
http://algs4.cs.princeton.edu

# 4.4 Shortest Paths

- APIs
- shortest-paths properties
- Dijkstra's algorithm
- *edge-weighted DAGs*
- negative weights

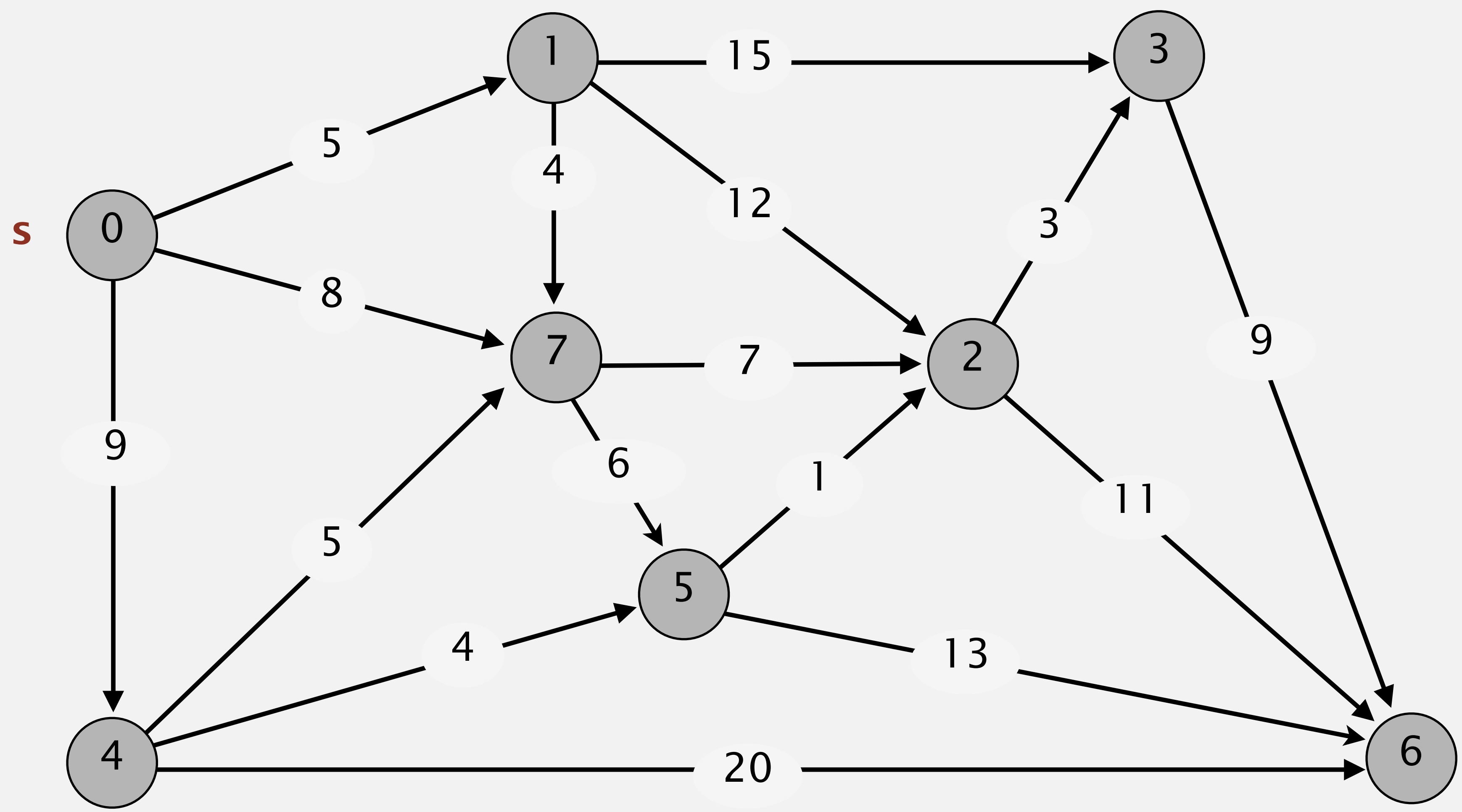
# Acyclic edge-weighted digraphs

Q. Suppose that an edge-weighted digraph has no directed cycles.
Is it easier to find shortest paths than in a general digraph?

A. Yes!

# Acyclic shortest paths demo

- Consider vertices in topological order.
- Relax all edges pointing from that vertex.

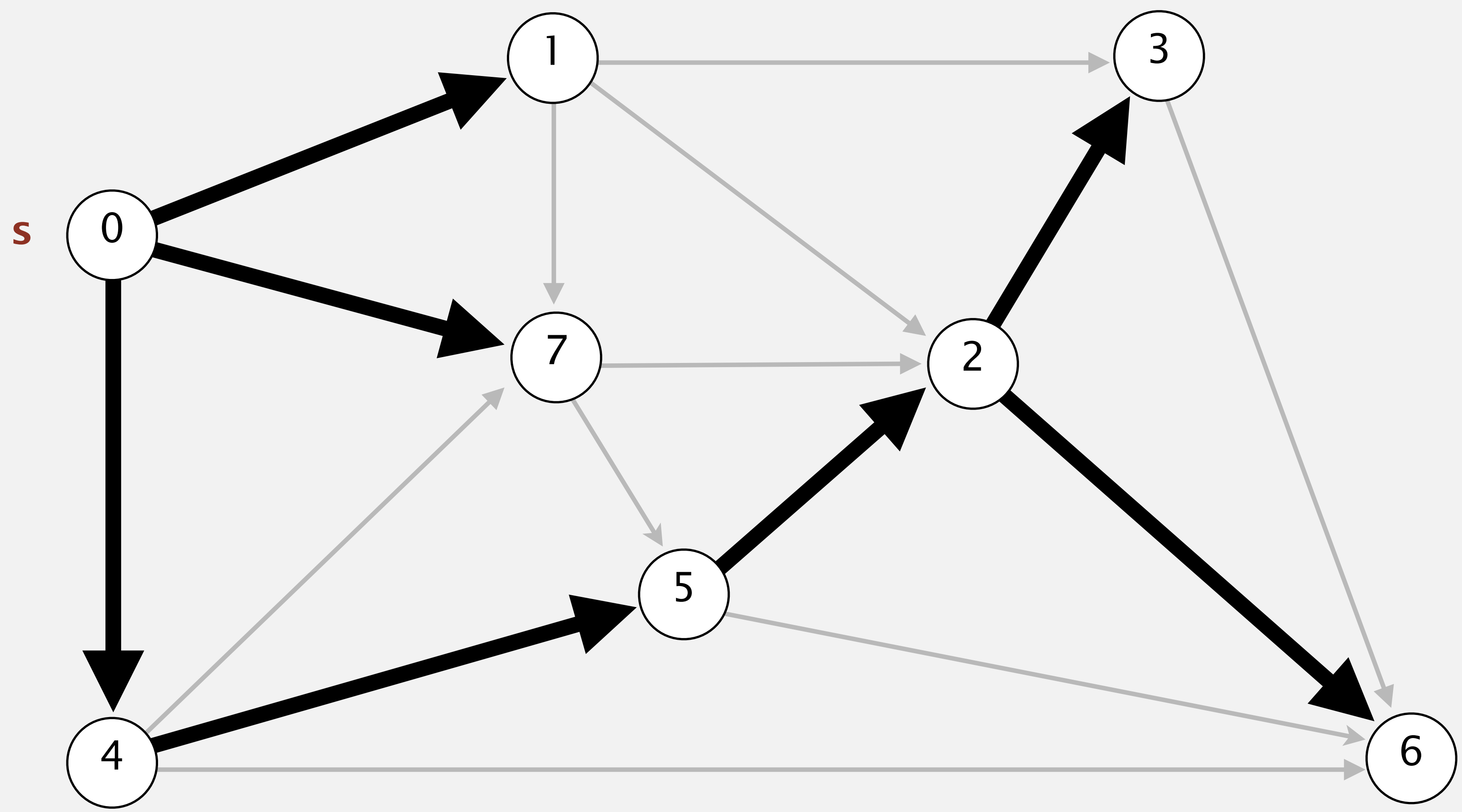



an edge-weighted DAG

| | |
|---|---|
| 0→1 | 5.0 |
| 0→4 | 9.0 |
| 0→7 | 8.0 |
| 1→2 | 12.0 |
| 1→3 | 15.0 |
| 1→7 | 4.0 |
| 2→3 | 3.0 |
| 2→6 | 11.0 |
| 3→6 | 9.0 |
| 4→5 | 4.0 |
| 4→6 | 20.0 |
| 4→7 | 5.0 |
| 5→2 | 1.0 |
| 5→6 | 13.0 |
| 7→5 | 6.0 |
| 7→2 | 7.0 |

# Acyclic shortest paths demo

- Consider vertices in topological order.
- Relax all edges pointing from that vertex.




shortest-paths tree from vertex s

**0 1 4 7 5 2 3 6**

| v | distTo[] | edgeTo[] |
|---|---|---|
| 0 | 0.0 | - |
| 1 | 5.0 | 0→1 |
| 2 | 14.0 | 5→2 |
| 3 | 17.0 | 2→3 |
| 4 | 9.0 | 0→4 |
| 5 | 13.0 | 4→5 |
| 6 | 25.0 | 2→6 |
| 7 | 8.0 | 0→7 |

# Shortest paths in edge-weighted DAGs

Proposition. Topological sort algorithm computes SPT in any edge-weighted DAG in time proportional to $E + V$.

edge weights
can be negative!

