Introduction to Algorithms

Chapter 25 : All-Pairs Shortest Paths

Xiang-Yang Li and Haisheng Tan

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2024

Outline of Topics

All-Pairs Shortest Paths

In this chapter, we consider the problem of finding shortest paths between all pairs of vertices in a graph $G = (V, E)$, where $|V| = n$.

Input: an $n \times n$ matrix W representing the edge weights of an n-vertex directed graph $G = (V, E)$

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

Output: an $n \times n$ matrix $D = (d_{ij})$ where entry d_{ij} contains the weight of a shortest path from vertex i to vertex j .

Shortest paths and matrix multiplication

Recall: Single-Source Shortest Paths
Bellman-Ford: $O(VE)$
Dijkstra: $O(E + V \log V)$ (no negative edge)

This section presents a dynamic-programming algorithm. Each major loop of the dynamic program will invoke an operation that is very similar to matrix multiplication, so that the algorithm will look like repeated matrix multiplication.

Start by developing a $\Theta\left(V^4\right)$ time algorithm for the all-pairs shortest-paths problem and then improve it to $\Theta\left(V^3 \lg V\right)$

A recursive solution

Optimal Substructure: let $l_{ij}^{(m)}$ be the minimum weight of any path from vertex i to vertex j that contains at most m edges. Thus,

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

For $m \geq 1$

$$\begin{aligned} l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) \\ &= \min_{1 \leq k \leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\} \end{aligned}$$

The latter equality follows since $w_{jj} = 0$ for all j

A recursive solution

If the graph contains no negative-weight cycles, then for every pair of vertices i and j for which $\delta(i,j) < \infty$, there is a shortest path from i to j that is simple and thus contains at most $n-1$ edges.

The actual shortest-path weights are therefore given by

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)}$$

Computing the weights bottom up

Given matrices $L^{(m-1)}$ and W, returns the matrix $L^{(m)}$, that is, extending one more edge.

EXTEND-SHORTEST-PATHS(L,W)

$n = L.rows$
let $L' = \left(l'_{ij}\right)$ be a new $n \times n$ matrix
for $i = 1$ to n do
    for $j = 1$ to n do
        $l'_{ij} = \infty$
        for $k = 1$ to n do
            $l'_{ij} = \min\left(l'_{ij}, l_{ik} + w_{kj}\right)$
return L'

Time: $\Theta\left(n^3\right)$ due to the three nested for loops

Computing the weights bottom up

Suppose we wish to compute the matrix product $C = A \cdot B$ of two $n \times n$ matrices A and B.

Then, for $i, j = 1, 2, \ldots, n$, we compute

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

Computing the weights bottom up

If we make the substitutions

$$
\begin{aligned}
l^{(m-1)} &\to a \\
w &\to b \\
l^{(m)} &\to c \\
\min &\to + \\
+ &\to \cdot
\end{aligned}
$$

Computing the weights bottom up

If we make these changes to EXTEND - SHORTEST - PATHS, and also replace ∞ (the identity for min) by 0 (the identity for addition)

SQUARE-MATRIX-MULTIPLY(A,B)

```
n = L.rows
let C be a new n × n matrix
for i = 1 to n do
    for j = 1 to n do
        c'_ij = 0
        for k = 1 to n do
            c_ij = c_ij + a_ik · b_kj
return C
```

Computing the weights bottom up

Letting $A \cdot B$ denote the matrix "product" returned by EXTEND-SHORTEST-PATHS(A, B), we compute the sequence of $n-1$ matrices.

$$
\begin{aligned}
L^{(1)} &= L^{(0)} \cdot W = W \\
L^{(2)} &= L^{(1)} \cdot W = W^2 \\
L^{(3)} &= L^{(2)} \cdot W = W^3 \\
&\cdots \\
L^{(n-1)} &= L^{(n-2)} \cdot W = W^{n-1}
\end{aligned}
$$

Computing the weights bottom up

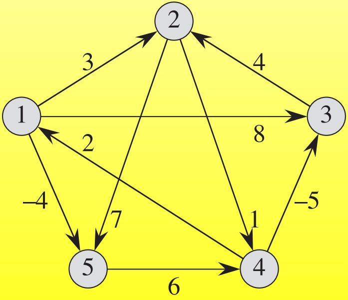
The following procedure computes this sequence in $\Theta\left(n^4\right)$ times.

