FLOYD-WARSHALL + BELLMAN-FORD ALGORITHMS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

1 Floyd-Warshall algorithm

2 Bellman-Ford algorithm

3 Bibliography

Centro de Informática UFPE

Universidade Federal de Pernambuco

Floyd-Warshall algorithm

Let G be a weighted graph, finds the shortest path between all pairs of nodes in G

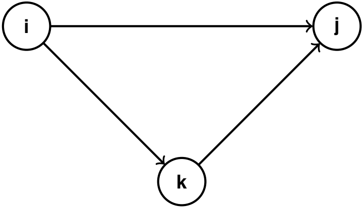
- All-pairs shortest paths

Floyd algorithm: can be used on weighted graphs with negative weights (but no negative cycles)

- Based on the Warshall algorithm for computing transitive closures

Design strategy: dynamic programming

Floyd-Warshall algorithm



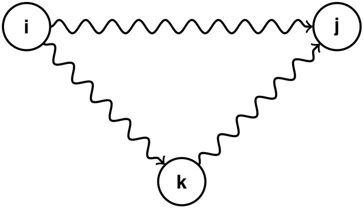
Before first iteration ($k = 0$)

- D[i][j]: direct distance between i and j

After first iteration ($k = 0$)

- D[i][j]: best option between $i \rightarrow j$ and $i \rightarrow 0 \rightarrow j$

Floyd-Warshall algorithm



Before second iteration ($k = 1$)

- D[i][j]: best option between $i \to j$ and $i \to 0 \to j$

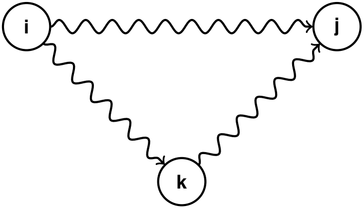
After second iteration ($k = 1$)

- D[i][j]: best option between $i \rightsquigarrow j$ and $i \rightsquigarrow 1 \rightsquigarrow j$
  - $i \rightsquigarrow j$ the best of $i \to j$, $i \to 0 \to j$
  - $i \rightsquigarrow 1$ the best of $i \to 1$, $i \to 0 \to 1$
  - $1 \rightsquigarrow j$ the best of $1 \to j$, $1 \to 0 \to j$
- Considering the best among $i \to j, i \to 0 \to j, i \to 1 \to j,$ $i \to 0 \to 1 \to j, i \to 1 \to 0 \to j$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Floyd-Warshall algorithm



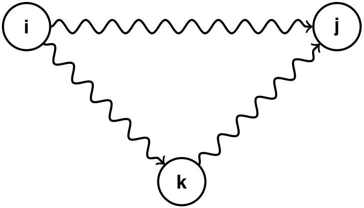
Before third iteration ($k = 2$)

- D[i][j]: best option between $i \rightsquigarrow j$ and $i \rightsquigarrow 1 \rightsquigarrow j$

After third iteration ($k = 2$)

- D[i][j]: best option between $i \rightsquigarrow j$ and $i \rightsquigarrow 2 \rightsquigarrow j$
  - $i \rightsquigarrow j$ the best of $i \to j$, $i \to 0 \to j$, $i \to 1 \to j$, $i \to 0 \to 1 \to j$, $i \to 1 \to 0 \to j$
  - $i \rightsquigarrow 2$ the best of $i \to 2$, $i \to 0 \to 2$, $i \to 1 \to 2$, $i \to 0 \to 1 \to 2$, $i \to 1 \to 0 \to 2$
  - $2 \rightsquigarrow j$ the best of ...

