Algorithms – CMSC-27200

Dijkstra's algorithm

László Babai

2-5-2015

The problem solved is the single source min-cost paths problem for digraphs with nonnegative edge weights.

INPUT: $G(V, E, w, s)$ where (V, E) is a digraph, $w : E \to \mathbb{R}$ is a weight function, $s \in V$ is a source vertex. All weights are assumed nonnegative.

OUTPUT: for each $v \in$ the cost $c(v)$ of a min-cost path from s to v; and for each $v \in V, v \neq s$ accessible from s, a parent link $p(V) \in v$ such that $c(v) = c(p(v)) + w(p(v), v)$.

The parent links define a tree (the "Dijkstra tree") and a min-cost path from s to v can be traced backwards from v to s along the parent links.

Note that the min-cost paths themselves are not part of the output because this would create an output of size potentially $\Omega(|V|^2)$. Instead, the Dijkstra tree provides a data structure such that for all v, a min-cost path from s to v can be found at the cost of the combinatoral length (number of steps taken) in a min-cost path.

REMARK on cost. We use a model where the following operations are unit cost: arithmetic with and comparison of reals; link updates and other bookkeeping operations.

The algorithm will maintain the following variables: status(v) (current status: white, grey, or black), $c(v)$ (current cost) and $p(v)$ (current parent) for each $v \in V$; and a Priority Queue Q containing pairs $(v, c(v))$ prioritized by the key $c(v)$. When we write $v \in Q$ it is implicit that v is in the queue with key $c(v)$.

The meaning of the status indicators, as in the case of DFS, is that vertices that have not been discovered yet are white, vertices already discovered but not yet finished are grey, and vertices that have been finished ("explored") are black. While the status indicators are not absolutely necessary and many presentations of Dijkstra's algorithm merge the white and grey categories, we find that the distinction of the three categories is illuminating.

We describe the algorithm in pseudocode.

procedure DIJKSTRA(V, E, w, s)

```
for  v ∈ V do
    c(v) := ∞, p(v) = NIL, status(v) = white
c(s) = 0, p(s) = s, status(s) = grey
Q = ∅, INSERT(Q, s)                                 [variables initialized]
while  Q ≠ ∅  do                                    [main loop begins]
    u ← EXTRACT-MIN(Q)
        for v ∈Adj[u]                               [exploration of v begins]
            if  status(v) = white  then do
                RELAX(u, v)
                status(v) := grey
                INSERT(Q, v)
            elseif status(v) = grey  then do
                RELAX(u, v)                         [exploration of v ends]
    status(u) = black                               [main loop ends]
return arrays c, p
```

The key subroutine, called RELAX(u, v), is an update operation along an edge $(u, v) \in E$.

procedure RELAX(u, v) (where $(u, v) \in E$)

```
if c(v) > c(u) + w(u, v) then
    c(v) := c(u) + w(u, v)                          [this is a DECREASE-KEY operation]
    p(v) := u
```

ANALYSIS.

The *proof of correctness* of the procedure is the subject of a separate handout on **loop invariants.**

Timing. The algorithm refers to three PRIORITY QUEUE operations:

INSERT	$\|V_0\|$ times
EXTRACT-MIN	$\|V_0\|$ times
DECREASE-KEY	$\leq \|E_0\|$ times

where V_0 is the set of accessible vertices and E_0 is the set of accessible edges (edges with both ends accessible).

In the HEAP implementation of the PRIORITY QUEUE, the cost of each operation is $O(\log |V|)$ and therefore the total cost is $O((|V| + |E|) \log |V|)$, slightly superlinear.

We shall see that the FIBONACCI HEAP implementation of the PRIORITY QUEUE (Fredman - Tarjan 1987) reduces the overall cost to $O(|V| \log |V| + |E|)$. This is still slightly superlinear, but for digraphs that are not very sparse in the sense that their average degree is $\Omega(\log |V|)$, it becomes linear.