All-Pairs Shortest Paths

(The Floyd-Warshall Algorithm)

Mr. Ajaya Kumar Dash

ajaya@iiit-bh.ac.in
https://dash-ajay.github.io/

Department of Computer Science and Engineering

IIIT, Bhubaneswar

April 4, 2019

Shortest Path Algorithms: Comparison

- Dijkstra's
  - Shortest path from **one** node to all other nodes

Shortest Path Algorithms: Comparison

- Dijkstra's
  - Shortest path from **one** node to all other nodes
- Bellman-Ford
  - Shortest path from **one** node to all other nodes
  - Negative edges **allowed**
  - Detect the presence of **negative weight cycle**

Shortest Path Algorithms: Comparison

- Dijkstra's
  - Shortest path from **one** node to all other nodes
- Bellman-Ford
  - Shortest path from **one** node to all other nodes
  - Negative edges **allowed**
  - Detect the presence of **negative weight cycle**
- Floyd-Warshall
  - Shortest path between **all** pair of vertices
  - Negative edges **allowed**

Problem Definition

Problem Definition

- Suppose we are given a directed graph $G = (V, E)$ and a weight function $w : E \rightarrow R$.

Problem Definition

- Suppose we are given a directed graph $G = (V, E)$ and a weight function $w : E \to R$.

- We assume that 'G' doesn't contain any negative weight cycle.

Problem Definition

- Suppose we are given a directed graph $G = (V, E)$ and a weight function $w : E \rightarrow R$.
- We assume that 'G' doesn't contain any negative weight cycle.
- The All-Pairs Shortest Path problem asks to find the length of the shortest path between any pair of vertices in 'G'.

Solutions Using Previous Knowledge

Solutions Using Previous Knowledge

- If the *weight function* is nonnegative for all edges, then we can use *Dijkstra's single source shortest path* algorithm for all vertices to solve problem.

Solutions Using Previous Knowledge

- If the *weight function* is nonnegative for all edges, then we can use *Dijkstra's single source shortest path* algorithm for all vertices to solve problem.

Complexity of Above Approach

Solutions Using Previous Knowledge

- If the *weight function* is nonnegative for all edges, then we can use *Dijkstra's single source shortest path* algorithm for all vertices to solve problem.

Complexity of Above Approach

- Linear array implementation of min-priority queue: $O(V^3 + VE)$

Solutions Using Previous Knowledge

- If the *weight function* is nonnegative for all edges, then we can use *Dijkstra's single source shortest path* algorithm for all vertices to solve problem.

Complexity of Above Approach

- Linear array implementation of min-priority queue: $O(V^3 + VE)$
- Fibonacci Heap implementation of min-priority queue: $O(V^2 logV + VE)$

Solutions Using Previous Knowledge (Cont...)

- If negative edge weights are allowed, we cannot use Dijkstra's method, rather we have to use Bellman-Ford algorithm for all vetices to solve the problem i.e. $|V|$ times.

- If negative edge weights are allowed, we cannot use Dijkstra's method, rather we have to use Bellman-Ford algorithm for all vetices to solve the problem i.e. $|V|$ times.

Complexity of this Approach

Solutions Using Previous Knowledge (Cont...)

- If negative edge weights are allowed, we cannot use Dijkstra's method, rather we have to use Bellman-Ford algorithm for all vetices to solve the problem i.e. $|V|$ times.

Complexity of this Approach

- General case: $O(V^2E)$

Solutions Using Previous Knowledge (Cont...)

- If negative edge weights are allowed, we cannot use Dijkstra's method, rather we have to use Bellman-Ford algorithm for all vetices to solve the problem i.e. $|V|$ times.

Complexity of this Approach

- General case: $\boldsymbol{O(V^2E)}$
- Dense Graph: $\boldsymbol{O(V^4)}$

Solutions Using Previous Knowledge (Cont...)

- If negative edge weights are allowed, we cannot use Dijkstra's method, rather we have to use Bellman-Ford algorithm for all vetices to solve the problem i.e. $|V|$ times.

Complexity of this Approach

- General case: $O(V^2E)$
- Dense Graph: $O(V^4)$

CAN WE DO BETTER ?

Floyd - Warshall Algorithm

Floyd - Warshall Algorithm

- Uses Dynamic Programming Approach.

Floyd - Warshall Algorithm

- Uses Dynamic Programming Approach.
- For a graph $G = (V, E)$, runs in $O(V^3)$ time.

