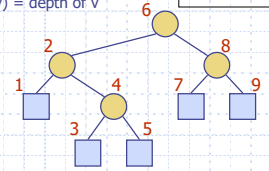


15

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(T, v)
    if T.isExternal (v) then
        visitExternal()
    else
        inOrder (T, T.leftChild (v))
        visit(v)
        inOrder (T, T.rightChild (v))
```

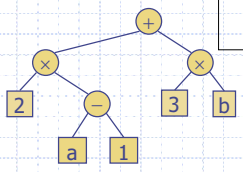


16

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 printExpression(T, v)
    if T.isInternal (T.leftChild(v)) then
        print("(")
        printExpression(T, T.leftChild(v))
    print(v.element())
    if isInternal (T.rightChild(v)) then
        printExpression(T, T.rightChild(v))
        print (")")
```



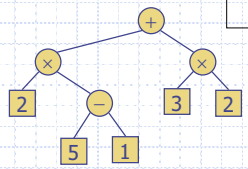
((2 × (a − 1)) + (3 × b))

17

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(T, v)
    if T.isExternal(T.leftChild(v)) then
        return v.element()
    else
        x ← evalExpr(T, T.leftChild(v))
        y ← evalExpr(T, T.rightChild(v))
        ◊ ← operator stored at v
        return x ◊ y
```



18

Exercise on Binary Trees

- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - positionIterator children(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update methods:
  - swapElements(p, q)
  - object replaceElement(p, o)
- Additional BinaryTree methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)

Exercise:

- Write a method to calculate the sum of the integers in a binary tree of integers
  - Assume that an integer is stored at each internal node and **nothing** in external nodes

Algorithm sum(T)

Hint: you also need a helper function with argument Position p

Algorithm sumHelper(T, p)

19

Exercise on Binary Trees

- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - positionIterator children(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update methods:
  - swapElements(p, q)
  - object replaceElement(p, o)
- Additional BinaryTree methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)

Exercise:

- Write a method to calculate the height of a binary tree

Algorithm height(T)

Hint: you also need a helper function with argument Position p

Algorithm heightHelper(T, p)

20

Euler Tour Template (pseudo-code)

```
Algorithm EulerTour(T, v)
    result ← new Array(3)
    if T.isExternal(v) then
        visitExternal(T, v, result)
    else
        visitPreOrder(T, v, result)
        result[0] ← EulerTour(T, T.leftChild(v))
        visitInOrder(v, result)
        result[2] ← EulerTour(T, T.rightChild(v))
        visitPostOrder(T, v, result)

    return result[1]
```


21

Example of the Template Method Pattern in JavaScript

- Generic algorithm that can be specialized by redefining certain steps
- Implemented by means of an abstract JavaScript class
- Visit methods that can be redefined by subclasses
- Template method eulerTour
  - Recursively called on the left and right children
  - A result array r with elements r[0], r[1], and r[2] keeps track of the output of the recursive calls to eulerTour

```
class EulerTour {
  visitExternal(T, p, r) { }
  visitPreOrder(T, p, r) { }
  visitInOrder(T, p, r) { }
  visitPostOrder(p, r) { }
  eulerTour(T, p) {
    let r = new Array(3);
    if (this._tree.isExternal(p)) { this.visitExternal(T, p, r); }
    else {
        this.visitPreOrder(T, p, r);
        r[0] = this.eulerTour(this._tree.leftChild(p));
        this.visitInOrder(T, p, r);
        r[2] = eulerTour(this._tree.rightChild(p));
        this.visitPostOrder(T, p, r);
    }
    return r[1];
  }
```


22

Main Point

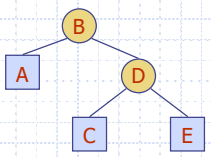
2. The positions (nodes and elements) of a Binary Tree are visited by traversing the tree in one of three ways: pre-order, in-order, post-order. The Euler Tour allows us to traverse any given binary tree in all three ways. The Euler Tour algorithm is non-changing, but we can insert actions (change) during the traversals by overriding the default (hook) methods of the template.

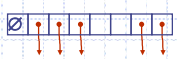
   Science of Consciousness: Pure consciousness is non-changing and supports the everchanging relative creation. When we practice the TM technique, scientific research shows that mind and body are changed for the betterment of the individual and ultimately for society too.

23

Data Structure for Binary Trees

- Another alternative: use an array to store the binary tree.
- Node objects are referenced by index:
  - Index 0 is empty and not used.
  - Root node is at index 1
  - Left child is at 2*index
  - Right child is at 2*index+1





24

Array-Based Implementation of Binary Tree

Operation	Time
size, isEmpty	
positions, elements	
swapElements(p, q), replaceElement(p, e)	
root, parent(p), leftChild(p), rightChild(p)	
isInternal(p), isExternal(p), isRoot(p)	

25

Array-Based Implementation of Binary Tree

Operation	Time
size, isEmpty	1
positions, elements	n
swapElements(p, q), replaceElement(p, e)	1
root, parent(p), leftChild(p), rightChild(p)	1
isInternal(p), isExternal(p), isRoot(p)	1

26

Connecting the Parts of Knowledge with the Wholeness of Knowledge

1. The tree ADT is a generalization of the linked-list in which each tree node can have any number of children instead of just one. A proper binary tree is a special case of the generic tree ADT in which each node has either 0 or 2 children (a left and right child).
2. Any ADT will have a variety of implementations of its operations with varying efficiencies, e.g., the binary tree can be implemented as either a set of recursively defined nodes or as an array of elements.

27

3. **Transcendental Consciousness** is pure intelligence, the abstract substance out of which the universe is made.
4. **Impulses within Transcendental Consciousness**: Within this field, the laws of nature continuously organize and govern all activities and processes in creation.
5. **Wholeness moving within itself** : In Unity Consciousness, awareness is awake to its own value, the full value of the intelligence of nature. One's consciousness supports the knowledge that outer is the expression of inner, creation is the play and display of the Self.

28