ALL-PAIRS SHORTEST PATHS

\[
\begin{pmatrix}
a_{i,1} & \ldots & a_{i,n}
\end{pmatrix}
\times
\begin{pmatrix}
w_{1,k} \\
\vdots \\
w_{n,k}
\end{pmatrix}
=
\begin{pmatrix}
b_{i,k}
\end{pmatrix}
\]
Previous lecture: Single-Source Shortest Paths

Dijkstra’s algorithm
- works only for non-negative edge weights
- running time $\Theta(|V| \log |V| + |E|)$

Bellman-Ford algorithm
- can handle negative edge weights, works even for negative-weight cycles
- running time $\Theta(|V| \cdot |E|)$

Today: the All-Pairs Shortest-Path Problem
All-Pairs Shortest Paths

If we only have non-negative edge-weights:

Run Dijkstra’s algorithm |V| times, once with each vertex as source

- running time $\Theta ( |V|^2 \log |V| + |V| \cdot |E| )$

What to do if we also have negative edge weights?
adjacency-matrix representation

\[
M[i,j] = \begin{cases} 
0 & \text{if } i = j \\
 w(v_i,v_j) & \text{if } i \neq j \text{ and edge } (v_i,v_j) \text{ exists; } w(v_i,v_j) \text{ is weight of } (v_i,v_j) \\
\infty & \text{if edge } (v_i,v_j) \text{ does not exist}
\end{cases}
\]
Warm-up/Recap: Bellman-Ford via dynamic programming
5 steps in designing dynamic-programming algorithms

1. define subproblems

2. guess first choice

3. give recurrence for the value of an optimal solution
   - define subproblem in terms of a few parameters
   - define variable $m[..] = \text{value of optimal solution for subproblem}$
   - relate subproblems by giving recurrence for $m[..]$

4. algorithm: fill in table for $m[..]$ in suitable order (or recurse & memoize)

5. solve original problem

Running time: #subproblems * time/subproblem
Correctness: (i) correctness of recurrence: relate OPT to recurrence
(ii) correctness of algorithm: induction using (i)
Single-Source Shortest Paths: structure of optimal solution

$V = \{ v_1, v_2, \ldots, v_n \}$  $s = v_1$

Subproblem:
For each vertex $v_i$ compute shortest path from $s$ to $v_i$ with at most $m$ edges

Define variable:
$L(i,m)$ = length of shortest path from $s$ to $v_i$ with at most $m$ edges

Recursive formula:
$L(i,m) = \begin{cases} 
0 & \text{if } i = 1 \text{ and } m = 0 \\
\infty & \text{if } 1 < i \leq n - 1 \text{ and } m = 0 \\
\min \{ L(j,m-1) + w(v_j, v_i) \} & \text{if } 0 < m \leq n - 1
\end{cases}$

Note: here adjacency matrix, but Bellman-Ford uses adjacency list
Single-Source Shortest Paths: dynamic-programming solution

Recursive formula:

\[ L(i,m) = \begin{cases} 
0 & \text{if } i = 1 \text{ and } m = 0 \\
\infty & \text{if } 1 < i \leq n - 1 \text{ and } m = 0 \\
\min \{ L(j,m-1) + w(v_j, v_i) \} & \text{if } 0 < m \leq n - 1 
\end{cases} \]

Solution to the original problem is in the last column.
Single-Source Shortest Paths: dynamic-programming solution

Recursive formula:
\[
L(i,m) = \begin{cases} 
0 & \text{if } i = 1 \text{ and } m = 0 \\
\infty & \text{if } 1 < i \leq n - 1 \text{ and } m = 0 \\
\min \left\{ L(j,m-1) + w(v_j, v_i) \right\} & \text{if } 0 < m \leq n - 1 \\
1 \leq j \leq n
\end{cases}
\]

SSSP-DynamicProgramming ( G,s )

// G = (V,E) with V = \{ v_1, v_2, \ldots, v_n \} and s = v_1

1. \( L[1,0] \leftarrow 0 \)
2. for \( i \leftarrow 2 \) to \( n \) do \( L[i,0] \leftarrow \infty \)
3. for \( m \leftarrow 1 \) to \( n - 1 \)
4. do for \( i \leftarrow 1 \) to \( n \)
5. do \( L[i,m] \leftarrow \min \{ L(j,m-1) + w(v_j, v_i) \} \) for all \( 1 \leq j \leq n \)

after algorithm: \( L[i,n-1] = \delta (v_1,v_i) \) for all \( i \)

running time: \( O(n^3) \)
storage: \( O(n^2) \)

Note: Bellman-Ford actually runs in \( O(|V||E|) \) time using \( O(|V|) \) space.

