Analysis of Algorithms

Single Source Shortest Path

Andres Mendez-Vazquez

November 21, 2018

Outline

Outline

Introduction

Problem description

- Given a single source vertex in a weighted, directed graph.
- We want to compute a shortest path for each possible destination (Similar to BFS).

Introduction

Problem description

- Given a single source vertex in a weighted, directed graph.
- We want to compute a shortest path for each possible destination (Similar to BFS).

Introduction

Problem description

- Given a single source vertex in a weighted, directed graph.
- We want to compute a shortest path for each possible destination (Similar to BFS).

Thus

The algorithm will compute a shortest path tree (again, similar to BFS).

Similar Problems

Single destination shortest paths problem

Find a shortest path to a given destination vertex t from each vertex.

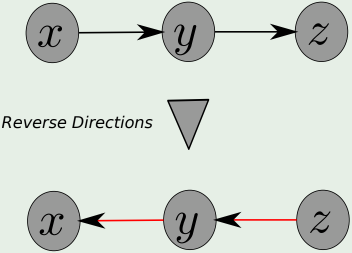
- By reversing the direction of each edge in the graph, we can reduce this problem to a single source problem.

Similar Problems

Single destination shortest paths problem

Find a shortest path to a given destination vertex t from each vertex.

- By reversing the direction of each edge in the graph, we can reduce this problem to a single source problem.



Similar Problems

Single pair shortest path problem

Find a shortest path from u to v for given vertices u and v.

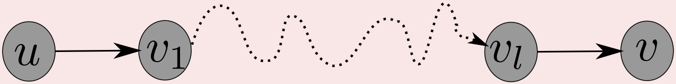
- If we solve the single source problem with source vertex u, we also solve this problem.

Similar Problems

Single pair shortest path problem

Find a shortest path from u to v for given vertices u and v.

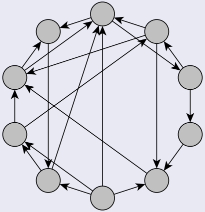
- If we solve the single source problem with source vertex u, we also solve this problem.



Similar Problems

All pairs shortest paths problem

Find a shortest path from u to v for every pair of vertices u and v.



Outline

Optimal Substructure Property

Lemma 24.1

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

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

We have then

- So, we have the optimal substructure property.
- Bellman-Ford's algorithm uses dynamic programming.
- Dijkstra's algorithm uses the greedy approach.

Optimal Substructure Property

Lemma 24.1

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

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

We have then

- So, we have the optimal substructure property.
- Bellman-Ford's algorithm uses dynamic programming.
- Dijkstra's algorithm uses the greedy approach.

Optimal Substructure Property

Lemma 24.1

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

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

We have then

- So, we have the optimal substructure property.
- Bellman-Ford's algorithm uses dynamic programming.
- Dijkstra's algorithm uses the greedy approach.

In addition, we have that

Let $\delta(u, v) = weight$ of **Shortest Path** from u to v.

Optimal Substructure Property

Lemma 24.1

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

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

We have then

- So, we have the optimal substructure property.
- Bellman-Ford's algorithm uses dynamic programming.
- Dijkstra's algorithm uses the greedy approach.

In addition, we have that

Let $\delta(u, v) = weight$ of **Shortest Path** from u to v.

Optimal Substructure Property

Lemma 24.1

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

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

We have then

- So, we have the optimal substructure property.
- Bellman-Ford's algorithm uses dynamic programming.
- Dijkstra's algorithm uses the greedy approach.

In addition, we have that

Let $\delta(u, v) = weight$ of **Shortest Path** from u to v.

Optimal Substructure Property

Lemma 24.1

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

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

We have then

- So, we have the optimal substructure property.
- Bellman-Ford's algorithm uses dynamic programming.
- Dijkstra's algorithm uses the greedy approach.

In addition, we have that

Let $\delta(u, v) = weight$ of **Shortest Path** from u to v.

Optimal Substructure Property

Corollary

Let p be a Shortest Path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \rightarrow v = p_1 \cup \{(u, v)\}$. Then $\delta(s, v) = \delta(s, u) + w(u, v)$.

The Lower Bound Between Nodes

Lemma 24.10

Let $s \in V$. For all edges $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Now

Then

We have the basic concepts

Still

We need to define an important one.

The Predecessor Graph

This will facilitate the proof of several concepts

Now

Then

We have the basic concepts

Still

We need to define an important one.

The Predecessor Graph

This will facilitate the proof of several concepts

Now

Then

We have the basic concepts

Still

We need to define an important one.

The Predecessor Graph

This will facilitate the proof of several concepts

Outline

Predecessor Graph

Representing shortest paths

For this we use the **predecessor subgraph**

- It is defined slightly differently from that on Breadth-First-Search

- $V_{\pi} = \{v \in V | v.\pi \neq NIL\} \cup \{s\}$
- $E_{\pi} = \{(v.\pi, v) | v \in V_{\pi} - \{s\}\}$

- The edges in E_{π} are called tree edges

Predecessor Graph

Representing shortest paths

For this we use the **predecessor subgraph**

- It is defined slightly differently from that on Breadth-First-Search

Definition of a Predecessor Subgraph

The predecessor is a subgraph $G_\pi = (V_\pi, E_\pi)$ where

- $V_\pi = \{v \in V | v.\pi \neq NIL\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) | \; v \in V_\pi - \{s\}\}$

Predecessor Graph

Representing shortest paths

For this we use the **predecessor subgraph**

- It is defined slightly differently from that on Breadth-First-Search

Definition of a Predecessor Subgraph

The predecessor is a subgraph $G_\pi = (V_\pi, E_\pi)$ where

- $V_\pi = \{v \in V | v.\pi \neq NIL\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) | \; v \in V_\pi - \{s\}\}$

Properties

- The predecessor subgraph G_π forms a depth first forest composed of several depth first trees.
- The edges in E_π are called tree edges.

Predecessor Graph

Representing shortest paths

For this we use the **predecessor subgraph**

- It is defined slightly differently from that on Breadth-First-Search

Definition of a Predecessor Subgraph

The predecessor is a subgraph $G_\pi = (V_\pi, E_\pi)$ where

- $V_\pi = \{v \in V | v.\pi \neq NIL\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) | \; v \in V_\pi - \{s\}\}$

Properties

- The predecessor subgraph G_π forms a depth first forest composed of several depth first trees.
- The edges in E_π are called tree edges.

Predecessor Graph

Representing shortest paths

For this we use the **predecessor subgraph**

- It is defined slightly differently from that on Breadth-First-Search

Definition of a Predecessor Subgraph

The predecessor is a subgraph $G_\pi = (V_\pi, E_\pi)$ where

- $V_\pi = \{v \in V | v.\pi \neq NIL\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) |\ v \in V_\pi - \{s\}\}$

Properties

- The predecessor subgraph G_π forms a depth first forest composed of several depth first trees.
- The edges in E_π are called tree edges.

Predecessor Graph

Representing shortest paths

For this we use the **predecessor subgraph**

- It is defined slightly differently from that on Breadth-First-Search

Definition of a Predecessor Subgraph

The predecessor is a subgraph $G_\pi = (V_\pi, E_\pi)$ where

- $V_\pi = \{v \in V | v.\pi \neq NIL\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) |\ v \in V_\pi - \{s\}\}$

Properties

- The predecessor subgraph G_π forms a depth first forest composed of several depth first trees.
- The edges in E_π are called tree edges.

Predecessor Graph

Representing shortest paths

For this we use the **predecessor subgraph**

- It is defined slightly differently from that on Breadth-First-Search

Definition of a Predecessor Subgraph

The predecessor is a subgraph $G_\pi = (V_\pi, E_\pi)$ where

- $V_\pi = \{v \in V | v.\pi \neq NIL\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) |\ v \in V_\pi - \{s\}\}$

Properties

- The predecessor subgraph G_π forms a depth first forest composed of several depth first trees.
- The edges in E_π are called tree edges.

Outline

The Relaxation Concept

We are going to use certain functions for all the algorithms

- Initialize
  - Here, the basic variables of the nodes in a graph will be initialized
    - $v.d$ = the distance from the source s.
    - $v.\pi$ = the predecessor node during the search of the shortest path.

The Relaxation Concept

We are going to use certain functions for all the algorithms

- Initialize
  - Here, the basic variables of the nodes in a graph will be initialized
    - ⋆ $v.d$ = the distance from the source s.
    - ⋆ $v.\pi$ = the predecessor node during the search of the shortest path.

Changing the $v.d$

This will be done in the Relaxation algorithm.

The Relaxation Concept

We are going to use certain functions for all the algorithms

- Initialize
  - Here, the basic variables of the nodes in a graph will be initialized
    - $v.d =$ the distance from the source s.
    - $v.\pi =$ the predecessor node during the search of the shortest path.

Changing the $v.d$

This will be done in the Relaxation algorithm.

The Relaxation Concept

We are going to use certain functions for all the algorithms

- Initialize
  - Here, the basic variables of the nodes in a graph will be initialized
    - ★ $v.d =$ the distance from the source s.
    - ★ $v.\pi =$ the predecessor node during the search of the shortest path.

Changing the $v.d$

This will be done in the Relaxation algorithm.

The Relaxation Concept

We are going to use certain functions for all the algorithms

- Initialize
  - Here, the basic variables of the nodes in a graph will be initialized
    - $v.d =$ the distance from the source s.
    - $v.\pi =$ the predecessor node during the search of the shortest path.

Changing the $v.d$

This will be done in the Relaxation algorithm.

Initialize and Relaxation

The Algorithms keep track of $v.d$, $v.\pi$. It is initialized as follows

```
Initialize($G, s$)
for each $v \in V[G]$
        $v.d = \infty$
        $v.\pi = NIL$
$s.d = 0$
```

These values are changed when an edge (u, v) is relaxed.

```
Relax($u, v, w$)
if $v.d > u.d + w(u, v)$
        $v.d = u.d + w(u, v)$
        $v.\pi = u$
```

Initialize and Relaxation

The Algorithms keep track of $v.d$, $v.\pi$. It is initialized as follows

```
Initialize($G, s$)
1 for each $v \in V[G]$
2         $v.d = \infty$
3         $v.\pi = NIL$
4 $s.d = 0$
```

These values are changed when an edge (u, v) is relaxed.

```
Relax($u, v, w$)
1 if $v.d > u.d + w(u, v)$
2         $v.d = u.d + w(u, v)$
3         $v.\pi = u$
```

How are these functions used?

These functions are used

1. Build a predecesor graph G_π.
2. Integrate the Shortest Path into that predecessor graph.
    1. Using the field d.

How are these functions used?

