CS213/293 Data Structure and Algorithms 2024

Lecture 5: Tree

Instructor: Ashutosh Gupta

IITB India

Compile date: 2024-08-28

Let us study

tree data structure,

which will help us solving many problems including the problem of dictionary.

Commentary: The purpose of programs is to solve problems. We need not invent a data structure until we have a purpose. The purpose will be clarified by the next lecture.

Topic 5.1

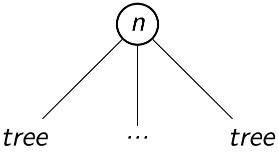
Tree

Tree

Definition 5.1

A tree is either a node

or the following structure consisting of a node and a set of children trees that are disjoint.



The above is our first recursive definition.

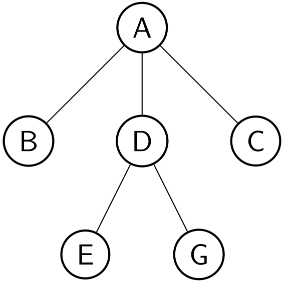
Exercise 5.1

Does the above definition include infinite trees? How would you define an infinite tree?

Example: tree

Example 5.1

An instance of tree.



Some tree terminology(2)

For nodes n_1 and n_2 in a tree T.

Definition 5.2

n_1 is child of n_2 if n_1 is immediately below n_2. We write $n_1 \in children(n_2)$.

Definition 5.3

We say n_2 is parent of n_1 if $n_1 \in children(n_2)$ and write $parent(n_1) = n_2$.
If there is no such n_2, we write $parent(n_1) = \perp$.

Definition 5.4

n_1 is ancestor of n_2 if $n_1 \in parent^*(n_2)$. We write $n_1 \in ancestors(n_2)$.
n_2 is descendant of n_1 if $n_1 \in ancestor(n_2)$. We write $n_1 \in descendants(n_2)$.

Commentary: For a function $f(x)$, we define $f^*(x) = y | y = f(..f(x))$,i.e., the function is applied 0 or more times (informal definition). What would be a mathematically formal definition?

Some tree terminology

Definition 5.5
n_1 and n_2 are siblings if $parent(n_1) = parent(n_2)$.

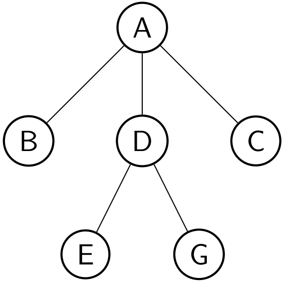
Definition 5.6
n_1 is a leaf if $children(n_1) = \emptyset$.
n_1 is an internal node if $children(n_1) \neq \emptyset$.

Definition 5.7
n_1 is a root if $parent(n_1) = Null$.

Exercise 5.2
Can the root be an internal node? Can the root be a leaf?

Example: Tree terminology

Example 5.2



B, D, and C are children of A.

D is the parent of G.

A is an ancestor of G and E is a descendant of A.

A is an ancestor of A.

G and E are siblings.

B, E, G, and C are leaves.

A and D are internal nodes.

A is the root.

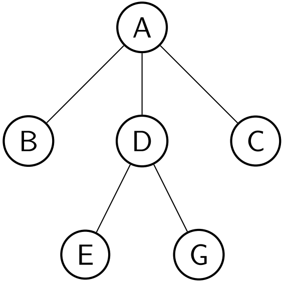
Degree of nodes

Definition 5.8

We define the degree of a node n as follows.

$$degree(n) = |children(n)|$$

Example 5.3



$$dgree(A) = 3$$

$$dgree(B) = 0$$

$$dgree(D) = 2$$

Label of tree

Usually, we store data on the tree nodes.

We define the $label(n)$ of a node n as the data stored on the node.

Level/Depth and height of nodes

Definition 5.9

We define the level/depth of a node n as follows.

$$level(n) = \begin{cases} 0 & \text{if } n \text{ is a root} \\ level(n') + 1 & n' = parent(n) \end{cases}$$

Definition 5.10

We define the height of a node n as follows.

