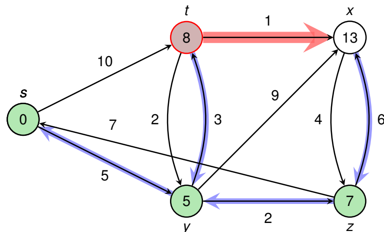


6.4: Single-Source Shortest Paths

Frank Stajano

Thomas Sauerwald

Lent 2015

UNIVERSITY OF CAMBRIDGE

Outline

Bellman-Ford

Dijkstra's Algorithm

All-Pairs Shortest Path

APSP via Matrix Multiplication

Johnson's Algorithm

The Bellman-Ford Algorithm

```
 BELLMAN-FORD(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V|-1 times
7:    for e in G.edges()
8:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
9:       if e.start.d + e.weight.d < e.end.d:
10:         e.end.d = e.start.d + e.weight
11:         e.end.predecessor = e.start
12:
13: for e in G.edges()
14:    if e.start.d + e.weight.d < e.end.d:
15:       return FALSE
16: return TRUE
```

The Bellman-Ford Algorithm

```
 BELLMAN-FORD(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V|-1 times
7:    for e in G.edges()
8:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
9:       if e.start.d + e.weight.d < e.end.d:
10:         e.end.d = e.start.d + e.weight
11:         e.end.predecessor = e.start
12:
13: for e in G.edges()
14:    if e.start.d + e.weight.d < e.end.d:
15:       return FALSE
16: return TRUE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

The Bellman-Ford Algorithm

```
 BELLMAN-FORD(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V|-1 times
7:    for e in G.edges()
8:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
9:       if e.start.d + e.weight.d < e.end.d:
10:         e.end.d = e.start.d + e.weight
11:         e.end.predecessor = e.start
12:
13: for e in G.edges()
14:    if e.start.d + e.weight.d < e.end.d:
15:       return FALSE
16: return TRUE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

Yes, because if pass i keeps all *.d* variables, then so does pass $i + 1$.

The Bellman-Ford Algorithm (modified)

```
 BELLMAN-FORD-NEW(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V| times
7:    flag = 0
8:    for e in G.edges()
9:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
10:        if e.start.d + e.weight.d < e.end.d:
11:           e.end.d = e.start.d + e.weight
12:           e.end.predecessor = e.start
13:           flag = 1
14:   if flag = 0 return TRUE
15:
16: return FALSE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

Yes, because if pass i keeps all *.d* variables, then so does pass $i + 1$.

The Bellman-Ford Algorithm (modified)

```
 BELLMAN-FORD-NEW(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V| times
7:    flag = 0
8:    for e in G.edges()
9:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
10:       if e.start.d + e.weight.d < e.end.d:
11:          e.end.d = e.start.d + e.weight
12:          e.end.predecessor = e.start
13:          flag = 1
14:   if flag = 0 return TRUE
15:
16: return FALSE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

Yes, because if pass i keeps all *.d* variables, then so does pass $i + 1$.

Outline

Bellman-Ford

Dijkstra's Algorithm

All-Pairs Shortest Path

APSP via Matrix Multiplication

Johnson's Algorithm

Dijkstra's Algorithm

Overview of Dijkstra

- Requires that all edges have non-negative weights
- Use a special order for relaxing edges

Dijkstra's Algorithm

Overview of Dijkstra

- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):

Dijkstra's Algorithm

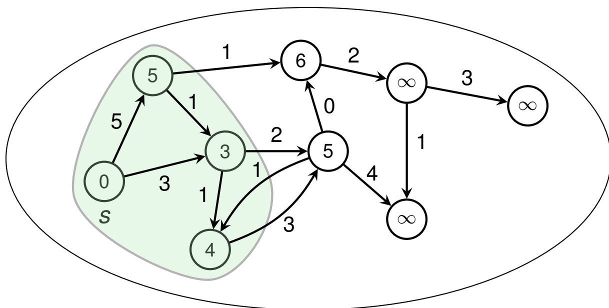
Overview of Dijkstra

- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$

Dijkstra's Algorithm

Overview of Dijkstra

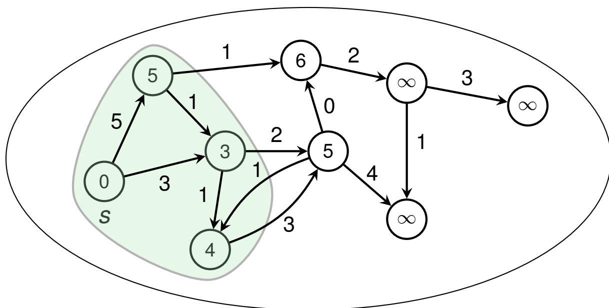
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$



Dijkstra's Algorithm

Overview of Dijkstra

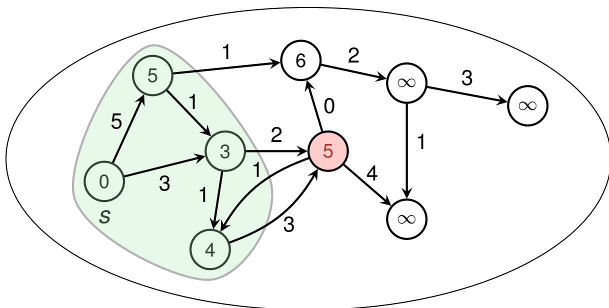
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$



Dijkstra's Algorithm

Overview of Dijkstra

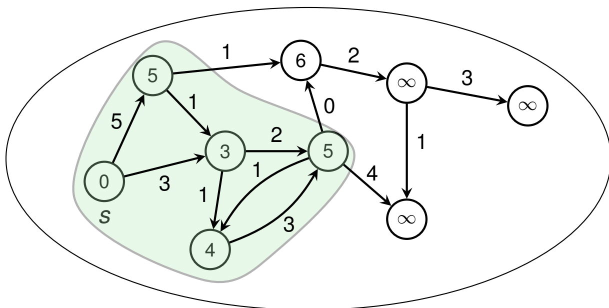
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$



Dijkstra's Algorithm

Overview of Dijkstra

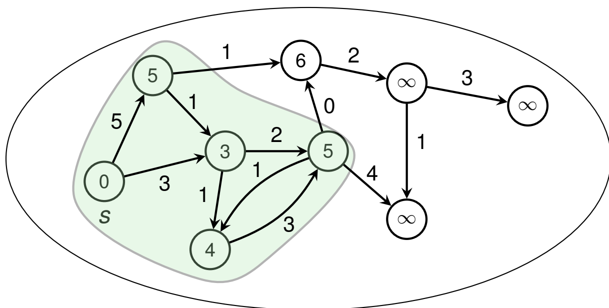
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$



Dijkstra's Algorithm

Overview of Dijkstra

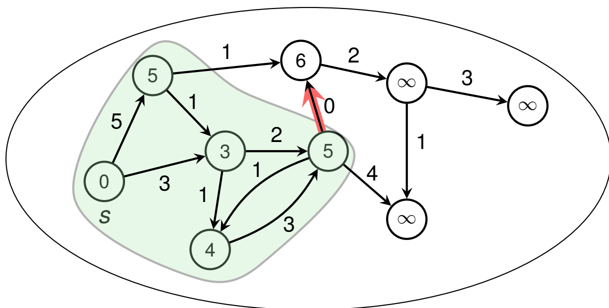
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

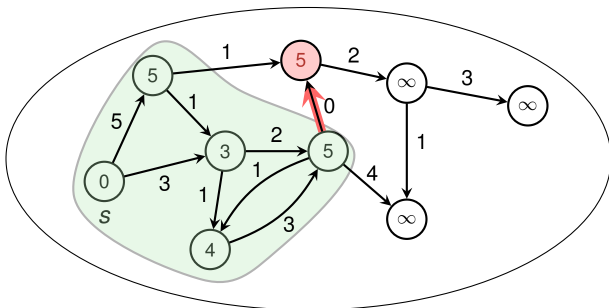
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

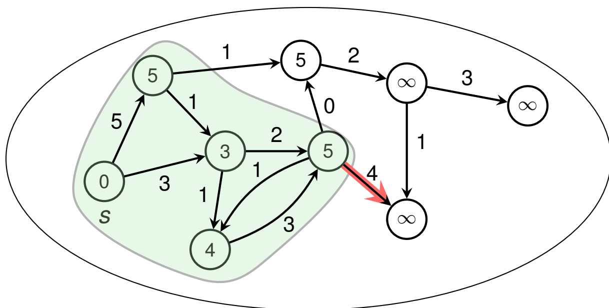
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

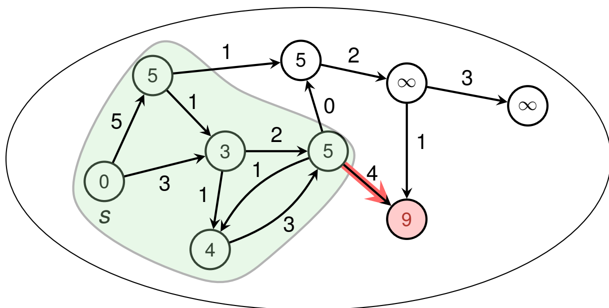
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

