Data Structures and Algorithms

Session 7. February 11, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Added office hour: Nikhil Friday 10 AM-12 PM
- Homework 2 is up. Due Feb. 23
- Late Policy starting homework 2:
    - 10% for each *unexcused* day late
    - up to maximum 3 days; zero credit after 3 days
    - Contact TA's when you submit late

Review

- (Header Nodes for Linked Lists)
- Stack Implementation recap
- Queues:
  - Circular Array

Today’s Plan

- Lists, Stacks, Queues in Linux
- Introduction to Trees
  - Definitions
  - Tree Traversal Algorithms
- Binary Trees

Lists, Stacks, Queues in Linux

- Linux:
  - processes stored in Linked List
  - FIFO scheduler schedules jobs using queue
  - function calls push memory onto stack

Drawbacks of Lists

- So far, the ADT's we've examined have been linear
- O(N) for simple operations
- Can we do better?
  - Recall binary search: log N for find :-)
  - But list must be sorted. N log N to sort :-(

Trees

- Extension of Linked List structure:
  - Each node connects to multiple nodes
- Examples include file systems, Java class hierarchies

Tree Terminology

- Just like Lists, **Trees** are collections of **nodes**
- Conceptualize trees upside down (like family trees)
  - the top node is the **root**
  - nodes are connected by **edges**
  - edges define **parent** and **child** nodes
  - nodes with no children are called **leaves**

More Tree Terminology

- Nodes that share the same parent are **siblings**
- A **path** is a sequence of nodes such that the next node in the sequence is a child of the previous
- a node’s **depth** is the length of the path from root
- the **height** of a tree is the maximum depth
- if a path exists between two nodes, one is an **ancestor** and the other is a **descendant**

Tree Implementation

- Each node is part of a Linked List of siblings
- Additionally, each node stores a reference to its children
-
```
public class TreeNode {
    Object   element;
    TreeNode firstChild;
    TreeNode nextSibling;
}
```

Tree Traversals

- Suppose we want to print all the nodes in a tree
- What order should we visit the nodes?
    - **Preorder** - read the parent before its children
    - **Postorder** - read the parent after its children

Preorder vs. Postorder

-
  ```
  preorder(node x)
      print(x)
      for child : Children
          preorder(child)
  ```

-
  ```
  postorder(node x)
      for child : Children
          postorder(child)
      print(x)
  ```

Binary Trees

- Nodes can only have two children:
  - left child and right child
- Simplifies implementation and logic

-
  ```
  public class BinaryNode {
      Object element;
      BinaryNode left;
      BinaryNode right;
  }
  ```
- Provides new **inorder** traversal

Inorder Traversal

- Read left child, then parent, then right child
- Essentially scans *whole* tree from left to right
-
  ```
  inorder(node x)
      inorder(x.left)
      print(x)
      inorder(x.right)
  ```

Binary Tree Properties

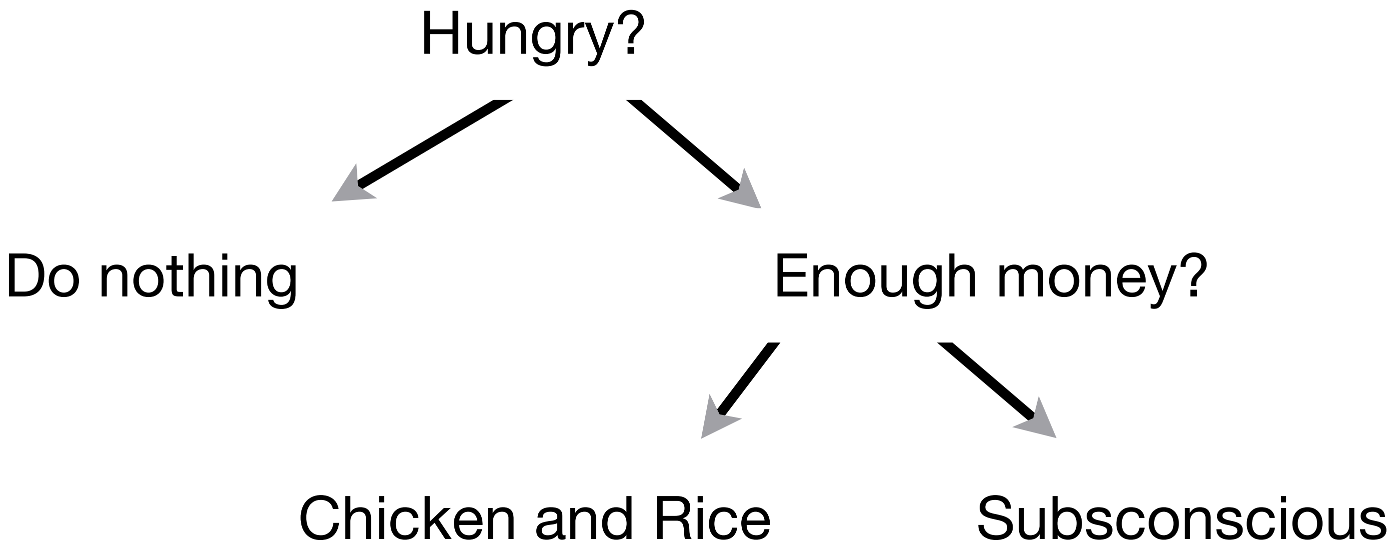
- A binary tree is **full** if each node has 2 or 0 children
- A binary tree is **perfect** if it is full and each leaf is at the same depth
  - That depth is O(log N)

Expression Trees

- Expression Trees are yet another way to store mathematical expressions
    - ((x + y) * z)/300
- Note that the main mathematical operators have 2 operands each
- Inorder traversal reads back infix notation
- Postorder traversal reads postfix notation

Decision Trees

- It is often useful to design decision trees
- Left/right child represents yes/no answers to questions



Hungry?

Do nothing

Enough money?

Chicken and Rice

Subsconscious

Reading

- Weiss Section 4.3: Binary Search Trees