Lecture 17
Shortest Paths I

• Properties of shortest paths
• Dijkstra’s algorithm
• Correctness
• Analysis
• Breadth-first search

Prof. Erik Demaine
Paths in graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The weight of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

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

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The weight of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

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

Example:

![Diagram of a digraph with vertices $v_1, v_2, v_3, v_4, v_5$ and edges $v_1 \to v_2$ with weight 4, $v_2 \to v_3$ with weight -2, $v_3 \to v_4$ with weight -5, $v_4 \to v_5$ with weight 1. The weight of the path $v_1 \to v_2 \to v_3 \to v_4 \to v_5$ is $w(p) = -2$.]

$w(p) = -2$
Shortest paths

A *shortest path* from $u$ to $v$ is a path of minimum weight from $u$ to $v$. The *shortest-path weight* from $u$ to $v$ is defined as

$$
\delta(u, v) = \min \{ w(p) : p \text{ is a path from } u \text{ to } v \}.
$$

**Note:** $\delta(u, v) = \infty$ if no path from $u$ to $v$ exists.
Optimal substructure

**Theorem.** A subpath of a shortest path is a shortest path.
Optimal substructure

**Theorem.** A subpath of a shortest path is a shortest path.

**Proof.** Cut and paste:
Optimal substructure

**Theorem.** A subpath of a shortest path is a shortest path.

**Proof.** Cut and paste:
Triangle inequality

**Theorem.** For all $u, v, x \in V$, we have
\[ \delta(u, v) \leq \delta(u, x) + \delta(x, v). \]
Triangle inequality

**Theorem.** For all \( u, v, x \in V \), we have
\[
\delta(u, v) \leq \delta(u, x) + \delta(x, v).
\]

**Proof.**

![Diagram showing the triangle inequality with nodes u, v, and x, and edges labeled with distances δ(u, v), δ(u, x), and δ(x, v).]
Well-definedness of shortest paths

If a graph $G$ contains a negative-weight cycle, then some shortest paths may not exist.
Well-definedness of shortest paths

If a graph $G$ contains a negative-weight cycle, then some shortest paths may not exist.

Example:
**Single-source shortest paths**

**Problem.** From a given source vertex \( s \in V \), find the shortest-path weights \( \delta(s, v) \) for all \( v \in V \).

If all edge weights \( w(u, v) \) are *nonnegative*, all shortest-path weights must exist.

**Idea:** Greedy.

1. Maintain a set \( S \) of vertices whose shortest-path distances from \( s \) are known.
2. At each step add to \( S \) the vertex \( v \in V - S \) whose distance estimate from \( s \) is minimal.
3. Update the distance estimates of vertices adjacent to \( v \).
Dijkstra’s algorithm

\[
\begin{align*}
&d[s] \leftarrow 0 \\
&\text{for each } v \in V - \{s\} \\
&\hspace{1cm} \text{do } d[v] \leftarrow \infty \\
&S \leftarrow \emptyset \\
&Q \leftarrow V \quad \triangleright Q \text{ is a priority queue maintaining } V - S
\end{align*}
\]
Dijkstra’s algorithm

\[ d[s] \leftarrow 0 \]
\[ \text{for each } v \in V - \{s\} \]
\[ \text{do } d[v] \leftarrow \infty \]
\[ S \leftarrow \emptyset \]
\[ Q \leftarrow V \quad \triangleright Q \text{ is a priority queue maintaining } V - S \]
\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \]
\[ S \leftarrow S \cup \{u\} \]
\[ \text{for each } v \in \text{Adj}[u] \quad \text{(with } v \text{ from } Q) \]
\[ \text{do if } d[v] > d[u] + w(u, v) \]
\[ \text{then } d[v] \leftarrow d[u] + w(u, v) \]

