Chapter 24
Single-Source Shortest Paths

Chi-Yeh Chen
陳奇業
成功大學資訊工程學系

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

國立成功大學 九十週年
90th Anniversary of NCKU

Overview

- Given a directed graph $G = (V, E)$, weight function $w: E \rightarrow \Re$, $|V| = n$.
- Goal: create an $n \times n$ matrix of shortest-path distances $\delta(u, v)$.
- Could run BELLMAN-FORD once from each vertex:
  $O(V^2E)$ — which is $O(V^4)$ if the graph is **dense** ($E = \theta(V^2)$).
- If no negative-weight edges, could run Dijkstra's algorithm once from each vertex:
  $O(VE \lg V)$ with binary heap—$O(V^3 \lg V)$ if dense.
  $O(V^2 \lg V + VE)$ with Fibonacci heap — $O(V^3)$ if dense.
- We'll see how to do in $O(V^3)$ in all cases, with no fancy data structure.

Shortest paths and matrix multiplication

NCKU
National Cheng Kung University
1931

藏行顯光
成就共好
Achieve Securely
Prosper Mutually
國立成功大學 九十週年
90th Anniversary of NCKU

Shortest paths and matrix multiplication

- Assume that G is given as adjacency matrix of weights: $W = (w_{ij})$, with vertices numbered 1 to n.

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

- Output is matrix $D = (d_{ij})$, where $d_{ij} = \delta(i,j)$.
  Won't worry about predecessor—see book.
- Will use dynamic programming at first.
- ***Optimal substructure***: Recall: subpaths of shortest paths are shortest paths.
- ***Recursive solution***: Let $l_{ij}^{(m)}$ = weight of shortest path $i \leadsto j$ that contains $\leq m$ edges.

藏行顯光 成就共好 國立成功大學 九十週年 90th Anniversary of NCKU
Achieve Securely · Prosper Mutually

- $m = 0$
  ⇒there is a shortest path $i \rightsquigarrow j$ with $\leq m$ edges if and only if $i = j$
  $$\Rightarrow l_{ij}^{(0)} = \begin{cases} 0 \text{ if } i = j \\ \infty \text{ if } i \neq j \end{cases}$$

- $m \geq 1$
  $$\Rightarrow \ l_{ij}^{(m)} = \min(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\}) \ (k \text{ is all predecessors of } j)$$
  $$= \min_{1 \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\} \ (\text{since } w_{jj} = 0 \text{ for all } j)$$

- Observer that when $m = 1$, must have $l_{ij}^{(1)} = w_{ij}$.
  Conceptually, when the path is restricted to at most 1 edge, the weight of the shortest path $i \rightsquigarrow j$ must be w_{ij}.
  And the math works out, too:

$$
\begin{aligned}
l_{ij}^{(1)} &= \min_{1\le k\le n}\{l_{ik}^{(0)} + w_{kj}\} \\
&= l_{ii}^{(0)} + w_{ij} \qquad (\mathrm{l}_{\mathrm{ii}}^{(0)} \text{ is the only non} - \infty \text{ among } l_{ik}^{(0)}) \\
&= w_{ij}.
\end{aligned}
$$

All simple shortest paths contain $\leq n - 1$ edges

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

Compute a solution bottom-up: : Compute $L^{(1)}, L^{(2)}, \dots, L^{(n-1)}$.

Start with $L^{(1)} = W$, since $l_{ij}^{(1)} = w_{ij}$.

Go from $L^{(m-1)}$ to $L^{(m)}$:

EXTEND-SHORTEST-PATH(L, W)

$n = L.rows$
let $L' = (l'_{ij})$ 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**
      $l'_{ij} = \min(l'_{ij}, l_{ij} + w_{kj})$
return L'

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

$n = W.rows$
$L^{(1)} = W$
for $m = 2$ ***to*** $n - 1$ **do**
    let $L^{(m)}$ be a new $n \times n$ matrix
    $L^{(m)} = \text{EXTEND-SHORTEST-PATH}(L^{(m-1)}, W)$
return $L^{(n-1)}$

***Time*:**

- EXTEND: $\Theta(n^3)$.
- SLOW-APSP: $\Theta(n^4)$.

***Observation*:** EXTEND is like matrix multiplication:

$L \rightarrow A$
$W \rightarrow B$
$L' \rightarrow C$
$min \rightarrow +$
$+ \rightarrow \cdot$
$\infty \rightarrow 0$

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

create C, an $n \times n$ matrix

for $i \leftarrow 1$ **to** n

  do for $j \leftarrow 1$ **to** n

    do $c_{ij} \leftarrow 0$

     for $k \leftarrow 1$ **to** n

      do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

So, we can view EXTEND as just like matrix multiplication!

Why do we care?

Because our goal is to compute $L^{(n-1)}$ as fast as we can.

