Teoria dos Grafos e Computabilidade

— Shortest path —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Aug 2022

Teoria dos Grafos e Computabilidade

— Some graph fundamentals —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Aug 2022

Graphs

- Model pairwise relationships (edges) between objects (nodes or vertices).
- **Undirected graph** $G = (V, E)$: set V of nodes and set E of edges, where $E \subseteq V \times V$. Elements of E are unordered pairs.
- **Directed graph** $G = (V, E)$: set V of nodes and set E of edges, where $E \subseteq V \times V$. Elements of E are ordered pairs.

Applications of Graphs

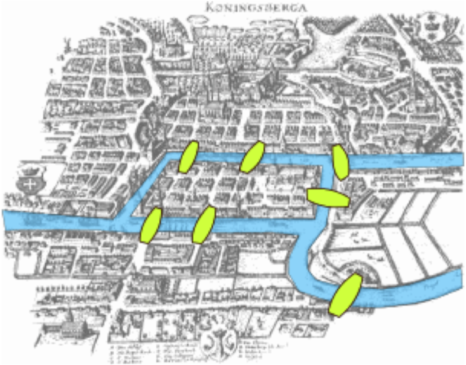
- Useful in a large number of applications:

Applications of Graphs

- Useful in a large number of applications: computer networks, the World Wide Web, ecology (food webs), social networks, software systems, job scheduling, VLSI circuits, cellular networks, . . .
- Problems involving graphs have a rich history dating back to Euler.

Applications of Graphs

- Useful in a large number of applications: computer networks, the World Wide Web, ecology (food webs), social networks, software systems, job scheduling, VLSI circuits, cellular networks, ...
- Problems involving graphs have a rich history dating back to Euler.



Questions?

Shortest path

– Some graph fundamentals –

Teoria dos Grafos e Computabilidade

— Shortest Path Problem —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Aug 2022

Shortest Path Problem

- $G = (V, E)$ is a connected directed graph. Each edge e has a length $l_e \geq 0$.
- V has n nodes and E has m edges.
- Length of a path P is the sum of lengths of the edges in P.
- Goal is to determine the shortest path from some start node s to each node in V.
- Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

Shortest Path Problem

- $G = (V, E)$ is a connected directed graph. Each edge e has a length $l_e \geq 0$.
- V has n nodes and E has m edges.
- Length of a path P is the sum of lengths of the edges in P.
- Goal is to determine the shortest path from some start node s to each node in V.
- Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

Shortest Paths

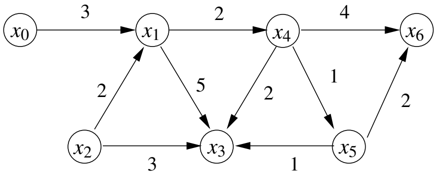
INSTANCE A directed graph $G = (V, E)$, a function $l : E \rightarrow \mathbb{R}^+$, and a node $s \in V$

SOLUTION A set $\{P_u, u \in V\}$, where P_u is the shortest path in G from s to u.

- Let $N = (G, W)$ be a positive length graph, let $x \in V$
- We define the map $L_x : V \to \mathbb{R} \cup \{\infty\}$ by:

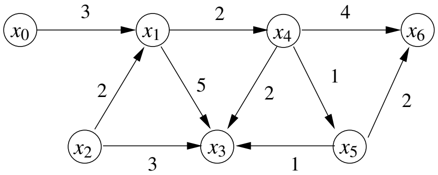
$$L_x(y) = \begin{cases} \text{the length of a shortest path from } x \text{ to } y\text{, if such path exists} \\ \infty \text{ , otherwise} \end{cases}$$

Illustration: the map L_x

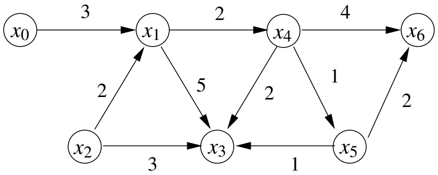


$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$							

Illustration: the map L_x

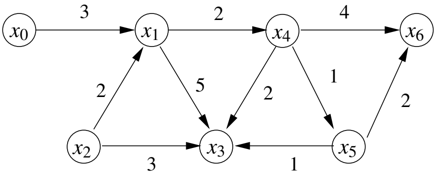


$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$	0						



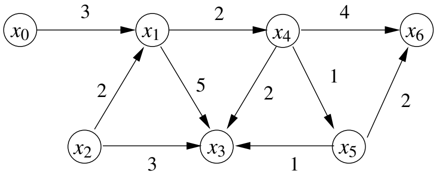
$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$	0	3					

