Note on shortest path

Problem 1. Input: A connected graph $G = (V, E)$, weights $w_e > 0$ for each edge $e \in E$ and two vertices $s, t \in V$. **Output:** A minimum weight path from s to t in G.

Algorithm 1. (Simplified) Dijkstra's algorithm

Initialize an array d, index by V, to some dummy value, say ∞.
Initialize an array prev, index by V, to some dummy value, say null.
$d[s] \leftarrow 0$
$S \leftarrow \{s\}$
While $t \notin S$
  Find $e = (u, v) \in E$ with $u \in S, v \in V \setminus S$ minimizing $d[u] + w_{(u,v)}$.
  $d[v] \leftarrow d[u] + w_{(u,v)}$
  $\text{prev}[v] \leftarrow u$
  $S \leftarrow S \cup \{v\}$
Return d and prev

To obtain the path from the output, repeatedly follow the prev pointers, starting from t.

Lemma 1. *Dijkstra's algorithm assigns d values in a non-decreasing order.*

Proof. Suppose not. Look at the first time we assign a value, say $d[v]$, which is less than the previously assigned value, say $d[u]$.

If $\text{prev}[v]$ is u then $d[v] = d[u] + w_{(u,v)} > d[u]$ (since all edges have positive weight). Contradiction.

If $\text{prev}[v]$ is not u then $\text{prev}[v]$ was in S when $d[u]$ was assigned a value. In particular, we could have chosen the edge $(\text{prev}[v], v)$ in that iteration (instead of the edge with u as an endpoint). Contradiction (to Dijkstra's algorithm choosing the minimizing edge in that iteration). $\square$

Lemma 2. *A subpath of a minimum weight path is a minimum weight path (between different endpoints).*

Proof. See Assignment 4. $\square$

Theorem 1. *The d values returned by Dijkstra's algorithm corresponds to minimum weight distance from s.*

In other words, the minimum weight s to v path has weight $d[v]$.

Proof. Suppose on some input graph G with weights w and vertices s, t, this is not the case. Let $d^*[v]$ be the minimum weight distances from s to v.

Choose v with $d^*[v] < d[v]$ such that $d^*[v]$ is minimized. Let $P = s, p_2, p_3, \ldots, p_{k-1}, v$ be a minimum weight path from s to v.

$d^*[v] = d^*[p_{k-1}] + w_{(u,v)}$ by lemma 2. Since all weights are positive, $d^*[p_{k-1}] < d^*[v]$. Therefore, $d^*[p_{k-1}] = d[p_{k-1}]$.

So $\text{prev}[v] \neq p_{k-1}$ as otherwise, $d[v] = d[p_{k-1}] + w_{(u,v)} = d^*[p_{k-1}] + w_{(u,v)} = d^*[v]$ contradicts $d^*[v] < d[v]$.

When the edge $(\text{prev}[v], v)$ was chosen by Dijkstra's algorithm, p_{k-1} was not in S. Otherwise, we could have picked (p_{k-1}, v) so $d[v]$ is at most $d[p_{k-1}] + w_{(u,v)} = d^*[v]$ which again contradicts $d^*[v] < d[v]$.

But this means $d[p_{k-1}]$ was set after $d[v]$. By lemma 1, $d[p_{k-1}] \geq d[v]$, a contradiction to $d[p_{k-1}] = d^*[p_{k-1}] < d^*[v] < d[v]$. $\square$

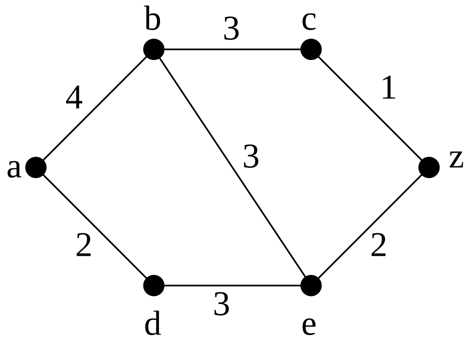
Note. Dijkstra's algorithm can be used to find the minimum weight path from s to all other vertices of the graph (rather than just t). To do this, we just need to replace the stopping condition of our while loop (currently, $t \not{} inS$) by $S \neq V$. Or, more generally, while there is still an edge from S to $V \setminus S$.

