It is clear that when the travel time does not vary, this problem is equivalent to the classical problem of finding the shortest path in the graph. We will extend Dijksra's algorithm to this problem.

To explain the idea we will use the analogy with water pipes from the text. Imagine the water spreads in the graph in all directions starting from the initial node at time 0. Suppose the water spreads at the same speed that we travel, i.e. if water reaches the node v at time t and there is an edge $e = (v, w)$ then the water reaches w at time $f_e(t)$. The question is at what time the water first reaches each given site.

The following modification of Dijkstra's algorithm solves this problem. Note that we need to find the fastest way to the destination, and not just the traveling time. To do this we will store for each explored node u the last site before u on the fastest way to u. Then we can restore the fastest path by traversing backward from the destination.

```
Let $S$ be the set of explored nodes.
  For each $u \in S$, we store the earliest time $d(u)$ when we can arrive
at $u$
  and the last site $r(u)$ before $u$ on the fastest path to $u$
Initially $S = \{s\}$ and $d(s) = 0$.
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from $S$ for which
       $d'(v) = \min_{e=(u,v):u \in S} f_e(d(u))$ is as small as possible.
    Add $v$ to $S$ and define $d(v) = d'(v)$ and $r(v) = u$.
EndWhile
```

To establish the correctness of the algorithm, we will prove by induction on the size of S that for all $v \in S$, the value $d(v)$ is the earliest time that water will reach v.

The case $|S| = 1$ is clear, because we know that water start spreads from the node s at time 0. Suppose this claim holds for $|S| = k \geq 1$; we now grow S to size $k + 1$ by adding the node v. For each $x \in S$, let P_x be a path taken by the water to reach x at time $d(x)$. Let (u, v) be the final edge on the path P_v from s to v.

Now, consider any other s-v path P; we wish to show that the time at which v is reached using P is at least $d(v)$. In order to reach v, this path P must leave the set S *somewhere*; let y be the first node on P that is not in S, and let $x \in S$ be the node just before y.

Let P' be the sub-path of P up to node y. By the choice of node v in iteration $k + 1$, the travel time to y using P' is at least as large as the travel time to v using P_v. Since the time-varying edges do not allow traveling back in time, this means that the route to v through y is at least $d(v)$, as desired.

The algorithm can be implemented using a priority queue, as in the basic version of Dijkstra's algorithm: when node v is added, we look at all edges (v, w), and change their keys according to the travel time from v. This takes time $O(\log n)$ per edge, for a total of $O(m \log n)$.

[1]ex670.460.2