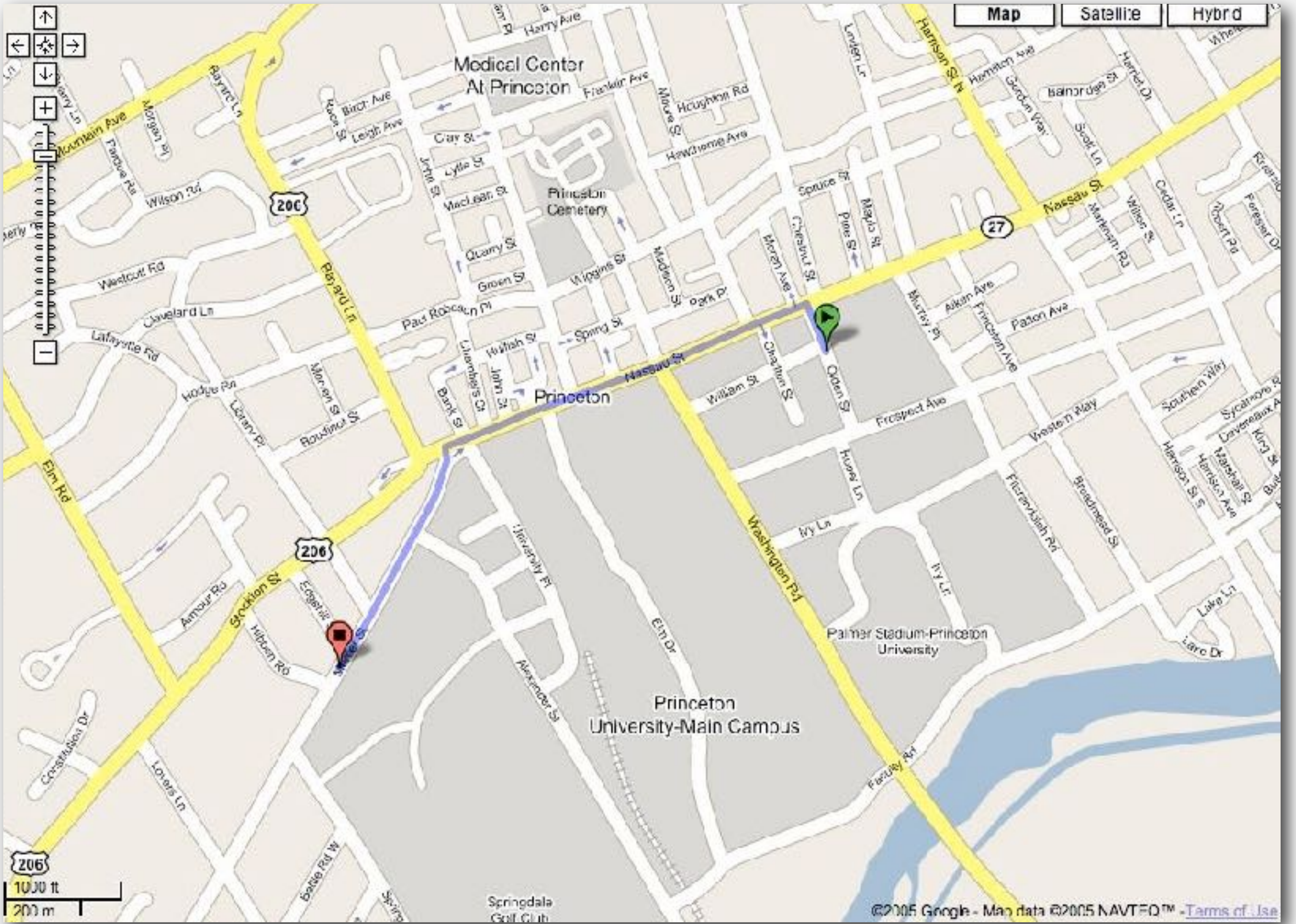
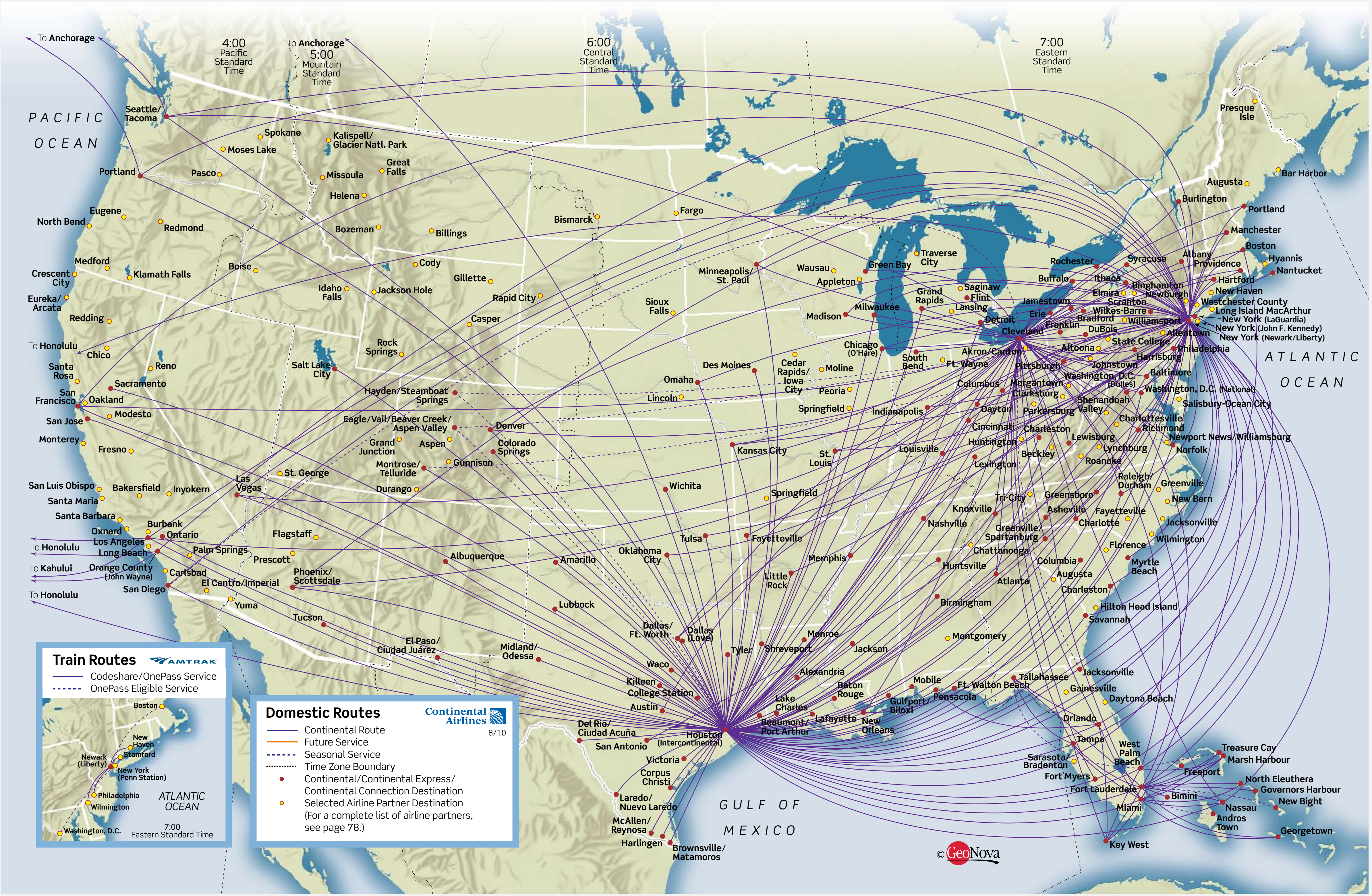
4.4 Shortest Paths