Illustration: the map L_x



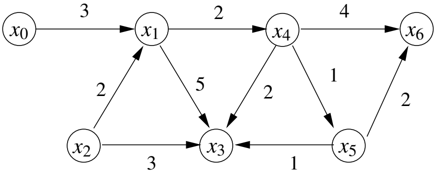
$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$	0	3	∞				

Illustration: the map L_x



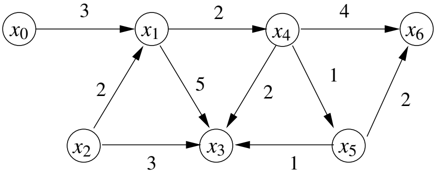
$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$	0	3	∞	7			

Illustration: the map L_x



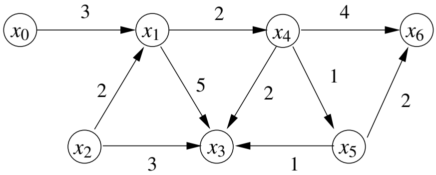
$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$	0	3	∞	7	5		

Illustration: the map L_x



$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$	0	3	∞	7	5	6	

Illustration: the map L_x



$y =$	x_0	x_1	x_2	x_3	x_4	x_5	x_6
$L_{x_0}(y) =$	0	3	∞	7	5	6	8

Questions?

Shortest path

– Shortest Path Problem –

Teoria dos Grafos e Computabilidade

— Algorithms for Single Source Shortest Path —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Aug 2022

Problems

1. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and two vertices x and y in V
   - Find a shortest path from x to y
   - Find the length $L_x(y)$ of a shortest path from x to y
2. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and a vertex x in v
   - Find for each vertex y in V the length $L_x(y)$ of a shortest path from x to y
3. Given a graph $G = (V, \Gamma)$ and a network (G, ℓ)
   - Find, for each pair x, y of vertices in V, the length of a shortest path from x to y
4. Having solved problem 2
   - Solve problem 1

Dijkstra algorithm

1. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and two vertices x and y in V
   - Find a shortest path from x to y
   - Find the length $L_x(y)$ of a shortest path from x to y
2. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and a vertex x in v
   - Find for each vertex y in V the length $L_x(y)$ of a shortest path from x to y
3. Given a graph $G = (V, \Gamma)$ and a network (G, ℓ)
   - Find, for each pair x, y of vertices in V, the length of a shortest path from x to y
4. Having solved problem 2
   - Solve problem 1

Computing the lengths of shortest paths

Algorithm DIJKSTRA (Data: A graph $G = (V, \Gamma)$, a network (G, ℓ), $n = |V|$, $x \in V$;

Result: L_x)

$\bar{S} := \emptyset$;

For each $y \in V$ **Do** $L_x[y] = \infty$; $\bar{S} := \bar{S} \cup \{y\}$;

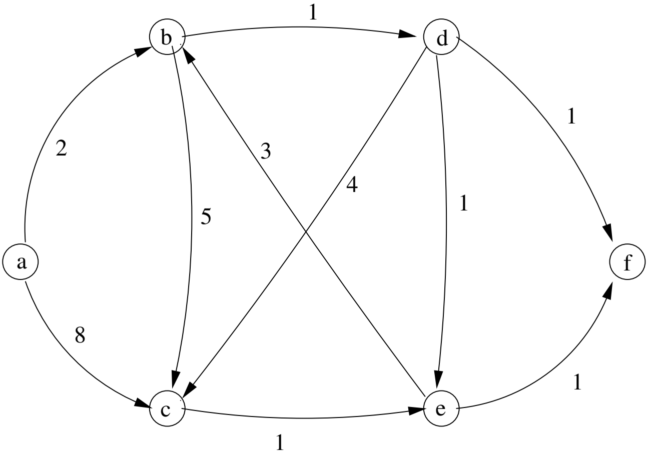
$L_x[x] := 0$; $k := 0$; $\mu := 0$;

