Analysis of Algorithms

All-Pairs Shortest Path

Andres Mendez-Vazquez

November 11, 2015

Outline

Cinvestav

Outline

Cinvestav

Problem

Definition

- Given u and v, find the shortest path.
- Now, what if you want **ALL PAIRS**!!!
- Use as a source all the elements in V.
- Clearly!!! you can fall back to the old algorithms!!!

Problem

Definition

- Given u and v, find the shortest path.
- Now, what if you want **ALL PAIRS**!!!
- Use as a source all the elements in V.
- Clearly!!! you can fall back to the old algorithms!!!

Problem

Definition

- Given u and v, find the shortest path.
- Now, what if you want **ALL PAIRS**!!!
- Use as a source all the elements in V.
- Clearly!!! you can fall back to the old algorithms!!!

Cinvestav

Problem

Definition

- Given u and v, find the shortest path.
- Now, what if you want **ALL PAIRS**!!!
- Use as a source all the elements in V.
- Clearly!!! you can fall back to the old algorithms!!!

What can we use?

Use Dijkstra's $|V|$ times!!!

- **If all the weights are non-negative.**

- Then, we have $O(V^2 E)$.
- Which is equal $O(V^4)$ in the case of $E = O(V^2)$.

What can we use?

Use Dijkstra's $|V|$ times!!!

- **If all the weights are non-negative.**
- This has, using Fibonacci Heaps, $O\left(V^2 \log V + VE\right)$ complexity.
- Which is equal $O\left(V^3\right)$ in the case of $E = O(V^2)$, but with a hidden large constant c.

What can we use?

Use Dijkstra's $|V|$ times!!!

- **If all the weights are non-negative.**
- This has, using Fibonacci Heaps, $O\left(V^2 \log V + VE\right)$ complexity.
- Which is equal $O\left(V^3\right)$ in the case of $E = O(V^2)$, but with a hidden large constant c.

Use Bellman-Ford $|V|$ times!!!

- If negative weights are allowed.
- Then, we have $O\left(V^2 E\right)$.
- Which is equal $O\left(V^4\right)$ in the case of $E = O(V^2)$.

Cinvestav

What can we use?

Use Dijkstra's $|V|$ times!!!

- **If all the weights are non-negative.**
- This has, using Fibonacci Heaps, $O(V^2 \log V + VE)$ complexity.
- Which is equal $O(V^3)$ in the case of $E = O(V^2)$, but with a hidden large constant c.

Use Bellman-Ford $|V|$ times!!!

- **If negative weights are allowed**.
- Then, we have $O(V^2 E)$.
- Which is equal $O(V^4)$ in the case of $E = O(V^2)$.

Cinvestav

What can we use?

Use Dijkstra's $|V|$ times!!!

- **If all the weights are non-negative.**
- This has, using Fibonacci Heaps, $O(V^2 \log V + VE)$ complexity.
- Which is equal $O(V^3)$ in the case of $E = O(V^2)$, but with a hidden large constant c.

Use Bellman-Ford $|V|$ times!!!

- **If negative weights are allowed**.
- Then, we have $O(V^2 E)$.
- Which is equal $O(V^4)$ in the case of $E = O(V^2)$.

What can we use?

Use Dijkstra's $|V|$ times!!!

- **If all the weights are non-negative.**
- This has, using Fibonacci Heaps, $O(V^2 \log V + VE)$ complexity.
- Which is equal $O(V^3)$ in the case of $E = O(V^2)$, but with a hidden large constant c.

Use Bellman-Ford $|V|$ times!!!

- **If negative weights are allowed**.
- Then, we have $O(V^2 E)$.
- Which is equal $O(V^4)$ in the case of $E = O(V^2)$.

This is not Good For Large Problems

Problems

- Computer Network Systems.
- Aircraft Networks (e.g. flying time, fares).
- Railroad network tables of distances between all pairs of cites for a road atlas.
- Etc.

This is not Good For Large Problems

Problems

- Computer Network Systems.
- Aircraft Networks (e.g. flying time, fares).
- Railroad network tables of distances between all pairs of cites for a road atlas.
- Etc.

This is not Good For Large Problems

Problems

- Computer Network Systems.
- Aircraft Networks (e.g. flying time, fares).
- Railroad network tables of distances between all pairs of cites for a road atlas.
- Etc.

Cinvestav

This is not Good For Large Problems

Problems

- Computer Network Systems.
- Aircraft Networks (e.g. flying time, fares).
- Railroad network tables of distances between all pairs of cites for a road atlas.
- Etc.

For more on this...

Something Notable

As many things in the history of analysis of algorithms the all-pairs shortest path has a long history.

For more on this...

Something Notable

As many things in the history of analysis of algorithms the all-pairs shortest path has a long history.

We have more from

- "*Studies in the Economics of Transportation*" by Beckmann, McGuire, and Winsten (1956) where the notation that we use for the matrix multiplication alike was first used.
- Chr. Wiener, Ueber eine Aufgabe aus der Geometria situs. Mathematische Annalen 6 (1873) 29–30, 1873.

For more on this...

Something Notable

As many things in the history of analysis of algorithms the all-pairs shortest path has a long history.

We have more from

- "*Studies in the Economics of Transportation*" by Beckmann, McGuire, and Winsten (1956) where the notation that we use for the matrix multiplication alike was first used.

In addition

- G. Tarry, Le probleme des labyrinthes, Nouvelles Annales de Mathématiques (3) 14 (1895) 187–190 [English translation in: N.L. Biggs, E.K. Lloyd, R.J. Wilson, Graph Theory 1736–1936, Clarendon Press, Oxford, 1976, pp. 18–20] (**For the theory behind depth-first search techniques**).
- Chr. Wiener, Ueber eine Aufgabe aus der Geometria situs, Mathematische Annalen 6 (1873) 29–30, 1873.

For more on this...

Something Notable

As many things in the history of analysis of algorithms the all-pairs shortest path has a long history.

We have more from

- *"Studies in the Economics of Transportation"* by Beckmann, McGuire, and Winsten (1956) where the notation that we use for the matrix multiplication alike was first used.

In addition

- G. Tarry, Le probleme des labyrinthes, Nouvelles Annales de Mathématiques (3) 14 (1895) 187–190 [English translation in: N.L. Biggs, E.K. Lloyd, R.J. Wilson, Graph Theory 1736–1936, Clarendon Press, Oxford, 1976, pp. 18–20] (**For the theory behind depth-first search techniques**).
- Chr. Wiener, Ueber eine Aufgabe aus der Geometria situs, Mathematische Annalen 6 (1873) 29–30, 1873.

Outline

Assumptions Matrix Representation

Matrix Representation of a Graph

For this, we have that each weight in the matrix has the following values

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

Then, we have $W = \begin{pmatrix} w_{11} & w_{22} & \dots & w_{1k-1} & w_{1n} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ w_{n1} & w_{n2} & \dots & w_{nn-1} & w_{nn} \end{pmatrix}$

Important

- There are not negative weight cycles.

Assumptions Matrix Representation

Matrix Representation of a Graph

For this, we have that each weight in the matrix has the following values

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

Then, we have $W = \begin{pmatrix} w_{11} & w_{22} & \dots & w_{1k-1} & w_{1n} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ w_{n1} & w_{n2} & \dots & w_{nn-1} & w_{nn} \end{pmatrix}$

Important

- There are not negative weight cycles.

Outline

Cinvestav

Observations

Ah!!!

- The next algorithm is a dynamic programming algorithm for
  - The all-pairs shortest paths problem on a directed graph $G = (V, E)$.

Observations

Ah!!!

- The next algorithm is a dynamic programming algorithm for
  - The all-pairs shortest paths problem on a directed graph $G = (V, E)$.

At the end of the algorithm will generate the following matrix

$$D = \begin{pmatrix} d_{11} & d_{22} & \cdots & d_{1k-1} & d_{1n} \\ \vdots & \vdots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \vdots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nn-1} & d_{nn} \end{pmatrix}$$

Each entry $d_{ij} = \delta(i, j)$.

