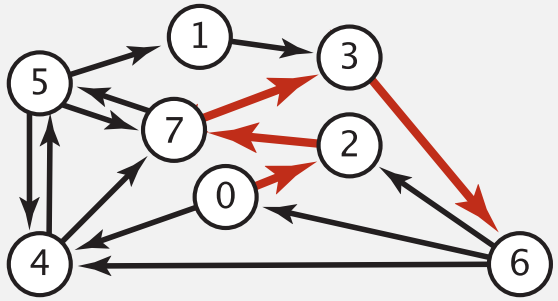
Shortest paths in an edge-weighted digraph

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

edge-weighted digraph

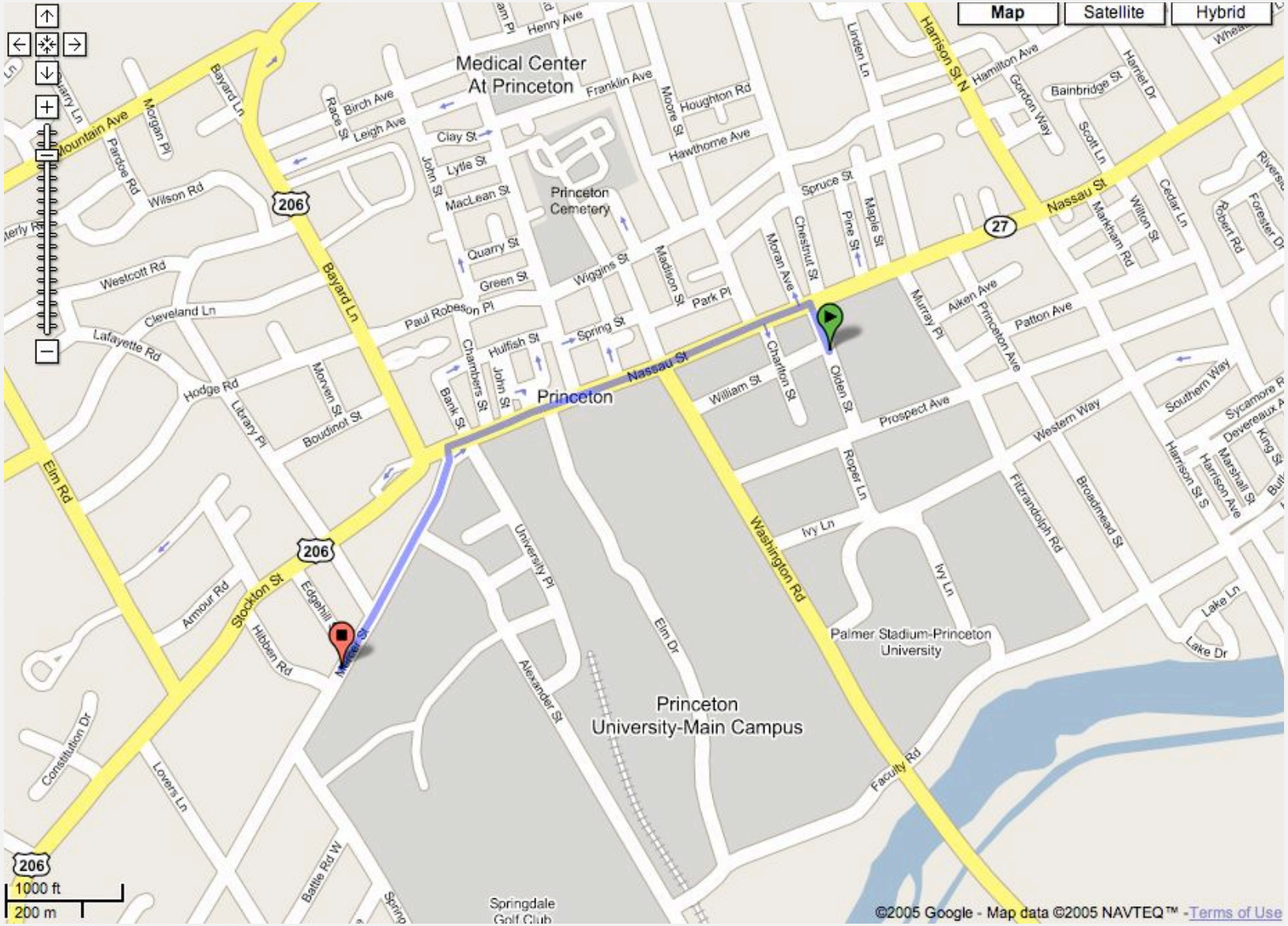
```
4->5  0.35
5->4  0.35
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```



shortest path from 0 to 6