While $k < n$ and $\mu \neq \infty$ **Do**

- Extract a vertex $y^\star \in \bar{S}$ such that $L_x[y^\star] = \min\{L_x[y], y \in \bar{S}\}$
- $k++$; $\mu := L_x[y^\star]$;
- **For each** $y \in \Gamma(y^\star) \cap \bar{S}$ **Do**
  - $L_x[y] := \min\{L_x[y], L_x[y^\star] + \ell(y^\star, y)\}$;

Computing the lengths of shortest paths

- *Exercise.* Execute "by hand" Dijsktra algorithm on the following network with $x = a$, and on any positive length network of your choice



Loop invariant of Dijkstra algorithm (# 1)

- Let $x \in V$ and $\mu \in \mathbb{R}$
- A subset S of V is called a μ-separating (for x) if the two following conditions hold true:

Loop invariant of Dijkstra algorithm (# 1)

- Let $x \in V$ and $\mu \in \mathbb{R}$
- A subset S of V is called a **μ-separating (for x)** if the two following conditions hold true:
  1. S contains any vertex y such that the length $L_x(y)$ of a shortest path from x to y is less than μ

Loop invariant of Dijkstra algorithm (# 1)

- Let $x \in V$ and $\mu \in \mathbb{R}$
- A subset S of V is called a **μ-separating (for x)** if the two following conditions hold true:
  1. S contains any vertex y such that the length $L_x(y)$ of a shortest path from x to y is less than μ
  2. $\bar{S} = V \setminus S$ contains any vertex y such that the length of a shortest path from x to y is greater than μ

Loop invariant of Dijkstra algorithm (# 2)

- Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ-separating for x
- An S-path is a path whose intermediary vertices are all in S

Loop invariant of Dijkstra algorithm (# 2)

- Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ-separating for x
- An S-path is a path whose intermediary vertices are all in S
- The length of a shortest S-path from x to y is denoted by $L_x^S(y)$

Loop invariant of Dijkstra algorithm (# 2)

- Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ-separating for x
- An S-path is a path whose intermediary vertices are all in S
- The length of a shortest S-path from x to y is denoted by $L_x^S(y)$

proof of Dijkstra algorithm

- Let $y^\star \in \bar{S}$ such that $L_x^S(y^\star) = \min\{L_x^s(y) \mid y \in \bar{S}\}$

Loop invariant of Dijkstra algorithm (# 2)

- Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ-separating for x
- An S-path is a path whose intermediary vertices are all in S
- The length of a shortest S-path from x to y is denoted by $L_x^S(y)$

proof of Dijkstra algorithm

- Let $y^\star \in \bar{S}$ such that $L_x^S(y^\star) = \min\{L_x^s(y) \mid y \in \bar{S}\}$
- Then, $L_x^S(y^\star) = L_x(y^\star)$

Loop invariant of Dijkstra algorithm (# 2)

- Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ-separating for x
- An S-path is a path whose intermediary vertices are all in S
- The length of a shortest S-path from x to y is denoted by $L_x^S(y)$

proof of Dijkstra algorithm

- Let $y^\star \in \bar{S}$ such that $L_x^S(y^\star) = \min\{L_x^s(y) \mid y \in \bar{S}\}$
- Then, $L_x^S(y^\star) = L_x(y^\star)$
- Thus, $S \cup \{y^\star\}$ is a set that is μ'-separating with $\mu' = L_x^S(y^\star)$

Computing the lengths of shortest paths

Algorithm DIJKSTRA (Data: A graph $G = (V, \Gamma)$, a network (G, ℓ), $n = |V|$, $x \in V$;

Result: L_x)

$\bar{S} := \emptyset$;

For each $y \in V$ **Do** $L_x[y] = \infty$; $\bar{S} := \bar{S} \cup \{y\}$;

$L_x[x] := 0$; $k := 0$; $\mu := 0$;

While $k < n$ and $\mu \neq \infty$ **Do**

- Extract a vertex $y^\star \in \bar{S}$ such that $L_x[y^\star] = \min\{L_x[y], y \in \bar{S}\}$
- $k++$; $\mu := L_x[y^\star]$;
- **For each** $y \in \Gamma(y^\star) \cap \bar{S}$ **Do**
  - $L_x[y] := \min\{L_x[y], L_x[y^\star] + \ell(y^\star, y)\}$;

Dijkstra's Algorithm

- Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u.