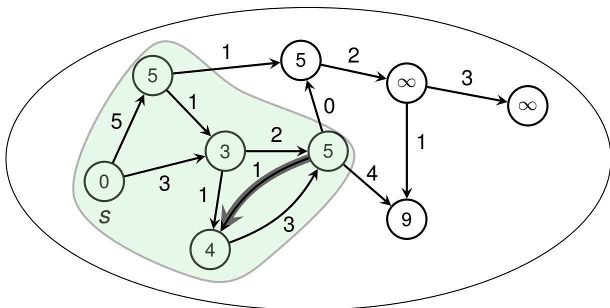
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

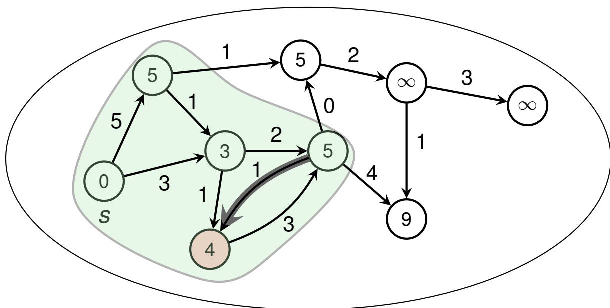
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

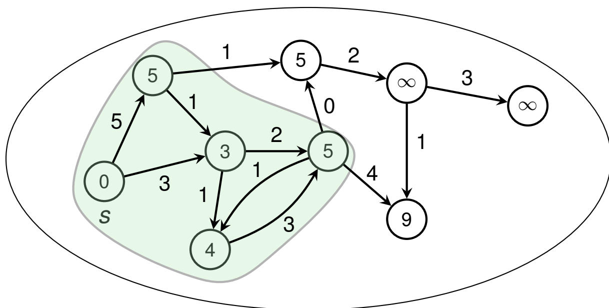
- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v



Dijkstra's Algorithm

Overview of Dijkstra

- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v

Dijkstra's Algorithm

Overview of Dijkstra

- Requires that all edges have non-negative weights
- Use a special order for relaxing edges
- The order follows a greedy-strategy (similar to Prim's algorithm):
  1. Maintain set S of vertices u with $u.\delta = u.d$
  2. At each step, add a vertex $v \in V \setminus S$ with minimal $v.d$ $(= v.\delta)$
  3. Relax all edges leaving v

DIJKSTRA(G,w,s)

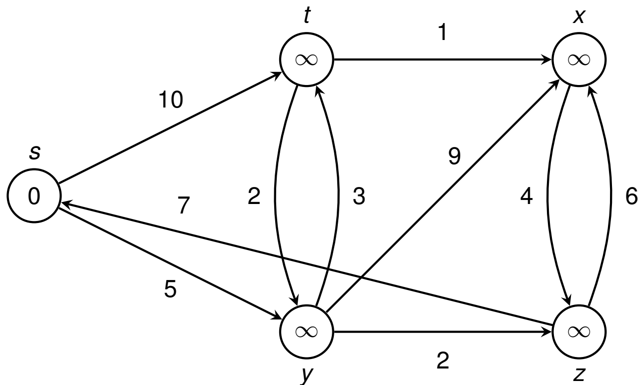
0: INITIALIZE(G,s)
1: $S = \emptyset$
2: $Q = V$
3: **while** $Q \neq \emptyset$ **do**
4:     u = Extract-Min(Q)
5:     $S = S \cup \{u\}$
6:     **for** each $v \in G.Adj[u]$ **do**
7:         RELAX(u, v, w)
8:     **end for**
9: **end while**

can actually skip edges which go to a vertex in S (cf. implementation of Dijksta in the handout)

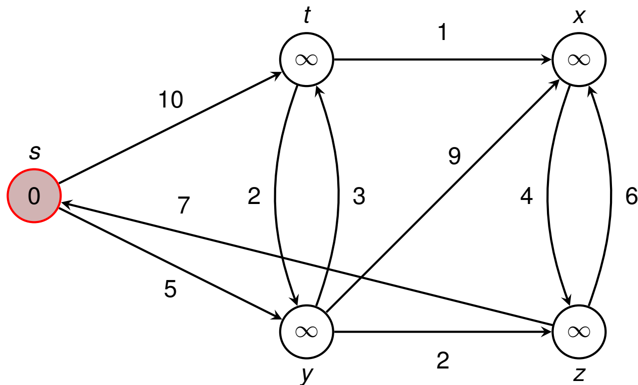
Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

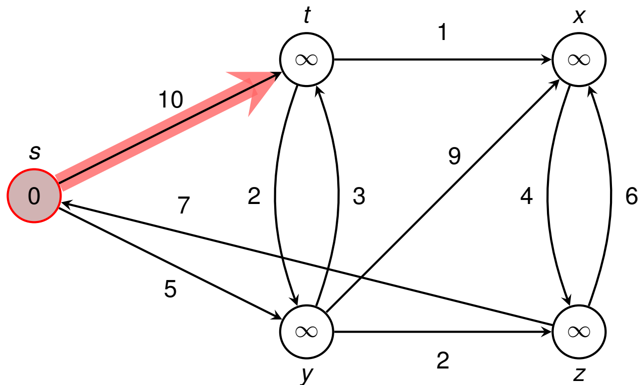
$(s, 0), (t, \infty), (x, \infty), (y, \infty), (z, \infty)$



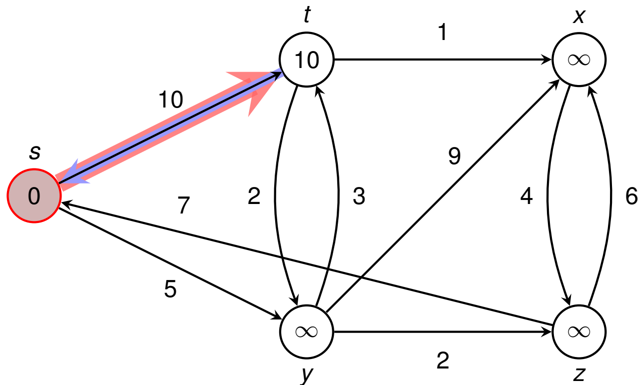
Complete Run of Dijkstra (Figure 24.6)



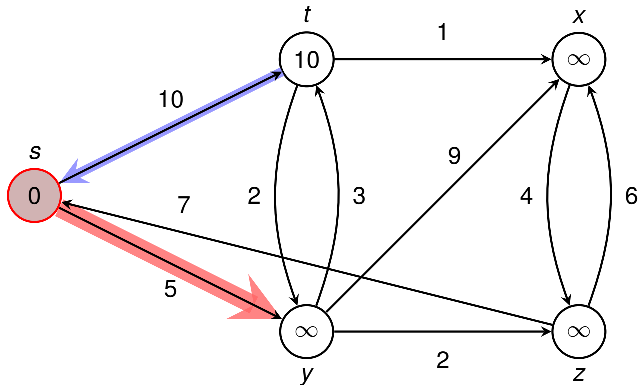
Complete Run of Dijkstra (Figure 24.6)



Complete Run of Dijkstra (Figure 24.6)



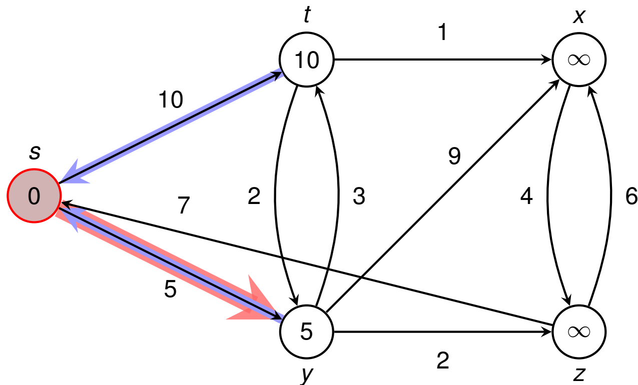
Complete Run of Dijkstra (Figure 24.6)



Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

~~$(s, 0)$~~, $(t, 10), (x, \infty), (y, 5), (z, \infty)$



Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

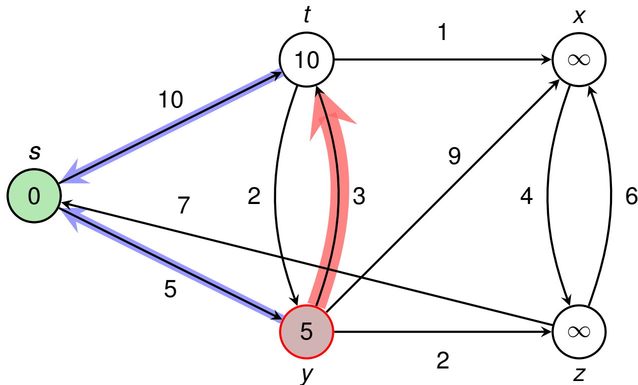
$\sout{(s, 0)}, (t, 10), (x, \infty), (y, 5), (z, \infty)$

Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

$(t, 10), (x, \infty), (y, 5), (z, \infty)$

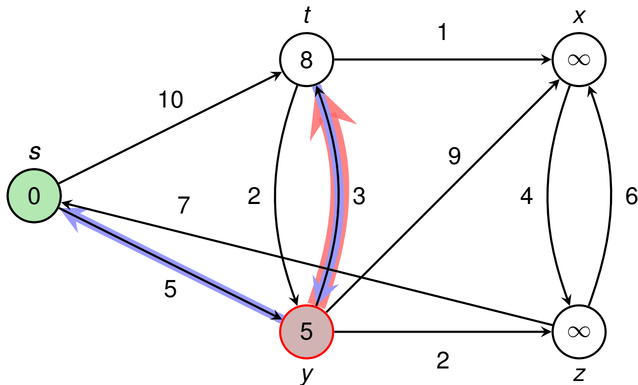
Complete Run of Dijkstra (Figure 24.6)



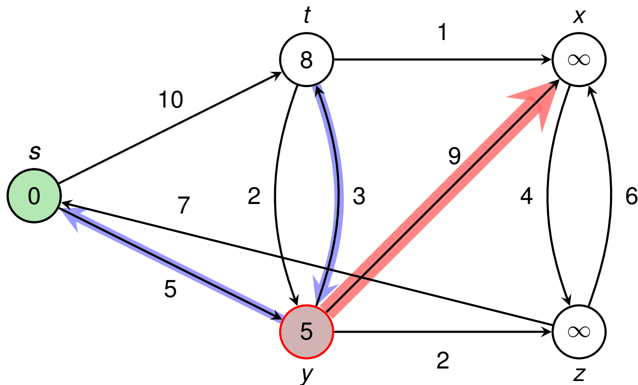
Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

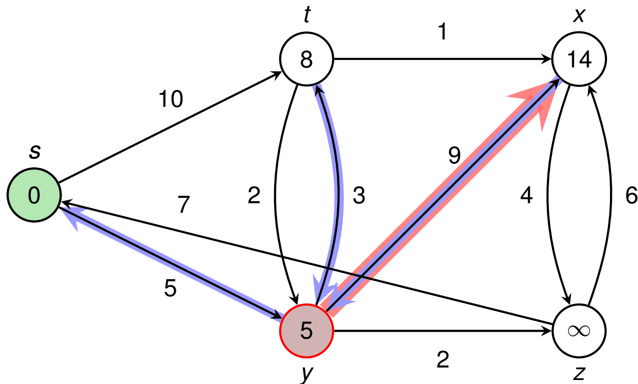
$(t, 8), (x, \infty), (y, 5), (z, \infty)$



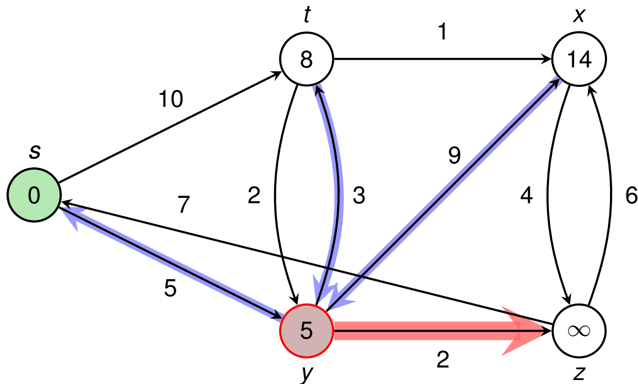
Complete Run of Dijkstra (Figure 24.6)



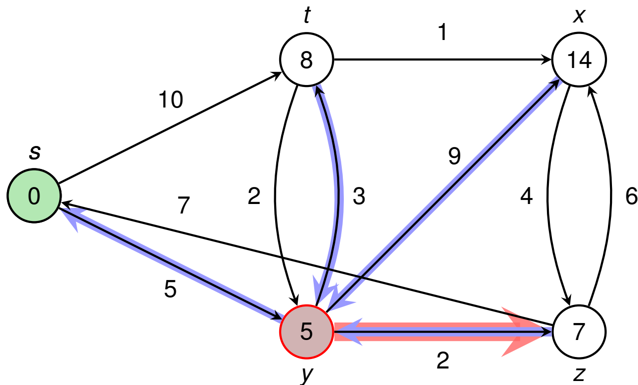
Complete Run of Dijkstra (Figure 24.6)



Complete Run of Dijkstra (Figure 24.6)



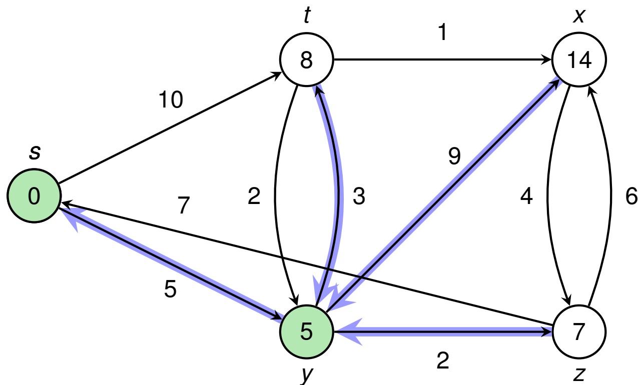
Complete Run of Dijkstra (Figure 24.6)



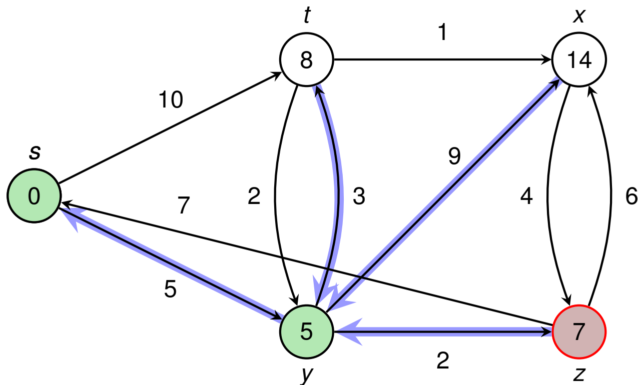
Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

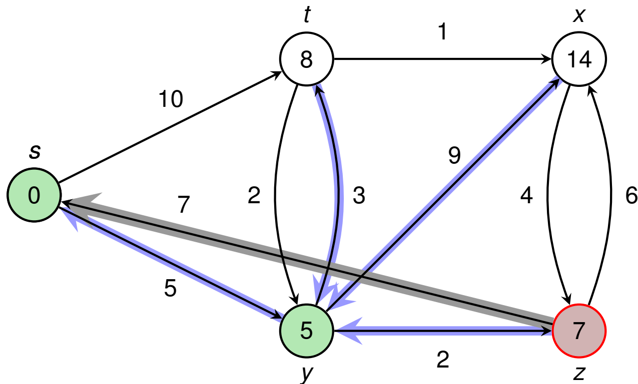
$(t, 8), (x, 14),$ ~~$(y, 5)$~~$, (z, 7)$



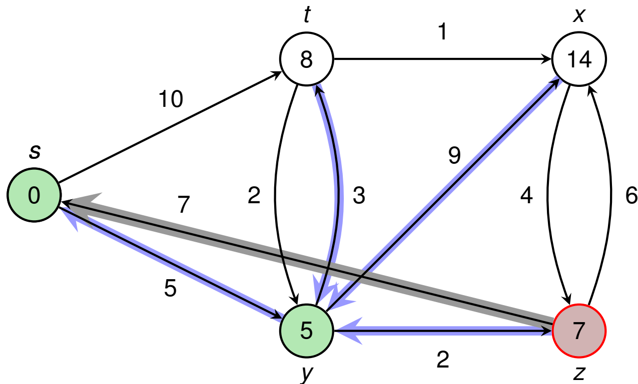
Complete Run of Dijkstra (Figure 24.6)



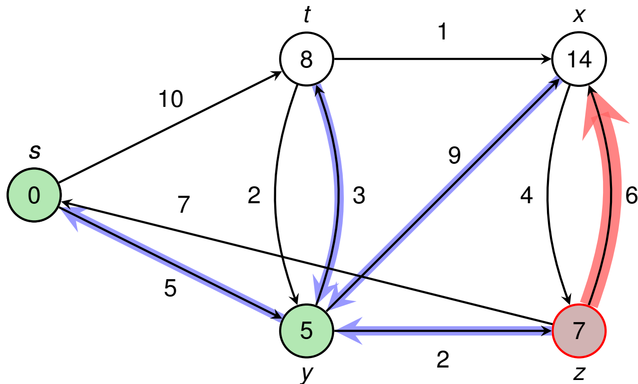
Complete Run of Dijkstra (Figure 24.6)



Complete Run of Dijkstra (Figure 24.6)



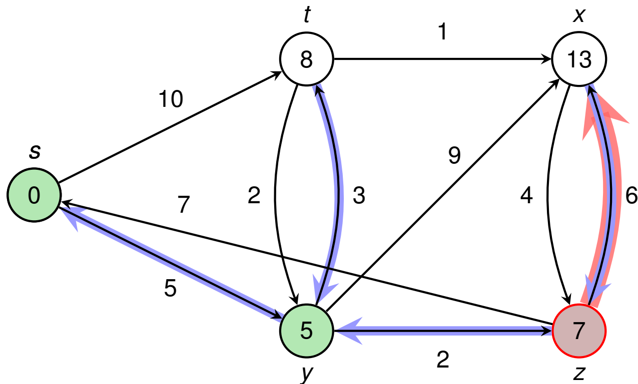
Complete Run of Dijkstra (Figure 24.6)



