The weight of a directed i→j path (i, j ∈ V) is the sum of the weights of each of its directed edges.

Problem: (APS algorithm)
For each pair (i,j) ∈ V×V, determine an i→j path of minimum weight. (Also called a shortest path.)

Again there are really two problems:

1. Determine the minimum path weights for each (i,j)
2. Determine shortest i→j paths

We concentrate on the first problem, leaving the second as an exercise.

**Floyd-Warshall algorithm**

An intermediate vertex of a directed path P = (v1, v2, ..., vl) is any vertex other than v1 or vl, i.e., one of the vertices {v2, ..., vl-1}. 
Let $G = (V, E)$ be a directed graph with $V = \{1, 2, ..., n\}$. Define subsets $V_k$ of $V$ as follows:

$$V_k = \begin{cases} 
\emptyset & k = 0 \\
\{1, 2, ..., k\} & 1 \leq k \leq n 
\end{cases}$$

Let $(i, j) \in V \times V$ and $1 \leq k \leq n$. Let $P$ denote a minimum weight path among all $i$-$j$ paths with intermediate vertices in $V_k$.

Now observe that we have two alternatives:

- $k$ is not an intermediate vertex of $P$. In this case, $P$ is also of minimum weight among all $i$-$j$ paths with intermediate vertices in $V_{k-1}$.

- $k$ is an intermediate vertex of $P$. We can decompose $P$ into subpaths $P_1$ and $P_2$:

  $i \xrightarrow{P_1} k \xrightarrow{P_2} j$

Note vertex $k$ is not intermediate to either $P_1$ or $P_2$. 
Thus \( P_1 \) has minimum weight amongst all 1-\( k \) paths with intermediate vertices in \( V_{k-1} \), and likewise \( P_2 \) has minimum weight amongst all \( k-1 \) paths with intermediate vertices in \( V_{k-1} \).

These observations show ADSP exhibits optimal substructure, necessary for dynamic programming.

Let \( d_{ij}^{(k)} \) denote the weight of a minimum weight 1-\( i \) path with all intermediate vertices in \( V_k \).

When \( k=0 \), such a path has no intermediate vertices, hence at most one edge. Thus \( d_{ij}^{(0)} = w_{ij} \).

The above observations show that for \( 1 \leq k \leq n \) we have

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

Let \( D^{(k)} \) denote the matrix \( (d_{ij}^{(k)}) \). Then we seek \( D^{(n)} \) given \( D^{(0)} = W \).
Floyd-Warshall (W)

1. \( n \leftarrow \text{Row of } [W] \)
2. \( D^{(0)} \leftarrow W \)
3. \( \text{for } k \leftarrow 1 \text{ to } n \)
4. \( \text{for } i \leftarrow 1 \text{ to } n \)
5. \( \text{for } j \leftarrow 1 \text{ to } n \)
6. \( d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + c_{kj}) \)
7. \( \text{Return } D^{(n)} \)

Since (6) takes time \( O(n) \), Floyd-Warshall runs in time \( O(n^3) \).

Note: The above algorithm also uses memory \( n^3 \). It is possible to accomplish this with just \( n^2 \) memory (Exercise.)

To construct shortest paths we could use \( D = D^{(n)} \) to determine the predecessor matrix \( \Pi = (\pi_{ij}) \), where

\[
\pi_{ij} = \text{Predecessor of } i \text{ along a shortest } i-j \text{ path}
\]

Alternatively we could determine intermediate predecessor matrices \( \Pi^{(k)} = (\pi_{ij}^{(k)}) \) (\( 0 \leq k \leq n \))

\[
\pi_{ij}^{(k)} = \text{Predecessor of } i \text{ along a shortest } i-j \text{ path amongst those with intermediate vertices in } V_k.
\]
(See p.632 for details.)

Exercises

- Run Floyd-Warshall on the weighted digraph in preceding example.

- Write an algorithm to determine $\Pi$ from $D = D^{(n)}$

- Alter Floyd-Warshall to build $\Pi^{(n)}$ (0 ≤ k ≤ n) as you go.

- Write an algorithm to print a shortest i-j path given $\Pi = \Pi^{(n)}$.

Read

- Longest common subsequence (15.4)
- Optimal binary search tree (15.5)
Greedy Algorithms

Ex. Continuous Knapsack

As before we seek to

1. Maximize \( \sum_{i=1}^{n} x_i v_i \)

2. Subject to \( \sum_{i=1}^{n} x_i w_i \leq W \)

where \( W > 0, v_i > 0, \) and \( w_i > 0 \) (1 \( \leq i \leq n \)).

However instead of \( x_i \in \{0,1\} \) we now allow \( 0 \leq x_i \leq 1 \) for \( 1 \leq i \leq n \).

A vector \( x = (x_1, \ldots, x_n) \) will be called a feasible solution if (2) is satisfied without regard to the optimality condition (1).

A greedy strategy consists of making a locally optimal (greedy) choice, then solving the subproblem arising from this choice.

In this problem that means including the "best" object which does not exceed the capacity constraint, then doing the same thing with the remaining objects and remaining capacity.
KnapSack \((v, w, W)\)

1. \(n \leftarrow \text{length } [v]\)
2. \(\text{for } i \leftarrow 1 \text{ to } n\)
3. \(x[i] \leftarrow 0\)
4. \(\text{weight } \leftarrow 0\)
5. \(\text{while } \text{weight } < W\)
6. \(i \leftarrow \text{the "best" remaining object}\)
7. \(\text{if } \text{weight }+ w[i] \leq W\)
8. \(x[i] \leftarrow 1\)
9. \(\text{weight } \leftarrow \text{weight }+ w[i]\)
10. \(\text{mark } i \text{ as included }\)
11. \(\text{else}\)
12. \(x[i] \leftarrow \frac{W - \text{weight}}{w[i]}\)
13. \(\text{weight } \leftarrow W\)
14. \(\text{return } x\)

The "best" object in (6) can be interpreted in several ways.

In general we define a selection function \(f[i]\) which encodes the desirability of object \(i\). Line (6) then maximizes \(f[i]\) over all remaining (unmarked) objects.
Some Possible Choices for $f$ in This Example Are:

- $f(i) = v_i \quad \text{(Process ordered in order of decreasing values in Greedy loop.)}$
- $f(i) = \frac{1}{w_i} \quad \text{(Increasing weights.)}$
- $f(i) = \frac{v_i}{w_i} \quad \text{(Decreasing value-to-weight ratio.)}$

**Ex. $n=5$, $W=10$.**

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_i$</td>
<td>2</td>
<td>3</td>
<td>6.6</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$w_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$v_i/w_i$</td>
<td>2</td>
<td>1.5</td>
<td>2.2</td>
<td>1</td>
<td>1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$f(i)$</th>
<th>$x$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_i$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$1/w_i$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$v_i/w_i$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In this example (at least) $f(i) = v_i/w_i$ gives the best result. Is this optimum?