Dijkstra's Algorithm

- Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u.
- Greedily add a node v to S that is closest to s.

Dijkstra's Algorithm

- Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u.
- Greedily add a node v to S that is closest to s.

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

Let S be the set of explored nodes;
foreach $u \in S$ **do** store distance $d[u] = \infty$;
Initially $d[s] = 0$ and $S = s$;
while $S \neq V$ **do**
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
    Add v to S and define d[v] = d'[v];
end

Dijkstra's Algorithm

- Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u.
- Greedily add a node v to S that is closest to s.

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

Let S be the set of explored nodes;
foreach $u \in S$ **do** store distance $d[u] = \infty$;
Initially $d[s] = 0$ and $S = s$;
while $S \neq V$ **do**
  Select a node $v \notin S$ with at least one edge from S for which $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
  Add v to S and define d[v] = d'[v];
end

Dijkstra's Algorithm

- Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u.
- Greedily add a node v to S that is closest to s.

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
Let S be the set of explored nodes;
foreach u ∈ S do store distance d[u] = ∞;
Initially d[s] = 0 and S = s;
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d′(v) = min_{e=(u,v):u∈S} d[u] + W(e) is as small as possible;
    Add v to S and define d[v] = d'[v];
end
```

Dijkstra's Algorithm

- Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u.
- Greedily add a node v to S that is closest to s.

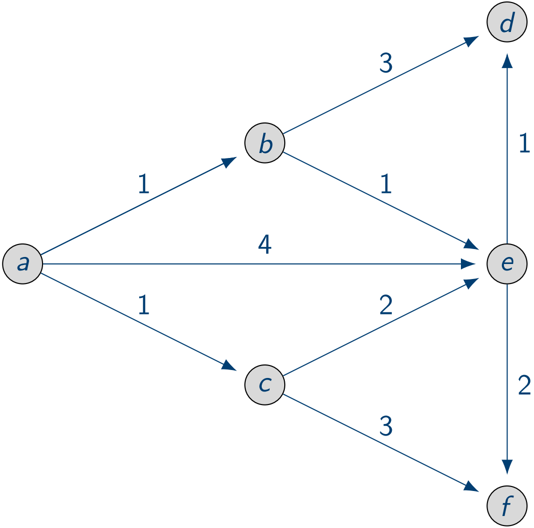
Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

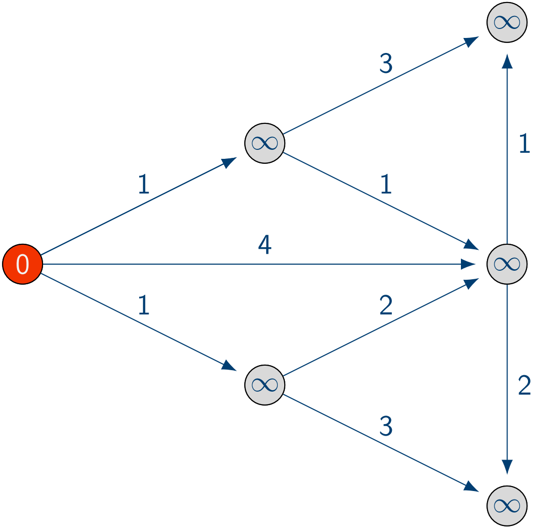
```
Let S be the set of explored nodes;
foreach u ∈ S do store distance d[u] = ∞;
Initially d[s] = 0 and S = s;