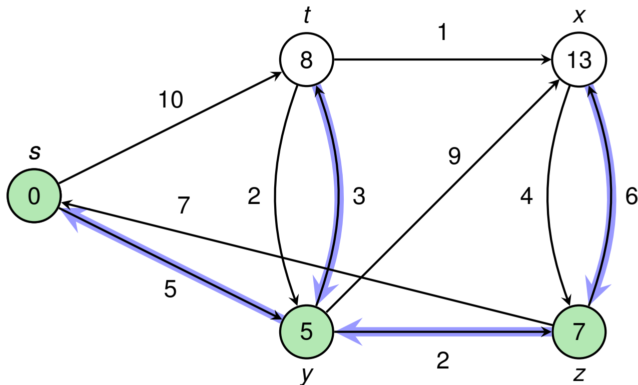
Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

$(t, 8), (x, 13), \cancel{(z, 7)}$



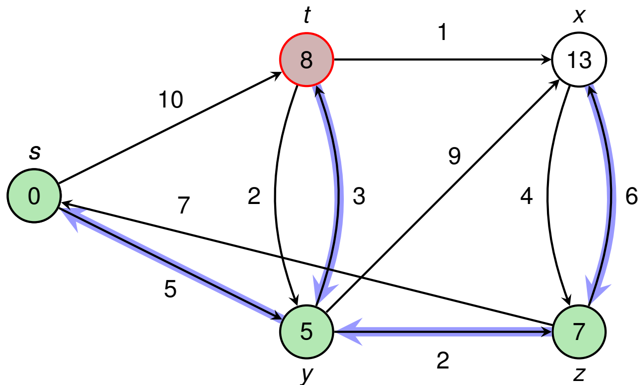
Complete Run of Dijkstra (Figure 24.6)



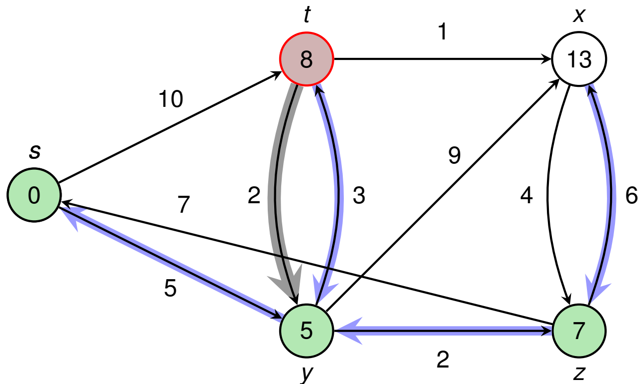
Complete Run of Dijkstra (Figure 24.6)

Priority Queue Q:

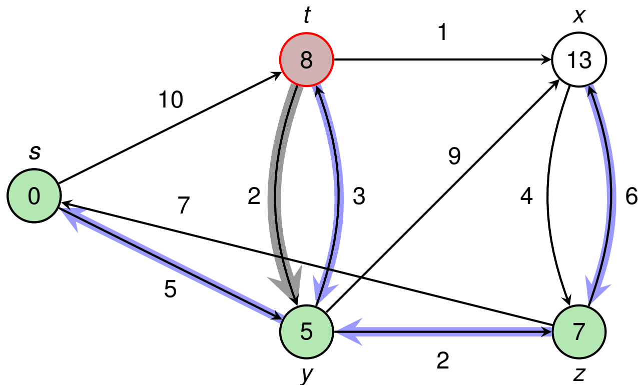
~~$(t, 8)$~~, $(x, 13)$



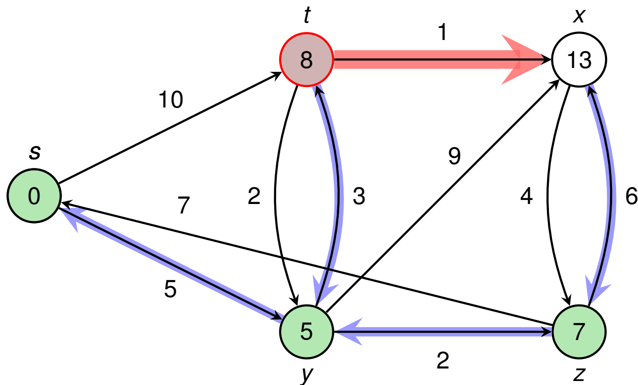
Complete Run of Dijkstra (Figure 24.6)



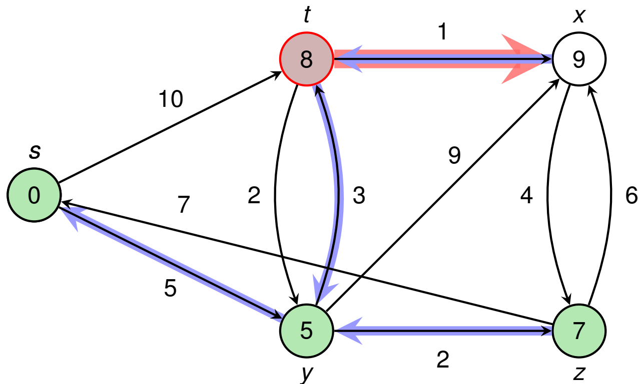
Complete Run of Dijkstra (Figure 24.6)



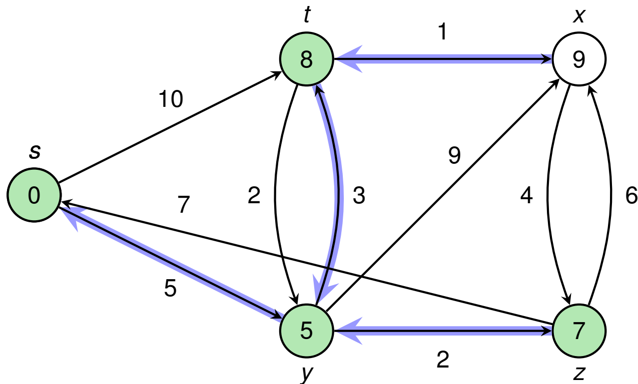
Complete Run of Dijkstra (Figure 24.6)



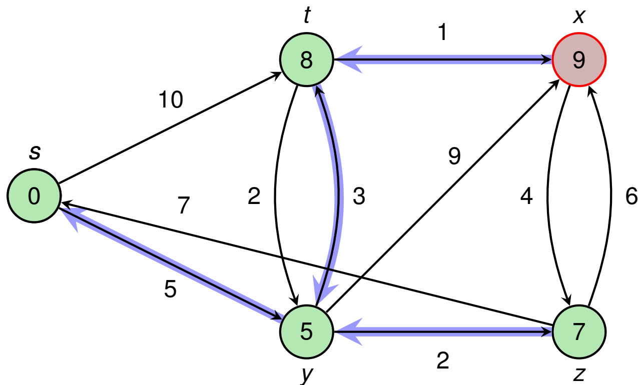
Complete Run of Dijkstra (Figure 24.6)



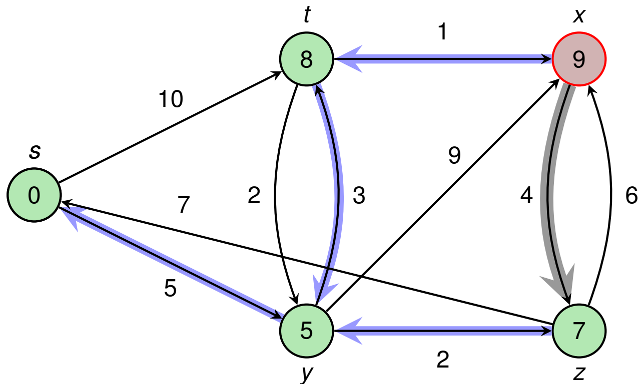
Complete Run of Dijkstra (Figure 24.6)



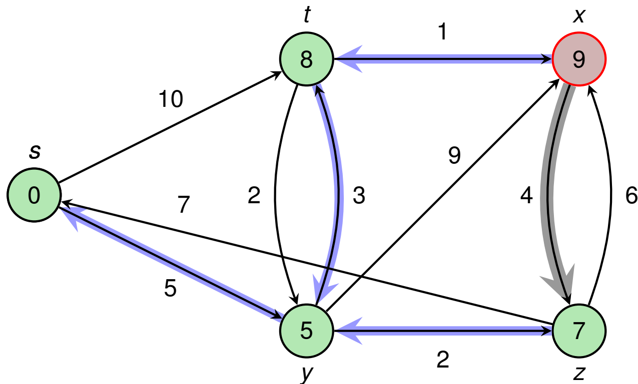
Complete Run of Dijkstra (Figure 24.6)



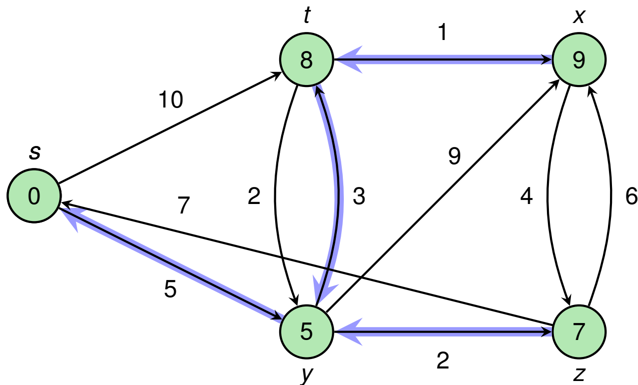
Complete Run of Dijkstra (Figure 24.6)



