Graph Algorithms

Shortest Paths II

Michael Lampis

October 3, 2025

Shortest Path Problems

Input: **Edge-weighted** (di)graph

Shortest Path Problems

Input: **Edge-weighted** (di)graph

Problems:

- Single-Pair Shortest Path (SPSP): find shortest path from s to t.
- Single-Source Shortest Path (SSSP): find shortest path from s to everyone.
- All-Pairs Shortest Paths: find shortest paths from everyone to everyone.

Shortest Path Problems

Input: **Edge-weighted** (di)graph
Problems:

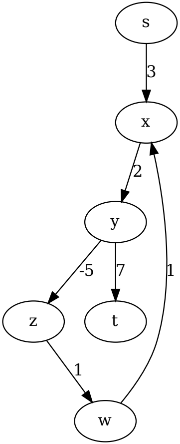
- Single-Pair Shortest Path (SPSP): find shortest path from s to t.
- Single-Source Shortest Path (SSSP): find shortest path from s to everyone.
- All-Pairs Shortest Paths: find shortest paths from everyone to everyone.

- Last lecture: Dijkstra's algorithm, SSSP with **positive weights**
- This lecture:
  - SSSP with possibly negative weights
  - APSP

Negative Weights

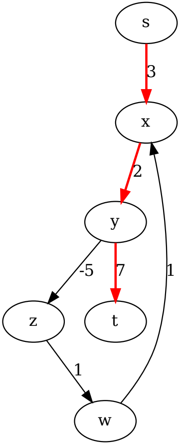
Case I: Negative cycles

- Negative cycle: a cycle whose **total** cost is negative.
- Problem: unclear what we mean by **shortest** path, if graph contains negative cycle.
    - We can repeat the cycle many times, to get the cost down to $-\infty$.
- **NB:** For positive weights, repeating cycles was always non-optimal.



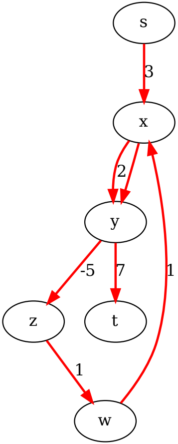
Case I: Negative cycles

- Negative cycle: a cycle whose **total** cost is negative.
- Problem: unclear what we mean by **shortest** path, if graph contains negative cycle.
  - We can repeat the cycle many times, to get the cost down to $-\infty$.
- **NB:** For positive weights, repeating cycles was always non-optimal.



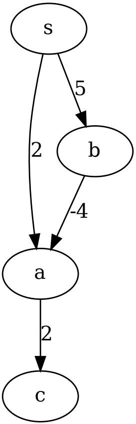
Case I: Negative cycles

- Negative cycle: a cycle whose **total** cost is negative.
- Problem: unclear what we mean by **shortest** path, if graph contains negative cycle.
  - We can repeat the cycle many times, to get the cost down to $-\infty$.
- **NB:** For positive weights, repeating cycles was always non-optimal.



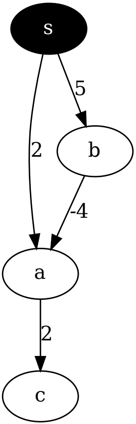
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - ⇒ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



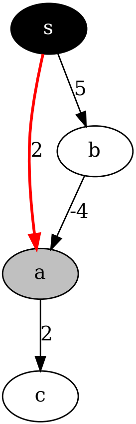
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
    - ⇒ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
    - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



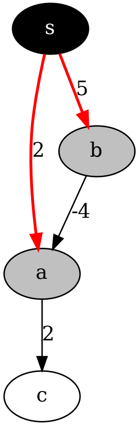
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - ⇒ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



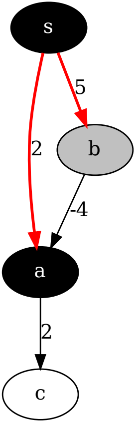
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - $\Rightarrow$ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



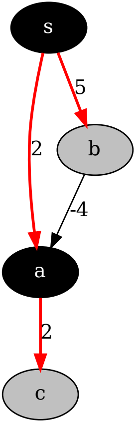
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - $\Rightarrow$ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



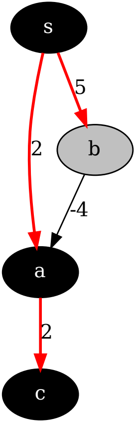
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - $\Rightarrow$ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



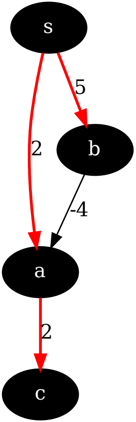
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - ⇒ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



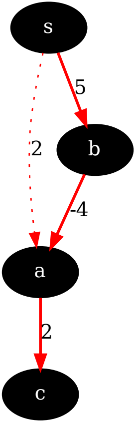
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - ⇒ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



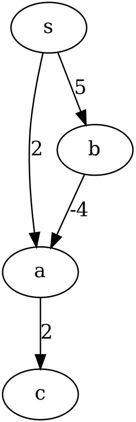
Case II: Negative weights, not cycles

- Key Dijkstra intuition: we know **optimal** path for closest Gray vertex
  - ⇒ Dequeue it and make it Black (fix solution)
- Justification: no shortcut can be found through other Gray vertices
  - (Because the other Gray vertices are farther away.)
- Reasoning is **False** for negative weights!!



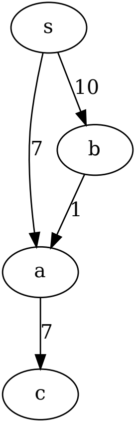
An easy fix?

- Suggestion: eliminate negative weights
- Find the max absolute value of any negative weight M.
- Add $M + 1$ to all weights.
- Run Dijkstra.
- Why not?



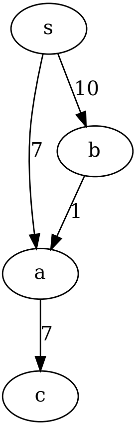
An easy fix?

- Suggestion: eliminate negative weights
- Find the max absolute value of any negative weight M.
- Add $M + 1$ to all weights.
- Run Dijkstra.
- Why not?



An easy fix?

- Suggestion: eliminate negative weights
- Find the max absolute value of any negative weight M.
- Add $M + 1$ to all weights.
- Run Dijkstra.
- Why not?
- Gives unfair advantage to paths with fewer edges. . .



