SHORTEST PATHS
Shortest paths in an edge-weighted digraph

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

edge-weighted digraph

What is the shortest Path from 0 to 6
Shortest path applications

- Map routing.
- **Seam carving**.
- Robot navigation.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting **arbitrage** opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.

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?

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

Cycles?

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

Simplifying assumption. Shortest paths from \( s \) to each vertex \( v \) exist.
4.4 Shortest Paths

- APIs
- shortest-paths properties
- Dijkstra's algorithm
- edge-weighted DAGs
- negative weights
public class DirectedEdge

DirectedEdge(int v, int w, double weight)  
  
  int from()  
  
  int to()  
  
  double weight()  
  
  String toString()  

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.

```java
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;
    }
}
```
## Edge-weighted digraph API

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public class EdgeWeightedDigraph</code></td>
<td></td>
</tr>
<tr>
<td><code>EdgeWeightedDigraph(int V)</code></td>
<td>edge-weighted digraph with $V$ vertices</td>
</tr>
<tr>
<td><code>EdgeWeightedDigraph(In in)</code></td>
<td>edge-weighted digraph from input stream</td>
</tr>
<tr>
<td><code>void addEdge(DirectedEdge e)</code></td>
<td>add weighted directed edge $e$</td>
</tr>
<tr>
<td><code>Iterable&lt;DirectedEdge&gt; adj(int v)</code></td>
<td>edges pointing from $v$</td>
</tr>
<tr>
<td><code>int V()</code></td>
<td>number of vertices</td>
</tr>
<tr>
<td><code>int E()</code></td>
<td>number of edges</td>
</tr>
<tr>
<td><code>Iterable&lt;DirectedEdge&gt; edges()</code></td>
<td>all edges</td>
</tr>
<tr>
<td><code>String toString()</code></td>
<td>string representation</td>
</tr>
</tbody>
</table>

**Conventions.** Allow self-loops and parallel edges.
Edge-weighted digraph: adjacency-lists representation

tinyEWD.txt

V

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

E

adj

0

1

2

3

4

5

6

7

Bag objects

reference to a DirectedEdge object
Same as EdgeWeightedGraph except replace Graph with Digraph.

```java
public class EdgeWeightedDigraph {
  private final int V;
  private final Bag<DirectedEdge>[] 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];
  }
}
```
## Single-source shortest paths API

**Goal.** Find the shortest path from \( s \) to every other vertex.

```java
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();
    }
}
```
4.4 Shortest Paths

- APIs
- 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-paths tree (SPT) solution exists. Why?

**Consequence.** Can represent the SPT with two vertex-indexed arrays:
- $\text{distTo}[v]$ is length of shortest path from $s$ to $v$.
- $\text{edgeTo}[v]$ is last edge on shortest path from $s$ to $v$.

<table>
<thead>
<tr>
<th></th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>null</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>5-&gt;1</td>
<td>0.32</td>
</tr>
<tr>
<td>2</td>
<td>0-&gt;2</td>
<td>0.26</td>
</tr>
<tr>
<td>3</td>
<td>7-&gt;3</td>
<td>0.37</td>
</tr>
<tr>
<td>4</td>
<td>0-&gt;4</td>
<td>0.38</td>
</tr>
<tr>
<td>5</td>
<td>4-&gt;5</td>
<td>0.35</td>
</tr>
<tr>
<td>6</td>
<td>3-&gt;6</td>
<td>0.52</td>
</tr>
<tr>
<td>7</td>
<td>2-&gt;7</td>
<td>0.34</td>
</tr>
</tbody>
</table>

shortest-paths tree from 0

parent-link representation
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:
- $\text{distTo}[v]$ is length of shortest path from $s$ to $v$.
- $\text{edgeTo}[v]$ is last edge on shortest path from $s$ to $v$.

```java
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 \( e = v \rightarrow w \).

- \( \text{distTo}[v] \) is length of shortest known path from \( s \) to \( v \).
- \( \text{distTo}[w] \) is length of shortest known path from \( s \) to \( w \).
- \( \text{edgeTo}[w] \) is last edge on shortest known path from \( s \) to \( w \).
- If \( e = v \rightarrow w \) gives shorter path to \( w \) through \( v \), update both \( \text{distTo}[w] \) and \( \text{edgeTo}[w] \).
Edge relaxation

Relax edge $e = v \rightarrow w$.

- $\text{distTo}[v]$ is length of shortest known path from $s$ to $v$.
- $\text{distTo}[w]$ is length of shortest known path from $s$ to $w$.
- $\text{edgeTo}[w]$ is last edge on shortest known path from $s$ to $w$.
- If $e = v \rightarrow w$ gives shorter path to $w$ through $v$, update both $\text{distTo}[w]$ and $\text{edgeTo}[w]$.