- edge-weighted digraph API
- shortest-paths properties
- Dijkstra's algorithm
- edge-weighted DAGs
- negative weights

Google maps

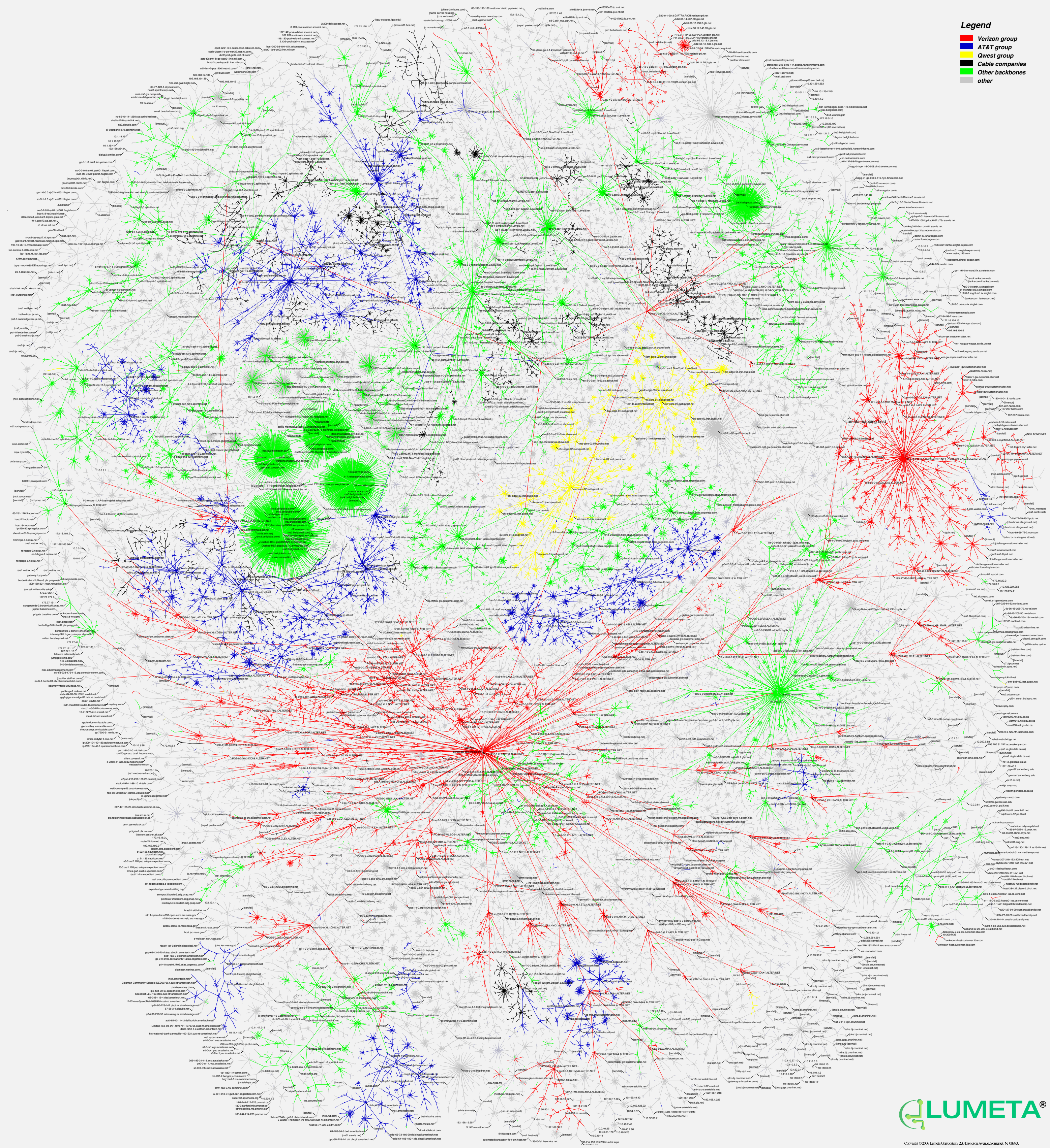

Map
Satellite
Hybrid
Medical Center
At Princeton
Princeton
Cemetery
Princeton
Nassau St
Washington Rd
Stockton St
Elm Rd
Bayard Ln
Mountain Ave
Harrison St N
Palmer Stadium-Princeton
University
Princeton
University-Main Campus
Springdale
Golf Club
206
27
1000 ft
200 m
©2005 Google - Map data ©2005 NAVTEQ™ - Terms of Use

Continental U.S. routes (August 2010)



http://www.continental.com/web/en-US/content/travel/routes

Shortest outgoing routes on the Internet from Lumeta headquarters



- Verizon group
- AT&T group
- Qwest group
- Cable companies
- Other backbones
- other

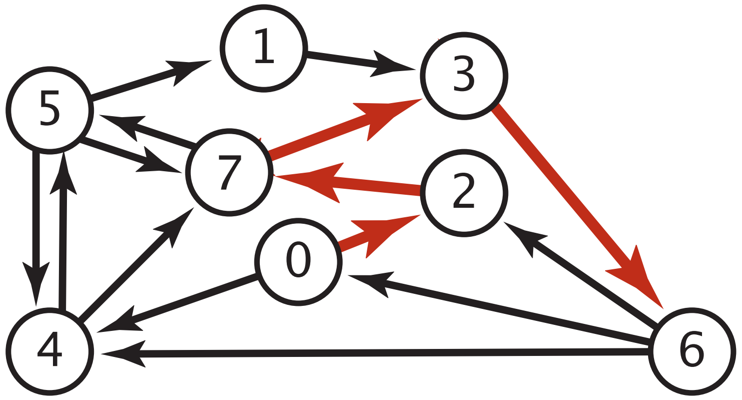
map by Lumeta Corporation, March 8, 2006

Shortest paths in a weighted digraph

Given an edge-weighted digraph, find the shortest (directed) 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

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 cycles.
- No "negative cycles."

Simplifying assumption. There exists a shortest path from s to each vertex v.