Pf.

- Each edge `e = v→w` is relaxed exactly once (when `v` is relaxed), leaving `distTo[w] ≤ distTo[v] + e.weight()`.
- Inequality holds until algorithm terminates because:
  - `distTo[w]` cannot increase ← `distTo[]` values are monotone decreasing
  - `distTo[v]` will not change ← because of topological order, no edge pointing to v will be relaxed after v is relaxed
- Thus, upon termination, shortest-paths optimality conditions hold. ▪

# Shortest paths in edge-weighted DAGs

```
public class AcyclicSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;

   public AcyclicSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];

      for (int v = 0; v < G.V(); v++)
         distTo[v] = Double.POSITIVE_INFINITY;
      distTo[s] = 0.0;

      Topological topological = new Topological(G);
      for (int v : topological.order())
         for (DirectedEdge e : G.adj(v))
            relax(e);
   }
}
```

topological order

# Content-aware resizing

Seam carving. [Avidan and Shamir] Resize an image without distortion for display on cell phones and web browsers.


http://www.youtube.com/watch?v=vIFCV2spKtg

# Content-aware resizing

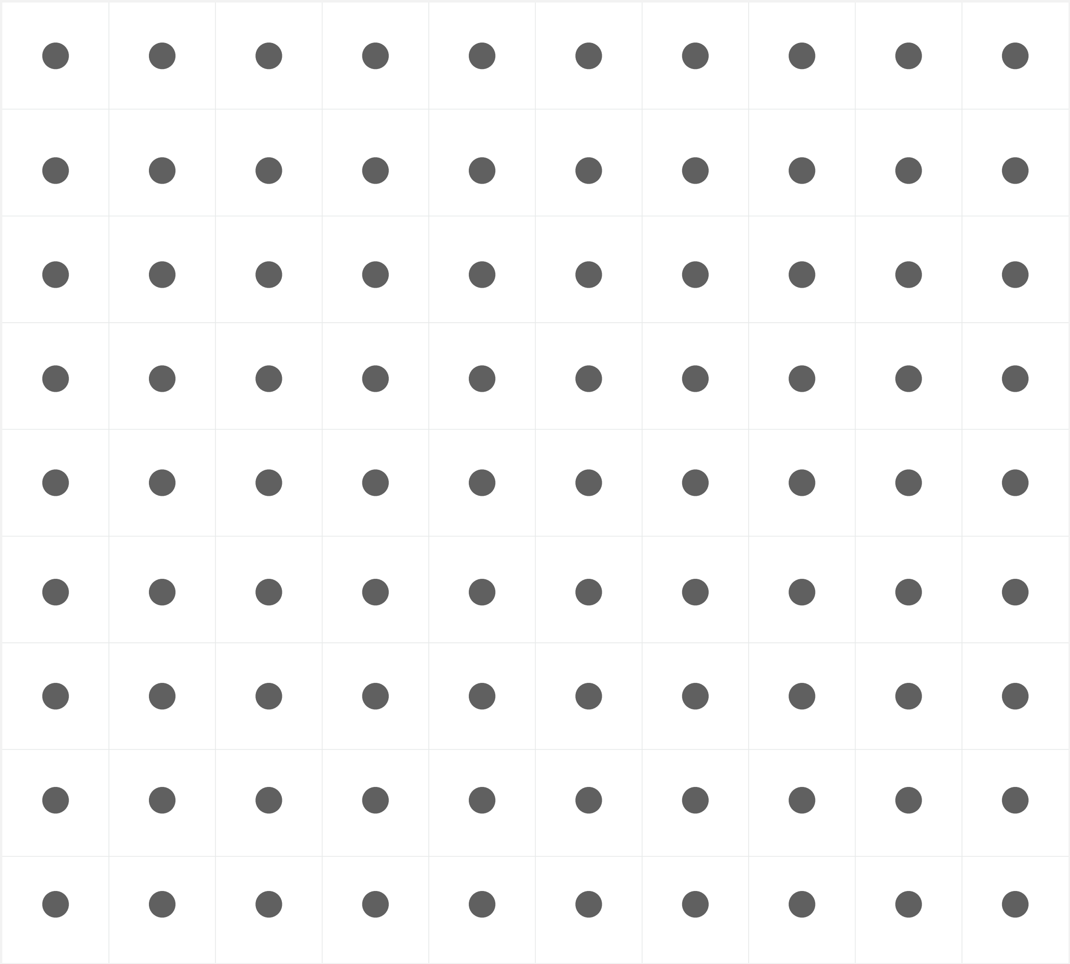
Seam carving. [Avidan and Shamir] Resize an image without distortion for display on cell phones and web browsers.

In the wild. Photoshop CS 5, Imagemagick, GIMP, ...

# Content-aware resizing

To find vertical seam:

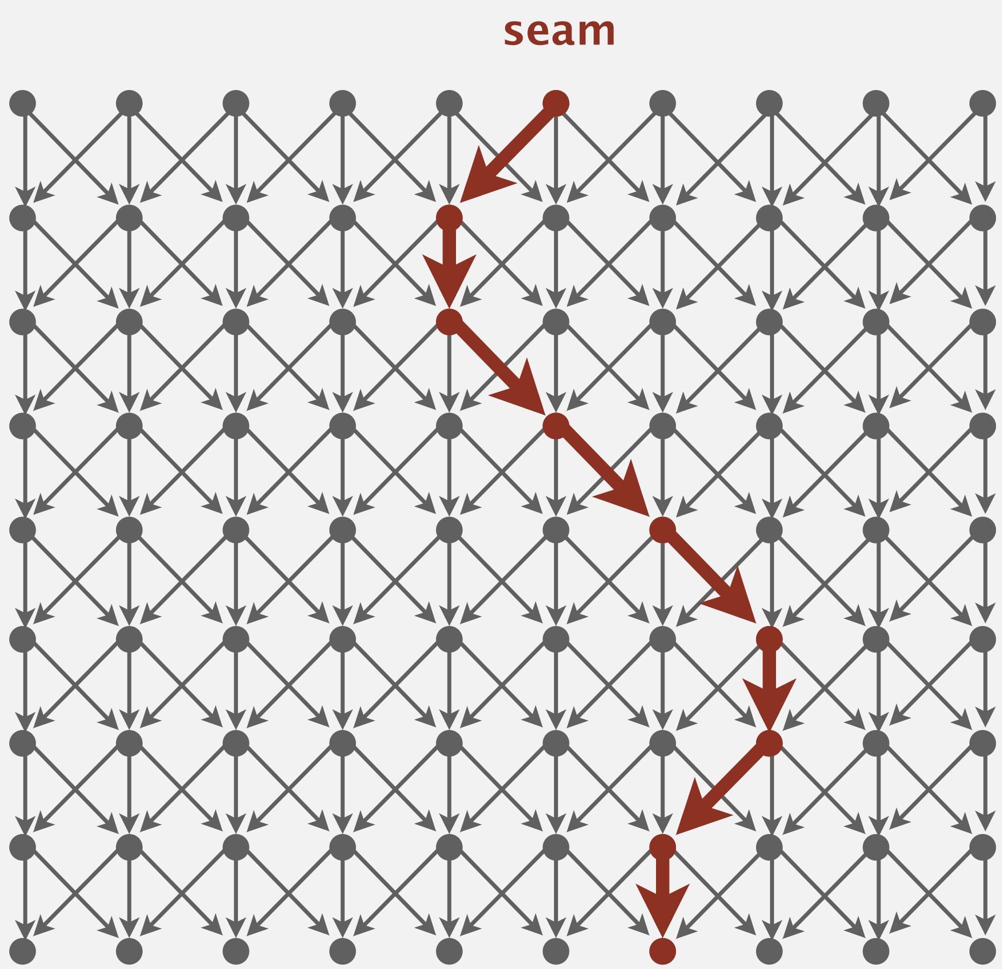
- Grid DAG: vertex = pixel; edge = from pixel to 3 downward neighbors.
- Weight of pixel = energy function of 8 neighboring pixels.
- Seam = shortest path (sum of vertex weights) from top to bottom.