Bellman-Ford Algorithm

Bellman-Ford

The Bellman-Ford algorithm:

- Can solve SSSP on digraphs with negative edge weights and no negative cycles.
- Can help **detect** negative cycles.
- Runs in time $O(nm)$ and space $O(n)$.
- Key step: **find a shortcut** (like Dijkstra)

Bellman-Ford

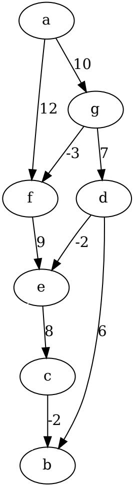
The Bellman-Ford algorithm:

- Can solve SSSP on digraphs with negative edge weights and no negative cycles.
- Can help **detect** negative cycles.
- Runs in time $O(nm)$ and space $O(n)$.
- Key step: **find a shortcut** (like Dijkstra)
- **NB**: Slower than Dijkstra, but solves more general problem.
- **NB**: We focus on digraphs, because undirected graphs with negative edges automatically have negative 2-cycles.

Bellman-Ford Algorithm

```
for v ∈ V do                                  ▷ Initialization
    dist[v] ← ∞, parent[v] ←NULL
end for
dist[s] ← 0
for i = 1 to n − 1 do
    for uv ∈ E do
        if dist[u] + w(uv) <dist[v] then      ▷ Shortcut!
            dist[v] ←dist[u] + w(uv)
            parent[v] ← u
        end if
    end for
end for
```

Example



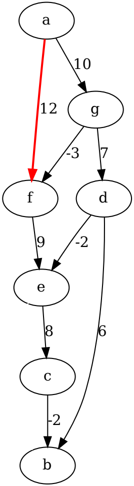
List of edges:

$\rightarrow$	af
	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞

Example



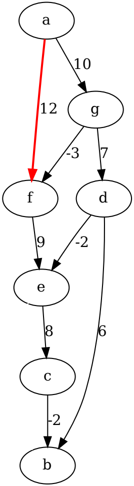
List of edges:

→	af
	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	∞

Example



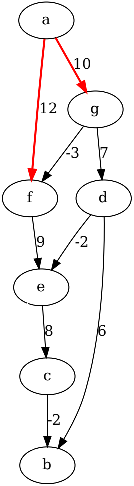
List of edges:

	af
$\rightarrow$	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	∞

Example



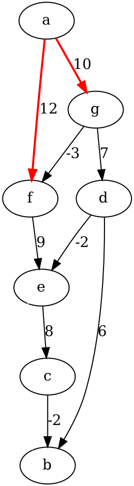
List of edges:

	af
$\rightarrow$	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



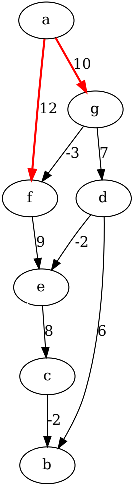
List of edges:

	af
	ag
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



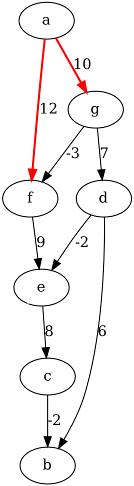
List of edges:

	af
	ag
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



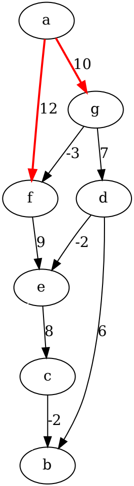
List of edges:

	af
	ag
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



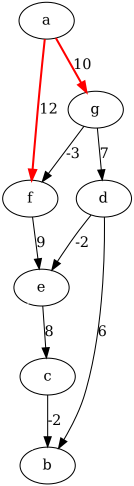
List of edges:

	af
	ag
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



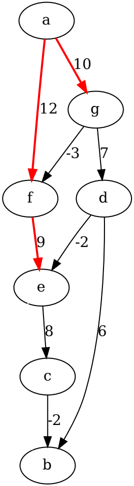
List of edges:

	af
	ag
	cb
	db
	de
	ec
$\rightarrow$	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



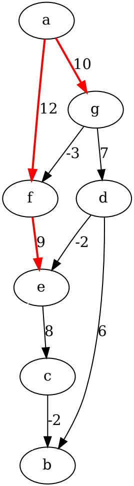
List of edges:

	af
	ag
	cb
	db
	de
	ec
→	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	21
f	12
g	10

Example



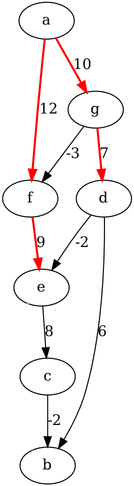
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
$\rightarrow$	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	21
f	12
g	10

Example



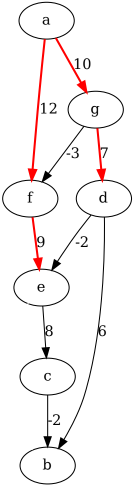
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
$\rightarrow$	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	12
g	10

Example



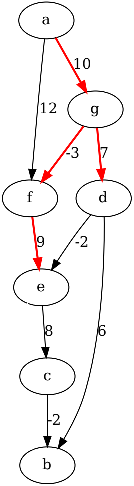
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
	gd
$\rightarrow$	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	12
g	10

Example



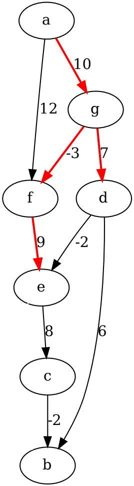
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
	gd
$\rightarrow$	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



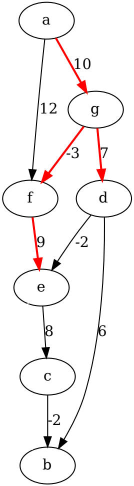
List of edges:

→	af
	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



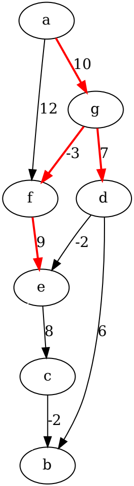
List of edges:

	af
$\rightarrow$	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



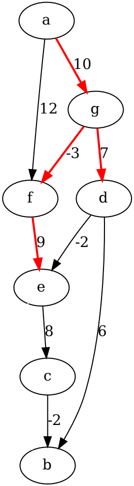
List of edges:

	af
	ag
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



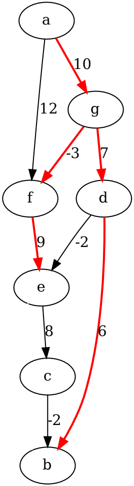
List of edges:

	af
	ag
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



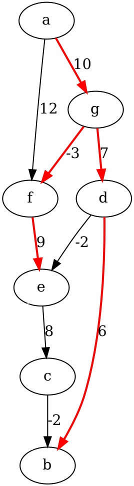
List of edges:

	af
	ag
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	∞
d	17
e	21
f	7
g	10

