Design and Analysis of Algorithms

Shortest Paths

Dijkstra • Bellman–Ford • Floyd–Warshall • Johnson

10^5 road segments, one route query per second — and a currency graph where some edges *pay you* to travel.
Four algorithms are waiting; the weights decide which one you may hire.

Dr. Golzari

University of Hormozgan

Fall 1405

Roadmap — Where We're Going

1. **One move** — *relaxation*: the single operation every shortest-path algorithm repeats

Roadmap — Where We're Going

1. **One move** — *relaxation*: the single operation every shortest-path algorithm repeats
2. **One source** — **Dijkstra**: greed, but only if $w \geq 0$; then **Bellman–Ford**: patience, any weights

Roadmap — Where We're Going

1. **One move** — *relaxation*: the single operation every shortest-path algorithm repeats
2. **One source** — **Dijkstra**: greed, but only if $w \geq 0$; then **Bellman–Ford**: patience, any weights
3. **Everyone to everyone** — **Floyd–Warshall**: the whole $n \times n$ table with three loops

Roadmap — Where We're Going

1. **One move** — *relaxation*: the single operation every shortest-path algorithm repeats
2. **One source** — **Dijkstra**: greed, but only if $w \geq 0$; then **Bellman–Ford**: patience, any weights
3. **Everyone to everyone** — **Floyd–Warshall**: the whole $n \times n$ table with three loops
4. **The sparse champion** — **Johnson**: reweight once, then unleash Dijkstra V times

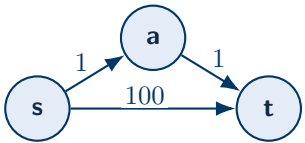
Roadmap — Where We're Going

1. **One move** — *relaxation*: the single operation every shortest-path algorithm repeats
2. **One source** — **Dijkstra**: greed, but only if $w \geq 0$; then **Bellman–Ford**: patience, any weights
3. **Everyone to everyone** — **Floyd–Warshall**: the whole $n \times n$ table with three loops
4. **The sparse champion** — **Johnson**: reweight once, then unleash Dijkstra V times

Today's promise

By the end you can look at any weighted graph and name the right algorithm — and say *exactly* why the other three fail.

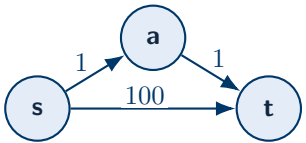
First: What Does “Shortest” Even Mean Now?



Pop Quiz

BFS returns *fewest edges*. Which route does it pick here, and what does that route cost?

First: What Does “Shortest” Even Mean Now?

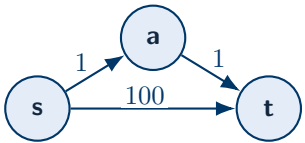


Pop Quiz

BFS returns *fewest edges*. Which route does it pick here, and what does that route cost?

- BFS: $s \to t$ — one edge … but it costs 100. The cheapest trip is $s \to a \to t = 2$.

First: What Does “Shortest” Even Mean Now?

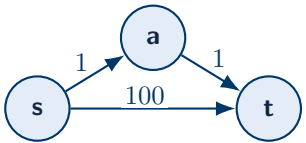


Pop Quiz

BFS returns *fewest edges*. Which route does it pick here, and what does that route cost?

- BFS: $s \to t$ — one edge … but it costs 100. The cheapest trip is $s \to a \to t = 2$.
- **New contract:** a path’s cost = sum of its edge weights; shortest path = *minimum-cost* path

First: What Does "Shortest" Even Mean Now?



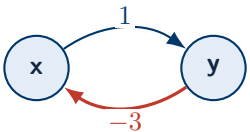
Pop Quiz

BFS returns *fewest edges*. Which route does it pick here, and what does that route cost?

- BFS: $s \to t$ — one edge … but it costs 100. The cheapest trip is $s \to a \to t = 2$.
- **New contract:** a path's cost = sum of its edge weights; shortest path = *minimum-cost* path
- **Today's problem (SSSP):** from one source s, find $\delta(s, v)$ for *every* v — and one route achieving it

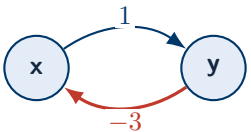
The Real Enemy: Negative Cycles

- Weights *below zero* are legal (refunds, time saved, arbitrage)… but watch this one:



The Real Enemy: Negative Cycles

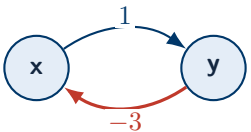
- Weights *below zero* are legal (refunds, time saved, arbitrage)… but watch this one:



- Loop once: $1 - 3 = -2$. Twice: -4. k times: $-2k \to -\infty$

The Real Enemy: Negative Cycles

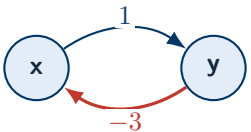
- Weights *below zero* are legal (refunds, time saved, arbitrage)… but watch this one:



- Loop once: $1 - 3 = -2$. Twice: -4. k times: $-2k \to -\infty$
- Past a negative cycle, "shortest path" stops existing — the problem itself dissolves

The Real Enemy: Negative Cycles

- Weights *below zero* are legal (refunds, time saved, arbitrage)… but watch this one:



- Loop once: $1 - 3 = -2$. Twice: -4. k times: $-2k \to -\infty$
- Past a negative cycle, "shortest path" stops existing — the problem itself dissolves

Our contract for today

Weights may be negative, but we assume no reachable negative cycle — and two of today's algorithms will *detect* one for free when we break the contract.

One Move Rules Them All: *Relax*

Relax(u, v, w)

1: **if** $d[u] + w(u, v) < d[v]$ **then**
2: $\quad d[v] \leftarrow d[u] + w(u, v)$
3: $\quad \pi[v] \leftarrow u$

One Move Rules Them All: *Relax*

Relax(u, v, w)
1: **if** $d[u] + w(u, v) < d[v]$ **then**
2: $\quad d[v] \leftarrow d[u] + w(u, v)$
3: $\quad \pi[v] \leftarrow u$

- "Going to v *via* u is cheaper than the best route I have? Then update — and remember the parent $\pi[v] = u$"

One Move Rules Them All: *Relax*

Relax(u, v, w)
1: **if** $d[u] + w(u, v) < d[v]$ **then**
2: $\quad d[v] \leftarrow d[u] + w(u, v)$
3: $\quad \pi[v] \leftarrow u$

- "Going to v *via* u is cheaper than the best route I have? Then update — and remember the parent $\pi[v] = u$"
- Invariant: $d[v]$ is always an overestimate of $\delta(s, v)$ — every Relax either improves it or leaves it

One Move Rules Them All: *Relax*

Relax(u, v, w)
1: **if** $d[u] + w(u, v) < d[v]$ **then**
2: $\quad d[v] \leftarrow d[u] + w(u, v)$
3: $\quad \pi[v] \leftarrow u$

- "Going to v *via* u is cheaper than the best route I have? Then update — and remember the parent $\pi[v] = u$"
- Invariant: $d[v]$ is always an overestimate of $\delta(s, v)$ — every Relax either improves it or leaves it
- The π pointers always form a tree rooted at s: read a route by walking parents backwards

One Move Rules Them All: *Relax*

Relax(u, v, w)
1: **if** $d[u] + w(u, v) < d[v]$ **then**
2: $\quad d[v] \leftarrow d[u] + w(u, v)$
3: $\quad \pi[v] \leftarrow u$

- "Going to v *via* u is cheaper than the best route I have? Then update — and remember the parent $\pi[v] = u$"
- Invariant: $d[v]$ is always an overestimate of $\delta(s, v)$ — every Relax either improves it or leaves it
- The π pointers always form a tree rooted at s: read a route by walking parents backwards

The whole session in one sentence

Dijkstra, Bellman–Ford, Floyd–Warshall, Johnson = four *schedules* for deciding which Relax to run when.