```
0->2  0.26
2->7  0.34
7->3  0.39
3->6  0.52
```

Google maps


Map
Satellite
Hybrid
Medical Center
At Princeton
Princeton
Cemetery
Princeton
University-Main Campus
Palmer Stadium-Princeton
University
Springdale
Golf Club
Nassau St
Bayard Ln
Mountain Ave
Elm Rd
Stockton St
Washington Rd
Harrison St N
Franklin Ave
Houghton Rd
Hawthorne Ave
Clay St
Lytle St
MacLean St
Quarry St
Green St
Wiggins St
Spring St
Paul Robeson Pl
Hulfish St
Park Pl
Moore St
Madison St
Moran Ave
Chestnut St
Spruce St
Pine St
Maple St
Olden St
Murray Pl
Aiken Ave
Patton Ave
Princeton Ave
Prospect Ave
Western Way
Southern Way
Markham Rd
Wilton St
Cedar Ln
Scott Ln
Bainbridge St
Harriet Dr
Gordon Way
Hamilton Ave
Linden Ln
Robert Rd
Forester Dr
Sycamore Rd
Devereaux Ave
King St
Harrison St
Marshall St
Harrison Ave
Broadmead St
Fitzrandolph Rd
Ivy Ln
Roper Ln
William St
Charlton St
University Pl
Alexander St
Elm Dr
Faculty Rd
Lake Ln
Lake Dr
Edgehill St
Hibben Rd
Armour Rd
Constitution Dr
Lovers Ln
Battle Rd W
Library Pl
Hodge Rd
Lafayette Rd
Cleveland Ln
Westcott Rd
Wilson Rd
Pardee Rd
Morgan Pl
Race St
Birch Ave
Leigh Ave
Henry Ave
Bank St
Chambers St
John St
Morven St
Boudinot St
206
27
1000 ft
200 m
©2005 Google - Map data ©2005 NAVTEQ™ -Terms of Use

Car navigation

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in TeX.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- 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?

