CSCI 136
Data Structures &
Advanced Programming

Jeannie Albrecht
Lecture 25
April 21, 2014

Administrative Details

- Lab 8 difficulties
  - Anyone stuck?
  - Office hours from 1:30-3 today
- Darwin lab
  - Part 1 due Monday Apr 28 (very easy milestone)
  - Part 2 due Monday May 5
- Midterm 2 is next Wed Apr 30 @1:00 in Wege
- Review session next week? Tuesday at 9:30pm? I'll let you know…

2

Midterm 2

- Will cover all new material since last midterm
- Format - same as Midterm 1
  - ~90 minutes to complete exam (I will give you 120 minutes)
  - Covers book, labs, and lecture material through Priority queues/Heaps and Lab 9 (part 1)
  - Closed book and notes
- Cumulative, but **emphasis** will be on new material
  - Approx one question on old material
- Focus on Ch 7, 8, 10-13
  - Stacks, Queues, Iterators, Comparables and Ordered Structures, Trees, Priority Queues, Heaps

3

Last Time

- Wrapped up Binary Trees
  - Finished discussing tree traversal methods and iterators
  - Talked about Huffman codes

4

Today's Outline

- Look at different ways to represent trees
- Learn about priority queues and heaps

5

Huffman Codes

- General idea
  - Use less bits for most common letters
  - AN ANTARCTIC PENGUIN
  - Compute letter frequencies

```
A: 3      N: 4
T: 2      R: 1
C: 2      I: 2
P: 1      E: 1
G: 1      U: 1
_: 2
```

  - Build tree by recursively creating trees of smallest weighted components
  - Result: 67 bits

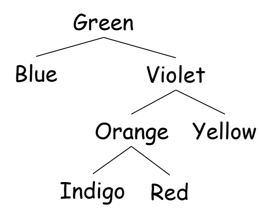
6

Other Compression Techniques

- Examine larger pieces of data for patterns
  - AAAAA BBBBBBBBB CC AAAAAAA
  - (5,A) (9,B) (2,C) (7,A)
- Lempel-Ziv-Welch (LZW)
  - Huffman code for longer substrings
    - ABCABCABC
      - 0-255: ASCII characters
      - 256: AB
      - 257: ABC

7

Alternative Tree Representations



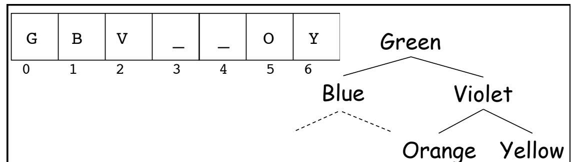
- Total # "slots" = 4n
  - Since each BinaryTree maintains a reference to left, right, parent, value
- Much more overhead than vector, SLL, array, …
- But trees capture successor and predecessor relationships that other data structures don't…

8

Using Arrays to Store Trees

- Encode structure of tree in array indexes
- Where are children of node i?
  - Children of node i are at 2i+1 and 2i+2
  - Look at example
- Where is parent of node j?
  - Parent of node j is at (j-1)/2

9




```
public class ArrayTree {
    protected Object[] data;

    protected int left(int node) {
        return 2*node+1;
    }

    protected int parent(int node) {
        return (node-1)/2;
    }

    …
}
```

10

ArrayTree Tradeoffs

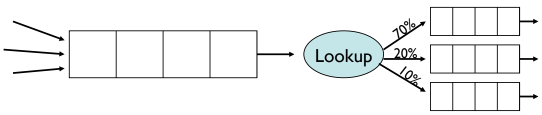
- Why are ArrayTrees good?
  - Save space for links (no "slots" needed)
  - No need for additional memory allocated/garbage collected
  - Works well for full or complete trees
    - Complete: All levels except last are full and all gaps are at right
    - "A *complete* binary tree of height h is a full binary tree with 0 or more of the rightmost leaves of level h removed"
- Why bad?
  - Could waste a lot of space (sparse trees)
  - Height of n requires 2^{n+1}-1 array slots even if only O(n) elements

11

Moving on…

12

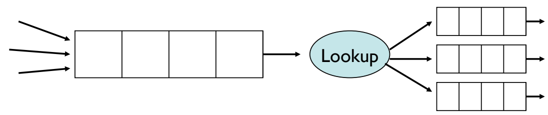
Priority Queues (PQ)



- Recall the use of a "priority queue" in routing
- Give higher priority to some packets so they are routed quicker than others
  - Receive packets in any order but process them according to priority

13

Priority Queues



Packet Sources May Be Ordered by Sender

```
sysnet.cs.williams.edu      priority = 1 (best)
bull.cs.williams.edu                   2
yahoo.com                              10
spammer.com                            100 (worst)
```

14

Priority Queues

- Name is misleading
- PQs are a bit like normal queues, except they are **not FIFO**
- Always dequeue object with **highest priority** regardless of when it was enqueued
- Data can be received/inserted in any order, but it is always returned/removed in same order (according to priority)

15

Priority Queues

- Like ordered structures (i.e., OrderedVectors and OrderedLists), PQs appear to keep data in order
- Unlike ordered structures, PQs allow the user only to remove its "smallest/best" element
- PQs are also similar to Linear structures (i.e., stacks and queues): values are added to the structure, and they later may be inspected or removed
- Unlike Linear structures, once a value is added to PQ it may only be removed if it is the minimum value (i.e., value with highest priority). Not FIFO or LIFO!

16

PQs

- Priority queues are used for:
  - Scheduling processes in an operating system
    - Priority is function of time lost + process priority
  - Order services on server
    - Backup is low priority, so don't do when high priority tasks need to happen
  - Scheduling future events in a simulation
  - Medical waiting room
  - Huffman codes - order by tree size/weight
  - To generally rank choices that are generated out of order

17

PQ Interface

```
public interface PriorityQueue<E extends Comparable<E>> {
   public E getFirst();
   public E remove();
   public void add(E value);
   public boolean isEmpty();
   public int size();
   public void clear();
}
```

18

Things to Note about PQ Interface

- Unlike previous structures, we do not extend any other interfaces
- PriorityQueue methods *consume* Comparable parameters and *return* Comparable values
- (Could be made to use Comparators instead…)

19

Implementing PQs

- Queue?
  - Wouldn't work so well because we can't insert and remove in the "right" way (i.e., keeping things ordered)
- OrderedVector?
  - Keep ordered vector of objects
  - O(n) to add/remove from vector
  - Details in book…
  - Can we do better than O(n)?
- Heap?
  - Partially ordered binary tree

20

Heap

- A heap is a special type of binary tree
- A heap is a **complete** binary tree where:
  - Root holds smallest (highest priority) value
  - Left and right subtrees are also heaps (this is important!)
- So values descend in order (priority) from root to leaf, or ascend as you go up to root from leaf
- Invariant for nodes
  - node.value() <= node.left.value()
  - node.value() <=node.right.value()
- Several valid heaps for same data set (no unique representation)

21

Implementing Heaps

- VectorHeap
  - Use logical array representation of BT (ArrayTree)
  - But use extensible vector instead of array (makes adding elements easier)
- Features
  - No gaps in array -- why?
    - Because BT is always complete in a heap!
  - *Invariant*
    - data[i] <= data[2i+1]; data[i]<=data[2i+2]
  - When elements are added and removed, do small amount of work to "re-heapify"
  - (Example on board)

22