https://algs4.cs.princeton.edu

4.4 Shortest Paths

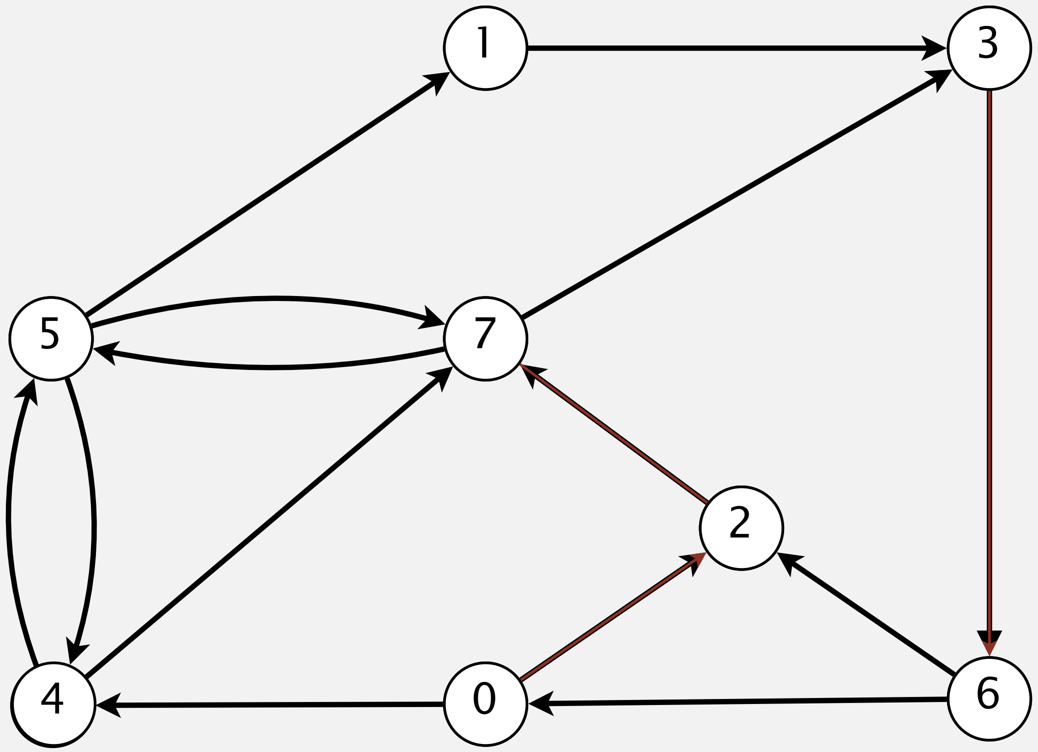
- *APIs*
- *properties*
- *Bellman–Ford algorithm*
- *Dijkstra's algorithm*
- *seam carving*

Shortest paths in an edge-weighted digraph

Given an edge-weighted digraph, find the shortest path from s to t.

edge-weighted digraph

4->5	0.35
5->4	0.35
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



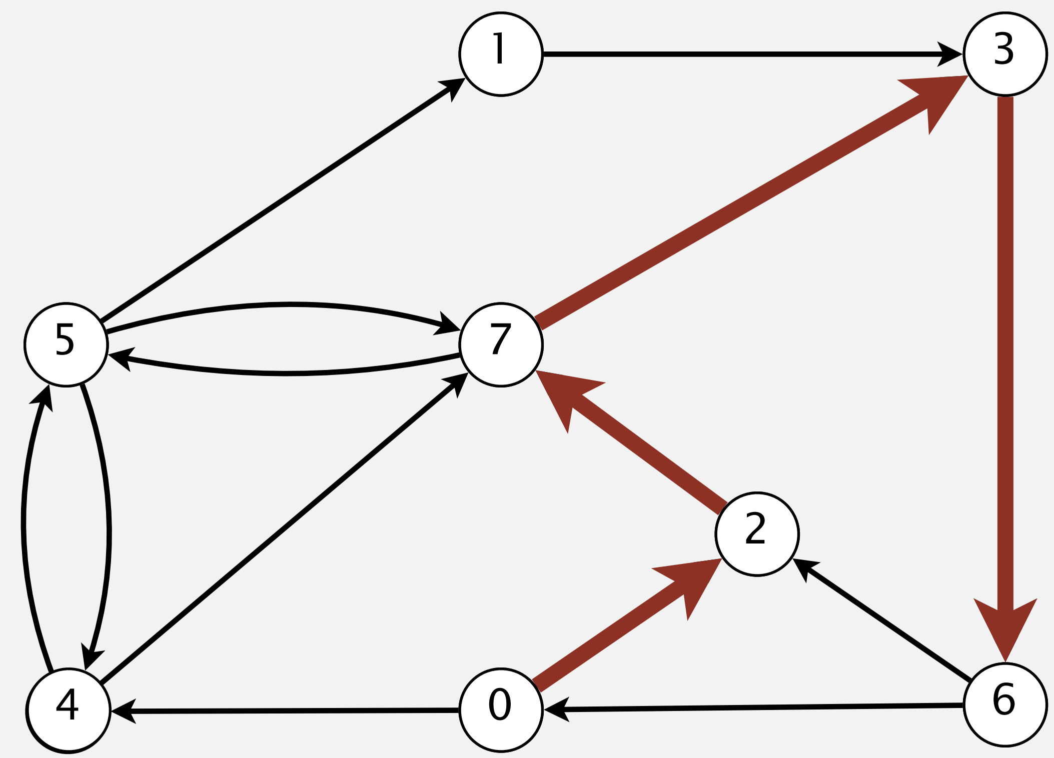
Exercise: find the shortest path from 0 to 6 in the above digraph

Shortest paths in an edge-weighted digraph

Given an edge-weighted digraph, find the shortest path from s to t.

edge-weighted digraph

```
4->5  0.35
5->4  0.35
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
```



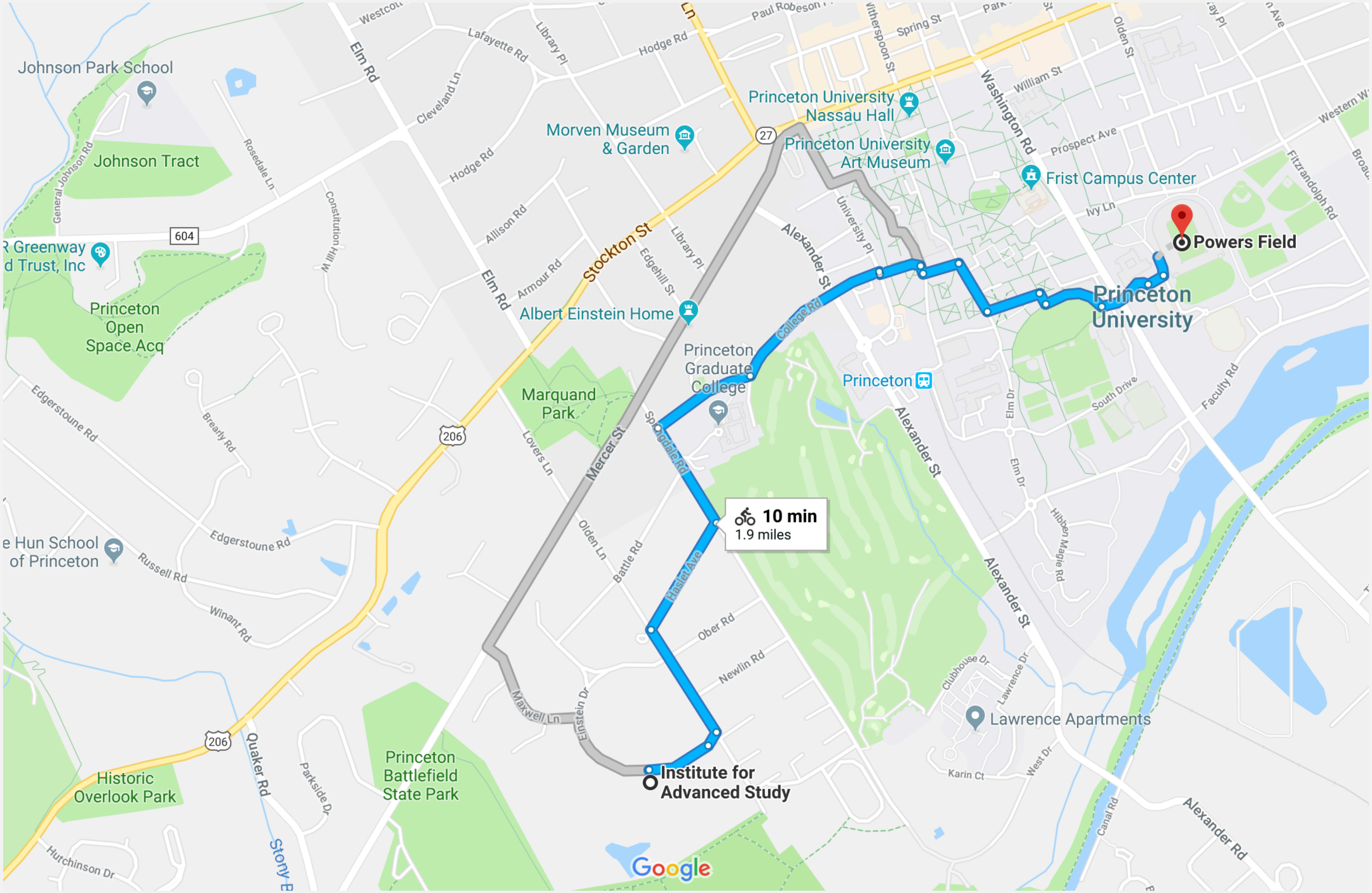
shortest path from 0 to 6

$0 \to 2 \to 7 \to 3 \to 6$

length of path = 1.51

$(0.26 + 0.34 + 0.39 + 0.52)$

Google maps



10 min
1.9 miles
Institute for Advanced Study
Powers Field
Princeton University
Princeton Graduate College
Princeton University Nassau Hall
Princeton University Art Museum
Frist Campus Center
Morven Museum & Garden
Albert Einstein Home
Marquand Park
Princeton Battlefield State Park
Historic Overlook Park
Johnson Park School
Johnson Tract
Princeton Open Space Acq
Lawrence Apartments
Princeton
Stockton St
Mercer St
Alexander St
Alexander Rd
Washington Rd
Elm Rd
Quaker Rd
Springdale Rd
Hasbet Ave
College Rd
Maxwell Ln
Einstein Dr
Olden Ln
Battle Rd
Ober Rd
Newlin Rd
Edgerstoune Rd
Russell Rd
Winant Rd
Hodge Rd
Lafayette Rd
Library Pl
Faculty Rd
Prospect Ave
William St
Spring St
Google

