Treel11: Trees I

Bohyung Han
CSE, POSTECH
bhhan@postech.ac.kr

Tree Data Structures

- An abstract model of a hierarchical structure
  - “2-dimensional” structure
  - Easy access (similar to binary search in array)
  - Easy insert and removal (similar to linked list)

Computers’R’Us

Sales
Manufacturing
R&D

US
International
Laptops
Desktops

Europe
Asia
Canada

A Hierarchical Structure

- Unix/Linux file systems

![Hierarchical Structure Example]

Computers’R’Us

Sales
Manufacturing
R&D

US
International
Laptops
Desktops

Europe
Asia
Canada

Tree Data Structures

- An abstract model of a hierarchical structure
  - Trees consist of nodes and links denoting parent-child relation
  - Special case of graphs without loops
  - Each element (except root) has a parent and zero or more children.
**Terminology**

- **Root**: node without parent
- **Internal node**: node with at least one child
- **Leaf**: node without children
- **Subtree**: tree consisting of a node and its descendants
- **Ancestor** of a node: parent, grandparent, grand-grandparent ...
- **Descendant** of a node: child, grandchild, grand-grandchild ...
- **Depth (level)** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)

**Tree Abstract Data Type**

- **Data structure**
  - Node: data element and link to children
  - **Root**: a special node
- **List of methods**

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>root()</td>
<td>Return the tree’s root; error if tree is empty</td>
</tr>
<tr>
<td>parent(v)</td>
<td>Return v’s parent; error if v is a root</td>
</tr>
<tr>
<td>children(v)</td>
<td>Return v’s children (an iterable collection of nodes)</td>
</tr>
<tr>
<td>isRoot(v)</td>
<td>Test whether v is a root</td>
</tr>
<tr>
<td>isExternal(v)</td>
<td>Test whether v is an external node</td>
</tr>
<tr>
<td>isInternal(v)</td>
<td>Test whether v is an internal node</td>
</tr>
</tbody>
</table>

**A Linked Structure for Trees**

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

![Diagram of a linked structure for trees](image)
A Linked Structure for Trees

Binary Tree

- Definition
  - a tree in which each node has at most two children
  - Each child is either the left child or the right child of its parent.

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
Proper Binary Trees

- Each internal node has exactly 2 children.
- Properties
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq \frac{n - 1}{2}$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

  | $n$: number of total nodes |
  | $e$: number of external nodes |
  | $i$: number of internal nodes |
  | $h$: height (maximum depth of a node) |

A Linked Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node

A Linked Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node

Binary Tree Implementation

```java
public class Node {
    int value; // data stored at node
    Node left; // pointer to left child
    Node right; // pointer to right child

    public Node(int i) {
        value = i;
        left = null;
        right = null;
    }

    public Node getLeftChild(Node node) { return left; }
    public Node getRightChild(Node node) { return right; }
    public void setLeftChild(Node node) { left = node; }
    public void setRightChild(Node node) { right = node; }
}
```
Binary Tree Implementation

```java
public class BinaryTree {
    Node root;

    public Tree() { root = null; }

    // insert an element to the current tree
    public void insert(int i) { ... }

    // delete a node from the current tree
    public Node delete(int i) { ... }

    // find a node containing a particular value
    public Node find(int i) { ... }
}
```

An Array-Based Representation

- Nodes are stored in an array A.
- Node v is stored at A[rank(v)]
  - rank(root) = 1
  - Left in even: if node is the left child of parent(node), rank(node) = 2 \cdot rank(parent(node))
  - Right in odd: if node is the right child of parent(node), rank(node) = 2 \cdot rank(parent(node)) + 1
- A[0] is always empty
- A[i] is empty if there is no node in the i-th position
- The array size N is \(2^{(h+1)}\)

An Array-Based Representation

![Binary Tree Representation](image)

Tree Balance

- Full binary tree (perfect binary tree)
  - A binary tree of height h is full if every node has exactly two children and all leaf nodes have the same level.
Tree Balance

- Complete binary tree
  - A binary tree is complete if it is full to level h-1 and level h is filled from the left with contiguous nodes.