Google maps
Given an edge-weighted digraph, find the shortest (directed) path from $s$ to $t$. 

**edge-weighted digraph**

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

**shortest path from 0 to 6**

- $0 \rightarrow 2 \ 0.26$
- $2 \rightarrow 7 \ 0.34$
- $7 \rightarrow 3 \ 0.39$
- $3 \rightarrow 6 \ 0.52$
Optimization problems expressible as shortest paths

Many optimization problems can be expressed as a shortest path computation in a suitable weighted graph. (Course goal: Be able to do this!)

Examples (board):
1. Fastest route (no traffic lights), where a turn takes 5 seconds.
2. Cheapest route, considering gasoline use and road tolls.

Problem session:
Model this problem as one shortest path computation:
3. Find the shortest path from vertex $s$ to a vertex $v$ in the set $S$ (e.g., $S$ could be vertices with gasoline stations).
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.

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.

Cycles?
- No cycles.
- No "negative cycles."

Simplifying assumption. There exists a shortest path from $s$ to each vertex $v$. 
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 &lt;DirectedEdge&gt; pathTo(int v)  
shortest path from $s$ to $v$

boolean hasPathTo(int v)  
is there a path from $s$ to $v$?
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:
- $\text{distTo}[v]$ 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>vertex</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 path tree from 0
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 $\text{distTo}[w]$ 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;
    }
}
```

$v \rightarrow w$ successfully relaxes
Shortest-paths optimality conditions

Proposition. Let $G$ be an edge-weighted digraph. Then $\text{distTo}[v]$ are the shortest path distances from $s$ iff:

- 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. $\iff$ [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 $\text{distTo}[w]$. 

\[ \text{distTo}[v] \]

\[ \text{distTo}[w] \]

weight of $v \rightarrow w$ is 1.3

Monday, November 7, 11
Shortest-paths optimality conditions

Proposition. Let $G$ be an edge-weighted digraph.
Then $\text{distTo}[]$ are the shortest path distances from $s$ iff:
• 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. $\Rightarrow$ [sufficient]

• Suppose that $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k = w$ is a shortest path from $s$ to $w$.

• Then, $\text{distTo}[v_k] \leq \text{distTo}[v_{k-1}] + e_k.\text{weight()}$
  $\text{distTo}[v_{k-1}] \leq \text{distTo}[v_{k-2}] + e_{k-1}.\text{weight()}$
  $\vdots$
  $\text{distTo}[v_1] \leq \text{distTo}[v_0] + e_1.\text{weight()}$

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

• Add inequalities; simplify; and substitute $\text{distTo}[v_0] = \text{distTo}[s] = 0$:
  $\text{distTo}[w] = \text{distTo}[v_k] \leq e_k.\text{weight()} + e_{k-1}.\text{weight()} + ... + e_1.\text{weight()}$

  weight of some path from $s$ to $w$  weight of shortest path from $s$ to $w$

• Thus, $\text{distTo}[w]$ is the weight of shortest path to $w$. ■
Generic shortest-paths algorithm

**Proposition.** Generic algorithm computes SPT from $s$.  

**Pf sketch.**

- Throughout algorithm, $\text{distTo}[v]$ is the length of a simple path from $s$ to $v$ and $\text{edgeTo}[v]$ is last edge on path.
- Each successful relaxation decreases $\text{distTo}[v]$ for some $v$.
- The entry $\text{distTo}[v]$ can decrease at most a finite number of times. ■
Generic shortest-paths algorithm

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).
"Object-oriented programming is an exceptionally bad idea which could only have originated in California."
-- Edsger Dijkstra
Dijkstra's algorithm

- 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 incident from that vertex.

<table>
<thead>
<tr>
<th></th>
<th>\text{distTo}[v]</th>
<th>\text{edgeTo}[v]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</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>

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  

An edge-weighted digraph and a shortest path from 0 to 6

5
\rightarrow 4 0.38
4 \rightarrow 2 0.26
2 \rightarrow 7 0.34
7 \rightarrow 3 0.39
3 \rightarrow 6 0.52
6 \rightarrow 0 0.58
6 \rightarrow 4 0.93

0 → 4 0.38
4 → 2 0.26
2 → 7 0.34
7 → 3 0.39
3 → 6 0.52
6 → 0 0.58
6 → 4 0.93

\begin{align*}
4 & \rightarrow 5 & 0.35 \\
5 & \rightarrow 4 & 0.35 \\
4 & \rightarrow 7 & 0.37 \\
5 & \rightarrow 7 & 0.28 \\
7 & \rightarrow 5 & 0.28 \\
5 & \rightarrow 1 & 0.32 \\
0 & \rightarrow 4 & 0.38 \\
0 & \rightarrow 2 & 0.26 \\
7 & \rightarrow 3 & 0.39 \\
1 & \rightarrow 3 & 0.29 \\