Example



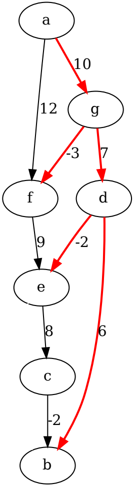
List of edges:

	af
	ag
	cb
	db
$\rightarrow$	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	∞
d	17
e	21
f	7
g	10

Example



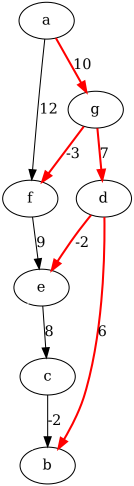
List of edges:

	af
	ag
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	∞
d	17
e	15
f	7
g	10

Example



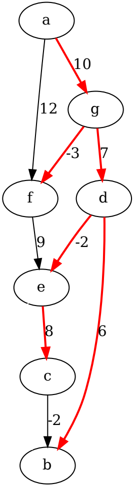
List of edges:

	af
	ag
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	∞
d	17
e	15
f	7
g	10

Example



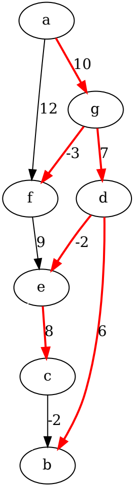
List of edges:

	af
	ag
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



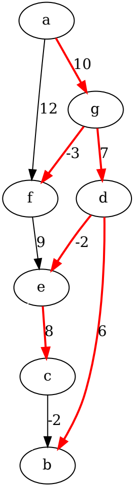
List of edges:

	af
	ag
	cb
	db
	de
	ec
→	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



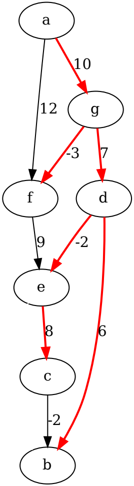
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
$\rightarrow$	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



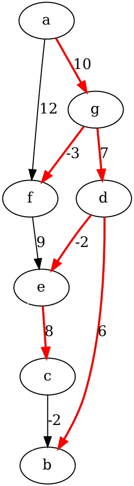
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
	gd
→	gf

Distances from *a*:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



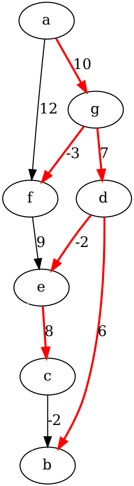
List of edges:

→	af
	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



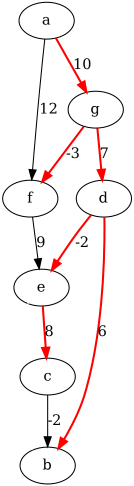
List of edges:

	af
→	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



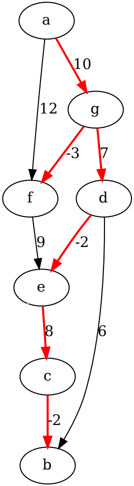
List of edges:

	af
	ag
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



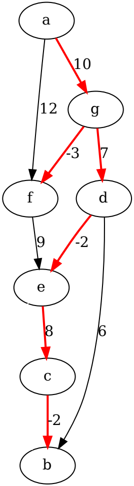
List of edges:

	af
	ag
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



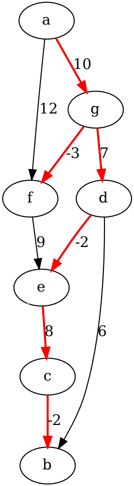
List of edges:

	af
	ag
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



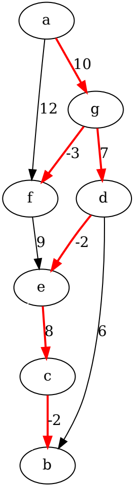
List of edges:

	af
	ag
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



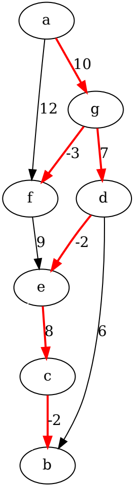
List of edges:

	af
	ag
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



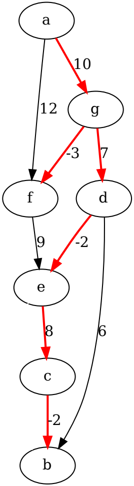
List of edges:

	af
	ag
	cb
	db
	de
	ec
→	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



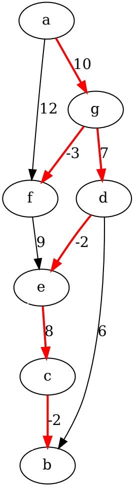
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
→	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



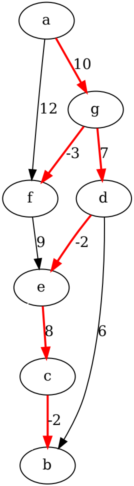
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
	gd
$\rightarrow$	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



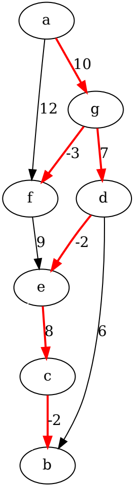
List of edges:

→	af
	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



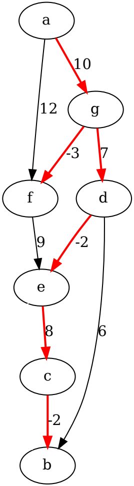
List of edges:

	af
→	ag
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



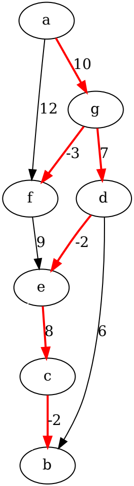
List of edges:

	af
	ag
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



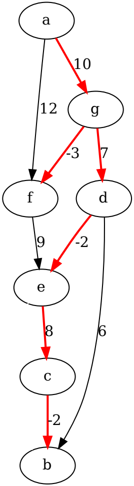
List of edges:

	af
	ag
	cb
$\rightarrow$	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



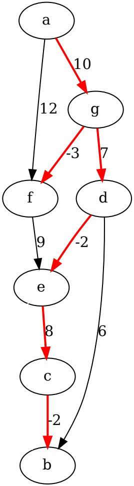
List of edges:

	af
	ag
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



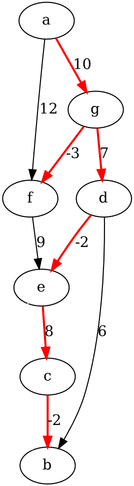
List of edges:

	af
	ag
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



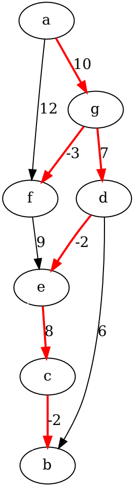
List of edges:

	af
	ag
	cb
	db
	de
	ec
→	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



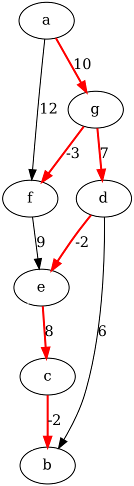
List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
→	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



List of edges:

	af
	ag
	cb
	db
	de
	ec
	fe
	gd
$\rightarrow$	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Analysis

- Running time: $O(mn)$. Space: $O(n)$
  - We assume lines 7-9 take $O(1)$ time.