Dijkstra: Which Vertex May We Settle Forever?

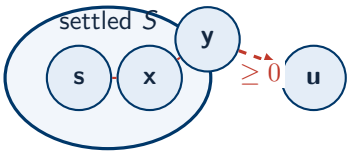
Pop Quiz

All weights ≥ 0. Each step we *settle* one vertex (fix its distance, never touch it again). Which one is safe to settle?

Dijkstra: Which Vertex May We Settle Forever?

Pop Quiz

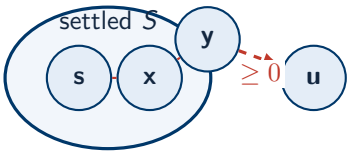
All weights ≥ 0. Each step we *settle* one vertex (fix its distance, never touch it again). Which one is safe to settle?



Dijkstra: Which Vertex May We Settle Forever?

Pop Quiz

All weights ≥ 0. Each step we *settle* one vertex (fix its distance, never touch it again). Which one is safe to settle?

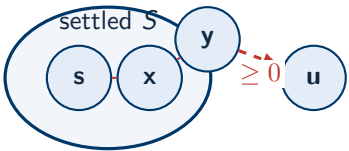


- Settle the unsettled vertex with the smallest $d[u]$

Dijkstra: Which Vertex May We Settle Forever?

Pop Quiz

All weights ≥ 0. Each step we *settle* one vertex (fix its distance, never touch it again). Which one is safe to settle?

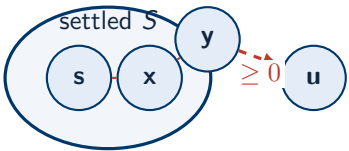


- Settle the unsettled vertex with the smallest $d[u]$
- Any real path to u must escape S through some frontier vertex y — and $d[y] \geq d[u]$ (else y, not u, was minimum)

Dijkstra: Which Vertex May We Settle Forever?

Pop Quiz

All weights ≥ 0. Each step we *settle* one vertex (fix its distance, never touch it again). Which one is safe to settle?



- Settle the unsettled vertex with the smallest $d[u]$
- Any real path to u must escape S through some frontier vertex y — and $d[y] \geq d[u]$ (else y, not u, was minimum)
- The escape edge was already relaxed when x settled, and weights ≥ 0 only add: no path can beat $d[u]$ — greedy is *proven*, not lucky

Dijkstra — The Pseudocode

Dijkstra(G, w, s)

1: $d[v] \leftarrow \infty$, *settled*$[v] \leftarrow$ false, for all $v \in V$; $d[s] \leftarrow 0$
2: $Q \leftarrow$ min-priority queue of all v, keyed by $d[v]$
3: **while** Q is not empty **do**
4: $u \leftarrow$ *extract-min*(Q); mark u settled
5: **for** $v \in Adj[u]$ with v not settled **do**
6: **Relax**(u, v, w) ▷ decrease-key when improved
7: **return** d

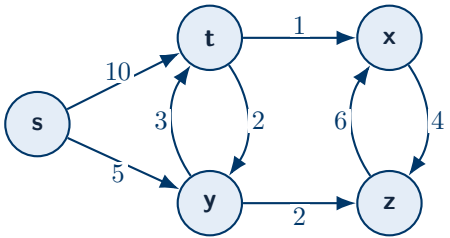
Dijkstra — The Pseudocode

Dijkstra(G, w, s)

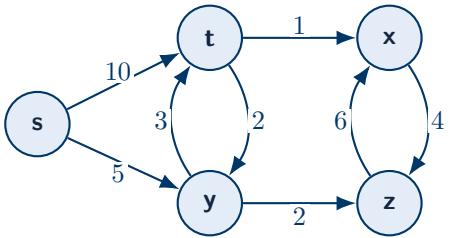
1: $d[v] \leftarrow \infty$, *settled*$[v] \leftarrow$ false, for all $v \in V$; $d[s] \leftarrow 0$
2: $Q \leftarrow$ min-priority queue of all v, keyed by $d[v]$
3: **while** Q is not empty **do**
4: $\quad u \leftarrow$ *extract-min*(Q); mark u settled
5: $\quad$ **for** $v \in Adj[u]$ with v not settled **do**
6: $\quad\quad$ **Relax**(u, v, w) $\triangleright$ decrease-key when improved
7: **return** d

- The settled guard is free insurance while $w \geq 0$ — and it is exactly the line that will fail us in ten minutes

Our Specimen for Dijkstra



Our Specimen for Dijkstra



Vitals

$n = 5$ vertices, $m = 8$ edges, every weight ≥ 0 — Dijkstra's home turf.
Keep it in your head: the next three slides settle this exact graph, one vertex at a time.

Walkthrough — Dijkstra, Part 1

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
init	—	$d[s] = 0$ only	∞	∞	∞	∞

Walkthrough — Dijkstra, Part 1

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
init	—	$d[s] = 0$ only	∞	∞	∞	∞
1	s (0)	t: $\infty \to 10$, y: $\infty \to 5$	10	∞	5	∞

Walkthrough — Dijkstra, Part 1

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
init	—	$d[s] = 0$ only	∞	∞	∞	∞
1	$s\,(0)$	t: $\infty \to 10$, y: $\infty \to 5$	10	∞	5	∞
2	$y\,(5)$	t: $10 \to 8$, z: $\infty \to 7$	8	∞	5	7

Walkthrough — Dijkstra, Part 1

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
init	—	$d[s] = 0$ only	∞	∞	∞	∞
1	$s\,(0)$	t: $\infty \to 10$, y: $\infty \to 5$	10	∞	5	∞
2	$y\,(5)$	t: $10 \to 8$, z: $\infty \to 7$	8	∞	5	7

- Red = the value was just improved by a Relax

Walkthrough — Dijkstra, Part 1

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
init	—	$d[s] = 0$ only	∞	∞	∞	∞
1	$s\,(0)$	t: $\infty \to 10$, y: $\infty \to 5$	10	∞	5	∞
2	$y\,(5)$	t: $10 \to 8$, z: $\infty \to 7$	8	∞	5	7

- Red = the value was just improved by a Relax
- *y stole t*'s route: $s \to y \to t = 8$ beats the direct $s \to t = 10$

Walkthrough — Dijkstra, Part 2

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
3	z (7)	x: $\infty \to 13$	8	13	5	7

Walkthrough — Dijkstra, Part 2

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
3	z (7)	x: $\infty \to 13$	8	13	5	7
4	t (8)	x: $13 \to 9$	8	9	5	7

Walkthrough — Dijkstra, Part 2

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
3	z (7)	x: $\infty \to 13$	8	13	5	7
4	t (8)	x: $13 \to 9$	8	9	5	7
5	x (9)	— (z already cheaper)	8	9	5	7

Walkthrough — Dijkstra, Part 2

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
3	$z\,(7)$	x: $\infty \to 13$	8	13	5	7
4	$t\,(8)$	x: $13 \to 9$	8	9	5	7
5	$x\,(9)$	— (z already cheaper)	8	9	5	7

Result & verification

π: $t \leftarrow y$, $z \leftarrow y$, $x \leftarrow t$; check $\delta(s,x)$: $s \to y \to t \to x = 5 + 3 + 1 = 9$ ✓