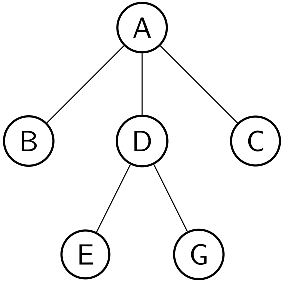
$$height(n) = max(\{height(n') + 1 | n' \in children(n)\} \cup \{0\})$$

Exercise 5.3

Why do we need to take a union with 0 *in the definition of height?*

Example: Level(Depth) and height of nodes

Example 5.4



$level(A) = 0$

$level(B) = 1$

$level(E) = 2$

$height(E) = 0$

$height(D) = 1$

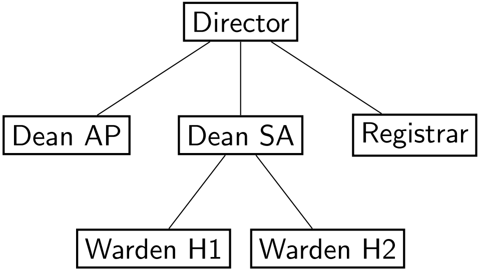
$$\begin{aligned} height(A) &= max(\{height(B) + 1, \\ &\qquad height(D) + 1, \\ &\qquad height(C) + 1\} \cup \{0\}) \\ &= max(\{1, 2, 1\} \cup \{0\}) = 2 \end{aligned}$$

Why do we need trees?

A tree represents a hierarchy.

Example 5.5

- *Organization structure of an organization*

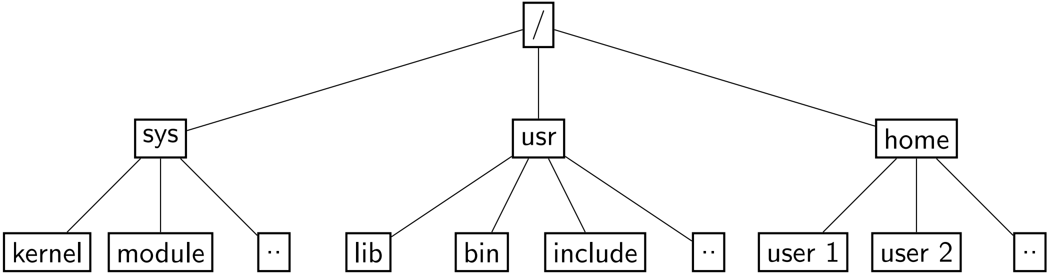


Example: File system

Files are stored in trees in Linux/Windows.

Example 5.6

Part of a Linux file system.



Topic 5.2

Binary tree

Ordered tree

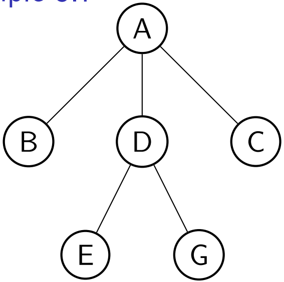
Definition 5.11

A tree is an ordered tree if we assign an order among children.

Definition 5.12

Let n be a node. In an ordered tree, $children(n)$ is a list instead of a set.

Example 5.7



In a tree, we define the children as follows.
$children(A) = \{B, D, C\}$

In an ordered tree, we define the children as follows.
$children(A) = [B, D, C]$

Binary tree

Definition 5.13

An ordered tree T is a binary tree if $|children(n)| \leq 2$ for each $n \in T$.

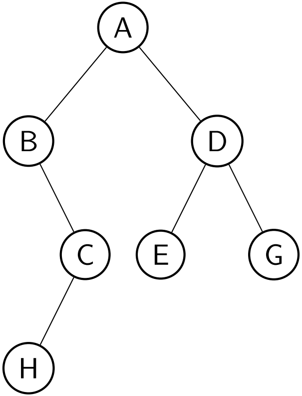
We define the left and right child of n as follows.

- *if $chidren(n) = [n_1, n_2]$,*
  - $left(n) = n_1$ *and* $right(n) = n_2$.
- *If $chidren(n) = [n_1]$, n_1 is either left or right child.*
  - $left(n) = n_1$ *and* $right(n) = Null$, *or*
  - $left(n) = Null$ *and* $right(n) = n_1$.
- *If $chidren(n) = []$,*
  - $left(n) = Null$ *and* $right(n) = Null$.

