Trees 2: Section 4.2 and 4.3

Binary trees
Definition: A *binary tree* is a rooted tree in which no vertex has more than two children.
Definition: A *binary tree* is a rooted tree in which no vertex has more than two children.
Binary Trees

Definition: A *binary tree* is a rooted tree in which no vertex has more than two children.

```c
struct BinaryNode
{
  Object element;  // The data in the node
  BinaryNode *left;  // Left child
  BinaryNode *right;  // Right child
};
```
Example: Expression Tree
Binary Trees

Definition: A binary tree is complete iff every layer but the bottom is fully populated with vertices.
Binary Trees

A complete binary tree with \( n \) vertices and height \( H \) satisfies:

- \( 2^H \leq n < 2^H + 1 \)
- \( 2^2 \leq 7 < 2^2 + 1 \)
Binary Trees

A complete binary tree with $n$ vertices and height $H$ satisfies:

- $2^H \leq n < 2^H + 1$
- $H \leq \log n < H + 1$
- $H = \text{floor}(\log n)$
Binary Trees

Theorem: In a complete binary tree with $n$ vertices and height $H$

- $2^H \leq n < 2^H + 1$
Binary Trees
Binary Trees

Proof:
Binary Trees

Proof:

- At level $k \leq H-1$, there are $2^k$ vertices
Binary Trees

Proof:
- At level $k \leq H-1$, there are $2^k$ vertices
- At level $k = H$, there are at most $2^H$ vertices
Binary Trees

Proof:

- At level \( k \leq H-1 \), there are \( 2^k \) vertices
- At level \( k = H \), there are \textit{at most} \( 2^H \) vertices
- Total number of vertices:
Binary Trees

Proof:
- At level $k \leq H-1$, there are $2^k$ vertices
- At level $k = H$, there are at most $2^H$ vertices
- Total number of vertices:
  $$n = 2^0 + 2^1 + \ldots + 2^k$$
Binary Trees

Proof:

- At level \( k \leq H-1 \), there are \( 2^k \) vertices
- At level \( k = H \), there are at most \( 2^H \) vertices
- Total number of vertices:
  \[ n = 2^0 + 2^1 + \ldots + 2^k \]
  \[ n = 1 + 2^1 + 2^2 + \ldots + 2^k \] (Geometric Progression)
Binary Trees

Proof:

○ At level $k \leq H-1$, there are $2^k$ vertices
○ At level $k = H$, there are at most $2^H$ vertices
○ Total number of vertices:
  ○ $n = 2^0 + 2^1 + ... 2^k$
  ○ $n = 1 + 2^1 + 2^2 + ... 2^k$ (Geometric Progression)
  ○ $n = (2^k + 1 - 1) / (2-1)$
Binary Trees

Proof:

- At level $k \leq H-1$, there are $2^k$ vertices.
- At level $k = H$, there are at most $2^H$ vertices.
- Total number of vertices:
  - $n = 2^0 + 2^1 + ... + 2^k$
  - $n = 1 + 2^1 + 2^2 + ... 2^k$ (Geometric Progression)
  - $n = (2^k + 1 - 1) / (2-1)$
  - $n = 2^k + 1 - 1$
Binary Trees
Binary Trees

Let $k = H$
Binary Trees

- Let $k = H$
  - $n = 2^H + 1 - 1$
Binary Trees

Let $k = H$

- $n = 2^H + 1 - 1$
- $n < 2^H + 1$
Binary Trees

- Let \( k = H \)
  - \( n = 2^H + 1 - 1 \)
  - \( n < 2^H + 1 \)
- Let \( k = H - 1 \)
Binary Trees

- Let $k = H$
  - $n = 2^H + 1 - 1$
  - $n < 2^H + 1$
- Let $k = H - 1$
  - $n' = 2^H - 1$
Binary Trees

- Let $k = H$
  - $n = 2^H + 1 - 1$
  - $n < 2^H + 1$

- Let $k = H - 1$
  - $n' = 2^H - 1$
  - $n \geq n' + 1 = 2^H$
Binary Trees

- Let $k = H$
  - $n = 2^H + 1 - 1$
  - $n < 2^H + 1$
- Let $k = H - 1$
  - $n' = 2^H - 1$
  - $n \geq n' + 1 = 2^H$
  - $2^H \leq n < 2^H + 1$
Binary Tree Traversals
Binary Tree Traversals

- Inorder traversal
  - Definition: left child, visit, right child (recursive)
  - Algorithm: depth-first search (visit between children)
Inorder traversal

```
root  
  1  
 /     
2       3
/     /
4     5 6
     /
    7
```
Inorder traversal
Inorder traversal

root

2

4

5

3

6

7
Inorder traversal

```
           root 1
          /   /
         2    3
        / 
       4   5
         / 
        6   7
```
Inorder traversal

```
        root 1
       /   \
      2     3
     /     /     \
    4     5       6
        /           /
       7           
```
Inorder traversal

```
    root
     /\    /
     1  2  /\ 3
    / \  / \ /
   4  5 6  7
```
Inorder traversal

```
root  1

2       3
4      5  6  7
```
Inorder traversal
Inorder traversal

```
        root
       /   \
      2     3
     /     /  \
    4     6    7
```
Inorder traversal
Inorder traversal

root 1

2

4 5

3

6

7
Inorder traversal
Inorder traversal
Binary Tree Traversals
Binary Tree Traversals

- Other traversals apply to binary case:
Binary Tree Traversals

- Other traversals apply to binary case:
  - Preorder traversal
    - vertex, left subtree, right subtree
Binary Tree Traversals

- Other traversals apply to binary case:
  - Preorder traversal
    - vertex, left subtree, right subtree
  - Inorder traversal
    - left subtree, vertex, right subtree
Binary Tree Traversals