Complete Run of Dijkstra (Figure 24.6)



Complete Run of Dijkstra (Figure 24.6)



Runtime of Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
 0: INITIALIZE(G,s)
 1: S = ∅
 2: Q = V
 3: while Q ≠ ∅ do
 4:    u = Extract-Min(Q)
 5:    S = S ∪ {u}
 6:    for each v ∈ G.Adj[u] do
 7:       RELAX(u, v, w)
 8:    end for
 9: end while
```

Runtime of Dijkstra's Algorithm

DIJKSTRA(G,w,s)
0: INITIALIZE(G,s)
1: $S = \emptyset$
2: $Q = V$
3: **while** $Q \neq \emptyset$ **do**
4: $u =$ Extract-Min(Q)
5: $S = S \cup \{u\}$
6: **for** each $v \in G.Adj[u]$ **do**
7: RELAX(u, v, w)
8: **end for**
9: **end while**

Runtime (using Fibonacci Heaps)

Runtime of Dijkstra's Algorithm

DIJKSTRA(G,w,s)

0: INITIALIZE(G,s)
1: $S = \emptyset$
2: $Q = V$
3: **while** $Q \neq \emptyset$ **do**
4: $\quad u =$ Extract-Min(Q)
5: $\quad S = S \cup \{u\}$
6: $\quad$ **for** each $v \in G.Adj[u]$ **do**
7: $\quad\quad$ RELAX(u, v, w)
8: $\quad$ **end for**
9: **end while**

Runtime (using Fibonacci Heaps)

Runtime of Dijkstra's Algorithm

DIJKSTRA(G,w,s)

```
0: INITIALIZE(G,s)
1: S = ∅
2: Q = V
3: while Q ≠ ∅ do
4:    u = Extract-Min(Q)
5:    S = S ∪ {u}
6:    for each v ∈ G.Adj[u] do
7:       RELAX(u, v, w)
8:    end for
9: end while
```

Runtime (using Fibonacci Heaps)

- Initialization (l. 0-2): $\mathcal{O}(V)$

Runtime of Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
0: INITIALIZE(G,s)
1: S = ∅
2: Q = V
3: while Q ≠ ∅ do
4:    u = Extract-Min(Q)
5:    S = S ∪ {u}
6:    for each v ∈ G.Adj[u] do
7:       RELAX(u, v, w)
8:    end for
9: end while
```

Runtime (using Fibonacci Heaps)

- Initialization (l. 0-2): $\mathcal{O}(V)$

Runtime of Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
0: INITIALIZE(G,s)
1: S = ∅
2: Q = V
3: while Q ≠ ∅ do
4:     u = Extract-Min(Q)
5:     S = S ∪ {u}
6:     for each v ∈ G.Adj[u] do
7:         RELAX(u, v, w)
8:     end for
9: end while
```

Runtime (using Fibonacci Heaps)

- Initialization (l. 0-2): $\mathcal{O}(V)$
- ExtractMin (l. 4): $\mathcal{O}(V \cdot \log V)$

Runtime of Dijkstra's Algorithm

DIJKSTRA(G,w,s)

0: INITIALIZE(G,s)
1: $S = \emptyset$
2: $Q = V$
3: **while** $Q \neq \emptyset$ **do**
4: $u =$ Extract-Min(Q)
5: $S = S \cup \{u\}$
6: **for** each $v \in G.Adj[u]$ **do**
7: RELAX(u, v, w)
8: **end for**
9: **end while**

Runtime (using Fibonacci Heaps)

- Initialization (l. 0-2): $\mathcal{O}(V)$
- ExtractMin (l. 4): $\mathcal{O}(V \cdot \log V)$

Runtime of Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
0: INITIALIZE(G,s)
1: S = ∅
2: Q = V
3: while Q ≠ ∅ do
4:     u = Extract-Min(Q)
5:     S = S ∪ {u}
6:     for each v ∈ G.Adj[u] do
7:         RELAX(u, v, w)
8:     end for
9: end while
```

Runtime (using Fibonacci Heaps)

- Initialization (l. 0-2): $\mathcal{O}(V)$
- ExtractMin (l. 4): $\mathcal{O}(V \cdot \log V)$
- DecreaseKey (l. 7): $\mathcal{O}(E \cdot 1)$

Runtime of Dijkstra's Algorithm

DIJKSTRA(G,w,s)

0: INITIALIZE(G,s)
1: $S = \emptyset$
2: $Q = V$
3: **while** $Q \neq \emptyset$ **do**
4: $\quad u =$ Extract-Min(Q)
5: $\quad S = S \cup \{u\}$
6: $\quad$ **for** each $v \in G.Adj[u]$ **do**
7: $\qquad$ RELAX(u, v, w)
8: $\quad$ **end for**
9: **end while**

Runtime (using Fibonacci Heaps)

- Initialization (l. 0-2): $\mathcal{O}(V)$
- ExtractMin (l. 4): $\mathcal{O}(V \cdot \log V)$
- DecreaseKey (l. 7): $\mathcal{O}(E \cdot 1)$