How to check if a negative-weight cycle is reachable?
Now adapt to all-pairs shortest paths

\[ V = \{ v_1, v_2, \ldots, v_n \} \]

**Subproblem:**
For each pair \( v_i, v_k \) compute shortest path from \( v_i \) to \( v_k \) with at most \( m \) edges

**Define variable:**
\( L(i,k,m) = \text{length of shortest path from } v_i \text{ to } v_k \text{ with at most } m \text{ edges} \)

**Recursive formula:**
\[
L(i,k,m) = \begin{cases} 
0 & \text{if } i = k \text{ and } m = 0 \\
\infty & \text{if } i \neq k \text{ and } m = 0 \\
\min \{ L(i,j,m-1) + w(v_j, v_k) \} & \text{if } 0 < m \leq n - 1 \\
\end{cases}
\]
All-Pairs Shortest Paths: dynamic-programming solution

Recursive formula:

\[
L(i,k,m) = \begin{cases} 
0 & \text{if } i = k \text{ and } m = 0 \\
\infty & \text{if } i \neq k \text{ and } m = 0 \\
\min \{ L(i,j,m-1) + w(v_j, v_k) \} & \text{if } 0 < m \leq n - 1 \\
1 \leq j \leq n
\end{cases}
\]

**Slow-All-Pairs-Shortest-Paths (G,s)**

1. \textbf{for} \( i \leftarrow 1 \) \textbf{to} \( n \)
2. \hspace{1em} \textbf{do for} \( k \leftarrow 1 \) \textbf{to} \( n \)
3. \hspace{2em} \textbf{do} \( L[i,k,0] \leftarrow \infty \)
4. \hspace{1em} \textbf{for} \( i \leftarrow 1 \) \textbf{to} \( n \) \textbf{do} \( L[i,i,0] \leftarrow 0 \)
5. \hspace{1em} \textbf{for} \( m \leftarrow 1 \) \textbf{to} \( n - 1 \)
6. \hspace{2em} \textbf{do for} \( i \leftarrow 1 \) \textbf{to} \( n \)
7. \hspace{3em} \textbf{do for} \( k \leftarrow 1 \) \textbf{to} \( n \)
8. \hspace{4em} \textbf{do} \( L[i,k,m] \leftarrow \min \{ L(i,j,m-1) + w(v_j, v_k) \} \)

**running time:** \( O(n^4) \)
**storage:** \( O(n^3) \)

NB: still need to check for negative-weight cycles
All-Pairs Shortest Paths: slightly different formulation

New recursive formula: \( L(i,k,m) \) =

\[
\begin{cases} 
    w(v_j, v_k) & \text{if } m = 1 \\
    \min \{ L(i,j,m-1) + w(v_j, v_k) \} & \text{if } 1 < m \leq n - 1 \\
    1 \leq j \leq n 
\end{cases}
\]

Slow-All-Pairs-Shortest-Paths ( \( G,s \) )

1. \( L \leftarrow W \)  // \( W = \) matrix of edge weights: \( W[i,j] = w(v_j, v_j) \)
2. \( \text{for } m \leftarrow 2 \text{ to } n-1 \)
3. \( \text{do } L \leftarrow \text{Extend-Shortest-Paths} (L, W) \)

Extend-Shortest-Paths ( \( L,W \) )

1. \( \text{Let } B \text{ be a new } n \times n \text{ matrix} \)
2. \( \text{for } i \leftarrow 1 \text{ to } n \)
3. \( \text{do for } k \leftarrow 1 \text{ to } n \)
4. \( \text{do } B[i,k] \leftarrow \min \{ L(i,j) + w(v_j, v_k) \} \)
5. \( \text{return } B \)
All-Pairs Shortest Paths: relation to matrix multiplication