$\delta(s,z) = 5 + 2 = 7$ ✓

Walkthrough — Dijkstra, Part 2

step	pop	who improves	$d[t]$	$d[x]$	$d[y]$	$d[z]$
3	$z\,(7)$	x: $\infty \to 13$	8	13	5	7
4	$t\,(8)$	x: $13 \to 9$	8	9	5	7
5	$x\,(9)$	— (z already cheaper)	8	9	5	7

Result & verification

π: $t \leftarrow y$, $z \leftarrow y$, $x \leftarrow t$; check $\delta(s,x)$: $s \to y \to t \to x = 5 + 3 + 1 = 9$ ✓
$\delta(s,z) = 5 + 2 = 7$ ✓

- Step 4: t popped *after* z ($8 > 7$) yet t pulls x from 13 down to 9 — the queue, not the eye, decides who is ready

Dijkstra: The Receipt

- Each vertex leaves Q once: V extract-mins; each edge is relaxed once from its tail: E decrease-key attempts

Dijkstra: The Receipt

- Each vertex leaves Q once: V extract-mins; each edge is relaxed once from its tail: E decrease-key attempts
- Array-scan queue: extract-min costs $O(V) \Rightarrow O(V^2 + E)$

Dijkstra: The Receipt

- Each vertex leaves Q once: V extract-mins; each edge is relaxed once from its tail: E decrease-key attempts
- Array-scan queue: extract-min costs $O(V) \Rightarrow O(V^2 + E)$
- Binary heap: both operations $O(\log V) \Rightarrow O\big((V + E) \log V\big)$

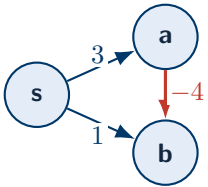
Dijkstra: The Receipt

- Each vertex leaves Q once: V extract-mins; each edge is relaxed once from its tail: E decrease-key attempts
- Array-scan queue: extract-min costs $O(V) \Rightarrow O(V^2 + E)$
- Binary heap: both operations $O(\log V) \Rightarrow O\big((V+E)\log V\big)$

In real units — the road network

$V = 10^5$, $E = 2 \times 10^5$, $\log_2 V \approx 17$:
heap $\approx 3 \times 10^5 \times 17 \approx 5 \times 10^6$ steps (milliseconds); array $\approx 10^{10}$ steps — $2000\times$ more work for the same answer.

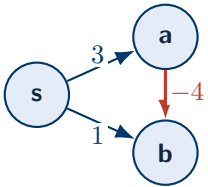
Pop Quiz: What Does Dijkstra Answer Here?



Trace it

Who gets settled, in which order — and is the answer true?

Pop Quiz: What Does Dijkstra Answer Here?

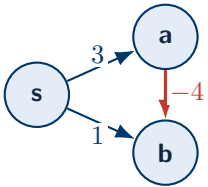


Trace it

Who gets settled, in which order — and is the answer true?

pop	what happens
s (0)	$d[a] \leftarrow 3, \quad d[b] \leftarrow 1$
b (1)	settled — frozen at 1 forever
a (3)	edge $a \to b$ wants $3 - 4 = -1 < 1$ … but b is settled: skipped

Pop Quiz: What Does Dijkstra Answer Here?



Trace it

Who gets settled, in which order — and is the answer true?

pop	what happens
s (0)	$d[a] \leftarrow 3, \quad d[b] \leftarrow 1$
b (1)	settled — frozen at 1 forever
a (3)	edge $a \to b$ wants $3 - 4 = -1 < 1$ … but b is settled: skipped

The verdict

Dijkstra reports $d[b] = 1$; the truth is $s \to a \to b = -1$.
The assumption that died: $w \geq 0$. Greedy froze b before its best route existed.

Bellman–Ford: Replace Greed with Patience

The question

Greed is dead with negative edges. What can we still *trust*?

Bellman–Ford: Replace Greed with Patience

The question

Greed is dead with negative edges. What can we still *trust*?

- Trust the budget, not the destination: after relaxing every edge once, $d[v]$ is the cheapest route to v using at most 1 edge; after round k: at most k edges

Bellman–Ford: Replace Greed with Patience

The question

Greed is dead with negative edges. What can we still *trust*?

- Trust the budget, not the destination: after relaxing every edge once, $d[v]$ is the cheapest route to v using at most 1 edge; after round k: at most k edges
- A shortest path never repeats a vertex $\Rightarrow$ it has at most $V-1$ edges $\Rightarrow$ $V-1$ rounds suffice

Bellman–Ford: Replace Greed with Patience

The question

Greed is dead with negative edges. What can we still *trust*?

- Trust the budget, not the destination: after relaxing every edge once, $d[v]$ is the cheapest route to v using at most 1 edge; after round k: at most k edges
- A shortest path never repeats a vertex $\Rightarrow$ it has at most $V-1$ edges $\Rightarrow$ $V-1$ rounds suffice
- Improvements cascade: round $k+1$ extends every round-k route by exactly one more edge

Bellman–Ford: Replace Greed with Patience

The question

Greed is dead with negative edges. What can we still *trust*?

- Trust the budget, not the destination: after relaxing every edge once, $d[v]$ is the cheapest route to v using at most 1 edge; after round k: at most k edges
- A shortest path never repeats a vertex $\Rightarrow$ it has at most $V-1$ edges $\Rightarrow$ $V-1$ rounds suffice
- Improvements cascade: round $k+1$ extends every round-k route by exactly one more edge

The trade, in one line

Dijkstra settles one *vertex* per step; Bellman–Ford buys one extra *edge of budget* per round.

Bellman–Ford — The Pseudocode

```
BellmanFord(G, w, s)
1:  d[v] ← ∞ for all v ∈ V;   d[s] ← 0
2:  for i ← 1 to |V| − 1 do
3:      changed ← false
4:      for each edge (u, v) ∈ E, in any fixed order do
5:          if d[u] + w(u, v) < d[v] then
6:              d[v] ← d[u] + w(u, v);   π[v] ← u;   changed ← true
7:      if not changed then
8:          break
9:  for each edge (u, v) ∈ E do
10:     if d[u] + w(u, v) < d[v] then
11:         return negative cycle reachable from s
12: return d
```

Bellman–Ford — The Pseudocode

```
    BellmanFord(G, w, s)
 1: d[v] ← ∞ for all v ∈ V;   d[s] ← 0
 2: for i ← 1 to |V| − 1 do
 3:     changed ← false
 4:     for each edge (u, v) ∈ E, in any fixed order do
 5:         if d[u] + w(u, v) < d[v] then
 6:             d[v] ← d[u] + w(u, v);   π[v] ← u;   changed ← true
 7:     if not changed then
 8:         break
 9: for each edge (u, v) ∈ E do
10:     if d[u] + w(u, v) < d[v] then
11:         return negative cycle reachable from s
12: return d
```

- Lines 3–10: $V - 1$ patient rounds (early exit when a round changes nothing)

Bellman–Ford — The Pseudocode

```
    BellmanFord(G, w, s)
 1: d[v] ← ∞ for all v ∈ V;   d[s] ← 0
 2: for i ← 1 to |V| − 1 do
 3:     changed ← false
 4:     for each edge (u, v) ∈ E, in any fixed order do
 5:         if d[u] + w(u, v) < d[v] then
 6:             d[v] ← d[u] + w(u, v);   π[v] ← u;   changed ← true
 7:     if not changed then
 8:         break
 9: for each edge (u, v) ∈ E do
10:     if d[u] + w(u, v) < d[v] then
11:         return negative cycle reachable from s
12: return d
```

- Lines 3–10: $V - 1$ patient rounds (early exit when a round changes nothing)
- Lines 11–15: one extra pass — if anything still improves, a negative cycle is alive; we return to it soon