2 & \rightarrow 7 & 0.34 \\
6 & \rightarrow 2 & 0.40 \\
3 & \rightarrow 6 & 0.52 \\
6 & \rightarrow 0 & 0.58 \\
6 & \rightarrow 4 & 0.93 \\
\end{align*}
Dijkstra's algorithm visualization
Shortest path trees

- Consider vertices in increasing order of distance from $s$ (non-tree vertex with the lowest $\text{distTo}[\cdot]$ value).
- Add vertex to tree and relax all edges incident from that vertex.
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 $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight}()$.
- Inequality holds until algorithm terminates because:
  - $\text{distTo}[w]$ cannot increase  \hspace{1cm} $\text{distTo}[]$ values are monotone decreasing
  - $\text{distTo}[v]$ will not change  \hspace{1cm} edge weights are nonnegative and we choose lowest $\text{distTo}[]$ value at each step

- Thus, upon termination, shortest-paths optimality conditions hold. □
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);
      }
   }

   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
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>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>Fibonacci heap</td>
<td>1 $\dagger$</td>
<td>log $V$ $\dagger$</td>
<td>1 $\dagger$</td>
<td>$E + V \log V$</td>
</tr>
</tbody>
</table>

$\dagger$ amortized

Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- Fibonacci heap best in theory, but not worth implementing.
Q. Suppose that an edge-weighted digraph has no directed cycles. Is it easier to find shortest paths than in a general digraph?

A. Yes!

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

Acyclic edge-weighted digraphs
Topological sort algorithm.

- Consider vertices in topologically order.
- Relax all edges incident from vertex.

Topological order: 5 1 3 6 4 7 0 2
Shortest paths in edge-weighted DAGs

Topological sort algorithm.
• Consider vertices in topologically order.
• Relax all edges incident from vertex.

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

Pf.
• Each edge \( e = v \rightarrow w \) is relaxed exactly once (when \( v \) is relaxed), leaving \( \text{distTo}[w] \leq \text{distTo}[v] + e\text{.weight()} \).
• Inequality holds until algorithm terminates because:
  - \( \text{distTo}[w] \) cannot increase
  - \( \text{distTo}[v] \) will not change
    ← distTo[] values are monotone decreasing
    ← because of topological order, no edge pointing to \( v \) will be relaxed after \( v \) is relaxed

• Thus, upon termination, shortest-paths optimality conditions hold. ■
Formulate as a shortest paths problem in edge-weighted DAGs.

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

**Key point.** Topological sort algorithm works even with negative edge weights.
Longest paths in edge-weighted DAGs: application

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

<table>
<thead>
<tr>
<th>job</th>
<th>duration</th>
<th>must complete before</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41.0</td>
<td>1 7 9</td>
</tr>
<tr>
<td>1</td>
<td>51.0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>50.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>36.0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>38.0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>21.0</td>
<td>3 8</td>
</tr>
<tr>
<td>7</td>
<td>32.0</td>
<td>3 8</td>
</tr>
<tr>
<td>8</td>
<td>32.0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>29.0</td>
<td>4 6</td>
</tr>
</tbody>
</table>

Parallel job scheduling solution
Critical path method

**CPM.** To solve a parallel job-scheduling problem, create acyclic edge-weighted digraph:

- Source and sink vertices.
- Two vertices (begin and end) for each job.
- Three edges for each job.
  - begin to end (weighted by duration)
  - source to begin (0 weight)
  - end to sink (0 weight)
**Critical path method**

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

![Diagram of a project network with critical path marked]

Parallel job scheduling solution

---

Monday, November 7, 11
Shortest paths with negative weights: failed attempts

Dijkstra. Doesn’t work with negative edge weights.

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

Bad news. Need a different algorithm.
Def. A **negative cycle** is a directed cycle whose sum of edge weights is negative.

**Proposition.** A SPT exists iff no negative cycles.

assuming all vertices reachable from $s$
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 phase $i$, found shortest path containing at most $i$ edges.

Dynamic programming algorithm

- Initialize $\text{distTo}[s] = 0$ and $\text{distTo}[v] = \infty$ for all other vertices.
- Repeat V times:
  - Relax each edge.