Floyd-Warshall algorithm



The algorithm stops when $k = n$

After the last iteration ($k = n - 1$)

- D[i][j]: best option between $i \rightsquigarrow j$ and $i \rightsquigarrow n - 1 \rightsquigarrow j$, which means the best among all possibilities

Floyd-Warshall algorithm

Algorithm: void Floyd(Graph G, int[][] D)

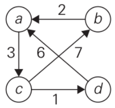
```
for i ← 0 to n(G) − 1 do
    for j ← 0 to n(G) − 1 do
        if i = j then D[i][j] ← 0;
        else if weight(G, i, j) ≠ 0 then D[i][j] ← weight(G, i, j);
        else D[i][j] ← ∞;
for k ← 0 to n(G) − 1 do
    for i ← 0 to n(G) − 1 do
        for j ← 0 to n(G) − 1 do
            if D[i][k] ≠ ∞ ∧ D[k][j] ≠ ∞ ∧ D[i][j] > D[i][k] + D[k][j] then
                D[i][j] ← D[i][k] + D[k][j];
```

Time efficiency in $\Theta(|V|^3)$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Floyd-Warshall algorithm[1]



$$D^{(0)} = \begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \\ \hline \end{array}$$

$$D^{(1)} = \begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \mathbf{5} & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \mathbf{9} & 0 \\ \hline \end{array}$$

$$D^{(2)} = \begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & \mathbf{9} & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \\ \hline \end{array}$$

$$D^{(3)} = \begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & \mathbf{10} & 3 & \mathbf{4} \\ b & 2 & 0 & 5 & \mathbf{6} \\ c & 9 & 7 & 0 & 1 \\ d & 6 & \mathbf{16} & 9 & 0 \\ \hline \end{array}$$

$$D^{(4)} = \begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & \mathbf{7} & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \\ \hline \end{array}$$

[1] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Agenda

1 Floyd-Warshall algorithm

2 Bellman-Ford algorithm

3 Bibliography

Bellman-Ford algorithm

Let G be a weighted graph and $v \in V$ (*source*), finds the shortest path from v to all other nodes in V

- Single-source shortest paths

Bellman-Ford algorithm: can be used on weighted graphs with negative cycles

Intuition:

- There can be maximum $|V| - 1$ edges in any simple path
- Processes all edges $|V| - 1$ times
- Iteration $k = 0$: all shortest paths which are at most 1 edge long
- Iteration $k = |V| - 2$: all shortest paths
- Iteration $k = |V| - 1$: if something better is found, there is a negative cycle

Centro de Informática UFPE

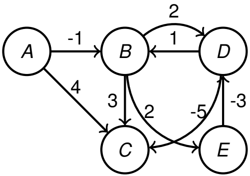
Universidade Federal de Pernambuco

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do** $D[i] \leftarrow \infty$;
$D[s] \leftarrow 0$;
for $k \leftarrow 0$ **to** $n(G) - 2$ **do**
    for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
        $j \leftarrow \mathit{first}(G, i)$;
        while $j < n(G)$ **do**
            if $D[j] > D[i] + \mathit{weight}(G, i, j)$ **then**
                $D[j] \leftarrow D[i] + \mathit{weight}(G, i, j)$;
            $j \leftarrow \mathit{next}(G, i, j)$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
    $j \leftarrow \mathit{first}(G, i)$;
    while $j < n(G)$ **do**
        if $D[j] > D[i] + \mathit{weight}(G, i, j)$ **then**
            negative cycle detected
        $j \leftarrow \mathit{next}(G, i, j)$;

Let $s = A$



	A	B	C	D	E
Distance	0	∞	∞	∞	∞

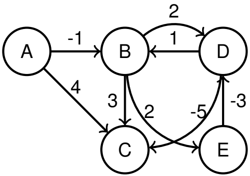
Centro de Informática UFPE
Universidade Federal de Pernambuco

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do** $D[i] \leftarrow \infty$;
$D[s] \leftarrow 0$;
for $k \leftarrow 0$ **to** $n(G) - 2$ **do**
    for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
        $j \leftarrow first(G, i)$;
        while $j < n(G)$ **do**
            if $D[j] > D[i] + weight(G, i, j)$ **then**
                $D[j] \leftarrow D[i] + weight(G, i, j)$;
            $j \leftarrow next(G, i, j)$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
    $j \leftarrow first(G, i)$;
    while $j < n(G)$ **do**
        if $D[j] > D[i] + weight(G, i, j)$ **then**
            negative cycle detected
        $j \leftarrow next(G, i, j)$;

