Data Structures & Programming

Binary Trees
Vector Implementation & Traversals

Golnar Sheikhshhab
Vector Implementation of Binary Trees

Based on level numbering

- If $v$ is the root of $T$, then $f(v) = 1$
- If $v$ is the left child of node $u$, then $f(v) = 2f(u)$
- If $v$ is the right child of node $u$, then $f(v) = 2f(u) + 1$
Figure 7.17: Representation of a binary tree $T$ by means of a vector $S$. 
Complexity analysis

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>left, right, parent, isExternal, isRoot</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>size, empty</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>root</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>expandExternal, removeAboveExternal</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>positions</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Table 7.3: Running times for a binary tree $T$ implemented with a vector $S$. We denote the number of nodes of $T$ with $n$, and $N$ denotes the size of $S$. The space usage is $O(N)$, which is $O(2^n)$ in the worst case.
Traversals of a binary tree

**Algorithm** binaryPreorder($T, p$):

1. perform the “visit” action for node $p$
2. **if** $p$ is an internal node **then**
   1. binaryPreorder($T, p$.left())
   2. binaryPreorder($T, p$.right())

**Algorithm** binaryPostorder($T, p$):

1. **if** $p$ is an internal node **then**
   1. binaryPostorder($T, p$.left())
   2. binaryPostorder($T, p$.right())
2. perform the “visit” action for the node $p$

**Algorithm** inorder($T, p$):

1. **if** $p$ is an internal node **then**
   1. inorder($T, p$.left())
   2. perform the “visit” action for node $p$
2. **if** $p$ is an internal node **then**
   1. inorder($T, p$.right())
Evaluating an Arithmetic Expression

**Algorithm** `evaluateExpression(T, p)`:  

1. **if** `p` is an internal node **then**  
   1. `x ← evaluateExpression(T, p.left())`  
   2. `y ← evaluateExpression(T, p.right())`  
   3. Let ◦ be the operator associated with `p`  
   4. **return** `x ◦ y`  
2. **else**  
   1. **return** the value stored at `p`
Binary Search Tree
Binary Search Tree

Do we have a binary search tree when we do binary search?

Time analysis of search in a binary search tree?

How to sort using a binary search tree?
Using Inorder Traversal for Tree Drawing

- $x(p)$ is the number of nodes visited before $p$ in the inorder traversal of $T$.
- $y(p)$ is the depth of $p$ in $T$.
The Euler Tour Traversal of a Binary Tree
The Euler Tour Traversal of a Binary Tree (2)

**Algorithm** `eulerTour(T, p)`:

- perform the action for visiting node `p` on the left
- **if** `p` is an internal node **then**
  - recursively tour the left subtree of `p` by calling `eulerTour(T, p.left())`
- perform the action for visiting node `p` from below
- **if** `p` is an internal node **then**
  - recursively tour the right subtree of `p` by calling `eulerTour(T, p.right())`
- perform the action for visiting node `p` on the right
The Euler Tour Traversal of a Binary Tree (3)

**Algorithm** templateEulerTour(T, p):

```
r ← initResult()
if p.isExternal() then
    r.finalResult ← visitExternal(T, p, r)
else
    visitLeft(T, p, r)
    r.leftResult ← templateEulerTour(T, p.left())
    visitBelow(T, p, r)
    r.rightResult ← templateEulerTour(T, p.right())
    visitRight(T, p, r)
return returnResult(r)
```
Reading Material

Sections 7.3.5 and 7.3.6 of the textbook

Optional: sections 7.3.6 (The Template Function Pattern) mainly to see an instance of using class inheritance