\text{NOTE: all remaining vertices are not accessible from source}
Dijkstra’s algorithm

\[ d[s] \leftarrow 0 \]
\[ \text{for each } v \in V - \{s\} \] 
\[ \text{do } d[v] \leftarrow \infty \]
\[ S \leftarrow \emptyset \]
\[ Q \leftarrow V \]  \( \triangleright Q \text{ is a priority queue maintaining } V - S \)
\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \]
\[ S \leftarrow S \cup \{u\} \]
\[ \text{for each } v \in \text{Adj}[u] \] (with v from Q)
\[ \text{do if } d[v] > d[u] + w(u, v) \]
\[ \quad \text{then } d[v] \leftarrow d[u] + w(u, v) \]
\[ \text{prev}[v] := u \]

Implicit DECREASE-KEY

NOTE: all remaining vertices are not accessible from source

relaxation step
Example of Dijkstra’s algorithm

Graph with nonnegative edge weights:
Example of Dijkstra’s algorithm

Initialize:

$Q$: $A$ $B$ $C$ $D$ $E$

$S$: $\{\}$
Example of Dijkstra’s algorithm

“$A$” $\leftarrow$ \textbf{EXTRACT-MIN}(\textit{Q}):
Example of Dijkstra’s algorithm

Relax all edges leaving $A$:

$Q$: $A$ $B$ $C$ $D$ $E$

0 $\infty$ $\infty$ $\infty$ $\infty$

10 3 $\infty$ $\infty$ $\infty$

$S$: $\{A\}$
Example of Dijkstra’s algorithm

“C” ← \texttt{EXTRACT-MIN}(Q):

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty
\end{array} \]

\[ S: \{ A, C \} \]
Example of Dijkstra’s algorithm

Relax all edges leaving C:

Q:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

S: \{ A, C \}
Example of Dijkstra’s algorithm

“E” ← \textbf{Extract-Min}(Q):

\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Q:} & \textbf{A} & \textbf{B} & \textbf{C} & \textbf{D} & \textbf{E} \\
\hline
0 & \infty & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty & \infty \\
7 & 11 & 5 & & & \\
\hline
\end{tabular}

\textbf{S:} \{ A, C, E \}
Example of Dijkstra’s algorithm

Relax all edges leaving $E$:

$Q$: $\begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \\
7 & 11 & 5 & \\
7 & 11 & & \\
\end{array}$

$S$: $\{A, C, E\}$
Example of Dijkstra’s algorithm

“B” ← **EXTRACT-MIN**(Q):

![Graph with nodes A, B, C, D, E and edges with weights]

**Q:**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

**S:** \{ A, C, E, B \}
Example of Dijkstra’s algorithm

Relax all edges leaving B:

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & 5 \\
7 & 7 & 11 & 9 & \\
\end{array} \]

\[ S: \{ A, C, E, B \} \]
Example of Dijkstra’s algorithm

“D” ← \textbf{Extract-Min}(Q):

\begin{itemize}
  \item \textbf{Q:} \begin{tabular}{cccccc}
    A & B & C & D & E \\
    0 & \infty & \infty & \infty & \infty \\
    10 & 3 & \infty & \infty & \infty \\
    7 & 11 & 5 \\
  \end{tabular}
  \item \textbf{S:} \{ A, C, E, B, D \}
\end{itemize}
Correctness — Part I

Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.
Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

Proof. Suppose not. Let $v$ be the first vertex for which $d[v] < \delta(s, v)$, and let $u$ be the vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u, v)$. Then,

$$d[v] < \delta(s, v) \quad \text{supposition}$$

$$\leq \delta(s, u) + \delta(u, v) \quad \text{triangle inequality}$$

$$\leq \delta(s, u) + w(u, v) \quad \text{sh. path} \leq \text{specific path}$$

$$\leq d[u] + w(u, v) \quad v \text{ is first violation}$$

Contradiction.
Correctness — Part II

**Lemma.** Let \( u \) be \( v \)’s predecessor on a shortest path from \( s \) to \( v \). Then, if \( d[u] = \delta(s, u) \) and edge \((u, v)\) is relaxed, we have \( d[v] = \delta(s, v) \) after the relaxation.
Correctness — Part II

**Lemma.** Let $u$ be $v$’s predecessor on a shortest path from $s$ to $v$. Then, if $d[u] = \delta(s, u)$ and edge $(u, v)$ is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