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

1: $n = W.rows$
2: $L^{(1)} = W$
3: for $m = 2$ to $n-1$ do
4: $\quad L^{(m)}$ be a new $n \times n$ matrix
5: $\quad L^{(m)} = \mathrm{EXTEND-SHORTEST-PATHS}\left(L^{(m-1)}, W\right)$
6: return $L^{(n-1)}$

Example

Recall: $l_{ij}^{(m)} = \min_{1\le k\le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}$

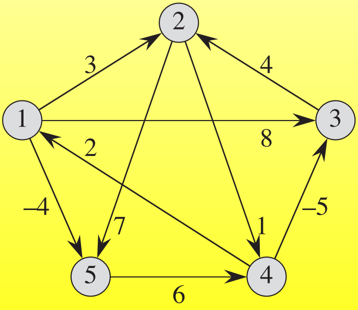


$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

Example

Recall: $l_{ij}^{(m)} = \min_{1 \le k \le n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\}$



$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Improving the running time

Our goal, is not to compute all the $L^{(m)}$ matrices, we are interested only in matrix $L^{(n-1)}$.

In the absence of negative-weight cycles, equation implies $L^{(m)} = L^{(n-1)}$ for all integers $m \geq n-1$

Therefore, we can compute $L^{(n-1)}$ with only $\lceil \lg(n-1) \rceil$ matrix products.

Improving the running time

The "matrix production" defined by EXTEND-SHORTEST-PATHES is associative.

$$L^{(1)} = L^{(0)} \cdot W = W$$

$$L^{(2)} = L^{(1)} \cdot W = W^2 = W \cdot W$$

$$L^{(4)} = W^4 = W^2 \cdot W^2$$

$$L^{(8)} = W^8 = W^4 \cdot W^4$$

$$\ldots$$

$$L^{\left(2^{\lceil \lg(n-1) \rceil}\right)} = W^{\left(2^{\lceil \lg(n-1) \rceil}\right)} = W^{\left(2^{\lceil \lg(n-1) \rceil - 1}\right)} \cdot W^{\left(2^{\lceil \lg(n-1) \rceil - 1}\right)}$$

Since $2^{\lceil \lg(n-1) \rceil} \geq n-1$, we have $L^{\left(2^{\lceil \lg(n-1) \rceil}\right)} = L^{(n-1)}$.

repeated squaring

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

```
n = W.rows
L^(1) = W
while m ≤ n − 1 do
    Let L^(2m) be a new n × n matrix
    L^(2m) = EXTEND − SHORTEST − PATHS (L^(m), L^(m))
    m=2m
return L^m
```

Because each of the $[\lg(n-1)]$ matrix products takes $\Theta\left(n^3\right)$ times, FASTER-ALL-PAIRS-SHORTEST-PATHS runs in $\Theta\left(n^3 \lg n\right)$ times.

The Floyd-Warshall algorithm

This section presents a different dynamic-programming algorithm known as the Floyd-Warshall algorithm that runs in $\Theta\left(n^3\right)$ times.

As before, negative-weight edges may be present, but we assume that there are no negative-weight cycles.

The structure of a shortest path

The Floyd-Warshall algorithm considers the intermediate vertices of a shortest path, where an intermediate vertex of a simple path $p = \langle v_1, v_2, \ldots, v_l \rangle$ is any vertex of p other than v_1 or v_l,that is, any vertex in the set $\{v_2, v_3, \ldots, v_{l-1}\}$.