Commentary: For a mathematical nerd, the given definition of left/right child is not satisfactory. How can we interpret $chidren(n) = [n_1]$ in two possible ways? There is an alternative way to define the binary tree. We may say that there are "Null" nodes, which are the leaves. By definition, all internal nodes will have two children. $chidren(n) = [n_1]$ will be written as either $chidren(n) = [Null, n_1]$ or $chidren(n) = [n_1, Null]$. Hence, we will have a clean definition of left and right child. For $chidren(n) = []$, we will write $chidren(n) = [Null, Null]$. This issue will come up again in Red-Black tree. Meanwhile, we will stick to our definition.

Example: binary tree

Example 5.8

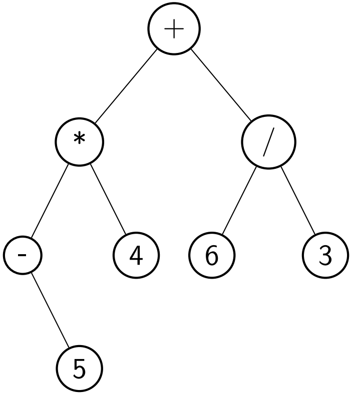


E is the left and G is the right child of D. C is the right child of B. B has no left child.

Usage of binary tree: representing expressions

Example 5.9

Representing mathematical expressions



Exercise 5.4

a. Why do we need an ordered tree?

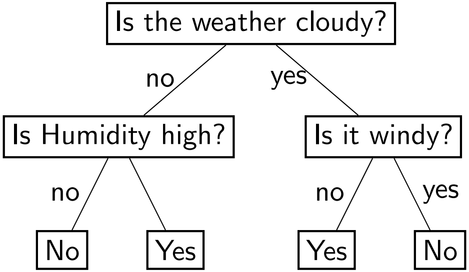
b. How would you evaluate a mathematical expression given as a binary tree?

Usage of binary tree: decision trees in AI

Example 5.10

Does one want to play given the weather?

Given the behavior, we may learn the following tree.



Complete binary tree (*aka* perfect binary tree)

Commentary: Some sources define the complete trees differently, where they allow last level to be not filled and all nodes are as left as possible. They also define full and balanced trees. Do an internet search.

Definition 5.14

A binary tree is complete if the height of the root is h and every level $i \leq h$ has 2^i nodes.

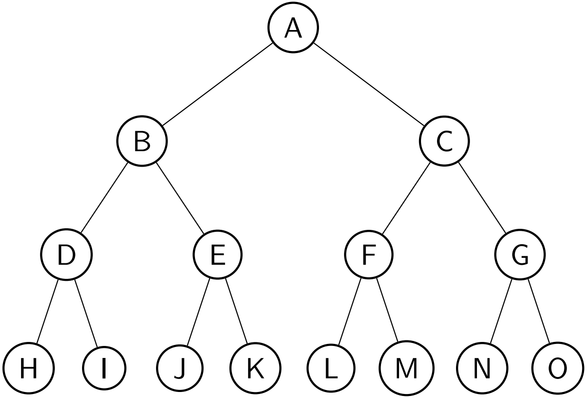
Leaves are only at level h.

The number of leaves $= 2^h$.

Number of internal nodes $= 1 + 2 + ... + 2^{h-1}$
$= 2^h - 1$.

The total number of nodes is $2^{h+1} - 1$.

Example 5.11



Exercise 5.5

a. Prove/Disprove: if no node in the binary tree has a single child, the binary tree is complete.
b. What fraction of nodes are leaves in a complete binary tree?

Maximum and minimum height of a binary tree

Exercise 5.6

Let us suppose there are n nodes in a binary tree.

- *What is the minimum height of the tree?*
- *What is the maximum height of the tree?*

Commentary: For a given height h, a complete binary tree has $2^{h+1} - 1$ nodes. All other binary trees with the height h have fewer nodes. Therefore, $n \leq 2^{h+1} - 1$. Therefore, $log_2 \frac{n+1}{2} \leq h$. The maximum possible height for n nodes is $n - 1$. Therefore, $log_2 \frac{n+1}{2} \leq h \leq n - 1$.