**Proof.** Observe that $\delta(s, v) = \delta(s, u) + w(u, v)$. Suppose that $d[v] > \delta(s, v)$ before the relaxation. (Otherwise, we’re done.) Then, the test $d[v] > d[u] + w(u, v)$ succeeds, because $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$, and the algorithm sets $d[v] = d[u] + w(u, v) = \delta(s, v)$.
Theorem. Dijkstra’s algorithm terminates with 
\[ d[v] = \delta(s, v) \] for all \( v \in V \).
Correctness — Part III

**Theorem.** Dijkstra’s algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

**Proof.** It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when $v$ is added to $S$. Suppose $u$ is the first vertex added to $S$ for which $d[u] > \delta(s, u)$. Let $y$ be the first vertex in $V - S$ along a shortest path from $s$ to $u$, and let $x$ be its predecessor:

$S$, just before adding $u$. 
Since $u$ is the first vertex violating the claimed invariant, we have $d[x] = \delta(s, x)$. When $x$ was added to $S$, the edge $(x, y)$ was relaxed, which implies that $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$. But, $d[u] \leq d[y]$ by our choice of $u$. Contradiction.
Analysis of Dijkstra

\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{Extract-Min}(Q) \]
\[ S \leftarrow S \cup \{u\} \]
\[ \text{for each } v \in Adj[u] \]
\[ \quad \text{do if } d[v] > d[u] + w(u, v) \]
\[ \quad \text{then } d[v] \leftarrow d[u] + w(u, v) \]
Analysis of Dijkstra

while $Q \neq \emptyset$
do $u \leftarrow \text{Extract-Min}(Q)$
$S \leftarrow S \cup \{u\}$
for each $v \in \text{Adj}[u]$
do if $d[v] > d[u] + w(u, v)$
then $d[v] \leftarrow d[u] + w(u, v)$
Analysis of Dijkstra

while $Q \neq \emptyset$
  do $u \leftarrow \text{Extract-Min}(Q)$
  $S \leftarrow S \cup \{u\}$
  for each $v \in \text{Adj}[u]$ do
    if $d[v] > d[u] + w(u, v)$ then
      $d[v] \leftarrow d[u] + w(u, v)$
Analysis of Dijkstra

while $Q \neq \emptyset$
    do $u \leftarrow \text{Extract-Min}(Q)$
        $S \leftarrow S \cup \{u\}$
        for each $v \in \text{Adj}[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit Decrease-Key's.
Analysis of Dijkstra

\[
\text{while } Q \neq \emptyset \\
\text{do } u \leftarrow \text{Extract-Min}(Q) \\
\text{S } \leftarrow \text{S } \cup \{u\} \\
\text{for each } v \in \text{Adj}[u] \\
\text{do if } d[v] > d[u] + w(u, v) \\
\text{then } d[v] \leftarrow d[u] + w(u, v)
\]

Handshaking Lemma \(\Rightarrow \Theta(E)\) implicit \text{Decrease-Key}'s.

Time = \(\Theta(V \cdot T_{\text{Extract-Min}} + E \cdot T_{\text{Decrease-Key}})\)

\textbf{Note:} Same formula as in the analysis of Prim's minimum spanning tree algorithm.
Analysis of Dijkstra (continued)

\[ \text{Time} = \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}} \]

<table>
<thead>
<tr>
<th>Q</th>
<th>(T_{\text{Extract-Min}})</th>
<th>(T_{\text{Decrease-Key}})</th>
<th>Total</th>
</tr>
</thead>
</table>
Analysis of Dijkstra (continued)

\[ \text{Time} = \Theta(V) \cdot T_{\text{extract-min}} + \Theta(E) \cdot T_{\text{decrease-key}} \]

<table>
<thead>
<tr>
<th>( Q )</th>
<th>( T_{\text{extract-min}} )</th>
<th>( T_{\text{decrease-key}} )</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>( O(V) )</td>
<td>( O(1) )</td>
<td>( O(V^2) )</td>
</tr>
</tbody>
</table>
Analysis of Dijkstra (continued)

\[ \text{Time} = \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}} \]