Shortest path applications

- Map routing.
- Robot navigation.
- Texture mapping.
- Typesetting in TeX.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Subroutine in advanced algorithms.
- Routing of telecommunications messages.
- Approximating piecewise linear functions.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.

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

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

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 int weight()
   {  return weight; }
}
```

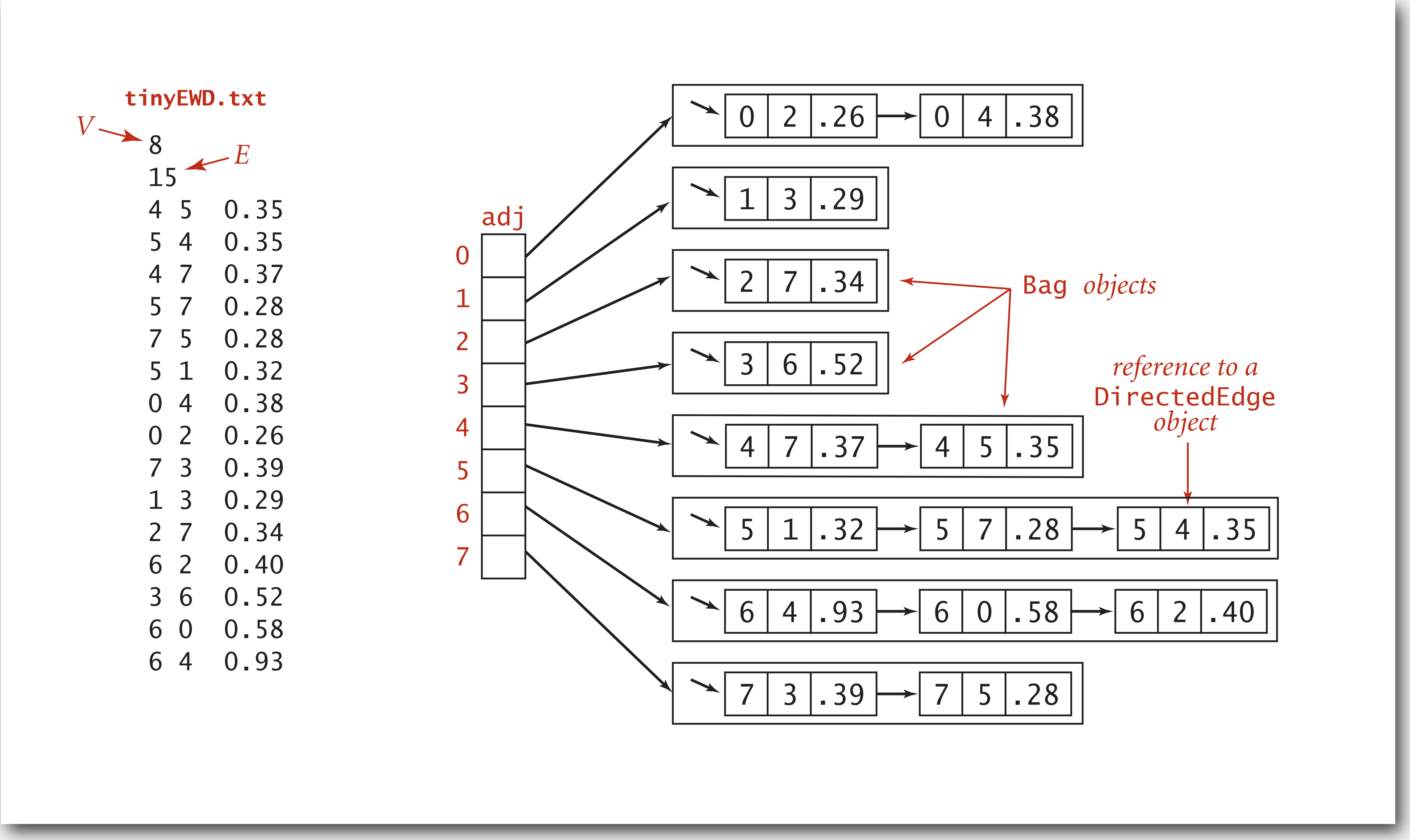
`from()` and `to()` replace `either()` and `other()`

Edge-weighted digraph API

`public class EdgeWeightedDigraph`		
	`EdgeWeightedDigraph(int V)`	*edge-weighted digraph with V vertices*
	`EdgeWeightedDigraph(In in)`	*edge-weighted digraph from input stream*
`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 in this digraph*
`String`	`toString()`	*string representation*

Conventions. Allow self-loops and parallel edges.

Edge-weighted digraph: adjacency-lists representation



Edge-weighted digraph: adjacency-lists implementation in Java

Same as `EdgeWeightedGraph` except replace `Graph` with `Digraph`.

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

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

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

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

similar to edge-weighted undirected graph, but only add edge to 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 graph 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?*

```
SP sp = new SP(G, s);
for (int v = 0; v < G.V(); v++)
{
   StdOut.printf("%d to %d (%.2f): ", s, v, sp.distTo(v));
   for (DirectedEdge e : sp.pathTo(v))
      StdOut.print(e + "  ");
   StdOut.println();
}
```

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 graph 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?*