Analysis

- Running time: $O(mn)$. Space: $O(n)$
  - We assume lines 7-9 take $O(1)$ time.
- Correctness based on two claims ($\forall v \in V$):
  - $\mathrm{dist}(s, v) \leq d_{BF}[v]$ at all times
  - $\mathrm{dist}(s, v) = d_{BF}[v]$ after i iterations of main loop, if shortest $s \to v$ path uses i arcs.

Analysis

- Running time: $O(mn)$. Space: $O(n)$
  - We assume lines 7-9 take $O(1)$ time.
- Correctness based on two claims ($\forall v \in V$):
  - $\mathrm{dist}(s, v) \leq d_{BF}[v]$ at all times
  - $\mathrm{dist}(s, v) = d_{BF}[v]$ after i iterations of main loop, if shortest $s \to v$ path uses i arcs.
  - Because in a digraph with no negative cycles, shortest paths have $\leq n - 1$ arcs, the above imply correctness.

Easy part

Lemma

For all $v \in V$, if $d_{BF}[v]$ is the distance computed for v by the Bellman-Ford algorithm (at any point in the execution), then $\mathrm{dist}(s, v) \leq d_{BF}[v]$.

Easy part

Lemma

For all $v \in V$, if $d_{BF}[v]$ is the distance computed for v by the Bellman-Ford algorithm (at any point in the execution), then $\mathrm{dist}(s, v) \leq d_{BF}[v]$.

Proof.

Induction on number of modifications on array d_{BF}.

- $d_{BF}[s] = 0 = \mathrm{dist}(s, s)$ and $d_{BF}[v] = \infty \geq \mathrm{dist}(s, v)$ for $s \neq v$.

Easy part

Lemma

For all $v \in V$, if $d_{BF}[v]$ is the distance computed for v by the Bellman-Ford algorithm (at any point in the execution), then $\mathrm{dist}(s, v) \leq d_{BF}[v]$.

Proof.

Induction on number of modifications on array d_{BF}.

- $d_{BF}[s] = 0 = \mathrm{dist}(s, s)$ and $d_{BF}[v] = \infty \geq \mathrm{dist}(s, v)$ for $s \neq v$.
- Suppose we examine $uv \in E$ and update $d_{BF}[v] = d_{BF}[u] + w(uv)$.
  - $\mathrm{dist}(s, u) \leq d_{BF}[u]$ (IH)
  - $\mathrm{dist}(s, v) \leq \mathrm{dist}(s, u) + w(uv) \leq d_{BF}[u] + w(uv) = d_{BF}[v]$

□

Interesting part

Lemma

If there exists a shortest $s \to v$ path using at most i arcs, then after i iterations of the main loop we have $d_{BF}[v] = \mathrm{dist}(s, v)$ $(\forall v, i)$.

Interesting part

Lemma

If there exists a shortest $s \to v$ path using at most i arcs, then after i iterations of the main loop we have $d_{BF}[v] = \mathrm{dist}(s, v)$ $(\forall v, i)$.

Proof.

Induction on i:

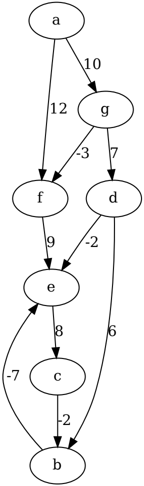
- $i = 0$: trivial. $i = 1$ not hard to see...
- Suppose claim true for i, $\exists s \to v$ path with $i + 1$ arcs.
  - If already $d_{BF}[v] \leq \mathrm{dist}(s, v)$ before the $(i + 1)$-th iteration, DONE!
  - Otherwise, let u be last vertex in $s \to v$ path.
  - (IH) $d_{BF}[u] = \mathrm{dist}(s, u)$
  - (Assumption) $\mathrm{dist}(s, v) = \mathrm{dist}(s, u) + w(uv)$
  - $d_{BF}[v]$ will be set to $\mathrm{dist}(s, u) + w(uv)$

□

Negative Cycle Detection

- Bellman-Ford algorithm can be used to detect if a digraph contains a negative-weight cycle.
- Key idea: if no cycle exists, distances **stabilize** after $n-1$ iterations of the outer loop.
- In fact, this is an **if and only if**: if distances stabilize, then no cycle exists.
  - Algorithm: check that no shortcut exists after $n-1$ iterations.

Example



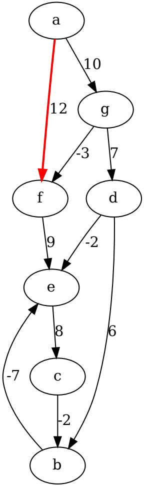
List of edges:

→	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞

Example



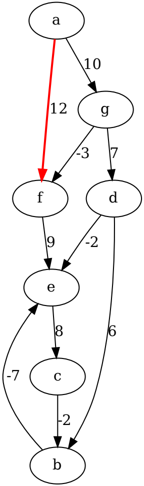
List of edges:

→	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	∞

Example



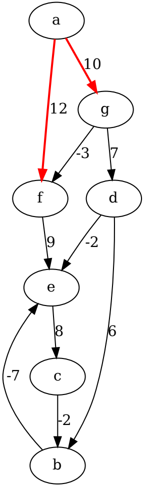
List of edges:

	af
→	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	∞

Example



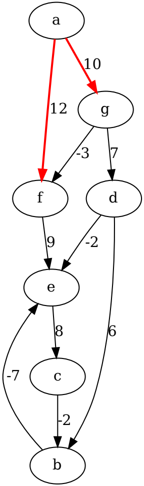
List of edges:

	af
→	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



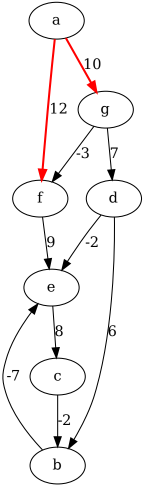
List of edges:

	af
	ag
$\rightarrow$	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



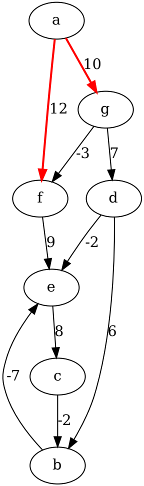
List of edges:

	af
	ag
	be
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



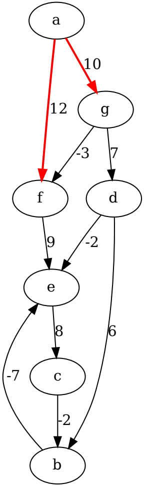
List of edges:

	af
	ag
	be
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



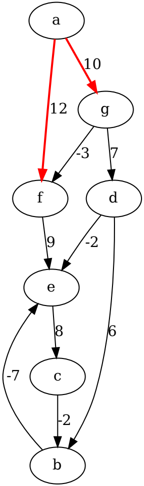
List of edges:

	af
	ag
	be
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



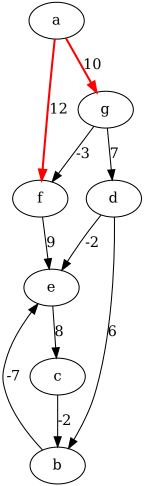
List of edges:

	af
	ag
	be
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



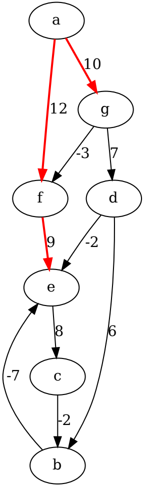
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
$\rightarrow$	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	∞
f	12
g	10

Example



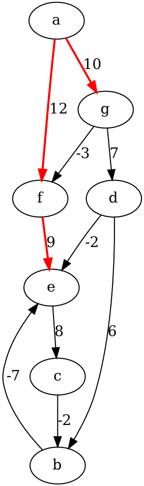
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
→	fe
	gd
	gf

Distances from *a*:

a	0
b	∞
c	∞
d	∞
e	21
f	12
g	10

Example



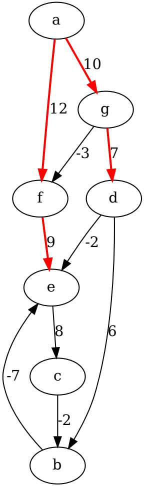
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
→	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	∞
e	21
f	12
g	10

Example



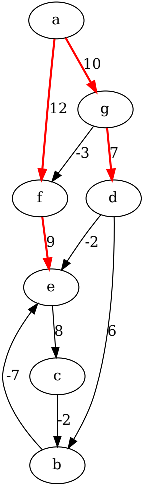
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
$\rightarrow$	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	12
g	10

Example



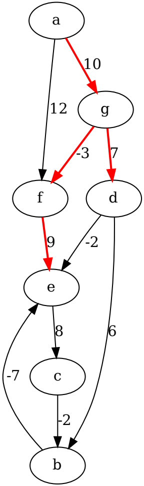
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
→	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	12
g	10

Example



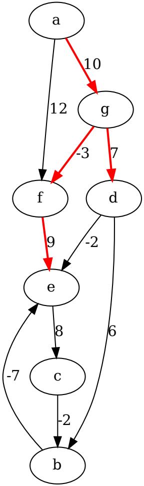
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
→	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



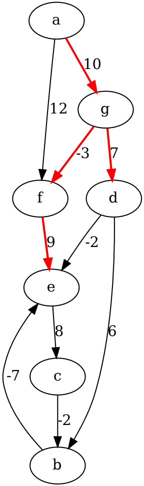
List of edges:

→	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



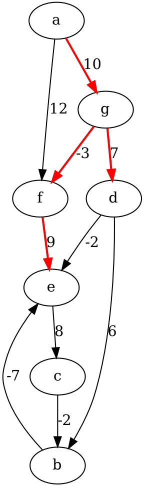
List of edges:

	af
→	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from *a*:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



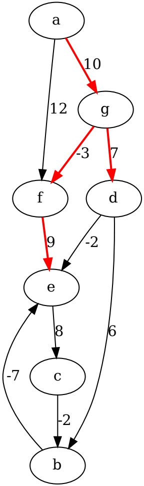
List of edges:

	af
	ag
→	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



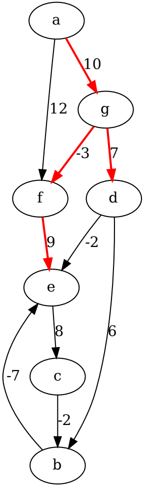
List of edges:

	af
	ag
	be
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



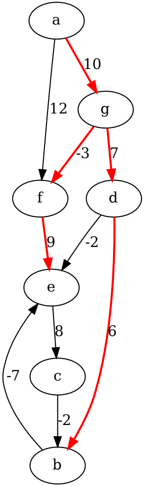
List of edges:

	af
	ag
	be
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	∞
c	∞
d	17
e	21
f	7
g	10

Example



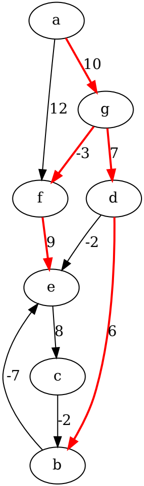
List of edges:

	af
	ag
	be
	cb
$\rightarrow$	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	∞
d	17
e	21
f	7
g	10

Example



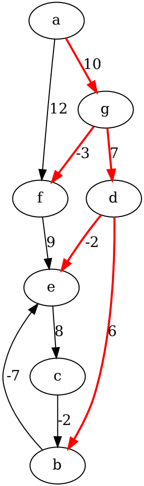
List of edges:

	af
	ag
	be
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	∞
d	17
e	21
f	7
g	10

Example



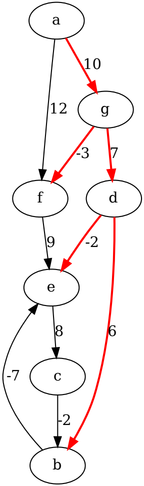
List of edges:

	af
	ag
	be
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from *a*:

a	0
b	23
c	∞
d	17
e	15
f	7
g	10

Example



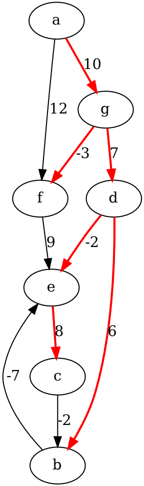
List of edges:

	af
	ag
	be
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	∞
d	17
e	15
f	7
g	10

Example



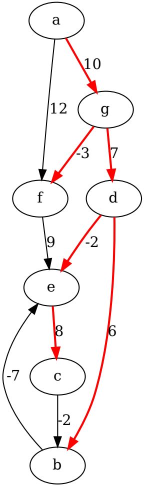
List of edges:

	af
	ag
	be
	cb
	db
	de
$\rightarrow$	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