<table>
<thead>
<tr>
<th></th>
<th>( Q )</th>
<th>( T_{\text{Extract-Min}} )</th>
<th>( T_{\text{Decrease-Key}} )</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td></td>
<td>( O(V) )</td>
<td>( O(1) )</td>
<td>( O(V^2) )</td>
</tr>
<tr>
<td>binary heap</td>
<td></td>
<td>( O(\lg V) )</td>
<td>( O(\lg V) )</td>
<td>( O(E \lg V) )</td>
</tr>
</tbody>
</table>

Analysis of Dijkstra (continued)

Time = $\Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}}$

<table>
<thead>
<tr>
<th></th>
<th>$Q$</th>
<th>$T_{\text{Extract-Min}}$</th>
<th>$T_{\text{Decrease-Key}}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>$O(V)$</td>
<td>$O(1)$</td>
<td></td>
<td>$O(V^2)$</td>
</tr>
<tr>
<td>binary heap</td>
<td>$O(\lg V)$</td>
<td>$O(\lg V)$</td>
<td></td>
<td>$O(E \lg V)$</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>$O(\lg V)$</td>
<td>$O(1)$</td>
<td></td>
<td>$O(E + V \lg V)$</td>
</tr>
<tr>
<td>amortized</td>
<td>amortized</td>
<td></td>
<td></td>
<td>worst case</td>
</tr>
</tbody>
</table>
Unweighted graphs

Suppose that $w(u, v) = 1$ for all $(u, v) \in E$. Can Dijkstra’s algorithm be improved?
Unweighted graphs

Suppose that $w(u, v) = 1$ for all $(u, v) \in E$. Can Dijkstra’s algorithm be improved?

• Use a simple FIFO queue instead of a priority queue.
Unweighted graphs

Suppose that \( w(u, v) = 1 \) for all \((u, v) \in E\). Can Dijkstra’s algorithm be improved?

- Use a simple FIFO queue instead of a priority queue.

**Breadth-first search**

```
while Q ≠ ∅
do u ← DEQUEUE(Q)
    for each v ∈ Adj[u]
do if d[v] = ∞
    then d[v] ← d[u] + 1
    ENQUEUE(Q, v)
```
Unweighted graphs

Suppose that $w(u, v) = 1$ for all $(u, v) \in E$. Can Dijkstra’s algorithm be improved?
• Use a simple FIFO queue instead of a priority queue.

**Breadth-first search**

while $Q \neq \emptyset$
    do $u \leftarrow \text{DEQUEUE}(Q)$
    for each $v \in \text{Adj}[u]$
        do if $d[v] = \infty$
            then $d[v] \leftarrow d[u] + 1$
            $\text{ENQUEUE}(Q, v)$

**Analysis:** Time $= O(V + E)$. 
Example of breadth-first search

Q:
Example of breadth-first search

- Initial state: Q: a
- Progression:
  - 0

Diagram:
```
  a -- b -- c
   |   |   |
   v   v   v
  d -- e -- f
   |   |   |
   v   v   v
  g -- h -- i
```

The algorithm explores nodes in a breadth-first manner.
Example of breadth-first search

Q: a b d
Example of breadth-first search

```
Q:  a  b  d  c  e
```

![Diagram of a graph with nodes labeled a, b, c, d, e, f, g, h, i, and edges marked with distances 0, 1, 2, and arrows indicating the order of exploration in breadth-first search.]
Example of breadth-first search

Q: a b d c e
Example of breadth-first search

Q: a b d c e
Example of breadth-first search

Q: a b d c e g i
Example of breadth-first search

Q: a b d c e g i f
Example of breadth-first search

Q: a b d c e g i f h
Example of breadth-first search

Q: a b d c e g i f h
Example of breadth-first search

Q: $a \ b \ d \ c \ e \ g \ i \ f \ h$
Example of breadth-first search

Q: a b d c e g i f h
Correctness of BFS

while $Q \neq \emptyset$
do $u \leftarrow \text{DEQUEUE}(Q)$
for each $v \in \text{Adj}[u]$
do if $d[v] = \infty$
then $d[v] \leftarrow d[u] + 1$
$\text{ENQUEUE}(Q, v)$

Key idea:
The FIFO $Q$ in breadth-first search mimics the priority queue $Q$ in Dijkstra.