Lecture 15: The Floyd-Warshall Algorithm
CLRS section 25.2

Outline of this Lecture

- Recalling the all-pairs shortest path problem.
- Recalling the previous two solutions.
- The Floyd-Warshall Algorithm.
The All-Pairs Shortest Paths Problem

Given a weighted digraph $G = (V, E)$ with a weight function $w : E \rightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers, determine the length of the shortest path (i.e., distance) between all pairs of vertices in $G$. Here we assume that there are no cycle with zero or negative cost.

Without negative cost cycle

With negative cost cycle
Solutions Covered in the Previous Lecture

Solution 1: Assume no negative edges.
Run Dijkstra’s algorithm, \( n \) times, once with each vertex as source.
\( O(n^3 \log n). \ O(n^3) \) with more sophisticated data structures.

Solution 2: Assume no negative cycles.
Dynamic programming solution, based on a natural decomposition of the problem.
\( O(n^4). \ O(n^3 \log n) \) using “repeated squaring”.

This lecture: Assume no negative cycles.
develop another dynamic programming algorithm, the Floyd-Warshall algorithm, with time complexity \( O(n^3) \).
Also illustrates that there can be more than one way of developing a dynamic programming algorithm.
Solution 3: the Input and Output Format

As in the previous dynamic programming algorithm, we assume that the graph is represented by an $n \times n$ matrix with the weights of the edges:

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

Output Format: an $n \times n$ distance $D = [d_{ij}]$ where $d_{ij}$ is the distance from vertex $i$ to $j$. 
Step 1: The Floyd-Warshall Decomposition

**Definition:** The vertices $v_2, v_3, \ldots, v_{l-1}$ are called the *intermediate vertices* of the path $p = \langle v_1, v_2, \ldots, v_l \rangle$.

- Let $d_{ij}^{(k)}$ be the length of the shortest path from $i$ to $j$ such that all intermediate vertices on the path (if any) are in set $\{1, 2, \ldots, k\}$.

  $d_{ij}^{(0)}$ is set to be $w_{ij}$, i.e., no intermediate vertex.
  Let $D^{(k)}$ be the $n \times n$ matrix $[d_{ij}^{(k)}]$.

- Claim: $d_{ij}^{(n)}$ is the distance from $i$ to $j$. So our aim is to compute $D^{(n)}$.

- **Subproblems:** compute $D^{(k)}$ for $k = 0, 1, \ldots, n$. 

**Step 2: Structure of shortest paths**

**Observation 1:**
A shortest path does not contain the same vertex twice.
Proof: A path containing the same vertex twice contains a cycle. Removing cycle gives a shorter path.

**Observation 2:** For a shortest path from $i$ to $j$ such that any intermediate vertices on the path are chosen from the set $\{1, 2, \ldots, k\}$, there are two possibilities:

1. $k$ is not a vertex on the path,
The shortest such path has length $d_{ij}^{(k-1)}$.

2. $k$ is a vertex on the path.
The shortest such path has length $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. 
Consider a shortest path from $i$ to $j$ containing the vertex $k$. It consists of a subpath from $i$ to $k$ and a subpath from $k$ to $j$. Each subpath can only contain intermediate vertices in $\{1, \ldots, k - 1\}$, and must be as short as possible, namely they have lengths $d_{ik}^{(k-1)}$ and $d_{kj}^{(k-1)}$.

Hence the path has length $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Combining the two cases we get

$$d_{ij}^{(k)} = \min \left\{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\}.$$
Step 3: the Bottom-up Computation

• Bottom: \( D^{(0)} = [w_{ij}] \), the weight matrix.

