CS W3134: Data Structures in Java

Lecture #18: Trees
11/11/04
Janak J Parekh

Administrivia

- HW#4 due today
  - 15 minutes and counting…
- HW#5 going out today
  - I'll talk about it briefly after we finish trees

Agenda

- Finish trees
- Start hashing if we have time

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

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?

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

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

Dictionary, set models

- Many applications are interested in keeping a 2-tuple (coordinate pair) of data
  - (key, value), i.e., index maps to data
- For example,
  - (Dictionary, definition) – this is why it's called a "dictionary" structure
  - (SSN, Employee Record)
- Also called *Map* model

Dictionary, set models (II)

- Alternative: *set* data model
  - Does it exist, or does it not?
- Trees support both
  - Trees can *index* dictionaries for fast lookup
- Book's tree code uses this model
  - Let's take a quick look at it
- Next data structure (hash table) will support even faster dictionary/set operations

Dealing with duplicates

- Especially common with dictionaries
- Example: given a last name, return all matching people in the database
- To do this, make the value/data node a List instead of just one data element
- Need to check for equality in insert/find/delete in addition to inequalities

HW#5 programming

- Email search capability
- How?
  - Read mbox mail format
  - Generate an EmailHeader object
  - Create a tree indexed (keyed) by each body word, linking to EmailHeader object
- Let's draw a picture
- Surprisingly straightforward to do

Quick review

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

Hash Table

- Believe it or not, we can build a data structure that has O(1) performance for insert, search, remove
- Several disadvantages
  - Array-based, so sometimes difficult to expand
  - Performance can suffer based on various parameters
  - Can't visit items in order

Keys?

- In general, we want to make lookup by keys very fast
- In an array, the *index number* is the key
  - Not useful as a "real" key, as this number may change
  - But numbers are very fast.
- OK, so how do we use a "word" as a key?
  - We convert it to a number somehow

Here's a simple one…

- Take the numeric value of all the letters
  - a = 1, b = 2, … , z = 26
  - Add them together
  - Put the word in that cell
    - cats == 43
- How well would this work?
  - What's the minimum value?
  - What's the maximum value for a 10-letter word?
  - How many words could be in between?

Next time

- Hashing