ADT
Odds & Ends

Lecture 19

CS211 – Fall 2005

Announcements

- Prelim 2
  - Tuesday, Nov 15, 7:30-9pm
  - Two weeks from today!
  - Topics: all material through today (Nov 1)
  - Does *not* include
    - GUIs in Java
    - Graphs
- Exam conflicts
  - Email Kelly Patwell (soon)
- Prelim 2 Review Session
  - Sunday, Nov 13,1:30-3:00pm, Kimball B11
  - See *Exams* on course website for more information
  - Individual appointments are available if you cannot attend the review session (email *one* TA to arrange appointment)
- Old exams are available for review on the course website

Mathematical Contest in Modeling 2006

- International competition
- A team of three undergrads chooses one of two open-ended ("real-world") problems
- Important dates
  - Nov 2, 8: info and training
  - Nov 11-15: local (Cornell) contest
  - Feb 2-6: International MCM 2006
- For more information:

http://www.math.cornell.edu/~mcm/

- Recent problems included
  - Estimating max "safe" number of people for a given type of public facilities
  - Studying hunting strategies for velociraptor dinosaurs based on fossil data
  - Comparing various grading policies to fight "grade inflation"
  - Providing guidelines for selecting among bicycle wheel designs to optimize the performance on a given track
  - Considering effects of different airline overbooking strategies on overall profitability

A5 Correction

- In problem 7b, the desired runtime should be O(n + k log n) [instead of O(n + k log k)]

Linear & Quadratic Probing

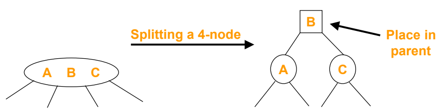
- These are techniques in which all data is stored directly within the hash table array
- Linear Probing
  - Probe at h(X), then at
    - h(X) + 1
    - h(X) + 2
    - …
    - h(X) + i
  - Leads to *primary clustering*
    - Long sequences of filled cells
- Quadratic Probing
  - Similar to Linear Probing in that data is stored within the table
  - Probe at h(X), then at
    - h(X)+1
    - h(X)+4
    - h(X)+9
    - …
    - h(X)+ i^2
  - Works well when
    - $\lambda < 0.5$
    - table size is *prime*

Hash Table Pitfalls

- Good hash function is *required*
- Watch the load factor (λ), especially for Linear & Quadratic Probing

Example Balancing Scheme: 234-Trees

- Nodes have 2, 3, or 4 children (and contain 1, 2, or 3 keys, respectively)
- All leaves are at the same level
- Basic rule for insertion: We hate 4-nodes
  - Split a 4-node whenever you find one while coming down the tree
  - Note: this requires that parent is not a 4-node
- Delete is harder than insert
  - For delete, we hate 2-nodes
  - As in BSTs, cannot delete from a nonleaf so we use same BST trick: delete successor and recopy its data



234-Tree Analysis

- Time for insert or get is proportional to tree's height
- How big is tree's height *h*?
- Let *n* be the number of nodes in a tree of height *h*
  - n is large if all nodes are 4-nodes
  - n is small if all nodes are 2-nodes
- Can use this to show
  h = O(log n)

Analysis of tree height:

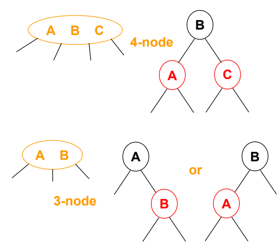
- Let *N* be the number of nodes, *n* be the number of items, and *h* be the height
- Define *h* so that a tree consisting of a single node is height 0
- It's easy to see $1+2+4+\ldots+2^h \leq N \leq 1+4+16+\ldots+4^h$
- It's also easy to see $N \leq n \leq 3N$
- Using the above, we have $n \geq 1+2+4+\ldots+2^h = 2^{h+1}-1$
- Rewriting, we have $h \leq \log(n+1) - 1$ or $h = O(\log n)$

Thus, Dictionary operations on 234-trees take time O(log n) in the worst case

234-Tree Implementation

- Can implement all nodes as 4-nodes
  - Wasted space
- Can allow various node sizes
  - Requires recopying of data whenever a node changes size
- Can use BST nodes to emulate 2-, 3-, or 4-nodes

Using BSTs to Emulate 234-Trees



- A 2-node can be represented with a standard BST node
- A 4-node can be represented with three BST nodes
- A 3-node can be represented with two BST nodes (in two different ways)

Red-Black Trees

- We need a way to tell when an emulated 234-node starts and ends
- We mark the nodes
  - Black: "root" of 234-node
  - Red: belongs to parent
  - Requires one bit per node
- 234-tree rules become rules for *rotations* and color changes in red-black trees
- Result:
  - One black node per 234-node
  - Number of black nodes on path from root to leaf is same as height of 234-tree
  - All paths from root to leaf have same number of black nodes
  - On any path: at most one red node per black node
  - Thus tree height for red-black tree is O(log n)

Balanced Tree Schemes

- AVL trees [1962]
  - Named for initials of Russian creators
  - Uses *rotations* to ensure heights of child-trees differ by at most 1
- 23-Trees [Hopcroft 1970]
  - Similar to 234-tree, but repairs have to move back up the tree
- B-Trees [Bayer & McCreight 1972]
- Red-Black Trees [Bayer 1972]
  - Not the original name
- Red-black convention & relation to 234-trees [Guibas & Stolfi 1978]
- Splay Trees [Sleator & Tarjan 1983]
- Skip Lists [Pugh 1990]
  - developed at Cornell

Selecting a Dictionary Scheme

- Use an unordered array for small sets (< 20 or so)
- Use a Hash Table if possible
  - Cannot efficiently do some ops that are easy with BSTs
  - Running times are expected rather than worst-case
- Use an ordered array if few changes after initialization
- B-Trees are best for large data sets, external storage
  - Widely used within database software
- Otherwise, Red-Black Trees are current scheme of choice
- Skip Lists are supposed to be easier to implement
  - But shouldn't have to implement—use existing code
- Splay trees are useful if some items are accessed more often than others
  - But if you know which items are most-commonly accessed, use a separate data structure

Selecting a Priority Queue Scheme

- Use an unordered array for small sets (< 20 or so)
- Use a sorted array or sorted linked list if few insertions are expected
- Use an array of linked lists if there are few priorities
  - Each linked list is a queue of equal-priority items
  - Very easy to implement
- Otherwise, use a Heap if you can
- Heap + Hashtable
  - Allow *change-priority* operation to be done in O(log n) expected time
- Balanced tree schemes
  - Useful if need special ops
- There are a number of alternate implementations that allow additional operations
  - Skew heaps
  - Pairing heaps
  - Fibonacci heaps
  - …

ADT Summary

- Stack
  - Push/pop
  - **O(1) worst-case** time using linked list
- Queue
  - Put/get
  - **O(1) worst-case** time using linked list
- Priority Queue
  - Put/getMax
  - **O(log n) worst-case** time using heap (if max heap-size is known)
  - **O(log n) expected** time using heap + table-doubling
- Set
  - Insert/remove/query
  - **O(1) worst-case** time using bit vector (if universe is small)
  - **O(1) expected** time using hash-table + table-doubling
- Dictionary
  - Insert/remove/update/find
  - **O(1) expected** time using hash-table + table-doubling
  - **O(log n) worst-case** time using balanced tree
- Still to come: Graphs
  - Not included on Prelim 2