



2007-03-13
Lecture #11
(Material on Test 2)

Today's Outline
Knapsack Problem
Jhave
Trees

Recap last class, important items

Merge Sort?

- **Given an unsorted list ... draw how merge sort would work**
- **What would the recursive c++ code look like?**
- **What is the Big O complexity of Merge Sort? $O(n \log n)$**
- **How much memory does Merge Sort use compared to Quick Sort?**

Quick Sort?

- **Given an unsorted list ... draw how quick sort would work**
- **What would the recursive c++ code look like?**
- **What is the Big O complexity of Quick Sort? $O(n \log n)$**
- **What role does the pivot index play in the Big O complexity?**
- **How much memory does Quick Sort use compared to Merge Sort?**

Recap from last class, important items...

Towers of Hanoi

- **What are the basic rules for Towers of Hanoi?**
- **What is the Big O complexity of Towers of Hanoi? $O(k^n)$**

Recursion

- **What are the pros & cons of recursive vs iterative?**
- **What is the stackframe of a recursive function?**
- **What is tail recursion?**
- **What is linear recursion?**
- **What is Divide & Conquer?**
- **Be able to convert simple tail or linear recursive functions to iterative**
- **Be able to determine the output of simple recursive functions**

Class Recursion

- **Fibonacci Numbers, Factorial, Towers of Hanoi**

0-1 Knapsack Problem?

Each item x_j has a value p_j and weight w_j
The maximum weight we can carry in a bag is C

Goal ...

maximize
$$\sum_{j=1}^n p_j x_j.$$

subject to
$$\sum_{j=1}^n w_j x_j \leq c, \quad x_j = 0 \text{ or } 1, \quad j = 1, \dots, n.$$

Examples:

Camping Backpack – you can only carry so many pounds of supplies, and you want to maximize the value of your supplies

0-1 Knapsack Problem?

Any for of the Knapsack problem is $O(k^n)$...

Why? Because it requires an exhaustive search, must check EVERY solution in order to guarantee you have the best one ...

Example:

6 items available ... max in bag is 100 lbs

a = 50lb, \$50 b = 10lb, \$10 c = 5lb, \$2

d = 75lb, \$60 e = 60lb, \$30 f = 100lb, \$50

Some of the Possible solutions:

a = 50lb, \$50 b, c, e = 75lb, \$42

a, b, c = 65lbs, \$62 d = 75lb, \$60

d, b, c = 90lb, \$75

Harder Variations of the Knapsack problem

Bounded Knapsack Problem

- Instead of 0 or 1 of each item ... you have 0 to M of each you can take for example, limit of 1 A, 3 Bs, 4c's, 2d's, 1e

Example:

2 of A = 100lb, \$100

1 d, 2 of B, 1 c = 100lb, \$82

UnBounded Knapsack Problem

- Instead of 0 to M of each item, there is no limit ...

Example:

20 of C = 100lb, \$40

1 d, 5 of C = 100lb, \$70

How to solve the Knapsack Problem?

1.) Greedy by Profit

Do we try to simply maximize the Profit?
Thus, grab the most valuable items first?

2.) Greedy by Weight

Do we try to simply minimize the Weight?
Thus, grab the lightest items first?

3.) Greedy by Profit Density

Do we try to maximize \$ per pound?

No matter which approach we choose ... we can't guarantee the optimal solution without exhaustively checking every item.

Decision Problem

a decision problem is a question in some formal system with a yes-or-no answer

Example:

the problem "given two numbers x and y , does x evenly divide y ?" is a decision problem. The answer can be either 'yes' or 'no', and depends upon the values of x and y .

Function Problem

**a problem that requires a more
complex answer than just yes-or-no**

Example:

"given two numbers x and y , what is x divided by y ?"

Function Problems vs Decision Problems

*****Important*****

For all function problems there is an analogous decision problem

Example 1:

Knapsack problem = function problem

Given a Knapsack filled with possible items, is the weight $< C$?

Example 2:

TSP = function problem

Given a set of cities / driving directions, is the distance he would travel $< M$ miles

P vs NP

P is the set of decision problems that can be solved in Polynomial time or less (looking at Big O, this is any of them but $O(k^n)$)

NP is the set of decision problems that cannot be solved in Polynomial time or less (looking at Big O, this is any that are $O(k^n)$)

P problem example ...

Given an array of integers, is the sum equal to 50?

NP problem example ...

Given an array of integers, does some subset's sum equal 50?

What if there were 10 million integers, and only 1 subset equals 50?

Does $P = NP$?

This questions basically asks ...

Can the decision problems currently classified as non-polynomial time algorithm actually be solved in polynomial time?

Can we find an algorithm that guarantees that for the example below ... we can answer yes / no with 100% certainty in $O(n^k)$ or less?