Floyd - Warshall Algorithm

- Uses Dynamic Programming Approach.
- For a graph $G = (V, E)$, runs in $O(V^3)$ time.
- Uses Adjacency matrix representation of graph.

Floyd - Warshall Algorithm

- Uses Dynamic Programming Approach.
- For a graph $G = (V, E)$, runs in $O(V^3)$ time.
- Uses Adjacency matrix representation of graph.

Alert

Negative-weight edges may be present, but we assume that "no Negative weight cycles" ..

Representation of the Input

Representation of the Input

- The input is represented by a weight matrix

$$\boxed{W = (w_{ij})_{(i,j)\ in\ E}}$$

and is defined by,

Representation of the Input

- The input is represented by a weight matrix

$$\boldsymbol{W = (w_{ij})_{(i,j)\ in\ E}}$$

and is defined by,

$$\boldsymbol{w_{ij}} = \begin{cases} 0, & \text{if } i = j \\ w(i,j), & \text{if } i \neq j \text{ and} (i,j) \text{ in } E \\ \infty, & \text{if } i \neq j \text{ and} (i,j) \text{ not in } E \end{cases}$$

Format of the Output

Format of the Output

- If the graph has **V** vertices, we return a distance matrix **D**, where each element **(**d_{ij}**)** is the shortest length of the path from i to j.

Intermediate Vertices

Intermediate Vertices

- Without loss of generality, we'll assume that $V = \{1, 2, \ldots, n\}$ i.e. the vertices of the graph are numbered form 1 to n.

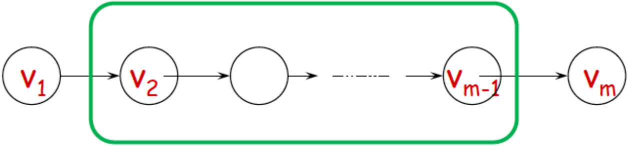
Intermediate Vertices

- Without loss of generality, we'll assume that $V = \{1, 2, \ldots, n\}$ i.e. the vertices of the graph are numbered form 1 to n.

- Given a path $p = \langle v_1, v_2, \ldots, v_m \rangle$ in the graph, we'll call the vertices v_k with index 'k' in $\{2, 3, \ldots, m-1\}$ as the intermediate vertices of **p**.

- Without loss of generality, we'll assume that $V = \{1, 2, \ldots, n\}$ i.e. the vertices of the graph are numbered form 1 to n.

- Given a path $p = \langle v_1, v_2, \ldots, v_m \rangle$ in the graph, we'll call the vertices v_k with index 'k' in $\{2, 3, \ldots, m-1\}$ as the intermediate vertices of **p**.



Key Ideas

Key Ideas

- The key to *Floyd-Warshall* algorithm is the following definition.

- The key to *Floyd-Warshall* algorithm is the following definition.
  - ***Definition:*** Let $d_{ij}^{(k)}$ denote the length of the shortest path form 'i' to 'j' such that all *intermediate vertices* are contained in the set $\{1, 2, \ldots, k\}$.

Key Ideas

- The key to *Floyd-Warshall* algorithm is the following definition.
  - ***Definition:*** Let $d_{ij}^{(k)}$ denote the length of the shortest path form 'i' to 'j' such that all *intermediate vertices* are contained in the set $\{1, 2, \ldots, k\}$.
- A shortest path doesn't contain any vertex twice, as this would imply that the path contains a cycle.

Key Ideas

- The key to *Floyd-Warshall* algorithm is the following definition.
  - ***Definition:*** Let $d_{ij}^{(k)}$ denote the length of the shortest path form 'i' to 'j' such that all *intermediate vertices* are contained in the set $\{1, 2, \ldots, k\}$.
- A shortest path doesn't contain any vertex twice, as this would imply that the path contains a cycle.
- By assumption, cycles in the graph have a positive weight. So removing the cycle would result in a shorter path.

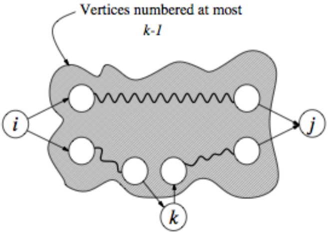
Definitions from Key Ideas

Definitions from Key Ideas

- Consider a shortest path **p** from '$\boldsymbol{i}$' to '$\boldsymbol{j}$' such that the intermediate vertices are form the set $\{1, 2, \ldots, k\}$.

