1 # CS3134 #17

10/30/03

Janak J Parekh

2 # Administrivia

3 # Agenda

- Binary non-search trees
  - Trees as arrays
  - Expression trees
  - Huffman trees

4 # Trees as arrays

- Array[0] is the root
- 2*index+1 is the left child
- 2*index+2 is the right child
- Parent of a node is, correspondingly, (index-1)/2
- Actually works surprisingly well, but…
  - No unlimited growth
  - Inefficient use of memory
  - Deletes are slow

5 # Expression trees

- Operators are root and intermediate nodes, operands are leaf nodes
- To create
  - Start with postfix expression and a stack
  - Operand: form unit tree with value and push onto the stack
  - Operator: pop two things off of stack, combine “by” operator, push result on stack
- When done, one element on stack
- What does inorder, preorder, postorder mean?

6 # Huffman trees

- Goal: form trees that let us figure out short binary string prefixes for each letter
  - We can then represent each letter with fewer # of bits
  - Ordinarily, each letter eats 8 or 16 bits (what’s a bit?)
- Procedure
  - Create unit trees with each character and its frequency
  - Put all of these in a priority queue sorted by frequency

7 # Huffman trees (II)

- Procedure (cont’d)
  - While there’s more than one element in the priority queue…
    - Pull off two elements
    - Combine them with a “blank” parent node, whose frequency is the sum of the two children
    - Push back onto priority queue
  - When priority queue has one element, pop it; that’s the Huffman tree
- Navigating the tree
  - Left == 0, Right == 1

8 ☐ Quick review

- We've learned…
  - Array Lists
  - Linked Lists
  - Stacks
  - Queues
  - Trees
- Various performance metrics?
- We can do better on a number of them!

9 ☐ Next time

- Start hashing