All pairs shortest path through dynamic programming

- The all pairs shortest path problem
- Dynamic programming method
- Matrix product algorithm
- Floyd-Warshall algorithm

Reading: Weiss, Sections 7.5-7.7, CLRS chapter 15

All-pairs shortest paths

Recall the Shortest Path Problem.

Now we turn our attention to constructing a complete table of shortest distances, which must contain the shortest distance between any pair of vertices.

If the graph has no negative edge weights then we could simply make V runs of Dijkstra's algorithm, at a total cost of $O(VE \lg V)$, whereas if there are negative edge weights then we could make V runs of the Bellman-Ford algorithm at a total cost of $O(V^2E)$.

The two algorithms we shall examine both use the adjacency matrix representation of the graph, hence are most suitable for dense graphs. Recall that for a weighted graph the weighted adjacency matrix A has $\mathit{weight}(i, j)$ as its ij-entry, where $\mathit{weight}(i, j) = \infty$ if i and j are not adjacent.

A dynamic programming method

Dynamic programming is a general algorithmic technique for solving problems that can be characterised by two features:

- The problem is broken down into a collection of smaller subproblems
- The solution is built up from the stored values of the solutions to all of the subproblems

For the all-pairs shortest paths problem we define the simpler problem to be

"What is the length of the shortest path from vertex i to j that uses at most m edges?"

We shall solve this for $m = 1$, then use that solution to solve for $m = 2$, and so on ...

The initial step

We shall let $d_{ij}^{(m)}$ denote the distance from vertex i to vertex j along a path that uses at most m edges, and define $D^{(m)}$ to be the matrix whose ij-entry is the value $d_{ij}^{(m)}$.

As a shortest path between any two vertices can contain at most $V-1$ edges, the matrix $D^{(V-1)}$ contains the table of all-pairs shortest paths.

Our overall plan therefore is to use $D^{(1)}$ to compute $D^{(2)}$, then use $D^{(2)}$ to compute $D^{(3)}$ and so on.

The case $m = 1$

Now the matrix $D^{(1)}$ is easy to compute — the length of a shortest path using at most one edge from i to j is simply the weight of the edge from i to j. Therefore $D^{(1)}$ is just the adjacency matrix A.

The inductive step

What is the smallest weight of the path from vertex i to vertex j that uses at most m edges? Now a path using at most m edges can either be

1. A path using less than m edges

2. A path using exactly m edges, composed of a path using $m-1$ edges from i to an auxiliary vertex k and the edge (k, j).

We shall take the entry $d_{ij}^{(m)}$ to be the lowest weight path from the above choices.

Therefore we get

$$d_{ij}^{(m)} = \min\left(d_{ij}^{(m-1)}, \min_{1\leq k\leq V}\{d_{ik}^{(m-1)} + w(k,j)\}\right)$$

$$= \min_{1\leq k\leq V}\{d_{ik}^{(m-1)} + w(k,j)\}$$

Example

Consider the weighted graph with the following weighted adjacency matrix:

$$A = D^{(1)} = \begin{pmatrix} 0 & \infty & 11 & 2 & 6 \\ 1 & 0 & 4 & \infty & \infty \\ 10 & \infty & 0 & \infty & \infty \\ \infty & 2 & 6 & 0 & 3 \\ \infty & \infty & 6 & \infty & 0 \end{pmatrix}$$

Let us see how to compute an entry in $D^{(2)}$, suppose we are interested in the $(1, 3)$ entry:

- $1 \to 1 \to 3$ has cost $0 + 11 = 11$
- $1 \to 2 \to 3$ has cost $\infty + 4 = \infty$
- $1 \to 3 \to 3$ has cost $11 + 0 = 11$
- $1 \to 4 \to 3$ has cost $2 + 6 = 8$
- $1 \to 5 \to 3$ has cost $6 + 6 = 12$

The minimum of all of these is 8, hence the $(1, 3)$ entry of $D^{(2)}$ is set to 8.