Walkthrough — Bellman–Ford on a Negative Edge

Specimen & scan order (order is free, keep it fixed)

$$E = \big[\ a{\to}b\ (-3),\quad b{\to}c\ (2),\quad s{\to}c\ (8),\quad s{\to}b\ (5),\quad s{\to}a\ (4)\ \big]$$

Walkthrough — Bellman–Ford on a Negative Edge

Specimen & scan order (order is free, keep it fixed)

$$E = \big[\ a \to b\ (-3),\quad b \to c\ (2),\quad s \to c\ (8),\quad s \to b\ (5),\quad s \to a\ (4)\ \big]$$

round	$d[a]$	$d[b]$	$d[c]$	best routes captured so far
init	∞	∞	∞	—

Walkthrough — Bellman–Ford on a Negative Edge

Specimen & scan order (order is free, keep it fixed)

$$E = \big[\ a \to b\ (-3),\quad b \to c\ (2),\quad s \to c\ (8),\quad s \to b\ (5),\quad s \to a\ (4)\ \big]$$

round	$d[a]$	$d[b]$	$d[c]$	best routes captured so far
init	∞	∞	∞	—
1	4	5	8	1-edge routes only

Walkthrough — Bellman–Ford on a Negative Edge

Specimen & scan order (order is free, keep it fixed)

$$E = \big[\ a{\to}b\ (-3),\quad b{\to}c\ (2),\quad s{\to}c\ (8),\quad s{\to}b\ (5),\quad s{\to}a\ (4)\ \big]$$

round	$d[a]$	$d[b]$	$d[c]$	best routes captured so far
init	∞	∞	∞	—
1	4	5	8	1-edge routes only
2	4	1	3	$s{\to}a{\to}b$, $s{\to}a{\to}b{\to}c$

Walkthrough — Bellman–Ford on a Negative Edge

Specimen & scan order (order is free, keep it fixed)

$$E = \big[\ a{\to}b\ (-3),\quad b{\to}c\ (2),\quad s{\to}c\ (8),\quad s{\to}b\ (5),\quad s{\to}a\ (4)\ \big]$$

round	$d[a]$	$d[b]$	$d[c]$	best routes captured so far
init	∞	∞	∞	—
1	4	5	8	1-edge routes only
2	4	1	3	$s{\to}a{\to}b$, $s{\to}a{\to}b{\to}c$
3	4	1	3	nothing improved $\Rightarrow$ early exit

Walkthrough — Bellman–Ford on a Negative Edge

Specimen & scan order (order is free, keep it fixed)

$$E = \big[\ a{\to}b\ (-3),\quad b{\to}c\ (2),\quad s{\to}c\ (8),\quad s{\to}b\ (5),\quad s{\to}a\ (4)\ \big]$$

round	$d[a]$	$d[b]$	$d[c]$	best routes captured so far
init	∞	∞	∞	—
1	4	5	8	1-edge routes only
2	4	1	3	$s{\to}a{\to}b$, $s{\to}a{\to}b{\to}c$
3	4	1	3	nothing improved ⇒ early exit

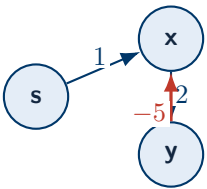
Verify & the moral

$\delta(a) = 4 = s{\to}a$ ✓; $\delta(b) = 4 - 3 = 1 = s{\to}a{\to}b$ ✓; $\delta(c) = 1 + 2 = 3 = s{\to}a{\to}b{\to}c$ ✓

Round 2 is where patience wins: the direct $s{\to}b = 5$ finally loses to $s{\to}a{\to}b = 1$ — greed froze, patience waited.

The Free Bonus: Detecting Negative Cycles

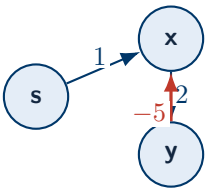
- After $V-1$ rounds every *legitimate* route is done — so if one more pass still improves, someone is looping: negative cycle



round	$d[x]$	$d[y]$
1	-2	3
2	-5	0
3	-8	-3

The Free Bonus: Detecting Negative Cycles

- After $V-1$ rounds every *legitimate* route is done — so if one more pass still improves, someone is looping: negative cycle

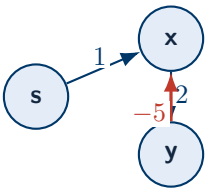


round	$d[x]$	$d[y]$
1	-2	3
2	-5	0
3	-8	-3

- Every round, both drop by exactly $-5+2=3$, the cycle's weight: $d \to -\infty$, no shortest path exists

The Free Bonus: Detecting Negative Cycles

- After $V-1$ rounds every *legitimate* route is done — so if one more pass still improves, someone is looping: negative cycle



round	$d[x]$	$d[y]$
1	-2	3
2	-5	0
3	-8	-3

- Every round, both drop by exactly $-5+2=3$, the cycle's weight: $d \to -\infty$, no shortest path exists

Spot the money pump

Currencies $x \to y \to x$ with total < 0 = *arbitrage*.
Bellman–Ford doesn't crash on it — it reports it.

Bellman–Ford: The Receipt

- $V-1$ rounds, each scanning all E edges once: $O(V \cdot E)$; the detection pass adds one more scan

Bellman–Ford: The Receipt

- $V - 1$ rounds, each scanning all E edges once: $O(V \cdot E)$; the detection pass adds one more scan
- Early exit helps in practice — our specimen stopped after 2 rounds — but the worst case keeps its $O(VE)$

Bellman–Ford: The Receipt

- $V - 1$ rounds, each scanning all E edges once: $O(V \cdot E)$; the detection pass adds one more scan
- Early exit helps in practice — our specimen stopped after 2 rounds — but the worst case keeps its $O(VE)$
- Edge order never affects *correctness*; it only decides how fast the cascade converges

Bellman–Ford: The Receipt

- $V-1$ rounds, each scanning all E edges once: $O(V \cdot E)$; the detection pass adds one more scan
- Early exit helps in practice — our specimen stopped after 2 rounds — but the worst case keeps its $O(VE)$
- Edge order never affects *correctness*; it only decides how fast the cascade converges

Greed vs. patience, side by side

Dijkstra: $O\big((V+E)\log V\big)$ — but only for $w \geq 0$.
Bellman–Ford: $O(VE)$ — for any weights, plus a free negative-cycle alarm.
That is the price of negative edges: from $\log V$ per vertex to V full rounds.

New Demand: Everyone Wants Everyone's Price

Pop Quiz

A flight table: $n = 400$ airports, you owe a cheapest fare for *every ordered pair*. Options?

New Demand: Everyone Wants Everyone's Price

Pop Quiz

A flight table: $n = 400$ airports, you owe a cheapest fare for *every ordered pair*. Options?

- $V\times$ Dijkstra: $O\big(V(V+E)\log V\big)$ — fine *if* all fares ≥ 0 … they are not

New Demand: Everyone Wants Everyone's Price

Pop Quiz

A flight table: $n = 400$ airports, you owe a cheapest fare for *every ordered pair*. Options?

- $V\times$ Dijkstra: $O\big(V(V+E)\log V\big)$ — fine *if* all fares ≥ 0 … they are not
- $V\times$ Bellman–Ford: $O(V^2E)$ — correct but heavy

New Demand: Everyone Wants Everyone's Price

Pop Quiz

A flight table: $n = 400$ airports, you owe a cheapest fare for *every ordered pair*. Options?

- $V\times$ Dijkstra: $O\big(V(V+E)\log V\big)$ — fine *if* all fares ≥ 0 … they are not
- $V\times$ Bellman–Ford: $O(V^2E)$ — correct but heavy
- **Floyd–Warshall**: $\Theta(V^3)$ flat, negative edges welcome, and the entire algorithm is *three nested loops*

New Demand: Everyone Wants Everyone's Price

Pop Quiz

A flight table: $n = 400$ airports, you owe a cheapest fare for *every ordered pair*. Options?

