Graph Algorithms: Shortest Path

We are given a weighted, directed graph $G = (V, E)$, with weight function $w: E \rightarrow \mathbb{R}$ mapping edges to real valued weights.

The weight of a path $p = (v_1, v_2, \ldots, v_k)$ is

$$\sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

The weight of the path along the red edges is

$$1 + 6 + 1 + 4 = 12.$$
**Dijkstra’s algorithm** solves this problem efficiently for the case in which all weights are nonnegative (as in the example graph).

**Dijkstra’s algorithm** maintains a set $S$ of vertices whose final shortest path weights have already been determined.

It also maintains, for each vertex $v$ not in $S$, an upper bound $d[v]$ on the weight of a shortest path from source $s$ to $v$.

The algorithm repeatedly selects the vertex $u \in V - S$ with minimum bound $d[u]$, inserts $u$ into $S$, and **relaxes** all edges leaving $u$ (determines if passing through $u$ makes it “faster” to get to a vertex adjacent to $u$).
Suppose vertex 1 is the source. **Dijkstra’s algorithm** maintains a set $S$ of vertices whose final shortest path weights have already been determined.

Set $S = \{ \} $ initially.

Also initialize a queue with all the vertices and their upper bounds.
Graph Algorithms: Shortest Path

Remove from the queue the vertex $u \in V - S$ with minimum bound $d[u]$, insert $u$ into $S$, . . .
Graph Algorithms: Shortest Path

Suppose vertex 1 is the source. . . and relax the edges from this vertex.
Suppose vertex 1 is the source.

\[ S = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \]

\[ d[v] = \begin{bmatrix} 3 & 4 & 2 & 5 & 6 \\ 1 & 5 & 10 & \infty & \infty \end{bmatrix} \]

... reordering the queue according to the new upper bounds.
Graph Algorithms: Shortest Path

Suppose vertex 1 is the source.

Repeat . . .
Graph Algorithms: Shortest Path

Suppose vertex 1 is the source.

... remove from queue and put in S
Suppose vertex 1 is the source.

\[ S = \begin{bmatrix} 1 & 3 \\ 0 & 1 \end{bmatrix} \]

\[ d[v] = \begin{bmatrix} 4 & 2 & 5 & 6 \\ 4 & 9 & 2 & \infty \end{bmatrix} \]

\[ \text{s} \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 5 \]

\[ \text{...relax the edges ...} \]
Graph Algorithms: Shortest Path

Suppose vertex 1 is the source.

\[ S = \begin{bmatrix} 1 & 3 \\ 0 & 1 \end{bmatrix} \]
\[ d[v] = \begin{bmatrix} 2 & 4 & 9 & \infty \end{bmatrix} \]

\[ v = \begin{bmatrix} 5 & 4 & 2 & 6 \end{bmatrix} \]

\[ s \rightarrow 1 \rightarrow \cdot \cdot \cdot \]

... and reorder the queue ...
Graph Algorithms: Shortest Path

Suppose vertex 1 is the source.

Repeat . . .
Suppose vertex 1 is the source.

\[ S = \begin{bmatrix}
1 & 3 & 5 \\
0 & 1 & 2 \\
\end{bmatrix} \]

\[ d[v] = \begin{bmatrix}
4 & 2 & 6 \\
4 & 9 & \infty \\
\end{bmatrix} \]

... remove from queue and put in \( S \)
Suppose vertex 1 is the source.

...relax the edges...
Suppose vertex 1 is the source.

... and reorder the queue ...
Graph Algorithms: Shortest Path

Suppose vertex 1 is the source.

\[ S = \begin{bmatrix}
  1 & 3 & 5 \\
  0 & 1 & 2 \\
\end{bmatrix} \]

\[ \nu = \begin{bmatrix}
  4 & 6 & 2 \\
\end{bmatrix} \]

\[ d[v] = \begin{bmatrix}
  4 & 4 & 9 \\
\end{bmatrix} \]

Repeat . . .
Suppose vertex 1 is the source.

\[ S = \begin{bmatrix}
1 & 3 & 4 & 5 \\
0 & 1 & 4 & 2 \\
6 & 2 & 4 & 9 \\
\end{bmatrix} \]

\[ d[v] \]

\ldots remove from queue and put in \( S \)
Suppose vertex 1 is the source.