Leaves of binary tree

Theorem 5.1

For a binary tree, $|leaves| \leq 1 + |internal\ nodes|$*.*

Proof.

We will prove the theorem by induction over the structure of a tree (Recall the recursive definition of a tree).

Base case:

We have a single node.
$|leaves| = 1$ and $|\text{internal nodes}| = 0$. Case holds. ...

Commentary: $|A|$ indicates the size of set A.

Leaves of binary tree(2)

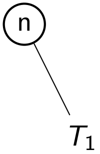
Proof(continued).

Induction step:

We have two cases in the induction step: Root has one child or two children.

Case 1:

Let tree T be constructed as follows.



For T_1, let $|leaves| = \ell_1$ and $|\text{internal nodes}| = i_1$.
T has ℓ_1 leaves and $i_1 + 1$ internal nodes.
By the induction hypothesis, $\ell_1 \leq 1 + i_1$.
Therefore, $\ell_1 \leq 1 + i_1 + 1$.
Therefore, $\ell_1 \leq 1 + (i_1 + 1)$. Case holds.

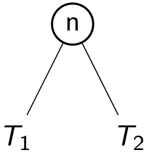
...

Leaves of binary tree(3)

Proof(continued).

Case 2:

Let tree T be constructed as follows.



For T_1, let $|leaves| = \ell_1$ and $|\text{internal nodes}| = i_1$.
For T_2, let $|leaves| = \ell_2$ and $|\text{internal nodes}| = i_2$.
T has $\ell_1 + \ell_2$ leaves and $i_1 + i_2 + 1$ internal nodes.
By induction hypothesis, $\ell_1 \leq 1 + i_1$ and $\ell_2 \leq 1 + i_2$.
Therefore, we have $\ell_1 + \ell_2 \leq 2 + i_1 + i_2$.
Therefore, $\ell_1 + \ell_2 \leq 1 + (i_1 + i_2 + 1)$. Case holds.

□

Exercise 5.7

Prove/Disprove: If no node in the binary tree has a single child, $|leaves| = 1 + |internal\ nodes|$. (Quiz 2023)

Maximum and minimum number of leaves

Let n be the number of nodes in a binary tree T.

Due to the previous theorem, we know $|\text{leaves}| \leq 1 + |\text{internal nodes}|$.

Since $|\text{leaves}| + |\text{internal nodes}| = n$, $|\text{leaves}| \leq 1 + n - |\text{leaves}|$.

$$|\text{leaves}| \leq \frac{(n+1)}{2}.$$

Exercise 5.8

a. When do $|leaves|$ meet the inequality?

b. When is the number of leaves minimum?

Commentary: If T is complete, the number of leaves is $\frac{(n+1)}{2}$.

Topic 5.3

Representing Tree

Container for tree

There is no C++ container for the tree.

Trees are the backbone of many abstract data structures.

For some reason, it is not explicitly there.

Exercise 5.9

Why is there no tree container in C++ STL? (Let us ask ChatGPT)

Commentary: I guess that we rarely explicitly need trees in our programming. We usually have higher goals such as stack,queue, set, and map, which may need a tree as an internal data structure, but users need not be exposed. However, there are applications where there is a clear need for trees. For example, the representation of arithmetic expressions. In my programming, whenever I needed a tree. I have implemented it myself.

Representation of a binary tree on a computer

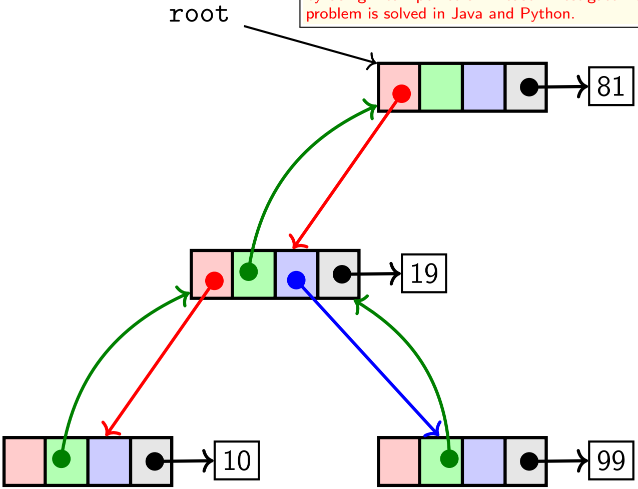
Commentary: parent pointer causes cycle of pointers, which makes the garbage collection difficult in the languages like Java. In C++, we may break the cycle by using weak pointers. Please investigate how the problem is solved in Java and Python.

Definition 5.15

A binary tree consists of nodes containing four pointer fields.

- *left child*
- *parent*
- *right child*
- *label*

An additional root pointer points to the root of the tree.

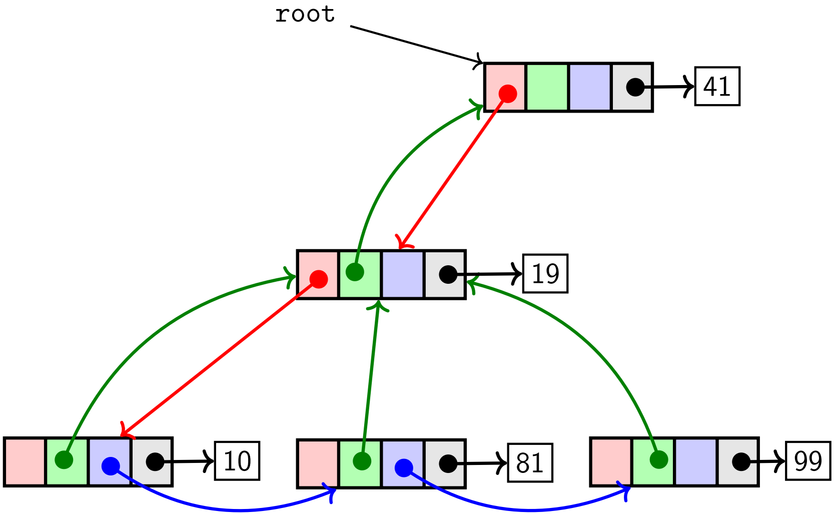


The pointers that are not pointing anywhere are NULL.

Exercise 5.10

Do we need the parent pointer?

Representation of a tree on a computer



Definition 5.16

A tree consists of nodes containing four pointer fields.

- *first child*
- *parent*
- *next sibling*
- *label*

An additional root pointer points to the root of the tree.

Exercise 5.11

Are we representing an ordered tree or an unordered tree?

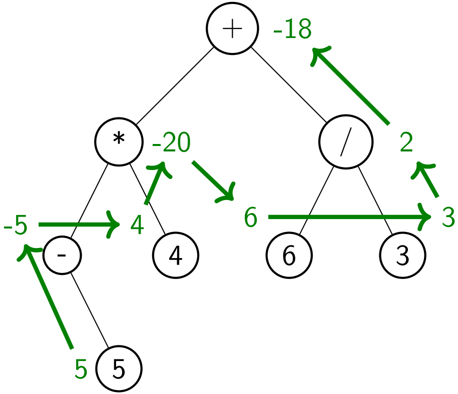
Topic 5.4

Tree walks

Application : Evaluating an expression

Example 5.12

If we want to evaluate an expression represented as a binary tree, we need to visit each node and evaluate the expression in a certain order.



In green, we have evaluated the value of the node. The path indicates the order of evaluation.

Tree walks

Visiting nodes of a tree in a certain order are called tree walks.

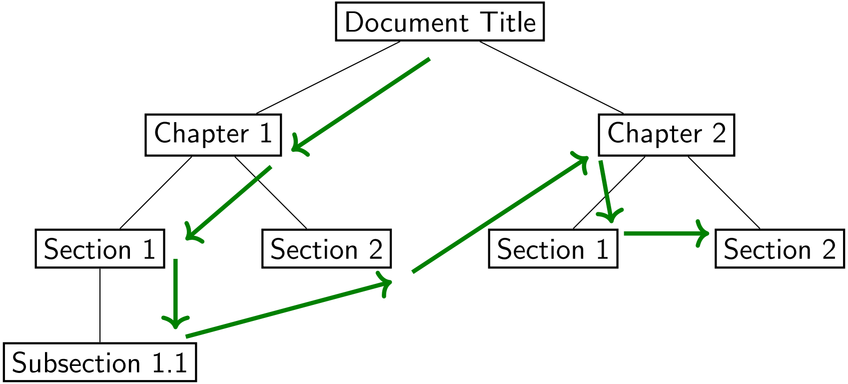
There are two kinds of walks for trees.

- preorder: visit parent first
- postorder: visit children first

Example: preorder

Example 5.13

Let a document be stored as a tree. We read the document in preorder.



Preorder/Postorder walk

Algorithm 5.1: PreOrderWalk(n)

```
visit(n);
for n' ∈ children(n) do
    PreOrderWalk(n');
```

Algorithm 5.2: PostOrderWalk(n)

```
for n' ∈ children(n) do
    PostOrderWalk(n');
visit(n);
```

The first example of expression evaluation is postorder walk.

Commentary: visit(v) is some action taken during the walk.

Walking on ordered tree

How do we walk on an ordered tree?

For an ordered tree, we may visit children in the given order among siblings.

We may have choices to change the order of visits among ordered siblings.

Commentary: Our algorithm works for both ordered and unordered trees. Our algorithm does not specify the order of visits of siblings for unordered trees. Please pay attention to the subtle differences among trees, ordered trees, and binary trees.

Topic 5.5

Walking binary trees

Preorder/Postorder walk over binary trees

We have more structure in binary trees. Let us write the algorithm for walks again.

Algorithm 5.3: PreOrderWalk(n)

```
if n == Null then
    return
visit(n);
PreOrderWalk(left(n));
PreOrderWalk(right(n));
```

Algorithm 5.4: PostOrderWalk(n)

```
if n == Null then
    return
PostOrderWalk(left(n));
PostOrderWalk(right(n));
visit(n);
```

Exercise 5.12

Are the above programs tail-recursive?

Inorder walk of binary trees

Definition 5.17

In an inorder walk of a binary tree, we visit the node after visiting the left subtree and before visiting the right subtree.

Algorithm 5.5: InOrderWalk(n)

```
if n == Null then
    return
InOrderWalk(left(n));
visit(n);
InOrderWalk(right(n));
```

Exercise 5.13

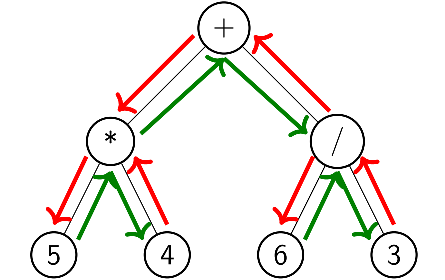
Given complete binary trees with 7 nodes, label the nodes such that the preorder, inorder, or postorder walks produce the sequence 1,2,...,7.

Application : Printing an expression

To print an expression (without unary minus), we need to visit the nodes in inorder.

Algorithm 5.6: PrintExpression(n)

1 **if** *n is leaf* **then**
2   print(label(n));
3   **return**
4 print("(");
5 PrintExpression(left(n));
6 print(label(n));
7 PrintExpression(right(n));
8 print(")");



((5 * 4) + (6 / 3))

Exercise 5.14

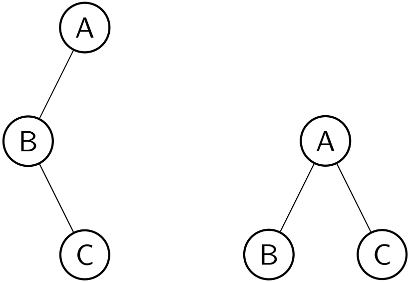
a. Modify the above algorithm to support unary minus.

*b. What will happen if "**if**" at line 1 is replaced by "**if** $n == NULL$ **then return**"?*

Commentary: The order of the walk is the pattern of recursive calls and actions on nodes. An application may need a mixed action pattern. In the above printing example, we need to print parentheses before and after making recursive calls. The parentheses are printed pre/post-order. All three walks are present in the above algorithm.

Many trees have the same walks

The following two ordered trees have the same preorder walks.



Commentary: Answer:

For postorder:

```
               A
              /
     A       C
    / \       \
   B   C       B
```

For inorder:

```
     A       B
    / \     / \
   B   C   D   A
  / \         / \
 D   E       E   C
```

For postorder and preorder:

```
   A   A
  /     \
 C       C
  \     /
   B   B
```

Exercise 5.15

a. Give two binary trees that have the same postorder walks.

b. Give two binary trees that have the same inorder walks.

c. Give two binary trees that have the same postorder and preorder walks.

Topic 5.6

Tutorial problems

Exercise: paths in a tree

Exercise 5.16

Given a tree with a maximum number of children as k. We give a label between 0 and k-1 to each node with the following simple rules. (i) the root is labeled 0. (ii) For any vertex v, suppose that it has r children, then arbitrarily label the children as 0,..,r-1. This completes the labeling. For such a labeled tree T, and a vertex v, let seq(v) be the labels of the vertices of the path from the root to v. Let $Seq(T) = \{seq(w)|winT\}$ be the set of label sequences. What properties does Seq(T) have? If a word w appears what words are guaranteed to appear in Seq(T)? How many times does a word w appear as a prefix of some words in Seq(T)?

Commentary: From Milind's notes!

Lowest common ancestor(LCA)

Definition 5.18

For two nodes n_1 and n_2 in a tree T, $LCA(n_1, n_2, T)$ is a node in $ancestors(n_1) \cap ancestors(n_2)$ that has the largest level.

Exercise 5.17

Write a function that returns $lca(v, w, T)$. What is the time complexity of the program?

Exercise: paths in a tree

Exercise 5.18

Given $n \in T$, Let $f(n)$ be a vector, where $f(n)[i]$ is the number of nodes at depth i from n.

- *Give a recursive equation for $f(n)$.*
- *Give a pseudo code to compute the vector $f(root(T))$. How is the time complexity of the program?*

The uniqueness of walks if two walks are the same.

Exercise 5.19

Give an algorithm for reconstructing a binary tree if we have the preorder and inorder walks.

Exercise 5.20

Let us suppose all internal nodes of a binary tree have two children. Give an algorithm for reconstructing the binary tree if we have the preorder and postorder walks.

Topic 5.7

Problems

Exercise: mean level

Exercise 5.21

a. Suppose that you are given a binary tree, where, for any node v, the number of children is no more than 2. We want to compute the mean of $ht(v)$, i.e., the mean level of nodes in T. Write a program to compute the mean level.

b. Suppose that we are given the level of all leaves in the tree. Can we compute the mean height? Given a sequence $(n_1,n_2,...,n_k)$ of the levels of k leaves, is there a binary tree with exactly k leaves at the given levels?

Reconstructing tree from preorder walks

Exercise 5.22

Let us suppose we can calculate the number of children of a node by looking at the label of a node of a binary tree, e.g., arithmetic expressions. Give an algorithm for reconstructing the binary tree if we have the preorder walk.

Exercise: previous print

Exercise 5.23

For a given binary tree, let $prevPrint(T, a)$ *give the node* n' *such that* $label(n')$ *will appear just before* $label(n)$ *in the inorder printing of* T*. Give a program that implements prevPrint.*

Exercise: level-order walk

Exercise 5.24

Give an algorithm for walking a tree such that nodes are visited in the order of their level. Two nodes at the same level can visit in any order.

Exercise 5.25

Give an algorithm for walking a tree such that nodes are visited in the order of their height.

Exercise: balanced trees

Definition 5.19

A binary tree T is called balanced if for each node $n \in T$

$$|height(right(n)) - height(left(n))| \leq 1.$$

Exercise 5.26

Prove/Disprove: if no node in the binary tree has a single child, the binary tree is balanced.

End of Lecture 5