```
% java SP tinyEWD.txt 0
0 to 0 (0.00):
0 to 1 (1.05): 0->4 0.38  4->5 0.35  5->1 0.32
0 to 2 (0.26): 0->2 0.26
0 to 3 (0.99): 0->2 0.26  2->7 0.34  7->3 0.39
0 to 4 (0.38): 0->4 0.38
0 to 5 (0.73): 0->4 0.38  4->5 0.35
0 to 6 (1.51): 0->2 0.26  2->7 0.34  7->3 0.39  3->6 0.52
0 to 7 (0.60): 0->2 0.26  2->7 0.34
```

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

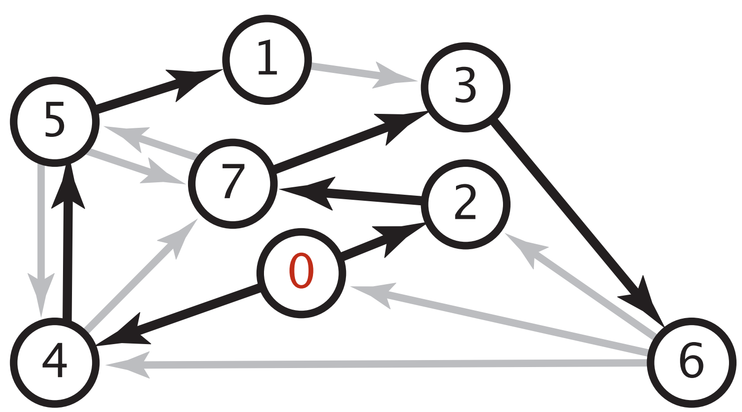
Data structures for single-source shortest paths

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

Observation. A shortest path 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.



	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

shortest path tree from 0

Data structures for single-source shortest paths

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

Observation. A shortest path 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.

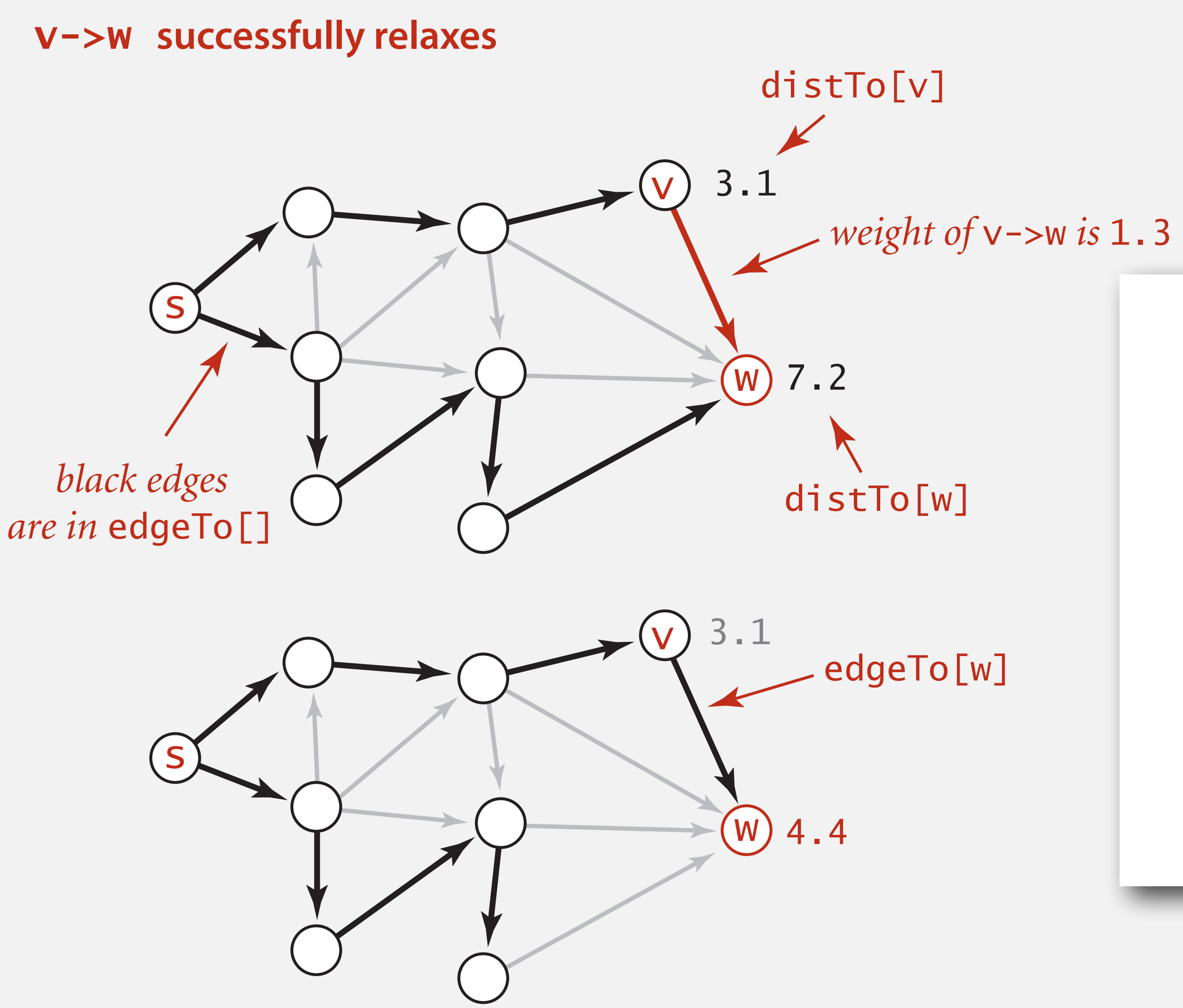
```
public double distTo(int v)
{  return distTo[v];  }
```

```
public Iterable<DirectedEdge> pathTo(int v)
{
   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 $e = v \to w$.

- `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 $e = v \to w$ gives shorter path to w through v, 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 optimality conditions

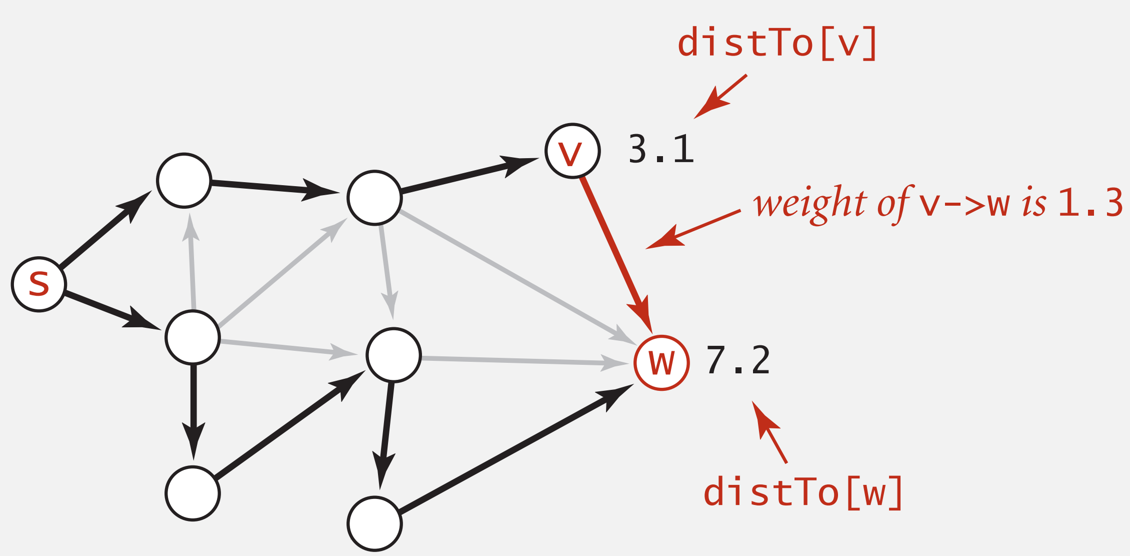
Proposition. Let G be an edge-weighted digraph.

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 $e = v \rightarrow w$, `distTo[w]` ≤ `distTo[v] + e.weight()`.

Pf. ⇐ [necessary]

- Suppose that `distTo[w] > distTo[v] + e.weight()` for some edge $e = v \rightarrow w$.
- 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 $e = v \rightarrow w$, `distTo[w] ≤ distTo[v] + e.weight()`.