Note. Normally, Dijkstra's algorithm is shown using a priority queue so that the step where we need to find the edge minimizing $d[u] + w_{(u,v)}$ is much faster. We would add all edges incident to a vertex v when we add v to S and each edge (v, w) would have value $d[v] + w_{(v,w)}$ in the queue. Each iteration, we would remove the minimum value edge from the queue until we found an edge with one endpoint in S and the other endpoint in $V \setminus S$ (i.e., not both endpoints in S).

However, priority queues are not part of this course.

Example 1. (from p.650 of Rosen's book)

Suppose we wish to find the shortest path from a to z in the following graph (i.e., $s = t$).



Initially $d[a]$ is set to 0.

The edge chosen in the first iteration is (a, d). $d[d]$ is set to $d[a] + w_{(a,d)} = 0 + 2 = 2$ and prev$[d]$ is set to a.

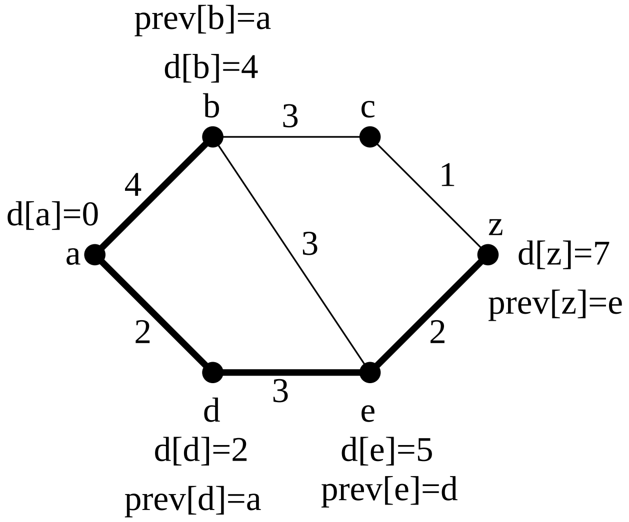
The edge chosen in the second iteration is (a, b). $d[b]$ is set to $d[a] + w_{(a,b)} = 0 + 4 = 4$ and prev$[b]$ is set to a.

The edge chosen in the third iteration is (d, e). $d[e]$ is set to $d[d] + w_{(d,e)} = 2 + 3 = 5$ and prev$[e]$ is set to d.

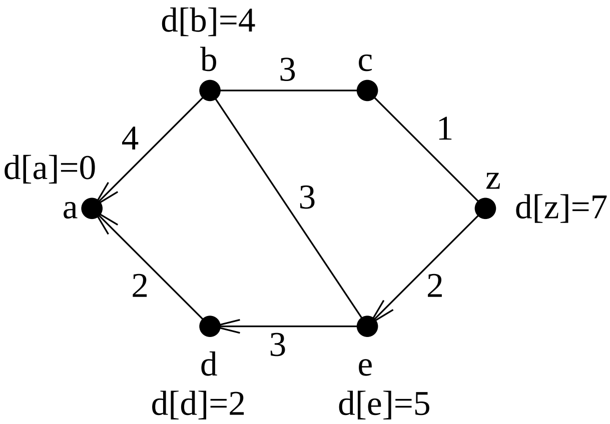
The edge chosen in the fourth iteration is (e, z). $d[z]$ is set to $d[z] + w_{(e,z)} = 5 + 2 = 7$ and prev$[z]$ is set to e.

So the length of the shortest path from a to z is 7. We can obtain this path (in reverse) by following prev pointers in reverse. prev$[z] = e$, prev$[e] = d$, prev$[d] = a$ so the path is a, d, e, z.

Here is the final "state" of the algorithm with selected edges highlighted.



We could draw all of this in a more compact form by drawing an arrow from v to prev$[v]$ (if prev$[v]$ was set to a non-null value).



If we wanted to continue (e.g., if we wanted the shortest path from a to every other vertex), the next edge we would selected is (b, c) and we would set