Binary Trees

COSC 2011

Definition of Binary Tree

• A binary tree is a tree with the following properties:
  – Each internal node has at most two children
  – The children of a node are an ordered pair (id. as left and right)
• We call the children of an internal node left child and right child
• 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 (left subtree, right subtree)

Applications:
• arithmetic expressions
• decision processes
• searching

Arithmetic Expression Tree

• Binary tree associated with an arithmetic expression
  – internal nodes: operators
  – external nodes: 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
  – external nodes: decisions
• Example: dining decision

Properties of Proper Binary Trees

• Convention: The binary trees we consider are proper binary trees (each node has either zero or two children)
• Level: depth
  – The root is at level 0
  – Level \(d\) has at most \(2^d\) nodes
• Notation:
  – \(n\) number of nodes
  – \(e\) number of external (leaf) nodes
  – \(i\) number of internal nodes
  – \(h\) height

Properties of (General) Binary Trees

• Level: depth
  – The root is at level 0
  – Level \(d\) has at most \(2^d\) nodes
• Notation:
  – \(n\) number of nodes
  – \(e\) number of external (leaf) nodes
  – \(i\) number of internal nodes
  – \(h\) height
Binary Tree ADT

• The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT.
• Additional methods:
  position leftChild(v):
  – returns left child of v.
  – error: v is a leaf node.
  position rightChild(v):
  – returns right child of v.
  – error: v is a leaf node.
  boolean hasLeft(v):
  – returns T if v has a left child; F otherwise.
  boolean hasRight(v):
  – returns T if v has a right child; F otherwise.

Preorder Traversal of a General Tree

• 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

Preorder Traversal of a Binary Tree

Algorithm binPreorder(T, v) {
  visit(v);
  if T.hasLeft(v) then
    binPreorder(T, T.leftChild(v));
  if T.hasRight(v) then
    binPreorder(T, T.rightChild(v));
}

Postorder Traversal of a General Tree

Algorithm postOrder(T, v) {
  for each child w of v
    postOrder(T, w);
  visit(v);
}

Postorder Traversal of a Binary Tree

Algorithm binPostorder(T, v) {
  if T.hasLeft(v) then
    binPostorder(T, T.leftChild(v));
  if T.hasRight(v) then
    binPostorder(T, T.rightChild(v));
  visit(v);
}

Inorder Traversal

Algorithm inOrder(T, v) {
  if T.hasLeft(v) {
    inOrder(T, T.leftChild(v));
    visit(v);
    if T.hasRight(v) {
      inOrder(T, T.rightChild(v));
    }
  }
}
Example: Drawing a Binary Tree

- Draw the above binary tree.
- Write a program for drawing a binary tree using the specs given above.

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)`

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

Array-Based Implementation

Each node v is stored at index p(v) defined as follows:
- If v is the root, p(v) = 1
- If v is the left child of node u, p(v) = 2.p(u)
- If v is the right child of node u, p(v) = 2.p(u) + 1

Space Analysis of Array-Based Implementation

- \( n \): number of nodes of binary tree \( T \)
- \( p_M \): index of the rightmost leaf of the corresponding full binary tree (or size of the full tree)
- \( N \): size of the array needed for storing \( T \); \( N = p_M + 1 \)

Best-case scenario: balanced, full binary tree
- Height \( h = (n - 1)/2 \)

Worst case scenario: unbalanced tree
- Size of the corresponding full tree:
  - \( p_M = 2^{h+1} - 1 = 2^{(n+1)/2} - 1 \)
  - \( N = 2^{n+1}/2 \)
- Space usage: \( O(2^{n+1}/2) \) or \( O(2^n) \)

Time Analysis of Array-Based Implementation

- All the tree and binary tree methods run in time \( O(1) \), except positions() and elements() each taking \( O(n) \).
- Homework: implement the tree and binary tree methods using an extendable array
- Efficient time-wise, but inefficient space-wise if the binary is unbalanced
- Alternative: ?

Linked Structure-Based Implementation

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT
An example of Linked Structure Implementation

A Linked 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

A Linked Structure for General Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes (list or vector)
- Node objects implement the Position ADT

Linked Structure for General Trees: Analysis

- Space usage: $O(n)$
- All the tree methods take time $O(1)$, except
  - positions(), elements(): $O(n)$
  - children(v): $O(c_v)$, $c_v = \text{number of children of } v$
- Note: we store the children of each node $v$ in a container (list or vector). We implement children(v) by calling method elements() of that container.

Representing General Trees with Binary Trees

Procedure:
- Start from the root
- The first child of a node $v$ becomes the left child of $v$
- The other children of $v$ form a chain of right children rooted at the left child

Homework: write a program that converts a general tree to a binary tree (using tree and binary tree methods)

Another Example
Implementing General Trees by Means of Binary Trees: Analysis

- Result: an unbalanced binary tree
- Best implementation for this binary tree: ?
- All the tree methods take time $O(1)$, except
  - positions(), elements(): $O(n)$
  - children(v): $O(c_v)$, $c_v$ = number of children of v
  - parent(v): $O(s_v)$, $s_v$ = number of siblings of v
- Homework: implement the tree ADT using binary tree as the data structure

Euler Tour Traversal

- Generic traversal of a binary tree
- Includes 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)

Algorithm eulerTour(T, v) {
    perform "on the left" action for v; // 1
    if v is an internal node
        eulerTour(T, T.leftChild(v));
    perform "from below" action for v; // 2
    if v is an internal node
        eulerTour(T, T.rightChild(v));
    perform "on the right" action for v; // 3
}

- Each node is processed 3 times
- Assuming each action takes $O(1)$, the overall running time is $O(n)$

Examples

- Print the contents of the above tree using Euler Tour traversal
- Algorithm "print arithmetic expressions"
  Algorithm printExpr(v) {
    if isInternal(v) {
        print('('); // "on the left" action
        printExpr(leftChild(v));
    }
    print(v.element()); // "from below" action
    if isInternal(v) {
        printExpr(rightChild(v));
        print(')'); // "on the right" action
    }
}