Let $s = A$



	A	B	C	D	E
Distance	0	-1	4	∞	∞

Considering $k = 0$

- After $i = 0$ (node A)

Centro de Informática UFPE

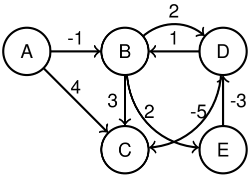
Universidade Federal de Pernambuco

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do  D[i] ← ∞ ;
D[s] ← 0;
for k ← 0 to n(G) − 2 do
    for i ← 0 to n(G) − 1 do
        j ← first(G, i);
        while j < n(G) do
            if D[j] > D[i] + weight(G, i, j) then
                D[j] ← D[i] + weight(G, i, j);
            j ← next(G, i, j);
for i ← 0 to n(G) − 1 do
    j ← first(G, i);
    while j < n(G) do
        if D[j] > D[i] + weight(G, i, j) then
            negative cycle detected
        j ← next(G, i, j);
```

Let $s = A$



	A	B	C	D	E
Distance	0	-1	2	1	1

Considering $k = 0$

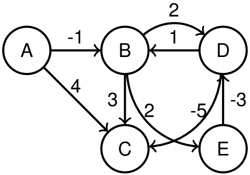
- After $i = 1$ (node B)

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do  D[i] ← ∞ ;
D[s] ← 0;
for k ← 0 to n(G) − 2 do
    for i ← 0 to n(G) − 1 do
        j ← first(G, i);
        while j < n(G) do
            if D[j] > D[i] + weight(G, i, j) then
                D[j] ← D[i] + weight(G, i, j);
            j ← next(G, i, j);
for i ← 0 to n(G) − 1 do
    j ← first(G, i);
    while j < n(G) do
        if D[j] > D[i] + weight(G, i, j) then
            negative cycle detected
        j ← next(G, i, j);
```

Let $s = A$



	A	B	C	D	E
Distance	0	-1	2	1	1

Considering $k = 0$

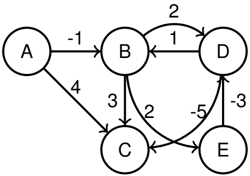
- After $i = 2$ (node C)

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do** $D[i] \leftarrow \infty$;
$D[s] \leftarrow 0$;
for $k \leftarrow 0$ **to** $n(G) - 2$ **do**
  for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
    $j \leftarrow first(G, i)$;
    while $j < n(G)$ **do**
      if $D[j] > D[i] + weight(G, i, j)$ **then**
        $D[j] \leftarrow D[i] + weight(G, i, j)$;
      $j \leftarrow next(G, i, j)$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  $j \leftarrow first(G, i)$;
  while $j < n(G)$ **do**
    if $D[j] > D[i] + weight(G, i, j)$ **then**
      negative cycle detected
    $j \leftarrow next(G, i, j)$;

Let $s = A$



	A	B	C	D	E
Distance	0	-1	-4	1	1

Considering $k = 0$

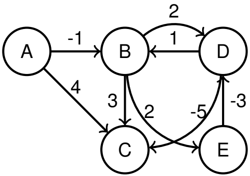
- After $i = 3$ (node D)

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do  D[i] ← ∞ ;
D[s] ← 0;
for k ← 0 to n(G) − 2 do
    for i ← 0 to n(G) − 1 do
        j ← first(G, i);
        while j < n(G) do
            if D[j] > D[i] + weight(G, i, j) then
                D[j] ← D[i] + weight(G, i, j);
            j ← next(G, i, j);
for i ← 0 to n(G) − 1 do
    j ← first(G, i);
    while j < n(G) do
        if D[j] > D[i] + weight(G, i, j) then
            negative cycle detected
        j ← next(G, i, j);
```

Let $s = A$



	A	B	C	D	E
Distance	0	-1	-4	-2	1

Considering $k = 0$

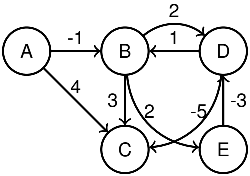
- After $i = 4$ (node E)

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do  D[i] ← ∞ ;
D[s] ← 0;
for k ← 0 to n(G) − 2 do
    for i ← 0 to n(G) − 1 do
        j ← first(G, i);
        while j < n(G) do
            if D[j] > D[i] + weight(G, i, j) then
                D[j] ← D[i] + weight(G, i, j);
            j ← next(G, i, j);
for i ← 0 to n(G) − 1 do
    j ← first(G, i);
    while j < n(G) do
        if D[j] > D[i] + weight(G, i, j) then
            negative cycle detected
        j ← next(G, i, j);
```