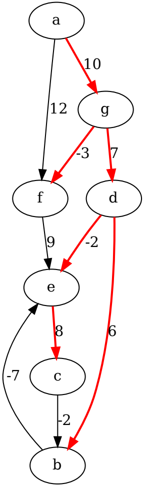
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
→	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



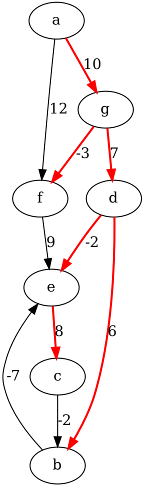
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
→	gd
	gf

Distances from *a*:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



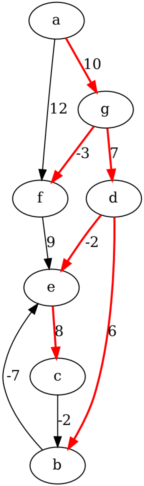
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
$\rightarrow$	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



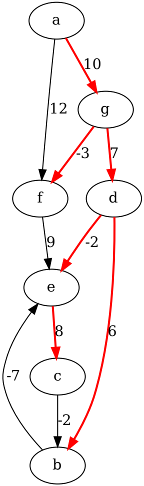
List of edges:

→	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



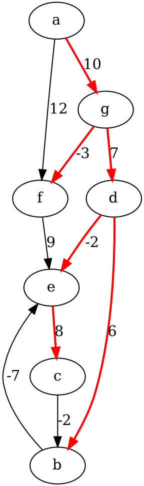
List of edges:

	af
→	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



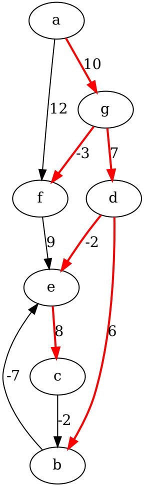
List of edges:

	af
	ag
$\rightarrow$	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



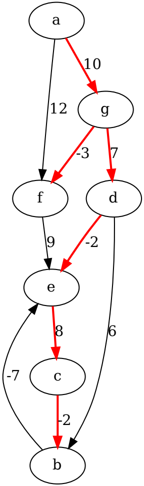
List of edges:

	af
	ag
	be
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	23
c	23
d	17
e	15
f	7
g	10

Example



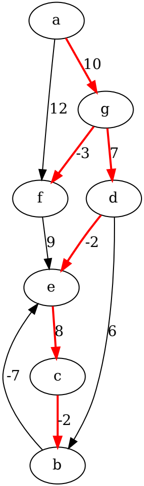
List of edges:

	af
	ag
	be
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



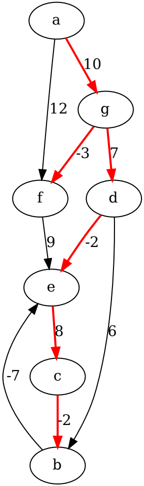
List of edges:

	af
	ag
	be
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



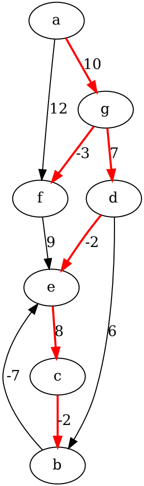
List of edges:

	af
	ag
	be
	cb
	db
→	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



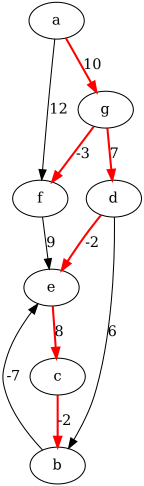
List of edges:

	af
	ag
	be
	cb
	db
	de
$\rightarrow$	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



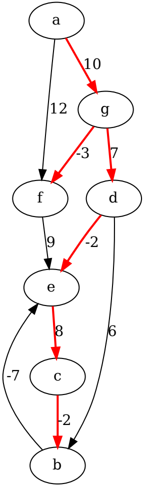
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
$\rightarrow$	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



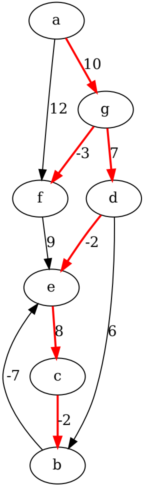
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
→	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



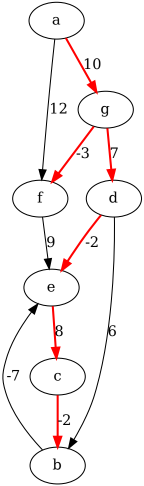
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
$\rightarrow$	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



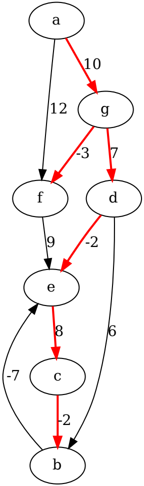
List of edges:

→	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



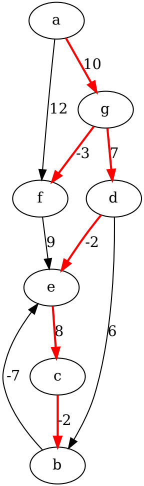
List of edges:

	af
→	ag
	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



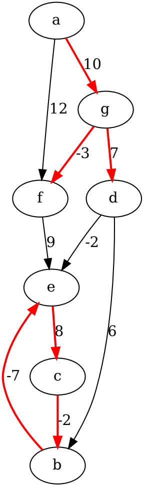
List of edges:

	af
	ag
→	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	15
f	7
g	10

Example



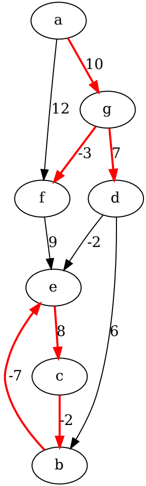
List of edges:

	af
	ag
$\rightarrow$	be
	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	14
f	7
g	10

Example



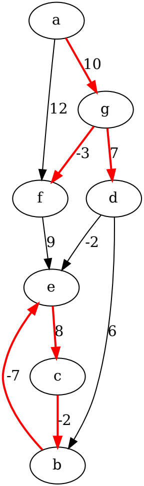
List of edges:

	af
	ag
	be
→	cb
	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	14
f	7
g	10

Example



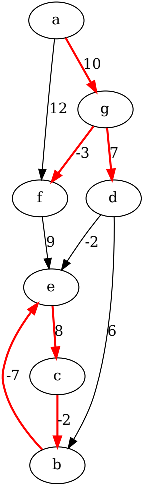
List of edges:

	af
	ag
	be
	cb
→	db
	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	14
f	7
g	10