These functions are used

1. Build a predecesor graph G_π.
2. Integrate the Shortest Path into that predecessor graph.
    1. Using the field d.

Outline

The Bellman-Ford Algorithm

Bellman-Ford can have negative weight edges. It will "detect" reachable negative weight cycles.

```
Bellman-Ford($G, s, w$)
Initialize($G, s$)
for $i = 1$ to $|V[G]| - 1$
    for each $(u, v)$ to $E[G]$
        Relax($u, v, w$)
for each $(u, v)$ to $E[G]$
    if $v.d > u.d + w(u, v)$
        return false
return true
```

The Bellman-Ford Algorithm

Bellman-Ford can have negative weight edges. It will "detect" reachable negative weight cycles.

```
Bellman-Ford($G, s, w$)
Initialize($G, s$)
for $i = 1$ to $|V[G]| - 1$
    for each $(u, v)$ to $E[G]$
        Relax($u, v, w$)
for each $(u, v)$ to $E[G]$
    if $v.d > u.d + w(u, v)$
        return false
return true
```

Time Complexity

$O(VE)$

The Bellman-Ford Algorithm

Bellman-Ford can have negative weight edges. It will “detect” reachable negative weight cycles.

```
Bellman-Ford($G, s, w$)
    Initialize($G, s$)
    for $i = 1$ to $|V[G]| - 1$
        for each $(u, v)$ to $E[G]$
            Relax($u, v, w$)
    for each $(u, v)$ to $E[G]$
        if $v.d > u.d + w(u, v)$
            return false
    return true
```

Time Complexity

$O(VE)$

The Bellman-Ford Algorithm

Bellman-Ford can have negative weight edges. It will "detect" reachable negative weight cycles.

```
Bellman-Ford($G, s, w$)
Initialize($G, s$)
for $i = 1$ to $|V[G]| - 1$
    for each $(u, v)$ to $E[G]$
        Relax($u, v, w$)
for each $(u, v)$ to $E[G]$
    if $v.d > u.d + w(u, v)$
        return false
return true
```

Time Complexity

$O(VE)$

Outline

Properties of Relaxation

Some properties

- $v.d$, if not ∞, is the length of some path from s to v.
- $v.d$ either stays the same or decreases with time.

Properties of Relaxation

Some properties

- $v.d$, if not ∞, is the length of some path from s to v.
- $v.d$ either stays the same or decreases with time.

Therefore

- If $v.d = \delta(s, v)$ at any time, this holds thereafter.

- After i iterations of relaxing an all (u, v), if the shortest path to v has i edges, then $v.d = \delta(s, v)$.
- If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.

Properties of Relaxation

Some properties

- $v.d$, if not ∞, is the length of some path from s to v.
- $v.d$ either stays the same or decreases with time.

Therefore

- If $v.d = \delta(s, v)$ at any time, this holds thereafter.

Something nice

- Note that $v.d \geq \delta(s, v)$ always (Upper-Bound Property).
- **After i iterations of relaxing an all (u, v), if the shortest path to v has i edges, then $v.d = \delta(s, v)$.**
- If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.

Properties of Relaxation

Some properties

- $v.d$, if not ∞, is the length of some path from s to v.
- $v.d$ either stays the same or decreases with time.

Therefore

- If $v.d = \delta(s, v)$ at any time, this holds thereafter.

Something nice

- Note that $v.d \geq \delta(s, v)$ always (Upper-Bound Property).
- **After i iterations of relaxing an all (u, v), if the shortest path to v has i edges, then $v.d = \delta(s, v)$.**
- If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.

Properties of Relaxation

Some properties

- $v.d$, if not ∞, is the length of some path from s to v.
- $v.d$ either stays the same or decreases with time.

Therefore

- If $v.d = \delta(s, v)$ at any time, this holds thereafter.

Something nice

- Note that $v.d \geq \delta(s, v)$ always (Upper-Bound Property).
- **After i iterations of relaxing an all (u, v), if the shortest path to v has i edges, then $v.d = \delta(s, v)$.**
- If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.

Properties of Relaxation

Some properties

- $v.d$, if not ∞, is the length of some path from s to v.
- $v.d$ either stays the same or decreases with time.

Therefore

- If $v.d = \delta(s, v)$ at any time, this holds thereafter.

Something nice

- Note that $v.d \geq \delta(s, v)$ always (Upper-Bound Property).
- **After i iterations of relaxing an all (u, v), if the shortest path to v has i edges, then $v.d = \delta(s, v)$.**
- If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.

Properties of Relaxation

Lemma 24.10 (Triangle inequality)

Let $G=(V,E)$ be a weighted, directed graph with weight function $w:E\rightarrow\mathbb{R}$ and source vertex s. Then, for all edges $(u,v)\in E$, we have:

$$\delta(s,v)\leq\delta(s,u)+w(u,v) \tag{1}$$

Properties of Relaxation

Lemma 24.10 (Triangle inequality)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \mathbb{R}$ and source vertex s. Then, for all edges $(u, v) \in E$, we have:

$$\delta(s, v) \leq \delta(s, u) + w(u, v) \tag{1}$$

Proof

1. Suppose that p is a shortest path from source s to vertex v.
2. Then, p has no more weight than any other path from s to vertex v.
3. Not only p has no more weiht tha a particular shortest path that goes from s to u and then takes edge (u, v).

Properties of Relaxation

Lemma 24.10 (Triangle inequality)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ and source vertex s. Then, for all edges $(u, v) \in E$, we have:

$$\delta(s, v) \leq \delta(s, u) + w(u, v) \tag{1}$$

Proof

1. Suppose that p is a shortest path from source s to vertex v.
2. Then, p has no more weight than any other path from s to vertex v.
3. Not only p has no more weiht tha a particular shortest path that goes from s to u and then takes edge (u, v).

Properties of Relaxation

Lemma 24.10 (Triangle inequality)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ and source vertex s. Then, for all edges $(u, v) \in E$, we have:

$$\delta(s, v) \leq \delta(s, u) + w(u, v) \tag{1}$$

Proof

1. Suppose that p is a shortest path from source s to vertex v.
2. Then, p has no more weight than any other path from s to vertex v.
3. Not only p has no more weiht tha a particular shortest path that goes from s to u and then takes edge (u, v).

Properties of Relaxation

Lemma 24.10 (Triangle inequality)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ and source vertex s. Then, for all edges $(u, v) \in E$, we have:

$$\delta(s, v) \leq \delta(s, u) + w(u, v) \tag{1}$$

Proof

1. Suppose that p is a shortest path from source s to vertex v.
2. Then, p has no more weight than any other path from s to vertex v.
3. Not only p has no more weiht tha a particular shortest path that goes from s to u and then takes edge (u, v).

Properties of Relaxation

Lemma 24.11 (Upper Bound Property)

- Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \mathbb{R}$. Consider any algorithm in which $v.d$, and $v.\pi$ are first initialized by calling $Initialize(G, s)$ (s is the source), and are only changed by calling $Relax$.
- Then, we have that $v.d \geq \delta(s, v)$ $\forall v \in V[G]$, and this invariant is maintained over any sequence of relaxation steps on the edges of G.
- Moreover, once $v.d = \delta(s, v)$, it never changes.

Properties of Relaxation

Lemma 24.11 (Upper Bound Property)

- Let $G=(V,E)$ be a weighted, directed graph with weight function $w:E\rightarrow\mathbb{R}$. Consider any algorithm in which $v.d$, and $v.\pi$ are first initialized by calling $Initialize(G,s)$ (s is the source), and are only changed by calling $Relax$.
- Then, we have that $v.d\geq\delta(s,v)\ \forall v\in V\left[G\right]$, and this invariant is maintained over any sequence of relaxation steps on the edges of G.
- Moreover, once $v.d=\delta(s,v)$, it never changes.

Properties of Relaxation

Lemma 24.11 (Upper Bound Property)

- Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \mathbb{R}$. Consider any algorithm in which $v.d$, and $v.\pi$ are first initialized by calling $Initialize(G, s)$ (s is the source), and are only changed by calling $Relax$.
- Then, we have that $v.d \geq \delta(s, v)$ $\forall v \in V\left[G\right]$, and this invariant is maintained over any sequence of relaxation steps on the edges of G.
- Moreover, once $v.d = \delta(s, v)$, it never changes.

Proof of Lemma

Loop Invariance

The Proof can be done by induction over the number of relaxation steps and the loop invariance:

- $v.d \geq \delta(s, v)$ for all $v \in V$

Proof of Lemma

Loop Invariance

The Proof can be done by induction over the number of relaxation steps and the loop invariance:

- $v.d \geq \delta\left(s,v\right)$ for all $v \in V$

For the Basis

$v.d \geq \delta\left(s,v\right)$ is true after initialization, since:

- $v.d = \infty$ making $v.d \geq \delta\left(s,v\right)$ for all $v \in V - \{s\}$.
- For s, $s.d = 0 \geq \delta\left(s,s\right)$.

Proof of Lemma

Loop Invariance

The Proof can be done by induction over the number of relaxation steps and the loop invariance:

- $v.d \geq \delta(s,v)$ for all $v \in V$

For the Basis

$v.d \geq \delta(s,v)$ is true after initialization, since:

- $v.d = \infty$ making $v.d \geq \delta(s,v)$ for all $v \in V - \{s\}$.
- For s, $s.d = 0 \geq \delta(s,s)$.

For the inductive step, consider the relaxation of an edge (u,v)

By the inductive hypothesis, we have that $x.d \geq \delta(s,x)$ for all $x \in V$ prior to relaxation.

Proof of Lemma

Loop Invariance

The Proof can be done by induction over the number of relaxation steps and the loop invariance:

- $v.d \geq \delta(s, v)$ for all $v \in V$

For the Basis

$v.d \geq \delta(s, v)$ is true after initialization, since:

- $v.d = \infty$ making $v.d \geq \delta(s, v)$ for all $v \in V - \{s\}$.
- For s, $s.d = 0 \geq \delta(s, s)$.

For the inductive step, consider the relaxation of an edge (u, v)

By the inductive hypothesis, we have that $x.d \geq \delta(s, x)$ for all $x \in V$ prior to relaxation.

Proof of Lemma

Loop Invariance

The Proof can be done by induction over the number of relaxation steps and the loop invariance:

- $v.d \geq \delta\left(s, v\right)$ for all $v \in V$

For the Basis

$v.d \geq \delta\left(s, v\right)$ is true after initialization, since:

- $v.d = \infty$ making $v.d \geq \delta\left(s, v\right)$ for all $v \in V - \{s\}$.
- For s, $s.d = 0 \geq \delta\left(s, s\right)$.

For the inductive step, consider the relaxation of an edge (u, v)

By the inductive hypothesis, we have that $x.d \geq \delta\left(s, x\right)$ for all $x \in V$ prior to relaxation.

Thus

If you call $Relax(u, v, w)$, it may change $v.d$

$$\begin{aligned} v.d &= u.d + w(u, v) \\ &\geq \delta(s, u) + w(u, v) \text{ by inductive hypothesis} \\ &\geq \delta\left(s, v\right) \text{ by the triangle inequality} \end{aligned}$$

Thus, the invariant is maintained.

Thus

If you call $Relax(u, v, w)$, it may change $v.d$

$$\begin{aligned} v.d &= u.d + w(u, v) \\ &\geq \delta(s, u) + w(u, v) \text{ by inductive hypothesis} \\ &\geq \delta(s, v) \text{ by the triangle inequality} \end{aligned}$$

Thus, the invariant is maintained.

Thus

If you call $Relax(u, v, w)$, it may change $v.d$

$$\begin{aligned} v.d &= u.d + w(u, v) \\ &\geq \delta(s, u) + w(u, v) \text{ by inductive hypothesis} \\ &\geq \delta\left(s, v\right) \text{ by the triangle inequality} \end{aligned}$$

Thus, the invariant is maintained.

Thus

If you call $Relax(u, v, w)$, it may change $v.d$

$$\begin{aligned} v.d &= u.d + w(u,v) \\ &\geq \delta(s,u) + w(u,v) \text{ by inductive hypothesis} \\ &\geq \delta\left(s,v\right) \text{ by the triangle inequality} \end{aligned}$$

Thus, the invariant is maintained.

Properties of Relaxation

Proof of lemma 24.11 cont...

- To proof that the value $v.d$ never changes once $v.d = \delta\left(s, v\right)$:
    - Note the following: Once $v.d = \delta\left(s, v\right)$, it cannot decrease because $v.d \geq \delta\left(s, v\right)$ and Relaxation never increases d.

