Chapter 25. All Pairs Shortest Paths

We are given a directed, connected, edge-weighted graph $G = (V, E)$ with length $w(u, v)$ for each edge $(u, v) \in E$.

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 $v \in V$.
- Use the Floyd-Warshall algorithm or other algorithms (matrix-based methods), which will be introduced in this lecture.

Chapter 25. Approaches to Solving All Pairs Shortest Paths

The single source shortest paths method:

- For a graph contains only non-negative edge lengths, use Dijkstra's algorithm starting at each vertex and treat that vertex as the source vertex. The running time of the algorithm depends on which data structure used.
  - Linear array implementation:
    $O(|V|^3 + |V||E|) = O(|V|^3)$.
  - MIN-HEAP (the priority queue) implementation:
    $O(|V|^2 \log |V| + |V||E| \log |V|) = O(|V||E| \log |V|)$.
- If G contains negative edge lengths, use the Bellman-Ford algorithm starting at each vertex $v \in V$. The running time of the algorithm is $O(|V|^2|E|)$.

Chapter 25. Approaches to Solving All Pairs Shortest Paths

(I. The Matrix 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** (WHY?).

Let $n = |V|$. Let $w(i, j)$ be the length of edge (v_i, v_j) 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 uses more than $n - 1$ edges, we have

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

II. The "Repeated Squaring" method (continued)

Initial value:

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

where $w(i,j)$ is the weight (length) of edge (v_i, v_j) from v_i to v_j.

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

$$\ell^{(1)}, \ell^{(2)}, \ell^{(4)}, \ell^{(8)}, \ldots, \ell^{2^{\lceil \log(n-1) \rceil}}$$

until the superscript is no less than $n-1$.

Each iteration takes $O(n^3)$ time (loops over i, j, k), and $O(\log n)$ iterations (WHY?) are needed. So, the total running time of the repeated squaring method is $O(n^3 \log n)$,

since $\ell^{2^{\lceil \log(n-1) \rceil}} = \ell^{2^{\lceil \log(n-1) \rceil + 1}} = \ell^{2^{\lceil \log(n-1) \rceil + 2}} = \ldots = \ell^{2^{\lceil \log(n-1) \rceil + k}}$ for any integer $k \geq 0$.

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.

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

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

Obviously, we have

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

The Floyd-Warshall algorithm (continued)

Initial value (no intermediate vertices allowed):

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

Iteration (allowing v_k as an intermediate vertex):

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

Therefore, for any $k \geq 1$:

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

The Floyd-Warshall algorithm (continued)

Computation: Starting with the initial value $k = 0$, apply the recurrence repeatedly until $k = n$. That is, compute $d^{(0)}, d^{(1)}, d^{(2)}, \ldots, d^{(n)}$. The total running time is $O(|V|^3)$.

Floyd-Warshall(W)

1 $n \leftarrow W.rows$;
2 $D^{(0)} \leftarrow W$;
3 **for** $k \leftarrow 1$ **to** n **do**
4 let $D^{(k)} \leftarrow \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
5 **for** $i \leftarrow 1$ **to** n **do**
6 **for** $j \leftarrow 1$ **to** n **do**
7 $d_{ij}^{(k)} \leftarrow \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$;
8 return $D^{(n)}$.

The Floyd-Warshall algorithm (continued)

The input of Procedure `Floyd-Warshall` is a matrix with an entry $d_{ij}^{(0)}$, and **its output** is $D^{(n)} = \left(d_{ij}^{(n)}\right)$, a matrix with an entry $d_{ij}^{(n)}$, the distance of vertex v_j from vertex v_i.

Note that we don't need to use a different matrix $D^{(k)}$ for each k when constructing $D^{(k)}$, we only use the values from $D^{(k-1)}$. So, two matrices are enough.

How to save even more space?

```
1   d_ij ← d_ij^(0) for all i, j
2   for k ← 1 to n do
4       for i ← 1 to n do
4           for j ← 1 to n do
5               d_ij ← 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

$$E^* = \{(v_i, v_j) \mid \text{there is a directed path in } G \text{ from } v_i \text{ to } v_j\}.$$

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 (continued)

We use a matrix $T = (t_{ij})$ containing 'true' for a path existing, and 'false' for a path not existing.

```
Transitive_Closure(G)
 1   n ← |V|;
 2   for i ← 1 to n do
 3       for j ← 1 to n do
 4           if (i = j or (v_i, v_j) ∈ E)
 5               then t_ij ←' true'
 6               else t_ij ←' false';
 7   for k ← 1 to n do
 8       for i ← 1 to n do
 9           for j ← 1 to n do
10               t_ij ← t_ij ∨ (t_ik ∧ t_kj);
11   return T.
```

Exercise questions related to the Floyd-Warshall algorithm

Exercise A: Based on the Floyd-Warshall algorithm, write a procedure that constructs a shortest path from v_i to v_j for each pair of vertices v_i and v_j.

Exercise B: How can we use the output of the Floyd-Warshall algorithm to detect the presence of a negative-weight cycle?