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d[u] + W(e) is as small as possible;
    Add v to S and define d[v] = d'[v];
end
```

- Can modify algorithm to compute the shortest paths themselves: record the predecessor u that minimizes $d'(v)$.

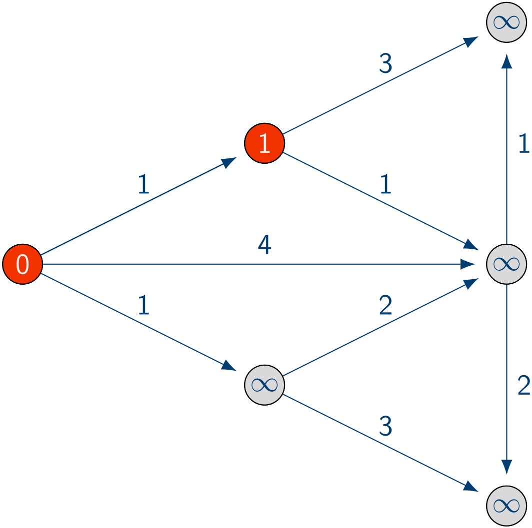
Example of Dijkstra's Algorithm



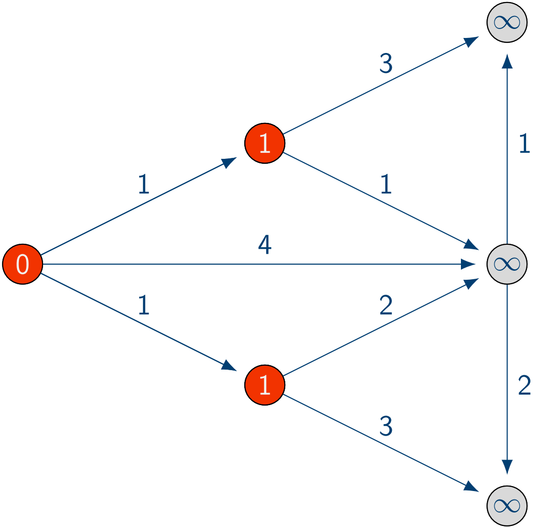
Example of Dijkstra's Algorithm



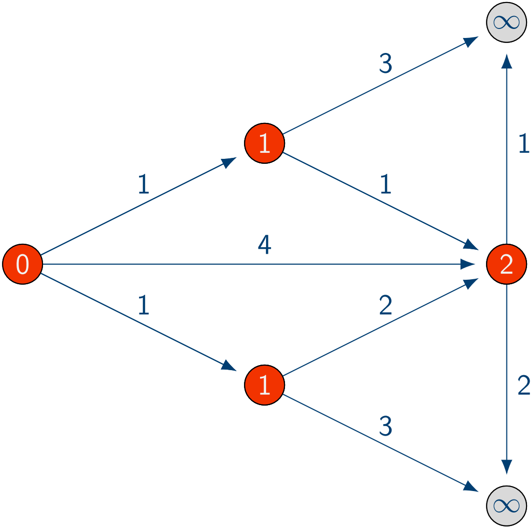
Example of Dijkstra's Algorithm



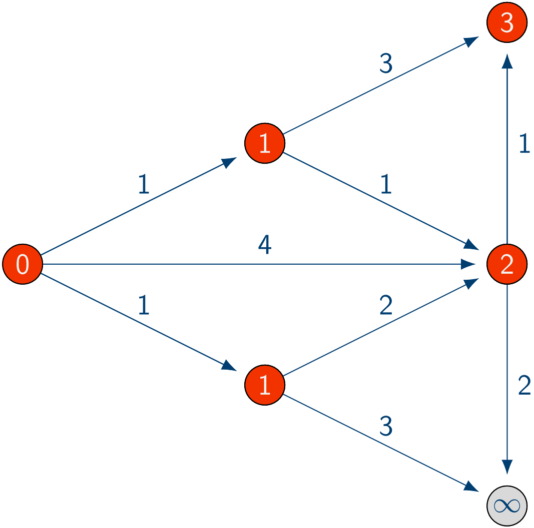
Example of Dijkstra's Algorithm



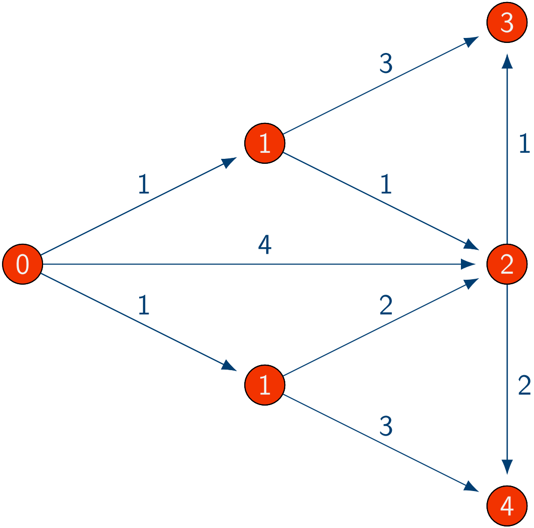
Example of Dijkstra's Algorithm



Example of Dijkstra's Algorithm



Example of Dijkstra's Algorithm

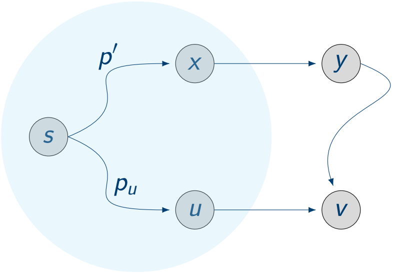


Proof of Correctness

- Let P_u be the shortest path computed for a node u.
- Claim: P_u is the shortest path from s to u.
- Prove by induction on the size of S.
  - Base case: $|S| = 1$. The only node in S is s.
  - Inductive step: we add the node v to S. Let u be the v's predecessor on the path P_v. Could there be a shorter path P from s to v?

Proof of Correctness

- Let P_u be the shortest path computed for a node u.
- Claim: P_u is the shortest path from s to u.
- Prove by induction on the size of S.
  - Base case: $|S| = 1$. The only node in S is s.
  - Inductive step: we add the node v to S. Let u be the v's predecessor on the path P_v. Could there be a shorter path P from s to v?



The alternate $s - v$ path P through x and y already too long by the time it had left the set S

Comments about Dijkstra's Algorithm

- Algorithm cannot handle negative edge lengths.
- Union of shortest paths output form a tree. Why?

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
Let S be the set of explored nodes;
foreach u ∈ S do store distance d[u] = ∞;
Initially d[s] = 0 and S = s;
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d[u] + W(e) is as small as possible;
    Add v to S and define d[v] = d'[v];
end
```