Computing $D^{(2)}$

$$\begin{pmatrix} 0 & \infty & 11 & 2 & 6 \\ 1 & 0 & 4 & \infty & \infty \\ 10 & \infty & 0 & \infty & \infty \\ \infty & 2 & 6 & 0 & 3 \\ \infty & \infty & 6 & \infty & 0 \end{pmatrix} \begin{pmatrix} 0 & \infty & 11 & 2 & 6 \\ 1 & 0 & 4 & \infty & \infty \\ 10 & \infty & 0 & \infty & \infty \\ \infty & 2 & 6 & 0 & 3 \\ \infty & \infty & 6 & \infty & 0 \end{pmatrix} = \begin{pmatrix} 0 & 4 & 8 & 2 & 5 \\ 1 & 0 & 4 & 3 & 7 \\ 10 & \infty & 0 & 12 & 16 \\ 3 & 2 & 6 & 0 & 3 \\ 16 & \infty & 6 & \infty & 0 \end{pmatrix}$$

If we multiply two matrices $AB = C$, then we compute

$$c_{ij} = \sum_{k=1}^{k=V} a_{ik} b_{kj}$$

If we replace the multiplication $a_{ik}b_{kj}$ by addition $a_{ik} + b_{kj}$ and replace summation Σ by the minimum min then we get

$$c_{ij} = \min_{k=1}^{k=V} a_{ik} + b_{kj}$$

which is precisely the operation we are performing to calculate our matrices.

The remaining matrices

Proceeding to compute $D^{(3)}$ from $D^{(2)}$ and A, and then $D^{(4)}$ from $D^{(3)}$ and A we get:

$$D^{(3)} = \begin{pmatrix} 0 & 4 & 8 & 2 & 5 \\ 1 & 0 & 4 & 3 & \boxed{6} \\ 10 & \boxed{14} & 0 & 12 & \boxed{15} \\ 3 & 2 & 6 & 0 & 3 \\ 16 & \infty & 6 & \boxed{18} & 0 \end{pmatrix} \qquad D^{(4)} = \begin{pmatrix} 0 & 4 & 8 & 2 & 5 \\ 1 & 0 & 4 & 3 & 6 \\ 10 & 14 & 0 & 12 & 15 \\ 3 & 2 & 6 & 0 & 3 \\ 16 & \boxed{20} & 6 & 18 & 0 \end{pmatrix}$$

A new matrix "product"

Recall the method for computing $d_{ij}^{(m)}$, the (i, j) entry of the matrix $D^{(m)}$ using the method similar to matrix multiplication.

$d_{ij}^{(m)} \leftarrow \infty$
for $k = 1$ **to** V **do**
  $d_{ij}^{(m)} = \min(d_{ij}^{(m)}, d_{ik}^{(m-1)} + w(k, j))$
end for

Let us use $\star$ to denote this new matrix product.
Then we have

$$D^{(m)} = D^{(m-1)} \star A$$

Hence it is an easy matter to see that we can compute as follows:

$$D^{(2)} = A \star A \qquad D^{(3)} = D^{(2)} \star A \ldots$$

Complexity of this method

The time taken for this method is easily seen to be $O(V^4)$ as it performs V matrix "multiplications" each of which involves a triply nested **for** loop with each variable running from 1 to V.

However we can reduce the complexity of the algorithm by remembering that we do not need to compute *all* the intermediate products $D^{(1)}$, $D^{(2)}$ and so on, but we are only interested in $D^{(V-1)}$. Therefore we can simply compute:

$$D^{(2)} = A \star A$$
$$D^{(4)} = D^{(2)} \star D^{(2)}$$
$$D^{(8)} = D^{(4)} \star D^{(4)}$$

Therefore we only need to do this operation at most $\lg V$ times before we reach the matrix we want. The time required is therefore actually $O(V^3 \lceil \lg V \rceil)$.

Floyd-Warshall

The Floyd-Warshall algorithm uses a different dynamic programming formalism.

For this algorithm we shall define $d_{ij}^{(k)}$ to be the length of the shortest path from i to j whose intermediate vertices all lie in the set $\{1, \ldots, k\}$.

As before, we shall define $D^{(k)}$ to be the matrix whose (i, j) entry is $d_{ij}^{(k)}$.

The initial case

What is the matrix $D^{(0)}$ — the entry $d_{ij}^{(0)}$ is the length of the shortest path from i to j with *no* intermediate vertices. Therefore $D^{(0)}$ is simply the adjacency matrix A.