Cinvestav

Observations

Ah!!!

- The next algorithm is a dynamic programming algorithm for
  - The all-pairs shortest paths problem on a directed graph $G = (V, E)$.

At the end of the algorithm will generate the following matrix

$$D = \begin{pmatrix} d_{11} & d_{22} & \dots & d_{1k-1} & d_{1n} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ d_{n1} & d_{n2} & \dots & d_{nn-1} & d_{nn} \end{pmatrix}$$

Each entry $d_{ij} = \delta(i, j)$.

Cinvestav

Outline

Structure of a Shortest Path

Consider Lemma 24.1

Given a weighted, directed graph $G = (V, E)$ with $p =< v_1, v_2, ..., v_k >$ be a SP from v_1 to v_k. Then,

- $p_{ij} =< v_i, v_{i+1}, ..., v_j >$ is a Shortest Path (SP) from v_i to v_j, where $1 \leq i \leq j \leq k$.

We can do the following

- Consider the shortest path p from vertex i and j, p **contains at most** m **edges.**

Structure of a Shortest Path

Consider Lemma 24.1

Given a weighted, directed graph $G = (V, E)$ with $p =< v_1, v_2, ..., v_k >$ be a SP from v_1 to v_k. Then,

- $p_{ij} =< v_i, v_{i+1}, ..., v_j >$ is a Shortest Path (SP) from v_i to v_j, where $1 \leq i \leq j \leq k$.

We can do the following

- Consider the shortest path p from vertex i and j, p **contains at most** m **edges**.
- Then, we can use the Corollary to make a decomposition

$$i \overset{p'}{\leadsto} k \to j \Longrightarrow \delta(i, j) = \delta(i, k) + w_{kj}$$

Cinvestav

Structure of a Shortest Path

Idea of Using Matrix Multiplication

- We define the following concept based in the decomposition Corollary!!!
- $l_{ij}^{(m)}$ =minimum weight of any path from i to j, it contains at most m edges i.e.

$$l_{ij}^{(m)} \text{ could be } \min_{k}\left\{ l_{ik}^{(m-1)} + w_{kj} \right\}$$

Structure of a Shortest Path

Idea of Using Matrix Multiplication

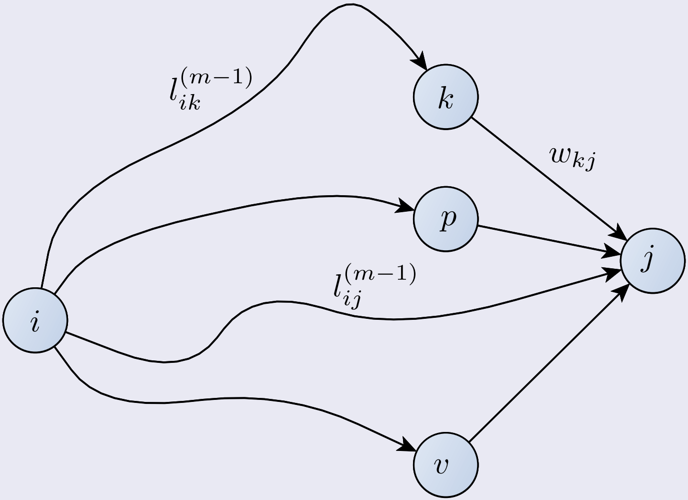
- We define the following concept based in the decomposition Corollary!!!
- $l_{ij}^{(m)}$ =minimum weight of any path from i to j, it contains at most m edges i.e.

$$l_{ij}^{(m)} \text{ could be } \min_{k} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\}$$

Cinvestav

Graphical Interpretation

Looking for the Shortest Path



Outline

Recursive Solution

Thus, we have that for paths with ZERO edges

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

Recursion Our Great Friend

- Consider the previous definition and decomposition. Thus

Recursive Solution

Thus, we have that for paths with ZERO edges

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

Recursion Our Great Friend

- Consider the previous definition and decomposition. Thus

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

Recursive Solution

Thus, we have that for paths with ZERO edges

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

Recursion Our Great Friend

- Consider the previous definition and decomposition. Thus

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

Cinvestav

Recursive Solution

Thus, we have that for paths with ZERO edges

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

Recursion Our Great Friend

- Consider the previous definition and decomposition. Thus

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

Recursive Solution

Why? A simple notation problem

$$l_{ij}^{(m)} = l_{ij}^{(m-1)} + 0 = l_{ij}^{(m-1)} + w_{jj}$$

Outline

Cinvestav

Transforming it to a iterative one

What is $\delta(i,j)$?

- If you do not have negative-weight cycles, and $\delta(i,j) < \infty$.
- Then, the shortest path from vertex i to j has at most $n-1$ edges

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = l_{ij}^{(n+2)} = \dots$$

- Then, we can compute first $L^{(1)}$ then compute $L^{(2)}$ all the way to $L^{(n-1)}$ which contains the actual shortest paths.

Transforming it to a iterative one

What is $\delta(i,j)$?

- If you do not have negative-weight cycles, and $\delta(i,j) < \infty$.
- Then, the shortest path from vertex i to j has at most $n-1$ edges

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = l_{ij}^{(n+2)} = \ldots$$

Back to Matrix Multiplication

- We have the matrix $L^{(m)} = \left(l_{ij}^{(m)}\right)$.
- Then, we can compute first $L^{(1)}$ then compute $L^{(2)}$ all the way to $L^{(n-1)}$ which contains the actual shortest paths.

Transforming it to a iterative one

What is $\delta(i,j)$?

- If you do not have negative-weight cycles, and $\delta(i,j) < \infty$.
- Then, the shortest path from vertex i to j has at most $n-1$ edges

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = l_{ij}^{(n+2)} = \dots$$

Back to Matrix Multiplication

- We have the matrix $L^{(m)} = \left(l_{ij}^{(m)}\right)$.
- Then, we can compute first $L^{(1)}$ then compute $L^{(2)}$ all the way to $L^{(n-1)}$ which contains the actual shortest paths.

What is $L^{(1)}$?

- First, we have that $L^{(1)} = W$, since $l_{ij}^{(1)} = w_{ij}$.

Transforming it to a iterative one

What is $\delta(i,j)$?

- If you do not have negative-weight cycles, and $\delta(i,j) < \infty$.
- Then, the shortest path from vertex i to j has at most $n-1$ edges

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = l_{ij}^{(n+2)} = \ldots$$

Back to Matrix Multiplication

- We have the matrix $L^{(m)} = \left(l_{ij}^{(m)}\right)$.
- Then, we can compute first $L^{(1)}$ then compute $L^{(2)}$ all the way to $L^{(n-1)}$ which contains the actual shortest paths.

What is $L^{(1)}$?

- First, we have that $L^{(1)} = W$, since $l_{ij}^{(1)} = w_{ij}$.

Transforming it to a iterative one

What is $\delta(i,j)$?

- If you do not have negative-weight cycles, and $\delta(i,j) < \infty$.
- Then, the shortest path from vertex i to j has at most $n-1$ edges

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = l_{ij}^{(n+2)} = ...$$

Back to Matrix Multiplication

- We have the matrix $L^{(m)} = \left(l_{ij}^{(m)}\right)$.
- Then, we can compute first $L^{(1)}$ then compute $L^{(2)}$ all the way to $L^{(n-1)}$ which contains the actual shortest paths.

What is $L^{(1)}$?

- First, we have that $L^{(1)} = W$, since $l_{ij}^{(1)} = w_{ij}$.