Pf. $\Rightarrow$ [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,

```
distTo[vk]     ≤  distTo[vk-1]  + ek.weight()
distTo[vk-1]   ≤  distTo[vk-2]  + ek-1.weight()
...
distTo[v1]     ≤  distTo[v0]    + e1.weight()
```

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

- Collapsing these inequalities and eliminate `distTo[v0] = distTo[s] = 0`:

$$\texttt{distTo[w]} = \texttt{distTo[}v_k\texttt{]} \le \underbrace{e_k\texttt{.weight()} + e_{k-1}\texttt{.weight()} + \ldots + e_1\texttt{.weight()}}_{\text{weight of shortest path from s to w}}$$

weight of some path from s to w

- Thus, `distTo[w]` is the weight of shortest path 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 from s. ← assuming SPT exists

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).

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

Edsger W. Dijkstra: select quotes

" Do only what only you can do. "

" In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind. "

" The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence. "

" 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. "

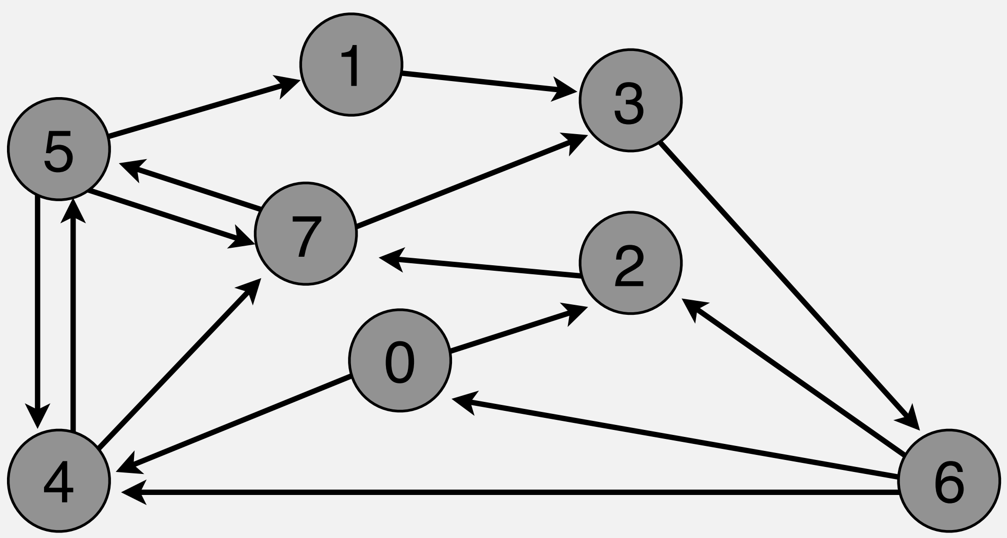
" APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums. "

Edsger W. Dijkstra
Turing award 1972

Dijkstra's algorithm

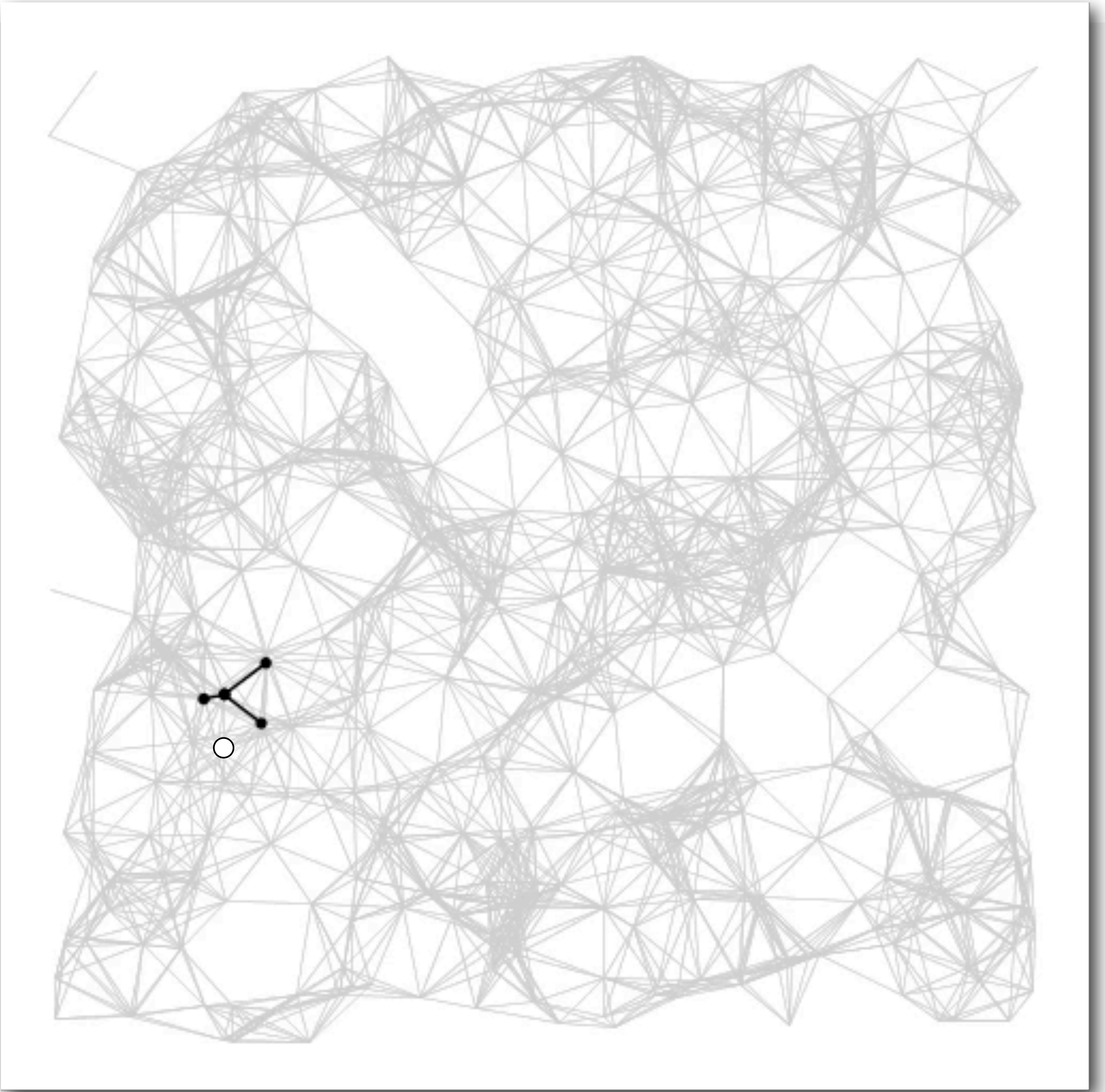
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