- Other traversals apply to binary case:
  - Preorder traversal
    - vertex, left subtree, right subtree
  - Inorder traversal
    - left subtree, vertex, right subtree
  - Postorder traversal
    - left subtree, right subtree, vertex
Binary Tree Traversals

Other traversals apply to binary case:

- Preorder traversal
  - vertex, left subtree, right subtree
- Inorder traversal
  - left subtree, vertex, right subtree
- Postorder traversal
  - left subtree, right subtree, vertex
- Levelorder traversal
  - vertex, left children, right children
Vector Representation of Complete Binary Tree
Vector Representation of Complete Binary Tree

- Tree data
  - Vector elements carry data
Vector Representation of Complete Binary Tree

- Tree data
  - Vector elements carry data

- Tree structure
  - Vector indices carry tree structure
  - Index order = levelorder
  - Tree structure is implicit
  - Uses integer arithmetic for tree navigation
Vector Representation of Complete Binary Tree
Vector Representation of Complete Binary Tree

- Tree navigation
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2 \times k + 1]$
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2*k + 1]$
  - Right child of $v[k] = v[2*k + 2]$

```
  root 0
   / \  / \  /  \
  l   r rl  rr
  l1  l2  rl1 rl2
```
Vector Representation of Complete Binary Tree
Vector Representation of Complete Binary Tree

Tree navigation

```
0  1  2  3  4  5  6
0  
```
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2*k + 1]$

```
0  1  2  3  4  5  6
0               1
```
Vector Representation of Complete Binary Tree

- **Tree navigation**
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2\times k + 1]$
  - Right child of $v[k] = v[2\times k + 2]$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑
Vector Representation of Complete Binary Tree
Vector Representation of Complete Binary Tree

Tree navigation
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2 \times k + 1]$
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2k + 1]$
  - Right child of $v[k] = v[2k + 2]$

```
0 1 2 3 4 5 6
0 1
```

↑
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2k + 1]$
  - Right child of $v[k] = v[2k + 2]$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2*k + 1]$
  - Right child of $v[k] = v[2*k + 2]$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>r</td>
<td>ll</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2*k + 1]$
  - Right child of $v[k] = v[2*k + 2]$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>r</td>
<td>ll</td>
<td>lr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2k + 1]$
  - Right child of $v[k] = v[2k + 2]$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>r</td>
<td>ll</td>
<td>lr</td>
<td>rl</td>
<td></td>
</tr>
</tbody>
</table>
Vector Representation of Complete Binary Tree

- Tree navigation
  - Parent of $v[k] = v[(k - 1)/2]$
  - Left child of $v[k] = v[2k + 1]$
  - Right child of $v[k] = v[2k + 2]$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Binary Search Tree

```
        4
       /|
      2 6
     /  |
    1  5 7
```

Binary Search Tree

Also known as Totally Ordered Tree
Binary Search Tree

- Also known as Totally Ordered Tree

Definition: A binary tree $B$ is called a binary search tree iff:
Binary Search Tree

- Also known as Totally Ordered Tree

- Definition: A binary tree $B$ is called a binary search tree iff:
  - There is an order relation $\leq$ defined for the vertices of $B$
Binary Search Tree

- Also known as Totally Ordered Tree

- Definition: A binary tree $B$ is called a binary search tree iff:
  - There is an order relation $\leq$ defined for the vertices of $B$
  - For any vertex $v$, and any descendant $u$ of $v.left$, $u \leq v$
Binary Search Tree

- Also known as Totally Ordered Tree

- Definition: A binary tree $B$ is called a binary search tree iff:
  - There is an order relation $\leq$ defined for the vertices of $B$
  - For any vertex $v$, and any descendant $u$ of $v$.left, $u \leq v$
  - For any vertex $v$, and any descendant $w$ of $v$.right, $v \leq w$
Binary Search Tree
Binary Search Tree

- Consequences:
Binary Search Tree

- Consequences:
  - The smallest element in a binary search tree (BST) is the “left-most” node
Binary Search Tree

Consequences:
- The smallest element in a binary search tree (BST) is the “left-most” node
- The largest element in a BST is the “right-most” node
Binary Search Tree

- Consequences:
  - The smallest element in a binary search tree (BST) is the “left-most” node
  - The largest element in a BST is the “right-most” node
  - Inorder traversal of a BST encounters nodes in increasing order
Binary Search using BST
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
  - Begin at root node
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
  - Begin at root node
  - Descend using comparison to make left/right decision
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
  - Begin at root node
  - Descend using comparison to make left/right decision
    - if (search_value < node_value) go to the left child
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
  - Begin at root node
  - Descend using comparison to make left/right decision
    - if (search_value < node_value) go to the left child
    - else if (search_value > node_value) go to the right child
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
  - Begin at root node
  - Descend using comparison to make left/right decision
    - if (search_value < node_value) go to the left child
    - else if (search_value > node_value) go to the right child
    - else return true (success)
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
  - Begin at root node
  - Descend using comparison to make left/right decision
    - if (search_value < node_value) go to the left child
    - else if (search_value > node_value) go to the right child
    - else return true (success)
  - Until descending move is impossible
Binary Search using BST

- Assumes nodes are organized in a totally ordered binary tree
  - Begin at root node
  - Descend using comparison to make left/right decision
    - if (search_value < node_value) go to the left child
    - else if (search_value > node_value) go to the right child
    - else return true (success)
  - Until descending move is impossible
  - Return false (failure)
Binary Search using BST

- Runtime <= descending path length <= depth of tree
- If tree has enough branching, runtime <= O(log size)
Reading assignment for next class

Section 4.3