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

Let S be the set of explored nodes;
foreach $u \in S$ **do** store distance $d[u] = \infty$;
Initially $d[s] = 0$ and $S = s$;

while $S \neq V$ **do**
  Select a node $v \notin S$ with at least one edge from S for which
    $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
  Add v to S and define d[v] = d'[v];
end

- How many iterations are there of the while loop? .

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
Let S be the set of explored nodes;
foreach u ∈ S do store distance d[u] = ∞;
Initially d[s] = 0 and S = s;
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d[u] + W(e) is as small as possible;
    Add v to S and define d[v] = d'[v];
end
```

- How many iterations are there of the while loop? $n - 1$.

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
Let S be the set of explored nodes;
foreach u ∈ S do store distance d[u] = ∞;
Initially d[s] = 0 and S = s;
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d[u] + W(e) is as small as possible;
    Add v to S and define d[v] = d'[v];
end
```

- How many iterations are there of the while loop? $n - 1$.
- In each iteration, for each node $v \notin S$, compute
  $$\min_{e=(u,v),u\in S} d(u) + l_e.$$

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
Let S be the set of explored nodes;
foreach u ∈ S do store distance d[u] = ∞;
Initially d[s] = 0 and S = s;

while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d[u] + W(e) is as small as possible;
    Add v to S and define d[v] = d'[v];
end
```

- Observation: If we add v to S, $d'(w)$ changes only for v's neighbours.

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
Let S be the set of explored nodes;
foreach u ∈ S do store distance d[u] = ∞;
Initially d[s] = 0 and S = s;
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d[u] + W(e) is as small as possible;
    Add v to S and define d[v] = d'[v];
end
```

- Observation: If we add v to S, $d'(w)$ changes only for v's neighbours.
- Store the minima $d'(v)$ for each node $v \in V - S$ in a priority queue.
- Determine the next node v to add to S using ExtractMin.
- After adding v, for each neighbour w of v, compute $d(v) + l_{(v,w)}$.
- If $d(v) + l_{(v,w)} < d'(w)$, update w's key using ChangeKey.

