Lecture 15:
• Bellman-Ford Shortest Paths
• Negative Cycle Detection
• All pairs shortest paths (Floyd-Warshall)

Mar 2, 2018
Shortest Paths with Negative Edges
Dijkstra Recap

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

• Output:
  • Arrays $d, p$
    • $d(\nu)$ is the length of the shortest $s - \nu$ path
    • $p(\nu)$ is the final hop on the shortest $s - \nu$ path

• Running time $O(m \log n)$ (Implement using heaps)
\[ d(1) = 1 \quad p(1) = s \]
\[ d(2) = 3 \quad p(2) = 1 \]
\[ d(3) = 4 \quad p(3) = 2 \]
\[ d(4) = 7 \quad p(4) = 1 \]

- Red edges are all edges \((p(v), v)\) \(v \in V\)
- Red edges form a tree
- The unique \(s \to v\) path using red edges a shortest \(s \to v\) path in \(G_c\).
Ask the Audience

• Does Dijkstra’s algorithm still solve shortest paths in graphs with negative edge lengths?

- Negative Cycle
  - go around
  - cycle $\infty$ times the length of the path is $-\infty$
Ask the Audience

• Does Dijkstra’s algorithm still solve shortest paths in graphs with negative edge lengths?

... but no negative-length cycles

Dijkstra would explore u first but would not have found the shortest path from s to u.
Why Care About Negative Lengths?

Suppose you're arbitraging cryptocurrencies.

\[ \log_2(100) \rightarrow \text{Bitcoin} \rightarrow \log_2(20) \rightarrow \text{Ethereum} \]

\[ \frac{1}{110} \rightarrow \text{Joncoin} \rightarrow \log_2\left(\frac{1}{900}\right) \]

\[ \text{Ripple} \rightarrow \text{Zerocon} \]

Negative weight cycle represents an arbitrage opportunity.
Algorithms for shortest paths with negative edge weights are "robust."

- Routing algorithms that have to handle changing graphs.
Shortest Paths with Negative Lengths

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

• **Output:**
  - Arrays $d, p$
    - $d(v)$ is the length of the shortest $s \rightarrow v$ path
    - $p(v)$ is the final hop on the shortest $s \rightarrow v$ path
Minimum Cycle Detection

• **Input:**
  • Directed, graph $G = (V, E, \{\ell_e\})$
    • Possibly negative edge lengths $\ell_e$

• **Output:**
  • A negative-length cycle $C$ if one exists
Ask the Audience

• \( G = (V, E, \{\ell_e\}) \) is a graph with negative lengths
• \( G' \) is the same but we add \( (\min \ell_e) \) to every length
• Why doesn’t it work to run Dijkstra on \( G' \)?
**Structure of Shortest Paths**

![Graph](image)

**Fact:** If the shortest path from \( s \) to \( v \) passes through \( u \), then \( s \rightarrow u \rightarrow v = s \rightarrow u \rightarrow v \) where \( s \rightarrow u \) and \( u \rightarrow v \) are shortest paths.

\[
d(s, v) = d(s, u) + d(u, v)
\]

If \((u, v) \in E\), then \( d(s, v) \leq d(s, u) + l_{u, v} \)
Dynamic Programming

- Consider the shortest $s \rightarrow v$ path
- The path must be $s \rightarrow u \rightarrow v$ for some $u$
- If I knew $u$ then $d(s, v) = d(s, u) + l_{u, v}$
- If $OPT(v)$ = the length of the shortest $s \rightarrow v$ path

$$OPT(s) = 0$$
$$OPT(v) = \min_{u: (u, v) \in E} \{ OPT(u) + l_{u, v} \}$$
What Goes Wrong?

Base Case

In order to implement bottom-up DP there needs to be an ordering to fill the table

If each subproblem is \( \text{OPT}(v) \) and \( G \) has cycles then there is no ordering of the nodes.
Dynamic Programming Take II

- Consider the shortest $s \rightarrow v$ path
  - the last edge is some $(u, v)$
  - it makes $k$ hops for some $k$

\[ s \quad \text{k-1 hops} \quad u \quad 1 \text{ hop} \quad v \]

- Let $\text{OPT}(v, i)$ be the length of the shortest $s \rightarrow v$ path making $\leq i$ hops

