Warm up:
Modify Dijkstra’s Algorithm to find the shortest paths by product of edge weights
(assume all weights are at least 1)

Dijkstra’s Algorithm
Initialize \(d_v = \infty\) for each node \(v\)
Keep a priority queue \(PQ\) of nodes, using \(d_v\) as key
Pick a start node \(s\), set \(d_s = 0\)
While \(PQ\) is not empty:
\[ v = PQ.\text{extractMin}(\)\]
for each \(u \in V\) s.t. \((v, u) \in E:\)
\[ PQ.\text{decreaseKey}(u, \min(d_v, d_u \cdot w(v, u))) \]

Modify Dijkstra’s Algorithm to find the shortest paths by product of edge weights
(assume all weights are at least 1)

Dijkstra’s Algorithm (for min product)
Initialize \(d_v = \infty\) for each node \(v\)
Keep a priority queue \(PQ\) of nodes, using \(d_v\) as key
Pick a start node \(s\), set \(d_s = 1\)
While \(PQ\) is not empty:
\[ v = PQ.\text{extractMin}(\)\]
for each \(u \in V\) s.t. \((v, u) \in E:\)
\[ PQ.\text{decreaseKey}(u, \min(d_v, d_u \cdot w(v, u))) \]

How do we know this works?
Shortest path by product

Goal: find the path \((s = v_1, v_2, ..., v_{k-1}, v_k)\)

which minimizes:

\[ w(v_1, v_2) \cdot w(v_2, v_3) \cdot \cdots \cdot w(v_{k-1}, v_k) \]

Observation: \(\log (x \cdot y) = \log x + \log y\)

\[ \log(w(v_1, v_2) \cdot w(v_2, v_3) \cdot \cdots \cdot w(v_{k-1}, v_k)) \]

\[ = \log w(v_1, v_2) + \log w(v_2, v_3) + \cdots + \log w(v_{k-1}, v_k) \]

New Goal: find the path \((s = v_1, v_2, ..., v_{k-1}, v_k)\) which minimizes:

\[ \log(w(v_1, v_2)) + \log(w(v_2, v_3)) + \cdots \]

Dijkstra’s Algorithm (for min product)

---

Dijkstra’s Algorithm (for min product)

Initialize \(d_v = \infty\) for each node \(v\)

Keep a priority queue \(PQ\) of nodes, using \(d_v\) as key

Pick a start node \(s\), set \(d_s = 0\)

While \(PQ\) is not empty:

\(v = PQ . \text{extractmin}()\)

for each \(u \in V \cdot (v, u) \in E:\)

\[ d_u = \min(d_u, d_v + \log(w(v, u))) \]

\(PQ . \text{decreaseKey}(u, \min(d_v, d_u + \log(w(v, u))))\)

---

Today’s Keywords

- Graphs
- Shortest path
- Bellman-Ford
- \(\text{OG DP}\)
- Floyd-Warshall
CLRS Readings

- Chapter 22
- Chapter 23
- Chapter 24

Homeworks

- HW7 Due **Tomorrow April 16 @11pm**
  - Written (use latex)
  - Graphs
- HW8 Released **Tomorrow April 16 @11:30pm**
  - Due **Tuesday April 23 @11pm**
  - Programming (Python or Java)
  - Graphs/Flow (hint, see Wednesday’s class)