Don't need to compute *all* the intermediate $L^{(1)}, L^{(2)}, L^{(3)}, \dots L^{(n-1)}$.

Suppose we had a matrix A and we wanted to compute A^{n-1} (like calling EXTEND $n-1$ times).

Could compute $A, A^2, A^4, A^8, \dots$

If we knew $A^m = A^{n-1}$ for all $m \geq n-1$, could just finish with A^r, where r is the smallest power of 2 that's $\geq n-1$. ($r = 2^{\lceil \lg(n-1) \rceil}$)

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

$n = W.rows$
$L^{(1)} = W$
$m = 1$
while $m < n - 1$ **do**
    let $L^{(2m)}$ be a new $n \times n$ matrix
    $L^{(2m)} = \text{EXTEND-SHORTEST-PATH}(L^{(m)}, L^{(m)})$
    $m = 2m$
return $L^{(m)}$

OK to overshoot, since products don't change after $L^{(n-1)}$.

Time: $\Theta(n^3 \lg n)$

The Floyd-Warshall algorithm

NCKU
National Cheng Kung University
1931

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

Floyd-Warshall algorithm

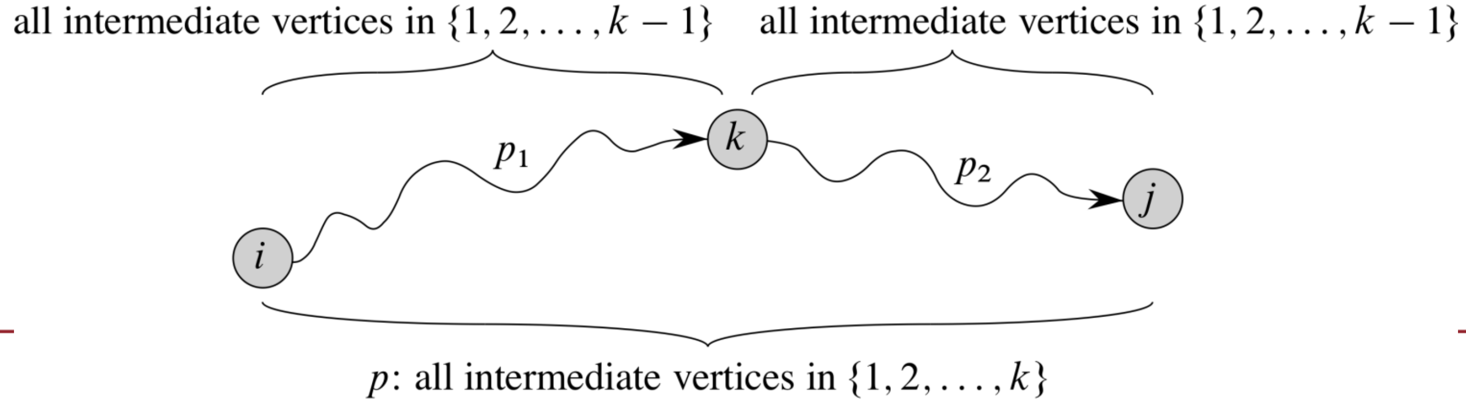
A different dynamic-programming approach.

For path $p = \langle v_1, v_2, \dots, v_l \rangle$, an ***intermediate vertex*** is any vertex of p other than v_1 or v_l.

Let $d_{ij}^{(k)}$ = shortest-path weight of any path $i \rightsquigarrow j$ with all intermediate vertices in $\{1, 2, \dots, k\}$.

Consider a shortest path $i \overset{p}{\rightsquigarrow} j$ with all intermediate vertices in $\{1, 2, \dots, k\}$:

- If k is not an intermediate vertex, then all intermediate vertices of p are in $\{1, 2, \dots, k-1\}$.
- If k is an intermediate vertex:



Recursive formulation

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

(Have $d_{ij}^{(0)} = w_{ij}$ because can't have intermediate vertices $\Rightarrow \leq 1$ edges.)
Want $D^{(n)} = \left(d_{ij}^{(n)}\right)$, since all vertices numbered $\leq n$.

Compute bottom-up
Compute in increasing order of *k*

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

FLOYD-WARSHALL(W)

$n = W.rows$

$D^{(0)} = W$

for $k = 1$ ***to*** n **do**

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

  for $i = 1$ ***to*** n **do**

    for $j = 1$ ***to*** n **do**

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

return $D^{(n)}$

Can drop superscripts. (See Exercise 25.2-4 in text.)

Time: $\boldsymbol{\Theta(n^3)}$.

Transitive closure

Given $G(V, E)$, directed.

Compute $G^* = (V, E^*)$.

- $E^* = \{(i, j)\text{: there is a path } i \leadsto j \text{ in } G\}$.

Could assign weight of 1 to each edge, then run FLOYD-WARSHALL.