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



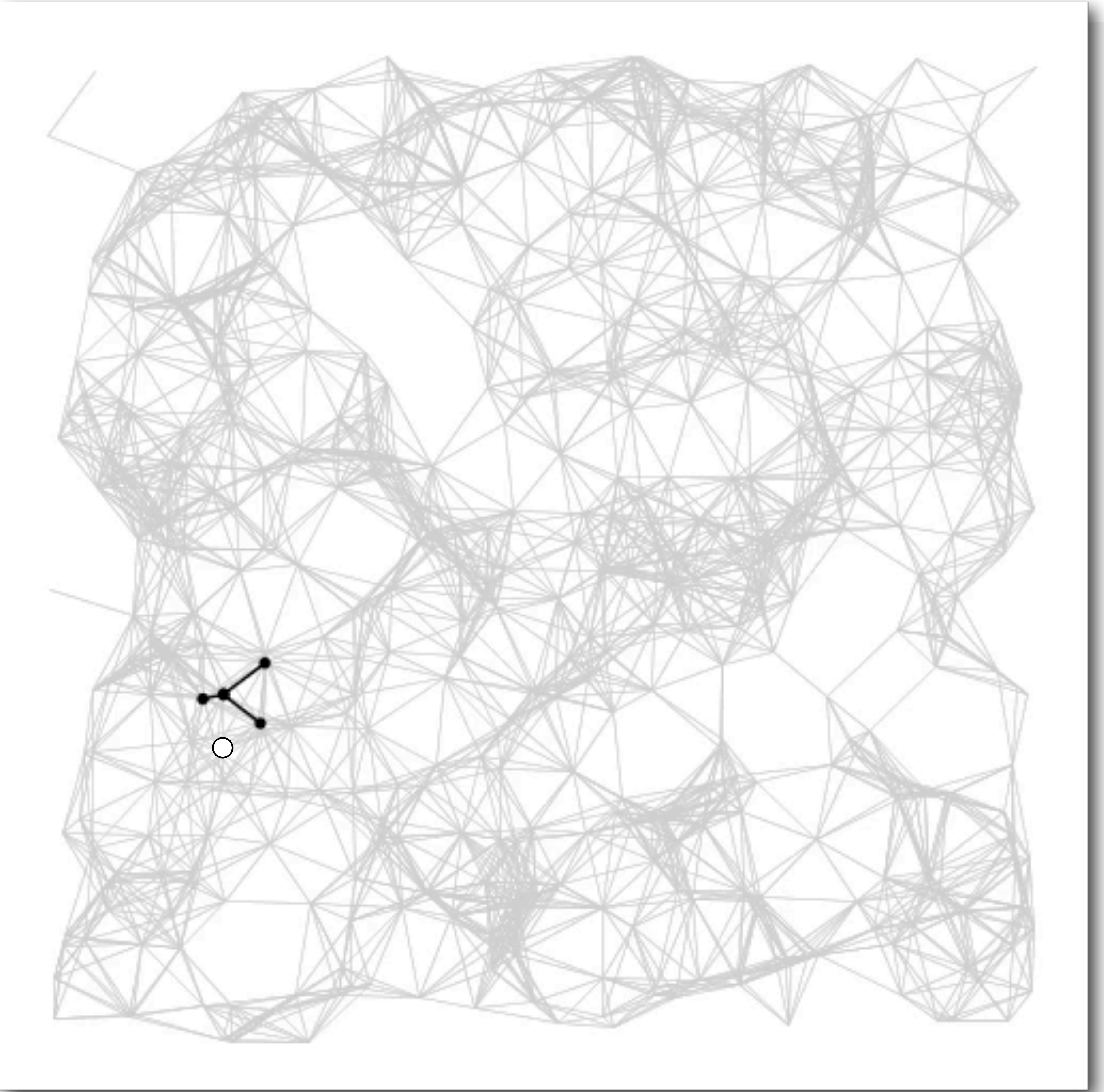
v	distTo[v]	edgeTo[v]
0	0.00	-
1		
2		
3		
4		
5		
6		
7		

Dijkstra's algorithm visualization



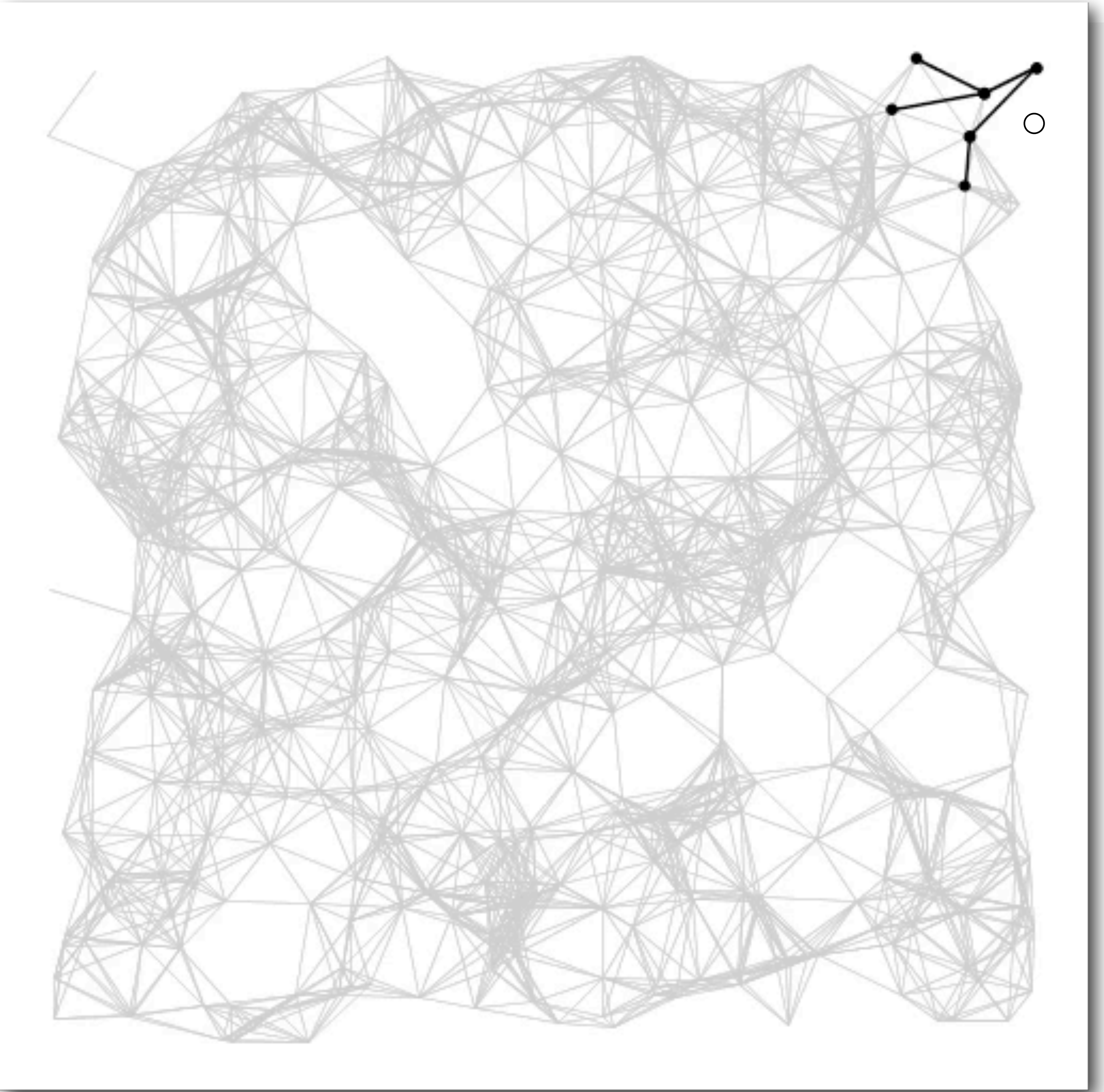
https://www.cs.purdue.edu/homes/cs251/slides/media/Dijkstra0.mov