Recursive formula:

\[ L(i,k,m) = \begin{cases} 
    w(v_j, v_k) & \text{if } m = 1 \\
    \min \{ L(i,j,m-1) + w(v_j, v_k) \} & \text{if } 1 < m \leq n - 1 \\
    1 \leq j \leq n
\end{cases} \]

\[ A = \text{matrix with } a_{i,k} = L[i,k,m-1] = \text{min length of path with } m-1 \text{ edges from } v_i \text{ to } v_k \]

\[ W = \text{matrix such that } w_{i,k} = w(v_i, v_k) = \text{weight of edge } w(v_i, v_k) \]

\[ B = \text{matrix with } b_{i,k} = L[i,k,m] = \text{min length of path with } m \text{ edges from } v_i \text{ to } v_k \]

\[
\begin{pmatrix}
a_{i,1} & \ldots & a_{i,n}
\end{pmatrix}
\begin{pmatrix}
\vdots \\
w_{1,k} \\
w_{n,k}
\end{pmatrix}
= 
\begin{pmatrix}
\vdots \\
b_{i,k}
\end{pmatrix}
\]

\[ b_{i,k} = \min_{1 \leq j \leq n} ( a_{i,k} + w_{i,k} ) \]

replacing "min" by "\( \sum \)" and "+" by "\( \cdot \)" gives normal matrix multiplication
Let’s change notation for the matrices:

\[ D_{(m)} = \text{matrix with } d_{i,k} = L[i,k,m] = \text{min length of path with } m \text{ edges from } v_i \text{ to } v_k \]

\[ W = \text{matrix such that } w_{i,k} = w(v_i, v_k) = \text{weight of edge } w(v_i, v_k) \]

Then

\[ D_{(m)} = D_{(m-1)} \otimes W \]

\[ = (D_{(m-2)} \otimes W) \otimes W \]

\[ = D_{(1)} \otimes W \otimes W \otimes \ldots \otimes W \]

\[ = W^m \]

\( \otimes \) is associative
Conclusion

- Solving the All-Pairs Shortest Paths Problem is equivalent to computing the matrix product $W^{n-1}$ with the $\otimes$-multiplication operator.

- Extend-Shortest-Paths $(L, W)$ computes matrix “product” $L \otimes W$

We can use this insight to speed up the algorithm:

- repeated squaring: $W \rightarrow W^2 = W \otimes W \rightarrow W^4 = W^2 \otimes W^2 \rightarrow \ldots$

$W^m = W^{n-1}$ for all $m \geq n-1$ if there are no negative-weight cycles
New algorithm

Faster-All-Pairs-Shortest-Paths (W)

\[ W \] is matrix with edge weights

1. \( L \leftarrow W; \quad n \leftarrow \text{number of rows of } W \) \hspace{1cm} \( \text{// } n = |V| \)
2. \( m \leftarrow 1 \)
3. \textbf{while} \( m < n-1 \)
4. \hspace{1cm} \textbf{do} \hspace{0.5cm} \( \text{// Invariant: } L = W^m \)
5. \hspace{1cm} \( L \leftarrow \text{Extend-Shortest-Paths} \left( L, L \right) \) \hspace{1cm} \( \text{// } L \leftarrow L \otimes L \)
6. \hspace{1cm} \( m \leftarrow 2m \)
7. \hspace{1cm} \textbf{return} \( L \)

running time = \( O \left( n^3 \right) \times \text{number of times line 5 is executed} \)

= \( O \left( n^3 \log n \right) \)

… still need to check for negative-weight cycles.
Theorem: The All-Pairs Shortest-Paths problem for a graph $G=(V,E)$ with (possibly negative) edge weights can be solved in $O(|V|^3 \log |V|)$ time.

PS There is a different algorithm – the Floyd-Warshall algorithm – that is also based on dynamic programming, but that runs in $O(|V|^3)$ time.
An alternative approach: Johnson’s algorithm

Suppose we want to solve a certain graph problem
- we have an algorithm ALG that only works for certain types of graphs
- we want to develop an algorithm for another type of graph

Possible approaches

A. try to adapt ALG

B. try to modify the input graph $G$ into a different graph $\tilde{G}$ such that
   i. ALG can be applied to $\tilde{G}$
   ii. we can easily compute the solution for $G$ from the solution for $\tilde{G}$
Recall:

If we only have non-negative edge-weights, we can run Dijkstra’s algorithm $|V|$ times, once with each vertex as source

- running time $\Theta(|V|^2 \log |V| + |V| \cdot |E|)$

This is $O(|V|^3)$ in the worst case, so faster than previous approach.