- $V\times$ Dijkstra: $O\big(V(V+E)\log V\big)$ — fine *if* all fares ≥ 0 … they are not
- $V\times$ Bellman–Ford: $O(V^2E)$ — correct but heavy
- **Floyd–Warshall**: $\Theta(V^3)$ flat, negative edges welcome, and the entire algorithm is *three nested loops*

A completely different question

Not "how far from s?" but *"fill the whole $n \times n$ distance table"* — with dynamic programming, not queues.

Floyd–Warshall: Ask a DP Question

The state

Number vertices $1, \ldots, n$. Define

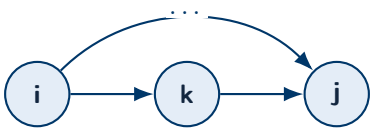
$d_k[i][j]$ = cheapest $i \to j$ route whose *intermediate* vertices all come from $\{1, \ldots, k\}$

Floyd–Warshall: Ask a DP Question

The state

Number vertices $1, \ldots, n$. Define

$d_k[i][j]$ = cheapest $i \to j$ route whose *intermediate* vertices all come from $\{1, \ldots, k\}$

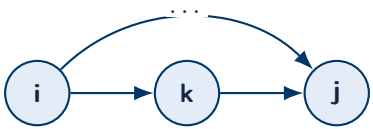


Floyd–Warshall: Ask a DP Question

The state

Number vertices $1, \ldots, n$. Define

$d_k[i][j]$ = cheapest $i \to j$ route whose *intermediate* vertices all come from $\{1, \ldots, k\}$



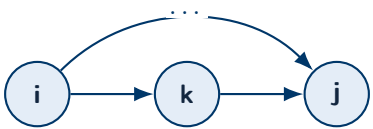
- Doesn't use k: $d_{k-1}[i][j]$

Floyd–Warshall: Ask a DP Question

The state

Number vertices $1, \ldots, n$. Define

$d_k[i][j]$ = cheapest $i \to j$ route whose *intermediate* vertices all come from $\{1, \ldots, k\}$



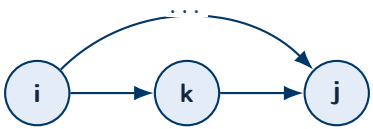
- Doesn't use k: $d_{k-1}[i][j]$
- Uses k: split there — $d_{k-1}[i][k] + d_{k-1}[k][j]$ (both halves stay inside $\{1, \ldots, k-1\}$: k appears exactly once)

Floyd–Warshall: Ask a DP Question

The state

Number vertices $1, \ldots, n$. Define

$d_k[i][j]$ = cheapest $i \to j$ route whose *intermediate* vertices all come from $\{1, \ldots, k\}$



- Doesn't use k: $d_{k-1}[i][j]$
- Uses k: split there — $d_{k-1}[i][k] + d_{k-1}[k][j]$ (both halves stay inside $\{1, \ldots, k-1\}$: k appears exactly once)

The recurrence

$$d_k[i][j] = \min\{\, d_{k-1}[i][j],\ d_{k-1}[i][k] + d_{k-1}[k][j] \,\}$$

Floyd–Warshall — The Pseudocode

FloydWarshall(G, w)

1: $D[i][j] \leftarrow w(i,j)$ for every edge; $D[i][i] \leftarrow 0$; $D[i][j] \leftarrow \infty$ if no edge
2: **for** $k \leftarrow 1$ **to** n **do**
3: **for** $i \leftarrow 1$ **to** n **do**
4: **for** $j \leftarrow 1$ **to** n **do**
5: **if** $D[i][k] + D[k][j] < D[i][j]$ **then**
6: $D[i][j] \leftarrow D[i][k] + D[k][j]$

Floyd–Warshall — The Pseudocode

FloydWarshall(G, w)

1: $D[i][j] \leftarrow w(i,j)$ for every edge; $D[i][i] \leftarrow 0$; $D[i][j] \leftarrow \infty$ if no edge
2: **for** $k \leftarrow 1$ **to** n **do**
3: **for** $i \leftarrow 1$ **to** n **do**
4: **for** $j \leftarrow 1$ **to** n **do**
5: **if** $D[i][k] + D[k][j] < D[i][j]$ **then**
6: $D[i][j] \leftarrow D[i][k] + D[k][j]$

- Yes — that is the entire algorithm. Three loops, one comparison

Floyd–Warshall — The Pseudocode

FloydWarshall(G, w)

1: $D[i][j] \leftarrow w(i,j)$ for every edge; $D[i][i] \leftarrow 0$; $D[i][j] \leftarrow \infty$ if no edge
2: **for** $k \leftarrow 1$ **to** n **do**
3: **for** $i \leftarrow 1$ **to** n **do**
4: **for** $j \leftarrow 1$ **to** n **do**
5: **if** $D[i][k] + D[k][j] < D[i][j]$ **then**
6: $D[i][j] \leftarrow D[i][k] + D[k][j]$

- Yes — that is the entire algorithm. Three loops, one comparison
- In-place is safe: during round k, row k and column k cannot improve (their shortcut would need $D[k][k] \neq 0$)

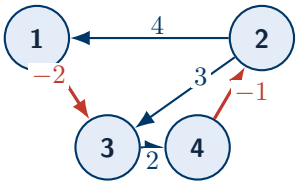
Floyd–Warshall — The Pseudocode

FloydWarshall(G, w)

1: $D[i][j] \leftarrow w(i,j)$ for every edge; $D[i][i] \leftarrow 0$; $D[i][j] \leftarrow \infty$ if no edge
2: **for** $k \leftarrow 1$ **to** n **do**
3: **for** $i \leftarrow 1$ **to** n **do**
4: **for** $j \leftarrow 1$ **to** n **do**
5: **if** $D[i][k] + D[k][j] < D[i][j]$ **then**
6: $D[i][j] \leftarrow D[i][k] + D[k][j]$

- Yes — that is the entire algorithm. Three loops, one comparison
- In-place is safe: during round k, row k and column k cannot improve (their shortcut would need $D[k][k] \neq 0$)
- Watch the diagonal at the end: $D[i][i] < 0 \iff$ negative cycle

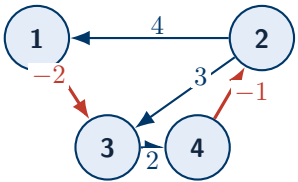
Our Specimen for Floyd–Warshall



$D^{(0)}$ = use *no* intermediates

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

Our Specimen for Floyd–Warshall



$D^{(0)}$ = use *no* intermediates

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

Vitals

$n = 4$, $m = 5$, two red negative edges. Read $D^{(0)}[i][j]$ as "direct edge, or ∞".
Now grow the allowed-intermediate set one vertex per round: $k = 1, 2, 3, 4$.