Outline

Cinvestav

Algorithm

Code

Extended-Shortest-Path(L, W)

1. $n = L.rows$
2. **let** $L' = \left(l'_{ij}\right)$ **be a new** $n \times n$
3. **for** $i = 1$ **to** n
4. **for** $j = 1$ **to** n
5. $l'_{ij} = \infty$
6. **for** $k = 1$ **to** n
7. $l'_{ij} = \min\left(l'_{ij}, l_{ik} + w_{kj}\right)$
8. **return** L'

Cinvestav

Algorithm

Code

Extended-Shortest-Path(L, W)

```
n = L.rows
let L' = (l'_ij) be a new n × n
for i = 1 to n
        for j = 1 to n
                l'_ij = ∞
                for k = 1 to n
                        l'_ij = min(l'_ij, l_ik + w_kj)
return L'
```

Algorithm

Code

Extended-Shortest-Path(L, W)

$n = L.rows$

let $L' = \left(l'_{ij}\right)$ **be a new** $n \times n$

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

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

    $l'_{ij} = \infty$

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

      $l'_{ij} = \min\left(l'_{ij}, l_{ik} + w_{kj}\right)$

return L'

Algorithm

Code

Extended-Shortest-Path(L, W)

$n = L.rows$

let $L' = \left(l'_{ij}\right)$ **be a new** $n \times n$

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

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

        $l'_{ij} = \infty$

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

            $l'_{ij} = \min\left(l'_{ij}, l_{ik} + w_{kj}\right)$

return L'

Algorithm

Complexity

If $|V| == n$ we have that $\Theta\left(V^3\right)$.

Algorithm

Complexity

If $|V| == n$ we have that $\Theta\left(V^3\right)$.

Outline

Cinvestav

Look Alike Matrix Multiplication Operations

Mapping That Can be Thought

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

Look Alike Matrix Multiplication Operations

Using the previous notation, we can rewrite our previous algorithm as

Square-Matrix-Multiply(A, B)

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

Complexity

Thus

The complexity of the **Extended-Shortest-Path** is equal to $O\left(n^3\right)$

Using the Analogy

Returning to the all-pairs shortest-paths problem

It is possible to compute the shortest path by extending such a path edge by edge.

Therefore

If we denote $A \cdot B$ as the "product" of the **Extended-Shortest-Path**

Using the Analogy

Returning to the all-pairs shortest-paths problem

It is possible to compute the shortest path by extending such a path edge by edge.

Therefore

If we denote $A \cdot B$ as the "product" of the **Extended-Shortest-Path**

Using the Analogy

We have that

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

Using the Analogy

We have that

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

Using the Analogy

We have that

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

Cinvestav

The Final Algorithm

We have that

Slow-All-Pairs-Shortest-Paths(W **)**

1. $n \leftarrow W.rows$
2. $L^{(1)} \leftarrow W$
3. **for** $m = 2$ **to** $n - 1$
4. $\quad L^{(m)} \leftarrow$ **EXTEND-SHORTEST-PATHS** $\left(L^{(m-1)}, W\right)$
5. **return** $L^{(n-1)}$

Cinvestav

With Complexity

Complexity

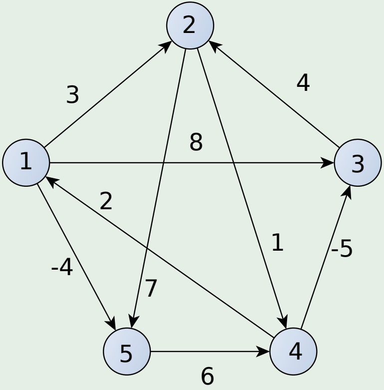
$$O\left(V^4\right) \tag{1}$$

Outline

Cinvestav

Example

We have the following

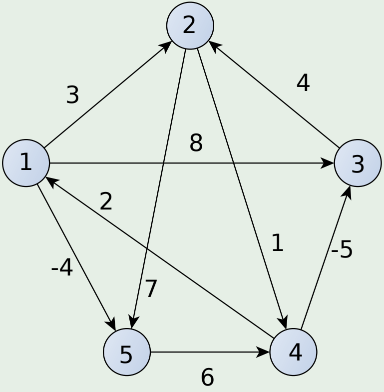


	1	2	3	4	5
1	0	2	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

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

Example

We have the following

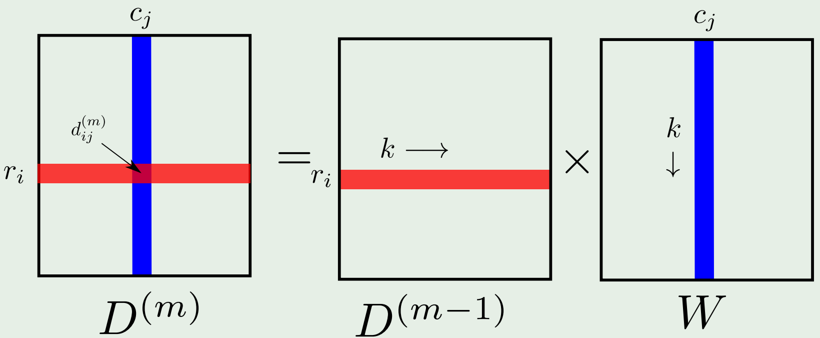


	1	2	3	4	5
1	0	2	8	2	-4
2	3	0	-4	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	8	∞	1	6	0

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

Here, we use the analogy of matrix multiplication

$D^1 W$



Thus, the update of an element l_{ij}

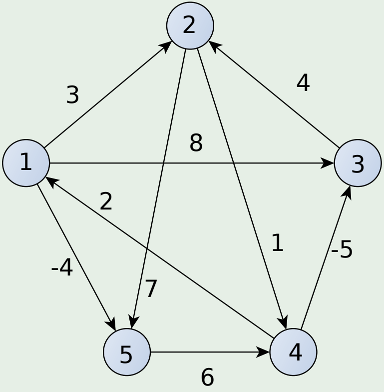
Example

$$
\begin{aligned}
l_{14}^{(2)} &= \min\left\{ \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \end{pmatrix} + \begin{pmatrix} \infty \\ 1 \\ \infty \\ 0 \\ 6 \end{pmatrix} \right\} \\
&= \min\begin{pmatrix} \infty & 4 & \infty & \infty & 2 \end{pmatrix} \\
&= 2
\end{aligned}
$$

Cinvestav

Example

We have the following

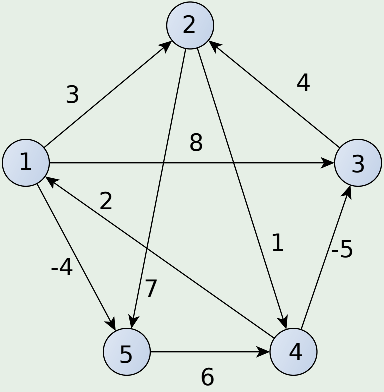


	1	2	3	4	5
1	0	3	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	11
4	2	-1	-5	0	-2
5	8	5	1	6	0

$L^{(3)} = L^{(2)}W$

Example

We have the following



	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$$L^{(4)} = L^{(3)}W$$

Outline

Cinvestav

Recall the following

We are interested only

In matrix $L^{(n-1)}$

In addition

Remember, we do not have negative weight cycles!!!

Therefore, given the equation

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n)} = \ldots \tag{2}$$

Cinvestav

Recall the following

We are interested only

In matrix $L^{(n-1)}$

In addition

Remember, we do not have negative weight cycles!!!

Therefore, given the equation

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n)} = \dots \qquad (2)$$

Recall the following

We are interested only

In matrix $L^{(n-1)}$

In addition

Remember, we do not have negative weight cycles!!!

