Shortest Paths Problems

Input: a directed graph $G = (V, E)$ and a **weight** function $w : E \to R$.

The weight of a path $p = v_0, v_1, v_2, ..., v_k$ is

$$w(p) \;=\; \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

The weight of the **shortest path** from u to v, $\delta(u, v)$ is the minimum of $w(p)$ for all p connecting u to v, and ∞ if there is no such path in G.

All-Pairs Shortest Paths

Single-Source Shortest Paths:
Compute shortest paths from a given source to all vertices in the graph.

All-Pairs Shortest Paths:
Given a graph $G = (V, E)$, $|V| = n$, and a weight function w on the edges, compute the shortest paths between all pairs of vertices.

The problem is not well defined in the presence of a **negative weight cycle** in the graph.

All-Pairs Shortest Path Algorithms

We can solve an all-pairs shortest-paths problem by running a single source shortest-paths algorithm n times, once for each vertex as a source. Then the running time is:

- $O(n^3)$ if we use the Dijkstra algorithm (assuming no negative edge weights);

- $O(n^2|E|)$ if we use Bellman-Ford algorithm - i.e., $O(n^4)$ if the graph is dense.

Instead we will give a direct approach to finding the shortest paths between all pairs of vertices. We assume that negative weights exist, but no negative weight cycle.

First we will give an $O(n^4)$ algorithm, then improve it to $O(n^3 \log n)$. Later we will give a even better algorithm running in $O(n^3)$ time.

All Pairs Shortest Paths

Input: an adjacency matrix W where $w_{i,j}$ is the weight of the edge (i,j):

$$w_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of } (i,j) & \text{if } i \neq j \text{ and } (i,j) \in E \\ \infty & \text{if } (i,j) \notin E \end{cases}$$

Output:
A matrix D where $d_{i,j}$ is the shortest path from i to j.

The Basic Idea

Define $d_{i,j}^{(m)}$ to be the shortest path between i and j using paths of up to m edges. When $m = 0$, we have

$$d_{ij}^{(0)} = \begin{cases} 0 & : \quad i = j \\ \infty & : \quad i \neq j \end{cases}$$

Recursively we define,

$$d_{i,j}^{(m)} \;=\; min_{1 \leq k \leq n}\left[d_{i,k}^{(m-1)} + w_{k,j}\right].$$

If there are no negative weight cycles then no shortest path has more than $n-1$ edges.

How to compute these matrices?

Computing the Matrices

Define a sequence of matrices $D^{(1)}, D^{(2)}, \ldots D^{(n-1)}$, where for $m = 1, 2, \ldots, n-1$, we have $D^{(m)} = \left(d_{ij}^{(m)}\right)$.

Note that $D^{(1)} = W$.

The key procedure is to compute the matrix $D^{(m)}$ given $D^{(m-1)}$ and W: extending the shortest paths computed so far by one more edge.

Theorem 1. *The procedure EXTEND-SHORTEST-PATHS($D^{(m-1)}, W$) returns the matrix $D' = D^{(m)}$.*

Theorem 2. *For $n \times n$ matrices D and W, the procedure EXTEND-SHORTEST-PATHS($D^{(m-1)}, W$) takes $\Theta(n^3)$ steps.*

Slow All-Pairs Algorithm

Theorem 3. *The SLOW-ALL-PAIRS-SHORTEST-PATHS algorithm computes the correct shortest paths and terminates in* $\Theta(n^4)$ *steps.*

Shortest Paths and Matrix Multiplication

EXTEND-SHORTEST-PATHS procedure is closely related to MATRIX-MULTIPLY.

Let $C = A \cdot B$ be the product of two $n \times n$ matrices A and B. For $i, j = 1, 2, \ldots, n$, we compute

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

Note that by substituting,

$$d^{(m-1)} \to a$$

$$w \to b$$

$$d^{(m)} \to c$$

$$min \to +$$

$$+ \to \cdot$$

in

$$d_{i,j}^{(m)} \;=\; min_{1\leq k\leq n}\left[d_{i,k}^{(m-1)} + w_{k,j}\right]$$

we get matrix multiplication.

Repeated Squaring

Let $D' = D \cdot W$, where the operation $\cdot$ is the output of the procedure EXTEND-SHORTEST-PATHS(D, W).

The SLOW-ALL-PAIRS-SHORTEST-PATHS algorithm starts with $D^{(1)} = W$ and computes $D^{(m)} = W^m$ for $m = 2, ..., n-1$.

Like the "product" operation our "$\cdot$" operation is **associative**, i.e.

$$A \cdot (B \cdot C) \;=\; (A \cdot B) \cdot C$$

Assume that $n - 1 = 2^k$. A faster method for computing D^{n-1} is:

For $t = 1$ to k do

$$D^{2^t} = D^{2^{t-1}} \cdot D^{2^{t-1}}$$

Fast All-Pairs Shortest Paths Algorithm

We need to compute $D^{(m)}$ for some $m \geq n-1$.

Let $m = 2^{\lceil \log_2(n-1) \rceil}$

Theorem 4. *The FAST-ALL-PAIRS-SHORTEST-PATHS algorithm computes the correct distances in* $\Theta(n^3 \log n)$ *steps.*

The Floyd-Warshall algorithm

The previous algorithm extends in each iteration the number of edges used by the paths.

This algorithm extends the set of vertices that can be used as **intermediate** vertices on the paths.

For a path $P = v_1, v_2, ..., v_{k-1}, v_k$, the edges $v_2, ..., v_{k-1}$ are intermediate edges.

Let $V = \{1, ..., n\}$.

In iteration k, the algorithms computes all pairs shortest paths with intermediate vertices in $\{1, ..., k\}$.

In iteration k, the algorithms computes all pairs shortest paths with intermediate vertices in $\{1, ..., k\}$.

Let $d_{i,j}^{(k)}$ = the distance of a shortest path from i to j using only vertices of $\{1, ..., k\}$.

Lemma 1.

$$d_{i,j}^{(k)} = \begin{cases} w_{i,j} & \textit{if } k = 0 \\ MIN\left[d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right] & \textit{if } k \geq 1 \end{cases}$$

Proof. Let P be a shortest path from i to j using vertices in $\{1, ..., k\}$.

If P does not use k then $d_{i,j}^{(k)} = d_{i,j}^{(k-1)}$.

Otherwise P consists of a path P_1 from i to k, followed by a path P_2 from k to j.

P_1 is a shortest path from i to k in $\{1, ..., k-1\}$ and P_2 is a shortest path from k to j in $\{1, ..., k-1\}$. $\square$

Theorem 5. *The run-time of the Floyd-Warshall algorithm is* $O(n^3)$ *steps.*

Transitive Closure

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

$$E^* \;=\; \{(i,j) \mid \text{there is a path from } i \text{ to } j \text{ in } G\}.$$

Giving G, we can compute G^* by computing all pairs shortest paths with all edges having weight 1.

More efficiently:

Let

$$t_{i,j}^{(0)} \;=\; \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1 & \text{if } i = j \text{ or } (i,j) \in E \end{cases}$$

For $k \geq 1$

$$t_{i,j}^{(k)} = t_{i,j}^{(k-1)} \vee \left(t_{i,k}^{(k-1)} \wedge t_{k,j}^{(k-1)}\right).$$