Walkthrough — $k = 1$ and $k = 2$

after $k = 1$ (may hop via 1)

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & \mathbf{2} & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

one change: $2 \to 3 = 4 + (-2) = 2 < 3$

after $k = 2$ (may also hop via 2)

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ \mathbf{3} & -1 & \mathbf{1} & 0 \end{pmatrix}$$

the -1 edge unlocks row 4:

$4 \to 1 = -1 + 4 = 3, \quad 4 \to 3 = -1 + 2 = 1$

Walkthrough — $k = 1$ and $k = 2$

after $k = 1$ (may hop via 1)

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & \mathbf{2} & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

one change: $2 \to 3 = 4 + (-2) = 2 < 3$

after $k = 2$ (may also hop via 2)

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ \mathbf{3} & -1 & \mathbf{1} & 0 \end{pmatrix}$$

the -1 edge unlocks row 4:

$4 \to 1 = -1 + 4 = 3, \quad 4 \to 3 = -1 + 2 = 1$

- $k = 1$: vertex 1 only *leaves* via -2, so it helps exactly one entry

Walkthrough — $k = 1$ and $k = 2$

after $k = 1$ (may hop via 1)

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & \mathbf{2} & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

one change: $2 \to 3 = 4 + (-2) = 2 < 3$

after $k = 2$ (may also hop via 2)

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ \mathbf{3} & -1 & \mathbf{1} & 0 \end{pmatrix}$$

the -1 edge unlocks row 4:

$4 \to 1 = -1 + 4 = 3, \quad 4 \to 3 = -1 + 2 = 1$

- $k = 1$: vertex 1 only *leaves* via -2, so it helps exactly one entry
- $k = 2$: a single negative edge into 2 makes vertex 2 a hub — row 4 wakes up all at once

Walkthrough — $k = 3$ and $k = 4$: The Final Table

after $k = 3$ (may also hop via 3)

$$\begin{pmatrix} 0 & \infty & -2 & \mathbf{0} \\ 4 & 0 & 2 & \mathbf{4} \\ \infty & \infty & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

$1 \to 4 = -2 + 2 = 0$, $2 \to 4 = 2 + 2 = 4$ (both via 3)

after $k = 4$ (final)

$$\begin{pmatrix} 0 & \mathbf{-1} & -2 & 0 \\ 4 & 0 & 2 & 4 \\ \mathbf{5} & \mathbf{1} & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

row 4 now feeds everyone: $1 \to 2 = 0 + (-1)$, $3 \to 1 = 2 + 3$, $3 \to 2 = 2 + (-1)$

Walkthrough — $k = 3$ and $k = 4$: The Final Table

after $k = 3$ (may also hop via 3)

$$\begin{pmatrix} 0 & \infty & -2 & \mathbf{0} \\ 4 & 0 & 2 & \mathbf{4} \\ \infty & \infty & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

$1 \to 4 = -2 + 2 = 0, \quad 2 \to 4 = 2 + 2 = 4$ (both via 3)

after $k = 4$ (final)

$$\begin{pmatrix} 0 & \mathbf{-1} & -2 & 0 \\ 4 & 0 & 2 & 4 \\ \mathbf{5} & \mathbf{1} & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

row 4 now feeds everyone: $1 \to 2 = 0 + (-1)$, $3 \to 1 = 2 + 3, \quad 3 \to 2 = 2 + (-1)$

Verify & diagnose

$\delta(1, 2) = -1$ via $1 \to 3 \to 4 \to 2 = -2 + 2 - 1$ ✓; diagonal $= 0$ everywhere $\Rightarrow$ no negative cycle.

Floyd–Warshall: The Receipt

- n rounds $\times$ n^2 cells $\times$ $O(1)$ work $= \Theta(V^3)$ time, $\Theta(V^2)$ space — *regardless* of how many edges exist

Floyd–Warshall: The Receipt

- n rounds $\times$ n^2 cells $\times$ $O(1)$ work $= \Theta(V^3)$ time, $\Theta(V^2)$ space — *regardless* of how many edges exist
- Negative cycle? The diagonal betrays it: $D[i][i] < 0$ after the run

Floyd–Warshall: The Receipt

- n rounds $\times$ n^2 cells $\times$ $O(1)$ work $= \Theta(V^3)$ time, $\Theta(V^2)$ space — *regardless* of how many edges exist
- Negative cycle? The diagonal betrays it: $D[i][i] < 0$ after the run
- Dense graphs ($E \approx V^2$): $V\times$ heap-Dijkstra costs $O(V^3 \log V)$ — Floyd–Warshall even wins (no log)

Floyd–Warshall: The Receipt

- n rounds $\times$ n^2 cells $\times$ $O(1)$ work $= \Theta(V^3)$ time, $\Theta(V^2)$ space — *regardless* of how many edges exist
- Negative cycle? The diagonal betrays it: $D[i][i] < 0$ after the run
- Dense graphs ($E \approx V^2$): $V\times$ heap-Dijkstra costs $O(V^3 \log V)$ — Floyd–Warshall even wins (no log)

But on sparse graphs ...

$V = 10^4$, $E = 3 \times 10^4$: FW does $(10^4)^3 = 10^{12}$ steps for a table with only 10^8 entries. Most of those steps recompute ∞s. Johnson to the rescue.

Johnson's Question: Sparse, Negative, *All* Pairs

The deadlock

$V = 10^4$ airports, $E = 3 \times 10^4$ flights, some fares *negative*, and you owe the full table. Floyd–Warshall: 10^{12} steps. $V\times$ Bellman–Ford: $O(V^2E)$, worse. Dijkstra: *refuses* negative weights.

Johnson's Question: Sparse, Negative, *All* Pairs

The deadlock

$V = 10^4$ airports, $E = 3 \times 10^4$ flights, some fares *negative*, and you owe the full table. Floyd–Warshall: 10^{12} steps. $V\times$ Bellman–Ford: $O(V^2E)$, worse. Dijkstra: *refuses* negative weights.

The idea: change the weights, not the answers

Find a *potential* $h(v)$ for every vertex and replace each edge by

$$w'(u,v) \;=\; w(u,v) + h(u) - h(v)$$

so that every $w' \geq 0$ — then $V\times$ Dijkstra becomes legal.

Johnson's Question: Sparse, Negative, *All* Pairs

The deadlock

$V = 10^4$ airports, $E = 3 \times 10^4$ flights, some fares *negative*, and you owe the full table. Floyd–Warshall: 10^{12} steps. $V\times$ Bellman–Ford: $O(V^2E)$, worse. Dijkstra: *refuses* negative weights.

The idea: change the weights, not the answers

Find a *potential* $h(v)$ for every vertex and replace each edge by

$$w'(u,v) \;=\; w(u,v) + h(u) - h(v)$$

so that every $w' \geq 0$ — then $V\times$ Dijkstra becomes legal.

- Two demands: (1) all $w' \geq 0$; (2) every pair's cheapest route *unchanged*. What h manages both?

Johnson: Two Questions, Two Answers

Q1 — when is $w'(u,v) = w(u,v) + h(u) - h(v) \geq 0$?

Johnson: Two Questions, Two Answers

Q1 — when is $w'(u,v) = w(u,v) + h(u) - h(v) \geq 0$?

- Exactly when $h(v) \leq h(u) + w(u,v)$ — and *shortest-path distances* satisfy precisely this! So: add a new vertex q with 0-weight edges $q \to v$ for all v, run **Bellman–Ford once** from q, set $h(v) \leftarrow \delta(q,v)$

Johnson: Two Questions, Two Answers

Q1 — when is $w'(u,v) = w(u,v) + h(u) - h(v) \geq 0$?

- Exactly when $h(v) \leq h(u) + w(u,v)$ — and *shortest-path distances* satisfy precisely this! So: add a new vertex q with 0-weight edges $q \to v$ for all v, run **Bellman–Ford once** from q, set $h(v) \leftarrow \delta(q,v)$

Q2 — why doesn't moving the weights move the answer?

Johnson: Two Questions, Two Answers

Q1 — when is $w'(u,v) = w(u,v) + h(u) - h(v) \geq 0$?

