IE170: Algorithms in Systems Engineering: Lecture 9

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

February 2, 2007

Taking Stock

Last Time

- Binary Search Trees
- Java Collections Interfaces: Maps
- `Heap != Binary Search Tree`

This Time

- Heaps
- Heap Sort

Heaps

- A heap is a balanced binary tree with additional structure that allows it to function efficiently as a priority queue.
- There are two types of heaps: $\max$ and $\min$. In lecture, I'll stick to $\max$

Priority Queue (Max)

- $\text{INSERT}(x)$
- $\text{MAXIMUM}()$
- $x = \text{EXTRACT-MAX}()$
- $\text{INCREASE-KEY}(x, k)$

Heaps

- Heaps are a bit like binary search trees, however, they enforce a different property

Heap Property: Children are Horrible!

- In a max-heap, the key of the parent node is always at least as big as its children:

$$k(p(x)) \geq k(x) \quad \forall x \neq root$$

- Children are great in min-heaps

How to Keep the Heap Property?

- Consider a tree in which all nodes except for one have the heap property.
- We can transform this into a tree in which every node has the heap property.
- This operation is called $\text{HEAPIFY}()$.

Heapify

HEAPIFY(x)

1. Find largest of $k(x)$, $k(\ell(x))$, $k(r(x))$
2. If $k(x)$ is largest, you are done
3. Swap x with largest node, and call $\text{HEAPIFY}()$ on the new subtree

- Intuition behind analysis: Heap is binary tree, so $\leq \lg n$ levels. There is a constant amount of work at each level: comparing three items and swapping two.
- $\Rightarrow$ HEAPIFY a node in $O(\lg n)$
- Alternatively, HEAPIFY node of height h is $O(h)$
    - Height of node: number of edges on path to leaf

To Build a Heap

- By calling `heapify()` on each node, starting at the next to last level and working upward, we can transform an unordered binary tree into a heap.

Analysis

- $O(n)$ calls to HEAPIFY, each of which takes $O(\lg n)$ $\Rightarrow n \lg n$
- But we can do better!

Building A Heap – Analysis

- Note that HEAPIFY really takes $O(h)$ on a node of height h
- There aren't "too many" high nodes. In fact, there are $\leq \lceil n/(2^{h+1}) \rceil$
- Total Running Time is no more than

$$\sum_{h=1}^{\lg n} \lceil \frac{n}{2^{h+1}} \rceil O(h) = O\left(n \sum_{h=0}^{\lg n} \frac{h}{2^h} \right).$$

- Since $\sum_{h=0}^{\infty} h/2^h = 2$, running time to make a heap is $O(n)$.

Operations on a Heap

- The node with the highest key is always the root.
- To delete a record
  - Exchange its record with that of a leaf.
  - Delete the leaf.
  - Call `heapify()`.
- To add a record
  - Create a new leaf.
  - Exchange the new record with that of the parent node if it has a higher key.
  - This is like insertion sort – just move it up the path...
  - Continue to do this until all nodes have the heap property.
  - Note that we can change the key of a node in a similar fashion.

Time for Heap Operations

CREATE	$O(n)$
MAXIMUM	$\Theta(1)$
HEAPIFY	$O(\lg n)$, or $O(h)$
EXTRACT-MAX	$O(\lg n)$
HEAP-INCREASE-KEY	$O(\lg n)$
INSERT	$O(\lg n)$

Heap Sort

- Suppose the list of items to be sorted are in an array of size n
- The heap sort algorithm is as follows.
  1. Put the array in heap order as described above.
  2. In the i^{th} iteration, exchange the item in position 0 with the item in position $n - i$ and call `heapify()`.
- Why is this correct?
- What is the running time?

Next Time?

- Review, Review, Review.
- We have covered chapters 1-4, 6, 10-11, and Appendices A and B: That's a lot!

News

- Homework due 2/5 – No late homework – We do review on 2/5
- Quiz on 2/7

Bear Down, Chicago Bears!