The structure of a shortest path

The vertices of G are $V = \{1, 2, 3, \ldots, n\}$.let us consider a subset$\{1, 2, 3, \ldots, k\}$,of vertices for some k.

For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, 3, \ldots, k\}$, and let p be a minimum-weight path from among them.

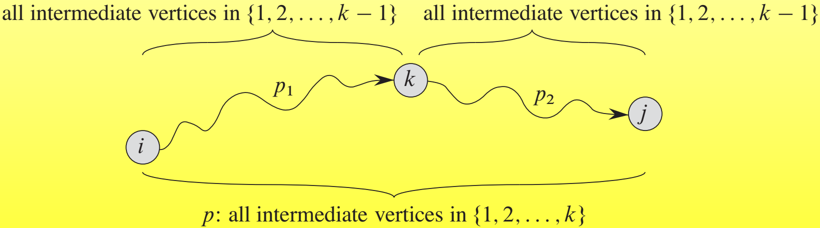
The Floyd-Warshall algorithm exploits a relationship between path p and shortest paths from i to j with all intermediate vertices in the set$\{1, 2, 3, \ldots, k-1\}$

The structure of a shortest path

The relationship depends on whether or not k is an intermediate vertex of path p.

- If not,then all intermediate vertices of path p are in the set $\{1,2,3,\ldots,k-1\}$.
- If yes, then we decompose p into $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$.$p_1$ is a shortest path from i to k with all intermediate vertices in the set $\{1,2,3,\ldots,k-1\}$, The same is true of p_2

The structure of a shortest path



A recursive solution

Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set $\{1, 2, 3, \ldots, k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases}$$

The final answer $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$

Computing bottom up

FLOYD-WARSHALL(W)

1: $n = W.rows$

2: $D^{(0)} = W$

3: for $k = 1$ to n do

4: let $D^{(k)} = d_{ij}^{(k)}$ be a new $n \times n$ matrix

5: for $i = 1$to n do

6: for $j = 1$to n do

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

8: return D^n

Because each execution of line 7 takes $O(1)$ time, the algorithm runs in time $\Theta\left(n^3\right)$.

Example

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

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Example

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Constructing a shortest path

We can compute the predecessor matrix Π while the algorithm computes the matrices $D^{(k)}$.

We compute a sequence of matrices $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$, where $\Pi = \Pi^{(n)}$ and we define $\pi_{ij}^{(k)}$ as the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, \ldots, k\}$.

Constructing a shortest path

When $k = 0$,

$$\pi_{ij}^{(0)} = \begin{cases} \mathrm{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

When $k \geq 1$,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Transitive closure of a directed graph

We define the transitive closure of G as the graph $G^* = (V, E^*)$, where $E^* = (i, j)$: there is a path from vertex i to vertex j in G.

One way to compute the transitive closure of a graph in $\Theta\left(n^3\right)$ time is to assign a weight of 1 to each edge of E and run the Floyd-Warshall algorithm.

If there is a path from vertex i to vertex j, we get $d_{ij} < n$, otherwise, we get $d_{ij} = \infty$

Transitive closure of a directed graph

Another way to compute the transitive closure of G in $\Theta\left(n^3\right)$ time that can save time and space in practice.

This method substitutes the logical operations $\vee$(logical OR) and $\wedge$ (logical AND) for the arithmetic operations min and + in the Floyd-Warshall algorithm.

Transitive closure of a directed graph

For $i, j, k = 1, 2, ..., n$, we define $t_{ij}^{(k)}$ to be 1 if there exists a path in graph G from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, 3, ..., k\}$, and 0 otherwise.