Runtime of Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
0: INITIALIZE(G,s)
1: S = ∅
2: Q = V
3: while Q ≠ ∅ do
4:    u = Extract-Min(Q)
5:    S = S ∪ {u}
6:    for each v ∈ G.Adj[u] do
7:       RELAX(u, v, w)
8:    end for
9: end while
```

Runtime (using Fibonacci Heaps)

- Initialization (l. 0-2): $\mathcal{O}(V)$
- ExtractMin (l. 4): $\mathcal{O}(V \cdot \log V)$
- DecreaseKey (l. 7): $\mathcal{O}(E \cdot 1)$

$\Rightarrow$ Overall: $\mathcal{O}(V \log V + E)$

Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

Correctness of Dijkstra's Algorithm

Theorem 24.6

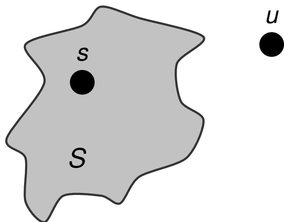
For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property



Correctness of Dijkstra's Algorithm

Theorem 24.6

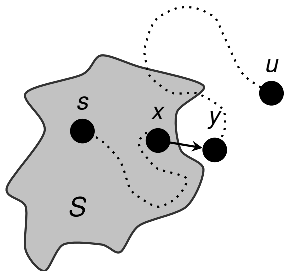
For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$



Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

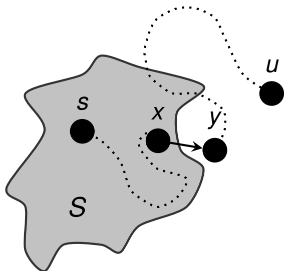
- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.d$$



Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

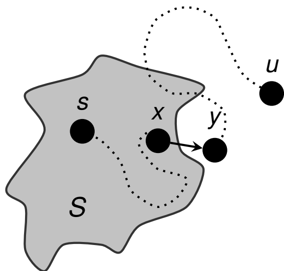
- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.d \leq y.d$$



Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

- Suppose there is $u \in V$, when extracted,

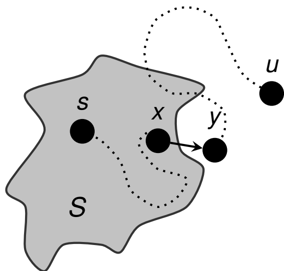
$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.d \leq y.d$$

u is extracted before y



Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

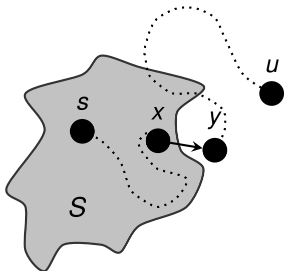
- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.d \leq y.d = y.\delta$$



Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

- Suppose there is $u \in V$, when extracted,

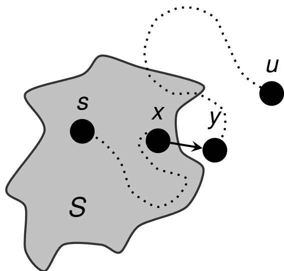
$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.d \leq y.d = y.\delta$$

since $x.d = x.\delta$ when x is extracted, and then (x, y) is relaxed $\Rightarrow$ Convergence Property



Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

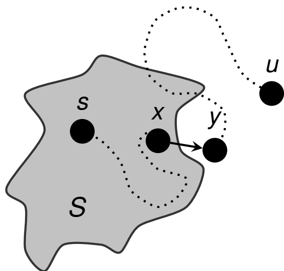
- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.\delta < u.d \leq y.d = y.\delta$$



Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

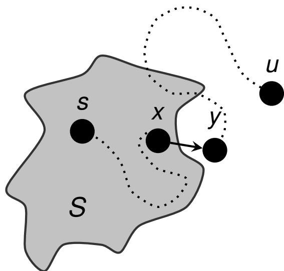
- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.\delta < u.d \leq y.d = y.\delta$$



This contradicts that y is on a shortest path from s to u. $\square$

Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

Proof: Invariant: If v is extracted, $v.d = v.\delta$

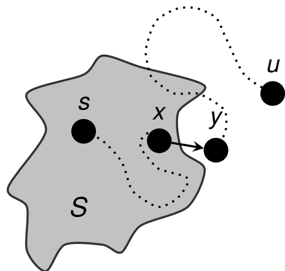
- Suppose there is $u \in V$, when extracted,

$$u.d > u.\delta$$

- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.\delta < u.d \leq y.d = y.\delta$$



This contradicts that y is on a shortest path from s to u.

This step requires non-negative weights!

Correctness of Dijkstra's Algorithm

Theorem 24.6

For any directed graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}^+$ and source s, Dijkstra terminates with $u.d = u.\delta$ for all $u \in V$.

There are edge cases like $s = x$ and/or $y = u$!

Proof: Invariant: If v is extracted, $v.d = v.\delta$

- Suppose there is $u \in V$, when extracted,

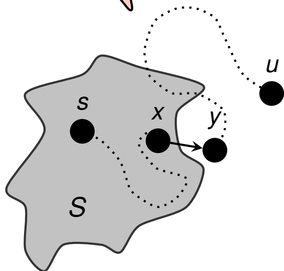
$$u.d > u.\delta$$

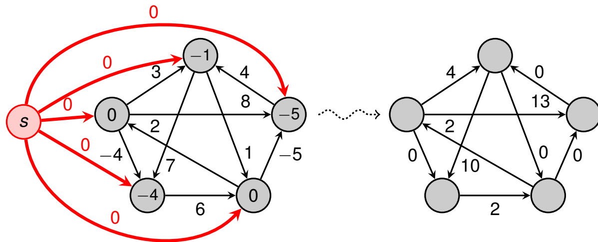
- Let u be the first vertex with this property
- Take a shortest path from s to u and let (x, y) be the first edge from S to $V \setminus S$

$\Rightarrow$

$$u.\delta < u.d \leq y.d = y.\delta$$

This contradicts that y is on a shortest path from s to u. $\square$





6.5: All-Pairs Shortest Paths

Frank Stajano

Thomas Sauerwald

Lent 2015

UNIVERSITY OF CAMBRIDGE

Outline

Bellman-Ford

Dijkstra's Algorithm

All-Pairs Shortest Path

APSP via Matrix Multiplication

Johnson's Algorithm

Formalizing the Problem

All-Pairs Shortest Path Problem

- Given: directed graph $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, with edge weights represented by a matrix W:

$$w_{i,j} = \begin{cases} \text{weight of edge } (i,j) & \text{for an edge } (i,j) \in E, \\ \infty & \text{if there is no edge from } i \text{ to } j, \\ 0 & \text{if } i = j. \end{cases}$$

Formalizing the Problem

All-Pairs Shortest Path Problem

- Given: directed graph $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, with edge weights represented by a matrix W:

$$w_{i,j} = \begin{cases} \text{weight of edge } (i,j) & \text{for an edge } (i,j) \in E, \\ \infty & \text{if there is no edge from } i \text{ to } j, \\ 0 & \text{if } i = j. \end{cases}$$

- Goal: Obtain a matrix of shortest path weights L, that is

$$\ell_{i,j} = \begin{cases} \text{weight of a shortest path from } i \text{ to } j, & \text{if } j \text{ is reachable from } i \\ \infty & \text{otherwise.} \end{cases}$$

Formalizing the Problem

All-Pairs Shortest Path Problem

- Given: directed graph $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, with edge weights represented by a matrix W:

$$w_{i,j} = \begin{cases} \text{weight of edge } (i,j) & \text{for an edge } (i,j) \in E, \\ \infty & \text{if there is no edge from } i \text{ to } j, \\ 0 & \text{if } i = j. \end{cases}$$

- Goal: Obtain a matrix of shortest path weights L, that is

$$\ell_{i,j} = \begin{cases} \text{weight of a shortest path from } i \text{ to } j, & \text{if } j \text{ is reachable from } i \\ \infty & \text{otherwise.} \end{cases}$$

Here we will only compute the weight of the shortest path without keeping track of the edges of the path!

Outline

Bellman-Ford

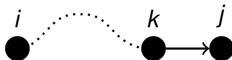
Dijkstra's Algorithm

All-Pairs Shortest Path

APSP via Matrix Multiplication

Johnson's Algorithm

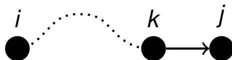
A Recursive Approach



Basic Idea

- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k - 1$ and an edge

A Recursive Approach

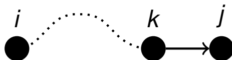


Basic Idea

- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges

A Recursive Approach

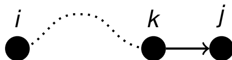


Basic Idea

- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges
- Then $\ell_{i,j}^{(1)} = w_{i,j}$, so $L^{(1)} = W$

A Recursive Approach

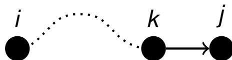


Basic Idea

- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges
- Then $\ell_{i,j}^{(1)} = w_{i,j}$, so $L^{(1)} = W$
- How can we obtain $L^{(2)}$ from $L^{(1)}$?

A Recursive Approach



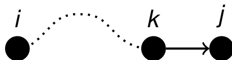
Basic Idea

- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges
- Then $\ell_{i,j}^{(1)} = w_{i,j}$, so $L^{(1)} = W$
- How can we obtain $L^{(2)}$ from $L^{(1)}$?

$$\ell_{i,j}^{(2)} = \min\left(\ell_{i,j}^{(1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(1)} + w_{k,j}\right)$$

A Recursive Approach



Basic Idea

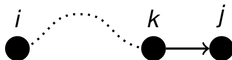
- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges
- Then $\ell_{i,j}^{(1)} = w_{i,j}$, so $L^{(1)} = W$
- How can we obtain $L^{(2)}$ from $L^{(1)}$?

$$\ell_{i,j}^{(2)} = \min\left(\ell_{i,j}^{(1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(1)} + w_{k,j}\right)$$

$$\ell_{i,j}^{(m)} =$$

A Recursive Approach



Basic Idea

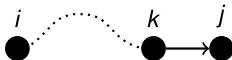
- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges
- Then $\ell_{i,j}^{(1)} = w_{i,j}$, so $L^{(1)} = W$
- How can we obtain $L^{(2)}$ from $L^{(1)}$?

$$\ell_{i,j}^{(2)} = \min\left(\ell_{i,j}^{(1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(1)} + w_{k,j}\right)$$

$$\ell_{i,j}^{(m)} = \min\left(\ell_{i,j}^{(m-1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(m-1)} + w_{k,j}\right)$$

A Recursive Approach



Basic Idea

- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