Example



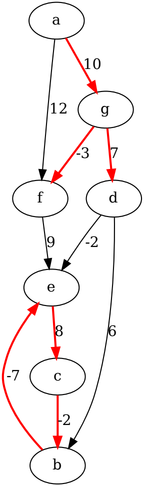
List of edges:

	af
	ag
	be
	cb
	db
$\to$	de
	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	14
f	7
g	10

Example



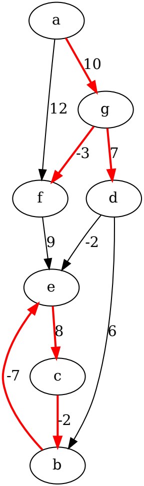
List of edges:

	af
	ag
	be
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	23
d	17
e	14
f	7
g	10

Example



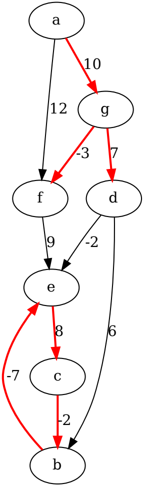
List of edges:

	af
	ag
	be
	cb
	db
	de
→	ec
	fe
	gd
	gf

Distances from a:

a	0
b	21
c	22
d	17
e	14
f	7
g	10

Example



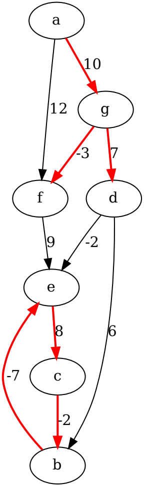
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
→	fe
	gd
	gf

Distances from *a*:

a	0
b	21
c	22
d	17
e	14
f	7
g	10

Example



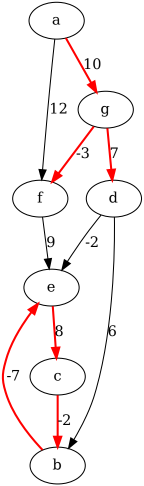
List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
→	gd
	gf

Distances from a:

a	0
b	21
c	22
d	17
e	14
f	7
g	10

Example



List of edges:

	af
	ag
	be
	cb
	db
	de
	ec
	fe
	gd
→	gf

Distances from a:

a	0
b	21
c	22
d	17
e	14
f	7
g	10

Proof of correctness

Lemma

If no negative cycle exists, distances stabilize after at most $n-1$ iterations.

Lemma

If distances stabilize after at most $n-1$ iterations, no negative cycle exists.

Proof of correctness

Lemma

If no negative cycle exists, distances stabilize after at most $n-1$ iterations.

Lemma

If distances stabilize after at most $n-1$ iterations, no negative cycle exists.

- First lemma follows from previous analysis and the fact that shortest paths use at most $n-1$ edges if no negative cycle exists.

Converse direction

Lemma

If distances stabilize after at most $n-1$ iterations, no negative cycle exists.

Converse direction

Lemma

If distances stabilize after at most $n-1$ iterations, no negative cycle exists.

Proof.

- Suppose an iteration of the main loop does not change $d_{BF}[v]$ for any v.
- Take a cycle $C = \{x_1, x_2, \ldots, x_k\}$ with weight $w(C)$.
- Because arc x_1x_2 is not a short-cut, $d_{BF}[x_2] \leq d_{BF}[x_1] + w(x_1x_2)$
- Because arc x_ix_{i+1} is not a short-cut, $d_{BF}[x_i] \leq d_{BF}[x_{i+1}] + w(x_ix_{i+1})$
- Because arc x_kx_1 is not a short-cut, $d_{BF}[x_1] \leq d_{BF}[x_k] + w(x_kx_1)$

Converse direction

Lemma

If distances stabilize after at most $n-1$ iterations, no negative cycle exists.

Proof.

- Suppose an iteration of the main loop does not change $d_{BF}[v]$ for any v.
- Take a cycle $C = \{x_1, x_2, \ldots, x_k\}$ with weight $w(C)$.
- Because arc x_1x_2 is not a short-cut, $d_{BF}[x_2] \leq d_{BF}[x_1] + w(x_1x_2)$
- Because arc x_ix_{i+1} is not a short-cut, $d_{BF}[x_i] \leq d_{BF}[x_{i+1}] + w(x_ix_{i+1})$
- Because arc x_kx_1 is not a short-cut, $d_{BF}[x_1] \leq d_{BF}[x_k] + w(x_kx_1)$
- Adding these together gives $w(C) \geq 0$.

□

APSP

Where we are

	SSSP	APSP
Pos. weights		
Gen. weights		

- Dijkstra $\rightarrow$ positive weight SSSP
- Bellman-Ford $\rightarrow$ general weight SSSP (w/o neg. cycles)

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	
Gen. weights		

- Dijkstra $\rightarrow$ positive weight SSSP
- Bellman-Ford $\rightarrow$ general weight SSSP (w/o neg. cycles)

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	
Gen. weights	$O(nm)$	

- Dijkstra $\rightarrow$ positive weight SSSP
- Bellman-Ford $\rightarrow$ general weight SSSP (w/o neg. cycles)

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	$O(n(n+m)\log n)$
Gen. weights	$O(nm)$	$O(n^2 m)$

- Dijkstra → positive weight SSSP
- Bellman-Ford → general weight SSSP (w/o neg. cycles)
- Easy idea: run previous algorithms n times for APSP
- Can we do better?

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	$O(n(n+m)\log n)$
Gen. weights	$O(nm)$	$O(n^2 m)$

- Dijkstra $\rightarrow$ positive weight SSSP
- Bellman-Ford $\rightarrow$ general weight SSSP (w/o neg. cycles)
- Easy idea: run previous algorithms n times for APSP
- Can we do better?
- Dijkstra is almost linear-time $\Rightarrow$ probably optimal.

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	$O(n(n+m)\log n)$
Gen. weights	$O(nm)$	$O(n^2 m)$

- Dijkstra $\rightarrow$ positive weight SSSP
- Bellman-Ford $\rightarrow$ general weight SSSP (w/o neg. cycles)
- Easy idea: run previous algorithms n times for APSP
- Can we do better?
- Dijkstra is almost linear-time $\Rightarrow$ probably optimal.
- BF is **NOT** optimal! (2022 breakthrough gives $O(m\log^8 n)$ time)
    - Out of scope for this class!

Where we are

	SSSP	APSP
Pos. weights	$O((n+m)\log n)$	$O(n(n+m)\log n)$
Gen. weights	$O(nm)$	$O(n^2 m)$

- Dijkstra $\rightarrow$ positive weight SSSP
- Bellman-Ford $\rightarrow$ general weight SSSP (w/o neg. cycles)
- Easy idea: run previous algorithms n times for APSP
- Can we do better?
- Dijkstra is almost linear-time $\Rightarrow$ probably optimal.
- BF is **NOT** optimal! (2022 breakthrough gives $O(m\log^8 n)$ time)
  - Out of scope for this class!
- Can APSP be solved faster than $n\times$SSSP?