We construct the transitive closure $G^* = (V, E^*)$, by putting edge (i, j) into E^* if and only if $t_{ij}^{(n)} = 1$

Transitive closure of a directed graph

A recursive definition of $t_{ij}^{(k)}$ is:

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

and for $k \geq 1$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right)$$

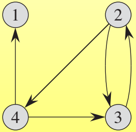
We compute the matrices $T^{(k)} = \left(t_{ij}^{(k)} \right)$ in order of increasing k.

TRANSITIVE-CLOSURE(G)

$n = |G.V|$
let $T^{(0)} = (t_{ij}^{(0)})$ be a new $n \times n$ matrix
for $i = 1$ to n do
    for $j = 1$ to n do
        if $i == j$ ot $(i,j) \in G.E$ then
            $t_{ij}^{(0)} = 1$
        else
            $t_{ij}^{(0)} = 0$
for $k = 1$ to n do
    let $T^{(k)} = (t_{ij}^{(k)})$ be a new $n \times n$ matrix
    for $i = 1$ to n do
        for $j = 1$ to n do
            $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$

The TRANSITIVE-CLOSURE procedure runs in $\Theta\left(n^3\right)$ times.

Example



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 25.5 A directed graph and the matrices $T^{(k)}$ computed by the transitive-closure algorithm.

Johnson's algorithm for sparse graphs

Johnson's algorithm uses the technique of reweighting:

If all edge weights w in a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ are nonnegative, we can find shortest paths between all pairs of vertices by running Dijkstra's algorithm once from each vertex

with the Fibonacci-heap min-priority queue, the running time of this all-pairs algorithm is $\mathrm{O}\left(\mathrm{V}^2 \lg \mathrm{V} + \mathrm{VE}\right)$

Johnson's algorithm for sparse graphs

The new set of edge weights $\hat{w}$ must satisfy two important: properties:

- For all pairs of vertices $u, v \in V$, a path p is a shortest path from u to v using weight function w if and only if p is also a shortest path from u to v using weight function $\hat{w}$
- For all edges (u, v) , the new weight $\hat{w}(u, v)$ is nonnegative

We use δ to denote shortest-path weights derived from weight function w and $\hat{\delta}$ to denote shortest-path weights derived from weight function $\hat{w}$.

Lemma 25.1 Reweighting does not change shortest paths

Given a weighted, directed graph $G=(V,E)$ with weight function $w:E\rightarrow\mathbb{R}$, let $h:V\rightarrow\mathbb{R}$ be any function mapping vertices to real numbers. For each edge $(u,v)\in E$,define

$$\hat{w}(u,v)=w(u,v)+h(u)-h(v)$$

Let $p=<v_0,v_1,...,v_k>$ be any path from vertex v_0 to vertex v_k. $w(p)=\delta(v_0,v_k)$ if and only if $\hat{w}(p)=\hat{\delta}(v_0,v_k)$. G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\hat{w}$.

Producing nonnegative weights by reweighting

Our goal is to ensure $\hat{w}(u, v)$ to be nonnegative for all edges $(u, v) \in E$.

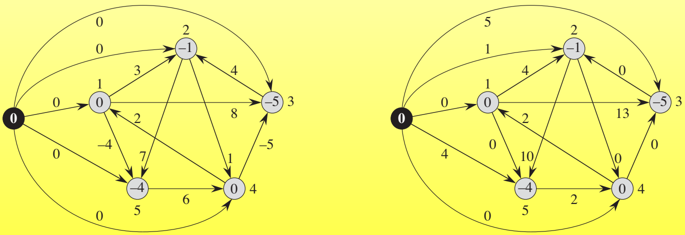
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$,we make a new graph $G' = (V', E')$, where $V' = V \cup \{s\}$ for some new vertex $s \notin V$ and $E' = E \cup \{(s, v) : v \in V\}$.

Weight function w is extended so that $w(s, v) = 0$ for all $v \in V$

No shortest paths in G , other than those with source s, contain s.

Therefore, G' has no negative-weight cycles if and only if G has no negative-weight cycles.

