

## Chapter 25. All Pairs Shortest Paths

Given a directed, connected weighted graph  $G(V, E)$ , for each edge  $\langle u, v \rangle \in E$ , a weight  $w(u, v)$  is associated with the edge. The **all-pairs shortest-paths problem** (APSP) is to find a shortest path from  $u$  to  $v$  for every pair of vertices  $u$  and  $v$  in  $V$ .

Approaches to solving APSP:

- Run a single-source shortest paths algorithm starting at each vertex.
- The Floyd-Warsall algorithm, etc

## Chapter 25. Approaches to Solving All Pairs Shortest Paths

Single source shortest path method:

- For only positive edge lengths, use Dijkstra's algorithm starting at each vertex. The running time depends on which data structure used, e.g.,
  - linear array implementation of the priority queue, it takes  $O(n^3 + mn) = O(n^3)$ ;
  - binary-heap implementation of the priority queue, it takes  $O(mn \log n)$  time;
  - Fibonacci-heap implementation of priority queue, it takes  $O(n^2 \log n + mn)$  time
- If  $G$  is allowed to contain negative length edges, use the Bellman-Ford algorithm starting at each vertex. The running time is  $O(n^2m)$ .

## The “Repeated Squaring” method

This is an example of dynamic programming.

Let  $G = (V, E)$  be a directed graph with edge lengths. The lengths can be negative, but negative-length cycles are not allowed. Let  $n = |V|$ .

Let  $w(i, j)$  be the length of the edge  $\langle v_i, v_j \rangle$ , if any.

For vertices  $v_i, v_j \in V$ , and integer  $m \geq 1$ , define

$\ell(i, j)^{(m)}$  = the length of the shortest path from  $v_i$  to  $v_j$  that uses at most  $m$  edges.

Since no shortest path in  $G$  can use more than  $n - 1$  edges, we have

$\ell(i, j)^{(m)}$  = the length of the shortest path from  $v_i$  to  $v_j$ , if  $m \geq n - 1$ .

## The “Repeated Squaring” method (continued)

Initial value:

$$\ell(i, j)^{(1)} = \begin{cases} 0, & \text{if } i = j; \\ \infty, & \text{if } i \neq j \text{ and there is no edge } \langle v_i, v_j \rangle; \\ w(i, j), & \text{if } i \neq j \text{ and there is an edge } \langle v_i, v_j \rangle \end{cases}$$

Iteration:

Any path of at most  $2m$  edges from  $v_i$  to  $v_j$  consists of a path of at most  $m$  edges from  $v_i$  to  $v_k$  (for some  $k$ ) followed by a path of at most  $m$  edges from  $v_k$  to  $v_j$ .

Therefore, for any  $m \geq 1$ :

$$\ell(i, j)^{(2m)} = \min_{k=1}^n (\ell(i, k)^{(m)} + \ell(k, j)^{(m)}), \text{ for all } i, j.$$

## The “Repeated Squaring” method (continued)

**Computation:** Starting with the initial value, apply the recurrence repeatedly until  $m \geq n - 1$ .

That is, compute

$$l^{(1)}, l^{(2)}, l^{(4)}, l^{(8)}, \dots$$

until the superscript is  $\geq n - 1$ .

Each step takes  $n^3$  time (loops over  $i, j, k$ ), and  $O(\lg n)$  steps are needed, so the total running time is  $O(n^3 \lg n)$ .

## 25.2 The Floyd-Warshall algorithm

This uses dynamic programming in a different manner.

Let  $G = (V, E)$  be a directed graph with edge lengths. The lengths can be negative, but negative-length cycles are not allowed. Let  $n = |V|$ .

Let  $w(i, j)$  be the length of the edge  $\langle v_i, v_j \rangle$ , if any.

For vertices  $v_i, v_j \in V$ , and integer  $0 \leq k \leq n$ , define

$d_{ij}^{(m)}$  = the length of the shortest path from  $v_i$  to  $v_j$   
for which the intermediate vertices are in  $\{v_1, v_2, \dots, v_k\}$ .