APSP on dense graphs with negative weights

APSP and Matrix Multiplication

- Suppose graph is given in **adjacency matrix** form.
  - $A[i,j] = w(ij)$
- We want to do APSP faster than $O(n^4)$ (=$n\times$BF).

APSP and Matrix Multiplication

- Suppose graph is given in **adjacency matrix** form.
  - $A[i,j] = w(ij)$
- We want to do APSP faster than $O(n^4)$ ($=n\times$BF).
- $A_1[i,j] = A[i,j]$ is the cost of the shortest 1-arc path from i to j.
- We can compute $A_2[i,j]$ to be the cost of the shortest (≤ 2)-arc path from i to j as follows:

APSP and Matrix Multiplication

- Suppose graph is given in **adjacency matrix** form.
  - $A[i,j] = w(ij)$
- We want to do APSP faster than $O(n^4)$ ($= n \times$BF).
- $A_1[i,j] = A[i,j]$ is the cost of the shortest 1-arc path from i to j.
- We can compute $A_2[i,j]$ to be the cost of the shortest (≤ 2)-arc path from i to j as follows:

```
for i = 1 to n do
    for j = 1 to n do
        A_2[i,j] ← A_1[i,j]
        for k = 1 to n do
            A_2[i,j] ← min{A_2[i,j], A_1[i,k] + A_1[k,j] }
        end for
    end for
end for
```

APSP and Matrix Multiplication cont'd

- Let $A_\ell[i,j]$ be the cost of the shortest $i \to j$ path with $\leq \ell$ arcs.
    - $A_{n-1}[i,j]$ is the correct output (if no neg. cycles).

APSP and Matrix Multiplication cont'd

- Let $A_\ell[i,j]$ be the cost of the shortest $i \to j$ path with $\leq \ell$ arcs.
  - $A_{n-1}[i,j]$ is the correct output (if no neg. cycles).
- Consider the following algorithm:

$A_1[i,j] \leftarrow A[i,j]$ for all i,j
for $\ell = 2$ to n **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $A_\ell[i,j] \leftarrow A_{\ell-1}[i,j]$
            for $k = 1$ to n **do**
                $A_\ell[i,j] \leftarrow \min\{A_\ell[i,j], A_{\ell-1}[i,k] + A_1[k,j]\}$
            end for
        end for
    end for
end for

Correctness

Lemma

Previous Algorithm is correct

Correctness

Lemma

Previous Algorithm is correct

Proof.

- Induction on ℓ. For $\ell = 1$ good.
- Define $\mathrm{dist}_i(x, y)$ be minimum cost of $x \to y$ path with $\leq i$ arcs.
- We have $\mathrm{dist}_\ell(i, j) = \min_{k \in V} \mathrm{dist}_{\ell-1}(i, k) + w(kj)$
  - Proof: let k be last vertex before j in optimal path.
- By induction, $A_{\ell-1}[i, k] = \mathrm{dist}_{\ell-1}(i, k)$.

□

Analysis

- What is the running time of the previous algorithm?

Analysis

- What is the running time of the previous algorithm?
  - $O(n^4)$ (no improvement over $n \times$BF!)

Analysis

- What is the running time of the previous algorithm?
  - $O(n^4)$ (no improvement over $n \times$BF!)
- However, the algorithm should feel familiar.

Matrix Multiplication

$A_1[i,j] \leftarrow A[i,j]$ for all i,j
for $\ell = 2$ to n **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $A_\ell[i,j] \leftarrow A_{\ell-1}[i,j]$
            for $k = 1$ to n **do**
                $A_\ell[i,j] \leftarrow \min\{A_\ell[i,j], A_{\ell-1}[i,k] + A_1[k,j]\ \}$
            end for
        end for
    end for
end for

Matrix Multiplication

$A_1[i,j] \leftarrow A[i,j]$ for all i,j
for $\ell = 2$ to n **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $A_\ell[i,j] \leftarrow A_{\ell-1}[i,j]$
            for $k = 1$ to n **do**
                $A_\ell[i,j] \leftarrow \sum\{A_\ell[i,j], A_{\ell-1}[i,k] \times A_1[k,j]\}$
            end for
        end for
    end for
end for

- n matrix multiplications with $(\min, +)$ in the place of $(+, \times)$
- Output: A^n

Fast Powers

- How many multiplications do we need to compute x^{1024}?

Fast Powers

- How many multiplications do we need to compute x^{1024}?
- Naive: 1023

Fast Powers

- How many multiplications do we need to compute x^{1024}?
- Naive: 1023
- Clever: 10
  - $x \to x^2 \to x^4 \to x^8 \to \dots$

Fast Powers

- How many multiplications do we need to compute x^{1024}?
- Naive: 1023
- Clever: 10
  - $x \rightarrow x^2 \rightarrow x^4 \rightarrow x^8 \rightarrow \ldots$
- Given A we want to calculate A^n (for our version of M.M.)
- By repeated squaring, this can be done in $O(n^3 \log n)$ time.

Strassen?

- APSP boils down to a form of Matrix Multiplication.
- We perform each MM in $O(n^3)$ time.
- **But:** faster MM algorithms exist (Strassen etc.). Why not use that?

Strassen?

- APSP boils down to a form of Matrix Multiplication.
- We perform each MM in $O(n^3)$ time.
- **But:** faster MM algorithms exist (Strassen etc.). Why not use that?
- Recall the basic idea behind Strassen's algorithm:

Strassen's algorithm

- We want to calculate $C = A \times B$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Strassen's algorithm

Need to calculate:

$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

Will calculate:

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22})B_{11} \\
M_3 &= A_{11}(B_{12} - B_{22}) \\
M_4 &= A_{22}(B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12})B_{22} \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
$$

Strassen's algorithm

Need to calculate:

$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

Will calculate:

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22})B_{11} \\
M_3 &= A_{11}(B_{12} - !!B_{22}) \\
M_4 &= A_{22}(B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12})B_{22} \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
$$

Problem with Strassen's Algorithm

- We want to replace $+$ with a min operation.
- The $+$ operation has an inverse (used by Strassen), the min operation **does not**!

Problem with Strassen's Algorithm

- We want to replace $+$ with a min operation.
- The $+$ operation has an inverse (used by Strassen), the min operation **does not**!
- Currently, no algorithm better than $O(n^3)$ known for computing $(\min, +)$-product of matrices.
- Also, no better than $O(n^3)$ algorithm known for APSP (even for positive weights).
  - However, fast MM improves APSP on unweighted graphs, or graphs with small weights.