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving. ← see Assignment 7
- Texture mapping.
- Robot navigation.
- Typesetting in TeX.
- Currency exchange.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Optimal truck routing through given traffic congestion pattern.

http://en.wikipedia.org/wiki/Seam_carving

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

Shortest path variants

Which vertices?

- Single source: from one vertex s to every other vertex.
- Single sink: from every vertex to one vertex t.
- Source–sink: from one vertex s to another t.
- All pairs: between all pairs of vertices.

Restrictions on edge weights?

- Non-negative weights. ← we assume this throughout today's lecture (even though some algorithms can handle negative weights)
- Euclidean weights.
- Arbitrary weights.

Cycles?

- No directed cycles.
- No "negative cycles."

Simplifying assumption. Each vertex is reachable from s.

Shortest paths: quiz 1

Which variant in car GPS?

A. Single source: from one vertex s to every other vertex.

B. Single sink: from every vertex to one vertex t.

C. Source–sink: from one vertex s to another t.

D. All pairs: between all pairs of vertices.

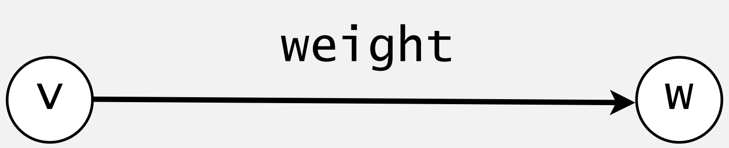
https://algs4.cs.princeton.edu

4.4 Shortest Paths

- *APIs*
- *properties*
- *Bellman–Ford algorithm*
- *Dijkstra's algorithm*
- *seam carving*

Weighted directed edge API

`public class DirectedEdge`		
	`DirectedEdge(int v, int w, double weight)`	*weighted edge v→w*
`int`	`from()`	*vertex v*
`int`	`to()`	*vertex w*
`double`	`weight()`	*weight of this edge*
`String`	`toString()`	*string representation*



Idiom for processing an edge `e`: `int v = e.from(), w = e.to();`

Weighted directed edge: implementation in Java

Similar to `Edge` for undirected graphs, but a bit simpler.

```
public class DirectedEdge
{
   private final int v, w;
   private final double weight;

   public DirectedEdge(int v, int w, double weight)
   {
     this.v = v;
     this.w = w;
     this.weight = weight;
   }

   public int from()
   {  return v;  }

   public int to()
   {  return w; }

   public double weight()
   {  return weight;  }

}
```

from() and to() replace either() and other()

Edge-weighted digraph API

public class	EdgeWeightedDigraph	
	EdgeWeightedDigraph(int V)	*edge-weighted digraph with V vertices*
void	addEdge(DirectedEdge e)	*add weighted directed edge e*
Iterable<DirectedEdge>	adj(int v)	*edges adjacent from v*
int	V()	*number of vertices*
int	E()	*number of edges*
Iterable<DirectedEdge>	edges()	*all edges*

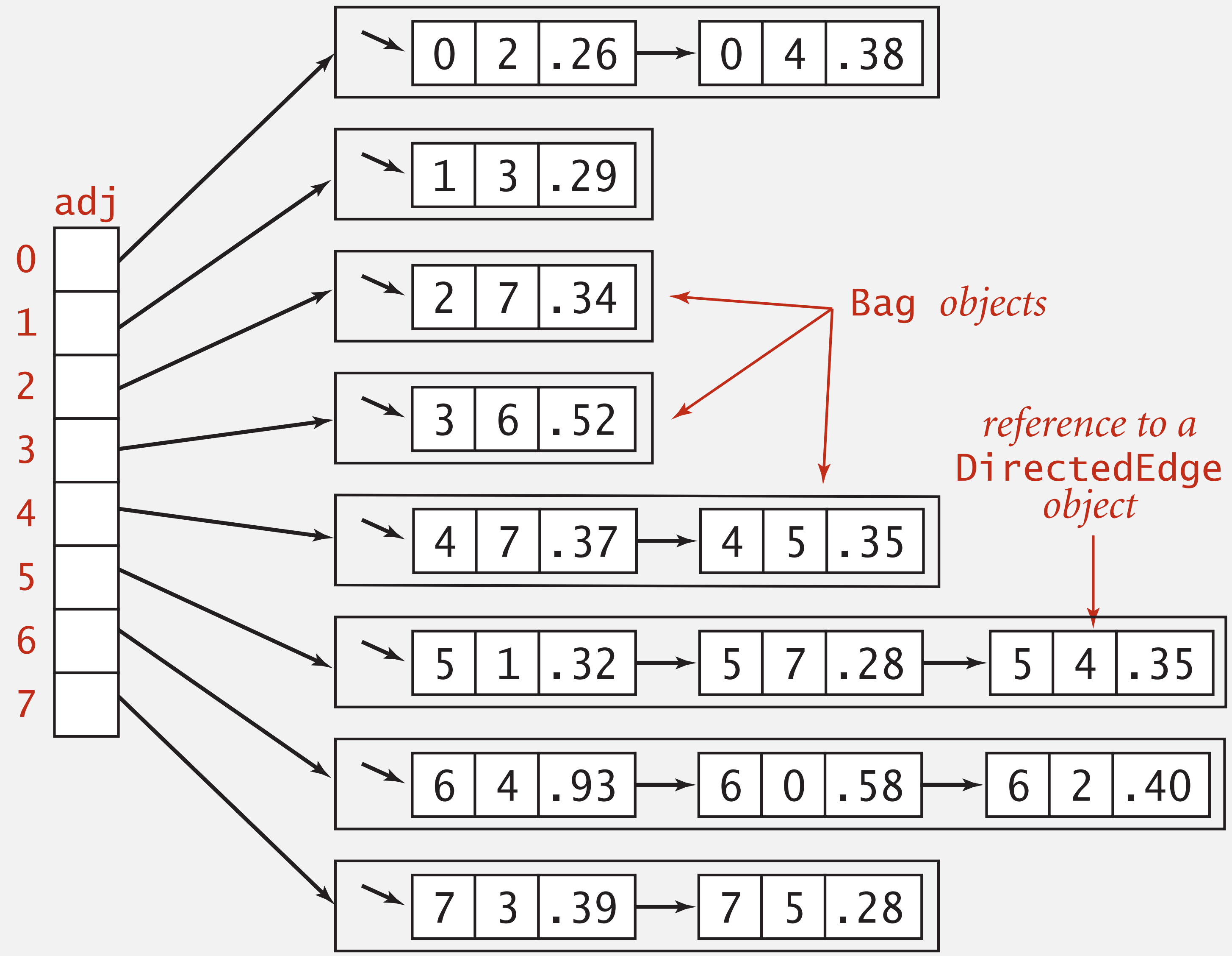
Conventions. Allow self-loops and parallel edges.

Edge-weighted digraph: adjacency-lists representation

tinyEWD.txt

V *E*

```
8
15
4 5  0.35
5 4  0.35
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
```



Edge-weighted digraph: adjacency-lists implementation in Java

Almost identical to `EdgeWeightedGraph`.

```
public class EdgeWeightedDigraph
{
   private final int V;
   private final Bag<DirectedEdge>[] adj;

   public EdgeWeightedDigraph(int V)
   {
      this.V = V;
      adj = (Bag<Edge>[]) new Bag[V];
      for (int v = 0; v < V; v++)
         adj[v] = new Bag<DirectedEdge>();
   }

   public void addEdge(DirectedEdge e)
   {
      int v = e.from(), w = e.to();
      adj[v].add(e);
   }

   public Iterable<DirectedEdge> adj(int v)
   {  return adj[v];  }
}
```

add edge e = v→w to
only v's adjacency list

Single-source shortest paths API

Goal. Find the shortest path from s to every other vertex.