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

Restrictions on edge weights?

- Nonnegative weights.
- Arbitrary weights.
- Euclidean weights.

Cycles?

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

Simplifying assumption. Shortest paths from s to each vertex v exist.

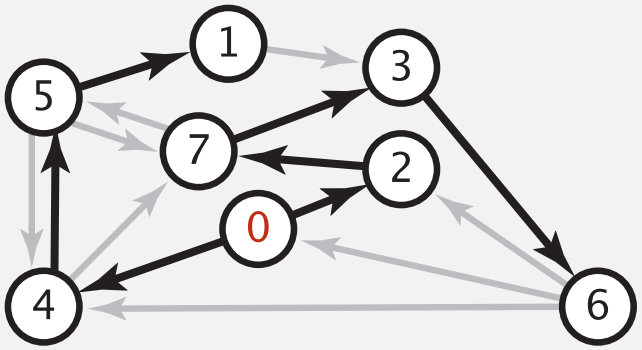
Data structures for single-source shortest paths

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

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

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

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



shortest-paths tree from 0

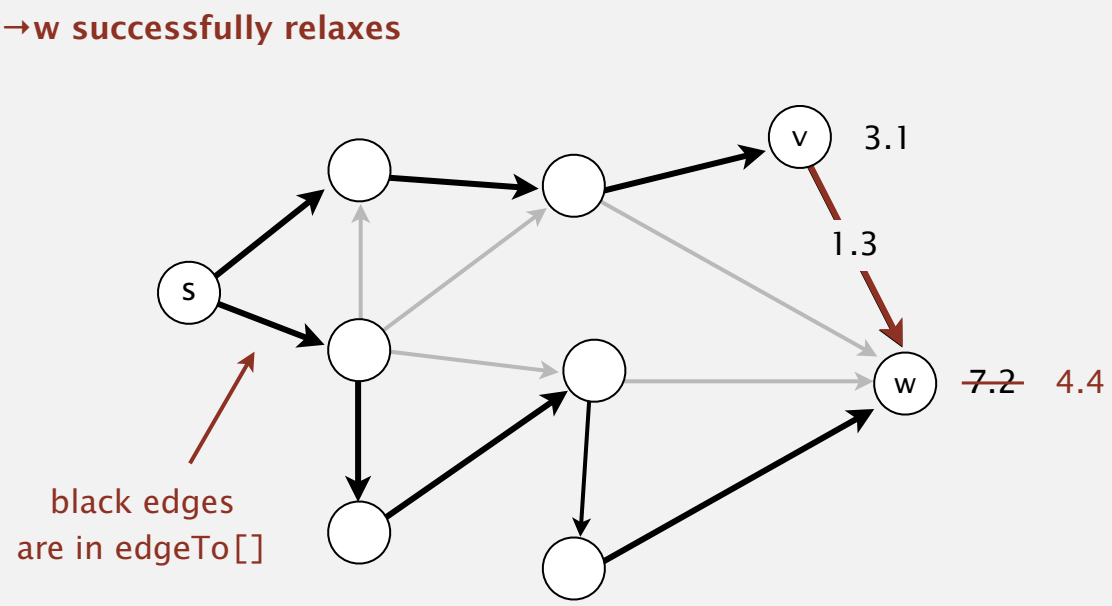
	edgeTo[]	distTo[]
0	null	0
1	5->1 0.32	1.05
2	0->2 0.26	0.26
3	7->3 0.37	0.97
4	0->4 0.38	0.38
5	4->5 0.35	0.73
6	3->6 0.52	1.49
7	2->7 0.34	0.60

parent-link representation

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` gives shorter path to `w` through `v`,
  update both `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` gives shorter path to `w` through `v`,
  update both `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 optimality conditions

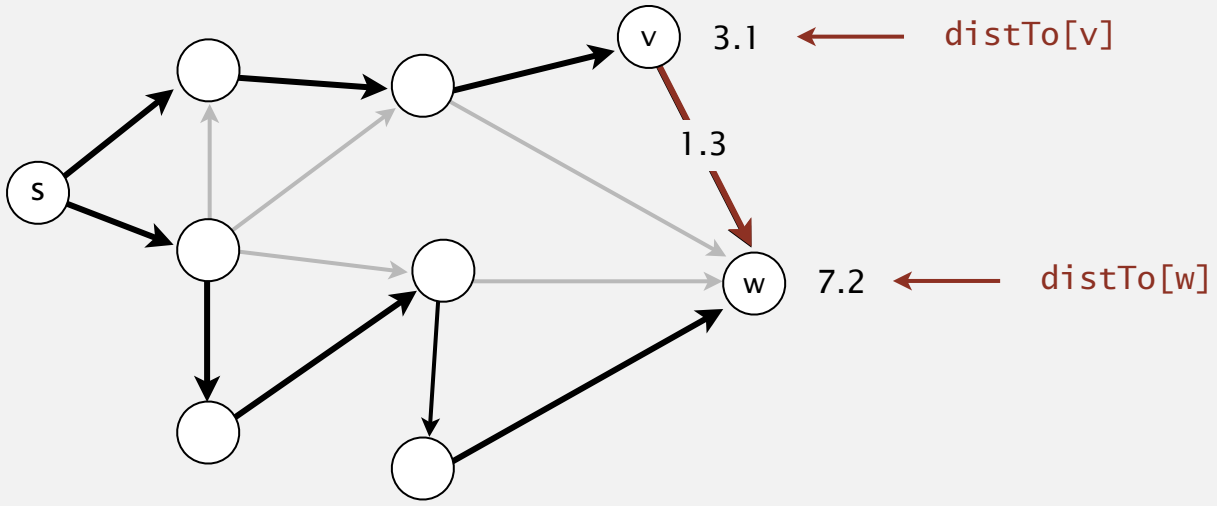
Proposition. Let G be an edge-weighted digraph.

Then `distTo[]` are the shortest path distances from `s` iff:

- For each vertex `v`, `distTo[v]` is the length of some path from `s` to `v`.
- For each edge `e` = `v`→`w`, `distTo[w]` ≤ `distTo[v]` + `e.weight()`.

Pf. ⇐ [necessary]

- Suppose that `distTo[w]` > `distTo[v]` + `e.weight()` for some edge `e` = `v`→`w`.
- Then, `e` gives a path from `s` to `w` (through `v`) of length less than `distTo[w]`.



Shortest-paths optimality conditions

Proposition. Let G be an edge-weighted digraph.

Then `distTo[]` are the shortest path distances from `s` iff:

- For each vertex `v`, `distTo[v]` is the length of some path from `s` to `v`.
- For each edge `e` = `v`→`w`, `distTo[w]` ≤ `distTo[v]` + `e.weight()`.

Pf. ⇒ [sufficient]

- Suppose that $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = w$ is a shortest path from s to w.
- Then,

$$\begin{aligned} \texttt{distTo}[v_1] &\le \texttt{distTo}[v_0] + \texttt{e}_1\texttt{.weight()} \\ \texttt{distTo}[v_2] &\le \texttt{distTo}[v_1] + \texttt{e}_2\texttt{.weight()} \\ &\ldots \\ \texttt{distTo}[v_k] &\le \texttt{distTo}[v_{k-1}] + \texttt{e}_k\texttt{.weight()} \end{aligned}$$

e_i = i^{th} edge on shortest path from s to w

- Add inequalities; simplify; and substitute `distTo[v0]` = `distTo[s]` = 0:

$$\texttt{distTo}[w] = \texttt{distTo}[v_k] \le \underbrace{\texttt{e}_1\texttt{.weight()} + \texttt{e}_2\texttt{.weight()} + \ldots + \texttt{e}_k\texttt{.weight()}}_{\text{weight of shortest path from s to w}}$$

