Algorithms in Systems Engineering
IE170

Lecture 17

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - CLRS Chapter 24
- References
  - R. Sedgewick, *Algorithms in C++* (Third Edition), 1998.

Dijkstra's Algorithm

- We will assume for now that the edge lengths are all positive.
- The idea of Dijkstra's Algorithm is to perform a graph search, updating the shortest known path to each encountered vertex as the search evolves.
- Throughout the algorithm, we maintain a quantity $d(v)$ for each node v, which represents the length of the shortest path found so far.
- We will call $d(v)$ the current *estimate* for node v.
- We start by assigning $d(r) = 0$ for the source node and $d(v) = \infty$ for all other nodes v.
- The node v to be processed next is the unprocessed node for which $d(v)$ is minimized.
- The processing step consists of updating the estimates for all the neighbors of v.

Algorithm Summary

- We are given a graph $G = (V, E)$ and a source node node r from which we want to find shortest paths to all other nodes.
- Algorithm
  - Initialize by assigning $d(r) = 0$ for the source node and $d(v) = \infty$ for all other nodes $v \in V \setminus \{r\}$.
  - Place r on the list L of unprocessed nodes.
  - While L is not empty
    * Choose $v \in L$ such that $d(v) = \min_{u \in L} d(u)$.
    * For each neighbor x of v, set $d(x) = \min\{d(x), d(v) + w_{\{v,x\}}\}$.
- When the algorithm is completed, we will have $d(v) = \delta(v)$ for all $v \in V$ and the search tree will be a shortest paths tree.
- Why is this algorithm correct?

Implementing the Algorithm

- To implement the algorithm, we need to maintain the node list L as a *priority queue*.
- Recall from Lecture 7 that 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.
- A “naive” implementation is to simply maintain a vector of the estimates for each node and then scan through it each time to find the minimum.
- We may be able to do better using a heap (remember those?).

Review: Heaps

- A *heap* is a 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.

Review: Operations on a Heap

- The node with the highest priority is always the root.
- To delete a record
- To add a record
- We can change the priority of a record in a similar fashion.

Running Time of Dijkstra's Algorithm

- This algorithm fits our definition of a graph search algorithm (roughly speaking), but cannot be analyzed in exactly the same way.
- We will consider the processing step to include both
  - deletion of the next node to be processed, and
  - adjustment of the estimates of all its neighbors.
- The "naive" implementation
- Using a heap
- Which algorithm is better?