Lecture 15:
• Bellman-Ford

Oct 30, 2018

Schedule
• Midterm II Nov 16
• No class Thanksgiving week
Dijkstra Recap

• **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

• **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 but no neg. length cycles.

1. Negative-length cycles — shortest path is undefined
   \[ \begin{array}{c}
   S \\
   2 \\
   3 \\
   \end{array} \quad \begin{array}{c}
   1 \\
   \text{distances can be } -\infty
   \end{array} \]

2. Wrong distances
   \[ \begin{array}{c}
   S \\
   3 \\
   \text{Wrong distances}
   \end{array} \quad \begin{array}{c}
   10 \\
   \end{array} \]

Running Dijkstra gives
\[ \begin{array}{cccc}
S & 1 & 2 & 3 \\
0 & \infty & \infty & \infty \\
0 & 3 & 10 & \infty \\
0 & 3 & 10 & 7 \\
0 & 0 & 10 & 7 \\
\end{array} \]
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 \( s \sim u \) path
  - \( p[u] \) is the final hop on shortest \( s \sim u \) 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?
• 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?
Structure of Shortest Paths

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

\[
\begin{array}{c}
\text{s} \\
\text{d}(s, u) \quad \text{u} \\
\text{v}
\end{array}
\]

\(\text{s} \rightarrow \text{v}\) path \(P\)

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

\[
\begin{array}{c}
\text{s} \\
\text{u} \\
\text{v}
\end{array}
\]

\(\text{s} \rightarrow \text{v}\) shortest path \(P^*\)

• 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

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

  - For every $v$, the shortest path makes some final hop $(u, v)$
  - Case $u$ (the final hop is $(u, v) \in E$)

    \[ \Rightarrow \quad OPT(v) = OPT(u) + l(u, v) \]

  - Recurrence:

    \[
    OPT(v) = \min_{(u, v) \in E} \{ OPT(u) + l(u, v) \}
    \]

    \[ OPT(s) = 0 \]
Bottom-Up Implementation?

\[ \text{OPT}(v) = \min_{(u,v) \in E} \text{OPT}(u) + \ell(u,v) \]

Need to fill the DP table in some order.

Cannot put the DP table “in order.”

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

\[ \exists \text{ a shortest path } u \} \leq n-1 \text{ hops} \]

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

- \( 0 \leq j \leq n-1 \)
- Any path with \( \geq n \) hops has a cycle, any cycle has \( \geq 0 \) length
  - \( \therefore \) shortest path has \( \leq n-1 \) hops

![Diagram of a graph with nodes s, a, b, and v, and an edge from s to b, and from b to v.](image)

- Could get a shorter path by eliminating the cycle.
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.

- Shortest path using $j$ hops uses some $(u, v)$ as the final hop.

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

\[
\text{OPT}(s, 0) = 0
\]

\[
\text{OPT}(v, 0) = \infty \quad \forall v \neq s
\]
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:

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

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

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

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

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

Recurrence:

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

If \( \text{OPT}(v_j, j) = \text{OPT}(u, j-1) + \ell(u,v) \) then there \( \exists \) a shortest \( s \leadsto v \) path \( u \) of \( \leq j \) hops, whose last hop is \( (u,v) \)

\[ \Rightarrow P(v_j, j) = u \]
Example

\[
\text{OPT}(v, j) = \min \left\{ \begin{array}{l}
\text{OPT}(v, j-1), \\
\min_{(u, v) \in E} \text{OPT}(u, j-1) + l_{u,v}
\end{array} \right\}
\]

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

\[
\text{OPT}(v, j) = \min \{ \text{OPT}(v, j-1) \leftarrow (u, v) \in E, \min \{ \text{OPT}(u, j-1) + f_{u,v} \} \}
\]
Example

```
<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:
- Nodes: s, b, c, d, e
- Edges with weights:
  - s to b: 0
  - s to c: 4
  - b to c: 3
  - b to d: -1
  - b to e: 2
  - c to b: 1
  - c to d: 2
  - c to e: 5
  - d to b: -2
  - d to e: -3
  - e to b: 1

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>
Example

The diagram shows a network of nodes (s, b, c, d, e) with directed edges and weights. The table below represents the transitions 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>-1</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>∞</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>e</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The diagram indicates that nothing changed.
Implementation (Bottom Up DP)

Shortest-Path(G, s)

\[
\begin{align*}
&\text{foreach node } v \in V \\
&D[v,0] \leftarrow \infty \\
&P[v,0] \leftarrow \bot \\
&D[s,0] \leftarrow 0 \\
\end{align*}
\]

for \( i = 1 \) to \( n-1 \) 

\[
\begin{align*}
&\text{foreach node } v \in V \\
&D[v,i] \leftarrow D[v,i-1] \\
&P[v,i] \leftarrow P[v,i-1] \\
&\text{foreach edge } (u,v) \in E \\
&\quad \text{if } (D[u,i-1] + l_{uv} < D[v,i]) \\
&\quad \quad D[v,i] \leftarrow D[u,i-1] + l_{uv} \\
&\quad \quad P[v,i] \leftarrow u \\
\end{align*}
\]

Running Time

\[ O(n^3) \text{ time} \]

\[ \leq n-1 \text{ iterations} \]

Total Time = \( O(nm) \)

Total Space = \( O(n^2) \)

\[ \sum_{r \in V} O(m \deg(r)) = O(m) \]
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 $s \rightarrow v$ path
  • After $i$ passes through the nodes, $M[v] \leq OPT(v, i)$
Efficient-Shortest-Path(G, s)

foreach node v ∈ V

\[ D[v] \leftarrow \infty \]
\[ P[v] \leftarrow \perp \text{ (null)} \]
\[ D[s] \leftarrow 0 \]

for i = 1 to n

foreach node u ∈ V

if (D[u] changed in the last iteration)

foreach edge (u,v) ∈ E

if (D[u] + \ell_{uv} < D[v])

\[ D[v] \leftarrow D[u] + \ell_{uv} \]
\[ P[v] \leftarrow u \]

if (no D[u] changed): return (D,P)

Running Time \( O(mn) \), but \( O(m) \) in practice

Space \( O(n) \)
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 $s \rightarrow v$ path contains a negative cycle

\[ \text{shortest } s \rightarrow v \text{ path uses exactly } n \text{ hops, and thus contains a cycle} \]

this cycle must have negative length!
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 \( a \rightarrow v \) 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