	`public class SP`	
	`SP(EdgeWeightedDigraph G, int s)`	*shortest paths from s in digraph G*
`double`	`distTo(int v)`	*length of shortest path from s to v*
`Iterable <DirectedEdge>`	`pathTo(int v)`	*shortest path from s to v*
`boolean`	`hasPathTo(int v)`	*is there a path from s to v?*

https://algs4.cs.princeton.edu

4.4 Shortest Paths

- APIs
- *properties*
- Bellman–Ford algorithm
- Dijkstra's algorithm
- seam carving

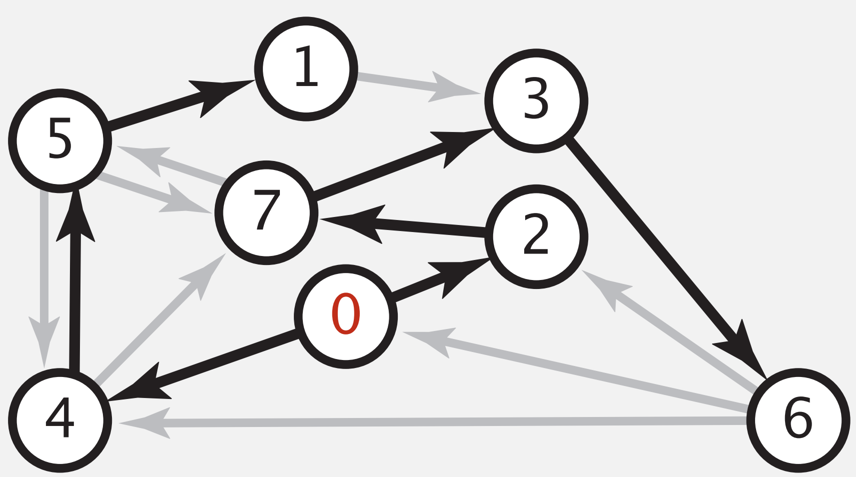
Data structures for single-source shortest paths

Goal. Find a shortest path from s to every other vertex.

Observation. A shortest-paths tree (SPT) solution exists. Why?

Consequence. Can represent a SPT with two vertex-indexed arrays:

- `distTo[v]` is length of a shortest path from s to v.
- `edgeTo[v]` is last edge on a shortest path from s to v.



shortest-paths tree from 0

	`distTo[]`	`edgeTo[]`
0	0	null
1	1.05	5->1 0.32
2	0.26	0->2 0.26
3	0.97	7->3 0.37
4	0.38	0->4 0.38
5	0.73	4->5 0.35
6	1.49	3->6 0.52
7	0.60	2->7 0.34

parent-link representation

Data structures for single-source shortest paths

Goal. Find a shortest path from s to every other vertex.

Observation. A shortest-paths tree (SPT) solution exists. Why?

Consequence. Can represent a SPT with two vertex-indexed arrays:

- `distTo[v]` is length of a shortest path from s to v.
- `edgeTo[v]` is last edge on a shortest path from s to v.

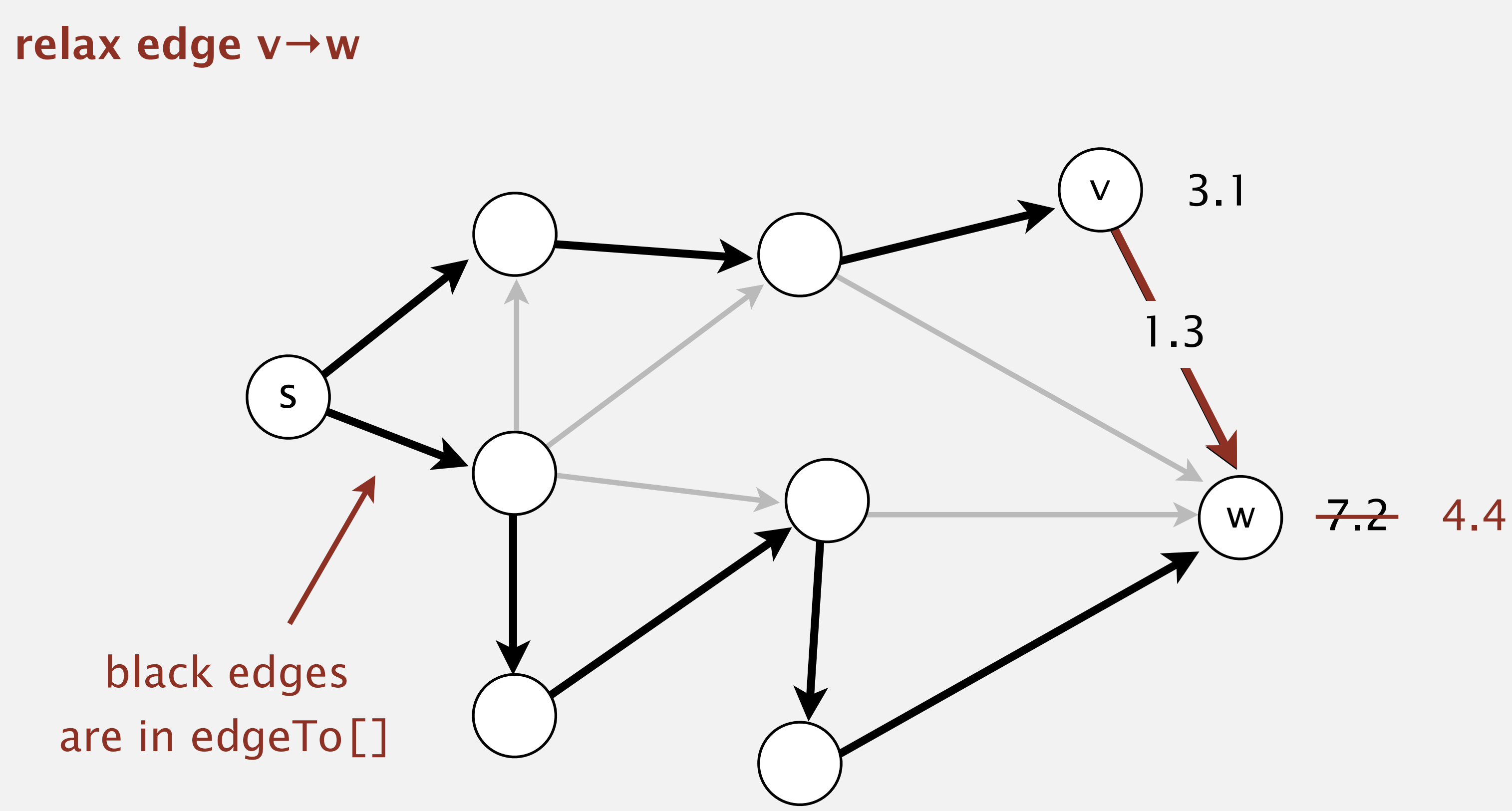
Computing path to specific vertex.

```
public Iterable<DirectedEdge> pathTo(int v)
{
   // Shortest-Paths Tree has already been computed and stored as
   // arrays distTo[] and edgeTo[]
   Stack<DirectedEdge> path = new Stack<DirectedEdge>();
   for (DirectedEdge e = edgeTo[v]; e != null; e = edgeTo[e.from()])
      path.push(e);
   return path;
}
```

Edge relaxation

Relax edge $e = v \rightarrow w$.

- `distTo[v]` is length of shortest known path from `s` to `v`.
- `distTo[w]` is length of shortest known path from `s` to `w`.
- `edgeTo[w]` is last edge on shortest known path from `s` to `w`.
- If `e = v→w` yields shorter path to `w`, update `distTo[w]` and `edgeTo[w]`.



Edge relaxation

Relax edge $e = v \rightarrow w$.

- distTo[v] is length of shortest known path from s to v.
- distTo[w] is length of shortest known path from s to w.
- edgeTo[w] is last edge on shortest known path from s to w.
- If e = v→w yields shorter path to w, update distTo[w] and edgeTo[w].