. . . relax the edges
(no change in this case) . . .
Suppose vertex 1 is the source.

\[ S = \begin{pmatrix} 1 & 3 & 4 & 5 \\ 0 & 1 & 4 & 2 \end{pmatrix} \]

\[ v = \begin{pmatrix} 6 \\ 2 \end{pmatrix} \]

\[ d[v] = \begin{pmatrix} 4 \\ 9 \end{pmatrix} \]

... and reorder the queue (no change in this case) ...
Graph Algorithms: Shortest Path

Suppose vertex 1 is the source.

\[ S = \begin{bmatrix} 1 & 3 & 4 & 5 \\ 0 & 1 & 4 & 2 \end{bmatrix} \]

\[ d[v] = \begin{bmatrix} 6 & 2 \\ 4 & 9 \end{bmatrix} \]

Repeat . . .
Suppose vertex 1 is the source.

\[ S = \begin{bmatrix} 1 & 3 & 4 & 5 & 6 \\ 0 & 1 & 4 & 2 & 4 \end{bmatrix} \]

\[ d[v] = \begin{bmatrix} 2 \\ 9 \end{bmatrix} \]

\[ s \]

\[ 1 \]

\[ 2 \]

\[ 3 \]

\[ 4 \]

\[ 5 \]

\[ 6 \]

\[ \ldots \text{remove from queue and put in} \ S \]
Suppose vertex 1 is the source.

\[
S = \begin{bmatrix}
1 & 3 & 4 & 5 & 6 \\
0 & 1 & 4 & 2 & 4 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
v \\
d[v] \\
\end{bmatrix} = \begin{bmatrix}
2 \\
9 \\
\end{bmatrix}
\]

\[
\ldots \text{relax the edges}
\]

\[
\ldots \text{(no change in this case)} \ldots
\]
Suppose vertex 1 is the source.

Repeat . . .
Suppose vertex 1 is the source.

\[ S = \begin{bmatrix} 1 & 0 \\ 2 & 9 \\ 3 & 1 \\ 4 & 4 \\ 5 & 2 \\ 6 & 4 \end{bmatrix} \]

\[ d[v] \]

**Done!**
Suppose vertex 1 is the source.

**Dijkstra’s algorithm** maintains a set $S$ of vertices whose final shortest path weights have already been determined.

The result is the bottom row which contains the length of the shortest path from $s$ to the vertex above it.
To compute the corresponding paths, we augment the data structure with an additional attribute, $p[v]$, which is the vertex that precedes $v$ in a shortest path.
Using the predecessor attribute, \( p[v] \), it’s easy to construct the path to \( v \), working backwards from \( v \) to \( s \).
Graph Algorithms: Shortest Path

E.g., $v = 6$. 

$S = \{ 1, 2, 3, 4, 5, 6 \}$. 

$d[v] = [0, 9, 1, 4, 2, 4]$. 

$p[v] = [3, 1, 3, 3, 3, 5]$. 

E.g., $v = 6$. 

Graph Algorithms: Shortest Path

E.g., $v = 6$. $p[6] = 5$,
Graph Algorithms: Shortest Path

E.g., $v = 6$.
$p[6] = 5$,
$p[5] = 3$,
Graph Algorithms: Shortest Path

E.g., $v = 6$.

- $p[6] = 5$,
- $p[5] = 3$,

$S = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 9 & 1 & 4 & 2 & 4 \end{bmatrix}$
Graph Algorithms: Shortest Path

\[ S = v \]

\[ d[v] \]

\[ p[v] \]

E.g., \( v = 6 \).

\( p[6] = 5, \)

\( p[5] = 3, \)

\( p[3] = 1. \)

Path: 1, 3, 5, 6.

Weight: 4.
Graph Algorithms: Shortest Path
PseudoCode

DIJKSTRA(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
S ← ∅
Q ← V[G]
while Q ≠ ∅
do u ← EXTRACT-MIN(Q)
    S ← S u {u}
    for each vertex v in AdjList[u]
do RELAX(u,v,w)

Notes:
- G is a directed, weighted graph with nonnegative edge weights
- Edges of G are represented using an Adjacency List
- Q is a Priority Queue. Current distance upper bound (d) is key value in Q
Graph Algorithms: Shortest Path

Computing time analysis: look at different implementations of the priority queue, with different costs for the queue operations.

• EXTRACT-MIN executed how many times?
• EXTRACT-MIN executed $|V|$ times.

• DECREASE-KEY is called by RELAX
• DECREASE-KEY executed how many times?
• DECREASE-KEY executed total of $|E|$ times.

• Total time = $|V| T_{\text{EXTRACT-MIN}} + |E| T_{\text{DECREASE-KEY}}$

<table>
<thead>
<tr>
<th>Priority queue</th>
<th>$T_{\text{EXTRACT-MIN}}$</th>
<th>$T_{\text{DECREASE-KEY}}$</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Array</td>
<td>$O(</td>
<td>V</td>
<td>)$</td>
</tr>
<tr>
<td>Binary heap</td>
<td>$O(\log</td>
<td>V</td>
<td>)$</td>
</tr>
</tbody>
</table>