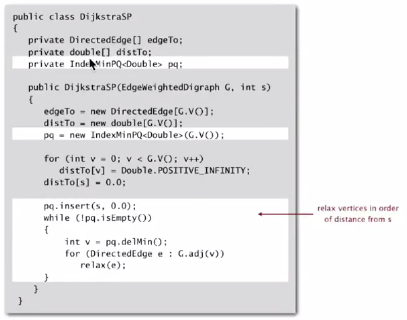
Djikstra’s Algorithm

* Consider vertices in increasing order of distance from s   
  (non-tree vertex with the lowest distTo[] value)
* Add vertex to tree and relax all edges pointing from that vertex

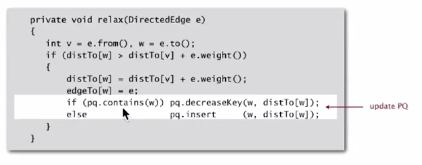
Proof:

* Each edge e = v -> w is relaxed exactly once (when v is relaxed),   
  leaving distTo[w] <= distTo[v] + e.weight()
* Inequality holds until algorithm terminates because:
  + distTo[w] cannot increase (distTo[] values are monotone decreasing)
  + distTo[v] will not change (edge weights are nonnegative and we choose lowest distTo[] value at each step)
* Thus, upon termination, shortest-paths optimality conditions hold.

Djikstra’s Algorithm implementation



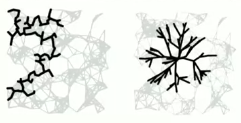
Relax:



Djikstra’s algorithm is familiar…

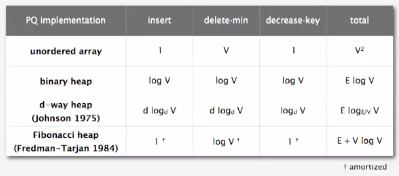
* Prim’s algorithm is essentially the same algorithm
* Both are in a family of algorithms that compute a graph’s spanning tree

Main distinction: rule used to choose next vertex for the tree



* Prim’s: closest vertex to the tree (via an undirected graph)
* Djikstra’s: closest vertex to the source (via a directed path)  
  \*\*Note: DFS and BFS are also in this family of algorithms

Which priority queue? Depends on PW implementation: V insert, V delete-min, E decrease-key



Original implementation used an unordered array, but this is not adequate for large sparse graphs.

* **Binary heap** far more feasible structure (use indexing trick for key decreases): E log V run time

Bottom line:

* Array implementation for dense graphs
* Binary heap implementation for sparse graphs
* 4-way heap only worth is when perf is critical
* Fibonacci heap is best in theory, but nor worth it