- Thus, `distTo[w]` is the weight of shortest path to w. ▪

weight of some path from s to w

Generic shortest-paths algorithm

Generic algorithm (to compute SPT from s)

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

Repeat until optimality conditions are satisfied:

- **Relax any edge.**

Proposition. Generic algorithm computes SPT (if it exists) from `s`.

Pf sketch.

- Throughout algorithm, `distTo[v]` is the length of a simple path from `s` to v (and `edgeTo[v]` is last edge on path).
- Each successful relaxation decreases `distTo[v]` for some `v`.
- The entry `distTo[v]` can decrease at most a finite number of times. ▪

Generic shortest-paths algorithm

Generic algorithm (to compute SPT from s)

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

Repeat until optimality conditions are satisfied:

- **Relax any edge.**

Efficient implementations. How to choose which edge to relax?

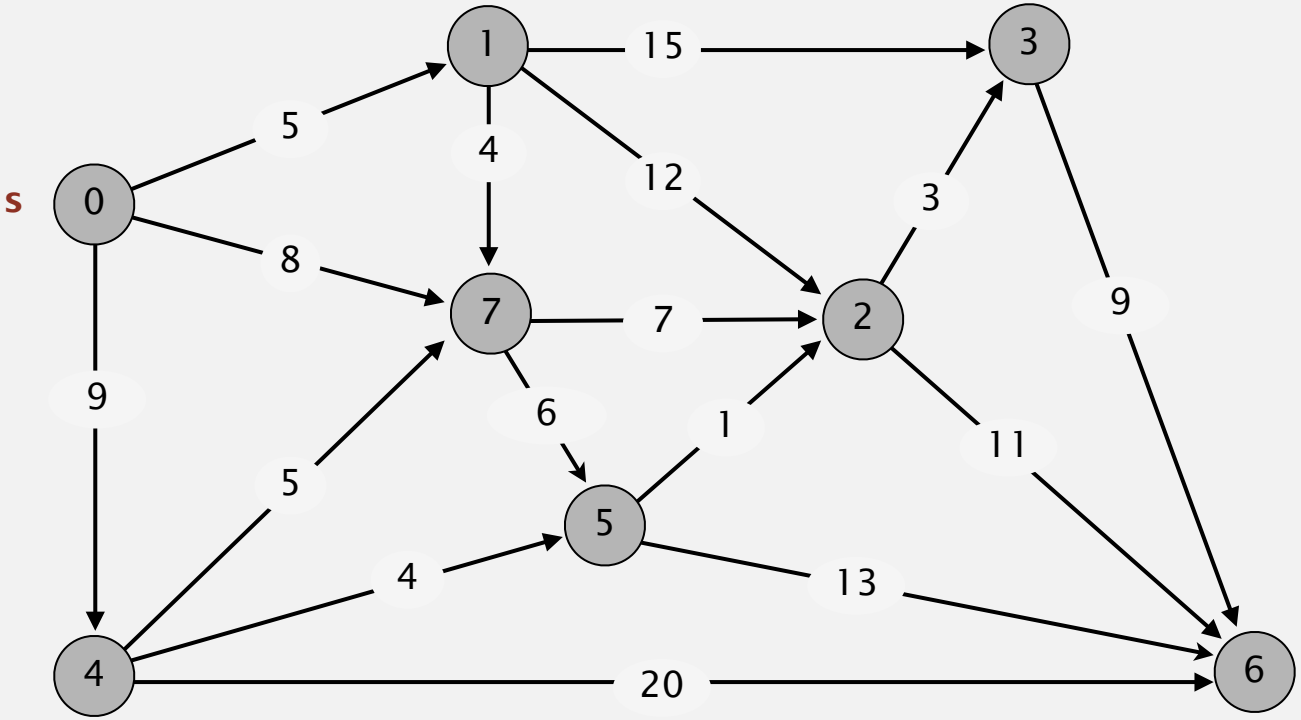
Ex 1. Dijkstra's algorithm (nonnegative weights).

Ex 2. Topological sort algorithm (no directed cycles).

Ex 3. Bellman-Ford algorithm (no negative cycles).

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 pointing from that vertex.

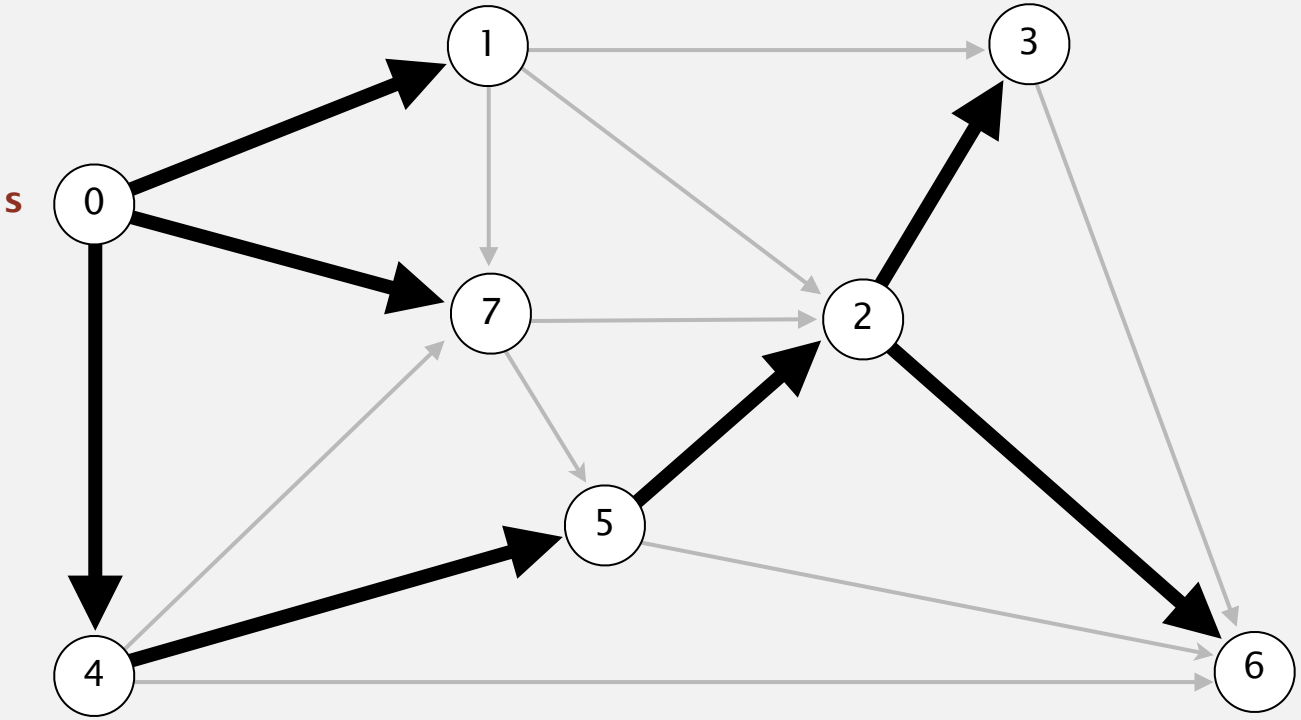


an edge-weighted digraph

