04 B: Lists, Stacks, and Queues IV; Trees I

CS1102S: Data Structures and Algorithms

Martin Henz

February 5, 2010

Generated on Thursday 4th February, 2010, 23:01

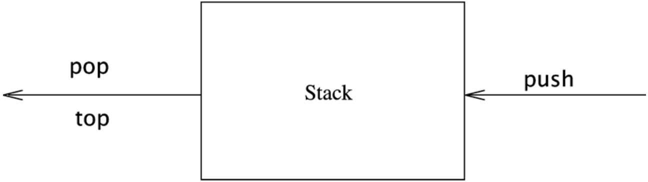
1. Review: The Stack ADT
2. The Queue ADT
3. Trees
4. Puzzlers

1. Review: The Stack ADT
   - Stack Model
   - Implementation of Stacks
2. The Queue ADT
3. Trees
4. Puzzlers

Motivation

Purpose of stacks

Collections that serve as intermediate storage of data items



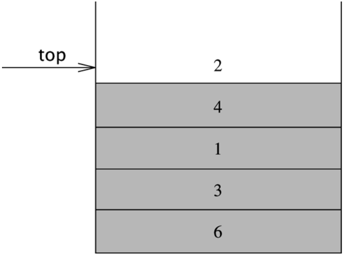
Stack Model

Stack access

Only the top element of a stack is accessible through top and pop operations

Stack discipline

Last in—first out: LIFO



Implementation of Stacks

- Possible based on either ArrayList or LinkedList
- Often Lists are used directly, for example by using a List and always using the index 0

1. Review: The Stack ADT

2. The Queue ADT
   - Motivation
   - Motivation
   - Implementation of Queues

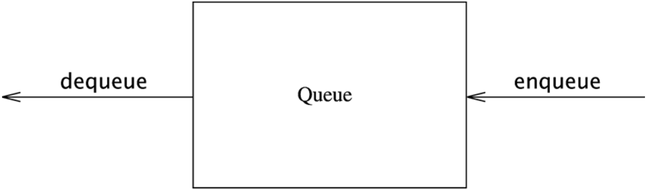
3. Trees

4. Puzzlers

Motivation

Purpose of queues

Collections that serve as intermediate storage of data items



Queue Model

Stack discipline

First in—first out: FIFO

Implementation of Queues using LinkedList

```
class LinkedListQueue<E> extends LinkedList<E> {
  public boolean empty() {
    return size() == 0; }
  public void enqueue(E item) {
    add(item,0); return item; }
  public E dequeue() {
    if (empty())
      throw new EmptyQueueException();
    else return remove(size()-1); } }
```

Implementation of Queues using LinkedList

```
class LinkedListQueue<E> extends LinkedList<E> {
  public boolean empty() {
    return size() == 0; }
  public void enqueue(E item) {
    add(item,0); return item; }
  public E dequeue() {
    if (empty())
      throw new EmptyQueueException();
    else return remove(size()-1); } }
```

Why does dequeue() run in $O(1)$?

Implementation of Queues using LinkedList

```
class LinkedListQueue<E> extends LinkedList<E> {
  public boolean empty() {
    return size() == 0; }
  public void enqueue(E item) {
    add(item,0); return item; }
  public E dequeue() {
    if (empty())
      throw new EmptyQueueException();
    else return remove(size()-1); } }
```

Why does dequeue() run in $O(1)$?

See
API Specification.

Implementation of Queues using Arrays

General idea

Keep items in array similar to ArrayList

Access

Keep a marker for adding items back and for removing items front

Optimization

Wrap back and front around when end of array is reached

1. Review: The Stack ADT
2. The Queue ADT
3. Trees
   - Preliminaries
   - Binary Trees
4. Puzzlers

Motivation

Trees in computer science

Trees are ubiquitous in CS, covering operating systems, computer graphics, data bases, etc.

Trees as data structures

Provide $O(\log N)$ search operations

Heaps

Serve as basis for other efficient data structures, such as heaps

Trees in Java API

Covered by API classes TreeSet and TreeMap

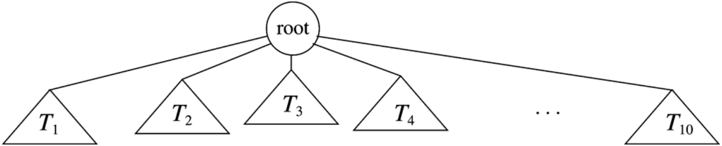
Definitions

Tree

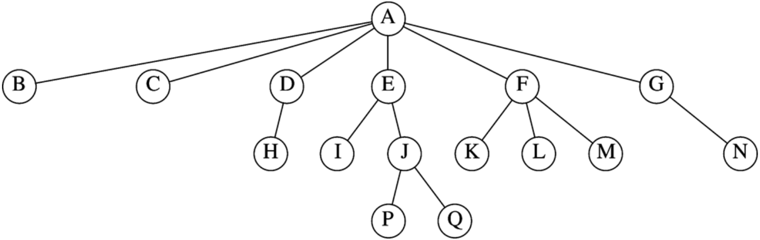
A *tree* is a collection of *nodes*. Non-empty trees have a distinguished node r, called *root*, and zero or more nonempty (sub)trees $T_1, T_2, \ldots, T_k$, each of whose roots are connected by a directed *edge* from r.