Example



(a) (b)

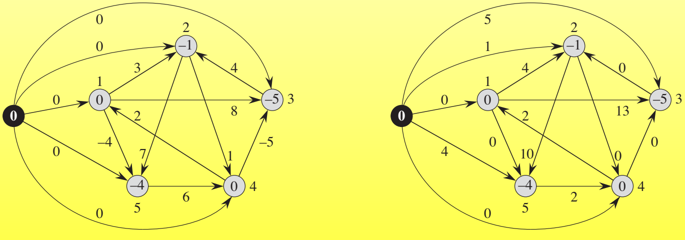
Figure (b) shows the graph G′ from Figure (a) with reweighted edges.

Producing nonnegative weights by reweighting

Suppose that G and G' have no negative-weight cycles.
Let us define $h(v) = \delta(s, v)$ for all $v \in V'$
By the triangle inequality, $h(v) \leq h(u) + w(u, v)$ for all edges $(u, v) \in E'$. And we have satisfied the second property:

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0$$

Example



(a) (b)

Figure (b) shows the graph G′ from Figure (a) with reweighted edges.

Computing all-pairs shortest paths

Johnson's algorithm to compute all-pairs shortest paths uses the Bellman-Ford algorithm and Dijkstra's algorithm as subroutines.

It assumes implicitly that the edges are stored in adjacency lists.

The algorithm returns the usual $|V| \times |V|$ matrix $D = d_{ij}$, where $d_{ij} = \delta(i,j)$, or it reports that the input graph contains a negative-weight cycle.

We assume that the vertices are numbered from 1 to $|V|$.

Johnsons algorithm

JOHNSON(G, w)

compute G' where $G'.V = G.V \cup \{s\}$, $G'.E = G.E \cup \{(s,v) : v \in G.V\}$ and $w(s,v) = 0$ for all $v \in G.V$

if BELLMAN-FORD(G', w, s) == False then

    print "the input graph contains a negative-weight cycle

else

    for each vertex $v \in G'.V$ do

        set $h(v)$ to the value of $\delta(s,v)$ computed by the Bellman-Ford algorithm

    for each edge $(u,v) \in G'.E$ do   // reweight each edge

        $\hat{w}(u,v) = w(u,v) + h(u) - h(v)$

    let $D = (d_{uv})$ to be a new $n \times n$ matrix

    for each vertex $u \in G.V$ do

        run DIJKSTRA$(G, \hat{w}, u)$ to compute $\hat{\delta}(u,v)$ for all $v \in G.V$

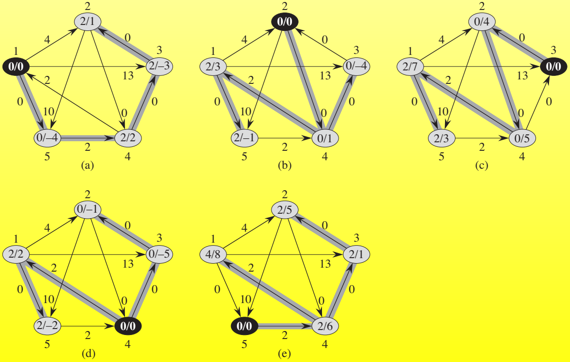
        for each vertex $v \in G.V$ do

            $d_{uv} = \hat{\delta}(u,v) + h(v) - h(u)$ // compute $\delta(u,v)$

return D

Example

Take each node as source u marked black. In each node v, record $\hat{\delta}(u,v)/\delta(u,v)$, where $\delta(u,v) = \hat{\delta}(u,v) + h(v) - h(u)$



Johnsons algorithm running time

If we implement the min-priority queue in Dijkstra's algorithm by a Fibonacci heap, Johnson's algorithm runs in $O(V^2 \lg V + VE)$

The simpler binary minheap implementation yields a running time of $O(VE \lg V)$,which is still asymptotically faster than the Floyd-Warshall algorithm if the graph is sparse.