Lecture 15:
• Bellman-Ford

Oct 30, 2018
Bellman-Ford
Dijkstra Recap

Single-source shortest paths

• **Input:** Directed, weighted graph \( G = (V, E, \{\ell_e\}) \), source node \( s \)
  - Non-negative edge lengths \( \ell_e \geq 0 \)

• **Output:** Two arrays \( d, p \)
  - \( d[u] \) is the length of the shortest \( s \leadsto u \) path
  - \( p[u] \) is the final hop on shortest \( s \leadsto u \) path
    \( \leadsto \) edges in the shortest path tree

• **Running time:** \( O(m \log n) \)
  - Implement using **binary heaps**
Ask the Audience

• Show that Dijkstra’s Algorithm can fail in graphs with negative edge lengths

1. Negative length cycles – shortest path may be undefined

![Graph example](image)
Show that Dijkstra’s Algorithm can fail in graphs with negative edge lengths, and no negative-length cycles.
Why Care About Negative Lengths?

• Models various phenomena
  • Transactions (credits and debits)
  • Currency exchange (log(exchange rate) can be + or -)
  • Chemical reactions (can be exo or endothermic)

• Leads to interesting algorithms
  • Variants of Bellman-Ford are used in internet routing
Bellman-Ford

• **Input:** Directed, weighted graph $G = (V, E, \{\ell_e\})$, source node $s$
  - Possibly negative edge lengths $\ell_e \in \mathbb{R}$
  - No negative-length cycles!

• **Output:** Two arrays $d, p$
  - $d[u]$ is the length of the shortest $s \leadsto u$ path
  - $p[u]$ is the final hop on shortest $s \leadsto u$ path
• Suppose we try the following algorithm
  • Take a graph \( G = (V, E, \{\ell(e)\}) \) with negative lengths
  • Add \( C \) to all lengths to make them non-negative
  • Run Dijkstra on the new graph

• Why won't this work?
Ask the Audience

• Suppose we try the following algorithm
  • Take a graph $G = (V, E, \{\ell(e)\})$ with negative lengths
  • Add C to all lengths to make them non-negative
  • Run Dijkstra on the new graph

• Why won't this work?

![Graph](image)
Structure of Shortest Paths

• If \((u, v) \in E\), then \(d(s, v) \leq d(s, u) + \ell(u, v)\) for every node \(s \in V\)

• For every \(v\), there exists an edge \((u, v) \in E\) such that \(d(s, v) = d(s, u) + \ell(u, v)\)

• If \((u, v) \in E\), and \(d(s, v) = d(s, u) + \ell(u, v)\) then there is a shortest \(s \leadsto v\)-path ending with \((u, v)\)
Dynamic Programming

$OPT(v) = d(s, v)$

• **Subproblems:** Let $OPT(v)$ be the length of the shortest path from $s$ to $v$

• If the shortest path from $s$ to $v$, has final hop $(u, v)$, then $d(s, v) = d(s, u) + l(u, v)$

  $OPT(v) = OPT(u) + l(u, v)$

• **Recurrence:** $OPT(v) = \min_{(u, v) \in E} OPT(u) + l(u, v)$

• **Base Cases:** $OPT(s) = 0$
Bottom-Up Implementation?

To topologically order the "DP Table".

If the graph has cycles, then there is no good order.

<table>
<thead>
<tr>
<th>v</th>
<th>s</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT(v)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Programming Take II

• **Subproblems:** Let $OPT(v, j)$ be the length of the shortest path from $s$ to $v$ with at most $j$ hops

• **Fact:** If there are no negative-length cycles, then there exists a shortest path with $\leq n-1$ hops

  **Pf:** If $P$ has $> n$ hops it contains a cycle. Removing that cycle cannot increase the length

• $\therefore \quad OPT(v, n-1) = d(s, v)$

• # of subproblems = $\sum_{v \in V} n \times n = n^2$

  $0 \leq j \leq n-1$
Dynamic Programming Take II

- **Subproblems:** Let $\text{OPT}(v, j)$ be the length of the shortest path from $s$ to $v$ with at most $j$ hops

  - If $P$ uses $\leq j-1$ hops then $\text{OPT}(v, j) = \text{OPT}(v, j-1)$
  - If $P$ uses exactly $j$ hops, and its final hop is $(u,v)$ then $\text{OPT}(v, j) = \text{OPT}(u, j-1) + l(u,v)$
Dynamic Programming Take II

• **Subproblems:** Let $\text{OPT}(v, j)$ be the length of the shortest path from $s$ to $v$ with at most $j$ hops

• **Recurrence:**

$$
\text{OPT}(v, j) = \min \left\{ \text{OPT}(v, j-1), \min_{(u, v) \in E} \text{OPT}(u, j-1) + l(u, v) \right\}
$$

• **Base Cases:**

$$
\text{OPT}(s, 0) = 0
$$

$$
\text{OPT}(v, 0) = \infty \quad \forall \ v \neq s
$$
Recurrence

- **Subproblems**: Let $\text{OPT}(v, j)$ be the length of the shortest path from $s$ to $v$ with at most $j$ hops.
- **Case $u$**: $(u, v)$ is final edge on the shortest $j$-hop $s \leadsto v$ path.