- Exactly when $h(v) \leq h(u) + w(u,v)$ — and *shortest-path distances* satisfy precisely this! So: add a new vertex q with 0-weight edges $q \to v$ for all v, run **Bellman–Ford once** from q, set $h(v) \leftarrow \delta(q,v)$

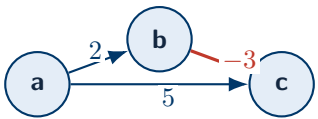
Q2 — why doesn't moving the weights move the answer?

- Telescoping: for any path $P : u \rightsquigarrow v$,

$$\text{cost}'(P) = \sum_{(u_i,v_i)\in P} \big(w(u_i,v_i) + h(u_i) - h(v_i)\big) = \text{cost}(P) + h(u) - h(v)$$

- Internal $\pm h$ cancel; the correction $h(u) - h(v)$ is the same constant for every $u \to v$ path — same winner, every time. Recover: $\delta(u,v) = \delta'(u,v) - h(u) + h(v)$

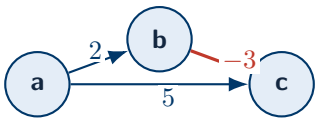
Reweighting by Hand — One Small Graph



Bellman–Ford from the invisible q (0-edges to all):

$h(a) = 0$	(only $q \to a$)
$h(b) = \min(0,\ 2) = 0$	($q \to b$ beats $q \to a \to b$)
$h(c) = \min(0,\ 5,\ -1,\ -3) = -3$	($q \to b \to c$)

Reweighting by Hand — One Small Graph



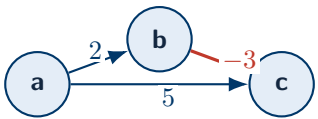
Bellman–Ford from the invisible q (0-edges to all):

$h(a) = 0$ (only $q \to a$)
$h(b) = \min(0,\ 2) = 0$ ($q \to b$ beats $q \to a \to b$)
$h(c) = \min(0,\ 5,\ -1,\ -3) = -3$ ($q \to b \to c$)

New weights: $w' = w + h(u) - h(v)$

$a \to b:\ 2 + 0 - 0 = 2$; $b \to c:\ -3 + 0 - (-3) = 0$; $a \to c:\ 5 + 0 - (-3) = 8$ — all ≥ 0

Reweighting by Hand — One Small Graph



Bellman–Ford from the invisible q (0-edges to all):

$h(a) = 0$ (only $q \to a$)
$h(b) = \min(0,\ 2) = 0$ ($q \to b$ beats $q \to a \to b$)
$h(c) = \min(0,\ 5,\ -1,\ -3) = -3$ ($q \to b \to c$)

New weights: $w' = w + h(u) - h(v)$

$a \to b:\ 2 + 0 - 0 = 2$; $b \to c:\ -3 + 0 - (-3) = 0$; $a \to c:\ 5 + 0 - (-3) = 8$ — all ≥ 0

Check the answer survived

Cheapest $a \to c$: was $\min(5,\ 2 - 3) = -1$; now $\min(8,\ 2 + 0) = 2$. Recover:
$2 - h(a) + h(c) = 2 - 0 - 3 = -1$ ✓

Johnson — The Pseudocode and the Receipt

Johnson(G)

1: add vertex q; edges $q \to v$ of weight 0, for all v
2: $h \leftarrow$ **BellmanFord**$(G \cup \{q\},\ q)$
3: **if** it reports a negative cycle **then**
4: **return** *report it*
5: **for** each edge $(u, v) \in E$ **do**
6: $w'(u, v) \leftarrow w(u, v) + h[u] - h[v]$ ▷ all ≥ 0
7: **for** each $u \in V$ **do**
8: $\hat{d}[u][\cdot] \leftarrow$ **Dijkstra**(G, w', u)
9: **return** $\hat{d}[u][v] - h[u] + h[v]$ for all pairs

Johnson — The Pseudocode and the Receipt

Johnson(G)
1: add vertex q; edges $q \to v$ of weight 0, for all v
2: $h \leftarrow$ **BellmanFord**$(G \cup \{q\},\ q)$
3: **if** it reports a negative cycle **then**
4: **return** *report it*
5: **for** each edge $(u, v) \in E$ **do**
6: $w'(u, v) \leftarrow w(u, v) + h[u] - h[v]$ ▷ all ≥ 0
7: **for** each $u \in V$ **do**
8: $\hat{d}[u][\cdot] \leftarrow$ **Dijkstra**(G, w', u)
9: **return** $\hat{d}[u][v] - h[u] + h[v]$ for all pairs

- Cost: one $O(VE)$ pass + V heap-Dijkstras $= O\big(VE + VE \log V\big)$; V=400, E=1200: $\approx 4.6 \times 10^6$ vs. FW's 6.4×10^7 — $14\times$ less work

Johnson — The Pseudocode and the Receipt

Johnson(G)
1: add vertex q; edges $q \to v$ of weight 0, for all v
2: $h \leftarrow$ **BellmanFord**$(G \cup \{q\},\ q)$
3: **if** it reports a negative cycle **then**
4: **return** *report it*
5: **for** each edge $(u, v) \in E$ **do**
6: $w'(u, v) \leftarrow w(u, v) + h[u] - h[v]$ ▷ all ≥ 0
7: **for** each $u \in V$ **do**
8: $\hat{d}[u][\cdot] \leftarrow$ **Dijkstra**(G, w', u)
9: **return** $\hat{d}[u][v] - h[u] + h[v]$ for all pairs

- Cost: one $O(VE)$ pass + V heap-Dijkstras $= O\big(VE + VE \log V\big)$; $V{=}400$, $E{=}1200$: $\approx 4.6 \times 10^6$ vs. FW's 6.4×10^7 — $14\times$ less work
- All weights already ≥ 0? Then $h \equiv 0$, $w' = w$ — Johnson quietly becomes $V\times$ Dijkstra. One wrapper, no special cases.

The Big Comparison — Four Algorithms, One Table

	Dijkstra	Bellman–Ford	Floyd–Warshall	Johnson
solves	SSSP	SSSP	APSP	APSP
time	$O\big((V+E)\log V\big)$	$O(VE)$	$\Theta(V^3)$	$O(VE\log V)$
negative edges	no	yes	yes	yes (erases them)
negative cycle	undefined	detects	$D[i][i] < 0$	detects (BF)
hire it when	$w \geq 0$, one source	neg. edges, one source	dense, all pairs	sparse, all pairs

The Big Comparison — Four Algorithms, One Table

	Dijkstra	Bellman–Ford	Floyd–Warshall	Johnson
solves	SSSP	SSSP	APSP	APSP
time	$O\big((V+E)\log V\big)$	$O(VE)$	$\Theta(V^3)$	$O(VE\log V)$
negative edges	no	yes	yes	yes (erases them)
negative cycle	undefined	detects	$D[i][i] < 0$	detects (BF)
hire it when	$w \geq 0$, one source	neg. edges, one source	dense, all pairs	sparse, all pairs

Verdict

Same Relax move everywhere — the *schedule* and the *weights* pick the algorithm, never fashion.

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city?

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city? **Dijkstra** (heap): sparse + $w \geq 0$, one source.

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city? **Dijkstra** (heap): sparse + $w \geq 0$, one source.

2 — Same map, 3 refund fares went negative

Still one hub, still every city.

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city? **Dijkstra** (heap): sparse + $w \geq 0$, one source.

2 — Same map, 3 refund fares went negative

Still one hub, still every city. **Bellman–Ford**: negative edges present, one source, $O(VE)$ is fine.

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city? **Dijkstra** (heap): sparse + $w \geq 0$, one source.

2 — Same map, 3 refund fares went negative

Still one hub, still every city. **Bellman–Ford**: negative edges present, one source, $O(VE)$ is fine.