Dijkstra's algorithm visualization



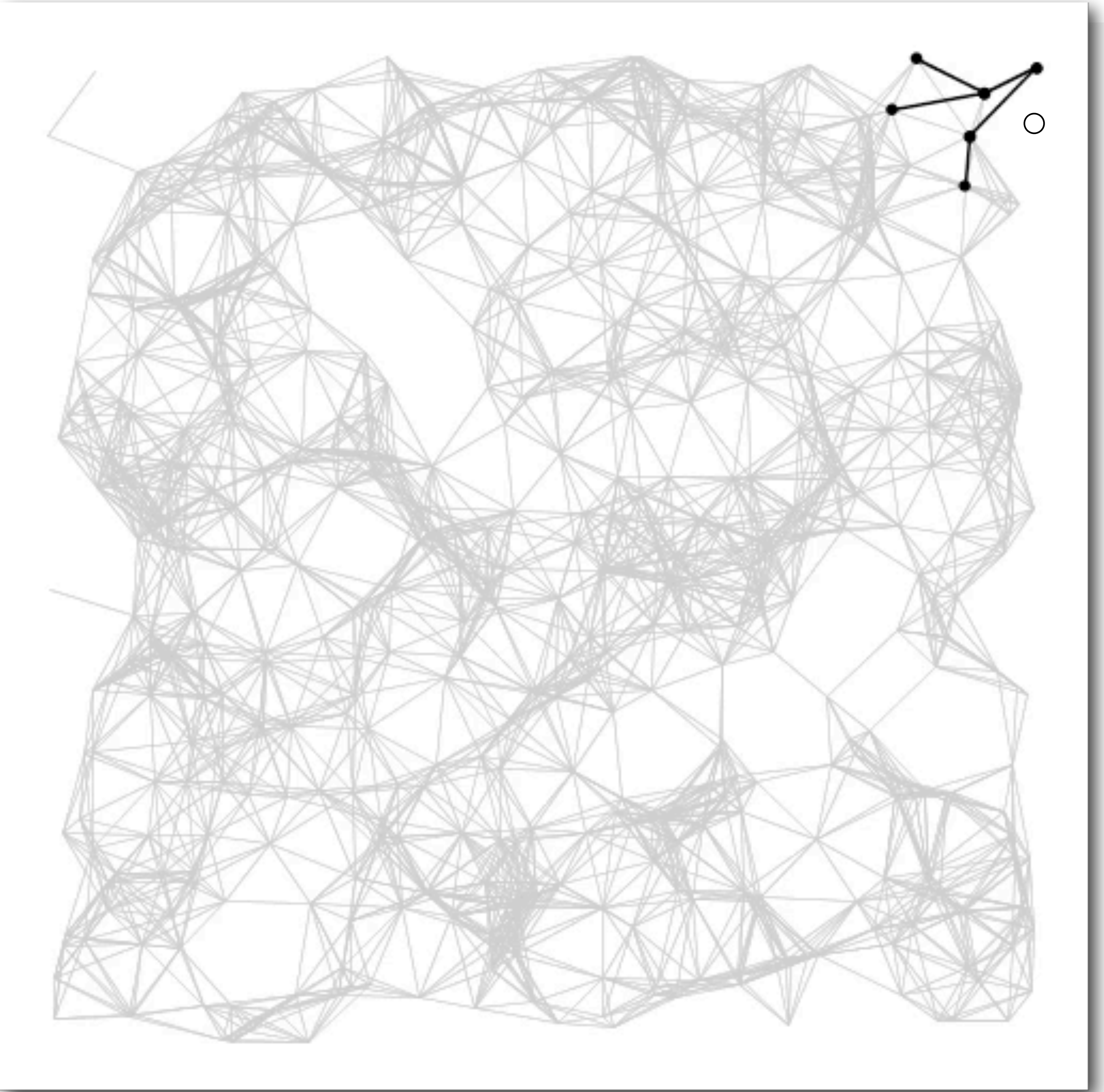
https://www.cs.purdue.edu/homes/cs251/slides/media/Dijkstra0.mov