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
       distTo[w] = distTo[v] + e.weight();
       edgeTo[w] = e;
   }
}
```

Shortest paths: quiz 2

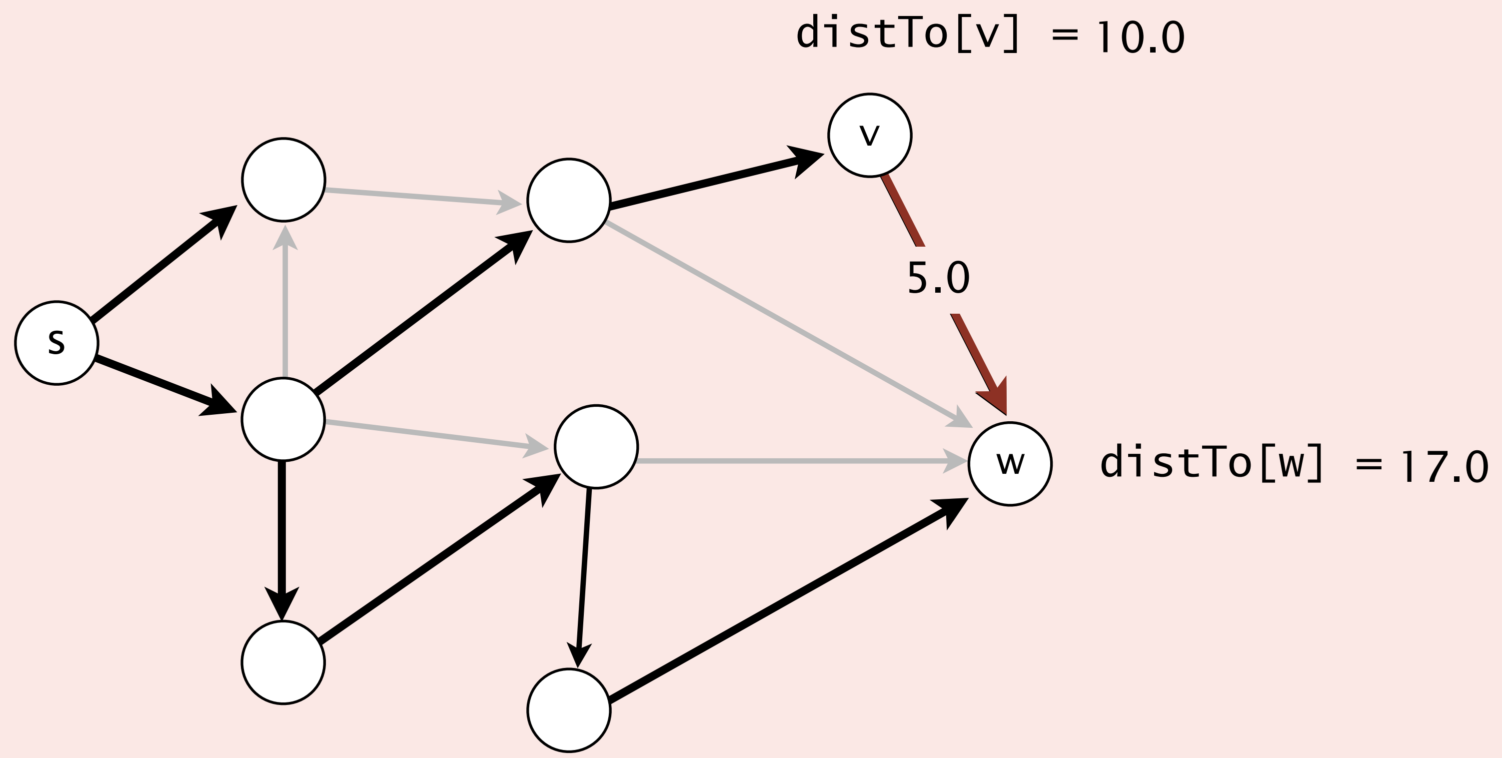
What are the values of `distTo[v]` and `distTo[w]` after relaxing `v→w` ?

A. 10.0 and 15.0

B. 10.0 and 17.0

C. 12.0 and 15.0

D. 12.0 and 17.0



Framework for shortest-paths algorithm

Generic algorithm (to compute a SPT from s)

For each vertex v: distTo[v] = ∞.

For each vertex v: edgeTo[v] = null.

distTo[s] = 0.

Repeat until done:

- **Relax any edge.**

Key properties.

- `distTo[v]` is the length of a simple path from `s` to `v`.
- `distTo[v]` does not increase.

Framework for shortest-paths algorithm

Generic algorithm (to compute a SPT from s)

For each vertex v: distTo[v] = ∞.

For each vertex v: edgeTo[v] = null.

distTo[s] = 0.

Repeat until done:

- **Relax any edge.**

Efficient implementations.

- Which edge to relax next?
- How many edge relaxations needed?

Ex 1. Bellman–Ford algorithm.

Ex 2. Dijkstra's algorithm.

Ex 3. Topological sort algorithm.

https://algs4.cs.princeton.edu

4.4 Shortest Paths

- *APIs*
- *properties*
- *Bellman–Ford algorithm*
- *Dijkstra's algorithm*
- *seam carving*

Bellman–Ford algorithm

Bellman–Ford algorithm

For each vertex v: distTo[v] = ∞.

For each vertex v: edgeTo[v] = null.

distTo[s] = 0.

Repeat V–1 times:

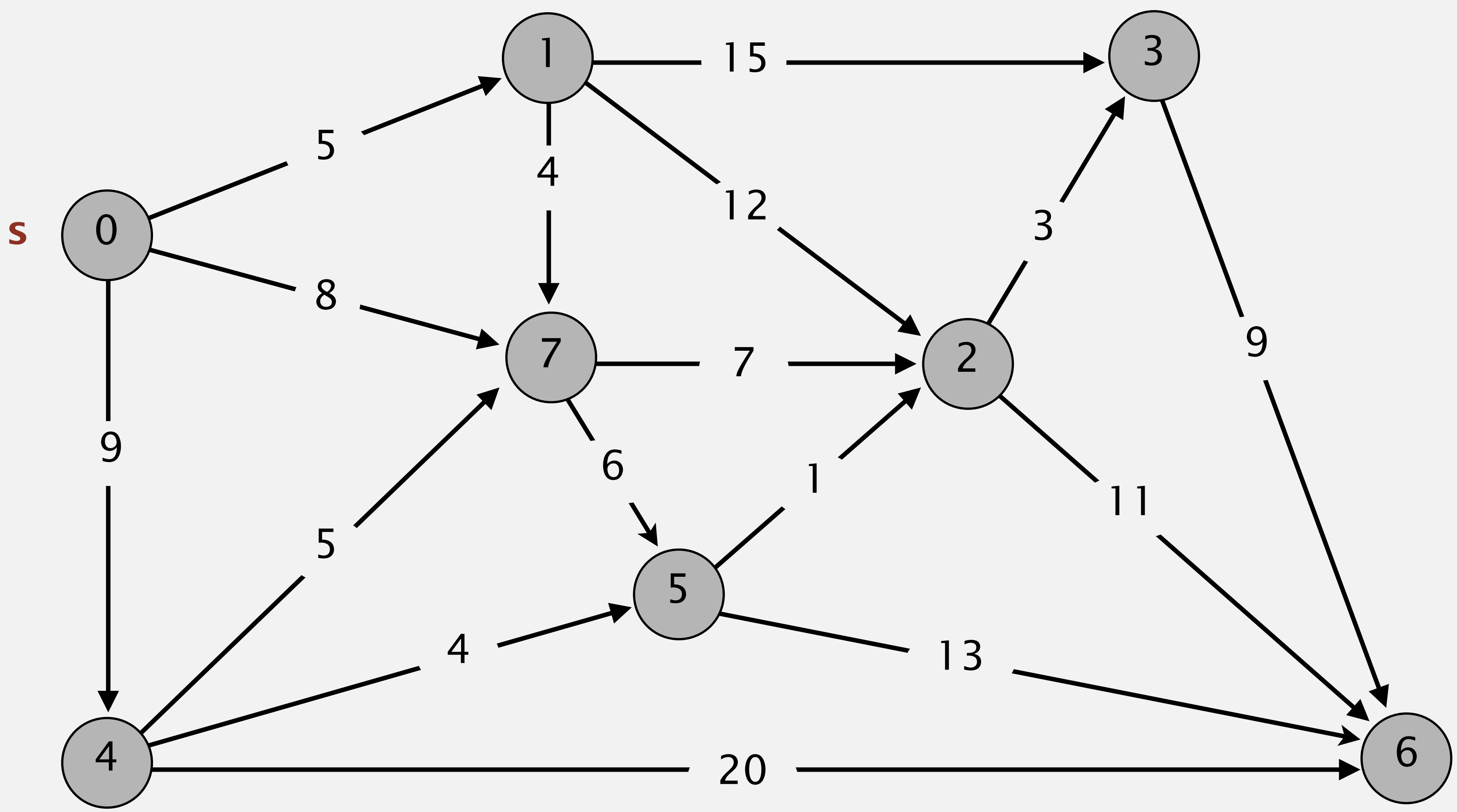
- **Relax each edge.**

```
for (int i = 1; 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 $V - 1$ times: relax all E edges.