- If $d_{ij} < n$, then there is a path $i \leadsto j$.
- Otherwise, $d_{ij} = \infty$ and there is no path.

Simpler way: Substitute other values and operators in FLOYD-WARSHALL.

- Use unweighted adjacency matrix
- $\min \to \vee$ (OR)
- $+ \to \wedge$ (AND)

- $d_{ij}^{(k)} = \begin{cases} 1 \text{ if there is path } i \rightsquigarrow j \text{ with alla intermediate vertices in } \{1,2,\dots,k\}, \\ 0 \text{ otherwise.} \end{cases}$

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

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

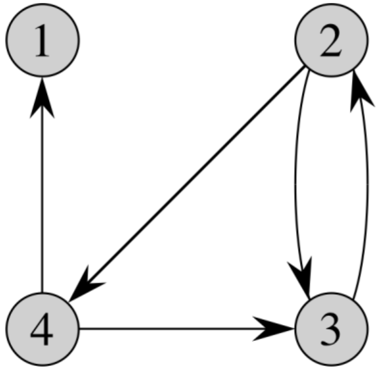
藏行顯光
成就共好
Achieve Securely
Prosper Mutually

TRANSITIVE-CLOSURE(G)

```
n = |G.V|
let T^(0) = (t_ij^(0)) be a new n × n matrix
for i = 1 to n do
    for j = 1 to n do
        if i == j or (i, j) ∈ 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 × n matrix
    for i = 1 to n do
        for j = 1 to n do
            t_ij^(k) = t_ij^(k-1) ∨ (t_ik^(k-1) ∧ t_kj^(k-1))
return T^(n)
```

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

Time: $\Theta(n^3)$, but simpler operations than FLOYD-WARSHALL.



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

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

Johnoson’s algorithm for sparse graphs

NCKU
National Cheng Kung University
1931

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

- ***Idea*:** If the graph is sparse, it pays to run Dijkstra's algorithm once from each vertex.
- If we use a Fibonacci heap for the priority queue, the running time is down to $O(V^2 \lg V + VE)$, which is better than FLOYD-WARSHALL's $\Theta(V^3)$ time if $E = o(V^2)$.
- But Dijkstra's algorithm requires that all edge weights be nonnegative.
- Donald Johnson figured out how to make an equivalent graph that *does* have all edge weights ≥ 0.

Reweighting

Compute a new weight function $\widehat{w}$ such that

1. For all $u, v \in V$, p is a shortest path $u \rightsquigarrow v$ using w if and only if p is a shortest path $u \rightsquigarrow v$ using $\widehat{w}$.
2. For all $(u, v) \in E$, the new weight $\widehat{w}(u, v)$ is nonnegative.

- Property(1) says that it suffices to find shortest paths with $\widehat{w}$.
- Property(2) says we can do so by running Dijkstra's algorithm from each vertex.

- How to come up with $\widehat{w}$?
- Lemma 25.1 shows it's easy to get property(1):

Lemma (Rewighting doesn't change shortest paths)

Given a directed, weighted graph $G = (V, E)$, $w: E \to \boldsymbol{R}$. Let h be any function such that $h: V \to \boldsymbol{R}$.
For all $(u, v) \in E$, define

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

Let $p = \langle v_0, v_1, \dots, v_k \rangle$ be any path $v_0 \rightsquigarrow v_k$.

Then, p is a shortest path $v_0 \rightsquigarrow v_k$ with w if and only if p is a shortest path $v_0 \rightsquigarrow v_k$ with $\widehat{w}$.

Also, G has a negative-weight cycle with weight w iff G has a negative-weight cycle with weight $\widehat{w}$.

藏行顯光 成就共好 國立成功大學九十週年 90th Anniversary of NCKU
Achieve Securely · Prosper Mutually

Proof

- First, we'll show that $\widehat{w}(p) = w(p) + h(v_0) - h(v_k)$:

$$\begin{aligned}\widehat{w}(p) &= \sum_{i=1}^{k} \widehat{w}(v_{i-1}, v_i) \\ &= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\ &= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \quad \text{(sum telescopes)} \\ &= w(p) + h(v_0) - h(v_k).\end{aligned}$$

- Therefore, any path $v_0 \rightsquigarrow v_k$ has $\widehat{w}(p) = w(p) + h(v_0) - h(v_k)$.
  Since $h(v_0)$ and $h(v_k)$ don't depend on the path from v_0 to v_k , if one path $v_0 \rightsquigarrow v_k$ is shorter than another with w, it's also shorter with $\widehat{w}$.
- Now show there exists a negative-weight cycle with w if and only if there exists a negative-weight cycle with $\widehat{w}$:

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

- Let cycle $C = \langle v_0, v_1, \dots, v_k \rangle$, where $v_0 = v_k$.
- Then $\widehat{w}(C) = w(C) + h(v_0) - h(v_k) = w(C)$ (since $v_0 = v_k$).