The inductive step

For the inductive step we assume that we have constructed already the matrix $D^{(k-1)}$ and wish to use it to construct the matrix $D^{(k)}$.

Let us consider all the paths from i to j whose intermediate vertices lie in $\{1, 2, \ldots, k\}$. There are two possibilities for such paths

(1) The path does not use vertex k

(2) The path does use vertex k

The shortest possible length of all the paths in category (1) is given by $d_{ij}^{(k-1)}$ which we already know.

If the path does use vertex k then it must go from vertex i to k and then proceed on to j, and the length of the shortest path in this category is $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

The overall algorithm

The overall algorithm is then simply a matter of running V times through a loop, with each entry being assigned as the minimum of two possibilities. Therefore the overall complexity of the algorithm is just $O(V^3)$.

$D^{(0)} \leftarrow A$
for $k = 1$ **to** V **do**
  for $i = 1$ **to** V **do**
    for $j = 1$ **to** V **do**
      $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
    end for j
  end for i
end for k

At the end of the procedure we have the matrix $D^{(V)}$ whose (i, j) entry contains the length of the shortest path from i to j, all of whose vertices lie in $\{1, 2, \ldots, V\}$ — in other words, the shortest path in total.

Example

Consider the weighted directed graph with the following adjacency matrix:

$$D^{(0)} = \begin{pmatrix} 0 & \infty & 11 & 2 & 6 \\ 1 & 0 & 4 & \infty & \infty \\ 10 & \infty & 0 & \infty & \infty \\ \infty & 2 & 6 & 0 & 3 \\ \infty & \infty & 6 & \infty & 0 \end{pmatrix} \qquad D^{(1)} = \begin{pmatrix} 0 & \infty & 11 & 2 & 6 \\ 1 & 0 & 4 & & \\ 10 & \infty & 0 & & \\ \infty & 2 & 6 & 0 & 3 \\ \infty & \infty & 6 & \infty & 0 \end{pmatrix}$$

To find the $(2, 4)$ entry of this matrix we have to consider the paths through the vertex 1 — is there a path from $2 - 1 - 4$ that has a better value than the current path? If so, then that entry is updated.

The entire sequence of matrices

$$D^{(2)} = \begin{pmatrix} 0 & \infty & 11 & 2 & 6 \\ 1 & 0 & 4 & \boxed{3} & \boxed{7} \\ 10 & \infty & 0 & \boxed{12} & \boxed{16} \\ \boxed{3} & 2 & 6 & 0 & 3 \\ \infty & \infty & 6 & \infty & 0 \end{pmatrix} \qquad D^{(3)} = \begin{pmatrix} 0 & \infty & 11 & 2 & 6 \\ 1 & 0 & 4 & 3 & 7 \\ 10 & \infty & 0 & 12 & 16 \\ 3 & 2 & 6 & 0 & 3 \\ \boxed{16} & \infty & 6 & \boxed{18} & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & \boxed{4} & \boxed{8} & 2 & \boxed{5} \\ 1 & 0 & 4 & 3 & \boxed{6} \\ 10 & \boxed{14} & 0 & 12 & \boxed{15} \\ 3 & 2 & 6 & 0 & 3 \\ 16 & \boxed{20} & 6 & 18 & 0 \end{pmatrix} \qquad D^{(5)} = \begin{pmatrix} 0 & 4 & 8 & 2 & 5 \\ 1 & 0 & 4 & 3 & 6 \\ 10 & 14 & 0 & 12 & 15 \\ 3 & 2 & 6 & 0 & 3 \\ 16 & 20 & 6 & 18 & 0 \end{pmatrix}$$

Finding the actual shortest paths

In both of these algorithms we have not addressed the question of actually finding the paths themselves.

For the Floyd-Warshall algorithm this is achieved by constructing a further sequence of arrays $P^{(k)}$ whose (i, j) entry contains a predecessor of j on the path from i to j. As the entries are updated the predecessors will change — if the matrix entry is not changed then the predecessor does not change, but if the entry does change, because the path originally from i to j becomes re-routed through the vertex k, then the predecessor of j becomes the predecessor of j on the path from k to j.