Properties of Relaxation

Proof of lemma 24.11 cont...

- To proof that the value $v.d$ never changes once $v.d = \delta(s, v)$:
  - Note the following: Once $v.d = \delta(s, v)$, it cannot decrease because $v.d \geq \delta(s, v)$ and Relaxation never increases d.

Next, we have

Corollary 24.12 (No-path property)

If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.

Proof

By the upper-bound property, we always have $\infty = \delta(s, v) \leq v.d$. Then, $v.d = \infty$.

Next, we have

Corollary 24.12 (No-path property)

If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.

Proof

By the upper-bound property, we always have $\infty = \delta\left(s, v\right) \leq v.d$. Then, $v.d = \infty$.

More Lemmas

Lemma 24.13

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \mathbb{R}$. Then, immediately after relaxing edge (u, v) by calling $Relax(u, v, w)$ we have $v.d \leq u.d + w(u, v)$.

Proof

First

If, just prior to relaxing edge (u, v),

- Case 1: if we have that $v.d > u.d + w(u, v)$
    - Then, $v.d = u.d + w(u, v)$ after relaxation.

Proof

First

If, just prior to relaxing edge (u, v),

- Case 1: if we have that $v.d > u.d + w(u, v)$
    - Then, $v.d = u.d + w(u, v)$ after relaxation.

Now, Case 2

If $v.d \leq u.d + w(u, v)$ just before relaxation, then:

- neither $u.d$ nor $v.d$ changes

Proof

First

If, just prior to relaxing edge (u, v),

- Case 1: if we have that $v.d > u.d + w(u, v)$
  - Then, $v.d = u.d + w(u, v)$ after relaxation.

Now, Case 2

If $v.d \leq u.d + w(u, v)$ just before relaxation, then:

- neither $u.d$ nor $v.d$ changes

Thus, afterwards

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

Proof

First

If, just prior to relaxing edge (u, v),

- Case 1: if we have that $v.d > u.d + w(u, v)$
  - Then, $v.d = u.d + w(u, v)$ after relaxation.

Now, Case 2

If $v.d \leq u.d + w(u, v)$ just before relaxation, then:

- neither $u.d$ nor $v.d$ changes

Thus, afterwards

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

Proof

First

If, just prior to relaxing edge (u, v),

- Case 1: if we have that $v.d > u.d + w(u, v)$
  - Then, $v.d = u.d + w(u, v)$ after relaxation.

Now, Case 2

If $v.d \leq u.d + w(u, v)$ just before relaxation, then:

- neither $u.d$ nor $v.d$ changes

Thus, afterwards

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

Proof

First

If, just prior to relaxing edge (u, v),

- Case 1: if we have that $v.d > u.d + w(u, v)$
  - Then, $v.d = u.d + w(u, v)$ after relaxation.

Now, Case 2

If $v.d \leq u.d + w(u, v)$ just before relaxation, then:

- neither $u.d$ nor $v.d$ changes

Thus, afterwards

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

More Lemmas

Lemma 24.14 (Convergence property)

- Let p be a shortest path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \rightarrow v = p_1 \cup \{(u,v)\}$.
- If $u.d = \delta(s,u)$ holds at any time prior to calling $Relax(u,v,w)$, then $v.d = \delta(s,v)$ holds at all times after the call.

More Lemmas

Lemma 24.14 (Convergence property)

- Let p be a shortest path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \to v = p_1 \cup \{(u,v)\}$.
- If $u.d = \delta(s,u)$ holds at any time prior to calling $Relax(u,v,w)$, then $v.d = \delta(s,v)$ holds at all times after the call.

Proof:

By the upper-bound property, if $u.d = \delta(s,u)$ at some moment before relaxing edge (u,v), holding afterwards.

More Lemmas

Lemma 24.14 (Convergence property)

- Let p be a shortest path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \to v = p_1 \cup \{(u, v)\}$.
- If $u.d = \delta(s, u)$ holds at any time prior to calling $Relax(u, v, w)$, then $v.d = \delta(s, v)$ holds at all times after the call.

Proof:

By the upper-bound property, if $u.d = \delta(s, u)$ at some moment before relaxing edge (u, v), holding afterwards.

Proof

Thus , after relaxing (u, v)

$$v.d \leq u.d + w(u,v) \text{ by lemma 24.13}$$

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

$$= \delta(s,v) \text{ by corollary of lemma 24.1}$$

Proof

Thus , after relaxing (u, v)

$$\begin{aligned} v.d &\leq u.d + w(u,v) \text{ by lemma 24.13} \\ &= \delta\left(s,u\right) + w\left(u,v\right) \\ &= \delta\left(s,v\right) \text{ by corollary of lemma 24.1} \end{aligned}$$

Now

By lemma 24.11, $v.d \geq \delta(s,v)$, so $v.d = \delta(s,v)$.

Proof

Thus , after relaxing (u, v)

$$\begin{aligned} v.d &\leq u.d + w(u,v) \text{ by lemma 24.13} \\ &= \delta\left(s,u\right) + w\left(u,v\right) \\ &= \delta\left(s,v\right) \text{ by corollary of lemma 24.1} \end{aligned}$$

Now

By lemma 24.11, $v.d \geq \delta(s,v)$, so $v.d = \delta(s,v)$.

Proof

Thus , after relaxing (u,v)

$$\begin{aligned} v.d &\leq u.d + w(u,v) \text{ by lemma 24.13} \\ &= \delta\left(s,u\right) + w\left(u,v\right) \\ &= \delta\left(s,v\right) \text{ by corollary of lemma 24.1} \end{aligned}$$

Now

By lemma 24.11, $v.d \geq \delta(s,v)$, so $v.d = \delta(s,v)$.

Outline

Predecessor Subgraph for Bellman

Lemma 24.16

Assume a given graph G that has no negative weight cycles reachable from s. Then, after the initialization, the predecessor subgraph G_π is always a tree with root s, and any sequence of relaxations steps on edges of G maintains this property as an invariant.

Predecessor Subgraph for Bellman

Lemma 24.16

Assume a given graph G that has no negative weight cycles reachable from s. Then, after the initialization, the predecessor subgraph G_π is always a tree with root s, and any sequence of relaxations steps on edges of G maintains this property as an invariant.

Proof

It is necessary to prove two things in order to get a tree:

1. G_π is acyclic.
2. There exists a unique path from source s to each vertex V_π.

Predecessor Subgraph for Bellman

Lemma 24.16

Assume a given graph G that has no negative weight cycles reachable from s. Then, after the initialization, the predecessor subgraph G_π is always a tree with root s, and any sequence of relaxations steps on edges of G maintains this property as an invariant.

Proof

It is necessary to prove two things in order to get a tree:

1. G_π is acyclic.
2. There exists a unique path from source s to each vertex V_π.

Predecessor Subgraph for Bellman

Lemma 24.16

Assume a given graph G that has no negative weight cycles reachable from s. Then, after the initialization, the predecessor subgraph G_π is always a tree with root s, and any sequence of relaxations steps on edges of G maintains this property as an invariant.

Proof

It is necessary to prove two things in order to get a tree:

1. G_π is acyclic.
2. There exists a unique path from source s to each vertex V_π.

Proof of G_π is acyclic

First

Suppose there exist a cycle $c =< v_0, v_1, ..., v_k >$, where $v_0 = v_k$. We have $v_i.\pi = v_{i-1}$ for $i = 1, 2, ..., k$.

Second

Assume relaxation of (v_{k-1}, v_k) created the cycle. We are going to show that the cycle has a negative weight.

We claim that

The cycle must be reachable from s (Why?)

Proof of G_π is acyclic

First

Suppose there exist a cycle $c = < v_0, v_1, ..., v_k >$, where $v_0 = v_k$. We have $v_i.\pi = v_{i-1}$ for $i = 1, 2, ..., k$.

Second

Assume relaxation of (v_{k-1}, v_k) created the cycle. We are going to show that the cycle has a negative weight.

We claim that

The cycle must be reachable from s (Why?)

Proof of G_π is acyclic

First

Suppose there exist a cycle $c =< v_0, v_1, ..., v_k >$, where $v_0 = v_k$. We have $v_i.\pi = v_{i-1}$ for $i = 1, 2, ..., k$.

Second

Assume relaxation of (v_{k-1}, v_k) created the cycle. We are going to show that the cycle has a negative weight.

We claim that

The cycle must be reachable from s (Why?)

Proof

First

Each vertex on the cycle has a non-NIL predecessor, and so each vertex on it was assigned a finite shortest-path estimate when it was assigned its non-NIL value.

Then

By the upper-bound property, each vertex on the cycle has a finite shortest-path weight.

Thus

Making the cycle reachable from s.

Proof

First

Each vertex on the cycle has a non-NIL predecessor, and so each vertex on it was assigned a finite shortest-path estimate when it was assigned its non-NIL value.

Then

By the upper-bound property, each vertex on the cycle has a finite shortest-path weight,

Thus

Making the cycle reachable from s.

Proof

First

Each vertex on the cycle has a non-NIL predecessor, and so each vertex on it was assigned a finite shortest-path estimate when it was assigned its non-NIL value.

Then

By the upper-bound property, each vertex on the cycle has a finite shortest-path weight,

Thus

Making the cycle reachable from s.

Proof

Before call to $Relax(v_{k-1}, v_k, w)$:

$$v_i.\pi = v_{i-1} \text{ for } i = 1, ..., k-1. \tag{2}$$

Thus

This Implies $v_i.d$ was last updated by

$$v_i.d = v_{i-1}.d + w(v_{i-1}, v_i) \tag{3}$$

for $i = 1, ..., k-1$ (Because $Relax$ updates π).

This implies

This implies

$$v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i) \tag{4}$$

for $i = 1, ..., k-1$ (Before Relaxation in Lemma 24.13).

Proof

Before call to $Relax(v_{k-1}, v_k, w)$:

$$v_i.\pi = v_{i-1} \text{ for } i = 1, ..., k-1. \tag{2}$$

Thus

This Implies $v_i.d$ was last updated by

$$v_i.d = v_{i-1}.d + w(v_{i-1}, v_i) \tag{3}$$

for $i = 1, ..., k-1$ (Because $Relax$ updates π).

This implies

This implies

$$v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i) \tag{4}$$

for $i = 1, ..., k-1$ (Before Relaxation in Lemma 24.13).

Proof

Before call to $Relax(v_{k-1}, v_k, w)$:

$$v_i.\pi = v_{i-1} \text{ for } i = 1, ..., k-1. \tag{2}$$

Thus

This Implies $v_i.d$ was last updated by

$$v_i.d = v_{i-1}.d + w(v_{i-1}, v_i) \tag{3}$$

for $i = 1, ..., k-1$ (Because $Relax$ updates π).

This implies

This implies

$$v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i) \tag{4}$$

for $i = 1, ..., k-1$ (Before Relaxation in Lemma 24.13).

Proof

Thus

Because $v_k.\pi$ is changed by call $Relax$ (Immediately before), $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$, we have that:

$$
\begin{aligned}
\sum_{i=1}^{k} v_i.d &> \sum_{i=1}^{k} \left(v_{i-1}.d + w\left(v_{i-1}, v_i\right)\right) \\
&= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w\left(v_{i-1}, v_i\right)
\end{aligned}
$$

Proof

Thus

Because $v_k.\pi$ is changed by call $Relax$ (Immediately before), $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$, we have that:

$$\sum_{i=1}^{k} v_i.d > \sum_{i=1}^{k} \left(v_{i-1}.d + w\left(v_{i-1}, v_i\right)\right)$$

$$= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w\left(v_{i-1}, v_i\right)$$

We have finally that

Because $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$, we have that $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$, i.e., a negative weight cycle!!!