Therefore, C has a negative-weight cycle with w if and only if it has a negative-weight cycle with $\widehat{w}$. ■ (lemma)

So, now to get property(2), we just need to come up with a function

$h: V \to \boldsymbol{R}$ such that when we compute $\widehat{w}(u, v) = w(u, v) + h(u) - h(v)$, it's ≥ 0..

Do what we did for difference constraints:

- $G' = (V', E')$
- $V' = V \cup \{s\}$, where *s* is a new vertex.
- $E' = E \cup \{(s, v): v \in V\}$.
- $w(s, v) = 0$ for all $v \in V$.

藏行顯光 成就共好 Achieve Securely Prosper Mutually 國立成功大學 九十週年 90th Anniversary of NCKU

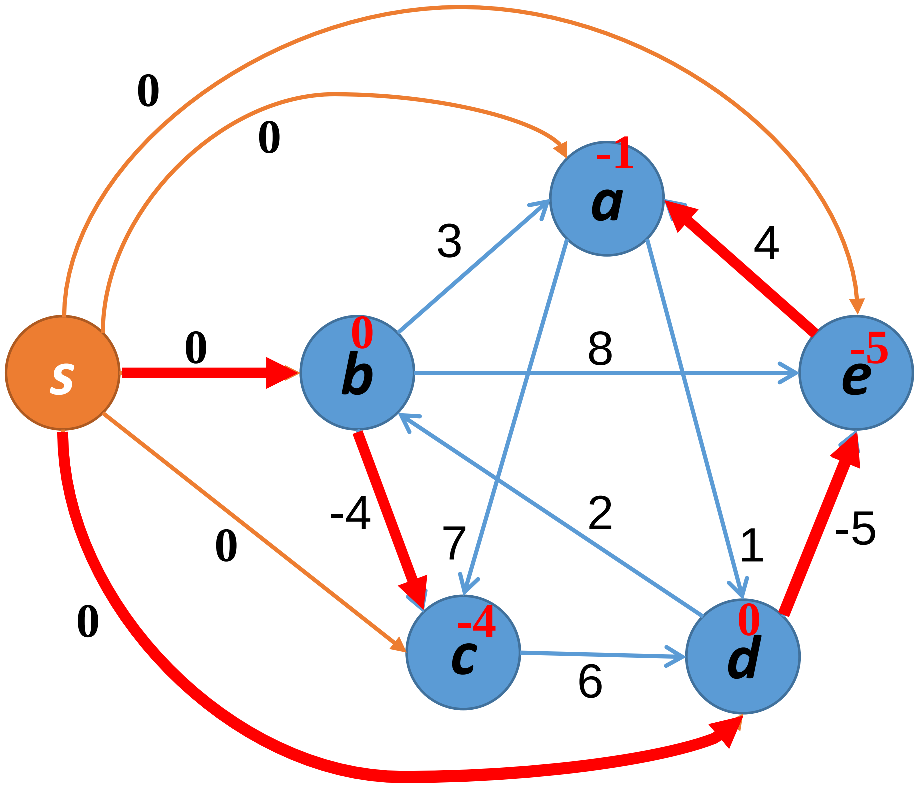


Figure 25.6 Johnson's all-pairs shortest-paths algorithm

(a) The graph *G'* with the original weight function *w*. The new vertex ***s*** is black.

- Since no edges enter s, G' has the same set of cycles as G.
  In particular, G' has a negative-weight cycle if and only if G does.

Define $h(v) = \delta(s, v)$ for all $v \in V$.

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

Proof By the triangle inequality,

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

$$h(v) \leq h(u) + w(u, v) .$$

Therefore, $w(u, v) + h(u) - h(v) \geq 0.$ ■(claim)

JOHNSON(G)

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**

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

  let $D = (d_{uv})$ 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**

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

return D

▷ Compute entry d_{uv} in matrix D

$$d_{uv} = \underbrace{\hat{\delta}(u,v) + \delta(s,v) - \delta(s,u)}_{\substack{\text{because if } p \text{ is a path } u \to v, \\ \text{then } \hat{w}(p) = w(p) + h(u) - h(v)}}$$

Time:

- $\Theta(V + E)$ to compute G' .
- $o(VE)$ to run BELLMAN-FORD.
- $\Theta\ (E)$ to compute $\widehat{w}$.
- $O(V^2 \lg V + VE)$ to run Dijkstra's algorithm $|V|$ times (using Fibonacci heap).
- $\Theta(V^2)$ to compute *D* matrix.

Total: $O(V^2 \lg V + VE)$.

藏行顯光
成就共好

Achieve Securely
Prosper Mutually

國立成功大學 九十週年
90th Anniversary of NCKU

國立成功大學
National Cheng Kung University