```
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 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 pointing from that vertex.



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

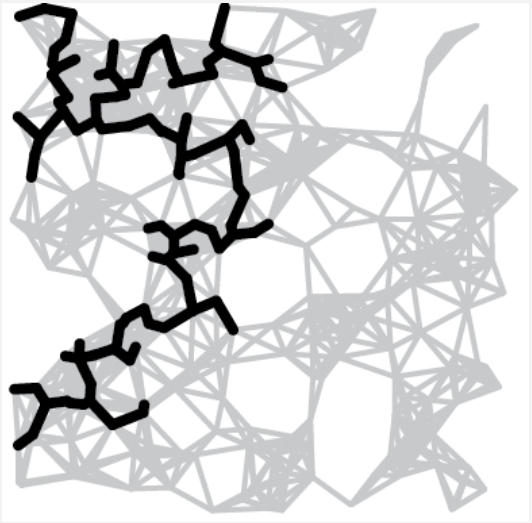
Computing spanning trees in graphs

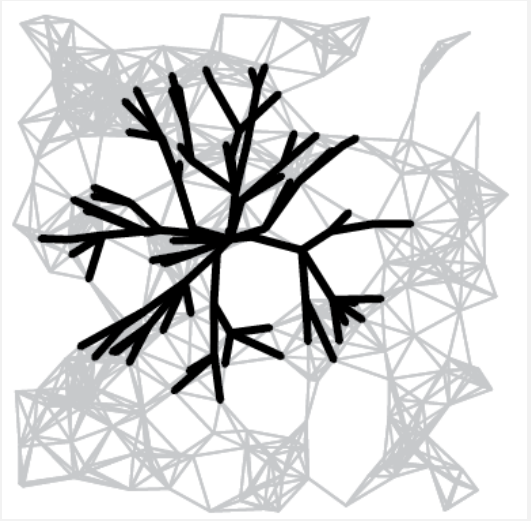
Dijkstra's algorithm seem familiar?

- Prim's algorithm is essentially the same algorithm.
- Both are in a family of algorithms that compute a graph's spanning tree.

Main distinction: Rule used to choose next vertex for the tree.

- Prim's: Closest vertex to the tree (via an undirected edge).
- Dijkstra's: Closest vertex to the source (via a directed path).





Note: DFS and BFS are also in this family of algorithms.

Dijkstra's algorithm: which priority queue?

Depends on PQ implementation: V insert, V delete-min, 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 (Johnson 1975)	d $\log_d$ V	d $\log_d$ V	$\log_d$ V	E $\log_{E/V}$ V
Fibonacci heap (Fredman-Tarjan 1984)	1 †	log V †	1 †	E + V log V

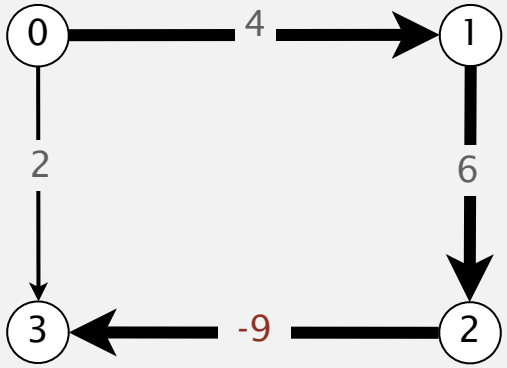
† amortized

Bottom line.