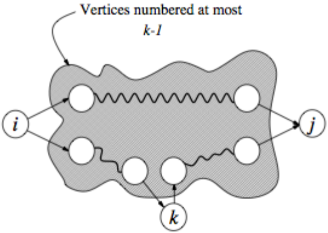
Definitions from Key Ideas

- Consider a shortest path **p** from 'i' to 'j' such that the intermediate vertices are form the set $\{1, 2, \ldots, k\}$.



Definitions from Key Ideas

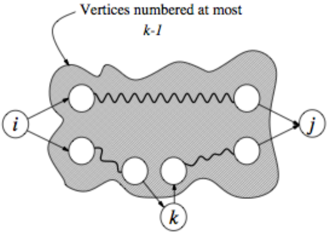
- Consider a shortest path **p** from 'i' to 'j' such that the intermediate vertices are form the set $\{1, 2, \ldots, k\}$.



- If the vertex k is not an intermediate vertex on **p**, then

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

- Consider a shortest path **p** from 'i' to 'j' such that the intermediate vertices are form the set $\{1, 2, \ldots, k\}$.



- If the vertex k is not an intermediate vertex on **p**, then

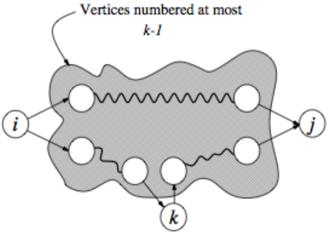
$$d_{ij}^{(k)} = d_{ij}^{(k-1)}$$

- If the vertex k is an intermediate vertex on **p**, then

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

Definitions from Key Ideas

- Consider a shortest path **p** from 'i' to 'j' such that the intermediate vertices are form the set $\{1, 2, \ldots, k\}$.



  - If the vertex k is not an intermediate vertex on **p**, then

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

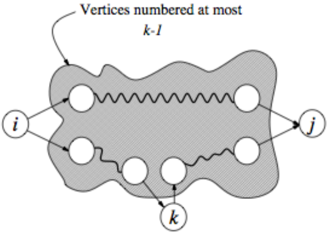
  - If the vertex k is an intermediate vertex on **p**, then

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

- Interestingly, in either case, the subpaths contain merely nodes from $\{1, 2, \ldots, k-1\}$.

Definitions from Key Ideas

- Consider a shortest path **p** from 'i' to 'j' such that the intermediate vertices are form the set $\{1, 2, \ldots, k\}$.



- If the vertex k is not an intermediate vertex on **p**, then

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

- If the vertex k is an intermediate vertex on **p**, then

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

- Interestingly, in either case, the subpaths contain merely nodes from $\{1, 2, \ldots, k-1\}$.

- Therefore, we can conclude that :

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

Recursive Formulation

Recursive Formulation

- If we don't use intermediate nodes i.e. when $\boldsymbol{k = 0}$, then

$$d_{ij}^{(0)} = w_{ij}$$

Recursive Formulation

- If we don't use intermediate nodes i.e. when $\boldsymbol{k = 0}$, then

$$d_{ij}^{(0)} = w_{ij}$$

- if $\boldsymbol{k > 0}$, then

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

Recursive Formulation

- If we don't use intermediate nodes i.e. when $\boldsymbol{k = 0}$, then

$$d_{ij}^{(0)} = w_{ij}$$

- if $\boldsymbol{k > 0}$, then

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

- Mathematically,

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

The Floyd-Warshall Algorithm

FLOYD-WARSHALL(W)

The Floyd-Warshall Algorithm

FLOYD-WARSHALL(W)

1 $n \leftarrow rows[W]$

The Floyd-Warshall Algorithm

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$

$D^{(0)} \leftarrow W$

The Floyd-Warshall Algorithm

FLOYD-WARSHALL(W)

```
n ← rows[W]
D^(0) ← W
for k ← 1 to n
        do for i ← 1 to n
                do for j ← 1 to n
```

The Floyd-Warshall Algorithm

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$
$D^{(0)} \leftarrow W$
for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do $d_{ij}^{(k)} \leftarrow min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$
return $D^{(n)}$

Time and Space Requirements

Time and Space Requirements

1. The running time is $O(V^3)$.

Time and Space Requirements

1. The running time is $O(V^3)$.
2. However, in this version the space requirements are very high.