Parent and child

The root of each subtree is called a *child* of r, and r is the *parent* of each subtree root.



Example and More Definitions



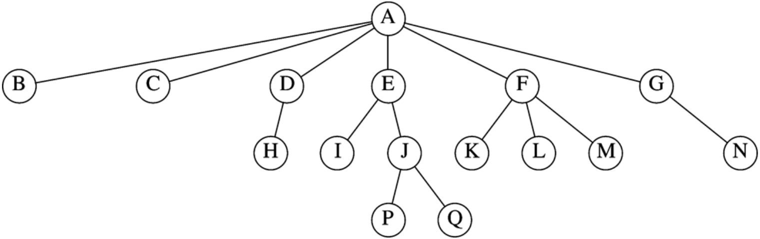
Leaf

Nodes with no children are called *leaves*.

Sibling

Nodes the same parents are called *siblings*.

Example and More Definitions



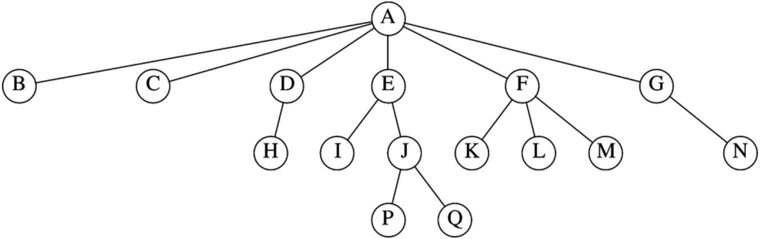
Path

A *path* from node n_1 to n_k id defined as a sequence of nodes $n_1, n_2, \ldots, n_k$ such that n_i is the parent of n_{i+1} for $1 \leq i < k$.

Length of Path

The *length* of a path is the number of edges on the path, namely $k - 1$.

Example and More Definitions



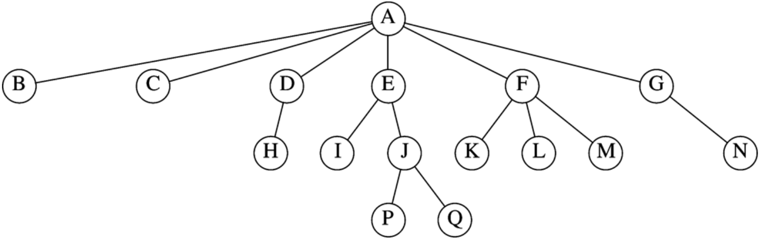
Paths of length 0

There is a path of length 0 from every node to itself.

Number of paths

There is exactly one path from the root to each node.

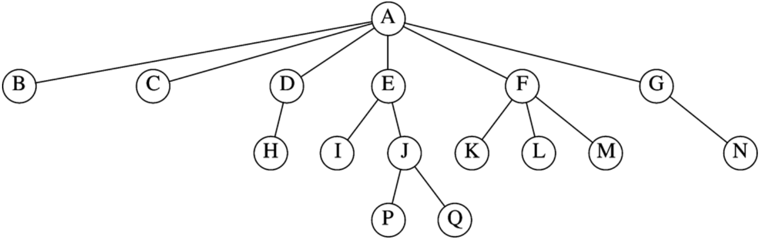
Example and More Definitions



Depth

The *depth* of node n_i is the length of the unique path from the root to n_i.

Example and More Definitions



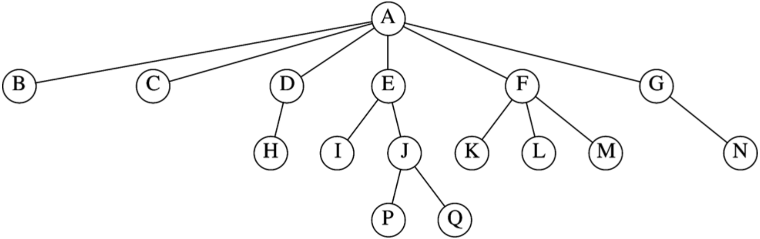
Height

The *height* of n_i is the length of the longest path from n_i to a leaf.

Height of a tree

The height of a tree is equal to the height of the root.

Example and More Definitions



Ancestor and descendant

If there is a path from n_1 to n_2, then n_1 is an *ancestor* of n_2, and n_2 is a *descendant* of n_1.

Proper Ancestor and proper descendant

If $n_1 \neq n_2$, and n_1 is an ancestor of n_2, then n_1 is a *proper ancestor* of n_2 and n_2 is a *proper descendant* of n_1.

Implementation

First idea

In each node, keep its data, and a reference to each child

Problem

We don’t know how many children a node may have (can also change, later)

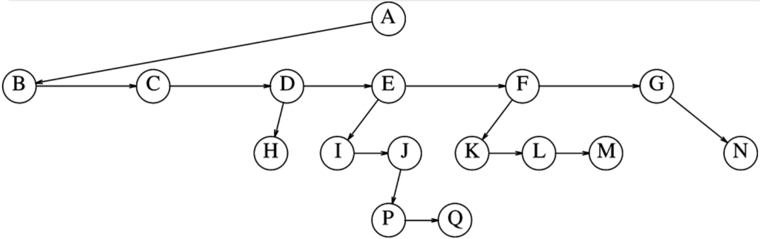
Solution

Keep children of each node in a linked list of tree nodes

Implementation

Node data type

```
class TreeNode<Any> {
    Any element;
    TreeNode<Any> firstChild;
    TreeNode<Any> nextSibling;
}
```



Tree Traversal

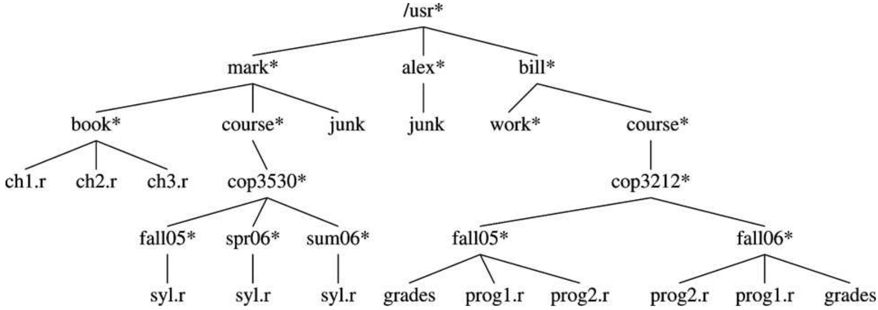
Common use of trees

File and folder structure in Windows and Unix: folder are nodes, ordinary files are leaf nodes

Common tasks involving files and folders

- List all files in a folder (and its subfolders)
- Compute the size of a folder (including all subfolders)

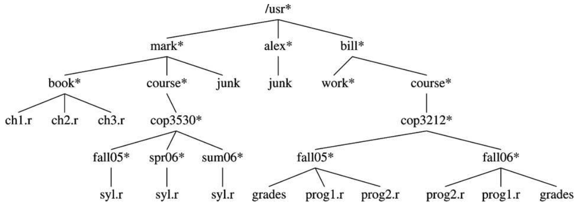
Example


/usr*
mark*
alex*
bill*
book*
course*
junk
junk
work*
course*
ch1.r
ch2.r
ch3.r
cop3530*
cop3212*
fall05*
spr06*
sum06*
fall05*
fall06*
syl.r
syl.r
syl.r
grades
prog1.r
prog2.r
prog2.r
prog1.r
grades

Algorithm for File Listing

```
private void listAll( int depth ) {
  printName( depth ); // print name of object
    if( isDirectory( ) )
      for each file c in this directory
        c.listAll( depth + 1 );
}
public void listAll( ) {
  listAll( 0 );
}
```

Example




```
/usr
    mark
        book
            ch1.r
            ch2.r
            ch3.r
        course
            cop3530
...
```

Reflection

What is going on?

Work (print file name) is done at each node *before* the children of the node are visited

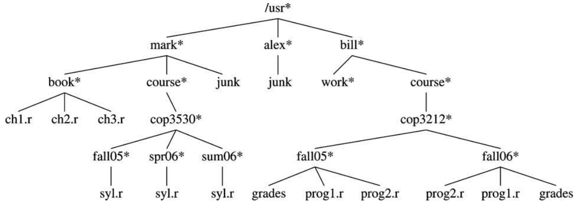
Tree traversal

If the work at each node is done *before* the children are visited, we talk about *preorder traversal*

Algorithm for File Size Calculation

```
public int size( ) {
  int totalSize = sizeOfThisFile( );
  if( isDirectory( ) )
    for each file c in this directory
      totalSize += c.size( );
  print(totalSize); // print size of object
  return totalSize;
}
```

Example




```
            ch1.r                3
            ch2.r                2
            ch3.r                4
        book                    10
        ...
    mark                        30
    ...
/usr                            72
```

Reflection

What is going on?