Therefore, given the equation

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n)} = \ldots \quad (2)$$

Thus

It implies

$$L^{(m)} = L^{(n-1)} \tag{3}$$

For all

$$m \geq n - 1 \tag{4}$$

Thus

It implies

$$L^{(m)} = L^{(n-1)} \tag{3}$$

For all

$$m \geq n - 1 \tag{4}$$

Something Faster

We want something faster!!! Observation!!!

$$L^{(1)} = \quad W$$

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

Something Faster

We want something faster!!! Observation!!!

$$
\begin{aligned}
L^{(1)} &= W \\
L^{(2)} &= W \cdot W = W^2 \\
L^{(4)} &= W^2 \cdot W^2 = W^4 \\
L^{(8)} &= W^4 \cdot W^4 = W^8 \\
&\vdots \\
L^{\left(2^{\lceil \log(n-1) \rceil}\right)} &= W^{\left\lceil 2^{\lceil \log(n-1) \rceil - 1} \right\rceil} \cdot W^{2^{\lceil \log(n-1) \rceil - 1}} = W^{2^{\lceil \log(n-1) \rceil}}
\end{aligned}
$$

Something Faster

We want something faster!!! Observation!!!

$$
\begin{aligned}
L^{(1)} &= W \\
L^{(2)} &= W \cdot W = W^2 \\
L^{(4)} &= W^2 \cdot W^2 = W^4 \\
L^{(8)} &= W^4 \cdot W^4 = W^8 \\
&\vdots \\
L^{\left(2^{\lceil \log(n-1) \rceil}\right)} &= W^{\left\lceil 2^{\lceil \log(n-1) \rceil - 1} \right\rceil} \cdot W^{2^{\lceil \log(n-1) \rceil - 1}} = W^{2^{\lceil \log(n-1) \rceil}}
\end{aligned}
$$

Because

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

Something Faster

We want something faster!!! Observation!!!

$$
\begin{aligned}
L^{(1)} &= W \\
L^{(2)} &= W \cdot W = W^2 \\
L^{(4)} &= W^2 \cdot W^2 = W^4 \\
L^{(8)} &= W^4 \cdot W^4 = W^8 \\
&\vdots \\
L^{\left(2^{\lceil \log(n-1) \rceil}\right)} &= W^{\left\lceil 2^{\lceil \log(n-1) \rceil - 1} \right\rceil} \cdot W^{2^{\lceil \log(n-1) \rceil - 1}} = W^{2^{\lceil \log(n-1) \rceil}}
\end{aligned}
$$

Because

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

Something Faster

We want something faster!!! Observation!!!

$$
\begin{aligned}
L^{(1)} &= & W & \\
L^{(2)} &= & W \cdot W = & \quad W^2 \\
L^{(4)} &= & W^2 \cdot W^2 = & \quad W^4 \\
L^{(8)} &= & W^4 \cdot W^4 = & \quad W^8 \\
& & \vdots & \\
L^{\left(2^{\lceil \log(n-1) \rceil}\right)} &= & W^{\left\lceil 2^{\lceil \log(n-1) \rceil - 1} \right\rceil} \cdot W^{2^{\lceil \log(n-1) \rceil - 1}} = & \quad W^{2^{\lceil \log(n-1) \rceil}}
\end{aligned}
$$

Because

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

Something Faster

We want something faster!!! Observation!!!

$$
\begin{aligned}
L^{(1)} &= & W & \\
L^{(2)} &= & W \cdot W = & \quad W^2 \\
L^{(4)} &= & W^2 \cdot W^2 = & \quad W^4 \\
L^{(8)} &= & W^4 \cdot W^4 = & \quad W^8 \\
& & \vdots & \\
L^{\left(2^{\lceil \log(n-1) \rceil}\right)} &= & W^{\left\lceil 2^{\lceil \log(n-1) \rceil - 1} \right\rceil} \cdot W^{2^{\lceil \log(n-1) \rceil - 1}} = & \quad W^{2^{\lceil \log(n-1) \rceil}}
\end{aligned}
$$

Because

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

The Faster Algorithm

Complexity of the Previous Algorithm

Slow-All-Pairs-Shortest-Paths(W **)**

$n \leftarrow W.rows$

$L^{(1)} \leftarrow W$

$m \leftarrow 1$

while $m < n - 1$

    $L^{(2m)} \leftarrow$ **EXTEND-SHORTEST-PATHS** $\left(L^{(m)}, L^{(m)}\right)$

    $m \leftarrow 2m$

return $L^{(m)}$

Complexity

If $n = |V|$ we have that $O\left(V^3 \lg V\right)$.

The Faster Algorithm

Complexity of the Previous Algorithm

Slow-All-Pairs-Shortest-Paths(W **)**

1. $n \leftarrow W.rows$
2. $L^{(1)} \leftarrow W$
3. $m \leftarrow 1$
4. **while** $m < n - 1$
5. $\quad L^{(2m)} \leftarrow$ **EXTEND-SHORTEST-PATHS** $\left(L^{(m)}, L^{(m)}\right)$
6. $\quad m \leftarrow 2m$
7. **return** $L^{(m)}$

Complexity

If $n = |V|$ we have that $O\left(V^3 \lg V\right)$.

Outline

The Shortest Path Structure

Intermediate Vertex

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

Define

$d_{ij}^{(k)}$ =weight of a shortest path between i and j with all intermediate vertices are in the set $\{1, 2, ..., k\}$.

The Shortest Path Structure

Intermediate Vertex

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

Define

$d_{ij}^{(k)}$ =weight of a shortest path between i and j with all intermediate vertices are in the set $\{1, 2, ..., k\}$.

Cinvestav

The Recursive Idea

Simply look at the following cases cases

- **Case I** k is not an intermediate vertex, then a shortest path from i to j with all intermediate vertices $\{1, ..., k-1\}$ is a shortest path from i to j with intermediate vertices $\{1, ..., k\}$.

$$\Longrightarrow d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- **Case II** if k is an intermediate vertice. Then, $i \overset{p1}{\rightsquigarrow} k \overset{p2}{\rightsquigarrow} j$ and we can make the following statements using Lemma 24.1:
  - p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, ..., k-1\}$.
  - p_2 is a shortest path from k to j with all intermediate vertices in the set $\{1, ..., k-1\}$.

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

The Recursive Idea

Simply look at the following cases cases

- **Case I** k is not an intermediate vertex, then a shortest path from i to j with all intermediate vertices $\{1, ..., k-1\}$ is a shortest path from i to j with intermediate vertices $\{1, ..., k\}$.

$$\Longrightarrow d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- **Case II** if k is an intermediate vertice. Then, $i \overset{p_1}{\leadsto} k \overset{p_2}{\leadsto} j$ and we can make the following statements using Lemma 24.1:
  - p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, ..., k-1\}$.
  - p_2 is a shortest path from k to j with all intermediate vertices in the set $\{1, ..., k-1\}$.

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

The Recursive Idea

Simply look at the following cases cases

- **Case I** k is not an intermediate vertex, then a shortest path from i to j with all intermediate vertices $\{1, ..., k-1\}$ is a shortest path from i to j with intermediate vertices $\{1, ..., k\}$.

$$\Longrightarrow d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- **Case II** if k is an intermediate vertice. Then, $i \overset{p_1}{\leadsto} k \overset{p_2}{\leadsto} j$ and we can make the following statements using Lemma 24.1:
  - p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, ..., k-1\}$.
  - p_2 is a shortest path from k to j with all intermediate vertices in the set $\{1, ..., k-1\}$.

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

The Recursive Idea

Simply look at the following cases cases

- **Case I** k is not an intermediate vertex, then a shortest path from i to j with all intermediate vertices $\{1, ..., k-1\}$ is a shortest path from i to j with intermediate vertices $\{1, ..., k\}$.

$$\Longrightarrow d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- **Case II** if k is an intermediate vertice. Then, $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$ and we can make the following statements using Lemma 24.1:
  - p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, ..., k-1\}$.
  - p_2 is a shortest path from k to j with all intermediate vertices in the set $\{1, ..., k-1\}$.

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

The Recursive Idea

Simply look at the following cases cases

- **Case I** k is not an intermediate vertex, then a shortest path from i to j with all intermediate vertices $\{1, ..., k-1\}$ is a shortest path from i to j with intermediate vertices $\{1, ..., k\}$.

$$\Longrightarrow d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- **Case II** if k is an intermediate vertice. Then, $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$ and we can make the following statements using Lemma 24.1:
  - p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, ..., k-1\}$.
  - p_2 is a shortest path from k to j with all intermediate vertices in the set $\{1, ..., k-1\}$.