# Content-aware resizing

To find vertical seam:

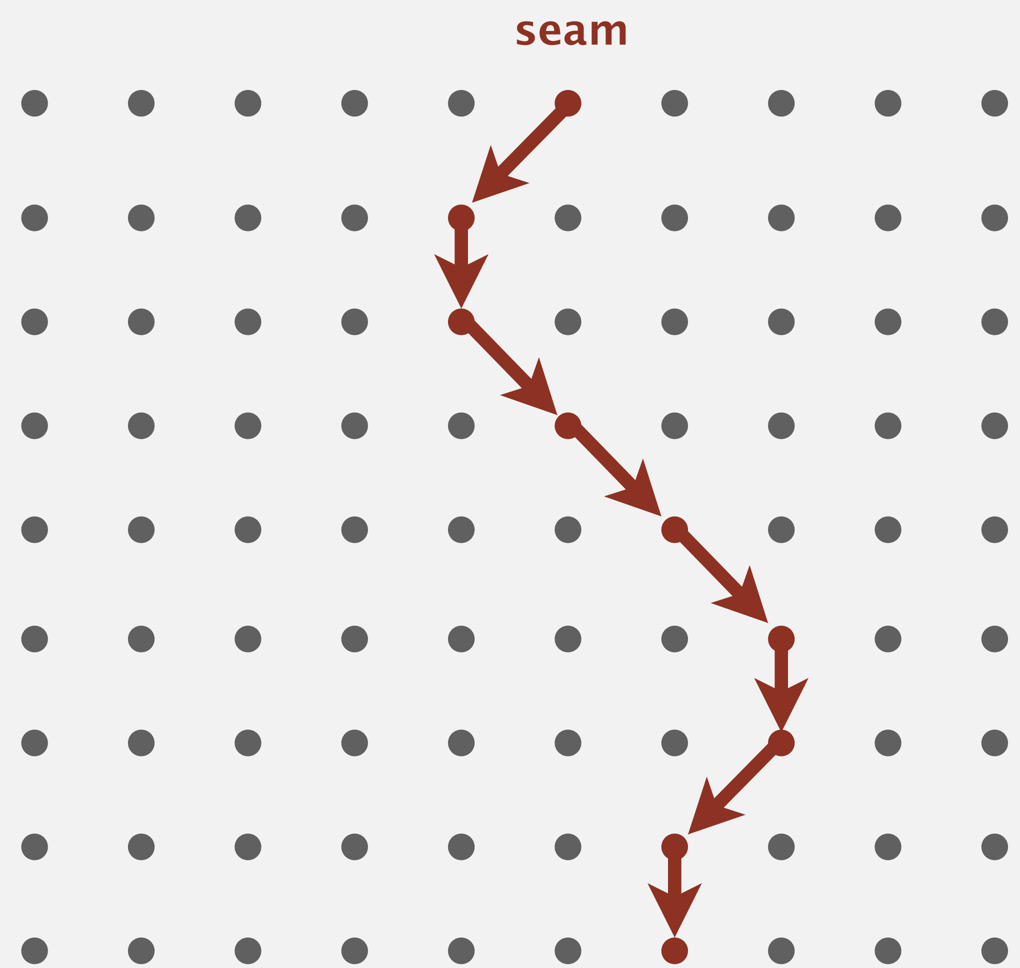
- Grid DAG: vertex = pixel; edge = from pixel to 3 downward neighbors.
- Weight of pixel = energy function of 8 neighboring pixels.
- Seam = shortest path (sum of vertex weights) from top to bottom.



# Content-aware resizing

To remove vertical seam:

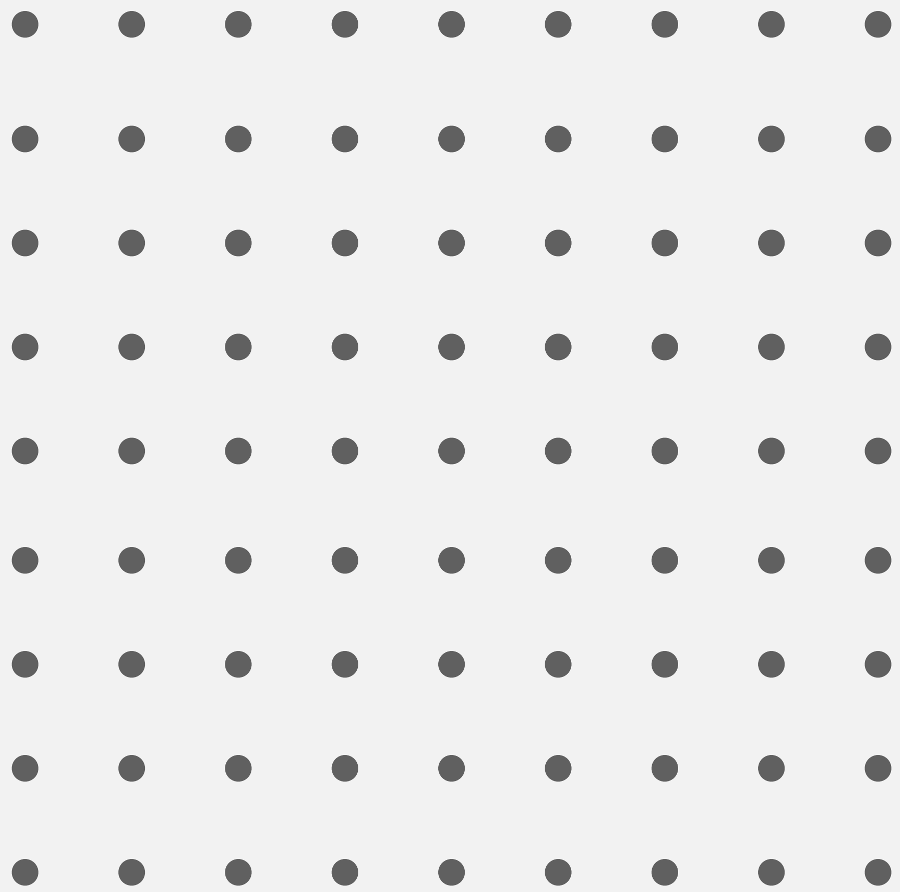
- Delete pixels on seam (one in each row).



# Content-aware resizing

To remove vertical seam:

- Delete pixels on seam (one in each row).



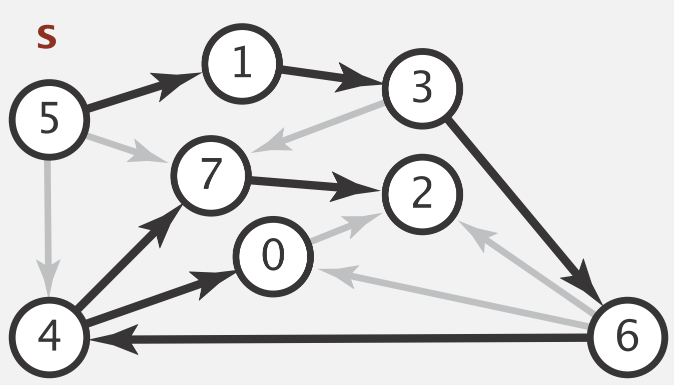
# Longest paths in edge-weighted DAGs

Formulate as a shortest paths problem in edge-weighted DAGs.