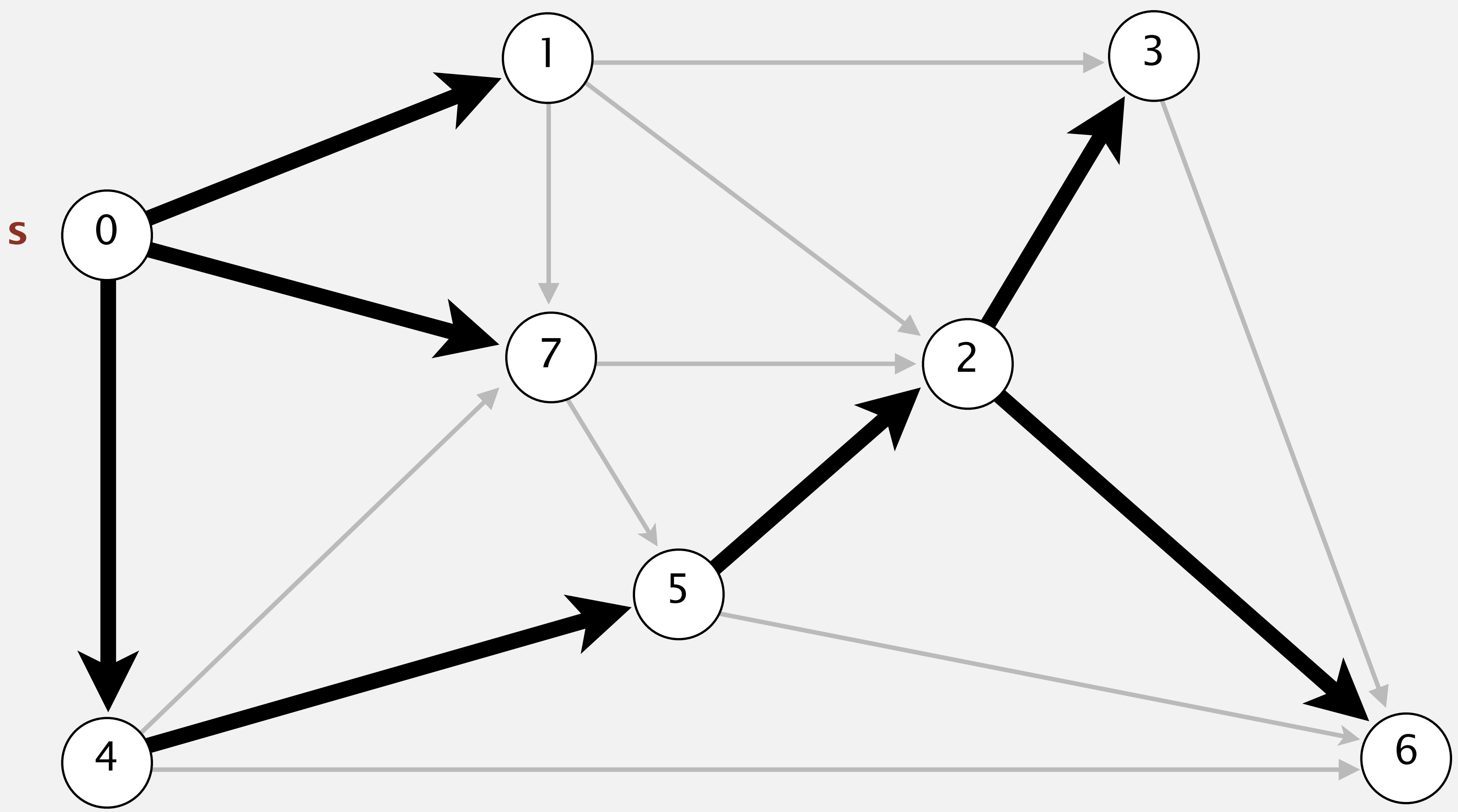


an edge-weighted digraph

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

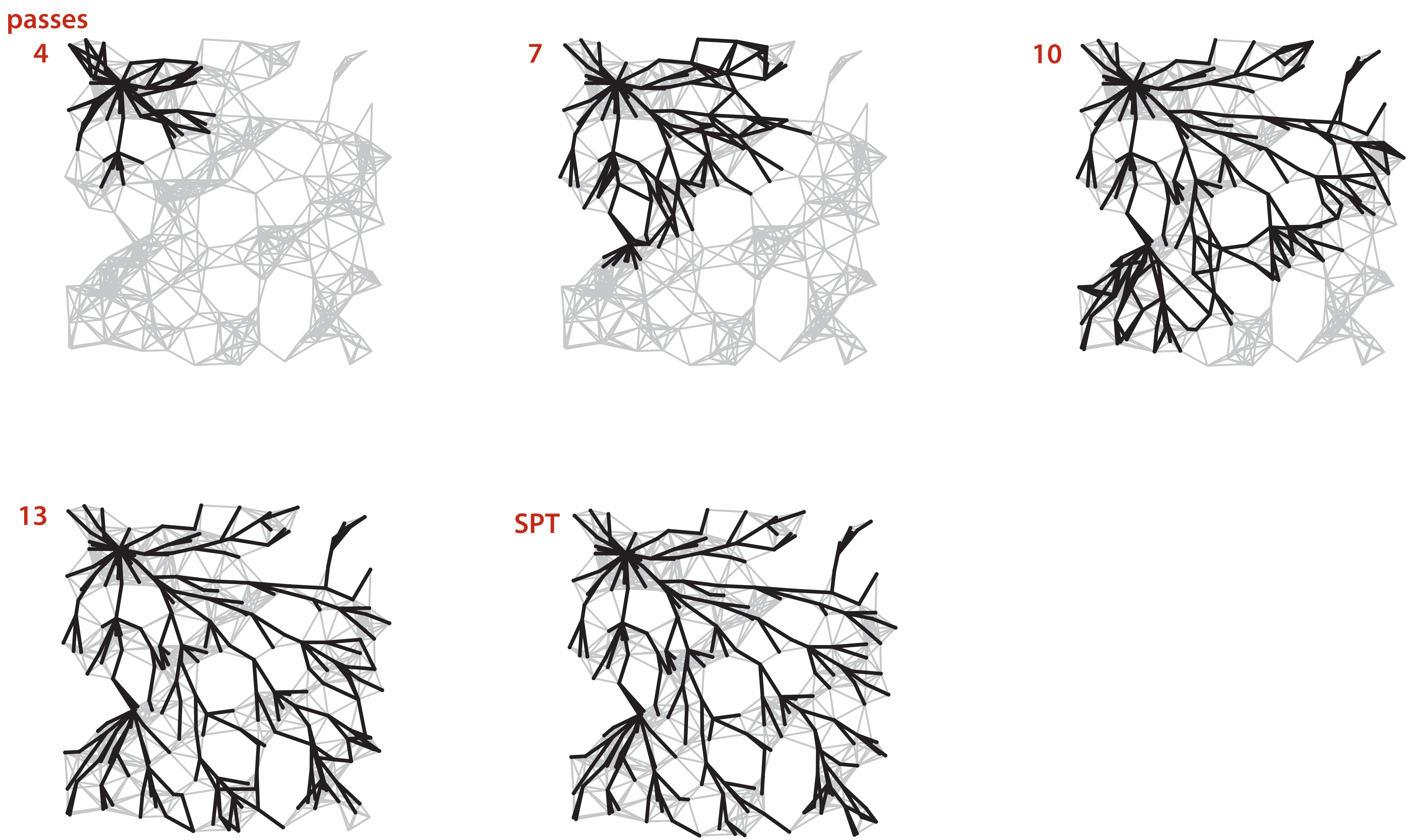
Repeat $V - 1$ times: relax all E edges.



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

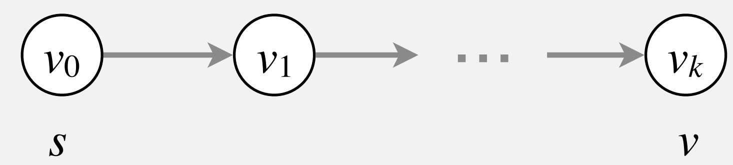
Bellman–Ford algorithm: visualization



Bellman–Ford algorithm: correctness proof

Proposition. Let $s = v_0 \to v_1 \to v_2 \to \ldots \to v_k = v$ be a shortest path from s to v.
Then, after pass i, `distTo[`v_i`]` $= d^*(v_i)$.

length of shortest path from s to v_i



Pf. [by induction on i]

- Inductive hypothesis: after pass i, `distTo[`v_i`]` $= d^*(v_i)$.
- Since `distTo[`v_{i+1}`]` is the length of some path from s to v_{i+1}, we must have `distTo[`v_{i+1}`]` $\geq d^*(v_{i+1})$.
- Immediately after relaxing edge $v_i \to v_{i+1}$ in pass $i+1$, we have

$$\texttt{distTo}[v_{i+1}] \leq \texttt{distTo}[v_i] + weight(v_i, v_{i+1})$$
$$= d^*(v_i) + weight(v_i, v_{i+1})$$
$$= d^*(v_{i+1}).$$

- Thus, at the end of pass $i+1$, `distTo[`v_{i+1}`]` $= d^*(v_{i+1})$. ▪

Corollary. Bellman–Ford computes shortest path distances.

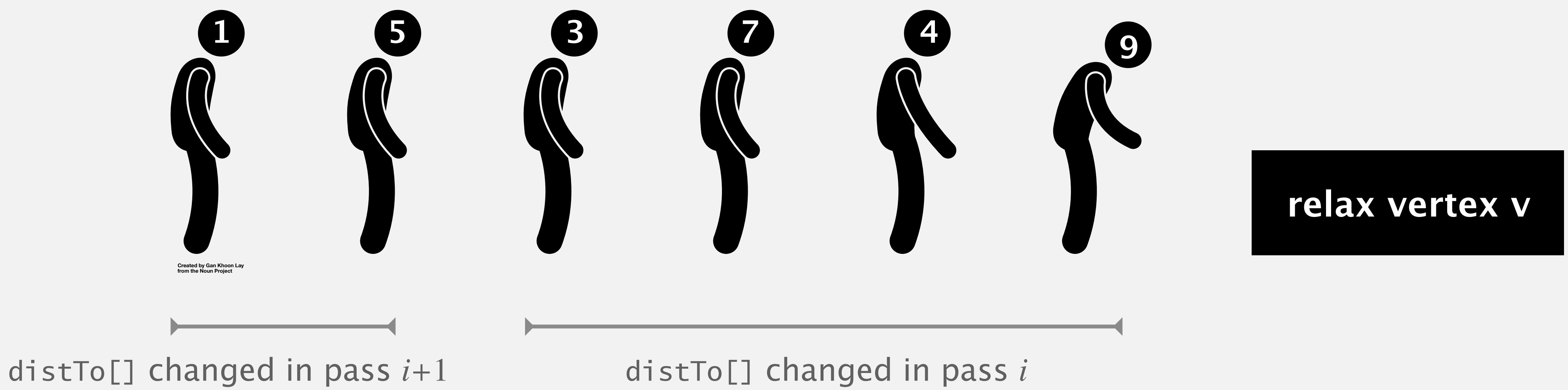
Pf. There exists a shortest path from s to v with at most $V - 1$ edges.
$\Rightarrow\ \leq\ V - 1$ passes.

edge weights are non-negative

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 $i + 1$.

Queue-based implementation of Bellman–Ford. Maintain queue of vertices whose `distTo[]` values needs updating.



Impact.