3 — 400 bus stops, fare table for *all pairs*, some negative

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city? **Dijkstra** (heap): sparse + $w \geq 0$, one source.

2 — Same map, 3 refund fares went negative

Still one hub, still every city. **Bellman–Ford**: negative edges present, one source, $O(VE)$ is fine.

3 — 400 bus stops, fare table for *all pairs*, some negative

Floyd–Warshall: $400^3 = 6.4 \times 10^7$ — a heartbeat, zero fuss.

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city? **Dijkstra** (heap): sparse + $w \geq 0$, one source.

2 — Same map, 3 refund fares went negative

Still one hub, still every city. **Bellman–Ford**: negative edges present, one source, $O(VE)$ is fine.

3 — 400 bus stops, fare table for *all pairs*, some negative

Floyd–Warshall: $400^3 = 6.4 \times 10^7$ — a heartbeat, zero fuss.

4 — Same stops but sparse: $1{,}200$ links, all pairs

Pop Quiz: Hire the Right Algorithm

1 — Road map of Hormozgan

$5{,}000$ junctions, $12{,}000$ roads, all times positive. Cheapest routes *from Bandar Abbas* to every city? **Dijkstra** (heap): sparse + $w \geq 0$, one source.

2 — Same map, 3 refund fares went negative

Still one hub, still every city. **Bellman–Ford**: negative edges present, one source, $O(VE)$ is fine.

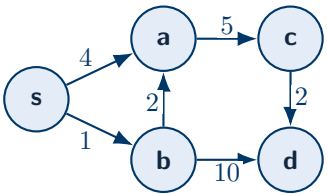
3 — 400 bus stops, fare table for *all pairs*, some negative

Floyd–Warshall: $400^3 = 6.4 \times 10^7$ — a heartbeat, zero fuss.

4 — Same stops but sparse: $1{,}200$ links, all pairs

Johnson: $VE + VE \log V \approx 4.6 \times 10^6$ — $14\times$ cheaper than FW here.

Try It Yourself #1 — Dijkstra by Hand

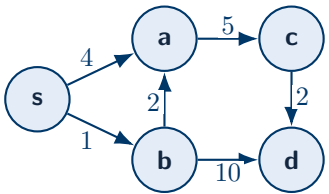


Your turn

Run **Dijkstra** from s (min first!).

Report: settle order, final d, and the π tree.

Try It Yourself #1 — Dijkstra by Hand



Your turn

Run **Dijkstra** from s (min first!).

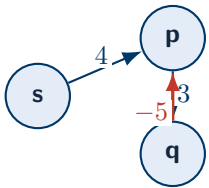
Report: settle order, final d, and the π tree.

Answer

settle s, $b(1)$, $a(3)$, $c(8)$, $d(10)$; π: $a \leftarrow b$, $c \leftarrow a$, $d \leftarrow c$.

Watch d: b improves a $(4 \to 3)$, and d improves $11 \to 10$ only when c settles — patience *inside* greed.

Try It Yourself #2 — Bellman–Ford by Hand



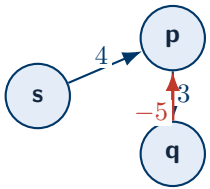
Your turn

Scan edges in order $[s \to p,\ p \to q,\ q \to p]$.

Run $V - 1 = 2$ rounds from s, then the detection pass.

What does Bellman–Ford announce — and what are the true distances?

Try It Yourself #2 — Bellman–Ford by Hand



Your turn

Scan edges in order $[s \to p,\ p \to q,\ q \to p]$.

Run $V - 1 = 2$ rounds from s, then the detection pass.

What does Bellman–Ford announce — and what are the true distances?

Answer

Round 1: $p = 2$ (the $q \to p$ edge fires late in the round), $q = 7$. Round 2: $p = 0$, $q = 5$. Pass 3 still improves $\Rightarrow$ negative cycle $p \to q \to p = -2$; $\delta(s, p) = \delta(s, q) = -\infty$: spiral $p: 2, 0, -2, \ldots$

Concept Corner

Q1

Which single assumption lets Dijkstra *never reopen* a settled vertex?

Concept Corner

Q1

Which single assumption lets Dijkstra *never reopen* a settled vertex?

$w \geq 0$. Any later path to a settled u must enter through the frontier where $d \geq d[u]$, and non-negative edges only add. (Recall b frozen at 1 instead of -1.)

Concept Corner

Q1

Which single assumption lets Dijkstra *never reopen* a settled vertex?

$w \geq 0$**.** Any later path to a settled u must enter through the frontier where $d \geq d[u]$, and non-negative edges only add. (Recall b frozen at 1 instead of -1.)

Q2

In Floyd–Warshall's final matrix, which entries expose a negative cycle?

Concept Corner

Q1

Which single assumption lets Dijkstra *never reopen* a settled vertex?

$w \geq 0$**.** Any later path to a settled u must enter through the frontier where $d \geq d[u]$, and non-negative edges only add. (Recall b frozen at 1 instead of -1.)

Q2

In Floyd–Warshall's final matrix, which entries expose a negative cycle?

The diagonal: $D[i][i] < 0$ means a cheap round trip from i back to itself exists.

Concept Corner

Q1

Which single assumption lets Dijkstra *never reopen* a settled vertex?

$w \geq 0$**.** Any later path to a settled u must enter through the frontier where $d \geq d[u]$, and non-negative edges only add. (Recall b frozen at 1 instead of -1.)

Q2

In Floyd–Warshall's final matrix, which entries expose a negative cycle?

The diagonal: $D[i][i] < 0$ means a cheap round trip from i back to itself exists.

Q3

Johnson runs on an all-positive graph. Is its Bellman–Ford phase wasted?

Concept Corner

Q1

Which single assumption lets Dijkstra *never reopen* a settled vertex?

$w \geq 0$**.** Any later path to a settled u must enter through the frontier where $d \geq d[u]$, and non-negative edges only add. (Recall b frozen at 1 instead of -1.)

Q2

In Floyd–Warshall's final matrix, which entries expose a negative cycle?

The diagonal: $D[i][i] < 0$ means a cheap round trip from i back to itself exists.

Q3

Johnson runs on an all-positive graph. Is its Bellman–Ford phase wasted?

Almost: $h(v) = 0$ for all v, so $w' = w$ — the reweighting is the identity and Johnson quietly degenerates into $V\times$ Dijkstra. No harm, no special case.

Summary — What You Own Now

One move

Relax: $d[v] \leftarrow \min\big(d[v],\, d[u] + w\big)$ — today's four algorithms are four *schedules* of this one move.

One source

Dijkstra $O\big((V{+}E)\log V\big)$ for $w \geq 0$ · **Bellman–Ford** $O(VE)$ for any weights + free negative-cycle alarm.

Everyone to everyone

Floyd–Warshall $\Theta(V^3)$: three loops, negatives welcome, diagonal betrays cycles · **Johnson**: reweight once with h, then V Dijkstras — the sparse champion.

Summary — What You Own Now

One move

Relax: $d[v] \leftarrow \min\big(d[v],\, d[u] + w\big)$ — today's four algorithms are four *schedules* of this one move.

One source

Dijkstra $O\big((V{+}E)\log V\big)$ for $w \geq 0$ · **Bellman–Ford** $O(VE)$ for any weights + free negative-cycle alarm.

Everyone to everyone

Floyd–Warshall $\Theta(V^3)$: three loops, negatives welcome, diagonal betrays cycles · **Johnson**: reweight once with h, then V Dijkstras — the sparse champion.

Next session

Minimum Spanning Trees — **Kruskal & Prim**: same weighted graphs, a cheaper promise: connect *everyone*, pay as little cable as possible.