Dynamic Programming

- Paradigm class of algorithm that involves solving a problem through combining solutions to subproblems (as do greedy and divide-and-conquer paradigms).
- In dynamic programming, the subproblems overlap and solutions to inner problems are accumulated in a table, avoiding work of repeatedly solving innermost problems.
- Often applied to optimization problems (e.g., maximum or minimum solutions to a problem).

※ The problem must exhibit **OPTIMAL SUBSTRUCTURE**, meaning that the optimal solution to a problem contains optimal solutions to subproblems.

**Note:**

Greedy algorithms are similar, but solutions to subproblems don't affect how solution to subproblem is augmented to solution to full problem (greedy-choice property). In dynamic programming, the inner solution affects augmentation to outer solution.

Divide-and-conquer has a different partitioning into subproblems in which subproblems are disjoint.
Shortest path problems

Given: Weighted, directed graph $G = (V, E)$ with real-value weights $w$ on edges

Find: $\delta(u, v) =$ "shortest" path weight from $u$ to $v$ (lightest, actually)

$$\delta(u, v) = \begin{cases} \min \{ w(p) : u \xrightarrow{p} v \} & \text{if path exists} \\ \infty & \text{otherwise} \end{cases}$$

where $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$

for path $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$

---

Optimal substructure property

CLRS lemma 24.1 (Subpaths of shortest paths are shortest paths,)

Statement:
- $G = (V, E)$ is weighted, directed graph $w$/real-valued weights $w$ on edges
- $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path
- $p_{ij} = \langle v_i, v_i, v_{j}, \ldots, v_j \rangle$ is a subpath of $p$ for $0 \leq i \leq j \leq k$

$\Rightarrow$ $p_{ij}$ is a shortest path (from $v_i$ to $v_j$)

Proof: $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path

$w(p) = w(p_{i0}) + w(p_{ij}) + w(p_{jk})$

If alternate path $p_{ij}$ has $w(p_{i0}) < w(p_{ij})$, so $p_{ij}$ is not a shortest path, then $p_{ij}$ would not be a shortest path (cut-and-paste) $p' = \langle v_0, v_1, \ldots, v_i, v_j, v_{k} \rangle$; $w(p') = w(p_{i0}) + w(p_{ij}) + w(p_{jk}) < w(p)$.
In recitation saw Dijkstra's algorithm for finding all shortest paths from a single source, for graphs with nonnegative weights.

Today want to solve problem of finding the shortest path between all pairs of vertices. ("All-pairs shortest paths"). Applications: Routing tables for courier services, airlines, subway passengers, apps, internet, etc.

Could use Dijkstra's algorithm \( |V| \) times, iterating through selecting each vertex as the source.

Running time depends on data structure for min-priority queue (and would be \( |V| \cdot \text{Dijkstra} \))

- Linear array: \( O(V^3 + VE) = O(V^3) \)
- Binary heap: \( O(VE \log V) \)
- Fibonacci heap: \( O(V^2 \log V + VE) \)

For negative-weight edges, Dijkstra fails. Could use slower Bellman-Ford once per vertex:

\[ O(V^2E) \quad \text{dense graph} \quad O(V^4) \]

We'd better beat this! \( \uparrow \)
How should we frame and formulate all-pairs shortest paths as a problem we can solve?

**Representation**

Single-source shortest paths were represented on the actual graph, which is possible because subgraphs of shortest paths are also shortest paths and because each vertex only has one source. Difficult to conceive of graphical representation here.

Rather, use a distance matrix, \( D = (d_{ij}) \)
with \( d_{ij} = \text{weight of shortest path from vertex } i \text{ to vertex } j = \delta(i, j) \)

\( n \times n, n = |V| \)

Together with predecessor matrix, \( T = (T_{ij}) \)
with \( T_{ij} = \begin{cases} \text{NIL, for } i = j \text{ or no path } i \rightarrow j \\ \text{predecessor of } j \text{ on shortest path from } i, \text{ otherwise} \end{cases} \)

\( n \times n, n = |V| \)

**Distance Matrix**

\[
\begin{array}{cccc}
1 & 2 & 3 & \cdots & n \\
1 & d_{12} & d_{13} & \cdots & d_{1n} \\
2 & d_{21} & d_{22} & \cdots & d_{2n} \\
3 & d_{31} & d_{32} & \cdots & d_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
n & d_{n1} & d_{n2} & \cdots & d_{nn} \\
\end{array}
\]