Proof

Thus

Because $v_k.\pi$ is changed by call $Relax$ (Immediately before), $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$, we have that:

$$\begin{aligned}\sum_{i=1}^{k} v_i.d &> \sum_{i=1}^{k} \left(v_{i-1}.d + w\left(v_{i-1}, v_i\right)\right) \\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w\left(v_{i-1}, v_i\right)\end{aligned}$$

We have finally that

Because $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$, we have that $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$, i.e., a negative weight cycle!!!

Proof

Thus

Because $v_k.\pi$ is changed by call $Relax$ (Immediately before), $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$, we have that:

$$\begin{aligned}\sum_{i=1}^{k} v_i.d &> \sum_{i=1}^{k} \left(v_{i-1}.d + w\left(v_{i-1}, v_i\right)\right)\\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w\left(v_{i-1}, v_i\right)\end{aligned}$$

We have finally that

Because $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$, we have that $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$, i.e., a negative weight cycle!!!

Some comments

Comments

- $v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i)$ for $i = 1, ..., k-1$ because when $Relax(v_{i-1}, v_i, w)$ was called, there was an equality, and $v_{i-1}.d$ may have gotten smaller by further calls to $Relax$.
- $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$ before the last call to $Relax$ because that last call changed $v_k.d$.

Some comments

Comments

- $v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i)$ for $i = 1, ..., k-1$ because when $Relax(v_{i-1}, v_i, w)$ was called, there was an equality, and $v_{i-1}.d$ may have gotten smaller by further calls to $Relax$.
- $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$ before the last call to $Relax$ because that last call changed $v_k.d$.

Proof of existence of a unique path from source s

Let G_π be the predecessor subgraph.

- So, for any v in V_π, the graph G_π contains at least one path from s to v.
- Assume now that you have two paths:
- This can only be possible if for two nodes x and $y \Rightarrow x \neq y$, but $z.\pi = x = y$!!!
- Contradiction!!! Therefore, we have only one path and G_π is a tree.

Proof of existence of a unique path from source s

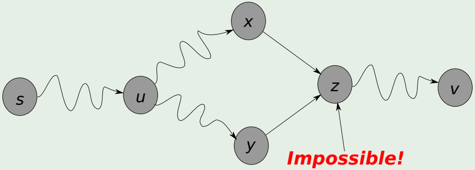
Let G_π be the predecessor subgraph.

- So, for any v in V_π, the graph G_π contains at least one path from s to v.
- Assume now that you have two paths:
- This can only be possible if for two nodes x and $y \Rightarrow x \neq y$, but $z.\pi = x = y$!!!
- Contradiction!!! Therefore, we have only one path and G_π is a tree.

Proof of existence of a unique path from source s

Let G_π be the predecessor subgraph.

- So, for any v in V_π, the graph G_π contains at least one path from s to v.
- Assume now that you have two paths:

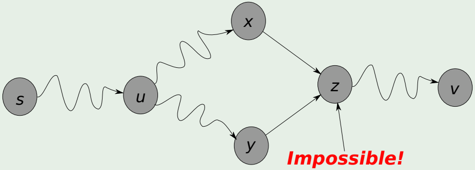


- This can only be possible if for two nodes x and $y \Rightarrow x \neq y$, but $z.\pi = x = y$!!!
- Contradiction!!! Therefore, we have only one path and G_π is a tree.

Proof of existence of a unique path from source s

Let G_π be the predecessor subgraph.

- So, for any v in V_π, the graph G_π contains at least one path from s to v.
- Assume now that you have two paths:

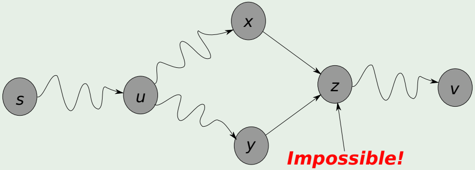


- This can only be possible if for two nodes x and $y \Rightarrow x \neq y$, but $z.\pi = x = y$!!!
- Contradiction!!! Therefore, we have only one path and G_π is a tree.

Proof of existence of a unique path from source s

Let G_π be the predecessor subgraph.

- So, for any v in V_π, the graph G_π contains at least one path from s to v.
- Assume now that you have two paths:



- This can only be possible if for two nodes x and $y \Rightarrow x \neq y$, but $z.\pi = x = y$!!!
- Contradiction!!! Therefore, we have only one path and G_π is a tree.

Outline

Lemma 24.17

Lemma 24.17

Same conditions as before. It calls Initialize and repeatedly calls Relax until $v.d = \delta(s, v)$ for all v in V. Then G_{π} is a shortest path tree rooted at s.

Lemma 24.17

Lemma 24.17

Same conditions as before. It calls Initialize and repeatedly calls Relax until $v.d = \delta(s, v)$ for all v in V. Then G_π is a shortest path tree rooted at s.

Proof

- For all v in V, there is a unique simple path p from s to v in G_π (Lemma 24.16).
- We want to prove that it is a shortest path from s to v in G.

Lemma 24.17

Lemma 24.17

Same conditions as before. It calls Initialize and repeatedly calls Relax until $v.d = \delta(s, v)$ for all v in V. Then G_π is a shortest path tree rooted at s.

Proof

- For all v in V, there is a unique simple path p from s to v in G_π (Lemma 24.16).
- We want to prove that it is a shortest path from s to v in G.

Proof

Then

Let $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$. Thus, we have $v_i.d = \delta(s, v_i)$.

And reasoning as before

$$v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i) \tag{5}$$

This implies that

$$w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1}) \tag{6}$$

Proof

Then

Let $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$. Thus, we have $v_i.d = \delta(s, v_i)$.

And reasoning as before

$$v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i) \tag{5}$$

This implies that

$$w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1}) \tag{6}$$

Proof

Then

Let $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$. Thus, we have $v_i.d = \delta(s, v_i)$.

And reasoning as before

$$v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i) \quad (5)$$

This implies that

$$w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1}) \quad (6)$$

Proof

Then, we sum over all weights

$$
\begin{aligned}
w(p) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\
&\leq \sum_{i=1}^{k} \left(\delta(s, v_i) - \delta(s, v_{i-1})\right) \\
&= \delta(s, v_k) - \delta(s, v_0) \\
&= \delta(s, v_k)
\end{aligned}
$$

Proof

Then, we sum over all weights

$$
\begin{aligned}
w(p) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\
&\leq \sum_{i=1}^{k} \left(\delta(s, v_i) - \delta(s, v_{i-1})\right) \\
&= \delta(s, v_k) - \delta(s, v_0) \\
&= \delta(s, v_k)
\end{aligned}
$$

Finally

So, equality holds and p is a shortest path because $\delta(s, v_k) \leq w(p)$.

Proof

Then, we sum over all weights

$$
\begin{aligned}
w\left(p\right) &= \sum_{i=1}^{k} w\left(v_{i-1}, v_i\right)\\
&\leq \sum_{i=1}^{k} \left(\delta\left(s, v_i\right) - \delta(s, v_{i-1})\right)\\
&= \delta(s, v_k) - \delta(s, v_0)\\
&= \delta(s, v_k)
\end{aligned}
$$

Finally

So, equality holds and p is a shortest path because $\delta\left(s, v_k\right) \leq w\left(p\right)$.

Proof

Then, we sum over all weights

$$\begin{aligned} w\left(p\right) &= \sum_{i=1}^{k} w\left(v_{i-1}, v_i\right) \\ &\leq \sum_{i=1}^{k} \left(\delta\left(s, v_i\right) - \delta(s, v_{i-1})\right) \\ &= \delta(s, v_k) - \delta(s, v_0) \\ &= \delta(s, v_k) \end{aligned}$$

Finally

So, equality holds and p is a shortest path because $\delta\left(s, v_k\right) \leq w\left(p\right)$.

Proof

Then, we sum over all weights

$$
\begin{aligned}
w(p) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\
&\leq \sum_{i=1}^{k} \left(\delta(s, v_i) - \delta(s, v_{i-1})\right) \\
&= \delta(s, v_k) - \delta(s, v_0) \\
&= \delta(s, v_k)
\end{aligned}
$$

Finally

So, equality holds and p is a shortest path because $\delta(s, v_k) \leq w(p)$.

Outline

Again the Bellman-Ford Algorithm

Bellman-Ford can have negative weight edges. It will "detect" reachable negative weight cycles.

```
Bellman-Ford($G, s, w$)
Initialize($G, s$)
for $i = 1$ to $|V[G]| - 1$
    for each $(u, v)$ to $E[G]$
        Relax($u, v, w$)◁ The Decision Part of the Dynamic Programming for $u.d$ and $u.\pi$.
for each $(u, v)$ to $E[G]$
    if $v.d > u.d + w(u, v)$
        return false
return true
```

Observation

If Bellman-Ford has not converged after $V(G) - 1$ iterations, then there cannot be a shortest path tree, so there must be a negative weight cycle.

Again the Bellman-Ford Algorithm

Bellman-Ford can have negative weight edges. It will "detect" reachable negative weight cycles.

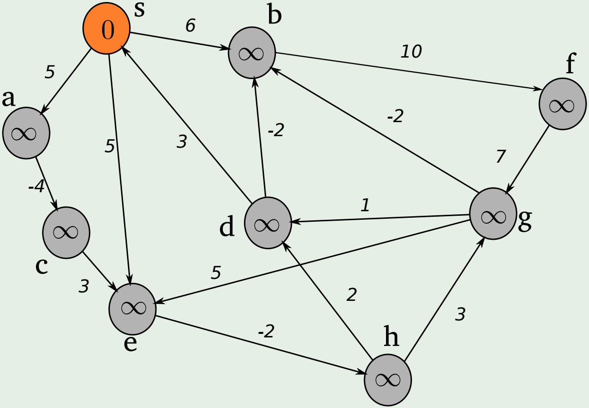
```
Bellman-Ford($G, s, w$)
    Initialize($G, s$)
    for $i = 1$ to $|V[G]| - 1$
        for each $(u, v)$ to $E[G]$
            Relax($u, v, w$)◁ The Decision Part of the Dynamic Programming for $u.d$ and $u.\pi$.
    for each $(u, v)$ to $E[G]$
        if $v.d > u.d + w(u, v)$
            return false
    return true
```

Observation

If Bellman-Ford has not converged after $V(G) - 1$ iterations, then there cannot be a shortest path tree, so there must be a negative weight cycle.