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

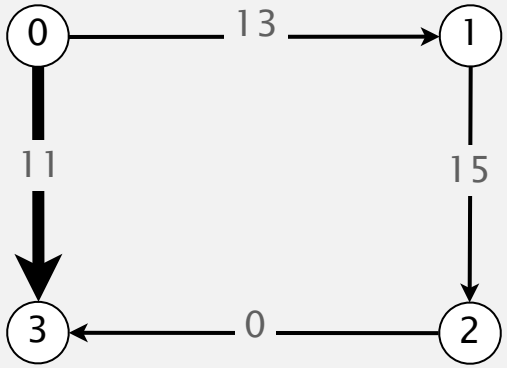
Shortest paths with negative weights: failed attempts

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.

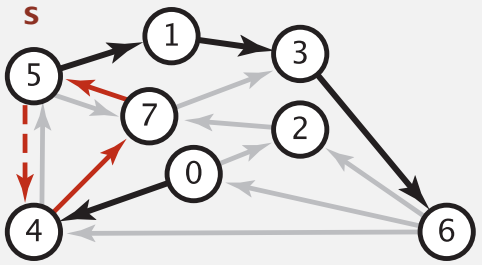
Conclusion. Need a different algorithm.

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:

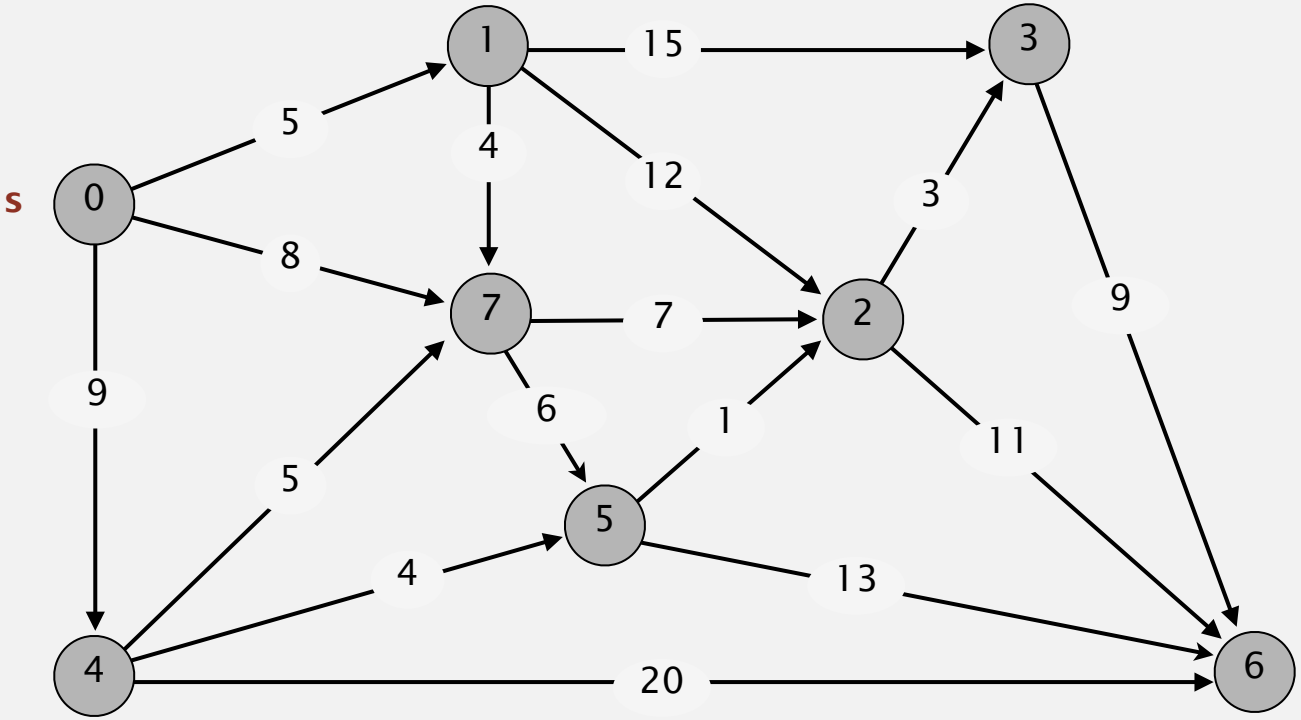
- **Relax each edge.**

```
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.

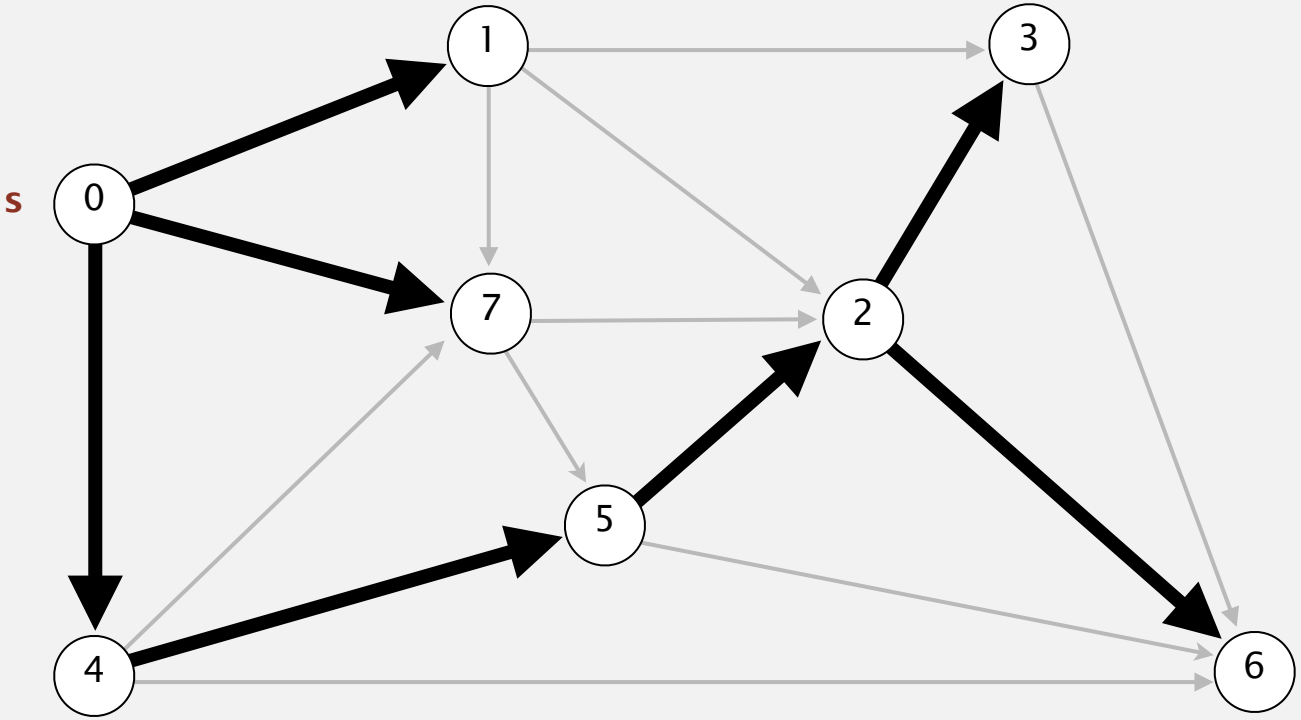


an edge-weighted digraph