Currency Exchange

<table>
<thead>
<tr>
<th>Currency code</th>
<th>Currency name</th>
<th>1 USD per 1 USD</th>
<th>1 USD per 1 Euro</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>US Dollar</td>
<td>1.00000000</td>
<td>1.00000000</td>
</tr>
<tr>
<td>BGN</td>
<td>Bulgarian Lev</td>
<td>0.67621207</td>
<td>1.48561780</td>
</tr>
<tr>
<td>INR</td>
<td>Indian Rupee</td>
<td>61.96000000</td>
<td>0.01613769</td>
</tr>
<tr>
<td>AUD</td>
<td>Australian Dollar</td>
<td>1.41050000</td>
<td>0.70924045</td>
</tr>
<tr>
<td>GBP</td>
<td>British Pound</td>
<td>0.82690000</td>
<td>1.20739890</td>
</tr>
<tr>
<td>HKD</td>
<td>Hong Kong Dollar</td>
<td>0.12950000</td>
<td>7.58335494</td>
</tr>
<tr>
<td>SGD</td>
<td>Singapore Dollar</td>
<td>1.40760000</td>
<td>0.71445113</td>
</tr>
<tr>
<td>BRL</td>
<td>Brazilian Real</td>
<td>0.34600000</td>
<td>2.90437900</td>
</tr>
<tr>
<td>JPY</td>
<td>Japanese Yen</td>
<td>119.07000000</td>
<td>0.00856505</td>
</tr>
<tr>
<td>DOP</td>
<td>Dominican Peso</td>
<td>48.94000000</td>
<td>0.01999949</td>
</tr>
<tr>
<td>NZD</td>
<td>New Zealand Dollar</td>
<td>0.68000000</td>
<td>0.02959270</td>
</tr>
<tr>
<td>TWD</td>
<td>Taiwan Dollar</td>
<td>30.50000000</td>
<td>0.04682943</td>
</tr>
<tr>
<td>HUF</td>
<td>Hungarian Forint</td>
<td>271.70000000</td>
<td>3.70271144</td>
</tr>
<tr>
<td>SGD</td>
<td>Singapore Dollar</td>
<td>1.40760000</td>
<td>0.71445113</td>
</tr>
<tr>
<td>HKD</td>
<td>Hong Kong Dollar</td>
<td>0.12950000</td>
<td>7.58335494</td>
</tr>
<tr>
<td>AUD</td>
<td>Australian Dollar</td>
<td>1.41050000</td>
<td>0.70924045</td>
</tr>
<tr>
<td>BGN</td>
<td>Bulgarian Lev</td>
<td>0.67621207</td>
<td>1.48561780</td>
</tr>
</tbody>
</table>

*1 Dollar = 0.8781211277 Euros*
Currency Exchange

1 Dollar = 4.1823100458 Dirham

1 Euro = 1.0548325619 Ringgit

1 Dollar = 0.8783121137 Euro

Directly: 1 Dollar = 3.87 Ringgit

Currency Exchange

1 Ringgit = 0.2583979328 Dollar

1 Dollar = 3.87479406049 Ringgit

1 Dollar = 3.87479406049 * 0.2583979328 Dollar = 1.00123877526 Dollar

Free Money!

Best way to transfer USD to MYR:

Given a graph of currencies (edges are exchange rates)

find the shortest path by product of edge weights

\[ \max \prod_{e \in E} \omega(e) \]

Invert edge weights to make it a minimization problem
Best Currency Exchange

Best way to transfer USD to MYR:
Given a graph of currencies (edges are exchange rates)
find the shortest path by product of edge weights

\[
\min \prod \frac{1}{w(x)} \quad \text{Take log of edge weights to make summation}
\]

<table>
<thead>
<tr>
<th>USD</th>
<th>Euro</th>
<th>AED</th>
<th>MYR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.14</td>
<td>0.24</td>
<td>3.87</td>
<td>0.26</td>
</tr>
</tbody>
</table>

\[\min \sum \log \frac{1}{w(x)} \quad \text{Now a shortest path problem!}\]

0.06 USD

- 0.63
- 0.57
- 0.585

Negative Edge Weights!

Problem with negative edges

There is no shortest path from A to I!

What we need: an algorithm that finds the shortest path in graphs with negative edge weights (if one exists)
**Note**

Any simple path has at most $V - 1$ edges

Pigeonhole Principle!

More than $V - 1$ edges means some node appears twice (i.e., there is a cycle)

If there is a shortest path of more than $V - 1$ edges, there is a negative weight cycle

---

**Bellman-Ford**

Idea: Use Dynamic Programming!

$\text{Short}(i, v)$ weight of the shortest path from $s$ to $v$ using at most $i$ edges

Two options:

- A path of $i - 1$ edges from $s$ to some node $x$, then edge $(x, v)$
- OR
- A path from $s$ to $v$ of at most $i - 1$ edges

$\text{Short}(i, v) = \min [\text{Short}(i - 1, x) + w(x, v), \text{Short}(i - 1, v)]$

---

**Bellman Ford**

Start node is $E$

Initialize all others to $\infty$

$\text{Short}(i, v)$ weight of the shortest path from $s$ to $v$ using at most $i$ edges