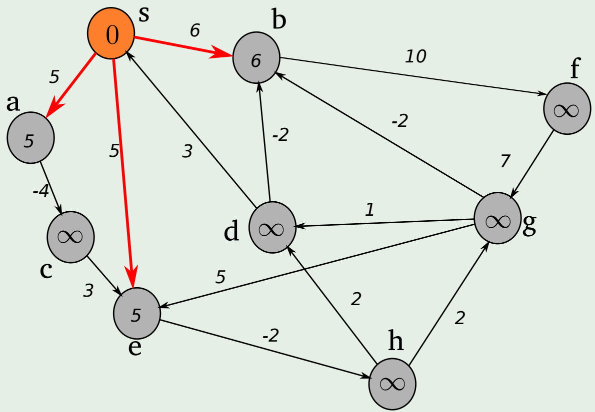
Example

Red Arrows are the representation of $v.\pi$



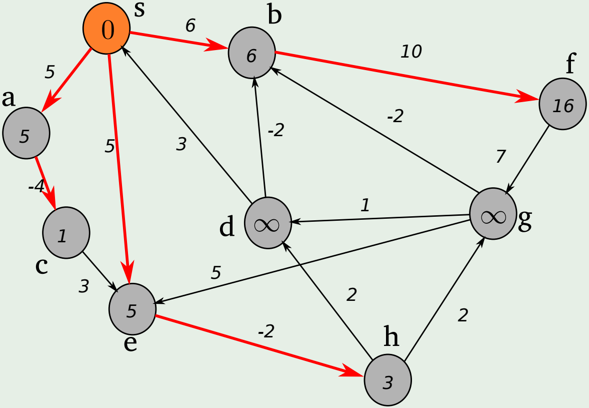
Example

Here, whenever we have $v.d = \infty$ and $v.u = \infty$ no change is done



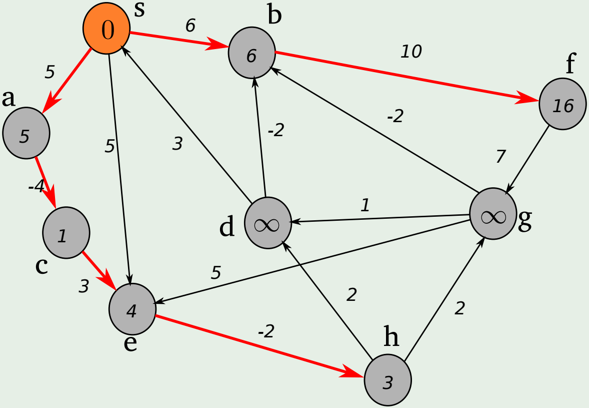
Example

Here, we keep updating in the second iteration



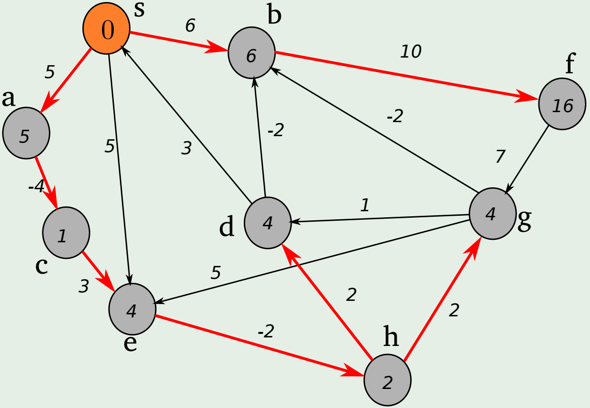
Example

Here, during it. we notice that e can be updated for a better value



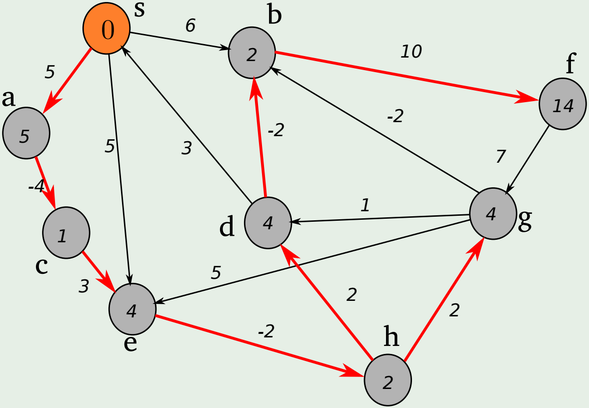
Example

Here, we keep updating in third iteration and d and g also is updated



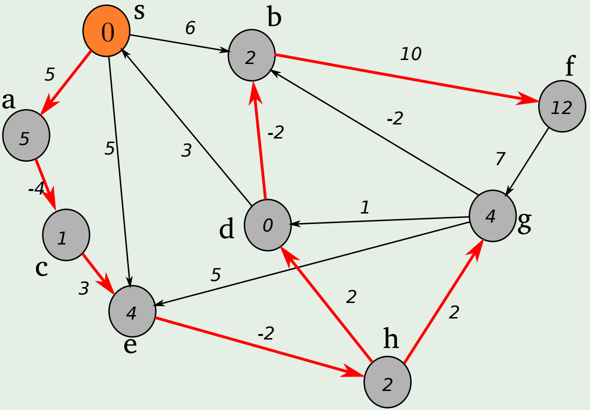
Example

Here, we keep updating in fourth iteration



Example

Here, f is updated during this iteration



Outline

$v.d == \delta(s, v)$ upon termination

Lemma 24.2

Assuming no negative weight cycles reachable from s, $v.d == \delta(s, v)$ holds upon termination for all vertices v reachable from s.

Proof

Consider a shortest path p, where $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$.

We know the following

Shortest paths are simple, p has at most $|V| - 1$, thus we have that $k \leq |V| - 1$.

$v.d == \delta(s, v)$ upon termination

Lemma 24.2

Assuming no negative weight cycles reachable from s, $v.d == \delta(s, v)$ holds upon termination for all vertices v reachable from s.

Proof

Consider a shortest path p, where $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$.

We know the following

Shortest paths are simple, p has at most $|V| - 1$, thus we have that $k \leq |V| - 1$.

$v.d == \delta(s, v)$ upon termination

Lemma 24.2

Assuming no negative weight cycles reachable from s, $v.d == \delta(s, v)$ holds upon termination for all vertices v reachable from s.

Proof

Consider a shortest path p, where $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$.

We know the following

Shortest paths are simple, p has at most $|V| - 1$, thus we have that $k \leq |V| - 1$.

Proof

Something Notable

Claim: $v_i.d = \delta(s, v_i)$ holds after the ith pass over edges.

Proof

Something Notable

Claim: $v_i.d = \delta(s, v_i)$ holds after the ith pass over edges.

In the algorithm

Each of the $|V| - 1$ iterations of the **for** loop (Lines 2-4) relaxes all edges in E.

- By lemma 24.11, once $v_i.d = \delta(s, v_i)$ holds, it continues to hold.

Proof

Something Notable

Claim: $v_i.d = \delta(s, v_i)$ holds after the ith pass over edges.

In the algorithm

Each of the $|V| - 1$ iterations of the **for** loop (Lines 2-4) relaxes all edges in E.

Proof follows by induction on i

- The edges relaxed in the i th iteration, for $i = 1, 2, ..., k$, is (v_{i-1}, v_i).
- By lemma 24.11, once $v_i.d = \delta(s, v_i)$ holds, it continues to hold.

Proof

Something Notable

Claim: $v_i.d = \delta(s, v_i)$ holds after the ith pass over edges.

In the algorithm

Each of the $|V| - 1$ iterations of the **for** loop (Lines 2-4) relaxes all edges in E.

Proof follows by induction on i

- The edges relaxed in the i th iteration, for $i = 1, 2, ..., k$, is (v_{i-1}, v_i).
- By lemma 24.11, once $v_i.d = \delta(s, v_i)$ holds, it continues to hold.

Finding a path between s and v

Corollary 24.3

Let $G = (V, E)$ be a weighted, directed graph with source vertex s and weight function $w : E \to \mathbb{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, for each vertex $v \in V$, there is a path from s to v if and only if Bellman-Ford terminates with $v.d < \infty$ when it is run on G

Proof

Left to you...

Finding a path between s and v

Corollary 24.3

Let $G = (V, E)$ be a weighted, directed graph with source vertex s and weight function $w : E \to \mathbb{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, for each vertex $v \in V$, there is a path from s to v if and only if Bellman-Ford terminates with $v.d < \infty$ when it is run on G

Proof

Left to you...

Outline

Correctness of Bellman-Ford

Claim: The Algorithm returns the correct value

Part of Theorem 24.4. Other parts of the theorem follow easily from earlier results.

Correctness of Bellman-Ford

Claim: The Algorithm returns the correct value

Part of Theorem 24.4. Other parts of the theorem follow easily from earlier results.

Case 1: There is no reachable negative weight cycle.

Upon termination, we have for all (u, v):

$$v.d = \delta(s, v)$$

by lemma 24.2 (last slide) if v is reachable or $v.d = \delta(s, v) = \infty$ otherwise.

Correctness of Bellman-Ford

Claim: The Algorithm returns the correct value

Part of Theorem 24.4. Other parts of the theorem follow easily from earlier results.

Case 1: There is no reachable negative weight cycle.

Upon termination, we have for all (u, v):

$$v.d = \delta(s, v)$$

by lemma 24.2 (last slide) if v is reachable or $v.d = \delta(s, v) = \infty$ otherwise.

Correctness of Bellman-Ford

Then, we have that

$$
\begin{aligned}
v.d = \delta(s,v) \\
&\leq \delta(s,u) + w(u,v) \\
&\leq u.d + w(u,v)
\end{aligned}
$$

Remember:

```
for each $(u,v)$ to $E[G]$
    if $v.d > u.d + w(u,v)$
        return false
```

Correctness of Bellman-Ford

Then, we have that

$$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \\ &\leq u.d + w(u, v) \end{aligned}$$

Remember:

```
for each $(u, v)$ to $E[G]$
    if $v.d > u.d + w(u, v)$
        return false
```

Thus

So algorithm returns ***true***.

Correctness of Bellman-Ford

Then, we have that

$$\begin{aligned} v.d &= \delta(s,v) \\ &\leq \delta(s,u) + w(u,v) \\ &\leq u.d + w\left(u,v\right) \end{aligned}$$

Remember:

```
for each $(u,v)$ to $E\left[G\right]$
    if $v.d > u.d + w\left(u,v\right)$
        return false
```

Thus

So algorithm returns ***true***.

Correctness of Bellman-Ford

Then, we have that

$$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \\ &\leq u.d + w\left(u, v\right) \end{aligned}$$

Remember:

```
for each $(u, v)$ to $E\left[G\right]$
    if $v.d > u.d + w\left(u, v\right)$
        return false
```

Thus

So algorithm returns *true*.

Correctness of Bellman-Ford

Then, we have that

$$\begin{aligned} v.d &= \delta(s,v) \\ &\leq \delta(s,u) + w(u,v) \\ &\leq u.d + w(u,v) \end{aligned}$$

Remember:

```
5. for each $(u,v)$ to $E[G]$
6.     if $v.d > u.d + w(u,v)$
7.         return false
```

Thus

So algorithm returns ***true***.

Correctness of Bellman-Ford

Case 2: There exists a reachable negative weight cycle $c =< v_0, v_1, ..., v_k >$, where $v_0 = v_k$.

Then, we have:

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0. \tag{7}$$

Suppose algorithm returns *true*

Then $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$ for $i = 1, ..., k$ because Relax did not change any $v_i.d$.

Correctness of Bellman-Ford

Case 2: There exists a reachable negative weight cycle $c =< v_0, v_1, ..., v_k >$, where $v_0 = v_k$.

Then, we have:

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0. \tag{7}$$

Suppose algorithm returns *true*

Then $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$ for $i = 1, ..., k$ because Relax did not change any $v_i.d$.

Correctness of Bellman-Ford

Thus

$$\begin{aligned}\sum_{i=1}^{k} v_i.d &\leq \sum_{i=1}^{k} \left(v_{i-1}.d + w(v_{i-1}, v_i)\right)\\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)\end{aligned}$$

Since $v_0 = v_k$

Each vertex in c appears exactly once in each of the summations, $\sum_{i=1}^{k} v_i.d$ and $\sum_{i=1}^{k} v_{i-1}.d$, thus

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d \qquad (8)$$

Correctness of Bellman-Ford

Thus

$$\begin{aligned}\sum_{i=1}^{k} v_i.d &\leq \sum_{i=1}^{k} \left(v_{i-1}.d + w(v_{i-1}, v_i)\right)\\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)\end{aligned}$$

Since $v_0 = v_k$

Each vertex in c appears exactly once in each of the summations, $\sum\limits_{i=1}^{k} v_i.d$ and $\sum\limits_{i=1}^{k} v_{i-1}.d$, thus

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d \tag{8}$$

Correctness of Bellman-Ford

By Corollary 24.3

$v_i.d$ is finite for $i = 1, 2, ..., k$, thus

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

Hence

This contradicts (Eq. 7). Thus, algorithm returns *false*.

