Single-Source Minimum Cost Paths

Given:
• a directed graph of \( n \) vertices and \( m \) edges
• a source vertex \( s \)
• for each edge \((v,w)\) a non-negative cost \( c(v,w) \)
  (defined to be \( \infty \) if the edge does not exist)

Problem:
Label each vertex \( v \) with \( D[v] \), the minimum cost of a path from \( s \) to \( v \).
(The cost of a path is the sum of the costs of its edges; we often refer to \( D[v] \) as the distance from the source to \( v \).)

Idea:
• Marked vertices are currently labeled with the correct minimum distance back to the source (initially only the source is marked).
• Unmarked vertices are labeled with the shortest distance back to the source that passes through only marked vertices.
• Each pass add one vertex to the set of marked vertices and updates the unmarked vertices (i.e., if a distance can be improved by using the vertex just marked).
**Dijkstra's Algorithm**

Initialize $D[s] := 0$ and $D[v] := c[s,v]$ for all other vertices.
Initialize all vertices to be unmarked.

**while** all vertices are not marked **do begin**

$v :=$ an unmarked vertex for which $D[v]$ is minimum

Label $v$ "marked".

**for** each unmarked vertex $w$ that is adjacent to $v$ **do**

$D[w] := \text{MINIMUM}\{D[w], D[v]+c(v,w)\}$

**end**

**Note:** Store a back-pointer field with each vertex that points to a vertex on a minimum cost path back to $s$ (so that the minimum cost path from any vertex $v$ to $s$ can be output in time proportional to its length).
Correctness of Dijkstra's algorithm
(induction on the number of marked vertices):

(hypothesis)
For marked vertices, \( D[v] \) is the length of a shortest path; otherwise, \( D[v] \) is the length of a shortest path for which all vertices except \( v \) are marked.

(basis)
Correctness when there is only one marked vertex follows from the initialization.

(inductive step)
• Assume the hypothesis true for \( k \geq 1 \) marked vertices and consider when the \( k+1^{th} \) vertex \( v \) is marked.
• Suppose that there is a path \( P \) from \( s \) to \( v \) such that \( cost(P) < D[v] \), where \( cost(P) \) denotes the sum of the costs of the edges of \( P \).
(Dijkstra correctness continued)

• Since by the inductive hypothesis $D[v]$ is the length of a shortest path from $s$ to $v$ for which all vertices except $v$ are marked, $P$ must contain at least one unmarked vertex besides $v$:

Let $w$ be the first unmarked vertex encountered when traversing $P$ from $s$ to $v$.

Let $Q$ be the portion of $P$ going from $s$ to $w$.

• By the inductive hypothesis, $D[w] \leq \text{cost}(Q)$, and since $\text{cost}(Q) \leq \text{cost}(P) < D[v]$, it follows that $w$ would have been selected instead of $v$ to be marked (so it is not possible that $P$ exists).

• Hence, the inductive hypothesis for marked vertices has been maintained.

• Now consider the unmarked vertices. Since $v$ is the only additional vertex to be marked at this step, and the for loop checks for each vertex adjacent to $v$ if it is shorter to go through $v$, the inductive hypothesis for unmarked vertices is maintained.
Adjacency Matrix Implementation of Dijkstra's Algorithm

Each iteration of the \textit{while} loop can in $O(n)$ time find the $v$ for which $D[v]$ is minimum and then in $O(n)$ time the \textit{for} loop can sequence through a row of the adjacency matrix.

\textbf{Time:} $O(n^2)$

\textbf{Space:} $O(n^2)$
Adjacency List Implementation of Dijkstra's Algorithm

**Time:**

- Keep unmarked vertices in a dictionary with the operations INSERT, DELETEMIN, and DECREASE (make item smaller and fix data structure).
- Initialization can in $O(n)$ time initialize the array $D$, in $O(n)$ time mark all vertices, and in $O(n\log(n))$ time INSERT all vertices into the dictionary.
- Each of the $n$ iterations of the `while` loop, excluding time needed for the inner `for` loop, is dominated by the DELETEMIN used to obtain $v$.
- The inner for loop is executed at most once for each edge, for a total of $O(m)$ DECREASE operations over all iterations of the `while` loop.
- By using a standard balanced tree data structure, the INSERT, DELETEMIN, and DECREASE operations are $O(\log(n))$ time (DECREASE is implemented with a DELETE and an INSERT); yielding a time of $O(m\log(n))$.
- With more complex heap data structures, amortized time for all DECREASE operations can be reduced to $O(m)$, yielding a $O(n\log(n)+m)$ time.

**Space:**

- $O(n+m)$ space is used for the adjacency list representation of the graph.