$\text{Short}(i, v) = \min [\text{Short}(i - 1, x) + w(x, v), \text{Short}(i - 1, v)]$
Start node is E
Initialize all others to \( \infty \)

\[
\begin{align*}
\text{Short}(0, \cdot) &= a \text{ to } \text{ using at most } 0 \text{ edges} \\
\text{Short}(1, \cdot) &= a \text{ to } \text{ using at most } 1 \text{ edges} \\
\text{Short}(2, \cdot) &= a \text{ to } \text{ using at most } 2 \text{ edges} \\
\text{Short}(3, \cdot) &= a \text{ to } \text{ using at most } 3 \text{ edges} \\
\text{Short}(4, \cdot) &= a \text{ to } \text{ using at most } 4 \text{ edges}
\end{align*}
\]
Bellman Ford

- Start node is E
- Initialize all others to $\infty$

Shortest path calculation:
- $\text{Short}(i, y) = \min\left(\text{Short}(i-1, y) + w(x, y)\right)$
- $\text{Short}(i, y)$ is the weight of the shortest path from E to y using at most i edges.

Bellman Ford: Negative cycles

- If we computed row V, values change.
- There is a negative weight cycle!

Bellman Ford Run Time

- Initialize array $\text{Short}[V][V]$ in $O(V^2)$
- Initialize $\text{Short}[0][v] = \infty$ for each vertex in $O(V)$
- Initialize $\text{Short}[0][s] = 0$ in $O(1)$
- For $i = 1, \ldots, V - 1$ in $O(V)$:
  - for each edge $(x, y) \in E$ in $O(E)$:
    - $\text{Short}[i][y] = \min\left(\text{Short}[i-1][x] + w(x, y), \text{Short}[i-1][y]\right)$ in $O(V^2)$

Time complexity:
- $\Theta(V^2 + EV)$
- $\Theta(EV)$
Why Use Bellman-Ford?

• Dijkstra's:
  – only works for positive edge weights
  – Run Time: $\Theta(E \log V)$
  – Not good for dynamic graphs (where edge weights are variable)
    • Must recalculate “from scratch”

• Bellman-Ford:
  – Works for negative edge weights
  – Run Time: $\Theta(E \cdot V)$
  – More efficient for dynamic graphs
    • $\Theta(C)$ time to recalculate

Bellman Ford: Dynamic

Each node will update its
eighbors if edge weight changes
$\text{Short}(i, j) =$ weight of the shortest path from $i$ to $j$ using at most $k$ edges

$\text{Prev}(x, y) =$ node \text{shortest path from $x$ to $y$ using at most $k$ edges}

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Bellman Ford: Dynamic
Bellman Ford: Dynamic

Each node will update its neighbors if edge weight changes.

\[ \text{Shortest path from } x \text{ to } y = \text{Shortest path from } x \text{ to } y \text{ using at most } i \text{ edges} \]

\[ \text{Shortest path from } x \text{ to } y = \min \{ \text{Shortest path from } x \text{ to } y \text{ using at most } i-1 \text{ edges}, \text{weight of edge from } x \text{ to } y + \text{Shortest path from } y \text{ to } y \text{ using at most } i-1 \text{ edges} \} \]

All-Pairs Shortest Path

Find the quickest way to get from each place to every other place.

Given a graph \( G = (V, E) \) for each start node \( s \in V \) and destination node \( t \in V \) find the least-weight path from \( s \rightarrow t \).

All-Pairs Shortest Path

- Can clearly be found in \( O(V^2 \cdot E) \)
  - Run Bellman-Ford with each node being the start

\[
\text{for each } s \in V, V \text{ times,} \\
\text{BellmanFord}(s) \quad O(V \cdot E)
\]
Floyd-Warshall

- Finds all-pairs shortest paths in \( \Theta(V^3) \)
- Uses Dynamic Programming

\[ \text{Shortest path from } i \text{ to } j \text{ includes } k \]

\[ \text{Shortest path from } i \text{ to } j \text{ excludes } k \]

\[ \text{Shortest } (i,j,k) = \begin{cases} 
\text{Shortest } (i,j,k - 1) + \text{Shortest } (k,j,k - 1) \\
\text{Shortest } (i,j,k - 1) 
\end{cases} \]

Shortest Paths Review

- Single Source Shortest Paths
  - Dijkstra's Algorithm \( \Theta(E \log V) \)
    - No negative edge weights
  - Bellman-Ford \( \Theta(VE) \)
    - First Dynamic Programming Algorithm
    - Allows negative edge weights (finds negative weight cycles)
    - Update memory in \( \Theta(V) \) time on edge weight updates

- All Pairs Shortest Paths
  - Floyd-Warshall \( \Theta(V^3) \)
    - Allows negative edge weights