- Negate all weights.
- Find shortest paths.
- Negate weights in result.

equivalent: reverse sense of equality in `relax()`

| longest paths input | | shortest paths input | |
|---|---|---|---|
| 5->4 | 0.35 | 5->4 | -0.35 |
| 4->7 | 0.37 | 4->7 | -0.37 |
| 5->7 | 0.28 | 5->7 | -0.28 |
| 5->1 | 0.32 | 5->1 | -0.32 |
| 4->0 | 0.38 | 4->0 | -0.38 |
| 0->2 | 0.26 | 0->2 | -0.26 |
| 3->7 | 0.39 | 3->7 | -0.39 |
| 1->3 | 0.29 | 1->3 | -0.29 |
| 7->2 | 0.34 | 7->2 | -0.34 |
| 6->2 | 0.40 | 6->2 | -0.40 |
| 3->6 | 0.52 | 3->6 | -0.52 |
| 6->0 | 0.58 | 6->0 | -0.58 |
| 6->4 | 0.93 | 6->4 | -0.93 |

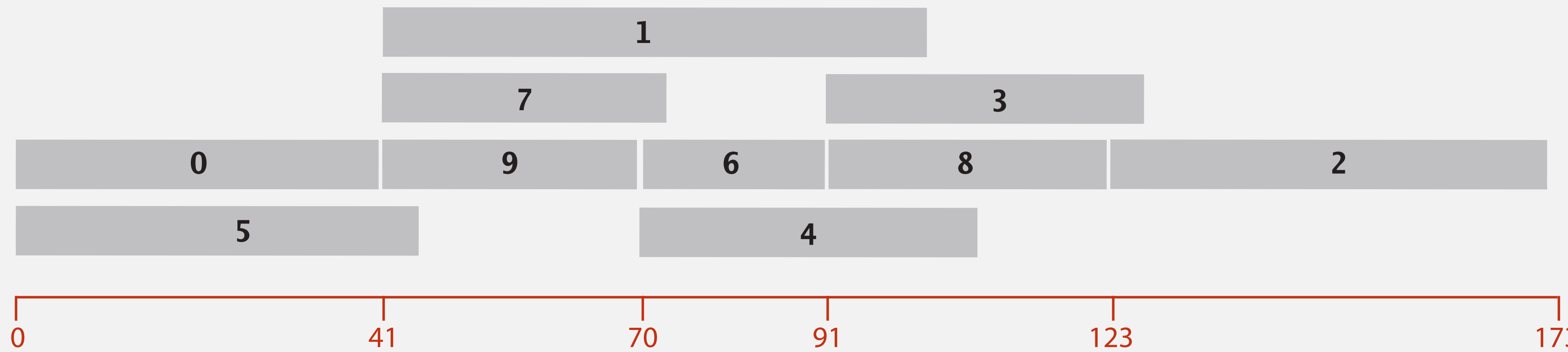



Key point. Topological sort algorithm works even with negative weights.

# Longest paths in edge-weighted DAGs: application

Parallel job scheduling. Given a set of jobs with durations and precedence constraints, schedule the jobs (by finding a start time for each) so as to achieve the minimum completion time, while respecting the constraints.

| *job* | *duration* | *must complete before* | | |
|---|---|---|---|---|
| 0 | 41.0 | 1 | 7 | 9 |
| 1 | 51.0 | 2 | | |
| 2 | 50.0 | | | |
| 3 | 36.0 | | | |
| 4 | 38.0 | | | |
| 5 | 45.0 | | | |
| 6 | 21.0 | 3 | 8 | |
| 7 | 32.0 | 3 | 8 | |
| 8 | 32.0 | 2 | | |
| 9 | 29.0 | 4 | 6 | |



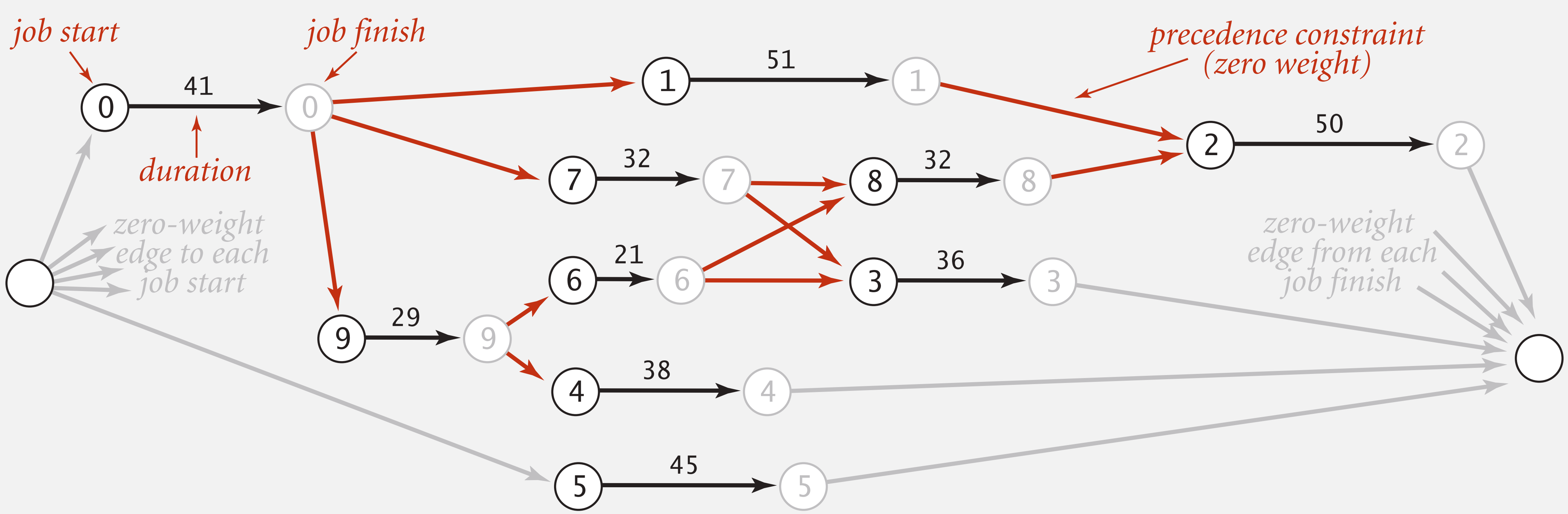
Parallel job scheduling solution

# Critical path method

CPM. To solve a parallel job-scheduling problem, create edge-weighted DAG:

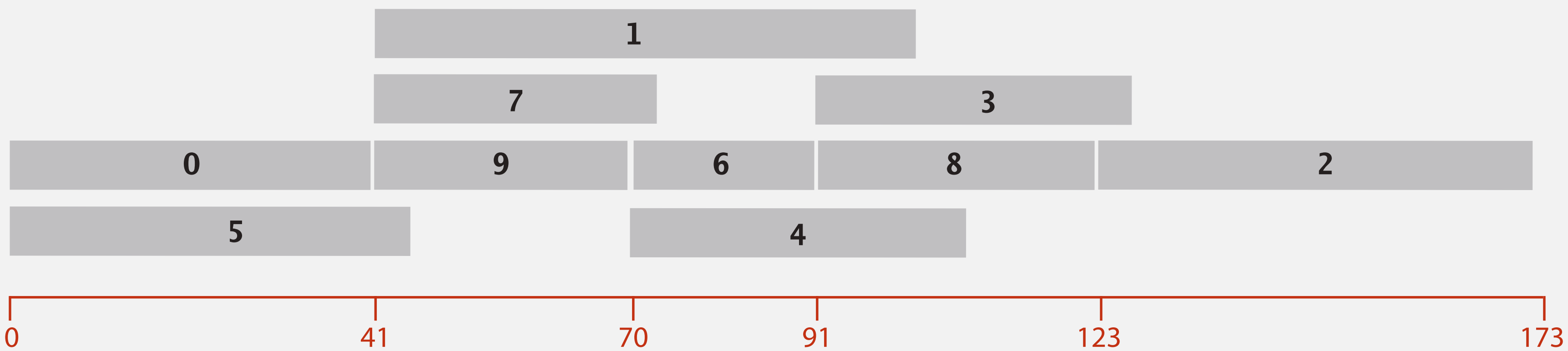
- Source and sink vertices.
- Two vertices (begin and end) for each job.
- Three edges for each job.
  - begin to end (weighted by duration)
  - source to begin (0 weight)
  - end to sink (0 weight)
- One edge for each precedence constraint (0 weight).

| *job* | *duration* | *must complete before* |
|---|---|---|
| 0 | 41.0 | 1 7 9 |
| 1 | 51.0 | 2 |
| 2 | 50.0 | |
| 3 | 36.0 | |
| 4 | 38.0 | |
| 5 | 45.0 | |
| 6 | 21.0 | 3 8 |
| 7 | 32.0 | 3 8 |
| 8 | 32.0 | 2 |
| 9 | 29.0 | 4 6 |

