Dijkstra's Shortest Path

![Graph Diagram]

Given - directed graph \( G(V,E) \)
- weight \( wt(u,v) \geq 0 \) \( \forall (u,v) \in E \)
- source \( S \in V \)

Goal is to find (length of) shortest path from \( S \) to every other vertex.

Def \( \text{dist}(u,v) = \) length of shortest path from \( u \) to \( v \).

Idea - Maintain label \( d[v] \) \( \forall v \in V \).
- \( d[v] = "current\ estimate" \) of \( \text{dist}(S,v) \), i.e. we have already found a \( S \rightarrow v \) path of length \( d[v] \).
- Initially $d[s] = 0$, $d[v] = \infty \quad \forall \, v \neq s$.

**Fact:** It always holds that

$$\text{dist}(s,v) \leq d[v] \quad \forall \, v \in V.$$  

**Edge-Update**

$$d[v] \xleftarrow{\text{update}} \min \{d[v], d[u] + w(u,v)\}.$$  

**Relax (u)**

- \(\forall \, v\) such that \((u,v) \in E\), update
  - $d[v] \leftarrow \min \{d[v], d[u] + w(u,v)\}$.
  - If $d[v]$ got set to $d[u] + w(u,v)$ then set $\text{parent}(v) = u$.

When algorithm terminates, shortest $s \rightarrow V$ path can be traced by tracing parent pointers backwards from $v$.  


Naive Algorithm \( |V| = n, \ |E| = m. \)

Initialize \( d[s] = 0, \ d[v] = \infty \ \forall v \neq s. \)

Repeat \( n \) times.

\[
\begin{array}{c}
\{ \text{For all } u \in V, \} \\
\text{Relax } (u).
\end{array}
\]

Claim. The algorithm, when terminates, gives \( d[v] = \text{dist}(s,v) \ \forall v \in V. \)

Proof. Fix any \( v \in V. \) Let

\[
S = u_0 \rightarrow u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_{k-1} \rightarrow u_k = v,
\]

be shortest \( s \rightarrow v \) path (hypothetical).

In Phase 1: \( s = u_0 \) relaxed: \( \therefore d[u_1] = \text{dist}(s,u_1). \)

In Phase 2: \( u_1 \rightarrow \therefore d[u_2] = d[u_1] + wt(u_1,u_2) = \text{dist}(s,u_1) + wt(u_1,u_2) = \text{dist}(s,u_2). \)

Thus in Phase \( i, \) \( d[u_i] \) gets set to \( \text{dist}(s,u_i). \)

Noting that \( k \leq n, \) we are done. \( \square \)
Dijkstra's Algorithm is a clever implementation of the naive idea:

- Sequence of `Relax(u)` operations, one vertex at a time.
- Always pick vertex `u` with minimum value of `d[u]` (among vertices not yet picked).

**Algorithm**

- `d[s] = 0`, `d[v] = \infty \quad \forall v \neq s`.
- `S = \emptyset` (set of vertices relaxed so far).

While `(V \setminus S \neq \emptyset)` {

- Pick `u \in V \setminus S` with minimum value of `d[u].`
- `Relax(u)`. - Move `u` to `S`.

}  

**Output** - `d[v]` are the distances `dist(s,v)`.
- Parent pointers give the shortest paths.
Claim. When a vertex \( u \) is picked in (*) to relax, it is already the case that \( d[u] = \text{dist}(s, u) \).

Proof.

Let \( S \rightarrow x \rightarrow y \rightarrow u \) be (hypothetical) shortest \( S \rightarrow u \) path where \( x \rightarrow y \) is first edge that jumps outside \( S \).

Note: \( S \rightarrow x \rightarrow y \rightarrow u \) is also shortest path from \( s \) to every vertex on that path.

The claim follows as:
\[ \text{dist}(s,u) \geq \text{dist}(s,y) \]  \quad \text{by shortest path from } s \text{ to } y.

\[ = \text{dist}(s,x) + \text{wt}(x,y) \]

\[ = d[x] + \text{wt}(x,y) \]

\[ \geq d[y] \]

\[ \geq \text{dist}(s,y) \]

\[ \geq d[u]. \quad \text{by inductive hypothesis} \]

\[ \geq d[u]. \quad \text{u had minimum value of } d[u] \text{ in } \forall \setminus S. \]

Hence \[ \text{dist}(s,u) = d[u]. \]
Running Time of Dijkstra's Algorithm

One needs to maintain set of \( n \) numbers \( \{d[v] \mid v \in V\} \) and

- Find/Delete minimum \( n \) operations
- Decrease key \( m \)

Using Fibonacci heaps, Find/Delete Min takes \( O(\log n) \) amortized time and Decrease key takes \( O(1) \) amortized time.

\[ \text{Overall } O(n \log n + m) \text{ time.} \]