Idea:

modify $G$ so that all weights become non-negative, then use approach above

- need to add something to the edge weights to make them non-negative
- but shortest paths should stay the same
$G = (V,E)$ weighted graph

$h: V \rightarrow \mathbb{R}$ function assigning a real number to each vertex $v$ in $V$

$G$ is the same graph as $G$, but with edge weights $w(u,v) = w(u,v) + h(u) - h(v)$

**Lemma**

(i) any shortest path in $G$ is a shortest path in $\overline{G}$, and vice versa

(ii) $G$ has negative-weight cycle iff $G$ has negative-weight cycle

**Proof.** Consider any path $v_1, v_2, \ldots, v_k$ in $G$

$$w(v_2, v_3) + h(v_2) - h(v_3)$$

$$w(v_{k-1}, v_k) + h(v_{k-1}) - h(v_k)$$

length of path in $\overline{G}$ = (length of path in $G$) + $h(v_1) - h(v_k)$. 

$$w(v_1, v_2) + h(v_1) - h(v_2)$$
G=(V,E) weighted graph

\( h: V \rightarrow \mathbb{R} \) function assigning a real number to each vertex \( v \) in \( V \)

\( G \) is same graph as \( G \), but with edge weights 
\[
\overline{w}(u,v) = w(u,v) + h(u) - h(v)
\]

**Lemma**
(i) any shortest path in \( G \) is a shortest path in \( \overline{G} \), and vice versa
(ii) \( G \) has negative-weight cycle iff \( \overline{G} \) has negative-weight cycle

need to find function \( h \) that makes all edge weights non-negative

**Observation**
- \( w(u,v) \) non-negative \( \iff \) \( h(v) - h(u) \leq w(u,v) \)
- Let \( s \) be any vertex that can reach \( u \). Then \( \delta(s,v) \leq \delta(s,u) + w(u,v) \)
need to find function $h$ that makes all edge weights non-negative

Observation

- $w(u,v)$ non-negative $\iff h(u) - h(v) \geq w(u,v)$
- Let $s$ be any vertex that can reach $u$. Then $\delta(s,v) \leq \delta(s,u) + w(u,v)$

Idea: compute $\delta(s,u)$ for all $u$, for suitable source $s$; set $h(u) = \delta(s,u)$

Add extra vertex $s$, with zero-weight edge to all other vertices.
Johnson \((G,w)\)

// Modify \(G\) into graph \(G^*\) by adding vertex \(s\) with edges to all other vertices
- \(V^* \leftarrow V \cup \{s\}\)
- \(E^* \leftarrow E \cup \{(s,u) : u \in V\};\) all new edges \((s,u)\) get weight 0

// Use \(G^*\) to transform \(G\) to graph \(G\) that has no negative edge weights
3. Run Bellman-Ford\((G^*,s)\) to compute distances \(\delta(s,u)\) for all \(u \in V\)
4. if Bellman-Ford reports that \(G^*\) has negative-weight cycle
5. then report that \(G\) has negative-weight cycle
6. else Let \(G=(V,E)\) be the same graph as \(G\),
   but with edge weights \(w(u,v) = w(u,v) + \delta(s,u) - \delta(s,v)\)
   \(\backslash\) and use \(G\) to solve the problem
7. for each vertex \(u \in V\) \(\backslash\) compute distances to all other vertices
8. do Run Dijkstra\((G,u)\) to compute distances \(\bar{\delta}(u,v)\)
9. For all \(v\), set \(\delta(u,v) \leftarrow \bar{\delta}(u,v) - \delta(s,u) + \delta(s,v)\)
Theorem: The All-Pairs Shortest-Paths problem for a graph $G=(V,E)$ with (possibly negative) edge weights can be solved in $O(|V|^2 \log |V| + |V| \cdot |E|)$ time.
All-Pairs Shortest Paths: Summary

Dynamic-programming algorithm
- running time $\Theta(\lvert V \rvert^4)$
- connection to matrix-multiplication
- improved version (repeated squaring) runs in $\Theta(\lvert V \rvert^3 \log \lvert V \rvert)$ time

Floyd-Warshall: different dynamic-programming algorithm running in $\Theta(\lvert V \rvert^3)$ and (also) very simple to implement

Johnson’s algorithm: reweighting
- modify graph to make all edge-weights non-negative
- then run Dijkstra’s algorithm $\lvert V \rvert$ times
- running time $\Theta(\lvert V \rvert^2 \log \lvert V \rvert + \lvert V \rvert \cdot \lvert E \rvert)$