Dijkstra's Algorithm

We are given a directed graph G in adjacency list format, together with a weight function $w[x, y] > 0$ defined for all edges (x, y) of G. We are given a designated souce vertex s.

We shall create a function $d[v]$ which (in the end) will be the minimal weight path from s to v. We shall create a parent function $\pi[v]$. Applying (in the end) π repeatedly to any vertex v will eventually reach the source. The minimal path from s to v will then be found by going in the opposite direction. (That is, if repeatedly applying π we go $v = v_0$, $v_1 = \pi[v_0]$, $v_2 = \pi[v_1]$, etc., until $v_r = \pi[v_{r-1}] = s$ then the path is $s = v_r$ to v_{r-1} to v_{r-2}, etc. until reaching $v_0 = v$. We shall have a set S of *processed* vertices. This can be stored as a Boolean array. Initially $S = \{s\}$ and at each "step" we add a new vertex u to S. We shall have (critically!) a MIN-HEAP Q consisting of those "reached points" v with $v \notin S$, the HEAP using the function d.

`Initialization:` Set $d[s] = 0$. Set $d[v] = \infty$ for all $v \neq s$. Set $S = \{s\}$. Let $Q = \emptyset$. For $v \neq s$ set $\pi[v] = NIL$.

`Step One:` (Not in text but convenient.)

FOR ALL $v \in ADJ[s]$
    $d[v] = w[s, v]$
    $\pi[v] = s$
    ADD v to Q
END FOR

`Caution:` Adding v to Q or changing a $d[v]$ will take time $O(\ln V)$ as we must retain the MIN-HEAP structure!

Now here is the main step. We write it as a WHILE loop. When all vertices are reachable from s by some path the loop will occur precisely $V-1$ times. The algorithm works even when some v are not reachable, they will still have their original values $d[v] = \infty$, $\pi[v] = NIL$.

`Dijkstra:`

WHILE $Q \neq \emptyset$
$u \leftarrow EXTRACT - MIN[Q]$
ADD u to S
FOR $v \in Adj[u]$ with $v \notin S$
    RELAX[u,v]
END FOR

The key now is RELAX which *updates* the values of $d[v], \pi[v]$. It is convenient to do it in two parts (IF, ELSEIF) depending on whether or not v has already been reached. Note that if neither of the conditions for IF or

ELSEIF are met then there is no updating.

RELAX[u,v]

IF $\pi[v] = NIL$ THEN DO

$pi[v] = u$

$d[v] = d[u] + w[u, v]$

ADD v to Q

END DO

ELSE IF $d[u] + w[u, v] < d[v]$ (*can improve*) THEN DO

$pi[v] = u$

$d[v] = d[u] + w[u, v]$

RESET Q

END DO

The algorithm can perhaps best be understood by its interpreteation just before the $u \leftarrow EXTRACT - MIN[Q]$ step. At that moment for all the processed vertices $w \in S$ the value of $d[w]$ is the correct final value and the value of $\pi[w]$ is correct – that is, the minimal path from s to w is found by starting at w, applying π until reaching s and then reversing. For $w \notin S$ we may think of $d[w], \pi[w]$ as *provisional* values. $d[w]$ represents the shortest total weight of a path that stays inside S (the processed vertices) until the last edge when it goes to w, and $\pi[w]$ represents the penultimate vertex (just before w) on that path. When there is no such path we still have the original $d[w] = \infty$, $\pi[w] = NIL$.

Here is the key mathematical point: Take that $u \notin S$ with minimal $d[u]$. Then that $d[u]$ is the correct final value (that is, smallest weight path over all) and $\pi[u]$ is correct. Why? Well, take *any* path from s to u. At some point it would have to go from S to some point u' not in S. Suppose $u' \neq u$. Then already the path from s to u' has weight $d[u']$. But this is at least $d[u]$ (thats why we picked u with $d[u]$ minimal!) and we still have to get from u' to u so the total weight of the path would be bigger than $d[u]$.