Algorithms in Systems Engineering
ISE 172

Lecture 15

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - Sections 6.5-6.7
- References
  - CLRS Chapter 22
  - R. Sedgewick, *Algorithms in C++* (Third Edition), 1998.

Priority Queues

- The next two lectures will cover the implementation of *priority queues*.
- Recall the Priority Queue ADT from Lecture 12.
  - A priority queue is a data structure for maintaining a list of items that have associated *priorities*.
  - The usual operations are
    * construct a queue from a list of items.
    * find the item with the highest priority.
    * insert an item.
    * delete an item.
    * change the priority of an item.
- Recall that any implementation of a priority queue can be used to sort a list of items.
  - Put the items in a priority queue.
  - Delete the maximum item n times.

Heaps

- A *heap* is a balanced binary tree with additional structure that allows it to function efficiently as a priority queue.
- The additional structure needed to support these operations is that the record stored at each node has a higher priority than either of its children.
- Any node with this property is said to satisfy the *heap property*.
- Consider a tree in which all nodes except for the root have the heap property.
- We can easily transform this into a tree in which every node has the heap property (how?).
- This operation is called `heapify()`.
- By calling `heapify()` on each node, starting at the lowest level and working upward, we can transform an unordered binary tree into a heap.

Operations on a Heap

- The node with the highest priority 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 priority.
  - Continue to do this until all nodes have the heap property.
- Note that we can change the priority of a record in a similar fashion.

Heap Sort

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