```java
for (int i = 1; i <= G.V(); i++)
   for (int v = 0; v < G.V(); v++)
      for (DirectedEdge e : G.adj(v))
         relax(e);
```
Bellman-Ford algorithm

**Observation.** If \( \text{distTo}[v] \) does not change during phase \( i \), no need to relax any edge incident from \( v \) in phase \( i+1 \).

**FIFO implementation.** Maintain queue of vertices whose \( \text{distTo}[] \) changed.

\[ \text{be careful to keep at most one copy} \]
\[ \text{of each vertex on queue (why?)} \]

**Overall effect.**

- The running time is still proportional to \( E \times V \) in worst case.
- But much faster than that in practice.
Bellman-Ford algorithm visualization

phases
4

edges on queue in red

13

SPT
Single source shortest-paths implementation: cost summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Restriction</th>
<th>Typical Case</th>
<th>Worst Case</th>
<th>Extra Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topological sort</td>
<td>No directed cycles</td>
<td>$E + V$</td>
<td>$E + V$</td>
<td>$V$</td>
</tr>
<tr>
<td>Dijkstra (binary heap)</td>
<td>No negative weights</td>
<td>$E \log V$</td>
<td>$E \log V$</td>
<td>$V$</td>
</tr>
<tr>
<td>Dynamic programming</td>
<td>No negative cycles</td>
<td>$E \times V$</td>
<td>$E \times V$</td>
<td>$V$</td>
</tr>
<tr>
<td>Bellman-Ford</td>
<td>No negative cycles</td>
<td>$E + V$</td>
<td>$E + V$</td>
<td>$V$</td>
</tr>
</tbody>
</table>

**Remark 1.** Directed cycles make the problem harder.

**Remark 2.** Negative weights make the problem harder.

**Remark 3.** Negative cycles makes the problem intractable.
Finding a negative cycle

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

![Diagram](image)

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

In practice. Check for negative cycles more frequently.
**Problem.** Given table of exchange rates, is there an arbitrage opportunity?

### Negative cycle application: arbitrage detection

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

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

<table>
<thead>
<tr>
<th></th>
<th>USD</th>
<th>EUR</th>
<th>GBP</th>
<th>CHF</th>
<th>CAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>1</td>
<td>0.741</td>
<td>0.657</td>
<td>1.061</td>
<td>1.011</td>
</tr>
<tr>
<td>EUR</td>
<td>1.350</td>
<td>1</td>
<td>0.888</td>
<td>1.433</td>
<td>1.366</td>
</tr>
<tr>
<td>GBP</td>
<td>1.521</td>
<td>1.126</td>
<td>1</td>
<td>1.614</td>
<td>1.538</td>
</tr>
<tr>
<td>CHF</td>
<td>0.943</td>
<td>0.698</td>
<td>0.620</td>
<td>1</td>
<td>0.953</td>
</tr>
<tr>
<td>CAD</td>
<td>0.995</td>
<td>0.732</td>
<td>0.650</td>
<td>1.049</td>
<td>1</td>
</tr>
</tbody>
</table>
Negative cycle application: arbitrage detection

Currency exchange graph.

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

**Challenge.** Express as a negative cycle detection problem.
Model as a negative cycle detection problem by taking logs.

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

Remark. Fastest algorithm is extraordinarily valuable!
**Conclusion**

Weighted graphs can be used to model a large variety of computational problems.

Shortest paths algorithms can sometimes be used as building blocks to solve optimization problems.

Efficiency of shortest paths depends on characteristics of the graph: Negative weights? Cycles? Negative cycles?

Related course goals:

- **Choose** among and make use of the most important algorithms and data structures in libraries, based on knowledge of their complexity.
- **Design** algorithms for ad hoc problems by using and combining known algorithms and data structures.