Lecture 21: Heaps & Heapsort

CS 62
Fall 2017
Kim Bruce & Alexandra Papoutsaki

Lab Today

• Build binary search trees
  Different from heap!
• A binary tree is a binary search tree iff
  • it is empty or
  • if the value of every node is both greater than or equal to every value in its left subtree and less than or equal to every value in its right subtree.
• How do you build binary search tree?
  • Insert by following from root until find empty slot

Quiz Friday

• Array representations of trees
• Priority queues
• Heapsort

Array Representation of Trees

• data[0..n-1] can hold values in trees
  • left subtree of node i in 2*i+1, right in 2*i+2,
  • parent in (i-1)/2

Indices: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
data[]: U O R C M E S -- -- P T -- --
How bad can it be?

- What if long stringy tree (e.g. only single left-most branch)?
  - How much space to hold n elements.
  - If complete what is height?

```
1
 / 
2 / 
3 / 
O(2^n) space in worst case!
```

Min-Heap

- Min-Heap H is complete binary tree s.t.
  - H is empty, or
  - Both of the following hold:
    - The value in root position is smallest value in H
    - The left and right subtrees of H are also heaps.
    
Equivalent to saying parent ≤ both left and right children

- Excellent implementation for priority queue
  - Dequeue elements with lowest priority values before higher

Implementations

- As regular queue (array or linked) where either keep in order or search for lowest to remove:
  - One of add or remove will be O(n)

- Heap representation (in arraylist) is more efficient: O(log n) for both add and remove.
  - Insert into heap:
    - Place in next free position,
    - "Percolate" it up.
  - Delete:
    - remove root,
    - move last element in array up to root. "Push" it down.

See VectorHeap code!

*Called PriorityQueue class in standard Java*
Sorting with Trees

Tree Sort

- Build Binary search tree (later)
- Do Inorder traversal, adding elts to array
  - Inorder traversal: $O(n)$
  - Building tree:
    - $\log 1 + \log 2 + \ldots + \log n = O(n \log n)$ in best (and average) case
    - $O(n)$ in worst case
- $O(n \log n)$ in best & average case
- $O(n^2)$ in worst case
- Heapsort is always better!

Heapsort

- Make vector into a heap:
  - $n$ add operations = $O(n \log n)$
- Remove elements in order
  - $n$ remove operations = $O(n \log n)$
- Total: $O(n \log n)$
  - If clever can make into heap in $O(n)$
  - ... but still $O(n \log n)$ total.
  - $O(1)$ extra space (for swaps)

Comparing Sorts

- Quicksort: fastest on average $O(n \log n)$, but worst case is $O(n^2)$ & takes $O(\log n)$ extra space
- Heapsort: $O(n \log n)$ in average & worst case. No extra space.
  - Bit slower on average than quick & mergesorts.
- Mergesort: $O(n \log n)$ in average and worst case. $O(n)$ extra space.
  - Performs well on external files where not all fit in memory.