- In the worst case, the running time is still proportional to $E \times V$.
- But much faster in practice.

https://algs4.cs.princeton.edu

4.4 Shortest Paths

- *APIs*
- *properties*
- *Bellman–Ford algorithm*
- ***Dijkstra's algorithm***
- *seam carving*

Edsger W. Dijkstra: select quotes

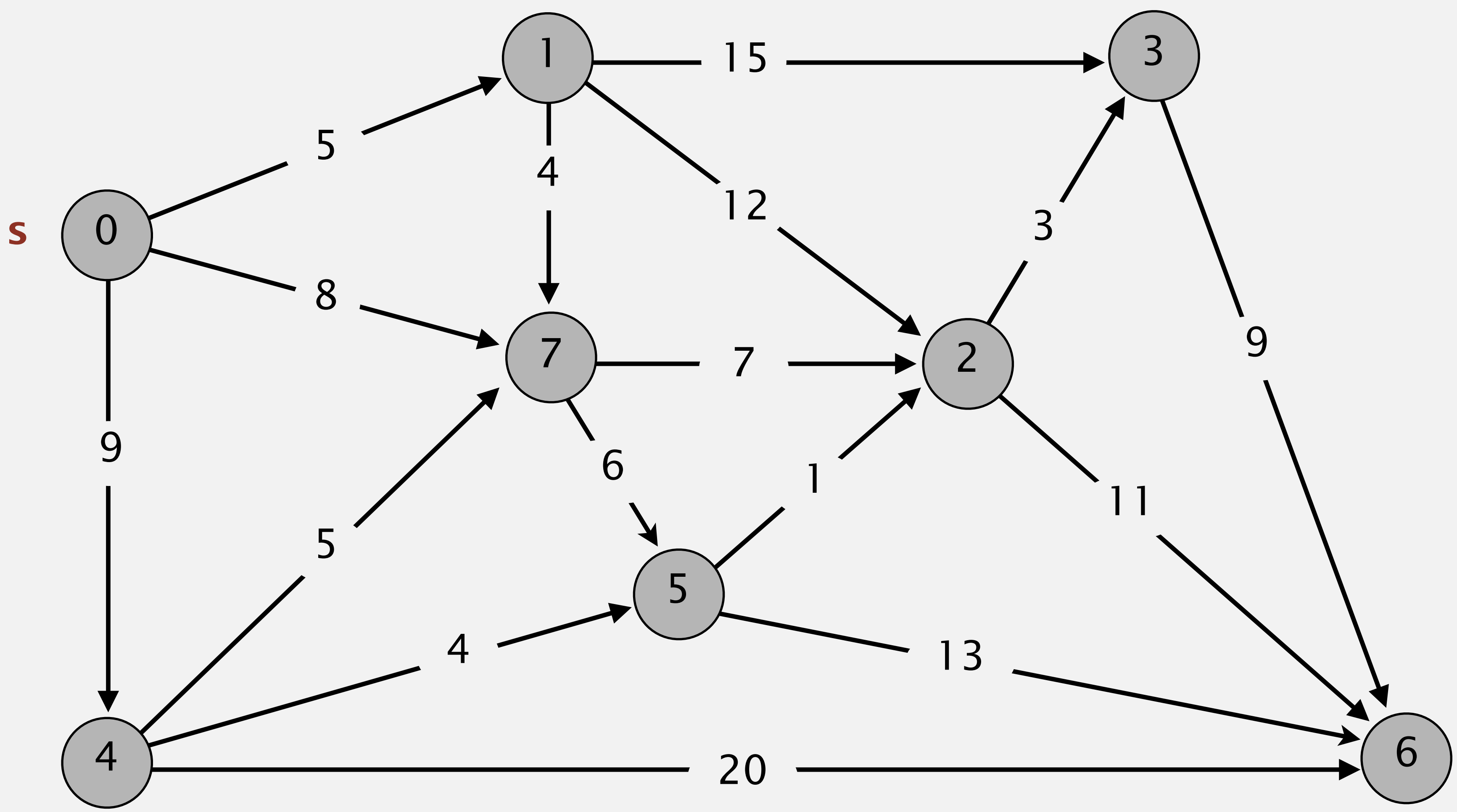
" It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration."

Edsger W. Dijkstra
Turing award 1972

Edsger W. Dijkstra: select quotes

Dijkstra's algorithm demo

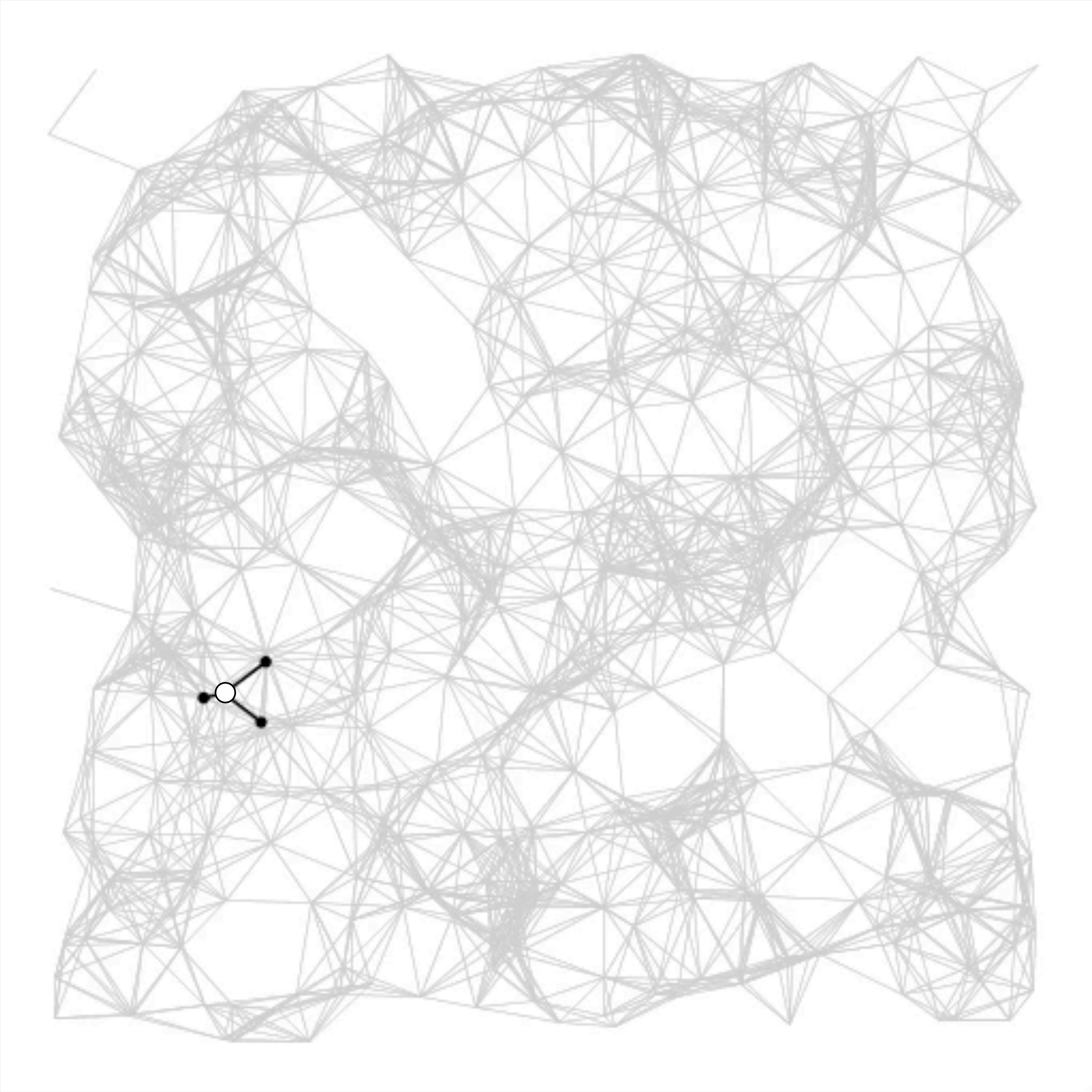
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



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

Dijkstra's algorithm visualization



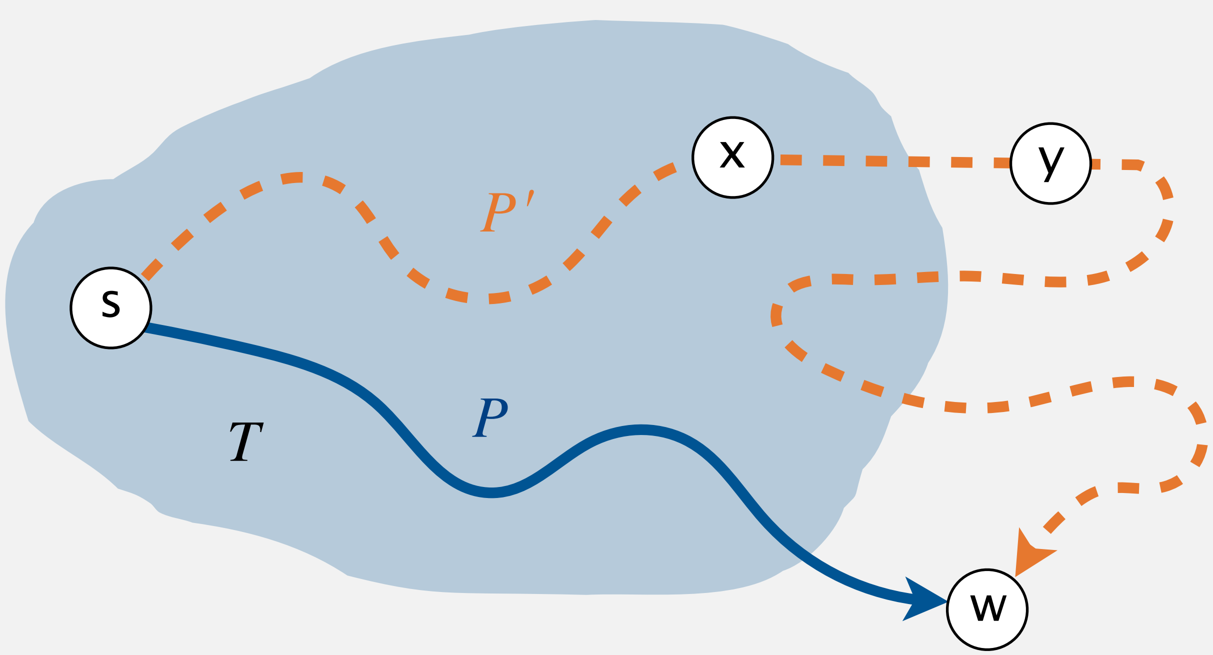
Dijkstra's algorithm: correctness proof

Invariant. For each vertex v in T, `distTo[v]` $= d^*(v)$.