**Predecessor Matrix**

\[
\begin{array}{cccc}
1 & 2 & 3 & \cdots & n \\
1 & T_{11} & T_{12} & \cdots & T_{1n} \\
2 & T_{21} & T_{22} & \cdots & T_{2n} \\
3 & T_{31} & T_{32} & \cdots & T_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
n & T_{n1} & T_{n2} & \cdots & T_{nn} \\
\end{array}
\]

Row \( i \) induces subgraph that is shortest paths free with root (source) at \( i \)

Shortest path distances from source \( = 1 \)

Not generally symmetric, \( d_{ij} \neq d_{ji} \) in general

\( \infty \) for non-existent edges

Not generally symmetric
Floyd-Warshall Algorithm (CLRS 25.2)

- Dynamic Programming
- $O(n^3)$
- Negative edges may be present, but assume no negative-weight cycles

Consider intermediate vertices of a shortest path

- An intermediate vertex of a simple path $P = \langle v_1, v_2, ..., v_e \rangle$ is any vertex on the path except $v_i$ or $v_e$. (Thus, a member of $\{v_2, v_3, ..., v_{e-1}\}$)

- A simple path has distinct vertices $\langle v_i, v_e, ..., v_e \rangle$
  - no cycles

Imagine we number our vertices arbitrarily $1 \ldots n$ so $V = \{1, 2, ..., n\}$
Consider a subset $\{1, 2, ..., k\}$ as the only allowed intermediate vertices, and for some $i, j \in V$ consider all paths $i \rightarrow j$ with intermediate vertices in $\{1, 2, ..., k\}$ and $P$ being the shortest simple path from that collection.
Then either

1. **k is not an intermediate vertex of p.**
   - So all intermediate vertices of p are in \( \{1, 2, \ldots, k-1\} \)
   - \( \Rightarrow \) A shortest path \( i \rightarrow j \) with all intermediate vertices in \( \{1, 2, \ldots, k-1\} \) is also a shortest path \( i \rightarrow j \) with all intermediate vertices in \( \{1, 2, \ldots, k\} \)

2. \( k \) is an intermediate vertex of p.
   - We can decompose p into
     \[ P_{ik} \rightarrow k \rightarrow P_{kj} \]
   - With \( P_{ik} \) shortest path \( i \rightarrow k \) and \( P_{kj} \) shortest path \( k \rightarrow j \), both with all intermediate vertices in \( \{1, 2, \ldots, k-1\} \) [By CLRS Lemma 24.1 - Subpaths of shortest paths are shortest paths]
Consider $d_{ij}^{(k)}$ to be the weight of the shortest path from $i$ to $j$ with all intermediate vertices in $\{1, 2, \ldots, k\}$.

Ex: $d_{ij}^{(0)}$ has zero intermediate vertices, so $d_{ij}^{(0)} = w_{ij}$

Then

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

defines recursive algorithm with $D^{(0)} = (d_{ij}^{(0)})$ giving all-pairs shortest path weights; $d_{ij}^{(n)} = \sum_{k=1}^{n} d_{ij}^{(k-1)}$ for all $i, j \in V$ because all intermediate vertices must be in $V = \{1, 2, \ldots, n\}$

Implementation

**Floyd-Warshall** $(W)$

1. Input $W$, rows
2. $D^{(0)} = W$
3. For $k = 1$ to $n$
   - Let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
     - For $i = 1$ to $n$
       - For $j = 1$ to $n$
         - $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$

Total: $\Theta(n^3) = \Theta(n^3)$
This provides only shortest-path distances, but not paths themselves.

→ Induction can prove correctness
→ Can get $T^*$ (predecessor matrix) multiple ways
  - Extract from $D = D^{(n)}$ in $O(n^3)$ ← exercise in CLRS
  - Compute $T^{(1)}, T^{(2)}, \ldots$ during Floyd-Warshall execution

\[
T^{(k)}_{ij} = \begin{cases} 
    \text{NIL} & \text{if } i = j \text{ or } W_{ij} = \infty \\
    i & \text{if } i \neq j \text{ and } W_{ij} < \infty
\end{cases}
\]

for $k \geq 1$ obtain $T^{(k)}_{ij}$ by

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

That is,

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