We’ve done

- Dynamic Programming
  - Assembly-line scheduling
  - Optimal Binary Search Trees

Now

- All-Pairs Shortest-Paths

Next

- NP-Completeness
The All Pairs Shortest Paths Problem

Given a directed graph $G = (V, E)$ and a weight function $w : E \rightarrow \mathbb{R}$.

We assume (for now) there is no negative-weight cycle.

**Input:** a weight matrix $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 
    w(i,j) & \text{if } ij \in E \\
    0 & \text{if } i = j \\
    \infty & \text{otherwise}
\end{cases}$$

**Output:**

- a **distance matrix** $D = (d_{ij})$, where $d_{ij}$ is the weight of a shortest path from $i$ to $j$, and it is $\infty$ otherwise.

- a **predecessor matrix** $\Pi = (\pi_{ij})$, where $\pi_{ij}$ points to $j$’s previous vertex on a shortest path from $i$ to $j$, and NIL if $j$ is not reachable from $i$ or $j = i$. 
A Dynamic Programming Solution

- Let $d_{ij}^{(m)}$ denote the “length” (i.e. weight) of a shortest path from $i$ to $j$ with at most $m$ edges ($m \geq 1$)

- Let $D^{(m)} = (d_{ij}^{(m)})$ (a matrix)

- As there is no negative cycle, $D = D^{(n-1)}$ (why?)

- Also, $D^{(1)} = W$ (why?)

Key observation: a shortest path from $i$ to $j$ with at most $m$ edges

- **either** has $m - 1$ edges or less, in which case
  $$d_{ij}^{(m)} = d_{ij}^{(m-1)},$$

- **or** has exactly $m$ edges including some $kj$, in which case
  $$d_{ij}^{(m)} = d_{ik}^{(m-1)} + w_{kj}$$

Hence,

$$d_{ij}^{(m)} = \min_{k=1..n,k\neq j} \{d_{ij}^{(m-1)}, d_{ik}^{(m-1)} + w_{kj}\}$$

$$= \min_{k=1..n} \{d_{ik}^{(m-1)} + w_{kj}\}$$

(note $w_{jj} = 0.$)
**Pseudo Code**

We can use a 3-dimensional table to hold the variables $d_{ij}^{(m)}$, and fill the table out “layer” by “layer” starting with $m = 1$:

**APSP**($W, n$)

1: $D^{(1)} \leftarrow W$ // this actually takes $O(n^2)$

2: for $m \leftarrow 2$ to $n - 1$ do

3: for $i \leftarrow 1$ to $n$ do

4: for $j \leftarrow 1$ to $n$ do

5: $d_{ij}^{(m)} \leftarrow \min_{k=1}^{n} \{d_{ik}^{(m-1)} + w_{kj}\}$

6: end for

7: end for

8: end for

9: Return $D^{(n-1)}$ // the last “layer”

$O(n^4)$-time, $O(n^3)$-space.

Space can be reduced to $O(n^2)$ since one layer only depends on the previous. We only need to use two layers and use them alternatively.
More Observations

Ignoring the outer loop, replace min by $\sum$ and $+$ by $\cdot$, the previous code becomes

1:  for $i \leftarrow 1$ to $n$ do
2:     for $j \leftarrow 1$ to $n$ do
3:         $d_{ij}^{(m)} \leftarrow \sum_{k=1}^{n} d_{ik}^{(m-1)} \cdot w_{kj}$
4:     end for
5: end for

- This is like $D^{(m)} \leftarrow D^{(m-1)} \odot W$, where $\odot$ is identical to matrix multiplication, except that instead of $\sum$ we do min, and instead of $\cdot$ we do $+$
- $D^{(n-1)}$ is just $W \odot W \cdots \odot W$, $n - 1$ times.
- It is easy (?) to show that $\odot$ is associative
- Hence, $D^{(n-1)}$ can be calculated from $W$ in $O(\lg n)$ steps by “repeated squaring”, for a total running time of $O(n^3 \lg n)$

Lastly, the $\Pi$ matrix can be updated after each step of the algorithm
Floyd-Warshall Algorithm

The key:

Let \( d_{ij}^{(k)} \) be the length of a shortest path from \( i \) to \( j \), all of whose intermediate vertices are in the set

\[
[k] := \{1, \ldots, k\}. 0 \leq k \leq n
\]

We agree that \([0] = \emptyset\), so that \( d_{ij}^{(0)} \) is the length of a shortest path between \( i \) and \( j \) with no intermediate vertex.