```java
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;
    }
}
```
Edge relaxation

Relax edge $e = v \rightarrow w$.

- $\text{distTo}[v]$ is length of shortest known path from $s$ to $v$.
- $\text{distTo}[w]$ is length of shortest known path from $s$ to $w$.
- $\text{edgeTo}[w]$ is last edge on shortest known path from $s$ to $w$.
- If $e = v \rightarrow w$ gives shorter path to $w$ through $v$, update both $\text{distTo}[w]$ and $\text{edgeTo}[w]$.

```java
private void relax(EdgeWeightedDigraph G, int v)
{
    for(DirectedEdge e: G.adj(V))
    {
        int 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 $\text{distTo}[]$ are the shortest path distances from $s$ iff:

- $\text{distTo}[s] = 0$.
- For each vertex $v$, $\text{distTo}[v]$ is the length of some path from $s$ to $v$.
- For each edge $e = v \rightarrow w$, $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight}()$.

Pf by contradiction. $\iff$ [necessary]

- Suppose that $\text{distTo}[w] > \text{distTo}[v] + e.\text{weight}()$ for some edge $e = v \rightarrow w$.
- Then, $e$ gives a path from $s$ to $w$ (through $v$) of length less than $\text{distTo}[w]$. 
Shortest-paths optimality conditions

**Proposition.** Let $G$ be an edge-weighted digraph.

Then $\text{distTo}[\cdot]$ are the shortest path distances from $s$ iff:

- $\text{distTo}[s] = 0$.
- For each vertex $v$, $\text{distTo}[v]$ is the length of some path from $s$ to $v$.
- For each edge $e = v \to w$, $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight()}$.

**Pf.** $\implies$ [sufficient]

- Suppose that $s = v_0 \to v_1 \to v_2 \to \ldots \to v_k = w$ is a shortest path from $s$ to $w$.
- Then,
  
  \[
  \begin{align*}
  \text{distTo}[v_1] & \leq \text{distTo}[v_0] + e_1.\text{weight()} \\
  \text{distTo}[v_2] & \leq \text{distTo}[v_1] + e_2.\text{weight()} \\
  & \vdots \\
  \text{distTo}[v_k] & \leq \text{distTo}[v_{k-1}] + e_k.\text{weight()} \\
  \end{align*}
  \]

- Add inequalities; simplify; and substitute $\text{distTo}[v_0] = \text{distTo}[s] = 0$:
  \[
  \text{distTo}[w] = \text{distTo}[v_k] \leq e_1.\text{weight()} + e_2.\text{weight()} + \ldots + e_k.\text{weight()} 
  \]

- Thus, $\text{distTo}[w]$ is the weight of shortest path to $w$. $\blacksquare$
**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.**

- The entry `distTo[v]` is always the length of a simple path from s to v.
- 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).
4.4 Shortest Paths

- APIs
- shortest-paths properties
- Dijkstra's algorithm
- edge-weighted DAGs
- negative weights
“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.”
"Object-oriented programming is an exceptionally bad idea which could only have originated in California."

-- Edsger Dijkstra
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

an edge-weighted digraph

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0→1</td>
<td>5.0</td>
</tr>
<tr>
<td>0→4</td>
<td>9.0</td>
</tr>
<tr>
<td>0→7</td>
<td>8.0</td>
</tr>
<tr>
<td>1→2</td>
<td>12.0</td>
</tr>
<tr>
<td>1→3</td>
<td>15.0</td>
</tr>
<tr>
<td>1→7</td>
<td>4.0</td>
</tr>
<tr>
<td>2→3</td>
<td>3.0</td>
</tr>
<tr>
<td>2→6</td>
<td>11.0</td>
</tr>
<tr>
<td>3→6</td>
<td>9.0</td>
</tr>
<tr>
<td>4→5</td>
<td>4.0</td>
</tr>
<tr>
<td>4→6</td>
<td>20.0</td>
</tr>
<tr>
<td>4→7</td>
<td>5.0</td>
</tr>
<tr>
<td>5→2</td>
<td>1.0</td>
</tr>
<tr>
<td>5→6</td>
<td>13.0</td>
</tr>
<tr>
<td>7→5</td>
<td>6.0</td>
</tr>
<tr>
<td>7→2</td>
<td>7.0</td>
</tr>
</tbody>
</table>
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.

