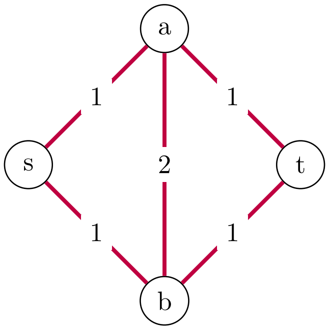
CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2016**

Lalla Mouatadid

Greedy Algorithms: Dijkstra's Shortest Path Algorithm

Let $G(V, E, w)$ be an edge weighted graph, where $w : E \to \mathbb{R}^+$. Let s, t be two vertices in G (think of s as a source, t as a terminal), and suppose you were asked to compute a shortest (i.e. cheapest) path between s and t. Notice that G could possibly have more than one shortest path between s and t. Consider the graph below for instance: Both $P = s, a, t$ and $Q = s, b, t$ are cheapest paths from s to t.

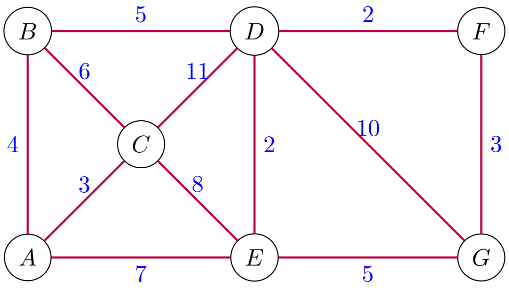


One way to solve this problem is to compute *all st* paths in G, and choose the cheapest. Notice that if the graph is unweighted (think of this as all the edges having equal weights), then starting BFS from s would solve this problem. Maybe we can adjust BFS to take into account the weights.

Recall in BFS vertices are pushed into a queue when visited. When visiting the neighbours of a vertex u, the ordering in which the neighbours enter the queue is arbitrary. If now the goal is to compute the cheapest path, then one way to modify BFS would be to push the cheapest neighbours first. By cheapest, we mean with shortest distance.

"Modified BFS": Consider using a priority queue instead of a queue to collect unvisited vertices. Set the priority to be the shortest distance so far. This is precisely the idea behind **Dijkstra's algorithm**.

Example: Consider the graph below for instance. Suppose we want to compute the cheapest path from $s = A$ to $t = F$.



The table below keeps track of the distances computed at every iteration from the source A to the every vertex in the graph. Read the table as follows: In the fist iteration, the distance from the source to itself is 0, and ∞ to any other vertex. At every iteration, choose the cheapest available vertex and try to build the next cheapest path from said vertex. Repeat the process until all vertices have been visited.

	A	B	C	D	E	F	G	S_i
$i = 0$	0	∞	∞	∞	∞	∞	∞	$\emptyset$
$i = 1$: A	0	4	3	∞	7	∞	∞	$\emptyset$
$i = 2$: C	0	4	3	14	7	∞	∞	$\{AC\}$
$i = 3$: B	0	4	3	9	7	∞	∞	$\{AC, AB\}$
$i = 4$: E	0	4	3	9	7	∞	12	$\{AC, AB, AE\}$
$i = 5$: D	0	4	3	9	7	11	12	$\{AC, AB, AE, BD\}$
$i = 6$: F	0	4	3	9	7	11	12	$\{AC, AB, AE, BD, DF\}$
$i = 7$: G	0	4	3	9	7	11	12	$\{AC, AB, AE, BD, DF, EG\}$

Notice form the table above that at every iteration i, the set S_i is the set of edges that could potentially lead to a shortest path. If $e = (u, v)$ is added to S_i, with $u \in S_i, v \notin S_i$, then the current value of $d[v]$ is a shortest sv path. Formally the algorithm is as follows:

Algorithm 1 Dijkstra's Shortest Path Algorithm

Input: An edge weighted connected graph $G(V, E, w)$ where $w : E \to \mathbb{R}^+$ and two vertices s, t.
Output: A path from s to t with minimum total cost (shortest path)

```
$S = \emptyset$.
Initialize empty priority queue.
foreach $v \in V$ do
    $p[v]$ = NIL                                ▷ predecessor of $v$ n shortest $s, v$ path so far
    $d[v] = \infty$                             ▷ priotity of $v$ = min distance $s, v$ so far.
    enqueue($v$)                                ▷ With priority $d[v] = \infty$.
end for
$d[s] = 0$
Update queue order of $s$
while queue is not empty do                     ▷ Main Loop
    $v$ = dequeue element with min priority $d[]$
    if $p[v]! =$ NIL then
        $S = S \cup \{(p[v], v)\}$
    end if
    foreach edge $(u, v)$ do
        if $u$ is in the queue and $d[v] + w(v, u) < d[u]$ then
            $p[u] = v$
            $d[u] = d[v] + w(v, u)$
            Update queue order of $u$
        end if
    end for
end while
return $S$
```