Dijkstra's algorithm visualization



https://www.cs.purdue.edu/homes/cs251/slides/media/Dijkstra1.mov

Dijkstra's algorithm visualization

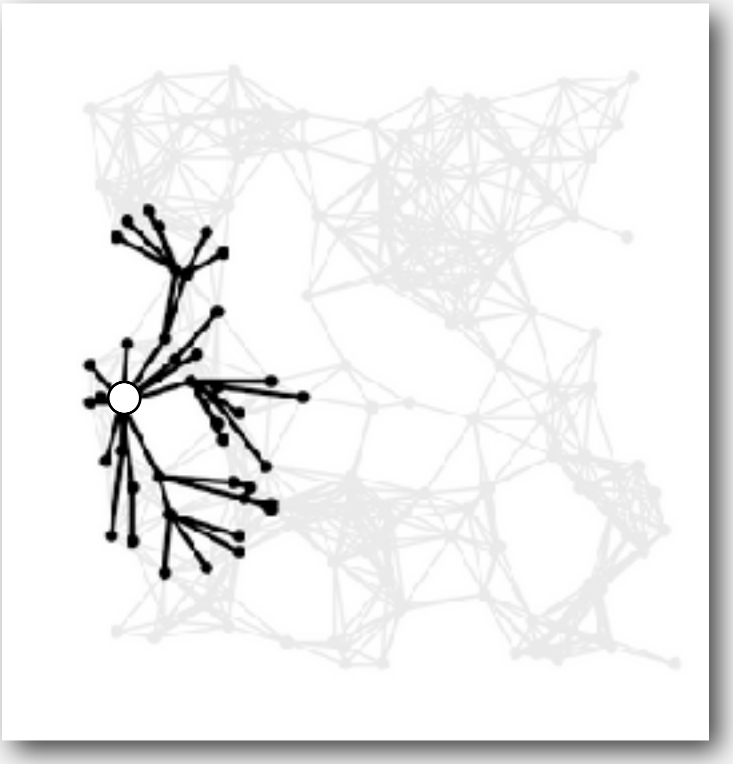


https://www.cs.purdue.edu/homes/cs251/slides/media/Dijkstra1.mov

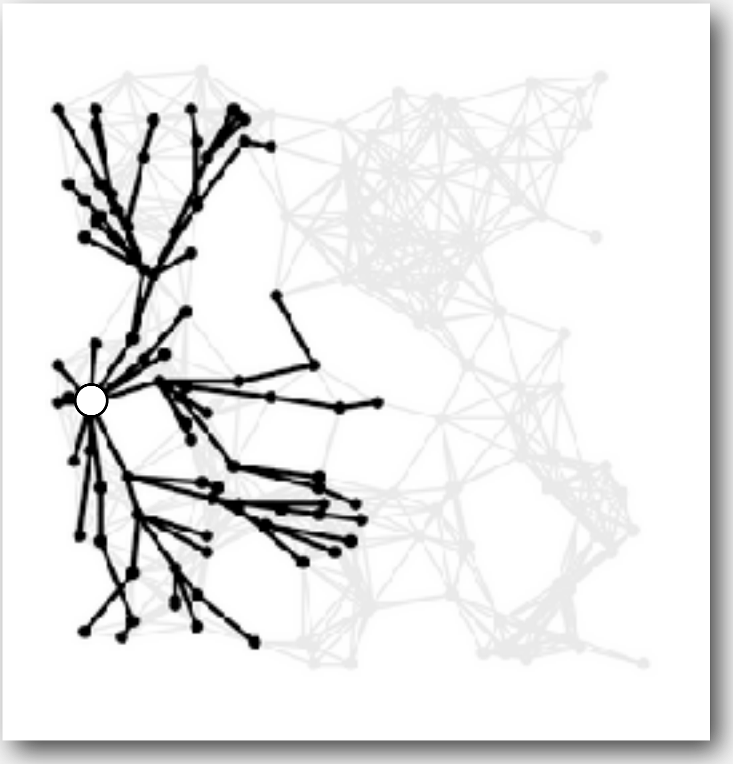
Shortest path trees

- 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 incident from that vertex.

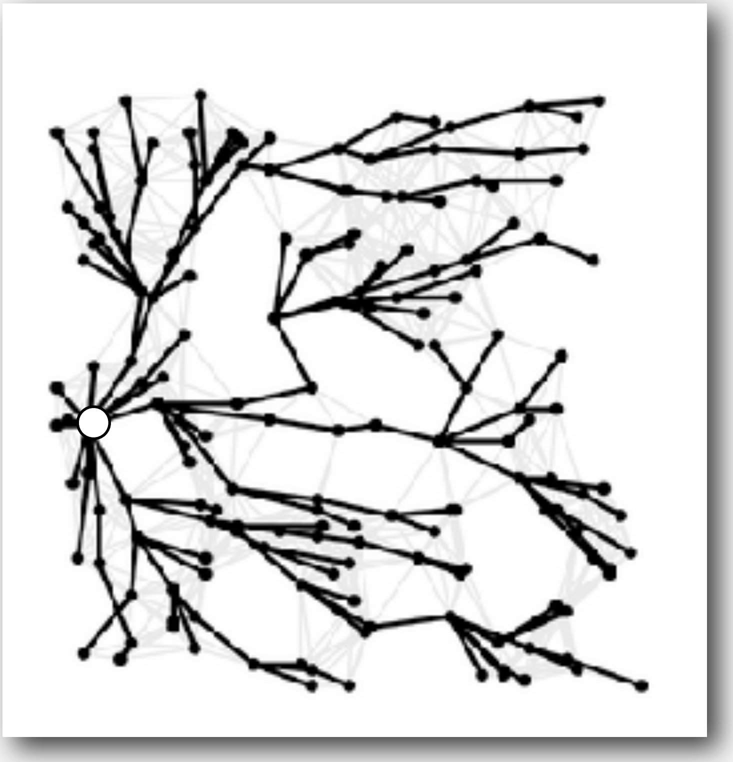
25%



50%



100%



Dijkstra's algorithm: correctness proof

Proposition. Dijkstra's algorithm computes SPT in any edge-weighted digraph with nonnegative weights.

Pf.

- Each edge $e = v \rightarrow w$ 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 ← edge weights are nonnegative and we choose lowest `distTo[]` value at each step
- Thus, upon termination, shortest-paths optimality conditions hold. ▪

Dijkstra's algorithm: Java implementation

```
public class DijkstaSP
{
   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

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
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.

Priority-first search

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

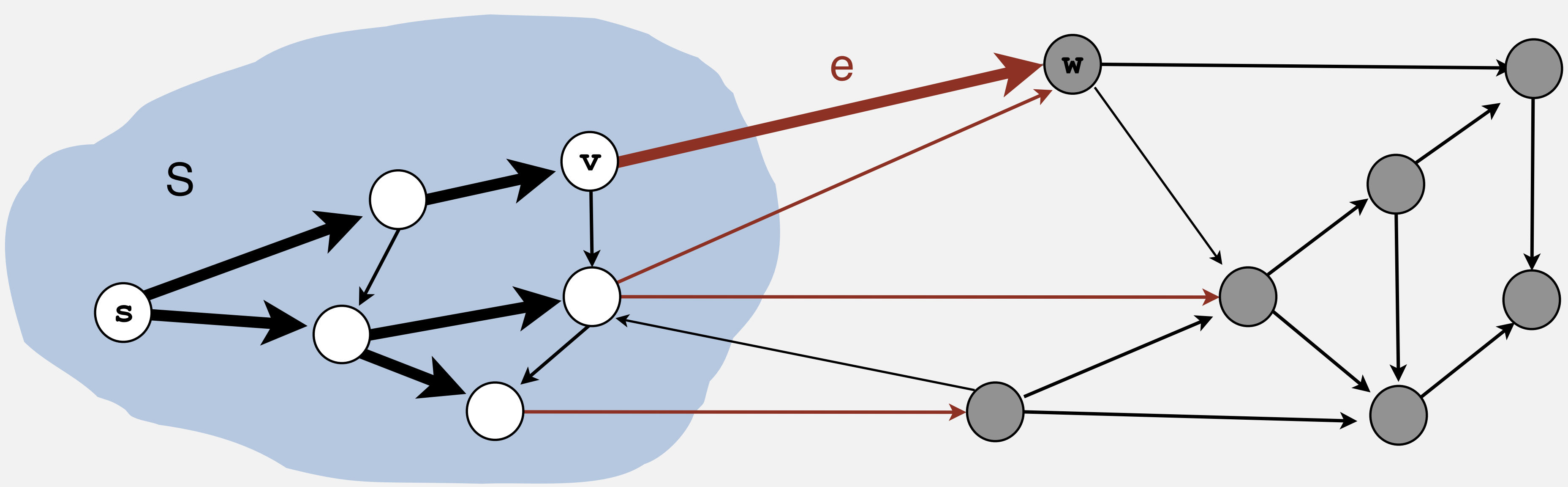
- Maintain a set of explored vertices S.
- Grow S by exploring edges with exactly one endpoint leaving S.

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.



Challenge. Express this insight in reusable Java code.