Comp 352 Winter 2019
Tutorial Week 5

February 6, 2019

Outline

1. Tree: Definition
2. Binary Tree
3. Traversal
4. Problem solving
5. Priority Queue
6. Heap
7. Credits

Announcement

1. Stock Span
2. Demo
3. Assignment 1

Tree: Definition

1. Tree: $T = (V, E)$, each element in a tree has 1 parent element (with the exception of the root element) and 0 or more children elements.
2. siblings: same parent
3. ancestor $\Rightarrow$ descendant. A node u is an ancestor of a node v if $u = v$ or u is an ancestor of the parent of v.
4. internal: one or more children
5. external: no children
6. leaves: external nodes

Tree: Definition

1. edge: $e = (u, v)$, u is the parent of v, or vice versa.
2. path: $P = v_1 \rightsquigarrow e_1 \rightsquigarrow v_2 \rightsquigarrow e_2 \rightsquigarrow \cdots$

Depth

- depth(v): The number of ancestors of v, excluding v itself.
- It can also be recursively defined as follows:
  - If v is the root, then the depth of v is 0.
  - Otherwise, the depth of v is one plus the depth of the parent of v.

Algorithm 1 depth(T, v):

Input: T a tree
Input: v a node of T

```
if v is the root of T then
    return 0
else
    return 1 + depth(T, parent(v))
end if
```

Height

- height(v): It is recursively defined as follows:
  - If v is an external, then the height of v is 0.
  - Otherwise, the depth of v is one plus the maximum height of a child of v.

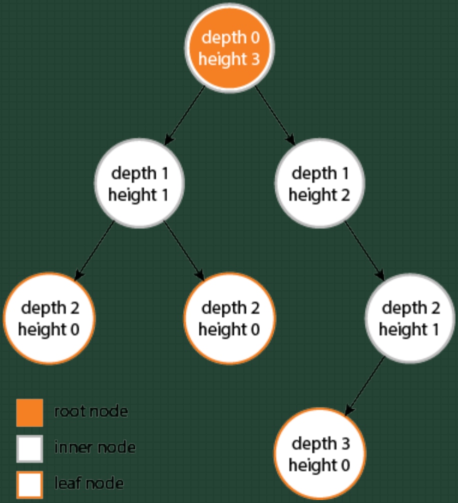
Height

Algorithm 2 height(T, v):

Input: T a tree
Input: v a node of T

```
if v is an external node in T then
    return 0
else
    h = 0
    for each child w of v do
        if h < height(T, w) then
            h = height(T, w)
        end if
    end for
    return 1 + h
end if
```

Depth V.S. Height



Binary Tree

A binary tree is an ordered tree with the following properties:

1. Each internal node has at most two children (exactly two for proper (full) binary trees)
2. The children of a node are an ordered pair

Traversal

- Preorder
- Postorder
- Inorder
- Breadth first traversal

Preorder

In a preorder traversal of a tree T, the root of T is visited first and then the subtrees rooted at its children are traversed recursively.

Algorithm 3 preorder(T, v):

Input: T a tree
Input: v a node of T

```
visit(v)
for each child w of v do
    preorder(T, w)
end for
```

Postorder

A postorder traversal recursively traverses the subtrees rooted at the children of the root first, and then visits the root.

Algorithm 4 postorder(T, v):

Input: T a tree
Input: v a node of T

for each child w of v **do**
    postorder(T, w)
end for
visit(v)

Inorder (Binary tree)

An inorder traversal recursively traverses the left subtrees rooted at the children of the root first then visits the root and then traverses the right subtree.

Algorithm 5 inOrder(T, v):

Input: T a tree
Input: v a node of T

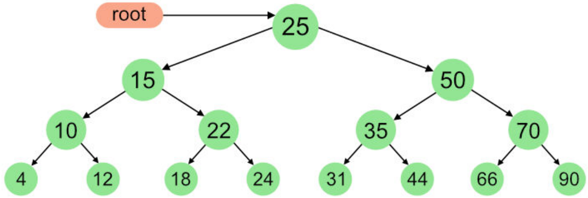
```
if left(v) ≠ null then
    inOrder(T,left(v))
end if
visit(v)
if right(v) ≠ null then
    inOrder(T,right(v))
end if
```

Traversal

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Breadth first traversal

Traverses the tree level by level

Problem solving

Let T be an ordered tree with more than one node.

1. Is it possible that the preorder traversal of T visits the nodes in the same order as the postorder traversal of T?
2. Likewise, is it possible that the preorder traversal of T visits the nodes in the reverse order of the postorder traversal of T?

Problem solving

Give an $O(n)$-time algorithm for computing the depth of all the nodes of a tree T, where n is the number of nodes of T.

Priority Queue

A priority queue stores a collection of entries.
Each entry is a pair $(key, value)$.

Priority Queue Sorting

1. Selection-Sort: unsorted sequence
2. Insertion-Sort: sorted sequence
    - In-place Insertion-Sort

Heap

A heap is a binary tree storing keys at its nodes and satisfying the following properties:

1. Heap-Order: for every internal node v other than the root, $key(v) \geq key(parent(v))$
2. Complete Binary Tree
   - Complete V.S. Proper (Full)
3. Last node: rightmost node of maximum depth

Upheap

1. Find the insertion node z (the new last node)
2. Store k at z
3. Restore the heap-order property

Downheap

1. Replace the root key with the key of the last node w
2. Remove w
3. Restore the heap-order property
    - Choose the child with minimal key (why?)

Credits

1. https://stackoverflow.com/questions/2603692/what-is-the-difference-between-tree-depth-and-height
2. https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/