

CSC 421: Algorithm Design Analysis

Spring 2016

Transform & conquer

- transform-and-conquer approach
- presorting
- balanced search trees, heaps
- Horner's Rule
- problem reduction

1

Transform & conquer

the idea behind transform-and-conquer is to transform the given problem into a slightly different problem that suffices

e.g., presorting data in a list can simplify many algorithms

- suppose we want to determine if a list contains any duplicates

BRUTE FORCE: compare each item with every other item

$$(N-1) + (N-2) + \dots + 1 = (N-1)N/2 \rightarrow O(N^2)$$

TRANSFORM & CONQUER: first sort the list, then make a single pass through the list and check for adjacent duplicates

$$O(N \log N) + O(N) \rightarrow O(N \log N)$$

- finding the mode of a list, finding closest points, ...

2

Balanced search trees

recall binary search trees – we need to keep the tree balanced to ensure $O(N \log N)$ search/add/remove

- OR DO WE?
- it suffices to ensure $O(\log N)$ height, not necessarily minimal height

transform the problem of "tree balance" to "relative tree balance"

several specialized structures/algorithms exist:

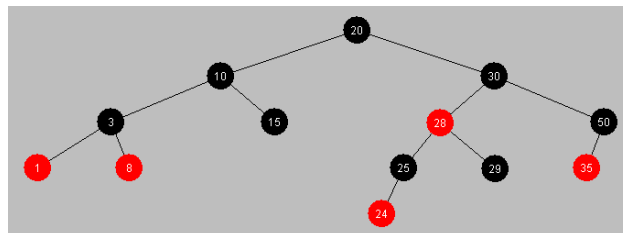
- AVL trees
- 2-3 trees
- red-black trees

3

Red-black trees

a red-black tree is a binary search tree in which each node is assigned a color (either red or black) such that

1. the root is black
 2. a red node never has a red child
 3. every path from root to leaf has the same number of black nodes
- add & remove preserve these properties (complex, but still $O(\log N)$)
 - red-black properties ensure that tree height $< 2 \log(N+1) \rightarrow O(\log N)$ search



4

TreeSets & TreeMaps

`java.util.TreeSet` uses *red-black trees* to store values

→ $O(\log N)$ efficiency on add, remove, contains

`java.util.TreeMap` uses *red-black trees* to store the key-value pairs

→ $O(\log N)$ efficiency on put, get, containsKey

thus, the original goal of an efficient tree structure is met

- even though the subgoal of balancing a tree was transformed into "relatively balancing" a tree

5

Scheduling & priority queues

many real-world applications involve optimal scheduling

- balancing transmission of multiple signals over limited bandwidth
- selecting a job from a printer queue
- selecting the next disk sector to access from among a backlog
- multiprogramming/multitasking

a *priority queue* encapsulates these three optimal scheduling operations:

- ✓ *add item (with a given priority)*
- ✓ *find highest priority item*
- ✓ *remove highest priority item*
- can be implemented as an unordered list
 - add is $O(1)$, findHighest is $O(N)$, removeHighest is $O(N)$
- can be implemented as an ordered list
 - add is $O(N)$, findHighest is $O(1)$, removeHighest is $O(1)$

6

Heaps

Java provides a `java.util.PriorityQueue` class

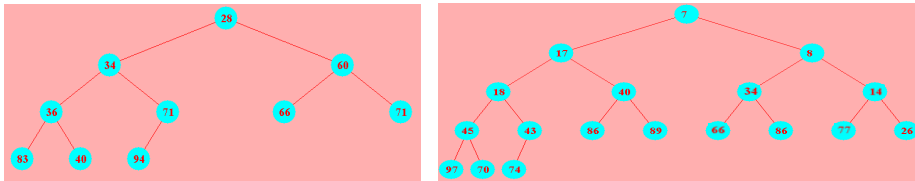
- the underlying data structure is not a list or queue at all
- it is a tree structure called a *heap*

a *complete tree* is a tree in which

- all leaves are on the same level or else on 2 adjacent levels
- all leaves at the lowest level are as far left as possible
- note: a complete tree with N nodes will have minimal height = $\lfloor \log_2 N \rfloor + 1$

a *heap* is complete binary tree in which

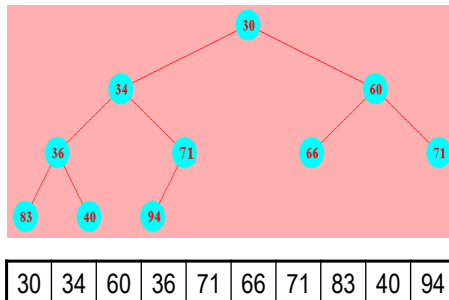
- for every node, the value stored is \leq the values stored in both subtrees
(technically, this is a *min-heap* -- can also define a *max-heap* where the value is \geq)



Implementing a heap

a heap provides for $O(1)$ find min, $O(\log N)$ insertion and min removal

- also has a simple, List-based implementation
- since there are no holes in a heap, can store nodes in an `ArrayList`, level-by-level



- root is at index 0
- last leaf is at index `size() - 1`
- for a node at index i , children are at $2*i+1$ and $2*i+2$
- to add at next available leaf, simply add at end

8

Horner's rule

polynomials are used extensively in mathematics and algorithm analysis

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_1 x + a_0$$

how many multiplications would it take to evaluate this function for some value of x ?