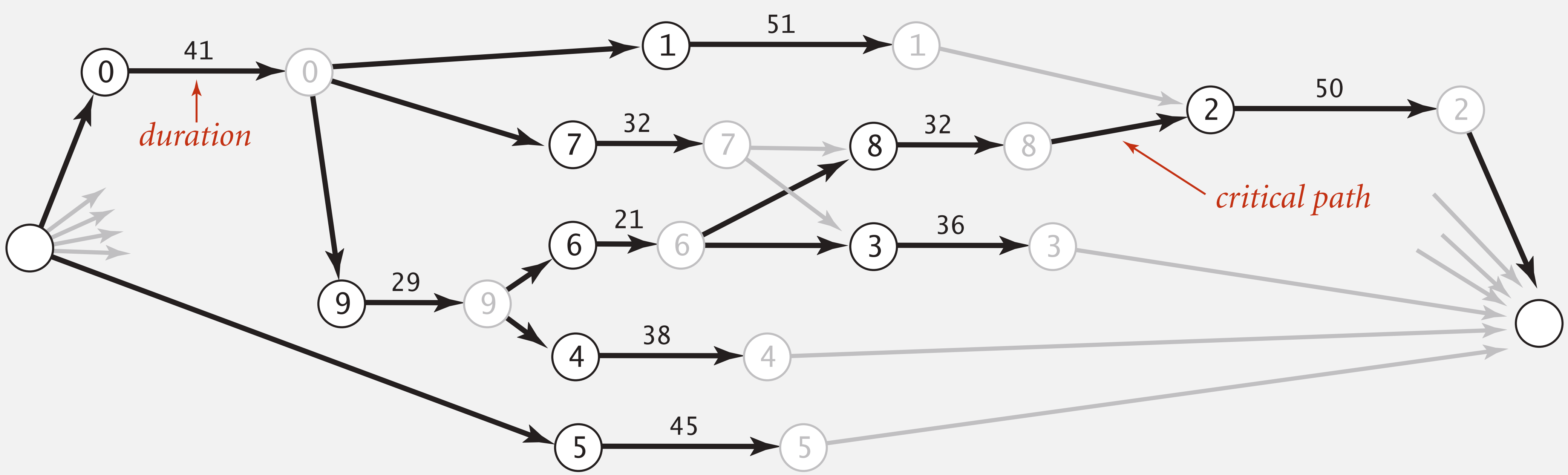


# Critical path method

CPM. Use longest path from the source to schedule each job.




**Parallel job scheduling solution**




http://algs4.cs.princeton.edu

# 4.4 Shortest Paths

- *APIs*
- *shortest-paths properties*
- *Dijkstra's algorithm*
- *edge-weighted DAGs*
- *negative weights*

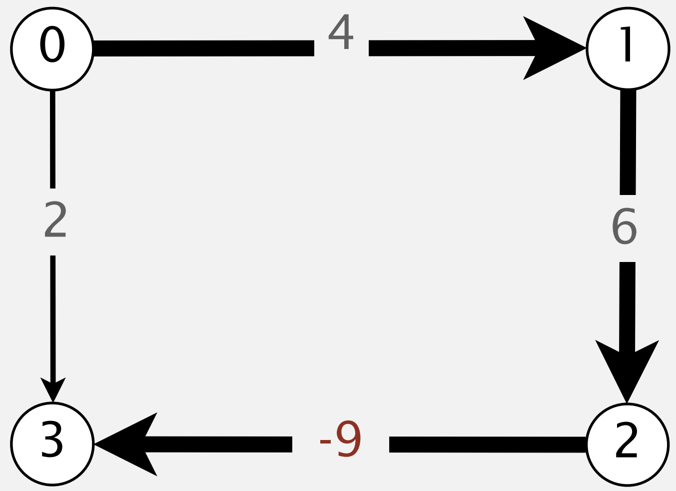

http://algs4.cs.princeton.edu

# 4.4 Shortest Paths

- APIs
- shortest-paths properties
- Dijkstra's algorithm
- edge-weighted DAGs
- *negative weights*

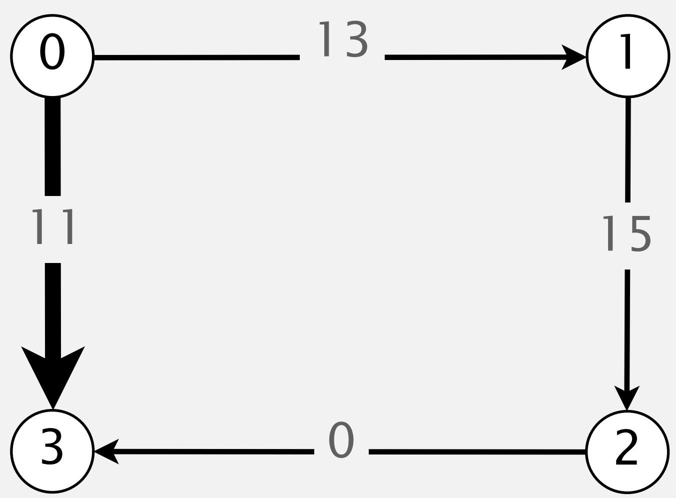
# Shortest paths with negative weights: failed attempts

Dijkstra. Doesn't work with negative edge weights.




Dijkstra selects vertex 3 immediately after 0.
But shortest path from 0 to 3 is 0→1→2→3.

Re-weighting. Add a constant to every edge weight doesn't work.




Adding 9 to each edge weight changes the shortest path from 0→1→2→3 to 0→3.

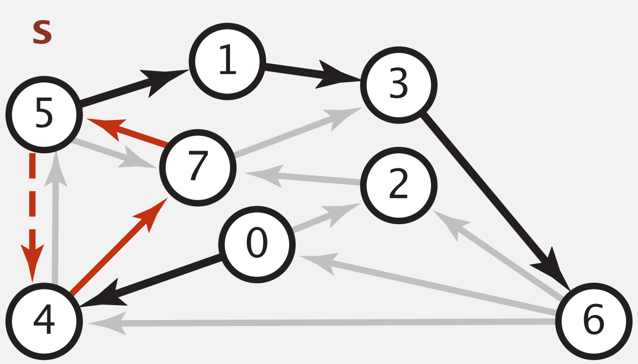
Conclusion. Need a different algorithm.

# Negative cycles

Def. A negative cycle is a directed cycle whose sum of edge weights is negative.

digraph

| | |
|---|---|
| 4->5 | 0.35 |
| 5->4 | -0.66 |
| 4->7 | 0.37 |
| 5->7 | 0.28 |
| 7->5 | 0.28 |
| 5->1 | 0.32 |
| 0->4 | 0.38 |
| 0->2 | 0.26 |
| 7->3 | 0.39 |
| 1->3 | 0.29 |
| 2->7 | 0.34 |
| 6->2 | 0.40 |
| 3->6 | 0.52 |
| 6->0 | 0.58 |
| 6->4 | 0.93 |




negative cycle (-0.66 + 0.37 + 0.28)

5->4->7->5

shortest path from 0 to 6

0->4->7->5->4->7->5...->1->3->6

Proposition. A SPT exists iff no negative cycles.

assuming all vertices reachable from s

# Bellman-Ford algorithm

**Bellman-Ford algorithm**

---

**Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.**

**Repeat V times:**

- **Relax each edge.**

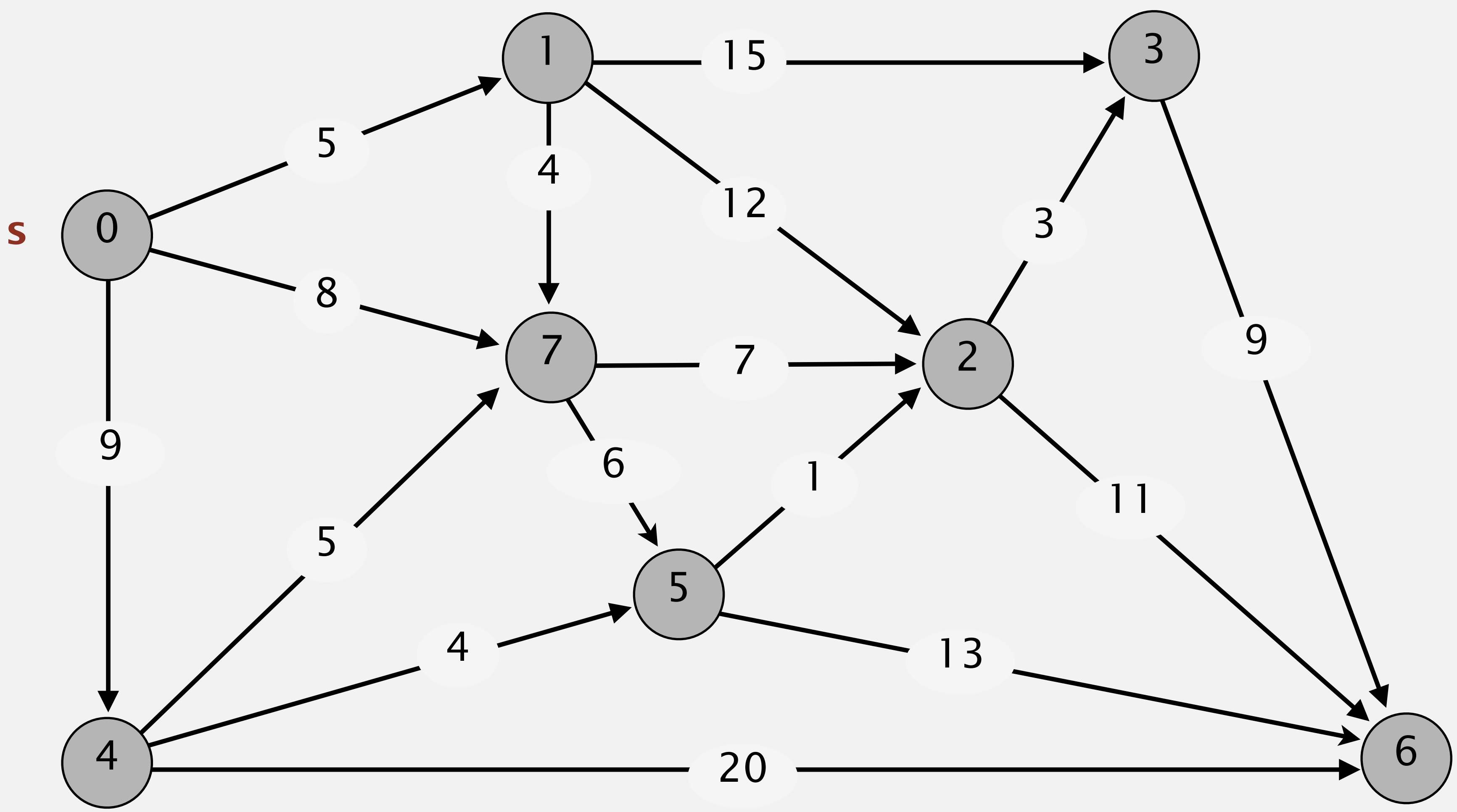
---

```
for (int i = 0; i < G.V(); i++)
   for (int v = 0; v < G.V(); v++)
      for (DirectedEdge e : G.adj(v))
         relax(e);
```