Correctness of Bellman-Ford

By Corollary 24.3

$v_i.d$ is finite for $i = 1, 2, ..., k$, thus

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

Hence

This contradicts (Eq. 7). Thus, algorithm returns ***false***.

Outline

1. Introduction
   - Introduction and Similar Problems
2. General Results
   - Optimal Substructure Properties
   - Predecessor Graph
   - The Relaxation Concept
   - The Bellman-Ford Algorithm
   - Properties of Relaxation
3. Bellman-Ford Algorithm
   - Predecessor Subgraph for Bellman
   - Shortest Path for Bellman
   - Example
   - Bellman-Ford finds the Shortest Path
   - Correctness of Bellman-Ford
4. Directed Acyclic Graphs (DAG)
   - Relaxing Edges
   - Example
5. Dijkstra's Algorithm
   - Dijkstra's Algorithm: A Greedy Method
   - Example
   - Correctness Dijkstra's algorithm
   - Complexity of Dijkstra's Algorithm
6. Exercises

Another Example

Something Notable

By relaxing the edges of a weighted DAG $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in time.

Why?

Shortest paths are always well defined in a DAG, since even if there are negative-weight edges, no negative-weight cycles can exist.

Another Example

Something Notable

By relaxing the edges of a weighted DAG $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in time.

Why?

Shortest paths are always well defined in a DAG, since even if there are negative-weight edges, no negative-weight cycles can exist.

Single-source Shortest Paths in Directed Acyclic Graphs

In a DAG, we can do the following (Complexity $\Theta\left(V+E\right)$)

```
DAG -SHORTEST-PATHS$(G, w, s)$
    Topological sort vertices in G
    Initialize$(G, s)$
    for each $u$ in $V\left[G\right]$ in topological sorted order
        for each $v$ to $Adj\left[u\right]$
            Relax$(u, v, w)$
```

Single-source Shortest Paths in Directed Acyclic Graphs

In a DAG, we can do the following (Complexity $\Theta(V+E)$)

```
DAG -SHORTEST-PATHS$(G, w, s)$
Topological sort vertices in G
Initialize$(G, s)$
for each $u$ in $V[G]$ in topological sorted order
    for each $v$ to $Adj[u]$
        Relax$(u, v, w)$
```

Single-source Shortest Paths in Directed Acyclic Graphs

In a DAG, we can do the following (Complexity $\Theta(V+E)$)

DAG -SHORTEST-PATHS(G, w, s)

```
Topological sort vertices in G
Initialize$(G, s)$
for each $u$ in $V[G]$ in topological sorted order
    for each $v$ to $Adj[u]$
        Relax$(u, v, w)$
```

It is based in the following theorem

Theorem 24.5

If a weighted, directed graph $G = (V, E)$ has source vertex s and no cycles, then at the termination of the DAG-SHORTEST-PATHS procedure, $v.d = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest path.

Proof

Left to you...

It is based in the following theorem

Theorem 24.5

If a weighted, directed graph $G = (V, E)$ has source vertex s and no cycles, then at the termination of the DAG-SHORTEST-PATHS procedure, $v.d = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest path.

Proof

Left to you...

Complexity

We have that

1. Line 1 takes $\Theta\left(V+E\right)$.
2. Line 2 takes $\Theta\left(V\right)$.
3. Lines 3-5 makes an iteration per vertex:
    1. In addition, the for loop in lines 4-5 relaxes each edge exactly once (Remember the sorting).
    2. Making each iteration of the inner loop $\Theta\left(1\right)$

Complexity

We have that

1. Line 1 takes $\Theta\left(V+E\right)$.
2. Line 2 takes $\Theta\left(V\right)$.
3. Lines 3-5 makes an iteration per vertex:
    1. In addition, the for loop in lines 4-5 relaxes each edge exactly once (Remember the sorting).
    2. Making each iteration of the inner loop $\Theta\left(1\right)$

Therefore

The total running time is equal to $\Theta\left(V+E\right)$.

Complexity

We have that

1. Line 1 takes $\Theta\left(V+E\right)$.
2. Line 2 takes $\Theta\left(V\right)$.
3. Lines 3-5 makes an iteration per vertex:
    1. In addition, the for loop in lines 4-5 relaxes each edge exactly once (Remember the sorting).
    2. Making each iteration of the inner loop $\Theta\left(1\right)$

Therefore

The total running time is equal to $\Theta\left(V+E\right)$.

Complexity

We have that

1. Line 1 takes $\Theta\left(V+E\right)$.
2. Line 2 takes $\Theta\left(V\right)$.
3. Lines 3-5 makes an iteration per vertex:
    1. In addition, the for loop in lines 4-5 relaxes each edge exactly once (Remember the sorting).
    2. Making each iteration of the inner loop $\Theta\left(1\right)$

Therefore

The total running time is equal to $\Theta\left(V+E\right)$.

Complexity

We have that

1. Line 1 takes $\Theta\left(V+E\right)$.
2. Line 2 takes $\Theta\left(V\right)$.
3. Lines 3-5 makes an iteration per vertex:
    1. In addition, the for loop in lines 4-5 relaxes each edge exactly once (Remember the sorting).
    2. Making each iteration of the inner loop $\Theta\left(1\right)$

Therefore

The total running time is equal to $\Theta\left(V+E\right)$.

Complexity

We have that

1. Line 1 takes $\Theta\left(V+E\right)$.
2. Line 2 takes $\Theta\left(V\right)$.
3. Lines 3-5 makes an iteration per vertex:
    1. In addition, the for loop in lines 4-5 relaxes each edge exactly once (Remember the sorting).
    2. Making each iteration of the inner loop $\Theta\left(1\right)$

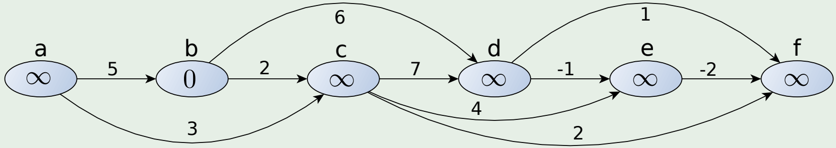
Therefore

The total running time is equal to $\Theta\left(V+E\right)$.

Outline

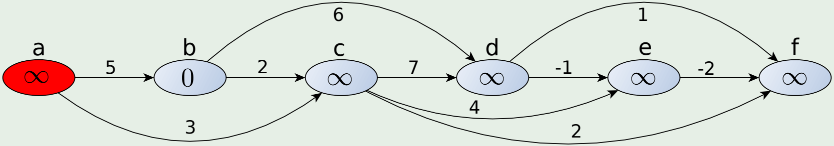
Example

After Initialization, we have **b** is the source



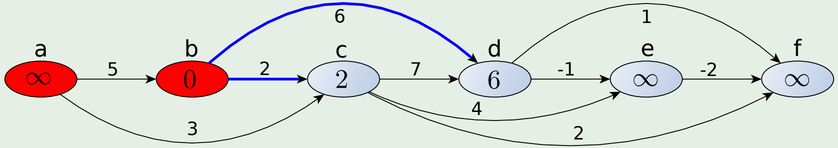
Example

a is the first in the topological sort, but no update is done



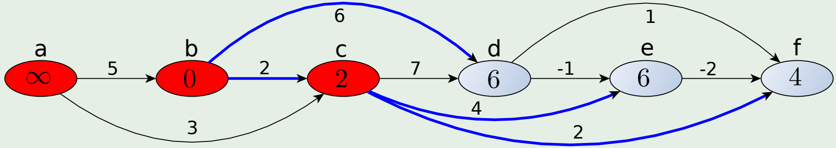
Example

b is the next one



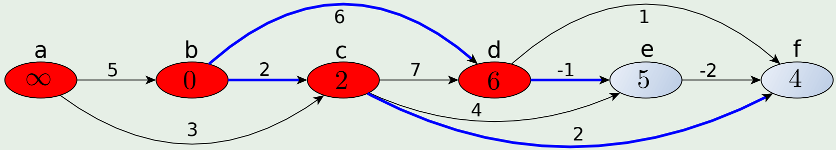
Example

c is the next one



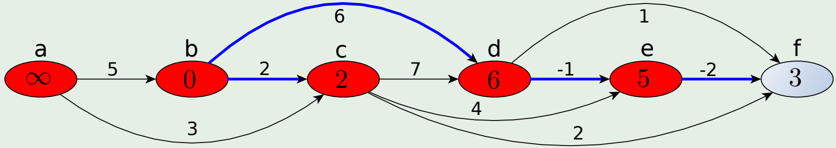
Example

d is the next one



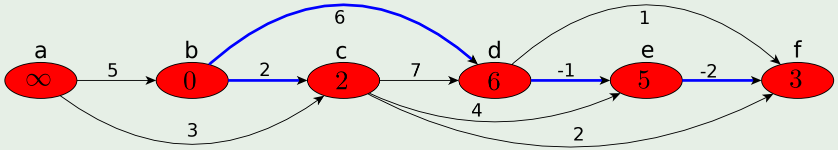
Example

e is the next one



Example

Finally, **w** is the next one



Outline

Dijkstra's Algorithm

It is a greedy based method

Ideas?

Yes

We need to keep track of the greedy choice!!!

Dijkstra's Algorithm

It is a greedy based method

Ideas?

Yes

We need to keep track of the greedy choice!!!

Dijkstra's Algorithm

Assume no negative weight edges

1. **Dijkstra's algorithm maintains a set S of vertices whose shortest path from s has been determined.**
2. It repeatedly selects u in $V - S$ with minimum shortest path estimate (greedy choice).
3. It store $V - S$ in priority queue Q.

Dijkstra's Algorithm

Assume no negative weight edges

1. **Dijkstra's algorithm maintains a set S of vertices whose shortest path from s has been determined.**
2. It repeatedly selects u in $V - S$ with minimum shortest path estimate (greedy choice).
3. It store $V - S$ in priority queue Q.

Dijkstra's Algorithm

Assume no negative weight edges

1. **Dijkstra's algorithm maintains a set S of vertices whose shortest path from s has been determined.**
2. It repeatedly selects u in $V - S$ with minimum shortest path estimate (greedy choice).
3. It store $V - S$ in priority queue Q.