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

The Recursive Idea

Simply look at the following cases cases

- **Case I** k is not an intermediate vertex, then a shortest path from i to j with all intermediate vertices $\{1, ..., k-1\}$ is a shortest path from i to j with intermediate vertices $\{1, ..., k\}$.

$$\Longrightarrow d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- **Case II** if k is an intermediate vertice. Then, $i \overset{p1}{\leadsto} k \overset{p2}{\leadsto} j$ and we can make the following statements using Lemma 24.1:
  - p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, ..., k-1\}$.
  - p_2 is a shortest path from k to j with all intermediate vertices in the set $\{1, ..., k-1\}$.

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

The Graphical Idea

Consider

All possible intermediate vertices in $\{1, 2, ..., k\}$

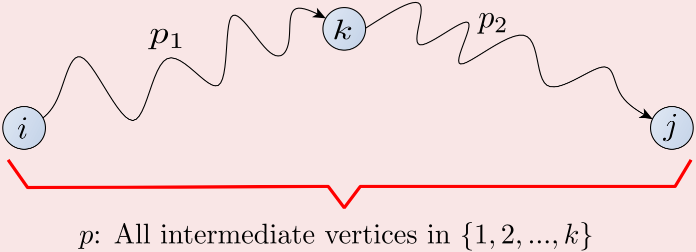


Figure: The Recursive Idea

The Recursive Solution

The Recursion

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

Final answer when $k = n$

We recursively calculate $D^{(n)} = \left(d_{ij}^{(n)}\right)$ or $d_{ij}^{(n)} = \delta(i,j)$ for all $i, j \in V$.

The Recursive Solution

The Recursion

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

Final answer when $k = n$

We recursively calculate $D^{(n)} = \left(d_{ij}^{(n)}\right)$ or $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$.

Cinvestav

Thus, we have the following

Recursive Version

Recursive-Floyd-Warshall(W)

1. $D^{(n)}$ the $n \times n$ matrix
2. for $i = 1$ to n
3. for $j = 1$ to n
4. $D^{(n)}[i, j] =$Recursive-Part(i, j, n, W)
5. return $D^{(n)}$

Thus, we have the following

Recursive Version

Recursive-Floyd-Warshall(W)

$D^{(n)}$ the $n \times n$ matrix

for $i = 1$ to n

    for $j = 1$ to n

        $D^{(n)}[i, j] =$Recursive-Part(i, j, n, W)

return $D^{(n)}$

Thus, we have the following

Recursive Version

Recursive-Floyd-Warshall(W)

1. $D^{(n)}$ the $n \times n$ matrix
2. for $i = 1$ to n
3. for $j = 1$ to n
4. $D^{(n)}[i,j] =$Recursive-Part(i, j, n, W)
5. return $D^{(n)}$

Thus, we have the following

The Recursive-Part

Recursive-Part(i, j, k, W)

1. if $k = 0$
2. return $W[i, j]$
3. if $k \geq 1$
4. t_1 =Recursive-Part$(i, j, k-1, W)$
5. t_2 =Recursive-Part$(i, k, k-1, W)$+...
6. Recursive-Part$(k, j, k-1, W)$
7. if $t_1 \leq t_2$
8. return t_1
9. else
10. return t_2

Thus, we have the following

The Recursive-Part

Recursive-Part(i, j, k, W)

if $k = 0$

  return $W[i, j]$

if $k \geq 1$

  $t_1 =$Recursive-Part$(i, j, k-1, W)$

  $t_2 =$Recursive-Part$(i, k, k-1, W)+$...

    Recursive-Part$(k, j, k-1, W)$

  if $t_1 \leq t_2$

    return t_1

  else

    return t_2

Thus, we have the following

The Recursive-Part

Recursive-Part(i, j, k, W)

if $k = 0$
    return $W[i, j]$
if $k \geq 1$
    $t_1 =$Recursive-Part$(i, j, k-1, W)$
    $t_2 =$Recursive-Part$(i, k, k-1, W)+$...
            Recursive-Part$(k, j, k-1, W)$
    if $t_1 \leq t_2$
        return t_1
    else
        return t_2

Thus, we have the following

The Recursive-Part

Recursive-Part(i, j, k, W)

```
if k = 0
        return W[i, j]
if k ≥ 1
        t_1 = Recursive-Part(i, j, k − 1, W)
        t_2 = Recursive-Part(i, k, k − 1, W)+...
                    Recursive-Part(k, j, k − 1, W)
        if t_1 ≤ t_2
                return t_1
        else
                return t_2
```

Outline

Cinvestav

Now

We want to use a storage to eliminate the recursion

For this, we are going to use two matrices

1. $D^{(k-1)}$ the previous matrix.
2. $D^{(k)}$ the new matrix based in the previous matrix

Now

We want to use a storage to eliminate the recursion

For this, we are going to use two matrices

1. $D^{(k-1)}$ the previous matrix.
2. $D^{(k)}$ the new matrix based in the previous matrix

Something Notable

With $D^{(0)} = W$ or all weights in the edges that exist.

Now

We want to use a storage to eliminate the recursion

For this, we are going to use two matrices

1. $D^{(k-1)}$ the previous matrix.
2. $D^{(k)}$ the new matrix based in the previous matrix

Something Notable

With $D^{(0)} = W$ or all weights in the edges that exist.

Now

We want to use a storage to eliminate the recursion

For this, we are going to use two matrices

1. $D^{(k-1)}$ the previous matrix.
2. $D^{(k)}$ the new matrix based in the previous matrix

Something Notable

With $D^{(0)} = W$ or all weights in the edges that exist.

In addition, we want to rebuild the answer

For this, we have the predecessor matrix Π

Actually, we want to compute a sequence of matrices

$$\Pi^{(0)}, \Pi^{(1)}, ..., \Pi^{(n)}$$

Where

$$\Pi = \Pi^{(n)}$$

In addition, we want to rebuild the answer

For this, we have the predecessor matrix Π

Actually, we want to compute a sequence of matrices

$$\Pi^{(0)}, \Pi^{(1)}, ..., \Pi^{(n)}$$

Where

$$\Pi = \Pi^{(n)}$$

What are the elements in $\Pi^{(k)}$

Each element in the matrix is as follow

$\pi_{ij}^{(k)}$ = the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, ..., k\}$

Thus, we have that

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

What are the elements in $\Pi^{(k)}$

Each element in the matrix is as follow

$\pi_{ij}^{(k)}$ = the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, ..., k\}$

Thus, we have that

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

Then

We have the following

For $k \geq 1$, if we take the path $i \rightsquigarrow k \rightsquigarrow j$ where $k \neq j$.

Then, if $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

For the predecessor of j, we chose k on a shortest path from k with all intermediate vertices in the set $\{1, 2, ..., k-1\}$

Otherwise, if $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

We choose the same predecessor of j that we chose on a shortest path from i with all all intermediate vertices in the set $\{1, 2, ..., k-1\}$.

Then

We have the following

For $k \geq 1$, if we take the path $i \rightsquigarrow k \rightsquigarrow j$ where $k \neq j$.

Then, if $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

For the predecessor of j, we chose k on a shortest path from k with all intermediate vertices in the set $\{1, 2, ..., k-1\}$

Otherwise, if $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

We choose the same predecessor of j that we chose on a shortest path from i with all all intermediate vertices in the set $\{1, 2, ..., k-1\}$.

Then

We have the following

For $k \geq 1$, if we take the path $i \rightsquigarrow k \rightsquigarrow j$ where $k \neq j$.

Then, if $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

For the predecessor of j, we chose k on a shortest path from k with all intermediate vertices in the set $\{1, 2, ..., k-1\}$

Otherwise, if $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

We choose the same predecessor of j that we chose on a shortest path from i with all all intermediate vertices in the set $\{1, 2, ..., k-1\}$.