pass i (relax each edge)

# Bellman-Ford algorithm demo

Repeat $V$ times: relax all $E$ edges.

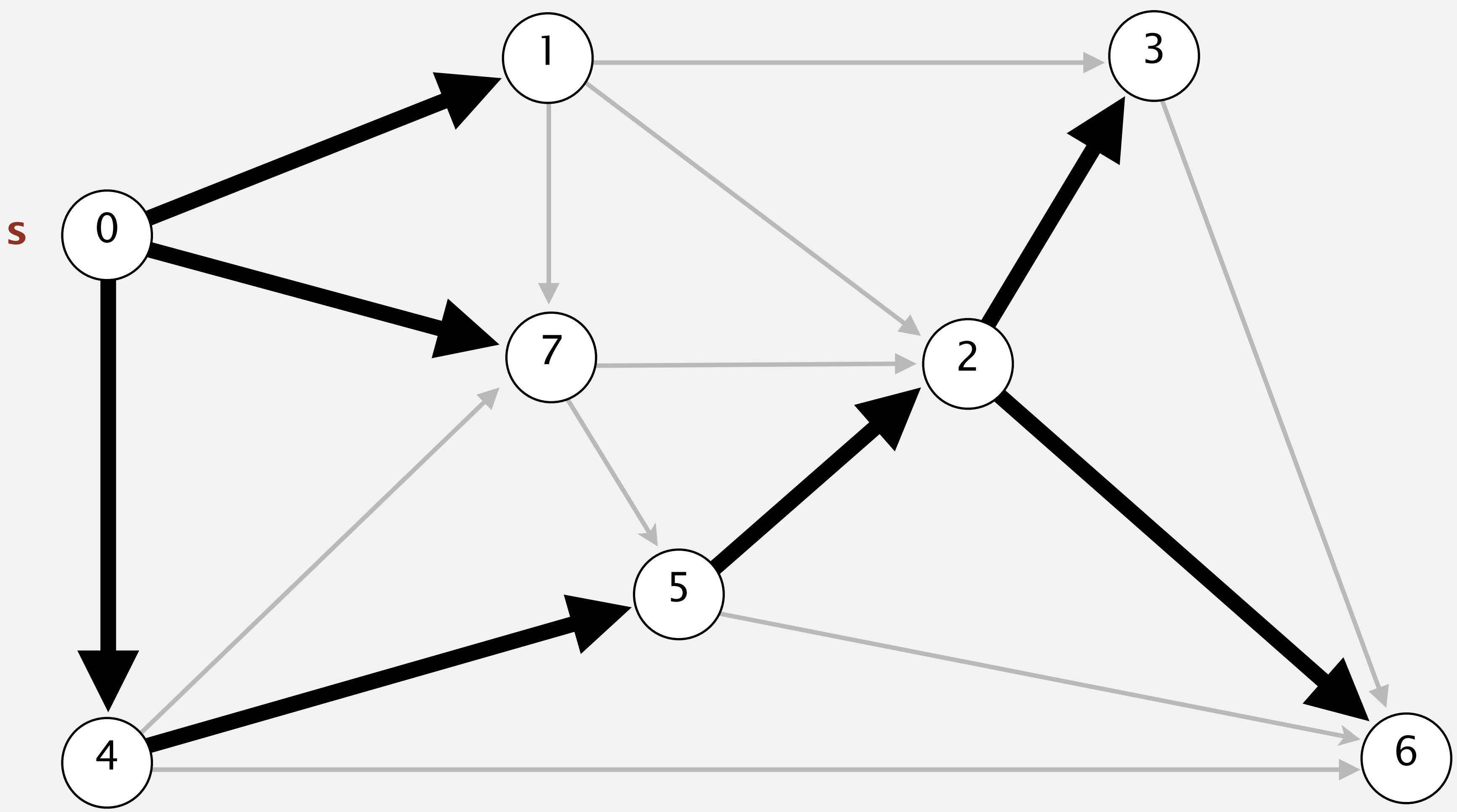



an edge-weighted digraph

| | |
|---|---|
| 0→1 | 5.0 |
| 0→4 | 9.0 |
| 0→7 | 8.0 |
| 1→2 | 12.0 |
| 1→3 | 15.0 |
| 1→7 | 4.0 |
| 2→3 | 3.0 |
| 2→6 | 11.0 |
| 3→6 | 9.0 |
| 4→5 | 4.0 |
| 4→6 | 20.0 |
| 4→7 | 5.0 |
| 5→2 | 1.0 |
| 5→6 | 13.0 |
| 7→5 | 6.0 |
| 7→2 | 7.0 |

# Bellman-Ford algorithm demo

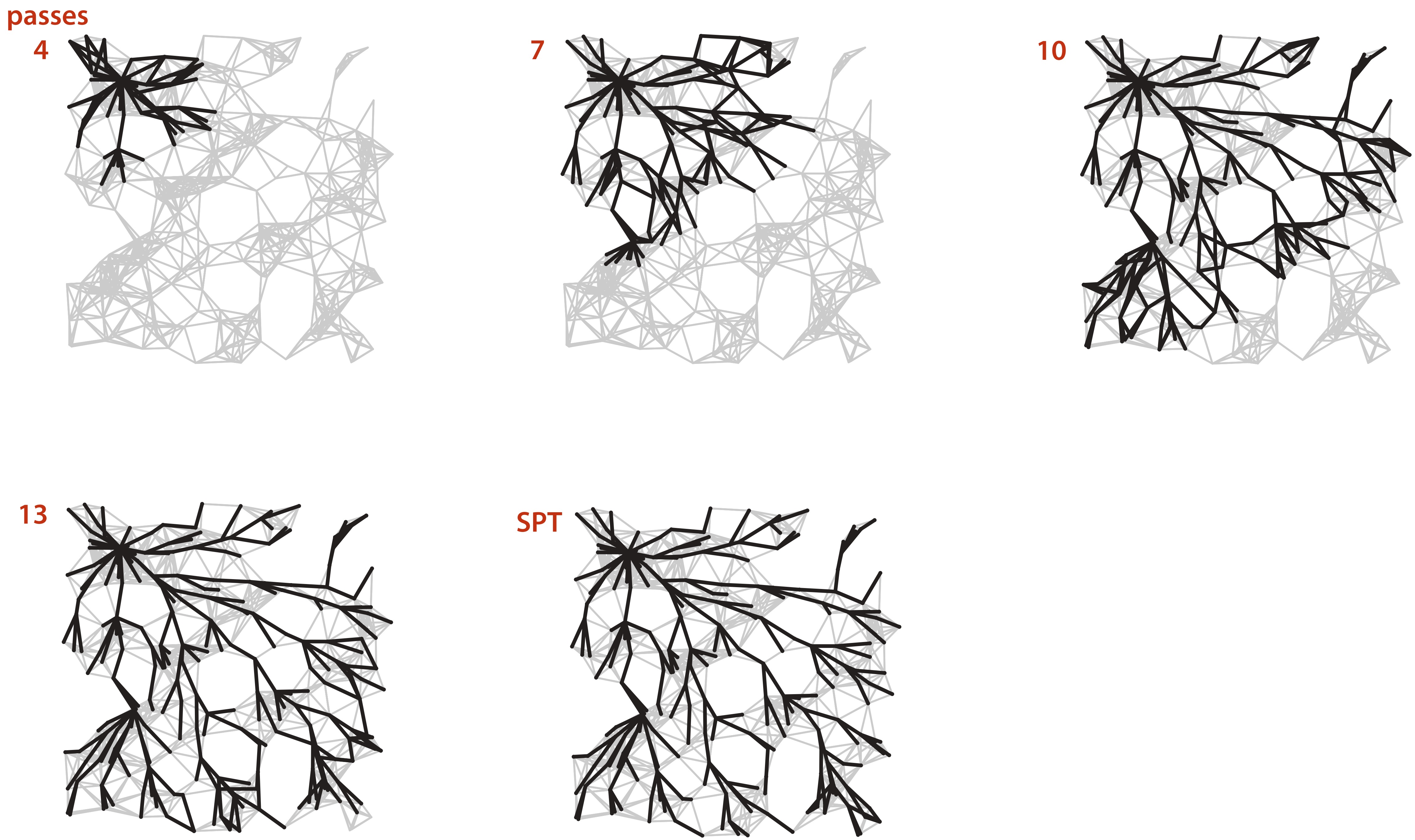
Repeat $V$ times: relax all $E$ edges.