Dijkstra's algorithm

```
DIJKSTRA($G, w, s$)
INITIALIZE($G, s$)
$S = \emptyset$
$Q = V[G]$
while $Q \neq \emptyset$
    $u$ =Extract-Min($Q$)
    $S = S \cup \{u\}$
    for each vertex $v \in Adj[u]$
        Relax(u,v,w)
```

Dijkstra's algorithm

```
DIJKSTRA$(G, w, s)$
INITIALIZE$(G, s)$
$S = \emptyset$
$Q = V[G]$
while $Q \neq \emptyset$
        $u =$Extract-Min$(Q)$
        $S = S \cup \{u\}$
        for each vertex $v \in Adj[u]$
                Relax(u,v,w)
```

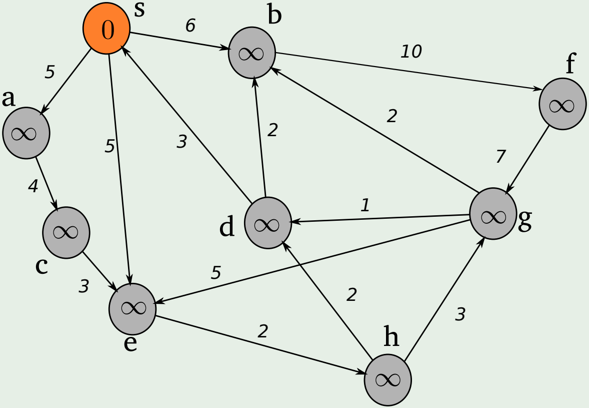
Dijkstra's algorithm

```
DIJKSTRA($G, w, s$)
INITIALIZE($G, s$)
$S = \emptyset$
$Q = V[G]$
while $Q \neq \emptyset$
        $u =$Extract-Min($Q$)
        $S = S \cup \{u\}$
        for each vertex $v \in Adj[u]$
                Relax(u,v,w)
```

Outline

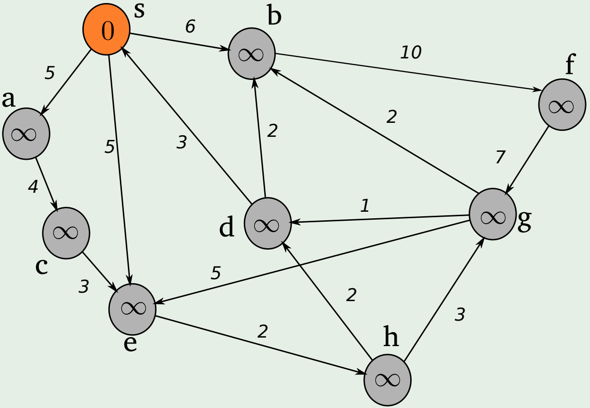
Example

The Graph After Initialization



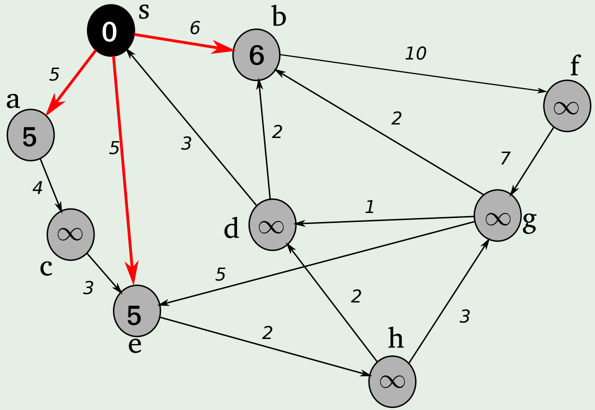
Example

We use red edges to represent $v.\pi$ and color black to represent the set S



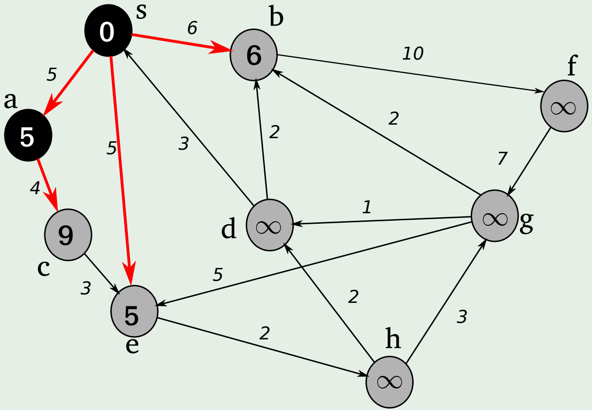
Example

$s \leftarrow$Extract-Min(Q) and update the elements adjacent to s



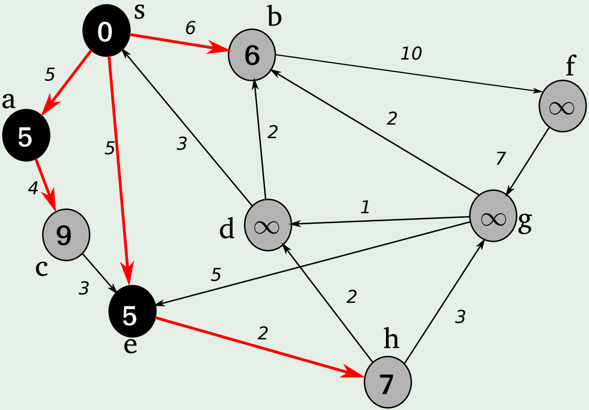
Example

a←Extract-Min(Q) and update the elements adjacent to a



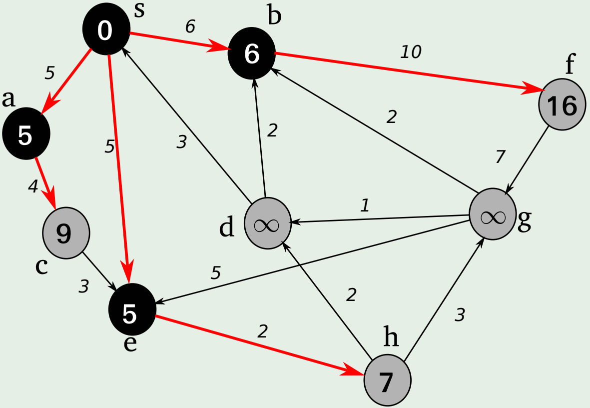
Example

e←Extract-Min(Q) and update the elements adjacent to e



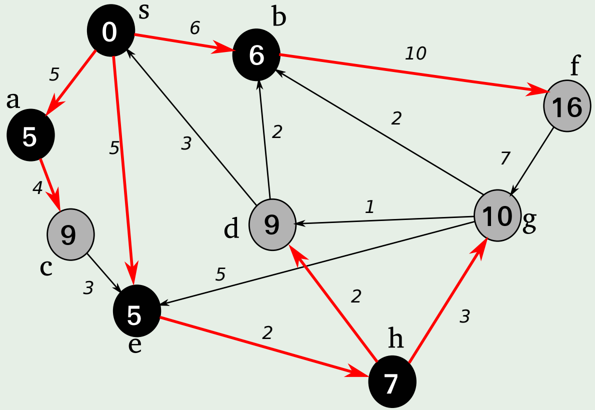
Example

b←Extract-Min(Q) and update the elements adjacent to b



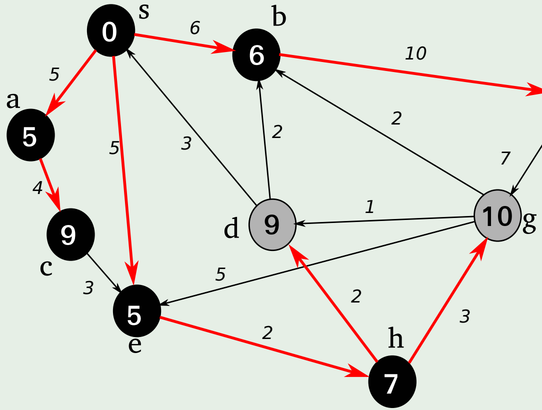
Example

h←Extract-Min(Q) and update the elements adjacent to h



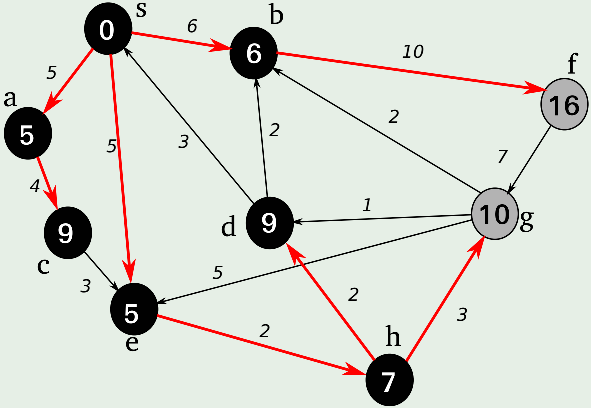
Example

c←Extract-Min(Q) and no-update



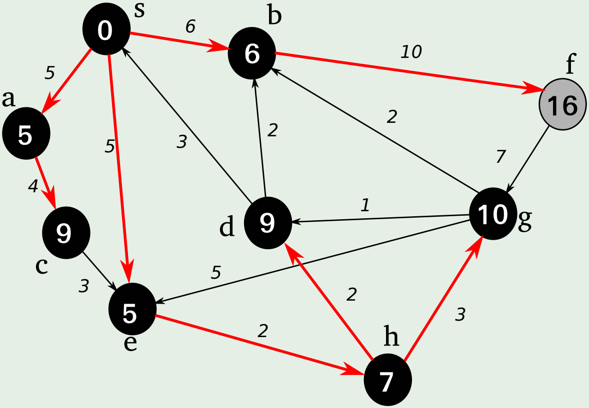
Example

d←Extract-Min(Q) and no-update



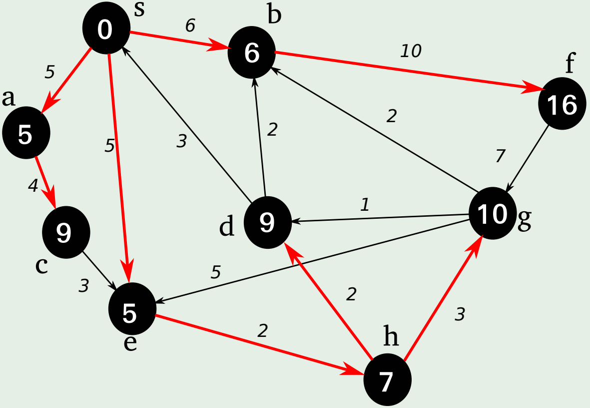
Example

g←Extract-Min(Q) and no-update



Example

f←Extract-Min(Q) and no-update



Outline

Correctness Dijkstra's algorithm

Theorem 24.6

Upon termination, $u.d = \delta(s, u)$ for all u in V (assuming non negative weights).

Proof

By lemma 24.11, once $u.d = \delta(s, u)$ holds, it continues to hold.

We are going to use the following loop Invariance

At the start of each iteration of the while loop of lines 4–8, $v.d = \delta(s, v)$ for each vertex $v \in S$.

Correctness Dijkstra's algorithm

Theorem 24.6

Upon termination, $u.d = \delta(s, u)$ for all u in V (assuming non negative weights).

Proof

By lemma 24.11, once $u.d = \delta(s, u)$ holds, it continues to hold.

We are going to use the following loop Invariance

At the start of each iteration of the while loop of lines 4–8, $v.d = \delta(s, v)$ for each vertex $v \in S$.

Correctness Dijkstra's algorithm

Theorem 24.6

Upon termination, $u.d = \delta(s, u)$ for all u in V (assuming non negative weights).

Proof

By lemma 24.11, once $u.d = \delta(s, u)$ holds, it continues to hold.

We are going to use the following loop Invariance

At the start of each iteration of the while loop of lines 4–8, $v.d = \delta\left(s, v\right)$ for each vertex $v \in S$.

Proof

Thus

We are going to prove for each u in V, $u.d = \delta(s,u)$ when u is inserted in S.

Proof

Thus

We are going to prove for each u in V, $u.d = \delta(s, u)$ when u is inserted in S.

Initialization

Initially $S = \emptyset$, thus the invariant is true.

- In addition, we have that $S \neq \emptyset$ before u is added.

Proof

Thus

We are going to prove for each u in V, $u.d = \delta(s, u)$ when u is inserted in S.

Initialization

Initially $S = \emptyset$, thus the invariant is true.

Maintenance

We want to show that in each iteration $u.d = \delta\left(s, u\right)$ for the vertex added to set S.

- In addition, we have that $S \neq \emptyset$ before u is added.

