Introduction to Algorithms
6.046J/18.401J/SMA5503

Lecture 17
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

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

Example:

![Graph diagram with nodes and edges labeled with weights. The weight of path $p$ is calculated as $w(p) = -2$.]
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.

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

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

**Proof.**
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

\[ d[s] \leftarrow 0 \]

for each \( v \in V - \{s\} \) \n
\[ d[v] \leftarrow \infty \]

\[ S \leftarrow \emptyset \]

\[ Q \leftarrow V \quad \triangleright \text{\( Q \) is a priority queue maintaining} \ V - S \]

while \( Q \neq \emptyset \) \n
\[ u \leftarrow \text{EXTRACT-MIN}(Q) \]

\[ S \leftarrow S \cup \{u\} \]

for each \( v \in \text{Adj}[u] \) \n
\[ \text{do if } d[v] > d[u] + w(u, v) \]

\[ \text{then } d[v] \leftarrow d[u] + w(u, v) \quad \text{relaxation step} \]

Implicit \text{DECREASE-KEY}
Example of Dijkstra’s algorithm

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

Initialize:

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

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

“$A$” ← \textsc{Extract-Min}($Q$):

$Q$: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
\end{array}

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

Relax all edges leaving $A$:

$$
\begin{array}{c|c|c|c|c|c}
 Q: & A & B & C & D & E \\
 \hline
 \text{0} & \infty & \infty & \infty & \infty & \infty \\
 \text{10} & 3 & \_ & \_ & \_ & \_ \\
\end{array}
$$

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

“C” ← EXTRACT-MIN(Q):

Q: A B C D E
   0 ∞ ∞ ∞ ∞
   10 3 – –

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

Relax all edges leaving C:

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

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

“E” ← \text{\textsc{extract-min}}(Q):

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

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

Relax all edges leaving E:

Q:  A  B  C  D  E
0  ∞  ∞  ∞  ∞
10  3  ∞  ∞
7  11  5

S: {A, C, E}

© 2001 by Charles E. Leiserson
Example of Dijkstra’s algorithm

“B” ← EXTRACT-MIN(Q):

Q: | A | B | C | D | E |
---|---|---|---|---|---|
0  | ∞ | ∞ | ∞ | ∞ | ∞ |
10 | 3 | ∞ | ∞ | ∞ | ∞ |
7  | 3 | 11 | 5 |
7  | 11 | 5 |

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

Relax all edges leaving B:

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

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

“D” ← \textbf{EXTRACT-MIN}(Q):

\begin{itemize}
  \item \textbf{Q:} \begin{array}{cccccc}
    A & B & C & D & E \\
    0 & \infty & \infty & \infty & \infty \\
    10 & 3 & \infty & \infty & \infty \\
    7 & 11 & 5 \\
    7 & 11 & 9 \\
  \end{array}
  \\
  \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.

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,

\[
\begin{align*}
  d[v] &< \delta(s, v) \quad \text{supposition} \\
  &\leq \delta(s, u) + \delta(u, v) \quad \text{triangle inequality} \\
  &\leq \delta(s, u) + w(u, v) \quad \text{sh. path} \leq \text{specific path} \\
  &\leq d[u] + w(u, v) \quad v \text{ is first violation}
\end{align*}
\]

Contradiction.
Correctness — Part II

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 $d[u] \neq \delta(s, u)$. Let $y$ be the first vertex in $V - S$ along a shortest path from $s$ to $u$, and let $x$ be its predecessor:
Correctness — Part II (continued)

Since \( u \) is the first vertex violating the claimed invariant, we have \( d[x] = \delta(s, x) \). Since subpaths of shortest paths are shortest paths, it follows that \( d[y] \) was set to \( \delta(s, x) + w(x, y) = \delta(s, y) \) when \((x, y)\) was relaxed just after \( x \) was added to \( S \). Consequently, we have \( d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u] \). But, \( d[u] \leq d[y] \) by our choice of \( u \), and hence \( d[y] = \delta(s, y) = \delta(s, u) = d[u] \). Contradiction.
Analysis of Dijkstra

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

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

Time = \( \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-Key}} \)

Note: Same formula as in the analysis of Prim’s minimum spanning tree algorithm.
### Analysis of Dijkstra (continued)

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>
<tr>
<td>binary heap</td>
<td>$O(\lg V)$</td>
<td>$O(\lg V)$</td>
<td>$O(E \lg V)$</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>$O(\lg V)$</td>
<td>$O(1)$</td>
<td>$O(E + V \lg V)$</td>
</tr>
</tbody>
</table>
Unweighted graphs

Suppose $w(u, v) = 1$ for all $(u, v) \in E$. Can the code for Dijkstra be improved?

- Use a simple FIFO queue instead of a priority queue.
- **Breadth-first search**

```plaintext
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

Q: a

0

a

b

c

d

e

f

h

g

i

0
Example of breadth-first search

Q: a b d
Example of breadth-first search

Q: a b d c e

© 2001 by Charles E. Leiserson
Example of breadth-first search

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

Q: a b d e c
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

\[ Q: \quad a \quad b \quad d \quad c \quad e \quad g \quad i \quad f \quad h \]
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.

- **Invariant:** \( v \) comes after \( u \) in \( Q \) implies that
  \( d[v] = d[u] \) or \( d[v] = d[u] + 1 \).