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

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

- $OPT(v, i)$ is the length of the shortest path from $s$ to $v$ that uses at most $i$ hops.
- Want to compute $OPT(v, n - 1)$ for all $v$.
- $\forall i \; OPT(s, i) = 0$
- $\forall v \neq s \; OPT(v, 0) = \infty$

$OPT(v, i) = \min \left\{ OPT(v, i - 1), \min_{w \in V} \{ OPT(w, i - 1) + \ell_{w,v} \} \right\}$
Finding the paths

- $OPT(v, i)$ is the length of the shortest path from $s$ to $v$ that uses at most $i$ hops.
- $P(v, i)$ is the last hop on the shortest path from $s$ to $v$ that uses at most $i$ hops.

\[
OPT(v, i) = \min \left\{ OPT(v, i - 1), \min_{w \in V} \{ OPT(w, i - 1) + \ell_{w,v} \} \right\}
\]

- If $\min \Rightarrow P(v, i) \leftarrow P(v, i - 1)$
- If $\min$ for some $w$ then $P(v, i) \leftarrow w$
Example

Graph:
- Vertices: s, b, c, d, e
- Edges and Weights:
  - s to b: -1
  - b to d: 2
  - c to s: 3
  - c to b: 2
  - d to b: 1
  - d to e: -3
  - e to d: 5

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>∞</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>∞</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>∞</td>
<td>∞</td>
<td>1</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>
Implementation (Bottom Up)

\[ \text{Shortest-Path}(G, s) \]

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

\[ \text{for } i = 1 \text{ to } n-1 \]
\[ \text{foreach } \text{node } v \in V \]
\[ M[i,v] \leftarrow M[i-1,v] \]
\[ P[i,v] \leftarrow P[i-1,v] \]
\[ \text{foreach } \text{edge } (v, w) \in E \]
\[ \text{if } (M[i-1,w] + \ell_{vw} < M[i,v]) \]
\[ M[i,v] \leftarrow M[i-1,w] + \ell_{vw} \]
\[ P[i,v] \leftarrow w \]

\[ \text{main loop: } \sum_{v \in V} O(\text{deg}(v)) = O(m) \text{ per iteration} \]
\[ \times (n-1) \text{ iterations} \]
\[ = O(nm) \text{ time} \]
Optimizations

• One array $M[v]$ containing shortest $s - v$ path found so far
• No need to check edges $(w, v)$ unless $M[w]$ has changed
• Stop if no $M[w]$ 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, $M[v] \leq OPT(v, i)$
Efficient-Shortest-Path(G, s)

foreach node v ∈ V
  M[v] ← ∞
  P[v] ← φ
  M[s] ← 0

for i = 1 to n-1
  foreach node w ∈ V
    if (M[w] changed in the last iteration)
      foreach edge (w,v) ∈ E
        if (M[w] + ℓ_{wv} < M[v])
          M[v] ← M[w] + ℓ_{wv}
          P[v] ← w
    if (no M[w] changed): return M

Worst-case running time is O(nm)
In practice only make a small # of passes ⇒ O(m)
Summary

- Can solve shortest path w/ negative length (but no negative cycles) in $O(mn)$ time
  - Faster in practice

- Can implement in a distributed /asynchronous fashion
  - Roughly how routing tables are kept
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

If $OPT(v, n) = OPT(v, n - 1)$

Then $OPT(v, i) = OPT(v, n - 1)$ for $i \geq n - 1$
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

$C$ 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 $a - v$ 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 $s - v$ path contains a negative cycle

$O(nm)$ time to run Bellman-Ford
Negative Cycle Detection

• Claim 1: if $OPT(v, n) = OPT(v, n - 1)$ then there are no negative cycles

• Claim 2: if $OPT(v, n) < OPT(v, n - 1)$ then any shortest $s - v$ path contains a negative cycle
Summary

- Bellman-Ford finds shortest paths in $O(nm)$
- Can be modified to find negative cycles also in $O(nm)$. 
Implementation (Bottom Up)

**Shortest-Path (G)**

```plaintext
foreach pair of nodes i, j ∈ V
    if (i = j): M[i, j, 0] ← 0
    elseif ((i, j) ∈ E): M[i, j, 0] ← ℓ_{ij}
    else: M[i, j, 0] ← ∞

for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            M(i, j, k) ← \min\left\{ M(i, j, k − 1), \frac{M(i, j, k − 1) + M(k, j, k − 1)}{M(i, k, k − 1)} \right\}
```