Then, we get the following recurrence:

\[
d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min \left\{ \left( d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right), d_{ij}^{(k-1)} \right\} & \text{if } k \geq 1 \end{cases}
\]

The matrix we are looking for is \( D = D^{(n)} \).
Pseudo Code for Floyd-Warshall Algorithm

\textbf{FLOYD-WARSHALL}(W, n)

1: \( D^{(0)} \leftarrow W \)

2: \textbf{for} \( k \leftarrow 1 \) \textbf{to} \( n \) \textbf{do}

3: \textbf{for} \( i \leftarrow 1 \) \textbf{to} \( n \) \textbf{do}

4: \textbf{for} \( j \leftarrow 1 \) \textbf{to} \( n \) \textbf{do}

5: \( d_{i,j}^{(k)} \leftarrow \min\{(d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}), d_{i,j}^{(k-1)}\} \)

6: \textbf{end for}

7: \textbf{end for}

8: \textbf{end for}

9: \textbf{Return} \( D^n \) \text{ // the last “layer”} \\

Time: \( O(n^3) \), space: \( O(n^3) \).
Constructing the \( \Pi \) matrix

The \( \Pi \) matrix can also be updated on-the-fly with the following observation:

\[
\pi_{ij}^{(0)} = \begin{cases} 
\text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \\
i & \text{otherwise}
\end{cases}
\]

and for \( k \geq 1 \)

\[
\pi_{ij}^{(k)} = \begin{cases} 
\pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\
\pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}
\end{cases}
\]

Question: is it correct if we do

\[
\pi_{ij}^{(k)} = \begin{cases} 
\pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} < d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\
\pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \geq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}
\end{cases}
\]

Finally, \( \Pi = \Pi^{(n)} \).
Floyd-Warshall with less space

FLOYD-WARSHALL-2(\(W, n\))

1: \(D \leftarrow W\)
2: \(\text{for } k \leftarrow 1 \text{ to } n \text{ do}\)
3: \(\quad \text{for } i \leftarrow 1 \text{ to } n \text{ do}\)
4: \(\quad \quad \text{for } j \leftarrow 1 \text{ to } n \text{ do}\)
5: \(\quad \quad \quad d_{ij} \leftarrow \min\{(d_{ik} + d_{kj}), d_{ij}\}\)
6: \(\quad \text{end for}\)
7: \(\text{end for}\)
8: \(\text{end for}\)
9: Return \(D\)

Time: \(O(n^3)\), space: \(O(n^2)\).

Why does this work?
Transitive Closure of a Graph

- Given a directed graph $G = (V, E)$

- We’d like to find out whether there is a path between $i$ and $j$ for every pair $i, j$.

- $G^* = (V, E^*)$, the transitive closure of $G$, is defined by

$$ij \in E^* \text{ iff there is a path from } i \text{ to } j \text{ in } G.$$ 

- Given the adjacency matrix $A$ of $G$

  ($a_{ij} = 1$ if $ij \in E$, and 0 otherwise)

- Compute the adjacency matrix $A^*$ of $G^*$

Questions:

- What’s the first thing that comes to mind?

- What’s the second thing?
A DP Algorithm Based on Floyd Warshall

Let \( a_{ij}^{(k)} \) be a boolean variable, indicating whether there is a path from \( i \) to \( j \) all of whose intermediate vertices are in the set \([k]\).

We want \( A^* = A^{(n)} \).

Note that

\[
a_{ij}^{(0)} = \begin{cases} 
\text{TRUE} & \text{if } ij \in E \text{ or } i = j \\
\text{FALSE} & \text{otherwise}
\end{cases}
\]

and for \( k \geq 1 \)

\[
a_{ij}^{(k)} = a_{ij}^{(k-1)} \lor (a_{ik}^{(k-1)} \land a_{kj}^{(k-1)})
\]

Time: \( O(n^3) \), space \( O(n^3) \)

So what’s the advantage of doing this instead of Floyd-Warshall?