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

Let S be the set of explored nodes;
foreach $u \in S$ **do** store distance $d[u] = \infty$;
Initially $d[s] = 0$ and $S = s$;
while $S \neq V$ **do**
  Select a node $v \notin S$ with at least one edge from S for which
    $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
  Add v to S and define d[v] = d'[v];
end

- Observation: If we add v to S, $d'(w)$ changes only for v's neighbours.
- Store the minima $d'(v)$ for each node $v \in V - S$ in a priority queue.
- Determine the next node v to add to S using ExtractMin.
- After adding v, for each neighbour w of v, compute $d(v) + l_{(v,w)}$.
- If $d(v) + l_{(v,w)} < d'(w)$, update w's key using ChangeKey.
- How many times are ExtractMin and ChangeKey invoked?

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

Let S be the set of explored nodes;
foreach $u \in S$ **do** store distance $d[u] = \infty$;
Initially $d[s] = 0$ and $S = s$;
while $S \neq V$ **do**
  Select a node $v \notin S$ with at least one edge from S for which $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
  Add v to S and define d[v] = d'[v];
end

- Observation: If we add v to S, $d'(w)$ changes only for v's neighbours.
- Store the minima $d'(v)$ for each node $v \in V - S$ in a priority queue.
- Determine the next node v to add to S using ExtractMin.
- After adding v, for each neighbour w of v, compute $d(v) + l_{(v,w)}$.
- If $d(v) + l_{(v,w)} < d'(w)$, update w's key using ChangeKey.
- How many times are ExtractMin and ChangeKey invoked? $n - 1$ and m times, respectively.

Single Source Shortest Path Problem

- $G = (V, E)$ is a connected directed graph. Each edge e has a length l_e. Note that the weights may be negative.
- V has n nodes and E has m edges.
- Length of a path P is the sum of lengths of the edges in P.
- Goal is to determine the shortest path from some start node s to all other nodes in V.
- Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

Single Source Shortest Path Problem

- $G = (V, E)$ is a connected directed graph. Each edge e has a length l_e. Note that the weights may be negative.
- V has n nodes and E has m edges.
- Length of a path P is the sum of lengths of the edges in P.
- Goal is to determine the shortest path from some start node s to all other nodes in V.
- Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

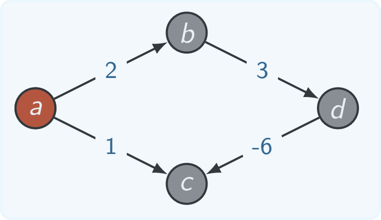
Shortest Paths

INSTANCE A directed graph $G(V, E)$, a function $l : E \to \mathbb{R}$, and a node $s \in V$

SOLUTION A set $\{P_u, u \in V\}$, where P_u is the shortest path in G from s to u.

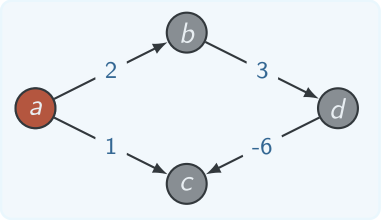
Bellman-Ford Algorithm

Dijkstra – Can fail if negative edge costs.

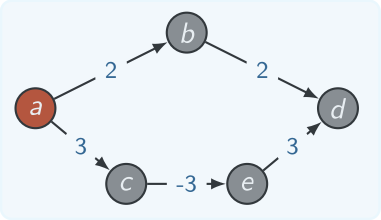


Bellman-Ford Algorithm

Dijkstra – Can fail if negative edge costs.

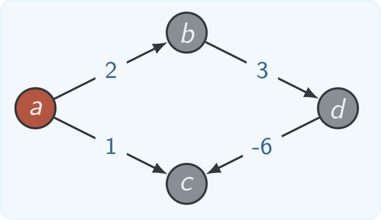


Re-weighting – Adding a constant to every edge weight can fail

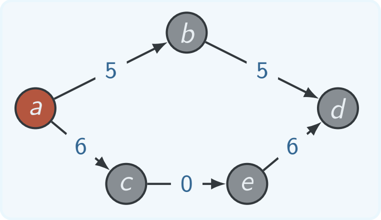


Bellman-Ford Algorithm

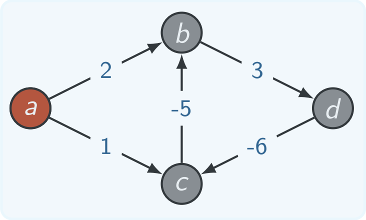
Dijkstra – Can **fail** if negative edge costs.