Obviously, we have

$d_{ij}^{(n)}$  = the length of the shortest path from  $v_i$  to  $v_j$ .

## The Floyd-Warshall algorithm (continued)

Initial value (no intermediate vertices allowed):

$$d_{ij}^{(0)} = \begin{cases} 0, & \text{if } i = j; \\ \infty, & \text{if } i \neq j \text{ and there is no edge } \langle v_i, v_j \rangle; \\ w(i, j), & \text{if } i \neq j \text{ and there is an edge } \langle v_i, v_j \rangle \end{cases}$$

Iteration (allowing  $v_k$  as an intermediate vertex):

Any path from  $v_i$  to  $v_j$  that has intermediate vertices from  $\{v_1, v_2, \dots, v_k\}$  either actually has  $v_k$  as an intermediate vertex or it doesn't.

Therefore, for any  $k \geq 1$ :

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}, \text{ for all } i, j.$$

## The Floyd-Warshall algorithm (continued)

**Computation:** Starting with the initial value, apply the recurrence repeatedly until  $k = n$ .

That is, compute

$$d^{(0)}, d^{(1)}, d^{(2)}, \dots, d^{(n)}.$$

Each step takes  $n^2$  time (loops over  $i, j$ ), and  $n$  steps are needed, so the total running time is  $O(n^3)$ .

Actually we don't need to use a different matrix  $d^{(k)}$  for each  $k$ , since the algorithm works if the same matrix is used repeatedly. Let  $D = (d_{ij})$  be a matrix.

```
1    $d_{ij} \leftarrow d_{ij}^{(0)}$  for all  $i, j$ 
2   for  $k \leftarrow 1$  to  $n$ 
3       do for  $i \leftarrow 1$  to  $n$ 
4           do for  $j \leftarrow 1$  to  $n$ 
5               do  $d_{ij} \leftarrow \min\{d_{ij}, d_{ik} + d_{kj}\}$ 
6   return  $D$ 
```

## 25.2 Transitive closure of a directed graph

Given a directed graph  $G = (V, E)$ , the **transitive closure** of  $G$  is defined as the directed graph  $G^* = (V, E^*)$  where

$$E^* = \{ \langle i, j \rangle \mid \text{there is a path from } v_i \text{ to } v_j \text{ in } G \}.$$

This is similar to the shortest path problem except that we are interested only in the existence of a path and not how long it is.

We can use any APSP algorithm to solve it. We can also avoid integer arithmetic by using boolean operations.

## 25.2 Transitive closure of a directed graph (continued)

We use a matrix  $T = (t_{ij})$  containing 1 for a path existing and 0 for a path not existing.

Transitive\_Closure( $G$ )

```
1       $n \leftarrow |V|$ 
2      for  $i \leftarrow 1$  to  $n$ 
3          do for  $j \leftarrow 1$  to  $n$ 
4              do if  $i = j$  or  $\langle i, j \rangle \in E$ 
5                  then  $t_{ij} \leftarrow 1$  else  $t_{ij} \leftarrow 0$ 
6      for  $k \leftarrow 1$  to  $n$ 
7          do for  $i \leftarrow 1$  to  $n$ 
8              do for  $j \leftarrow 1$  to  $n$ 
9                   $t_{ij} \leftarrow t_{ij} \vee (t_{ik} \wedge t_{kj})$ 
10     return  $T$ 
```

## Transitive closure of a directed graph (continued)

Another method for transitive closure uses matrix multiplication.

Let  $A$  be the adjacency matrix of digraph  $G$  and let  $I$  be the identity matrix of the same size.

Then, if  $m \geq n - 1$ , the transitive closure of  $G$  has edges wherever the entries of  $(I + A)^m$  are non-zero. Start with  $I + A$  and square repeatedly to get sufficiently large  $m$ .

This takes time  $O(n^3 \lg n)$  if the standard matrix multiplication method is used. However, there are faster matrix multiplication methods that make this method faster than others.