Proof

Thus

We are going to prove for each u in V, $u.d = \delta(s, u)$ when u is inserted in S.

Initialization

Initially $S = \emptyset$, thus the invariant is true.

Maintenance

We want to show that in each iteration $u.d = \delta\left(s, u\right)$ for the vertex added to set S.

For this, note the following

- Note that $s.d = \delta(s, s) = 0$ when s is inserted, so $u \neq s$.
- In addition, we have that $S \neq \emptyset$ before u is added.

Proof

Thus

We are going to prove for each u in V, $u.d = \delta(s,u)$ when u is inserted in S.

Initialization

Initially $S = \emptyset$, thus the invariant is true.

Maintenance

We want to show that in each iteration $u.d = \delta\left(s,u\right)$ for the vertex added to set S.

For this, note the following

- Note that $s.d = \delta(s,s) = 0$ when s is inserted, so $u \neq s$.
- In addition, we have that $S \neq \emptyset$ before u is added.

Proof

Use contradiction

Now, suppose not. Let u be the first vertex such that $u.d \neq \delta$ (s,u) when inserted in S.

Note the following

Note that $s.d = \delta(s, s) = 0$ when s is inserted, so $u \neq s$; thus $S \neq \emptyset$ just before u is inserted (in fact $s \in S$).

Proof

Use contradiction

Now, suppose not. Let u be the first vertex such that $u.d \neq \delta$ (s,u) when inserted in S.

Note the following

Note that $s.d = \delta(s,s) = 0$ when s is inserted, so $u \neq s$; thus $S \neq \varnothing$ just before u is inserted (in fact $s \in S$).

Proof

Now

Note that there exist a path from s to u, for otherwise $u.d = \delta(s, u) = \infty$ by corollary 24.12.

- **“If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant.”**

Thus exist a shortest path p

Between s and u.

Observation

Prior to adding u to S, path p connects a vertex in S, namely s, to a vertex in $V - S$, namely u.

Proof

Now

Note that there exist a path from s to u, for otherwise $u.d = \delta(s,u) = \infty$ by corollary 24.12.

- **“If there is no path from s to v, then $v.d = \delta(s,v) = \infty$ is an invariant.”**

Thus exist a shortest path p

Between s and u.

Observation

Prior to adding u to S, path p connects a vertex in S, namely s, to a vertex in $V - S$, namely u.

Proof

Now

Note that there exist a path from s to u, for otherwise $u.d = \delta(s, u) = \infty$ by corollary 24.12.

- **"If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ is an invariant."**

Thus exist a shortest path p

Between s and u.

Observation

Prior to adding u to S, path p connects a vertex in S, namely s, to a vertex in $V - S$, namely u.

Proof

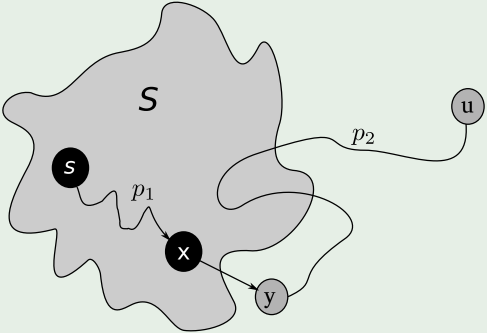
Consider the following

- The first y along p from s to u such that $y \in V - S$.
- And let $x \in S$ be y's predecessor along p.

Proof

Proof (continuation)

Then, shortest path from s to u: $s \overset{p_1}{\rightsquigarrow} x \to y \overset{p_2}{\rightsquigarrow} u$ looks like...



Remark: Either of paths p_1 or p_2 may have no edges.

Proof

We claim

$y.d = \delta(s, y)$ when u is added into S.

Proof of the claim

1. Observe that $x \in S$.

Proof

We claim

$y.d = \delta(s, y)$ when u is added into S.

Proof of the claim

1. Observe that $x \in S$.
2. In addition, we know that u is the first vertex for which $u.d \neq \delta(s, u)$ when it id added to S

Proof

Then

In addition, we had that $x.d = \delta(s, x)$ when x was inserted into S.

Proof

Then

In addition, we had that $x.d = \delta(s,x)$ when x was inserted into S.

Then, we relaxed the edge between x and y

Edge (x,y) was relaxed at that time!

Proof

Remember? Convergence property (Lemma 24.14)

Let p be a shortest path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \rightarrow v$. If $u.d = \delta(s, u)$ holds at any time prior to calling $Relax(u, v, w)$, then $v.d = \delta(s, v)$ holds at all times after the call.

$$y.d = \delta(s, y) = \delta(s, x) + w(x, y) \tag{9}$$

- The claim is implied!!!

Proof

Remember? Convergence property (Lemma 24.14)

Let p be a shortest path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \rightarrow v$. If $u.d = \delta(s,u)$ holds at any time prior to calling $Relax(u,v,w)$, then $v.d = \delta(s,v)$ holds at all times after the call.

Then

- **Then, using this convergence property.**

$$y.d = \delta(s,y) = \delta(s,x) + w(x,y) \tag{9}$$

- The claim is implied!!!

Proof

Remember? Convergence property (Lemma 24.14)

Let p be a shortest path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \rightarrow v$. If $u.d = \delta(s, u)$ holds at any time prior to calling $Relax(u, v, w)$, then $v.d = \delta(s, v)$ holds at all times after the call.

Then

- **Then, using this convergence property.**

$$y.d = \delta(s, y) = \delta(s, x) + w(x, y) \tag{9}$$

- The claim is implied!!!

Proof

Remember? Convergence property (Lemma 24.14)

Let p be a shortest path from s to v, where $p = s \overset{p_1}{\rightsquigarrow} u \rightarrow v$. If $u.d = \delta(s, u)$ holds at any time prior to calling $Relax(u, v, w)$, then $v.d = \delta(s, v)$ holds at all times after the call.

Then

- **Then, using this convergence property.**

$$y.d = \delta(s, y) = \delta(s, x) + w(x, y) \tag{9}$$

- The claim is implied!!!

Proof

Now

1. We obtain a contradiction to prove that $u.d = \delta\left(s,u\right)$.
2. y appears before u in a shortest path on a shortest path from s to u.
3. In addition, all edges have positive weights.
4. Then, $\delta(s,y) \leq \delta(s,u)$, thus

$$\begin{aligned} y.d &= \delta\left(s,y\right) \\ &\leq \delta\left(s,u\right) \\ &\leq u.d \end{aligned}$$

    - The last inequality is due to the Upper-Bound Property (Lemma 24.11).

Proof

Now

1. We obtain a contradiction to prove that $u.d = \delta(s, u)$.
2. y appears before u in a shortest path on a shortest path from s to u.
3. In addition, all edges have positive weights.
4. Then, $\delta(s, y) \leq \delta(s, u)$, thus

$$\begin{aligned} y.d &= \delta(s, y) \\ &\leq \delta(s, u) \\ &\leq u.d \end{aligned}$$

   - The last inequality is due to the Upper-Bound Property (Lemma 24.11).

Proof

Now

1. We obtain a contradiction to prove that $u.d = \delta(s, u)$.
2. y appears before u in a shortest path on a shortest path from s to u.
3. In addition, all edges have positive weights.
4. Then, $\delta(s, y) \leq \delta(s, u)$, thus

$$\begin{aligned} y.d &= \delta(s, y) \\ &\leq \delta(s, u) \\ &\leq u.d \end{aligned}$$

    - The last inequality is due to the Upper-Bound Property (Lemma 24.11).

Proof

Now

1. We obtain a contradiction to prove that $u.d = \delta(s, u)$.
2. y appears before u in a shortest path on a shortest path from s to u.
3. In addition, all edges have positive weights.
4. Then, $\delta(s, y) \leq \delta(s, u)$, thus

$$\begin{aligned} y.d &= \delta(s, y) \\ &\leq \delta(s, u) \\ &\leq u.d \end{aligned}$$

- The last inequality is due to the Upper-Bound Property (Lemma 24.11).

Proof

Now

1. We obtain a contradiction to prove that $u.d = \delta(s,u)$.
2. y appears before u in a shortest path on a shortest path from s to u.
3. In addition, all edges have positive weights.
4. Then, $\delta(s,y) \leq \delta(s,u)$, thus

$$\begin{aligned} y.d &= \delta(s,y) \\ &\leq \delta(s,u) \\ &\leq u.d \end{aligned}$$

- The last inequality is due to the Upper-Bound Property (Lemma 24.11).

Proof

Now

1. We obtain a contradiction to prove that $u.d = \delta\left(s, u\right)$.
2. y appears before u in a shortest path on a shortest path from s to u.
3. In addition, all edges have positive weights.
4. Then, $\delta(s, y) \leq \delta(s, u)$, thus

$$\begin{aligned} y.d &= \delta\left(s, y\right) \\ &\leq \delta\left(s, u\right) \\ &\leq u.d \end{aligned}$$

- The last inequality is due to the Upper-Bound Property (Lemma 24.11).

Proof

Now

1. We obtain a contradiction to prove that $u.d = \delta\left(s, u\right)$.
2. y appears before u in a shortest path on a shortest path from s to u.
3. In addition, all edges have positive weights.
4. Then, $\delta(s, y) \leq \delta(s, u)$, thus

$$\begin{aligned} y.d &= \delta\left(s, y\right) \\ &\leq \delta\left(s, u\right) \\ &\leq u.d \end{aligned}$$

  - The last inequality is due to the Upper-Bound Property (Lemma 24.11).

Proof

Then

But because both vertices u and y where in $V - S$ when u was chosen in line 5 $\Rightarrow u.d \leq y.d$.

Proof

Then

But because both vertices u and y where in $V - S$ when u was chosen in line 5 $\Rightarrow u.d \leq y.d$.

Thus

$$y.d = \delta(s, y) = \delta(s, u) = u.d$$

- Conclusion: $u.d = \delta(s, u)$ when u is added to S and the equality is maintained afterwards

Proof

Then

But because both vertices u and y where in $V - S$ when u was chosen in line 5 $\Rightarrow u.d \leq y.d$.

Thus

$$y.d = \delta(s, y) = \delta(s, u) = u.d$$

Consequently

- We have that $u.d = \delta\left(s, u\right)$, which contradicts our choice of u.
- Conclusion: $u.d = \delta\left(s, u\right)$ when u is added to S and the equality is maintained afterwards.

Proof

Then

But because both vertices u and y where in $V - S$ when u was chosen in line 5 $\Rightarrow u.d \leq y.d$.

Thus

$$y.d = \delta(s, y) = \delta(s, u) = u.d$$

Consequently

- We have that $u.d = \delta(s, u)$, which contradicts our choice of u.
- Conclusion: $u.d = \delta(s, u)$ when u is added to S and the equality is maintained afterwards.

Finally

Termination

- At termination $Q = \emptyset$
- Thus, $V - S = \emptyset$ or equivalent $S = V$

Thus

$u.d = \delta(s, u)$ for all vertices $u \in V$!!!

Finally

Termination

- At termination $Q = \emptyset$
- Thus, $V - S = \emptyset$ or equivalent $S = V$

Thus

$u.d = \delta(s, u)$ for all vertices $u \in V$!!!

Outline

Complexity

Running time is

$O(V^2)$ using linear array for priority queue.
$O((V+E)\log V)$ using binary heap.
$O(V \log V + E)$ using Fibonacci heap.

Exercises

From Cormen's book solve

- 24.1-1
- 24.1-3
- 24.1-4
- 23.3-1
- 23.3-3
- 23.3-4
- 23.3-6
- 23.3-7
- 23.3-8
- 23.3-10