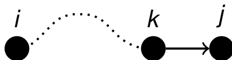
- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges
- Then $\ell_{i,j}^{(1)} = w_{i,j}$, so $L^{(1)} = W$
- How can we obtain $L^{(2)}$ from $L^{(1)}$?

$$\ell_{i,j}^{(2)} = \min\left(\ell_{i,j}^{(1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(1)} + w_{k,j}\right)$$

Recall that $w_{j,j} = 0$!

$$\ell_{i,j}^{(m)} = \min\left(\ell_{i,j}^{(m-1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(m-1)} + w_{k,j}\right)$$

A Recursive Approach



Basic Idea

- Any shortest path from i to j of length $k \geq 2$ is the concatenation of a shortest path of length $k-1$ and an edge

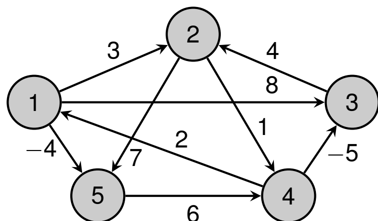
- Let $\ell_{i,j}^{(m)}$ be min. weight of any path from i to j with at most m edges
- Then $\ell_{i,j}^{(1)} = w_{i,j}$, so $L^{(1)} = W$
- How can we obtain $L^{(2)}$ from $L^{(1)}$?

$$\ell_{i,j}^{(2)} = \min\left(\ell_{i,j}^{(1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(1)} + w_{k,j}\right)$$

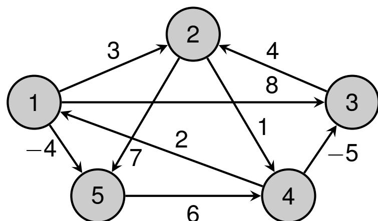
Recall that $w_{j,j} = 0$!

$$\ell_{i,j}^{(m)} = \min\left(\ell_{i,j}^{(m-1)}, \min_{1 \leq k \leq n} \ell_{i,k}^{(m-1)} + w_{k,j}\right) = \min_{1 \leq k \leq n}\left(\ell_{i,k}^{(m-1)} + w_{k,j}\right)$$

Example of Shortest Path via Matrix Multiplication (Figure 25.1)

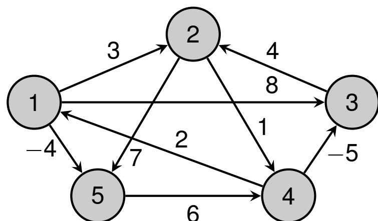


Example of Shortest Path via Matrix Multiplication (Figure 25.1)



$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

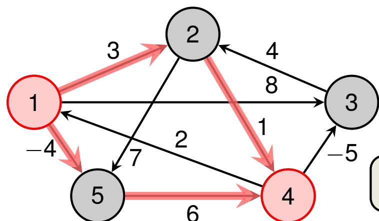
Example of Shortest Path via Matrix Multiplication (Figure 25.1)



$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & ? & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

Example of Shortest Path via Matrix Multiplication (Figure 25.1)

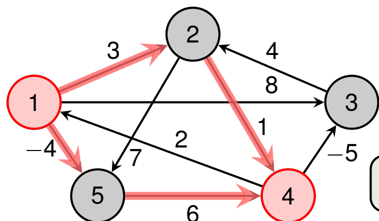


$$\ell_{1,4}^{(2)} = \min\{0+\infty, 3+1, 8+\infty, \infty+0, -4+6\}$$

$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & ? & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

Example of Shortest Path via Matrix Multiplication (Figure 25.1)

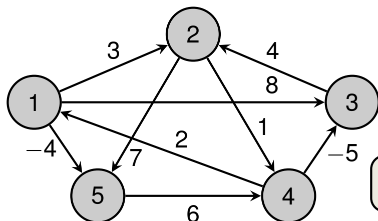


$$\ell_{1,4}^{(2)} = \min\{0+\infty, 3+1, 8+\infty, \infty+0, -4+6\}$$

$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

Example of Shortest Path via Matrix Multiplication (Figure 25.1)



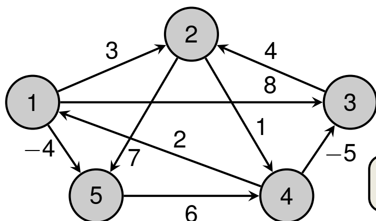
$$\ell^{(2)}_{1,4} = \min\{0+\infty, 3+1, 8+\infty, \infty+0, -4+6\}$$

$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Example of Shortest Path via Matrix Multiplication (Figure 25.1)



$$\ell_{1,4}^{(2)} = \min\{0+\infty, 3+1, 8+\infty, \infty+0, -4+6\}$$

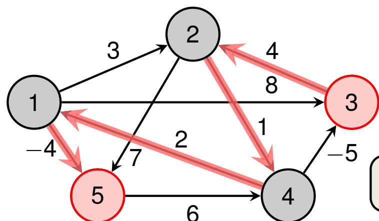
$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & ? \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Example of Shortest Path via Matrix Multiplication (Figure 25.1)



$$\ell_{1,4}^{(2)} = \min\{0+\infty, 3+1, 8+\infty, \infty+0, -4+6\}$$

$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

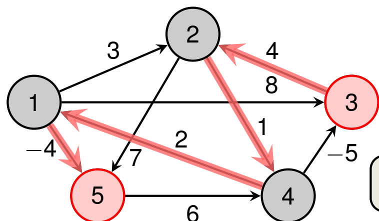
$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & ? \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\ell_{3,5}^{(4)} = \min\{7-4, 4+7, 0+\infty, 5+\infty, 11+0\}$$

Example of Shortest Path via Matrix Multiplication (Figure 25.1)



$$\ell^{(2)}_{1,4} = \min\{0+\infty, 3+1, 8+\infty, \infty+0, -4+6\}$$

$$L^{(1)} = W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\ell^{(4)}_{3,5} = \min\{7-4, 4+7, 0+\infty, 5+\infty, 11+0\}$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n - 1 = |V| - 1$ edges (assuming absence of negative-weight cycles)

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n-1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n-1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n-1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

$$(L^{(m-1)} \cdot W)_{i,j} = \sum_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} \times w_{k,j} \right)$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n - 1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

$$(L^{(m-1)} \cdot W)_{i,j} = \sum_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} \times w_{k,j} \right)$$

- The correspondence is as follows:

$$\min \Leftrightarrow \sum$$

$$+ \Leftrightarrow \times$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n-1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

$$(L^{(m-1)} \cdot W)_{i,j} = \sum_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} \times w_{k,j} \right)$$

- The correspondence is as follows:

$$\min \Leftrightarrow \sum$$

$$+ \Leftrightarrow \times$$

$$\infty \Leftrightarrow ?$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n - 1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

$$(L^{(m-1)} \cdot W)_{i,j} = \sum_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} \times w_{k,j} \right)$$

- The correspondence is as follows:

$$\min \Leftrightarrow \sum$$

$$+ \Leftrightarrow \times$$

$$\infty \Leftrightarrow 0$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n - 1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

$$(L^{(m-1)} \cdot W)_{i,j} = \sum_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} \times w_{k,j} \right)$$

- The correspondence is as follows:

$$\begin{aligned} \min &\Leftrightarrow \sum \\ + &\Leftrightarrow \times \\ \infty &\Leftrightarrow 0 \\ 0 &\Leftrightarrow ? \end{aligned}$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n-1 = |V| - 1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

$$(L^{(m-1)} \cdot W)_{i,j} = \sum_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} \times w_{k,j} \right)$$

- The correspondence is as follows:

$$\begin{aligned} \min &\Leftrightarrow \sum \\ + &\Leftrightarrow \times \\ \infty &\Leftrightarrow 0 \\ 0 &\Leftrightarrow 1 \end{aligned}$$

Computing $L^{(m)}$

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \ldots = L$, since every shortest path uses at most $n-1 = |V|-1$ edges (assuming absence of negative-weight cycles)
- Computing $L^{(m)}$:

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

$$(L^{(m-1)} \cdot W)_{i,j} = \sum_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} \times w_{k,j} \right)$$

$L^{(m)}$ can be computed in $\mathcal{O}(n^3)$

- The correspondence is as follows:

$$\min \Leftrightarrow \sum$$

$$+ \Leftrightarrow \times$$

$$\infty \Leftrightarrow 0$$

$$0 \Leftrightarrow 1$$

Computing $L^{(n-1)}$ efficiently

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

- For, say, $n = 738$, we subsequently compute

$$L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}, \ldots, L^{(737)} = L$$

Computing $L^{(n-1)}$ efficiently

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

Takes $\mathcal{O}(n \cdot n^3) = \mathcal{O}(n^4)$

- For, say, $n = 738$, we subsequently compute

$$L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}, \ldots, L^{(737)} = L$$

Computing $L^{(n-1)}$ efficiently

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

Takes $\mathcal{O}(n \cdot n^3) = \mathcal{O}(n^4)$

- For, say, $n = 738$, we subsequently compute

$$L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}, \ldots, L^{(737)} = L$$

- Since we don't need the intermediate matrices, a more efficient way is

$$L^{(1)}, L^{(2)}, L^{(4)}, \ldots, L^{(512)}, L^{(1024)} = L$$

Computing $L^{(n-1)}$ efficiently

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

Takes $\mathcal{O}(n \cdot n^3) = \mathcal{O}(n^4)$

- For, say, $n = 738$, we subsequently compute

$$L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}, \ldots, L^{(737)} = L$$

- Since we don't need the intermediate matrices, a more efficient way is

$$L^{(1)}, L^{(2)}, L^{(4)}, \ldots, L^{(512)}, L^{(1024)} = L$$