Proof of Correctness:

We will argue on the S_i's. In particular, at every iteration, we generate subsets of edges $S_1, S_2, \ldots, S_n$. We say that S_i can be extended to some collection of shortest paths S_i^* (this is just a tree of shortest paths) using only edges that do **not** have both endpoints in S_i. That is, we only add edges to S_i with at least one endpoint in the queue.

Loop Invariant

$$S_i \text{ is promising, } \textbf{and} \tag{1}$$

$$\forall u \in S_i, \forall v \notin S_i : d[u] = dist[s, v] \leq dist[s, v] \leq d[v] \tag{2}$$

Where $dist[s,u]$ is the minimum cost of *all* paths from s to u.

Proof. By induction on i.

Base Case: $S_0 = \emptyset$ is trivially promising.

Induction Hypothesis: For some i, suppose S_i can be extended to some shortest paths tree S_i*, using only edges without both endpoints in S_i, and that:

$$d[u] = dist[s,u] \leq dist[s,v] \leq d[v], \text{ for all } u \in S_i, v \notin S_i$$

Induction Step: Consider $S_{i+1} = S_i \cup \{(u,v)\}$ with $u \in S_i$, v outside S_i. We have 2 possible cases:

1. $(u,v) \in S_i^*$:

   If $(u,v) \in S_i^*$ then S_i^* extends S_{i+1} and $dist[s,v] = dist[s,u] + w(u,v)$ since $(u,v) \in S_i^*$. Thus we have:

   $$\begin{aligned} d[v] &= d[u] + w(u,v) \\ &= dist[s,u] + w(u,v) \\ &= dist[s,v] \end{aligned}$$

   Moreover, since $d[v]$ was the cheapest of all $d[\cdot]$ values for vertices outside S_i, it follows that $\forall x \in S_{i+1}$ and $\forall y \notin S_{i+1} : d[x] = d[s,x] \leq d[s,y] \leq d[y]$.

2. $(u,v) \notin S_i^*$:

   Then consider the path P in S_i^* from s to v. P is optimal. Let (z,v) be the last edge on this path.

   If z **were outside** S_i, then let (x,y) be the first edge on P with $x \in S_i, y \notin S_i$ (why does such an edge exist?). We have $d[y] \leq d[x] + w(x,y)$ (because $d[y]$ is the smallest value of $d[t] + w(t,y), \forall e = (t,y), t \in S_i$.) We thus have:

   $$\begin{aligned} d[y] &\leq d[x] + w(x,y) \\ &= dist[s,x] + w(x,y) \\ &< dist[s,x] + w(x,y) + dist[y,v] && \text{(Since } w : E \to \mathbb{R}^+\text{.)} \\ &= dist[s,v] \\ &\leq d[v] \\ \therefore d[y] &< d[v] \end{aligned}$$

   But this contradicts the fact that $d[v]$ is the smallest $d[\cdot]$ value for all vertices outside S_i.

   Therefore z **must be in** S_i. This implies that $dist[s,v] = dist[s,z] + w(z,v) = d[z] + w(z,v)$.

   Since $d[v]$ is the minimum of **all** $d[x] + w(x,v) \forall x \in S_i$, it follows that:

   $$\begin{aligned} d[v] &\leq d[z] + w(z,v) \\ &= dist[s,v] \\ \therefore d[v] &= dist[s,v] && \text{(i.e. it is the shortest distance already.)} \end{aligned}$$

   So we can let $S_{i+1}^* = S_i^* \backslash \{(z,v)\} \cup \{(u,v)\}$ and after the update to $d[a]$ for all $e = (v,a)$ with $a \notin S_i$, we still have that $d[x] = dist[s,x] \leq dist[s,y] \leq d[y]$, for all $x \in S_i, y \notin S_i$.

Hence the loop invariant holds. When the algorithm terminates, we have: $d[u] = dist[s,u], \forall u \in V$. □