All-Pairs Shortest Paths

Problem: G is a weighted graph or digraph with n vertices, which for simplicity we label simply 1, 2, ..., n.

We are given the adjacency matrix \( W = (w_{ij}) \),
\( w_{ij} \) = weight of edge from i to j (\( \infty \) if no such edge, 0 if \( i = j \)). All weights are positive.

Find the distance matrix \( D = (d_{ij}) \),
\( d_{ij} \) = distance from i to j.

Idea: For \( k = 0, 1, ..., n \) let

\( \text{short}_k(i, j) = \text{shortest path from } i \text{ to } j \text{ all of whose intermediate vertices lie in the set } \{1, 2, ..., k\} \).

\( d_{ij}^k = \text{length of } \text{short}_k(i, j) \).

\( k = 0: \) \( \text{short}_0(i, j) = \text{edge from } i \text{ to } j \).
\( d_{ij}^0 = w_{ij} \).

\( k = n: \) \( \text{short}_n(i, j) = \text{shortest path from } i \text{ to } j \).
\( d_{ij}^n = d_{ij} \).

Initially, we know all the \( d_{ij}^0 \).

Our goal is to find all the \( d_{ij}^n \).

How can we find all of the \( d_{ij}^k \), assuming we already know the \( d_{ij}^{k-1} \)?

Case 1: \( k \) is not an intermediate vertex on \( \text{short}_k(i, j) \).

\( \text{short}_k(i, j) = \text{short}_{k-1}(i, j) \)
\( d_{ij}^k = d_{ij}^{k-1} \).

Vertex k doesn’t help. (Always the case if \( k = i \) or \( k = j \).)

Case 2: \( k \) is an intermediate vertex on \( \text{short}_k(i, j) \).

\( \text{short}_k(i, j) = \text{short}_{k-1}(i, k) + \text{short}_{k-1}(k, j) \).
\( d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1} \).
Which case applies, case 1 or case 2?

**Answer:** Whichever minimizes $d_{ij}^k$.

$$d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$$

If we let

$$p_{ij}^k = \begin{cases} 
  \text{true} & \text{if } d_{ik}^{k-1} + d_{kj}^{k-1} \text{ produces the minimum above,} \\
  \text{false} & \text{otherwise,} 
\end{cases}$$

then

$p_{ij}^k$ is true if and only if $k$ is an intermediate point of $\text{short}_k(i,j)$.

Rather than compute all the $p_{ij}^k$, the algorithm below computes $p_{ij} = \text{largest } k \text{ for which } p_{ij}^k \text{ is true, or } 0 \text{ if } p_{ij}^k \text{ is false for all } k$.

Note

$p_{ij} = \text{the highest-numbered intermediate point on the shortest path from } i \text{ to } j, \text{ or } 0 \text{ if there are no intermediate points.}$

We can compute all the $d_{ij}^k$ and $p_{ij}$ in $\Theta(n^3)$ time by:

```plaintext
for (i = 1, 2, ..., n) for (j = 1, 2, ..., n) 
  d_{ij}^0 = w_{ij}; 
  p_{ij} = 0; 
for (k = 1, 2, ..., n) for (i = 1, 2, ..., n) for (j = 1, 2, ..., n) 
  if (d_{ik}^{k-1} + d_{kj}^{k-1} < d_{ij}^{k-1}) 
    d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}; 
    p_{ij} = k; 
  else 
    d_{ij}^k = d_{ij}^{k-1};
```

The matrix $D = (d_{ij}^n)$ is the distance matrix, and the matrix $P = (p_{ij})$ has the information needed to find the shortest path between any pair of points.
Using the matrix $P$, we may print the shortest path from $i$ to $j$:

```plaintext
print(i);
print_intermediate_points(i, j);
print(j);

void print_intermediate_points(int i, int j)
    k = pij;
    if ( k > 0 )
        print_intermediate_points(i, k);
        print(k);
        print_intermediate_points(k, j);
    return;
```

Our algorithm for computing $D$ and $P$ uses $\Theta(n^3)$ space. We can reduce space (but not time) to $\Theta(n^2)$ by updating the $d_{ij}^k$ in place.

Consider a single pass through the outer loop (fixed $k$).

$d_{ij}$ doesn't change if $i=k$ or $j=k$, so there is no problem of using the new value of $d_{ik}$ or $d_{kj}$ when we need the old.

```plaintext
for ( i = 1,2,...,n )
    for ( j = 1,2,...,n )
        d_{ij} = w_{ij};
        pij = 0;
    for ( k = 1,2,...,n )
        for ( i = 1,2,...,n )
            for ( j = 1,2,...,n )
                if ( d_{ik} + d_{kj} < d_{ij} )
                    d_{ij} = d_{ik} + d_{kj};
                    pij = k;
```