Takes $\mathcal{O}(\log n \cdot n^3)$.

Computing $L^{(n-1)}$ efficiently

$$\ell_{i,j}^{(m)} = \min_{1 \leq k \leq n} \left(\ell_{i,k}^{(m-1)} + w_{k,j} \right)$$

Takes $\mathcal{O}(n \cdot n^3) = \mathcal{O}(n^4)$

- For, say, $n = 738$, we subsequently compute

$$L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}, \ldots, L^{(737)} = L$$

- Since we don't need the intermediate matrices, a more efficient way is

$$L^{(1)}, L^{(2)}, L^{(4)}, \ldots, L^{(512)}, L^{(1024)} = L$$

We need $L^{(4)} = L^{(2)} \cdot L^{(2)} = L^{(3)} \cdot L^{(1)}$! (see Ex. 25.1-4)

Takes $\mathcal{O}(\log n \cdot n^3)$.

Outline

Bellman-Ford

Dijkstra's Algorithm

All-Pairs Shortest Path

APSP via Matrix Multiplication

Johnson's Algorithm

Johnson's Algorithm

Overview

Johnson's Algorithm

Overview

- allow negative-weight edges and negative-weight cycles

Johnson's Algorithm

Overview

- allow negative-weight edges and negative-weight cycles
- one pass of Bellman-Ford and $|V|$ passes of Dijkstra

Johnson's Algorithm

Overview

- allow negative-weight edges and negative-weight cycles
- one pass of Bellman-Ford and $|V|$ passes of Dijkstra
- after Bellman-Ford, edges are reweighted s.t.

Johnson's Algorithm

Overview

- allow negative-weight edges and negative-weight cycles
- one pass of Bellman-Ford and $|V|$ passes of Dijkstra
- after Bellman-Ford, edges are reweighted s.t.
  - all edge weights are non-negative

Johnson's Algorithm

Overview

- allow negative-weight edges and negative-weight cycles
- one pass of Bellman-Ford and $|V|$ passes of Dijkstra
- after Bellman-Ford, edges are reweighted s.t.
  - all edge weights are non-negative
  - shortest paths are maintained

Johnson's Algorithm

Overview

- allow negative-weight edges and negative-weight cycles
- one pass of Bellman-Ford and $|V|$ passes of Dijkstra
- after Bellman-Ford, edges are reweighted s.t.
  - all edge weights are non-negative
  - shortest paths are maintained

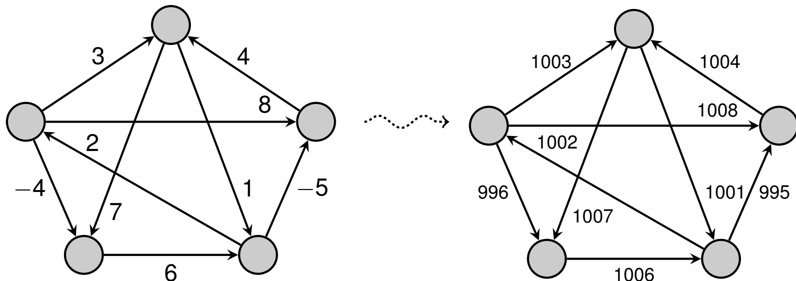
Adding a constant to every edge doesn't work!

Johnson's Algorithm

Overview

- allow negative-weight edges and negative-weight cycles
- one pass of Bellman-Ford and $|V|$ passes of Dijkstra
- after Bellman-Ford, edges are reweighted s.t.
  - all edge weights are non-negative
  - shortest paths are maintained

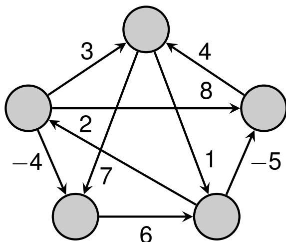
Adding a constant to every edge doesn't work!



How Johnson's Algorithm works

Johnson's Algorithm

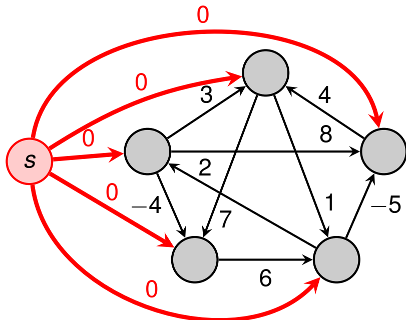
1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0



How Johnson's Algorithm works

Johnson's Algorithm

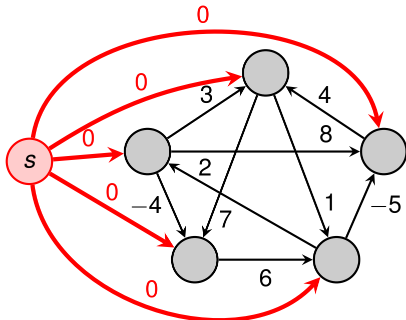
1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0



How Johnson's Algorithm works

Johnson's Algorithm

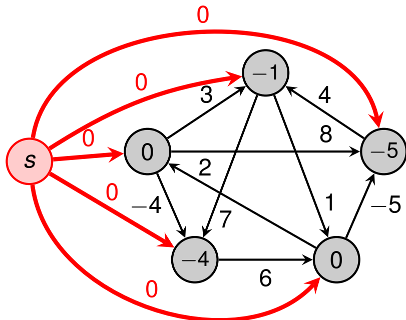
1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s



How Johnson's Algorithm works

Johnson's Algorithm

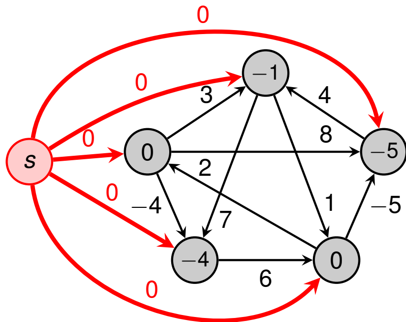
1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s



How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort



How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

Direct: 7, Detour: -1

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

Direct: 7, Detour: -1

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

Direct: 7, Detour: −1

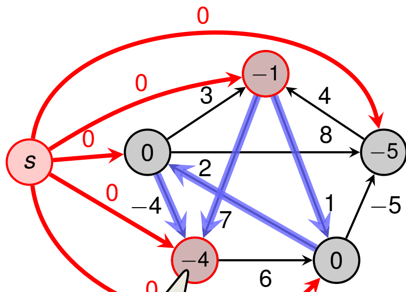
Direct: 10, Detour: 2

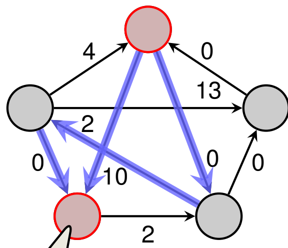
How Johnson's Algorithm works

Johnson's Algorithm

1. Add a new vertex s and directed edges (s, v), $v \in V$, with weight 0
2. Run Bellman-Ford on this augmented graph with source s
   - If there are negative weight cycles, abort
   - Otherwise:
     1) Reweight every edge (u, v) by $\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$
     2) Remove vertex s and its incident edges
3. For every vertex $v \in V$, run Dijkstra on $(G, E, \widetilde{w})$

Runtime: $O(V \cdot E + V \cdot (V \log V + E))$





Correctness of Johnson's Algorithm

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Correctness of Johnson's Algorithm

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 1.

Correctness of Johnson's Algorithm

$$\widetilde{w}(u,v) = w(u,v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 1. Let $u.\delta$ and $v.\delta$ be the distances from the fake source s

Correctness of Johnson's Algorithm

$$\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 1. Let $u.\delta$ and $v.\delta$ be the distances from the fake source s

$$u.\delta + w(u, v) \geq v.\delta \qquad \text{(triangle inequality)}$$

Correctness of Johnson's Algorithm

$$\widetilde{w}(u,v) = w(u,v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 1. Let $u.\delta$ and $v.\delta$ be the distances from the fake source s

$$u.\delta + w(u,v) \geq v.\delta \qquad \text{(triangle inequality)}$$

$$\Rightarrow \qquad \widetilde{w}(u,v) + u.\delta + w(u,v) \geq w(u,v) + u.\delta - v.\delta + v.\delta$$

Correctness of Johnson's Algorithm

$$\widetilde{w}(u,v) = w(u,v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 1. Let $u.\delta$ and $v.\delta$ be the distances from the fake source s

$$u.\delta + w(u,v) \geq v.\delta \qquad \text{(triangle inequality)}$$

$$\Rightarrow \quad \widetilde{w}(u,v) + u.\delta + w(u,v) \geq w(u,v) + u.\delta - v.\delta + v.\delta$$

$$\Rightarrow \quad \widetilde{w}(u,v) \geq 0$$

□

Correctness of Johnson's Algorithm

$$\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 2.

Correctness of Johnson's Algorithm

$$\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 2.

Let $p = (v_0, v_1, \ldots, v_k)$ be any path

Correctness of Johnson's Algorithm

$$\widetilde{w}(u,v) = w(u,v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 2.

Let $p = (v_0, v_1, \ldots, v_k)$ be any path

- In the original graph, the weight is $\sum_{i=1}^{k} w(v_{i-1}, v_i) = w(p)$.

Correctness of Johnson's Algorithm

$$\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 2.

Let $p = (v_0, v_1, \ldots, v_k)$ be any path

- In the original graph, the weight is $\sum_{i=1}^{k} w(v_{i-1}, v_i) = w(p)$.
- In the reweighted graph, the weight is

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

Correctness of Johnson's Algorithm

$$\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 2.

Let $p = (v_0, v_1, \ldots, v_k)$ be any path

- In the original graph, the weight is $\sum_{i=1}^{k} w(v_{i-1}, v_i) = w(p)$.
- In the reweighted graph, the weight is

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

Correctness of Johnson's Algorithm

$$\widetilde{w}(u, v) = w(u, v) + u.\delta - v.\delta$$

Theorem

For any graph $G = (V, E, w)$ without negative-weight cycles:

1. After reweighting, all edges are non-negative
2. Shortest Paths are preserved

Proof of 2.

Let $p = (v_0, v_1, \ldots, v_k)$ be any path

- In the original graph, the weight is $\sum_{i=1}^{k} w(v_{i-1}, v_i) = w(p)$.
- In the reweighted graph, the weight is

$$\sum_{i=1}^{k} \widetilde{w}(v_{i-1}, v_i) = \sum_{i=1}^{k} \left(w(v_{i-1}, v_i) + v_{i-1}.\delta - v_i.\delta\right) = w(p) + v_0.\delta - v_k.\delta \quad \square$$

Comparison of all Shortest-Path Algorithms

Algorithm	SSSP		APSP		negative weights
	sparse	dense	sparse	dense	
Bellman-Ford	V^2	V^3	V^3	V^4	✓
Dijkstra	$V \log V$	V^2	$V^2 \log V$	V^3	X
Matrix Mult.	–	–	$V^3 \log V$	$V^3 \log V$	(✓)
Johnson	–	–	$V^2 \log V$	V^3	✓

Comparison of all Shortest-Path Algorithms

Algorithm	SSSP		APSP		negative weights
	sparse	dense	sparse	dense	
Bellman-Ford	V^2	V^3	V^3	V^4	✓
Dijkstra	$V \log V$	V^2	$V^2 \log V$	V^3	X
Matrix Mult.	–	–	$V^3 \log V$	$V^3 \log V$	(✓)
Johnson	–	–	$V^2 \log V$	V^3	✓

can handle negative weight edges, but not negative-weight cycles