Re-weighting – Adding a **constant** to every edge weight can fail

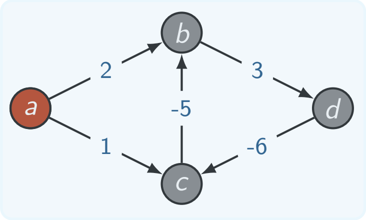


If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path; otherwise, there exists one that is simple.



Bellman-Ford Algorithm

If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path; otherwise, there exists one that is simple.



The Bellman-Ford algorithm is a way to find single source shortest paths in a graph with negative edge weights (but no negative cycles).

Bellman-Ford Algorithm

OPT(i, v) = length of shortest v-t path P using at most i edges.

Bellman-Forg Algorithm

OPT(i, v) = length of shortest v-t path P using at most i edges.

- Case 1: P uses at most $i-1$ edges.

$$OPT(i,v) = OPT(i-1,v)$$

Bellman-Ford Algorithm

OPT(i, v) = length of shortest v-t path P using at most i edges.

- Case 1: P uses at most $i-1$ edges.

$$OPT(i, v) = OPT(i-1, v)$$

- Case 2: P uses exactly i edges
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best w-t path using at most $i-1$ edges

Bellman-Ford Algorithm

OPT(i, v) = length of shortest v-t path P using at most i edges.

- Case 1: P uses at most $i-1$ edges.

$$OPT(i,v) = OPT(i-1,v)$$

- Case 2: P uses exactly i edges
  - if (v,w) is first edge, then OPT uses (v,w), and then selects best w-t path using at most $i-1$ edges

$$OPT(i,v) = \begin{cases} 0, & \textit{if } i = 0 \\ \min \begin{cases} OPT(i-1,v) \\ \min\{OPT(i-1,w) + c_{vw}\} \end{cases}, & \textit{otherwise} \end{cases}$$

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
foreach v ∈ V do d[0, v] = ∞;
Initially d[0, s] = 0;
for i = 1 to n − 1 do
    foreach v ∈ V do
        d[i, v] = d[i − 1, v]
    end
    foreach edge (w, v) ∈ E do
        d[i, v] = min{d[i, v], d[i − 1, w] + c_wv}
    end
end
```

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

```
foreach v ∈ V do d[0, v] = ∞;
Initially d[0, s] = 0;
for i = 1 to n − 1 do
    foreach v ∈ V do
        d[i, v] = d[i − 1, v]
    end
    foreach edge (w, v) ∈ E do
        d[i, v] = min{d[i, v], d[i − 1, w] + c_wv}
    end
end
```

- Computational cost: O(mn)

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

foreach $v \in V$ **do** $d[0, v] = \infty$;
Initially $d[0, s] = 0$;

for $i = 1$ *to* $n - 1$ **do**
    foreach $v \in V$ **do**
        $d[i, v] = d[i - 1, v]$
    end
    foreach *edge* $(w, v) \in E$ **do**
        $d[i, v] = \min\{d[i, v], d[i - 1, w] + c_{wv}\}$
    end
end

- Computational cost: O(mn)
- For finding the shortest paths, it is necessary to maintain a successor for each table entry.

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s.
output: The distances of the vertices from s

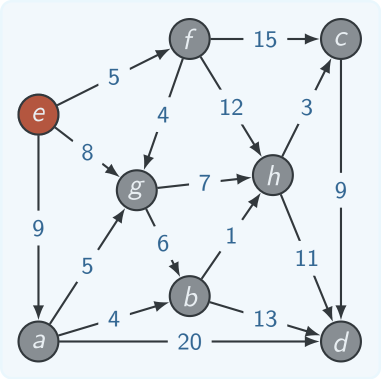
```
foreach v ∈ V do d[0, v] = ∞;
Initially d[0, s] = 0;

for i = 1 to n − 1 do
    foreach v ∈ V do
        d[i, v] = d[i − 1, v]
    end
    foreach edge (w, v) ∈ E do
        d[i, v] = min{d[i, v], d[i − 1, w] + c_wv}
    end
end
```

- Computational cost: O(mn)
- For finding the shortest paths, it is necessary to maintain a successor for each table entry.

How to detect negative cycles?

Shortest path – an example



Compute the shortest path from e to all other nodes!

Questions?

Shortest path
– Bellman-Ford –