W.G. Horner devised a new formula that transforms the problem

$$p(x) = (\dots (((a_n x + a_{n-1}) x + a_{n-2}) x + \dots + a_1) x + a_0$$

can evaluate in only n multiplications and n additions

9

Problem reduction

in CSC321, we looked at a number of examples of reducing a problem from one form to another

- e.g., generate the powerset (set of all subsets) of an N element set

$$S = \{ x_1, x_2, x_3, x_4 \} \quad \text{powerset}_S = \{ \{ \}, \{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \\ \{x_1, x_2\}, \{x_1, x_3\}, \{x_1, x_4\}, \{x_2, x_3\}, \{x_2, x_4\}, \{x_3, x_4\}, \\ \{x_1, x_2, x_3\}, \{x_1, x_2, x_4\}, \{x_1, x_3, x_4\}, \{x_2, x_3, x_4\}, \\ \{x_1, x_2, x_3, x_4\} \}$$

- PROBLEM REDUCTION: simplify by reducing it to a problem about bit sequences
can map each subset into a sequence of N bits: $b_i = 1 \rightarrow x_i$ in subset

$$\{ x_1, x_4, x_5 \} \leftrightarrow 10011000\dots 0$$

much simpler to generate all possible N -bit sequences

$$\{ 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, \\ 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 \}$$

10

lcm & gcd

consider calculating the least common multiple of two numbers m & n

- BRUTE FORCE: reduce each number to its prime factors
then multiply (factors in both m & n) (factors only in m) (factors only in n)

$$24 = 2 \cdot 2 \cdot 2 \cdot 3$$

$$60 = 2 \cdot 2 \cdot 3 \cdot 5$$

$$\text{lcm}(24, 60) = (2 \cdot 2 \cdot 3) \cdot (2) \cdot (5) = 12 \cdot 2 \cdot 5 = 120$$

- PROBLEM REDUCTION: can recognize a relationship between lcm & gcd

$$\text{lcm}(m, n) = m \times n / \text{gcd}(m, n)$$

$$\text{lcm}(24, 60) = 24 \cdot 60 / 12 = 2 \cdot 60 = 120$$

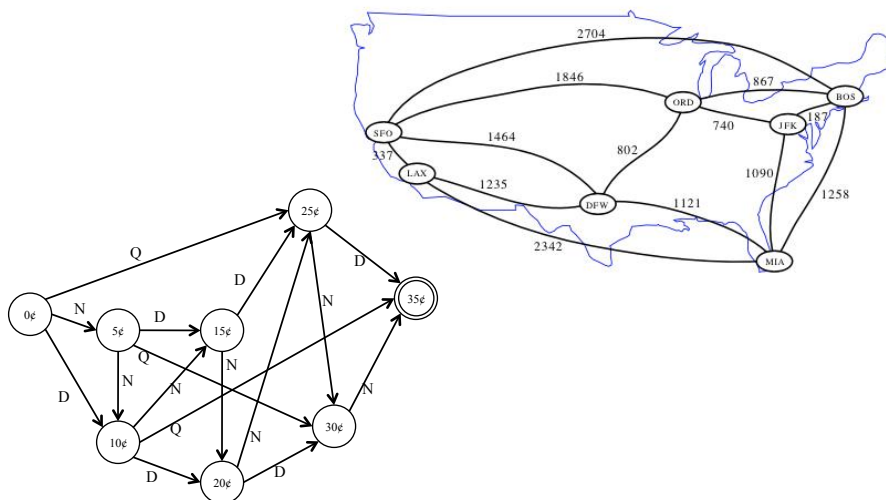
gcd can be calculated efficiently using Euclid's algorithm:

$$\text{gcd}(a, 0) = a \quad \text{gcd}(a, b) = \text{gcd}(b, a \% b)$$

11

Reduction to graph searches

many problems can be transformed into graph problems

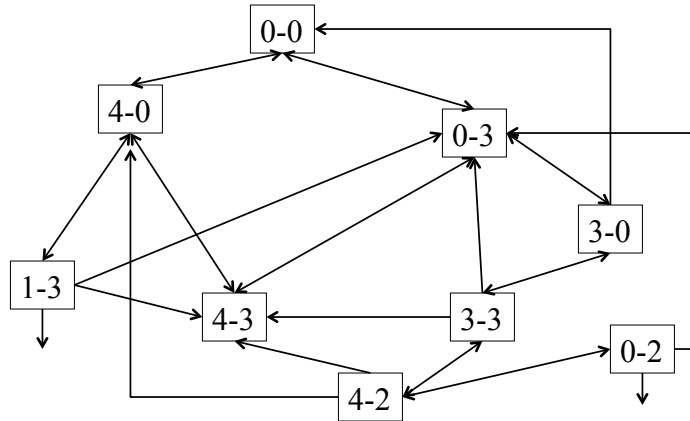


12

Water jug problem

recall from *Die Hard with a Vengeance*

- you have two empty water jugs (4 gallon & 3 gallon capacity) & water supply
- want to end up with exactly 2 gallons in a jug



13

Other interesting examples

multiplication can be transformed into squaring

$$a \times b = ((a + b)^2 - (a - b)^2) / 4$$

find the shortest word ladder

- construct a graph:
 - vertices are the words in the dictionary
 - there is an edge between v_i and v_j if they differ by one letter
- perform a breadth first search, from start word to end word



[Andrew's algorithm for complex hull](#)

- transforms the graph into a sorted list of coordinates, then traverses & collects

14