NP problem example ...

Given an array of integers, does some subset's sum equal 50?

*****Important*** It has been proven that if we can find just 1 polynomial time solution to ANY NP problem ... then ALL NP problems are solvable in polynomial time, and therefore $P=NP$**

JHave

Dr. Naps of UW-Oshkosh has created an algorithm animation tool

Several classic algorithms are available for you to view and work with

<http://www.jhave.org>

Including ...

- MergeSort**
- QuickSort**

In order to better understand these sorting algorithms, it is strongly recommended you work with JHave as practice.

Trees

Chapter 7 in your book

**There are many variations of the Tree data structure
but first we want to just discuss trees in general and
what is common to pretty much all tree data structures**

Variations include: Binary Trees, B Trees, Red Black Trees, Heaps, ...

Keywords associated with Trees

Node

- **Contains Data**
- **Contains Pointers to the Next Nodes (similar to linked list node)**

Branch

- **Directed Line (Pointer) goes in only 1 direction (similar to linked list)**

Tree

- **Finite set of Elements (Data) contained with Nodes and a Finite Set of Directed Lines (Pointers) called Branches.**

Keywords associated with Trees

Degree

- **Numbers of Branches associated with a Node**

InDegree

- **Number of Branches coming into that Node**

OutDegree

- **Number of Branches going out of that Node**

Root

- **The top node in the tree, there is only 1 root in a tree**
- **The only node with an InDegree of 0**

Leaf

- **The bottom nodes in the tree**
- **The only nodes with an OutDegree of 0**

Internal – Any non-root, non-leaf node

Keywords associated with Trees

Child Node

- **A node that has a predecessor node**

Parent Node

- **A node that has successor nodes**
- **Each child has only 1 parent, but parent can have many children**

Siblings

- **2 or more Child Nodes with the same parent**

Ancestor

- **Any node in the path from the root to the node**

Desendant

- **Any node in the path from the node to a any leaf**

Tree keywords

Path

- Sequence of nodes where each is adjacent to the next
- Note: Every node can be reached with a path from the root

Level

- Distance from the root
- root is at level 0, cause it's no distance from itself

Height

- level of the leaf with longest path from root ... + 1

Depth

- since trees are drawn upside down (starting from root)
many books and reference call Height the Depth

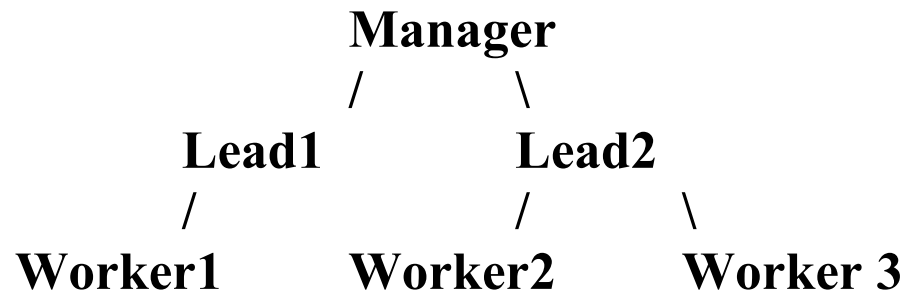
Tree Keywords

Subtree

- any connected structure below the root

Tree Representation

1.) General Tree



Tree Representation

2.) Indented List

Manager

Lead1

Worker1

Lead2

Worker 2

Worker 3

3.) Parenthetical Notation

Manager (Lead1 (Worker1) Lead 2 (Worker 2, Worker 3))

Tree Example

We should be able to draw a tree and point out

- **the root**
- **all nodes that are parents**
- **all nodes that are children**
- **which nodes are siblings of each other?**
- **What is the degree, indegree, and outdegree of each node?**
- **Which nodes are leaves?**
- **Which nodes are internal?**
- **height / depth of tree**
- **level of a node?**
- **Parents, children, ancestors, descendants?**
- **Path from root to node?**

Types of Trees

General Tree

- **Each node can have any number of subtrees**

Binary Tree

- **Each node can have NO MORE than 2 sub trees**
- **Thus it could have 0, 1, or 2 subtrees**
- **Left Subtree**
- **Right Subtree**

Binary Tree

C++ node class (just like a singly linked list)

```
struct treenode  
{  
    type data ;  
    treenode * left ;  
    treenode * right ;  
};
```

C++ Recap ... what's the different here? Pros & cons?

```
struct treenode2  
{  
    type * data ; // for pros & cons, think about empty node  
    treenode * left ;  
    treenode * right ;  
};
```

Generalcs271 – Data Structures
Justin Miller – Univ. of Wisconsin Oshkosh
10:20am to 12:20pm Tu,Th – Harrington 119