Single-source shortest path

- \( d(\ell, j) = \) Length of shortest path from \( s \) to \( j \) that uses at most \( \ell \) edges.
- \( d(0, j) = 0 \) if \( j = s \) and \( \infty \) otherwise.
- \( d(\ell, j) = \min \left\{ d(\ell - 1, j), \min_k \{ d(\ell - 1, k) + w_{kj} \} \right\} \)
This gives rise to an obvious DP algorithm:

\[
\begin{align*}
&\text{for } i = 1..n: \quad d[i] = \infty \\
&d[s] = 0 \\
&\text{for } \ell = 1..n - 1: \\
&\quad \text{for } j = 1..n: \\
&\quad \quad \text{for } k = 1..n: \\
&\quad \quad \quad d[j] = \min(d[j], d(k) + w_{kj})
\end{align*}
\]

Which runs in $\Theta(n^3)$ time.
Single-source shortest path

- Runtime can be improved by only looking at the \((j, k)\) pairs corresponding to edges; this makes \(\Theta(n|E|)\) runtime.
- Worse than Dijkstra, but works for negative-length edges.
- This is the Bellman-Ford algorithm (Chapter 24.1; the book explains it different and before APSP)
- (From the 1950’s ...)
All-pairs shortest path

- Given a directed graph with edge lengths $w_{i,j}$, for each ordered pair of vertices $(u, v)$, compute $\delta(u, v)$ (shortest path from $u$ to $v$).
- If edge lengths are nonnegative, can use Dijkstra’s algorithm $n$ times (treat each vertex as source).
- This costs $\Theta(n(m + n \log n))$ with best possible implementation of Dijkstra’s algorithm.
- What if we permit negative lengths?
If a graph has a negative cycle, then shortest paths are not well-defined.

The shortest path from \( u \) to \( v \) with at most one edge is the edge \((u, v)\) of length \( w_{u,v} \).

Define \( \text{distfirst}(u, v, k) \) to be the length of the shortest path from \( u \) to \( v \) with at most \( k \) edges.

Can we come up with a recurrence?
First try recurrence

\[
distfirst(u, v, k) = \begin{cases} 
  w_{u,v} & k = 1 \\
  \min_t \{distfirst(u, t, k - 1) + w_{t,v}\} & k > 1
\end{cases}
\]

- This works since optimal \( k \)-edge path contains an optimal \((k - 1)\)-edge path
- Answers are \( distfirst(u, v, n - 1) \)
- Order of computation is by increasing \( k \)
- Each entry takes \( \Theta(n) \) time to compute, and there are \( \Theta(n^3) \) entries
- Total running time is \( \Theta(n^4) \) - not very good
Second try: find middle

- A shortest $k$-edge path from $u$ to $v$ has some middle vertex $m$
- The sections of the paths from $u$ to $m$ and from $m$ to $v$ are $[k/2]$-edge shortest paths
- Define $\text{distmid}(u, v, j)$ to be the length of the shortest path from $u$ to $v$ with at most $2^j$ edges
- Can define $\text{distmid}(u, v, j)$ in terms of $\text{distmid}(\ast, \ast, j - 1)$
Second try recurrence

\[
distmid(u, v, j) = \begin{cases} 
w_{u,v} & j = 0 \\ 
\min_m \left\{ distmid(u, m, j - 1) + distmid(m, v, j - 1) \right\} & j > 0 
\end{cases}
\]

- Answers are \( distmid(u, v, \lfloor \log n \rfloor) \)
- Order of computation is by increasing \( j \)
- Each entry takes \( \Theta(n) \) time to compute, and there are \( \Theta(n^2 \log n) \) entries
- Total running time is \( \Theta(n^3 \log n) \) - better
Third try: add a vertex

- Use idea from Dijkstra (and Prim) of adding one vertex at a time to a set and maintaining shortest paths within that set.
- Consider a shortest path $P$ from $u$ to $v$ whose internal vertices are in the set $\{1, 2, \cdots, k\}$.
- If vertex $k$ is in the path, it splits $P$ into paths from $u$ to $k$ and from $k$ to $v$.
- Both of these have internal vertices from $\{1, 2, \cdots, k - 1\}$.
- Define $\text{distset}(u, v, k)$ to be the length of the shortest path $P$ mentioned above.
Third try recurrence

\[
distset(u, v, k) = \begin{cases} 
    w_{u,v} & k = 0 \\
    \min \left\{ \begin{array}{l}
    \distset(u, k, k - 1) + \distset(k, v, k - 1), \\
    \distset(u, v, k - 1)
    \end{array} \right\} & k > 0
\end{cases}
\]

- Answers are \(\distset(u, v, n)\)
- Order of computation is by increasing \(k\)
- Each entry takes \(\Theta(1)\) time to compute, and there are \(\Theta(n^3)\) entries
- Total running time is \(\Theta(n^3)\) - best
- Can be implemented in \(n^2\) space
Pseudocode for Floyd-Warshall

\[
D \leftarrow W \\
\text{for } k \leftarrow 1 \text{ to } n \text{ do} \\
\quad \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
\quad \quad \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
\quad \quad \quad D[i, j] = \min(D[i, j], D[i, k] + D[k, j])
\]