Formally

We have then

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

Outline

Cinvestav

Final Iterative Version of Floyd-Warshall (Correction by Diego - Class Tec 2015)

Floyd-Warshall(W **)**

```
n = W.rows
D^(0) = W
for k = 1 to n − 1
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    let Π^(k) be a new predecessor n × n matrix
    ▷ Given each k, we update using D^(k−1)
    for i = 1 to n
        for j = 1 to n
            if d_ij^(k−1) ≤ d_ik^(k−1) + d_kj^(k−1)
                d_ij^(k) = d_ij^(k−1)
                π_ij^(k) = π_ij^(k−1)
            else
                d_ij^(k) = d_ik^(k−1) + d_kj^(k−1)
                π_ij^(k) = π_kj^(k−1)
return D^(n) and Π^(n)
```

Cinvestav

Final Iterative Version of Floyd-Warshall (Correction by Diego - Class Tec 2015)

Floyd-Warshall(W **)**

1. $n = W.rows$
2. $D^{(0)} = W$
3. **for** $k = 1$ **to** $n - 1$
4. let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
5. let $\Pi^{(k)}$ be a new predecessor $n \times n$ matrix

▷ Given each k, we update using $D^{(k-1)}$

6. **for** $i = 1$ **to** n
7. **for** $j = 1$ **to** n
8. **if** $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
9. $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
10. $\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$
11. **else**
12. $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
13. $\pi_{ij}^{(k)} = \pi_{kj}^{(k-1)}$
14. **return** $D^{(n)}$ and $\Pi^{(n)}$

Floyd-Warshall(W **)**

$n = W.rows$

$D^{(0)} = W$

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

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

  let $\Pi^{(k)}$ be a new predecessor $n \times n$ matrix

  ▷ Given each k, we update using $D^{(k-1)}$

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

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

      if $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

        $d_{ij}^{(k)} = d_{ij}^{(k-1)}$

        $\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$

      else

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

        $\pi_{ij}^{(k)} = \pi_{kj}^{(k-1)}$

return $D^{(n)}$ **and** $\Pi^{(n)}$

Floyd-Warshall(W **)**

```
n = W.rows
D^(0) = W
for k = 1 to n − 1
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    let Π^(k) be a new predecessor n × n matrix
    ▷ Given each k, we update using D^(k−1)
    for i = 1 to n
        for j = 1 to n
            if d_ij^(k−1) ≤ d_ik^(k−1) + d_kj^(k−1)
                d_ij^(k) = d_ij^(k−1)
                π_ij^(k) = π_ij^(k−1)
            else
                d_ij^(k) = d_ik^(k−1) + d_kj^(k−1)
                π_ij^(k) = π_kj^(k−1)
return D^(n) and Π^(n)
```

Explanation

Lines 1 and 2

Initialization of variables n and $D^{(0)}$

Explanation

Lines 1 and 2

Initialization of variables n and $D^{(0)}$

Line 3

In the loop, we solve the smaller problems first with $k = 1$ to $k = n - 1$

- Remember the largest number of edges in any shortest path

Explanation

Lines 1 and 2

Initialization of variables n and $D^{(0)}$

Line 3

In the loop, we solve the smaller problems first with $k = 1$ to $k = n - 1$

- Remember the largest number of edges in any shortest path

Line 4 and 5

Instantiation of the new matrices $D^{(k)}$ and $\prod^{(k)}$ to generate the shortest pats with at least k edges.

Explanation

Lines 1 and 2

Initialization of variables n and $D^{(0)}$

Line 3

In the loop, we solve the smaller problems first with $k = 1$ to $k = n - 1$

- Remember the largest number of edges in any shortest path

Line 4 and 5

Instantiation of the new matrices $D^{(k)}$ and $\prod^{(k)}$ to generate the shortest pats with at least k edges.

Cinvestav

Explanation

Line 6 and 7

This is done to go through all the possible combinations of i's and j's

Line 8

Deciding if $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Explanation

Line 6 and 7

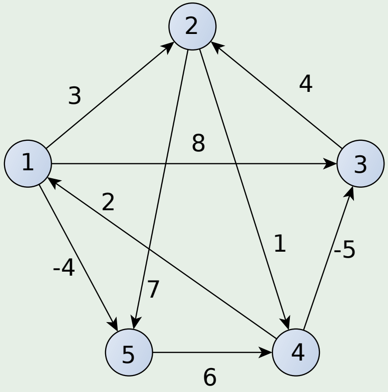
This is done to go through all the possible combinations of i's and j's

Line 8

Deciding if $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

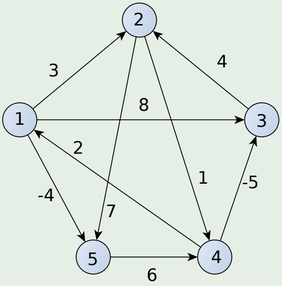
Example

Graph



Example

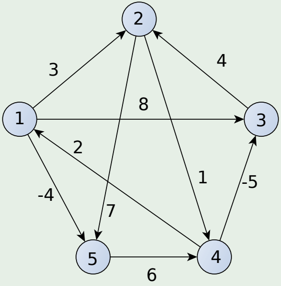
$D^{(0)}$ and $\Pi^{(0)}$



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

Example

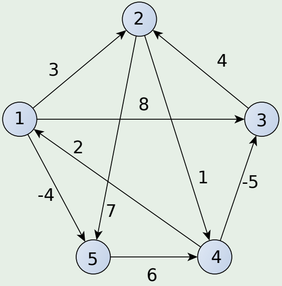
$D^{(1)}$ and $\Pi^{(1)}$



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

Example

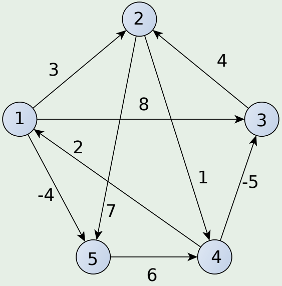
$D^{(2)}$ and $\Pi^{(2)}$



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

Example

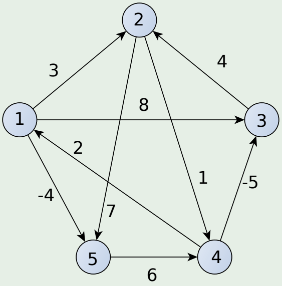
$D^{(3)}$ and $\Pi^{(3)}$



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

Example

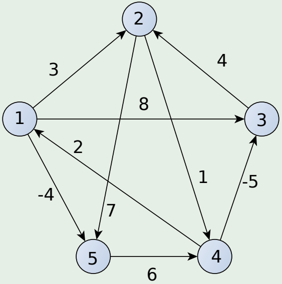
$D^{(4)}$ and $\Pi^{(4)}$



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

Example

$D^{(5)}$ and $\Pi^{(5)}$



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

Remarks

Something Notable

Because the comparison in line 8 takes $O(1)$

Remarks

Something Notable

Because the comparison in line 8 takes $O(1)$

Complexity of Floyd-Warshall is

Time Complexity $\Theta\left(V^3\right)$

- Making the Floyd-Warshall Algorithm practical even with moderate-sized graphs!!!

Remarks

Something Notable

Because the comparison in line 8 takes $O(1)$

Complexity of Floyd-Warshall is

Time Complexity $\Theta\left(V^3\right)$

We do not have elaborate data structures as Binary Heap or Fibonacci Heap!!!

The hidden constant time is quite small:

- Making the Floyd-Warshall Algorithm practical even with moderate-sized graphs!!!

Remarks

Something Notable

Because the comparison in line 8 takes $O(1)$

Complexity of Floyd-Warshall is

Time Complexity $\Theta(V^3)$

We do not have elaborate data structures as Binary Heap or Fibonacci Heap!!!

The hidden constant time is quite small:

- Making the Floyd-Warshall Algorithm practical even with moderate-sized graphs!!!

Outline

Cinvestav

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

  - A shortest path by w is a shortest path by $\hat{w}$
  - All edges are not negative using $\hat{w}$

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

What we require

- **A re-weighting function** $\widehat{w}(u, v)$
  - A shortest path by w is a shortest path by $\widehat{w}$.
  - All edges are not negative using $\widehat{w}$.