| v | distTo[] | edgeTo[] |
|---|---|---|
| 0 | 0.0 | - |
| 1 | 5.0 | 0→1 |
| 2 | 14.0 | 5→2 |
| 3 | 17.0 | 2→3 |
| 4 | 9.0 | 0→4 |
| 5 | 13.0 | 4→5 |
| 6 | 25.0 | 2→6 |
| 7 | 8.0 | 0→7 |

shortest-paths tree from vertex s

# Bellman-Ford algorithm visualization



# Bellman-Ford algorithm: analysis

**Bellman-Ford algorithm**

---

**Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.**

**Repeat V times:**

- **Relax each edge.**

---

Proposition. Dynamic programming algorithm computes SPT in any edge-weighted digraph with no negative cycles in time proportional to $E \times V$.

Pf idea. After pass $i$, found shortest path containing at most $i$ edges.

# Bellman-Ford algorithm: practical improvement

Observation. If `distTo[v]` does not change during pass `i`, no need to relax any edge pointing from `v` in pass `i+1`.

FIFO implementation. Maintain queue of vertices whose `distTo[]` changed.

be careful to keep at most one copy of each vertex on queue (why?)

Overall effect.

- The running time is still proportional to $E \times V$ in worst case.
- But much faster than that in practice.

# Single source shortest-paths implementation: cost summary

| algorithm | restriction | typical case | worst case | extra space |
|---|---|---|---|---|
| **topological sort** | no directed cycles | E + V | E + V | V |
| **Dijkstra (binary heap)** | no negative weights | E log V | E log V | V |
| **Bellman-Ford** | no negative cycles | E V | E V | V |
| **Bellman-Ford (queue-based)** | | E + V | E V | V |

Remark 1. Directed cycles make the problem harder.

Remark 2. Negative weights make the problem harder.

Remark 3. Negative cycles makes the problem intractable.

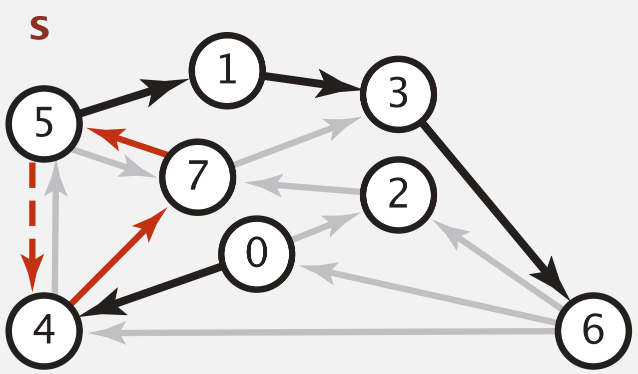
# Finding a negative cycle

Negative cycle. Add two method to the API for SP.

| | | |
|---|---|---|
| `boolean` | `hasNegativeCycle()` | *is there a negative cycle?* |
| `Iterable <DirectedEdge>` | `negativeCycle()` | *negative cycle reachable from s* |

**digraph**

| | |
|---|---|
| 4->5 | 0.35 |
| 5->4 | -0.66 |
| 4->7 | 0.37 |
| 5->7 | 0.28 |
| 7->5 | 0.28 |
| 5->1 | 0.32 |
| 0->4 | 0.38 |
| 0->2 | 0.26 |
| 7->3 | 0.39 |
| 1->3 | 0.29 |
| 2->7 | 0.34 |
| 6->2 | 0.40 |
| 3->6 | 0.52 |
| 6->0 | 0.58 |
| 6->4 | 0.93 |

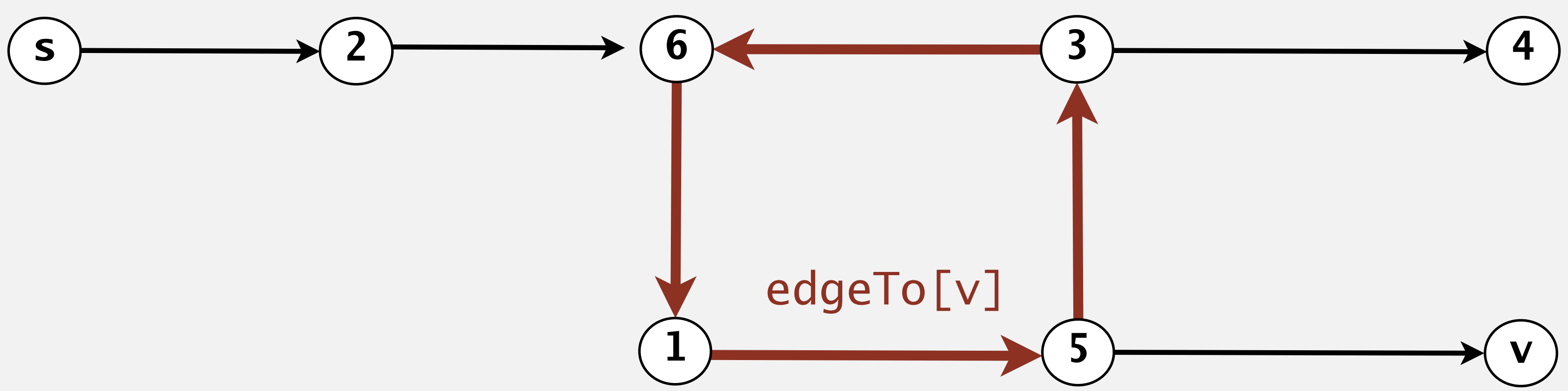



**negative cycle (-0.66 + 0.37 + 0.28)**

5->4->7->5

# Finding a negative cycle

Observation. If there is a negative cycle, Bellman-Ford gets stuck in loop, updating `distTo[]` and `edgeTo[]` entries of vertices in the cycle.




Proposition. If any vertex `v` is updated in phase V, there exists a negative cycle (and can trace back `edgeTo[v]` entries to find it).

In practice. Check for negative cycles more frequently.

# Negative cycle application: arbitrage detection

Problem. Given table of exchange rates, is there an arbitrage opportunity?

| | USD | EUR | GBP | CHF | CAD |
|---|---|---|---|---|---|
| USD | 1 | 0.741 | 0.657 | 1.061 | 1.011 |
| EUR | 1.350 | 1 | 0.888 | 1.433 | 1.366 |
| GBP | 1.521 | 1.126 | 1 | 1.614 | 1.538 |
| CHF | 0.943 | 0.698 | 0.620 | 1 | 0.953 |
| CAD | 0.995 | 0.732 | 0.650 | 1.049 | 1 |