length of shortest $s \rightarrow v$ path

Pf. [by induction on $|T|$]

- Let w be next vertex added to T.
- Let P be the $s \rightarrow w$ path of length `distTo[w]`.
- Consider any other $s \rightarrow w$ path P'.
- Let $x \rightarrow y$ be first edge in P' that leaves T.
- P' is no shorter than P :



by construction

$$\begin{aligned}
\text{length}(P) &= \texttt{distTo[w]} \\
\text{Dijkstra chose } w \text{ instead of } y \;\longrightarrow\; &\leq \texttt{distTo[y]} \\
\text{relax vertex } x \;\longrightarrow\; &\leq \texttt{distTo[x]} + \mathit{weight}(x, y) \\
\text{induction} \;\longrightarrow\; &= d^*(x) + \mathit{weight}(x, y) \\
\text{weights are non-negative} \;\longrightarrow\; &\leq \text{length}(P') \quad \blacksquare
\end{aligned}$$

Dijkstra's algorithm: correctness proof

Invariant. For each vertex v in T, `distTo[v]` $= d^*(v)$.

length of shortest $s \rightarrow v$ path

Corollary. Dijkstra's algorithm computes shortest path distances.

Pf. Upon termination, T contains all vertices (reachable from s).

Dijkstra's algorithm: Java implementation

```
public class DijkstraSP
{
   private DirectedEdge[] edgeTo;
   private double[] distTo;
   private IndexMinPQ<Double> pq;

   public DijkstraSP(EdgeWeightedDigraph G, int s)
   {
      edgeTo = new DirectedEdge[G.V()];
      distTo = new double[G.V()];
      pq = new IndexMinPQ<Double>(G.V());

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

      pq.insert(s, 0.0);
      while (!pq.isEmpty())
      {
         int v = pq.delMin();
         for (DirectedEdge e : G.adj(v))
            relax(e);
      }
   }
}
```