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

What we require

- **A re-weighting function** $\widehat{w}(u, v)$
  - A shortest path by w is a shortest path by $\widehat{w}$.
  - All edges are not negative using $\widehat{w}$.

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

What we require

- **A re-weighting function** $\widehat{w}(u, v)$
  - A shortest path by w is a shortest path by $\widehat{w}$.
  - All edges are not negative using $\widehat{w}$.

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

What we require

- **A re-weighting function** $\widehat{w}(u, v)$
  - A shortest path by w is a shortest path by $\widehat{w}$.
  - All edges are not negative using $\widehat{w}$.

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

What we require

- **A re-weighting function** $\widehat{w}(u,v)$
  - A shortest path by w is a shortest path by $\widehat{w}$.
  - All edges are not negative using $\widehat{w}$.

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

What we require

- **A re-weighting function** $\widehat{w}(u, v)$
  - A shortest path by w is a shortest path by $\widehat{w}$.
  - All edges are not negative using $\widehat{w}$.

Johnson's Algorithm

Observations

- Used to find all pairs in a sparse graphs by using Dijkstra's algorithm.
- It uses a re-weighting function to obtain positive edges from negative edges to deal with them.
- It can deal with the negative weight cycles.

Therfore

- It uses something to deal with the negative weight cycles.
  - Could be a Bellman-Ford detector as before?
- Maybe, we need to transform the weights in order to use them.

What we require

- **A re-weighting function** $\widehat{w}(u, v)$
  - A shortest path by w is a shortest path by $\widehat{w}$.
  - All edges are not negative using $\widehat{w}$.

Proving Properties of Re-Weigthing

Lemma 25.1

Given a weighted, directed graph $G = (D, V)$ 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

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

Let $p = \langle v_0, v_1, ..., v_k \rangle$ be any path from vertex 0 to vertex k. Then:

1. p is a shortest path from 0 to k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$.
2. Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.

Proving Properties of Re-Weigthing

Lemma 25.1

Given a weighted, directed graph $G = (D, V)$ 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

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

Let $p = \langle v_0, v_1, ..., v_k \rangle$ be any path from vertex 0 to vertex k. Then:

1. p is a shortest path from 0 to k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$.
2. Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.

Proving Properties of Re-Weigthing

Lemma 25.1

Given a weighted, directed graph $G = (D, V)$ 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

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

Let $p = \langle v_0, v_1, ..., v_k \rangle$ be any path from vertex 0 to vertex k. Then:

1. p is a shortest path from 0 to k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$.
2. Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.

Proving Properties of Re-Weigthing

Lemma 25.1

Given a weighted, directed graph $G = (D, V)$ 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

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

Let $p = \langle v_0, v_1, ..., v_k \rangle$ be any path from vertex 0 to vertex k. Then:

1. p is a shortest path from 0 to k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$.
2. Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.

Proving Properties of Re-Weigthing

Lemma 25.1

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

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

Let $p = \langle v_0, v_1, ..., v_k \rangle$ be any path from vertex 0 to vertex k. Then:

1. p is a shortest path from 0 to k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$.
2. Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.

Using This Lemma

Select h

Such that $w(u,v) + h(u) - h(v) \geq 0$.

- $V' = V \cup \{s\}$, where s is a new vertex.
- $E' = E \cup \{(s,v) | v \in V\}$.
- $w(s,v) = 0$ for all $v \in V$, in addition to all the other weights.

Using This Lemma

Select h

Such that $w(u,v) + h(u) - h(v) \geq 0$.

Then, we build a new graph G'

- It has the following elements
  - $V' = V \cup \{s\}$, where s is a new vertex.
  - $E' = E \cup \{(s,v) | v \in V\}$.
  - $w(s,v) = 0$ for all $v \in V$, in addition to all the other weights.

Using This Lemma

Select h

Such that $w(u,v) + h(u) - h(v) \geq 0$.

Then, we build a new graph G'

- It has the following elements
  - $V' = V \cup \{s\}$, **where s is a new vertex.**
  - $E' = E \cup \{(s,v) \mid v \in V\}$.
  - $w(s,v) = 0$ for all $v \in V$, in addition to all the other weights.

Select h

Simply select $h(v) = \delta(s,v)$.

Using This Lemma

Select h

Such that $w(u,v) + h(u) - h(v) \geq 0$.

Then, we build a new graph G'

- It has the following elements
  - $V' = V \cup \{s\}$, **where s is a new vertex.**
  - $E' = E \cup \{(s,v) \mid v \in V\}$.
  - $w(s,v) = 0$ for all $v \in V$, in addition to all the other weights.

Select h

Simply select $h(v) = \delta(s,v)$.

Using This Lemma

Select h

Such that $w(u, v) + h(u) - h(v) \geq 0$.

Then, we build a new graph G'

- It has the following elements
  - $V' = V \cup \{s\}$, **where s is a new vertex.**
  - $E' = E \cup \{(s, v) \, | v \in V\}$.
  - $w(s, v) = 0$ for all $v \in V$, in addition to all the other weights.

Select h

Simply select $h(v) = \delta(s, v)$.

Using This Lemma

Select h

Such that $w(u,v) + h(u) - h(v) \geq 0$.

Then, we build a new graph G'

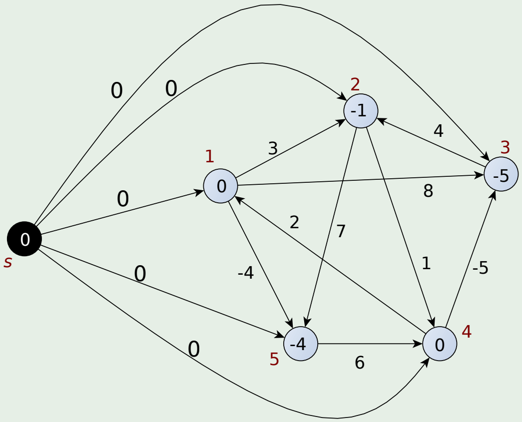
- It has the following elements
  - $V' = V \cup \{s\}$, **where s is a new vertex.**
  - $E' = E \cup \{(s,v) | v \in V\}$.
  - $w(s,v) = 0$ for all $v \in V$, in addition to all the other weights.

Select h

Simply select $h(v) = \delta(s,v)$.

Example

Graph G' with original weight function w with new source s and $h(v) = \delta(s, v)$ at each vertex



Proof of Claim

Claim

$$w(u, v) + h(u) - h(v) \geq 0 \quad (5)$$

By Triangle Inequality

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

Proof of Claim

Claim

$$w(u,v) + h(u) - h(v) \geq 0 \quad (5)$$

By Triangle Inequality

- $\delta(s,v) \leq \delta(s,u) + w(u,v)$
- Then by the way we selected h, we have:

$$h(v) \leq h(u) + w(u,v) \quad (6)$$

Proof of Claim

Claim

$$w\left(u, v\right) + h\left(u\right) - h\left(v\right) \geq 0 \tag{5}$$

By Triangle Inequality

- $\delta\left(s, v\right) \leq \delta(s, u) + w(u, v)$
- Then by the way we selected h, we have:

$$h(v) \leq h(u) + w\left(u, v\right) \tag{6}$$

Finally

$$w\left(u, v\right) + h\left(u\right) - h\left(v\right) \geq 0 \tag{7}$$

Proof of Claim

Claim

$$w(u,v) + h(u) - h(v) \geq 0 \quad (5)$$

By Triangle Inequality

- $\delta(s,v) \leq \delta(s,u) + w(u,v)$
- Then by the way we selected h, we have:

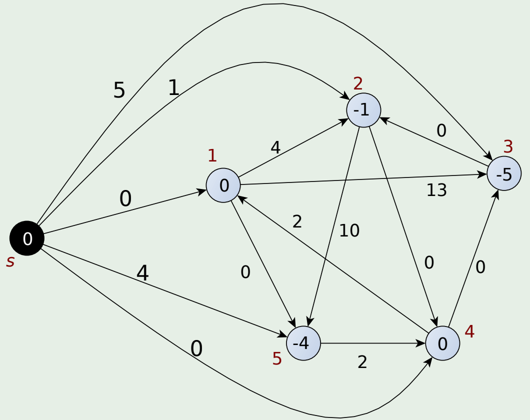
$$h(v) \leq h(u) + w(u,v) \quad (6)$$

Finally

$$w(u,v) + h(u) - h(v) \geq 0 \quad (7)$$

Example

The new Graph G after re-weighting G'



Final Algorithm

Pseudo-Code

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

If Bellman-Ford$(G', w, s) == FALSE$

    print "Graphs contains a Neg-Weight Cycle"

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

        set $h(v) = v.d$ computed by Bellman-Ford

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

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

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

    for each vertex $u \in G.V$

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

        for each vertex $v \in G.V$

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

    return D

Final Algorithm

Pseudo-Code

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

If Bellman-Ford$(G', w, s) == FALSE$

  print "Graphs contains a Neg-Weight Cycle"

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

    set $h(v) = v.d$ computed by Bellman-Ford

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

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

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

  for each vertex $u \in G.V$

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

    for each vertex $v \in G.V$

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

  return D

Final Algorithm

Pseudo-Code

1. Compute G', where: $G'.V = G.E \cup \{(s,v) \,|v \in G.V\}$ and $w(s,v) = 0$ for all $v \in G.V$
2. **If** Bellman-Ford$(G', w, s) == FALSE$
3.   **print** "Graphs contains a Neg-Weight Cycle"
4. **else for** each vertex $v \in G'.V$
5.   set $h(v) = v.d$ computed by Bellman-Ford
6.   **for** each edge $(u,v) \in G'.E$
7. $\widehat{w}(u,v) = w(u,v) + h(u) - h(v)$
8.   **Let** $D = (d_{uv})$ be a new $n \times n$ matrix
9.   **for** each vertex $u \in G.V$
10.   run Dijkstra$(G, \widehat{w}, u)$ to compute $\widehat{\delta}(u,v)$ for all $v \in G.V$
11.   **for** each vertex $v \in G.V$
12. $d_{uv} = \widehat{\delta}(u,v) + h(v) - h(u)$
13.   **return** D

Final Algorithm

Pseudo-Code

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

If Bellman-Ford$(G', w, s) == FALSE$

    print "Graphs contains a Neg-Weight Cycle"

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

        set $h(v) = v.d$ computed by Bellman-Ford

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

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

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

    for each vertex $u \in G.V$

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

        for each vertex $v \in G.V$

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

    return D

Final Algorithm

Pseudo-Code

1. Compute G', where: $G'.V = G.E \cup \{(s,v) \,|v \in G.V\}$ and$w(s,v) = 0$ for all $v \in G.V$
2. **If** Bellman-Ford$(G', w, s) == FALSE$
3. **print** "Graphs contains a Neg-Weight Cycle"
4. **else for** each vertex $v \in G'.V$
5. set $h(v) = v.d$ computed by Bellman-Ford
6. **for** each edge $(u,v) \in G'.E$
7. $\widehat{w}(u,v) = w(u,v) + h(u) - h(v)$
8. **Let** $D = (d_{uv})$ be a new $n \times n$ matrix
9. **for** each vertex $u \in G.V$
10. run Dijkstra$(G, \widehat{w}, u)$ to compute $\widehat{\delta}(u,v)$ for all $v \in G.V$
11. **for** each vertex $v \in G.V$
12. $d_{uv} = \widehat{\delta}(u,v) + h(v) - h(u)$
13. **return** D

Final Algorithm

Pseudo-Code

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

If Bellman-Ford$(G', w, s) == FALSE$

    print "Graphs contains a Neg-Weight Cycle"

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

        set $h(v) = v.d$ computed by Bellman-Ford

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

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

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

    for each vertex $u \in G.V$

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

        for each vertex $v \in G.V$

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

    return D

Final Algorithm

Pseudo-Code

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

If Bellman-Ford$(G', w, s) == FALSE$

  print "Graphs contains a Neg-Weight Cycle"

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

    set $h(v) = v.d$ computed by Bellman-Ford

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

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

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

  for each vertex $u \in G.V$

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

    for each vertex $v \in G.V$

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

  return D

Final Algorithm

Pseudo-Code

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

If Bellman-Ford$(G', w, s) == FALSE$

    print "Graphs contains a Neg-Weight Cycle"

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

        set $h(v) = v.d$ computed by Bellman-Ford

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

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

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

    for each vertex $u \in G.V$

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

        for each vertex $v \in G.V$

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

    return D

Complexity

The Final Complexity

- Times:
  - $\Theta(V+E)$ to compute G'
  - $O(VE)$ to run Bellman-Ford
  - $\Theta(E)$ to compute $\widehat{w}$
  - $O\left(V^2 \lg V + VE\right)$ to run Dijkstra's algorithm $|V|$ time using Fibonacci Heaps
  - $O\left(V^2\right)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O\left(V^2\right) \implies O\left(V^3\right)$

Cinvestav

Complexity

The Final Complexity

- Times:
  - $\Theta(V + E)$ to compute G'
  - $O(VE)$ to run Bellman-Ford
  - $\Theta(E)$ to compute $\widehat{w}$
  - $O\left(V^2 \lg V + VE\right)$ to run Dijkstra's algorithm $|V|$ time using Fibonacci Heaps
  - $O\left(V^2\right)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O\left(V^2\right) \implies O\left(V^3\right)$

Cinvestav

Complexity

The Final Complexity

- Times:
  - $\Theta(V+E)$ 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 $|V|$ time using Fibonacci Heaps
  - $O(V^2)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O(V^2) \Longrightarrow O(V^3)$

Cinvestav

Complexity

The Final Complexity

- Times:
  - $\Theta(V + E)$ 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 $|V|$ time using Fibonacci Heaps
  - $O(V^2)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O(V^2) \implies O(V^3)$

Complexity

The Final Complexity

- Times:
  - $\Theta(V+E)$ to compute G'
  - $O(VE)$ to run Bellman-Ford
  - $\Theta(E)$ to compute $\widehat{w}$
  - $O\left(V^2 \lg V + VE\right)$ to run Dijkstra's algorithm $|V|$ time using Fibonacci Heaps
  - $O\left(V^2\right)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O\left(V^2\right) \implies O\left(V^3\right)$

Complexity

The Final Complexity

- Times:
  - $\Theta(V + E)$ to compute G'
  - $O(VE)$ to run Bellman-Ford
  - $\Theta(E)$ to compute $\widehat{w}$
  - $O\left(V^2 \lg V + VE\right)$ to run Dijkstra's algorithm $|V|$ time using Fibonacci Heaps
  - $O\left(V^2\right)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O\left(V^2\right) \Longrightarrow O\left(V^3\right)$

Complexity

The Final Complexity

- Times:
  - $\Theta(V + E)$ to compute G'
  - $O(VE)$ to run Bellman-Ford
  - $\Theta(E)$ to compute $\widehat{w}$
  - $O\left(V^2 \lg V + VE\right)$ to run Dijkstra's algorithm $|V|$ time using Fibonacci Heaps
  - $O\left(V^2\right)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O\left(V^2\right) \implies O\left(V^3\right)$

Cinvestav

Complexity

The Final Complexity

- Times:
  - $\Theta(V + E)$ to compute G'
  - $O(VE)$ to run Bellman-Ford
  - $\Theta(E)$ to compute $\widehat{w}$
  - $O\left(V^2 \lg V + VE\right)$ to run Dijkstra's algorithm $|V|$ time using Fibonacci Heaps
  - $O\left(V^2\right)$ to compute D matrix
- Total : $O(V^2 \lg V + VE)$
- If $E = O\left(V^2\right) \implies O\left(V^3\right)$

Outline

Excercises

- 25.1-4
- 25.1-8
- 25.1-9
- 25.2-4
- 25.2-6
- 25.2-9
- 25.3-3