![Diagram of a weighted digraph](image)

**an edge-weighted digraph**

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0→1</td>
<td>5.0</td>
</tr>
<tr>
<td>0→4</td>
<td>9.0</td>
</tr>
<tr>
<td>0→7</td>
<td>8.0</td>
</tr>
<tr>
<td>1→2</td>
<td>12.0</td>
</tr>
<tr>
<td>1→3</td>
<td>15.0</td>
</tr>
<tr>
<td>1→7</td>
<td>4.0</td>
</tr>
<tr>
<td>2→3</td>
<td>3.0</td>
</tr>
<tr>
<td>2→6</td>
<td>11.0</td>
</tr>
<tr>
<td>3→6</td>
<td>9.0</td>
</tr>
<tr>
<td>4→5</td>
<td>4.0</td>
</tr>
<tr>
<td>4→6</td>
<td>20.0</td>
</tr>
<tr>
<td>4→7</td>
<td>5.0</td>
</tr>
<tr>
<td>5→2</td>
<td>1.0</td>
</tr>
<tr>
<td>5→6</td>
<td>13.0</td>
</tr>
<tr>
<td>7→5</td>
<td>6.0</td>
</tr>
<tr>
<td>7→2</td>
<td>7.0</td>
</tr>
</tbody>
</table>
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.

choose source vertex 0
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

relax all edges pointing from 0

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra's algorithm demo

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

![Graph with vertices and distances]

<table>
<thead>
<tr>
<th>( v )</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges pointing from 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.
Dijkstra's algorithm demo

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

```
\[
\begin{array}{lll}
\text{v} & \text{distTo[]} & \text{edgeTo[]} \\
0 & 0.0 & - \\
1 & 5.0 & 0 \rightarrow 1 \\
2 & & \\
3 & & \\
4 & 9.0 & 0 \rightarrow 4 \\
5 & & \\
6 & & \\
7 & 8.0 & 0 \rightarrow 7 \\
\end{array}
\]
```

choose vertex 1
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>8.0</td>
<td>0→7</td>
</tr>
<tr>
<td>6</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>
```

relax all edges pointing from 1
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

#### Table

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>17.0</td>
<td>1→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

Relax all edges pointing from 1
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

```
choose vertex 7
```

```
<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>17.0</td>
<td>1→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest \texttt{distTo[]} value).
- Add vertex to tree and relax all edges pointing from that vertex.

```
\begin{verbatim}
\begin{array}{ccc}
\text{v} & \text{distTo[]} & \text{edgeTo[]} \\
0 & 0.0 & - \\
1 & 5.0 & 0 \rightarrow 1 \\
2 & 17.0 & 1 \rightarrow 2 \\
3 & 20.0 & 1 \rightarrow 3 \\
4 & 9.0 & 0 \rightarrow 4 \\
5 & & \\
6 & & \\
7 & 8.0 & 0 \rightarrow 7 \\
\end{array}
\end{verbatim}
```

relax all edges pointing from 7
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest \texttt{distTo[]} value).
- Add vertex to tree and relax all edges pointing from that vertex.

relax all edges pointing from 7
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

![Dijkstra's algorithm diagram]

<table>
<thead>
<tr>
<th>$v$</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0$\rightarrow$1</td>
</tr>
<tr>
<td>2</td>
<td>15.0</td>
<td>7$\rightarrow$2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1$\rightarrow$3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0$\rightarrow$4</td>
</tr>
<tr>
<td>5</td>
<td>14.0</td>
<td>7$\rightarrow$5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0$\rightarrow$7</td>
</tr>
</tbody>
</table>

select vertex 4
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo[]}$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

![Graph with labels and edge distances]