relax vertices in order of distance from s

Dijkstra's algorithm: Java implementation

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
       distTo[w] = distTo[v] + e.weight();
       edgeTo[w] = e;
       if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
       else                pq.insert     (w, distTo[w]);
   }
}
```

update PQ

Indexed priority queue [see Section 2.4 of textbook for details]

Associate an index between 0 and $n-1$ with each key in a priority queue.

- Insert a key associated with a given index.
- Delete a minimum key and return associated index.
- Decrease the key associated with a given index.

`public class IndexMinPQ<Key extends Comparable<Key>>`

	`IndexMinPQ(int n)`	*create indexed PQ with indices* $0, 1, \ldots, n-1$
`void`	`insert(int i, Key key)`	*associate key with index i*
`int`	`delMin()`	*remove a minimal key and return its associated index*
`void`	`decreaseKey(int i, Key key)`	*decrease the key associated with index i*
`boolean`	`contains(int i)`	*is i an index on the priority queue?*
`boolean`	`isEmpty()`	*is the priority queue empty?*
`int`	`size()`	*number of keys in the priority queue*

Shortest paths: quiz 3

What is the order of growth of the running time of Dijkstra's algorithm in the worst case when using a binary heap for the priority queue?

A. $V + E$

B. $V \log V$

C. $E \log V$

D. $E \log E$

Dijkstra's algorithm: which priority queue?

Depends on PQ implementation: V INSERT, V DELETE-MIN, $\leq E$ DECREASE-KEY.

PQ implementation	INSERT	DELETE-MIN	DECREASE-KEY	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap	$1^{\dagger}$	$\log V^{\dagger}$	$1^{\dagger}$	$E + V \log V$

† amortized

Bottom line.

- Array implementation optimal for complete graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.

Priority-first search

Insight. Four of our graph-search methods are the same algorithm!

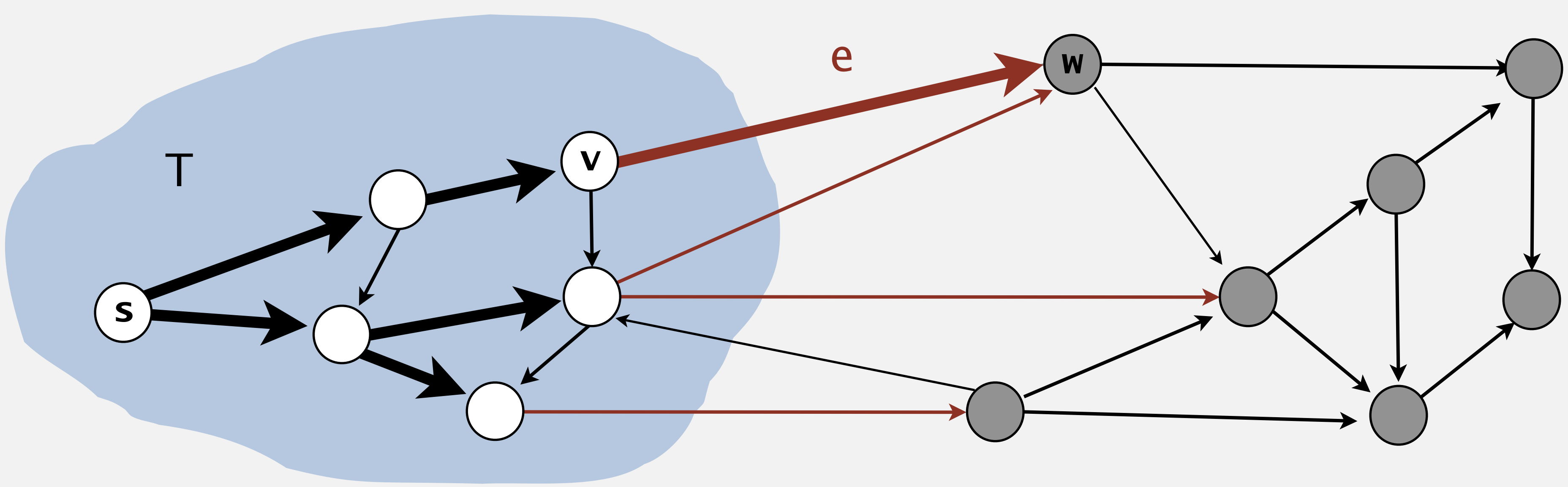
- Maintain a tree of explored vertices T.
- Grow T by exploring edges with exactly one endpoint leaving T.

DFS. Take edge from vertex which was discovered most recently.

BFS. Take edge from vertex which was discovered least recently.

Prim. Take edge of minimum weight.

Dijkstra. Take edge to vertex that is closest to s.



Each algorithm results in a tree of paths from the source node:

DFS tree / BFS tree / Minimal Spanning Tree / Shortest-Paths Tree.

Algorithm for shortest paths

Variations on a theme: vertex relaxations.

- Bellman–Ford: relax all vertices; repeat $V - 1$ times.
- Dijkstra: relax vertices in order of distance from s.
- Topological sort: relax vertices in topological order.

algorithm	worst-case running time	negative weights †	directed cycles
Bellman-Ford	$E V$	✔	✔
Dijkstra	$E \log V$		✔
topological sort	E	✔	

† no negative cycles

Design principle: pick algorithm based on known properties of input

Arbitrary graph (with no negative cycles)? Bellman-Ford.

Graph with no negative weights? Dijkstra.

DAG? Relax vertices in topological order.

Most specialized algorithm is usually (but not always) the fastest.

algorithm	worst-case running time	negative weights †	directed cycles
Bellman-Ford	$E\ V$	✔	✔
Dijkstra	$E \log V$		✔
topological sort	E	✔	

https://algs4.cs.princeton.edu

4.4 Shortest Paths

- APIs
- properties
- Bellman–Ford algorithm
- Dijkstra's algorithm
- seam carving

Content-aware resizing

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

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

Content-aware resizing

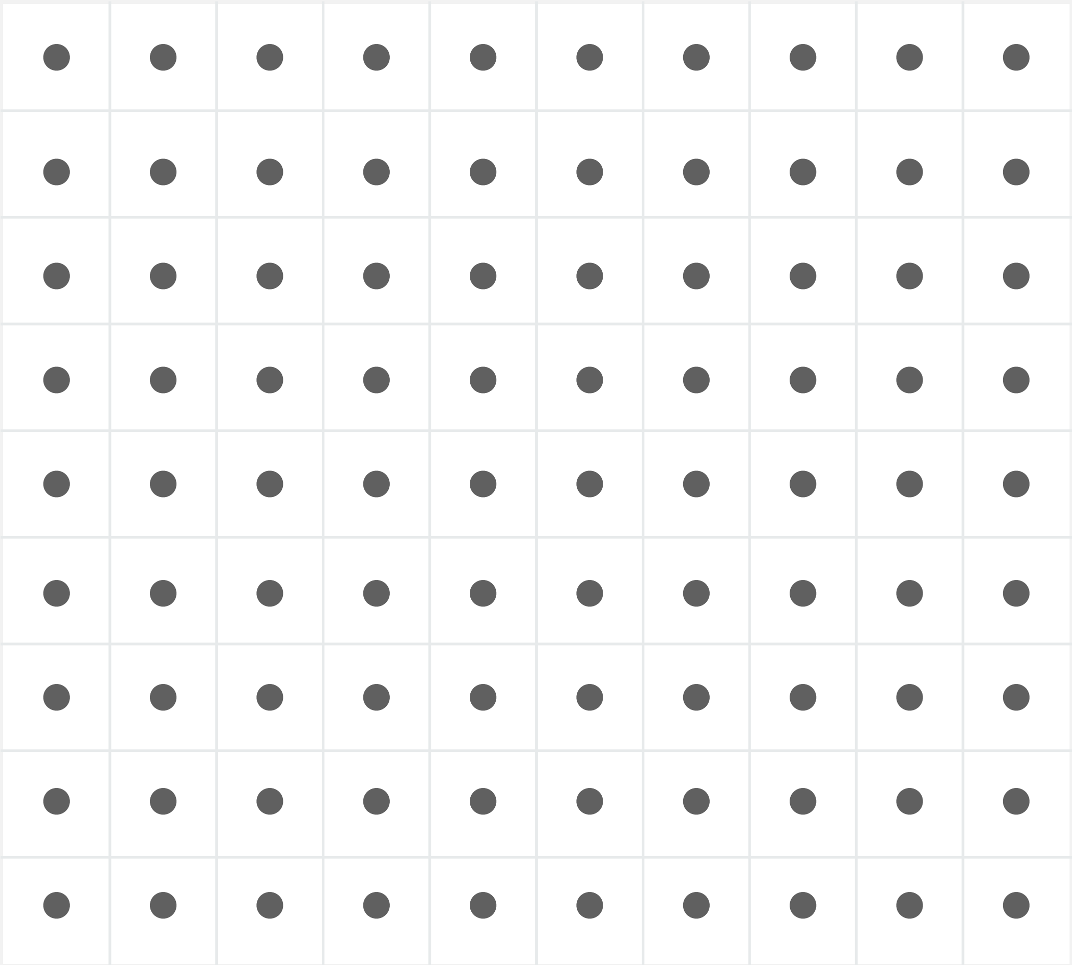
Seam carving. [Avidan–Shamir] Resize an image without distortion for display on cell phones and web browsers.

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

Content-aware resizing

To find vertical seam:

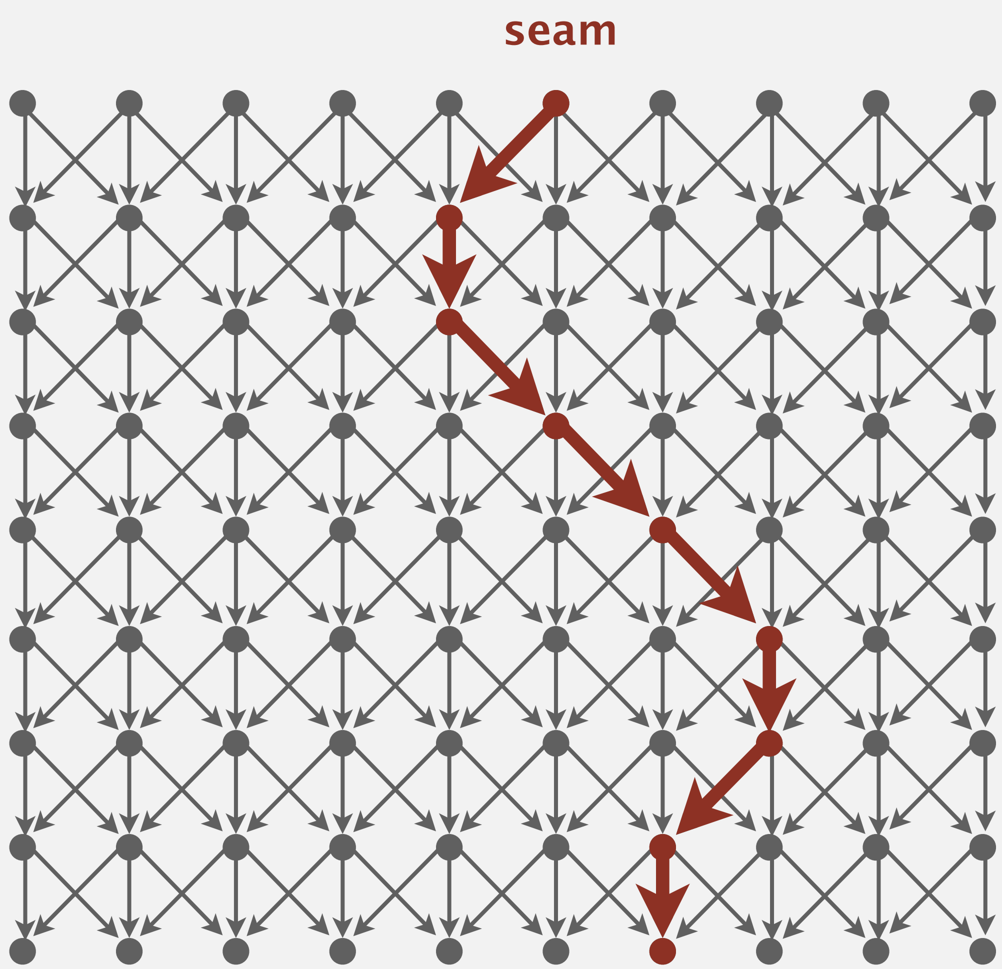
- Grid graph: vertex = pixel; edge = from pixel to 3 downward neighbors.
- Weight of pixel = "energy function" of 8 neighboring pixels.
- Seam = shortest path (sum of vertex weights) from top to bottom.



Content-aware resizing

To find vertical seam:

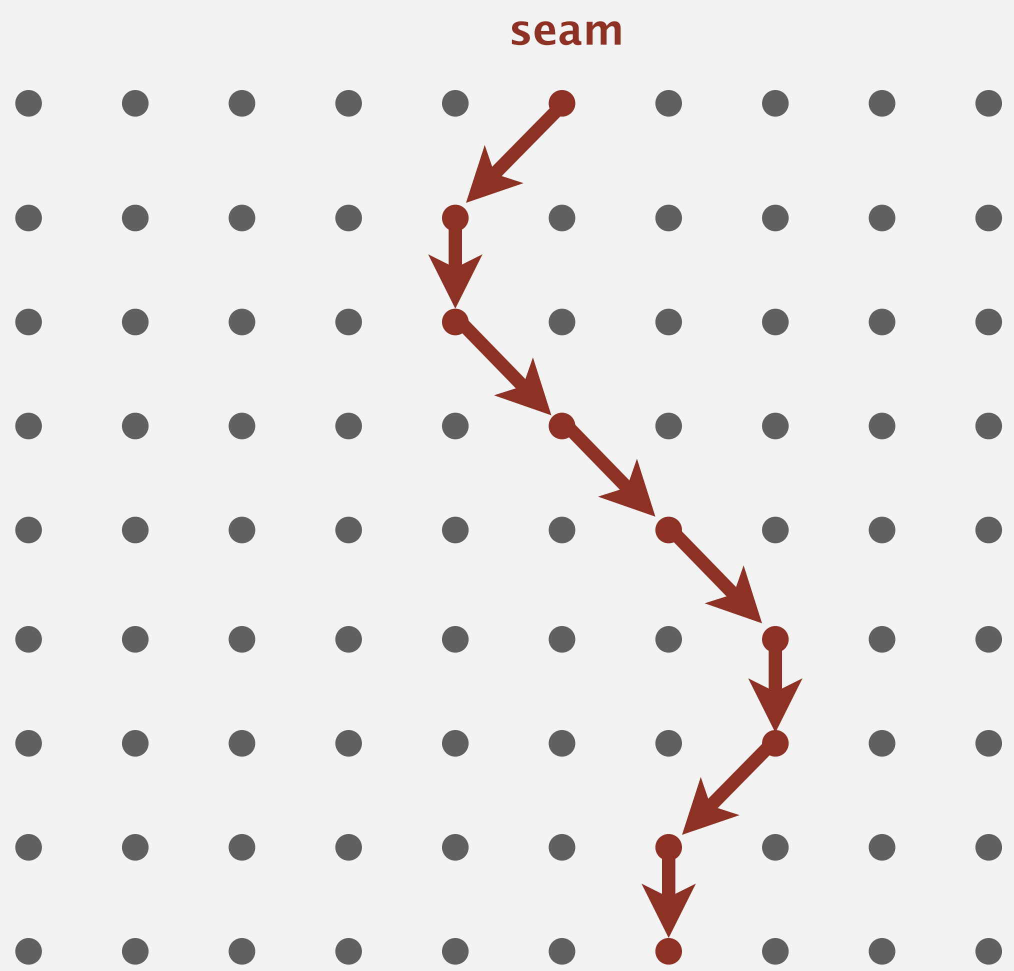
- Grid graph: vertex = pixel; edge = from pixel to 3 downward neighbors.
- Weight of pixel = "energy function" of 8 neighboring pixels.
- Seam = shortest path (sum of vertex weights) from top to bottom.



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).