• Compute \( D^{(k)} \) from \( D^{(k-1)} \) using

\[
d_{ij}^{(k)} = \min \left( d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right)
\]

for \( k = 1, \ldots, n \).
The Floyd-Warshall Algorithm: Version 1

Floyd-Warshall\((w, n)\)
\{
    for \(i = 1\) to \(n\) do
        \{ initialize \}
        for \(j = 1\) to \(n\) do
            \{ \(D^0[i, j] = w[i, j]\); \(\text{pred}[i, j] = \text{nil}\); \}

    for \(k = 1\) to \(n\) do
        \{ dynamic programming \}
        for \(i = 1\) to \(n\) do
            for \(j = 1\) to \(n\) do
                if \((d^{(k-1)}[i, k] + d^{(k-1)}[k, j] < d^{(k-1)}[i, j])\)
                    \{ \(d^{(k)}[i, j] = d^{(k-1)}[i, k] + d^{(k-1)}[k, j]\); \(\text{pred}[i, j] = k\); \}
                else \(d^{(k)}[i, j] = d^{(k-1)}[i, j]\);

        return \(d^{(n)}[1..n, 1..n]\);
    \}
\}
Comments on the Floyd-Warshall Algorithm

- The algorithm’s running time is clearly $\Theta(n^3)$.

- The predecessor pointer $\text{pred}[i, j]$ can be used to extract the final path (see later).

- Problem: the algorithm uses $\Theta(n^3)$ space. It is possible to reduce this down to $\Theta(n^2)$ space by keeping only one matrix instead of $n$. Algorithm is on next page. Convince yourself that it works.
The Floyd-Warshall Algorithm: Version 2

Floyd-Warshall\((w, n)\)
{ for \(i = 1\) to \(n\) do
  initialize
  for \(j = 1\) to \(n\) do
    \{ \(d[i, j] = w[i, j]; \) \(\) \(pred[i, j] = nil; \) \}

  for \(k = 1\) to \(n\) do dynamic programming
    for \(i = 1\) to \(n\) do
      for \(j = 1\) to \(n\) do
        if \((d[i, k] + d[k, j] < d[i, j])\)
          \{ \(d[i, j] = d[i, k] + d[k, j]; \) \(\) \(pred[i, j] = k;\) \}

      return \(d[1..n, 1..n];\) }
}
Extracting the Shortest Paths

The predecessor pointers $\text{pred}[i, j]$ can be used to extract the final path. The idea is as follows.

Whenever we discover that the shortest path from $i$ to $j$ passes through an intermediate vertex $k$, we set $\text{pred}[i, j] = k$.

If the shortest path does not pass through any intermediate vertex, then $\text{pred}[i, j] = \text{nil}$.

To find the shortest path from $i$ to $j$, we consult $\text{pred}[i, j]$. If it is nil, then the shortest path is just the edge $(i, j)$. Otherwise, we recursively compute the shortest path from $i$ to $\text{pred}[i, j]$ and the shortest path from $\text{pred}[i, j]$ to $j$. 
The Algorithm for Extracting the Shortest Paths

\[ \text{Path}(i, j) \]
\[
\begin{cases} 
\text{if } (\text{pred}[i, j] = \text{nil}) & \text{single edge} \\
\quad \text{output } (i, j); \\
\text{else} & \text{compute the two parts of the path} \\
\quad \{ \\
\quad \quad \text{Path}(i, \text{pred}[i, j]); \\
\quad \quad \text{Path}(\text{pred}[i, j], j); \\
\quad \} \\
\end{cases}
\]
Example of Extracting the Shortest Paths

Find the shortest path from vertex 2 to vertex 3.

2..3  Path(2, 3)   pred[2, 3] = 4
2..4..3 Path(2, 4)   pred[2, 4] = 5
2..5..4..3 Path(2, 5)   pred[2, 5] = nil   Output(2,5)
25..4..3 Path(5, 4)   pred[5, 4] = nil   Output(5,4)
254..3 Path(4, 3)   pred[4, 3] = 6
254..6..3 Path(4, 6)   pred[4, 6] = nil   Output(4,6)
2546..3 Path(6, 3)   pred[6, 3] = nil   Output(6,3)
25463