**Recurrence:**

$$\text{OPT}(v, j) = \min \left\{ \text{OPT}(v, i - 1), \min_{(u,v) \in E} \left\{ \text{OPT}(u, i - 1) + \ell_{u,v} \right\} \right\}$$

$\text{OPT}(s, j) = 0$ for every $j$

$\text{OPT}(v, 0) = \infty$ for every $v$
Finding the paths

• $\text{OPT}(v, j)$ is the length of the shortest $s \sim v$ path with at most $j$ hops

• $P(v, j)$ is the last hop on some shortest $s \sim v$ path with at most $j$ hops

Recurrence:

$$\text{OPT}(v, j) = \min \left\{ \text{OPT}(v, j - 1), \min_{(u, v) \in E} \{ \text{OPT}(u, j - 1) + \ell_{u,v} \} \right\}$$
Example

- A graph with nodes s, c, d, b, and e.
- Edge weights: s to b (3), s to c (4), c to b (1), b to d (2), d to c (5), d to e (2), e to b (∞), e to c (∞), e to d (∞).
- A distance matrix:
  
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

\[ \text{OPT}(v, i) = \min \left\{ \text{OPT}(v, o), \min \text{OPT}(u, o) + l(u, v) \right\} \]
Example

The graph above shows a network of nodes labeled s, b, c, d, and e, connected by directed edges with numerical labels. The table on the right provides a summary of the weights between these nodes:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>∞</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>∞</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

Graph:
- Vertex s
- Vertex b
- Vertex c
- Vertex d
- Vertex e

Edges with weights:
- s to b: -1
- b to c: -1
- b to d: 2
- c to d: 3
- d to e: -3
- e to b: 1
- s to d: 4
- e to s: 0

Table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>∞</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>∞</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
**Implementation (Bottom Up DP)**

Shortest-Path\((G, s)\)

\[
\text{foreach node } v \in V \\
D[v,0] \leftarrow \infty \\
P[v,0] \leftarrow \perp \\
D[s,0] \leftarrow 0
\]

\[
\text{for } i = 1 \text{ to } n-1 \quad \rightarrow \quad n-1 \text{ iterations (one per column)} \\
\text{foreach node } v \in V \\
D[v,i] \leftarrow D[v,i-1] \\
P[v,i] \leftarrow P[v,i-1] \\
\text{foreach edge } (u,v) \in E \\
\quad \text{if } (D[u,i-1] + \ell_{uv} < D[v,i]) \\
\quad D[v,i] \leftarrow D[u,i-1] + \ell_{uv} \\
\quad P[v,i] \leftarrow u
\]

Total Time: \(O(nm)\)

Space: \(O(n^2)\)
Optimizations

- One array $d[v]$ containing shortest path found so far
- No need to check edges $(u, v)$ unless $d[u]$ has changed
- Stop if no $d[v]$ has changed for a full pass through $V$

**Theorem:**
- Throughout the algorithm $M[v]$ is the length of some $s - v$ path
- After $i$ passes through the nodes, $M[v] \leq OPT(v, i)$
Example

\[ \text{OPT}(v, i) = \min \left\{ \text{OPT}(v, o), \min_{(u,v) \in E} \text{OPT}(u, o) + l(u, v) \right\} \]

\[
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
s & 0 & 0 & 0 & 0 & 0 & 0 \\
b & \infty & -1 & & & & \\
c & \infty & 4 & & & & \\
d & \infty & \infty & & & & \\
e & \infty & \infty & & & & \\
\end{array}
\]

only \( d[b] \) changed
Implementation II

Efficient-Shortest-Path(G, s)

\[ \text{foreach node } v \in V \]
\[ D[v] \leftarrow \infty \]
\[ P[v] \leftarrow \bot \]
\[ D[s] \leftarrow 0 \]

\[ \text{for } i = 1 \text{ to } n \]
\[ \text{foreach node } u \in V \]
\[ \text{if (D[u] changed in the last iteration)} \]
\[ \text{foreach edge (u,v) } \in E \]
\[ \text{if (D[u] + } \ell_{uv} < D[v] \]
\[ D[v] \leftarrow D[u] + \ell_{uv} \]
\[ P[v] \leftarrow u \]
\[ \text{if (no D[u] changed): return (D,P)} \]

Running Time: \( O(nm) \) [but typically much faster]

Space: \( O(n+m) \)
Negative Cycle Detection

- **Claim 1:** if $OPT(v, n) = OPT(v, n - 1)$ then there are no negative cycles reachable from $s$

- **Claim 2:** if $OPT(v, n) < OPT(v, n - 1)$ then any shortest $s - v$ path contains a negative cycle
Negative Cycle Detection

- **Algorithm:**
  - Pick a node \( a \in V \)
  - Run Bellman-Ford for \( n \) iterations
  - Check if \( OPT(v, n) \neq OPT(v, n - 1) \) for some \( v \in V \)
    - If no, then there are no negative cycles
    - If yes, the shortest \( a - v \) path contains a negative cycle
Negative Cycle Detection

- **Algorithm:**
  - Add a new node \( s \in V \), add edges \((s, v)\) for every \( v \in V \)
  - Run Bellman-Ford for \( n \) iterations
  - Check if \( OPT(v, n) \neq OPT(v, n - 1) \) for some \( v \in V \)
    - If no, then there are no negative cycles
    - If yes, the shortest \( s \rightarrow v \) path contains a negative cycle
Shortest Paths Summary

- **Input:** Directed, weighted graph \( G = (V, E, \{\ell_e\}) \), source node \( s \)

- **Output:** Two arrays \( d, p \)
  - \( d[u] \) is the length of the shortest \( s \leadsto u \) path
  - \( p[u] \) is the final hop on shortest \( s \leadsto u \) path

- **Non-negative lengths:** Dijkstra’s Algorithm solves in \( O(m \log n) \) time

- **Negative lengths:** Bellman-Ford solves in \( O(nm) \) time \( O(n + m) \) space, or finds a negative cycle