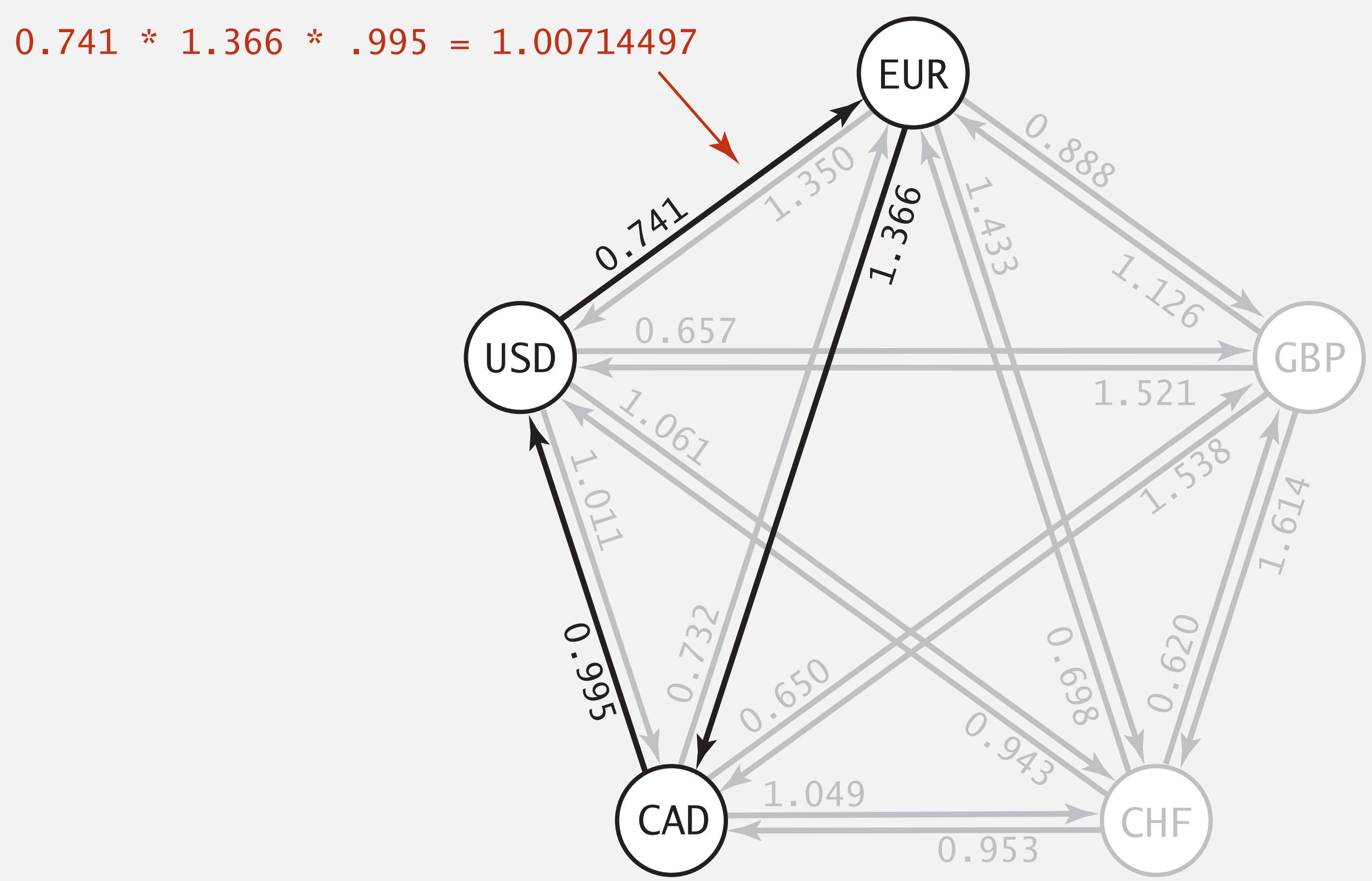
Ex. \$1,000 $\Rightarrow$ 741 Euros $\Rightarrow$ 1,012.206 Canadian dollars $\Rightarrow$ \$1,007.14497.

$1000 \times 0.741 \times 1.366 \times 0.995 = 1007.14497$

# Negative cycle application: arbitrage detection

Currency exchange graph.

- Vertex = currency.
- Edge = transaction, with weight equal to exchange rate.
- Find a directed cycle whose product of edge weights is $> 1$.

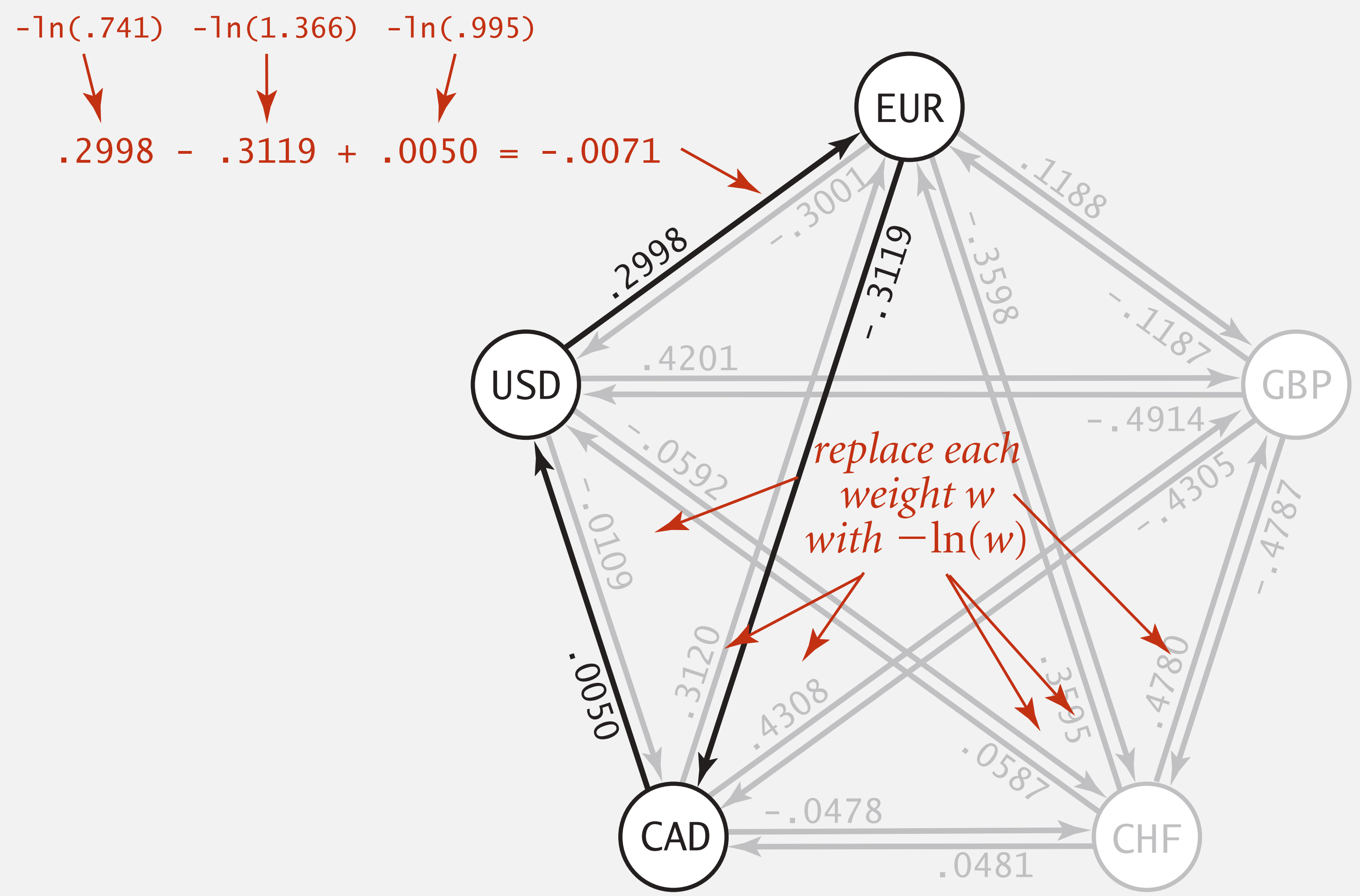



Challenge. Express as a negative cycle detection problem.

# Negative cycle application: arbitrage detection

Model as a negative cycle detection problem by taking logs.

- Let weight of edge $v \rightarrow w$ be $-\ln$ (exchange rate from currency $v$ to $w$).
- Multiplication turns to addition; $> 1$ turns to $< 0$.
- Find a directed cycle whose sum of edge weights is $< 0$ (negative cycle).




Remark. Fastest algorithm is extraordinarily valuable!

# Shortest paths summary

Dijkstra's algorithm.

- Nearly linear-time when weights are nonnegative.
- Generalization encompasses DFS, BFS, and Prim.

Acyclic edge-weighted digraphs.

- Arise in applications.
- Faster than Dijkstra's algorithm.
- Negative weights are no problem.

Negative weights and negative cycles.

- Arise in applications.
- If no negative cycles, can find shortest paths via Bellman-Ford.
- If negative cycles, can find one via Bellman-Ford.

Shortest-paths is a broadly useful problem-solving model.


http://algs4.cs.princeton.edu

# 4.4 Shortest Paths

- APIs
- shortest-paths properties
- Dijkstra's algorithm
- edge-weighted DAGs
- *negative weights*


http://algs4.cs.princeton.edu

# 4.4 Shortest Paths

- *APIs*
- *shortest-paths properties*
- *Dijkstra's algorithm*
- *edge-weighted DAGs*
- *negative weights*