Work (print file size) is done at each node *after* the children of the node are visited

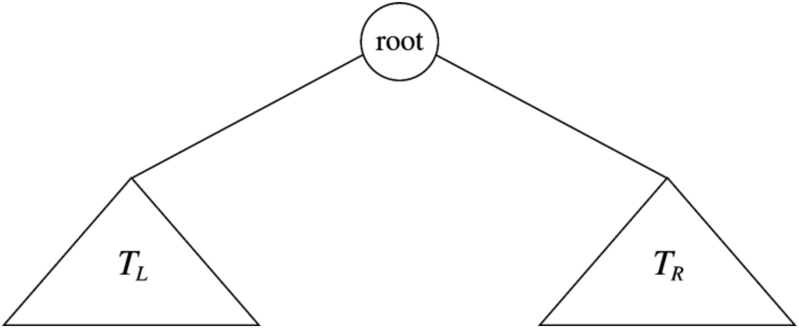
Tree traversal

If the work at each node is done *after* the children are visited, we talk about *postorder traversal*

Binary Trees

Definition

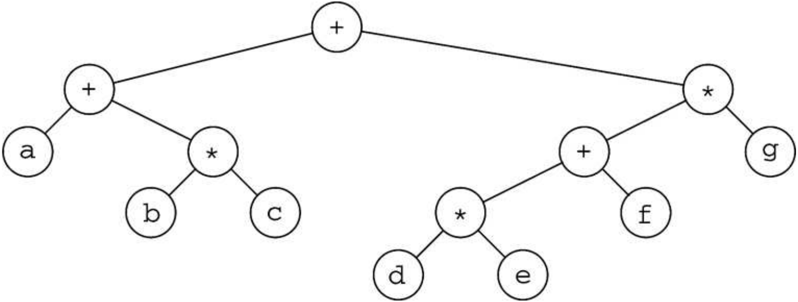
A binary tree is a tree in which no node can have more than two children.



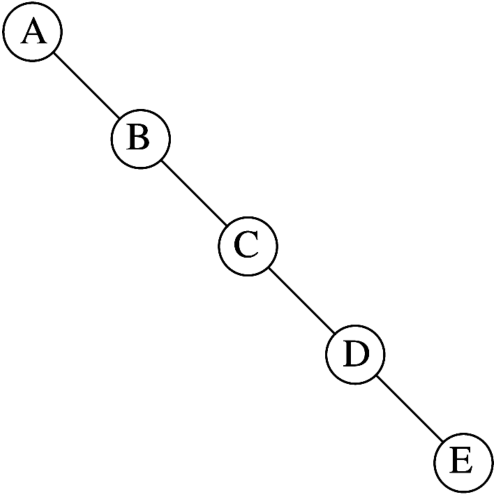
Implementation

```
class BinaryNode {
  // accessible by other package routines
  Object      element;       // The data in the node
  BinaryNode  left;          // Left child
  BinaryNode  right;         // Right child
}
```

Example: Expression Trees



Example: Degenerate Binary Tree



1. Review: The Stack ADT
2. The Queue ADT
3. Trees
4. Puzzlers
   - Solution Puzzler “Animal Farm”
   - New Puzzler: “Generic Drugs”

Puzzler: Animal Farm

```
public class AnimalFarm {
  public static void main(String[] args) {
    final String pig = "length: 10";
    final String dog = "length: " + pig.length();
    System.out.println("Animals are equal: "
                       + pig == dog);
  }
}
```

Solution

Operator + has higher precedence than ==.
Thus

```
System.out.println("Animals are equal: "
                   + pig == dog);
```

means

```
System.out.println(
    ("Animals are equal: " + pig) == dog
);
```

A Quick Fix

```
System.out.println("Animals are equal: "
                   + (pig == dog));
```

What will be printed?

What does == mean?

Primitive Data Types

For primitive data types, == implements literal equality. It tests whether the values are identical (to the bit).

References

For object references, == checks whether the references refer to the same object.

Diagnostics

The two `String` references do not refer to the same object. They only contain the same characters.

New Puzzler: Generic Drugs

```
public class LinkedList<E> {
  private Node<E> head = null;
  private class Node<E> {
    E value;
      Node<E> next;
      // constructor links the node as new head
      Node(E value) {
        this.value = value;
        this.next = head;
        head = this;
      }
  }
```

New Puzzler: Generic Drugs

```
  public void add(E e) {
    new Node<E>(e); // Link node as new head
  }
  public void dump() {
    for (Node<E> n = head; n != null; n = n.next)
      System.out.print(n.value + " ");
  }
  public static void main(String[] args) {
    LinkedList<String> list
    = new LinkedList<String>();
    list.add("world");
    list.add("Hello");
    list.dump();
} }
```

Next Week

- Monday:
  - Lab: Lab tasks on lists, queues, stacks
  - Assignment 3 due
- Wednesday: Lecture on Binary Trees
- Thursday: Tutorial on Assignment 3
- Friday: Midterm 1 on first 100 pages