```
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.



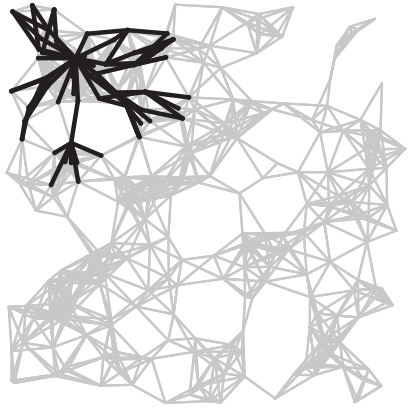
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

shortest-paths tree from vertex s

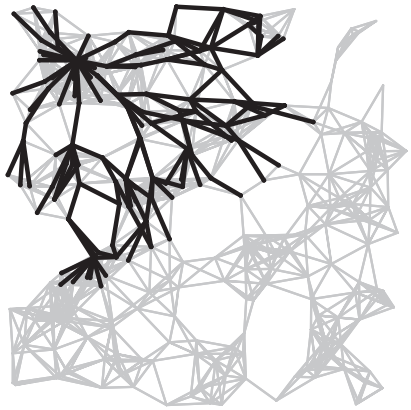
Bellman-Ford algorithm visualization

passes

4



7



10

13

SPT

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.

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.