<table>
<thead>
<tr>
<th>$v$</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>15.0</td>
<td>7→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>14.0</td>
<td>7→5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges pointing from 4
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.

```
relax all edges pointing from 4
```
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>15.0</td>
<td>7→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>29.0</td>
<td>4→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

select vertex 5
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.

relax all edges pointing from 5
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

![Graph diagram]

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>26.0</td>
<td>5→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges pointing from 5
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>20.0</td>
<td>1→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>26.0</td>
<td>5→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

\[
\begin{array}{c|c|c}
v & \text{distTo[]} & \text{edgeTo[]} \\
0 & 0.0 & - \\
1 & 5.0 & 0\rightarrow1 \\
2 & 14.0 & 5\rightarrow2 \\
3 & 20.0 & 1\rightarrow3 \\
4 & 9.0 & 0\rightarrow4 \\
5 & 13.0 & 4\rightarrow5 \\
6 & 26.0 & 5\rightarrow6 \\
7 & 8.0 & 0\rightarrow7 \\
\end{array}
\]

select vertex 2
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

```
\begin{array}{c|c|c}
\text{v} & \text{distTo[]} & \text{edgeTo[]} \\
\hline
0 & 0.0 & - \\
1 & 5.0 & 0\rightarrow1 \\
2 & 14.0 & 5\rightarrow2 \\
3 & 20.0 & 1\rightarrow3 \\
4 & 9.0 & 0\rightarrow4 \\
5 & 13.0 & 4\rightarrow5 \\
6 & 26.0 & 5\rightarrow6 \\
7 & 8.0 & 0\rightarrow7 \\
\end{array}
```

relax all edges pointing from 2
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.

![Graph showing Dijkstra's algorithm demonstration]

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges pointing from 2
Dijkstra’s algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
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.

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```

select vertex 3
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest \texttt{distTo[]} value).
- Add vertex to tree and relax all edges pointing from that vertex.

relax all edges pointing from 3
Dijkstra's algorithm demo

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

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>✔️ 2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```

relax all edges pointing from 3
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

```plaintext
<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```
Dijkstra's algorithm demo

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

```
select vertex 6
```

```
<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
```
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

relax all edges pointing from 6
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.
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.
Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{distTo}[]$</th>
<th>$\text{edgeTo}[]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>

shortest-paths tree from vertex $s$
Dijkstra's algorithm visualization
Dijkstra's algorithm visualization
**Dijkstra's algorithm: correctness proof 1**

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

**Pf.**

- Each edge $e = v \rightarrow w$ is relaxed exactly once (when vertex $v$ is relaxed), leaving $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight()}$.
- Inequality holds until algorithm terminates because:
  - $\text{distTo}[w]$ cannot increase $\Leftarrow$ $\text{distTo[]}$ values are monotone decreasing
  - $\text{distTo}[v]$ will not change $\Leftarrow$ we choose lowest $\text{distTo[]}$ value at each step (and edge weights are nonnegative)

- Thus, upon termination, shortest-paths optimality conditions hold.  ■
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 8 to each edge weight changes the shortest path from 0→1→2→3 to 0→3.

**Conclusion.** Need a different algorithm.
Dijkstra's algorithm: Java implementation

```java
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

```java
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
Computing a spanning tree in a graph

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

Main distinction: Rule used to choose next vertex for the tree.
- Prim: Closest vertex to the tree (via an undirected edge).
- Dijkstra: Closest vertex to the source (via a directed path).
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.
Dijkstra's algorithm: which priority queue?

Depends on PQ implementation: $V$ insert, $V$ delete-min, $E$ decrease-key.

<table>
<thead>
<tr>
<th>PQ implementation</th>
<th>insert</th>
<th>delete-min</th>
<th>decrease-key</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>1</td>
<td>$V$</td>
<td>1</td>
<td>$V^2$</td>
</tr>
<tr>
<td>binary heap</td>
<td>$\log V$</td>
<td>$\log V$</td>
<td>$\log V$</td>
<td>$E \log V$</td>
</tr>
<tr>
<td>d-way heap</td>
<td>$\log_d V$</td>
<td>$d \log_d V$</td>
<td>$\log_d V$</td>
<td>$E \log_{E/V} V$</td>
</tr>
</tbody>
</table>

$E \log D V$ where the best value for $D$ is $E/D$

**Bottom line.**

- Array implementation optimal for dense 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.