Let $s = A$



	A	B	C	D	E
Distance	0	-1	-7	-2	1

Considering $k = 1$

- After $i = 0..4$

Centro de Informática UFPE

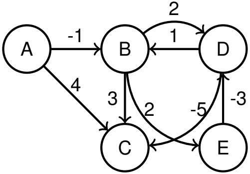
Universidade Federal de Pernambuco

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

```
for i ← 0 to n(G) − 1 do  D[i] ← ∞ ;
D[s] ← 0;
for k ← 0 to n(G) − 2 do
    for i ← 0 to n(G) − 1 do
        j ← first(G, i);
        while j < n(G) do
            if D[j] > D[i] + weight(G, i, j) then
                D[j] ← D[i] + weight(G, i, j);
            j ← next(G, i, j);
for i ← 0 to n(G) − 1 do
    j ← first(G, i);
    while j < n(G) do
        if D[j] > D[i] + weight(G, i, j) then
            negative cycle detected
        j ← next(G, i, j);
```

Let $s = A$



	A	B	C	D	E
Distance	0	-1	-7	-2	1

Considering $k = 2..3$

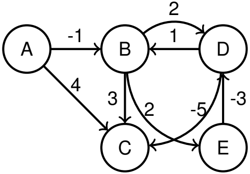
- Nothing changes

Bellman-Ford algorithm

Algorithm: void BellmanFord(Graph G, int s, int[] D)

for $i \leftarrow 0$ **to** $n(G) - 1$ **do** $D[i] \leftarrow \infty$;
$D[s] \leftarrow 0$;
for $k \leftarrow 0$ **to** $n(G) - 2$ **do**
  for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
    $j \leftarrow first(G, i)$;
    while $j < n(G)$ **do**
      if $D[j] > D[i] + weight(G, i, j)$ **then**
        $D[j] \leftarrow D[i] + weight(G, i, j)$;
      $j \leftarrow next(G, i, j)$;
for $i \leftarrow 0$ **to** $n(G) - 1$ **do**
  $j \leftarrow first(G, i)$;
  while $j < n(G)$ **do**
    if $D[j] > D[i] + weight(G, i, j)$ **then**
      negative cycle detected
    $j \leftarrow next(G, i, j)$;

Let $s = A$



	A	B	C	D	E
Distance	0	-1	-7	-2	1

One more loop

- Nothing changes, implies no neg. cycle

Centro de Informática UFPE

Universidade Federal de Pernambuco

Bellman-Ford algorithm

Another example of dynamic programming

Time efficiency in $\Theta(\mid V \mid\mid E \mid) = \Theta(\mid V \mid^3)$, since $\mid E \mid \in O(\mid V \mid^2)$

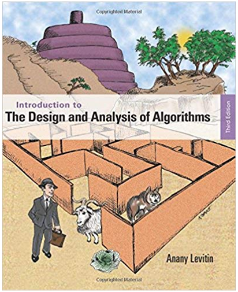
Agenda

1 Floyd-Warshall algorithm

2 Bellman-Ford algorithm

3 Bibliography

Bibliography



Chapter 9 (pp. 308–311)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3rd edition. Pearson. 2011.

Chapter 16 (pp. 513–515)
Clifford Shaffer.
Data Structures and Algorithm Analysis. Dover, 2013.

Centro de Informática UFPE

Universidade Federal de Pernambuco

FLOYD-WARSHALL + BELLMAN-FORD ALGORITHMS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil