COMS21103

Finding the shortest path

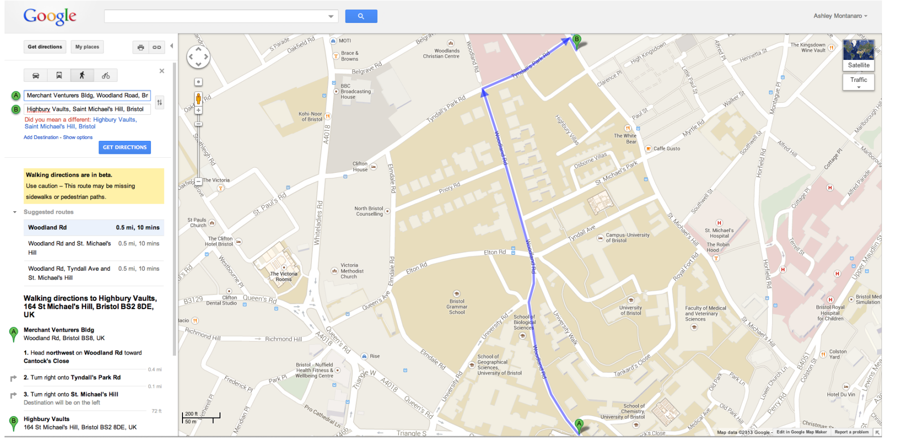
Ashley Montanaro
ashley@cs.bris.ac.uk

Department of Computer Science, University of Bristol
Bristol, UK

28 October 2013

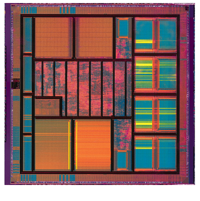
Given a (weighted, directed) graph G and a pair of vertices s and t, we would like to find a shortest path from s to t.

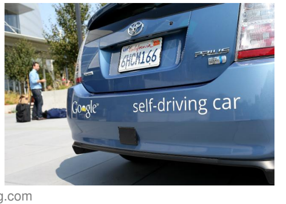
A fundamental task with many applications:



Other applications

- Internet routing (e.g. the OSPF routing algorithm)
- VLSI routing
- Traffic information systems
- Robot motion planning
- Routing telephone calls
- Avoiding nuclear contamination
- Destabilising currency markets
- . . .

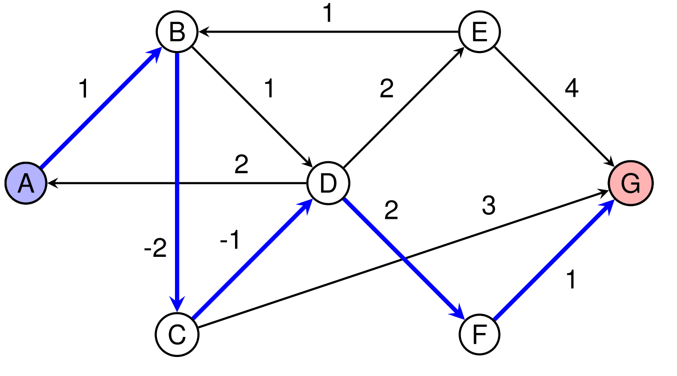




Pics: Wikipedia, autoevolution.com, autoblog.com

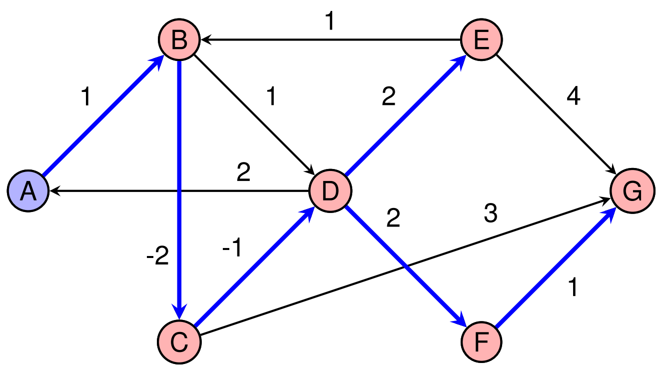
Shortest paths problem

Formally, a shortest path from s to t in a graph G is a sequence $v_1, v_2, \dots, v_m$ such that the total weight of the edges $s \to v_1$, $v_1 \to v_2, \dots,$ $v_m \to t$ is minimal.



Single-source shortest paths

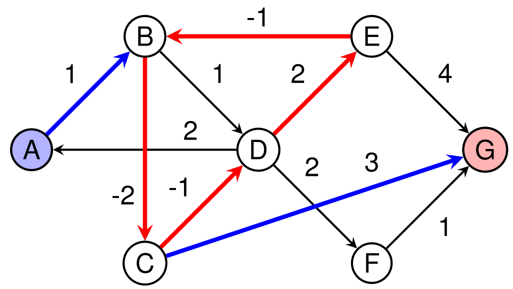
- In fact, the algorithms we will discuss for this problem give us more: given a source *s*, they output a shortest path from *s* to every other vertex.
- This is known as the single-source shortest path problem (SSSP).



Ashley Montanaro
ashley@cs.bris.ac.uk

Negative-weight edges

- If some of the edges have negative weights, the idea of a shortest path might not make sense.
- If there is a cycle in *G* which is reachable on a path from *s* to *t*, and the sum of the weights of the edges in the cycle is negative, then we can get from *s* to *t* with a path of arbitrarily low weight by repeatedly going round the cycle.



Ashley Montanaro
ashley@cs.bris.ac.uk

Today's lecture

- Today we will discuss an algorithm for the single-source shortest paths problem called the Bellman-Ford algorithm.
- The algorithm can be used for graphs with negative weights and can detect negative-weight cycles.
- It also has applications to solving systems of difference constraints and detecting arbitrage.

Remark: One algorithmic idea to solve the SSSP that doesn't work is to try every possible path from *s* to *t* in turn.

- There can be exponentially many paths so such an algorithm cannot be efficient.

Ashley Montanaro
ashley@cs.bris.ac.uk

Notation

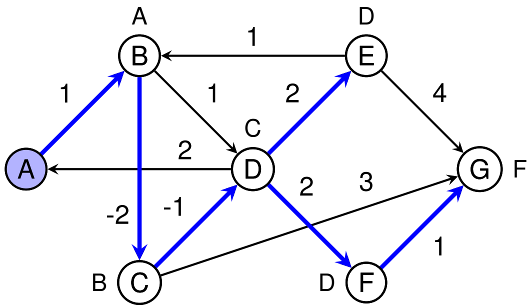
We will use the following notation (essentially the same as CLRS):

- We always let G denote the graph in which we want to find a shortest path. We use V for the number of vertices in G, and E for the number of edges. s always denotes the source.
- We write $u \to v$ for an edge from u to v, and $w(u, v)$ for the weight of this edge.
- We write $\delta(u, v)$ for the distance from u to v, i.e. the length (total weight) of a shortest path from u to v.
- We write $\delta(u, v) = \infty$ when there is no path from u to v. (Mathematical note: in practice, ∞ would be represented by a number so large it could never occur in distance calculations...)
- For each vertex v, we will maintain a guess for its distance from s; call this $v.d$.

Ashley Montanaro
ashley@cs.bris.ac.uk

Predecessors and shortest paths

- For each vertex v, we try to determine its predecessor $v.\pi$, which is the previous vertex in some shortest path from s to v.
- Knowledge of v's predecessor suffices to compute the whole path from s to v, by following the predecessors back to s and reversing the path.



A general framework

The basic idea behind both shortest-path algorithms we will discuss is:

1. Initialise a guess $v.d$ for the distance from the source s: $s.d = 0$, and $v.d = \infty$ for all other vertices v.
2. Update our guesses by relaxing edges:

- If there is an edge $u \to v$ and our guess for the distance from s to v is greater than our guess for the distance from s to u, plus $w(u, v)$, then we can improve our guess by using this edge.

Relax(u, v)

1. if $v.d > u.d + w(u, v)$
2. $\quad v.d \leftarrow u.d + w(u, v)$
3. $\quad v.\pi = u$

Note that $\infty + x = \infty$ for any real number x.

The Bellman-Ford algorithm

This algorithm simply consists of repeatedly relaxing every edge in G.

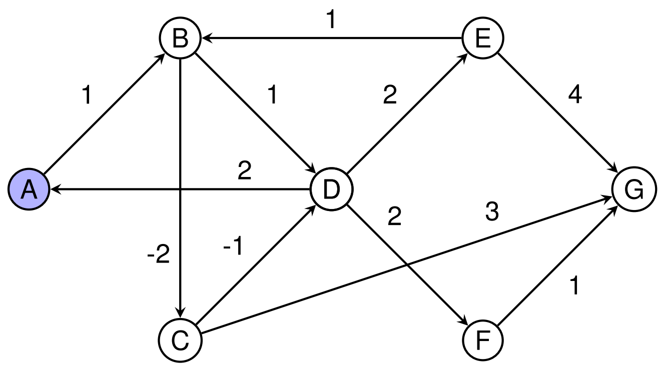
BellmanFord(G, s)

1. for each vertex $v \in G$: $v.d \leftarrow \infty$, $v.\pi \leftarrow$ nil
2. $s.d \leftarrow 0$
3. for $i = 1$ to $V - 1$
4. $\quad$ for each edge $u \to v$ in G
5. $\quad\quad$ Relax(u, v)
6. for each edge $u \to v$ in G
7. $\quad$ if $v.d > u.d + w(u, v)$
8. $\quad\quad$ error("Negative-weight cycle detected")

- Time complexity: $\Theta(V) + \Theta(VE) + \Theta(E) = \Theta(VE)$.

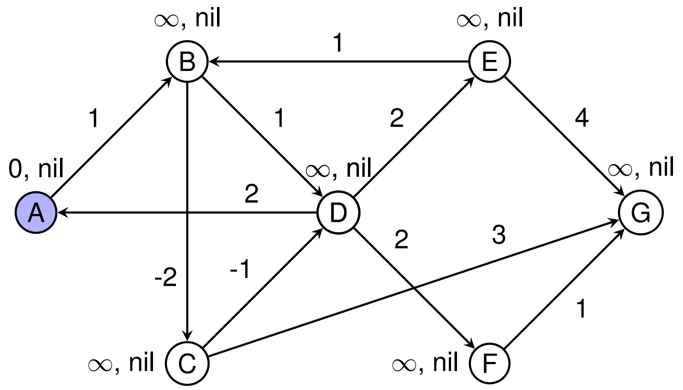
Example 1: no negative-weight cycles

Imagine we want to find shortest paths from vertex A in the following graph:



Example 1: no negative-weight cycles

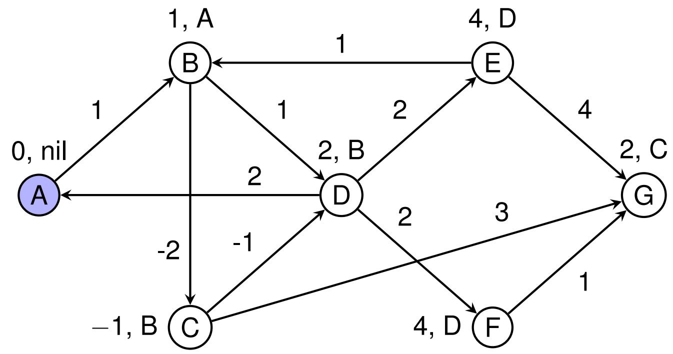
At the start of the algorithm:



- In the above diagram, the red text is the distance from the source A, (i.e. $v.d$), and the green text is the predecessor vertex (i.e. $v.\pi$).

Example 1: no negative-weight cycles

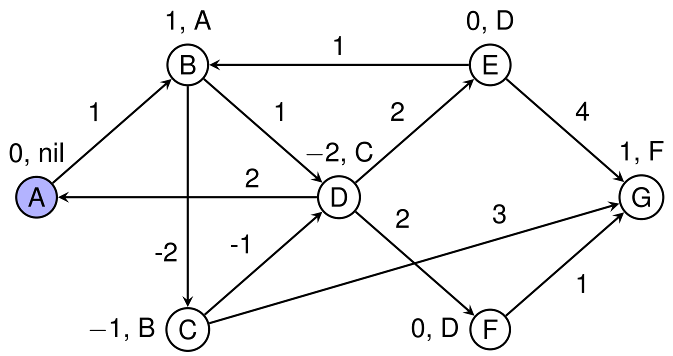
The first iteration of the for loop:



- Note that the edges are picked in arbitrary order.

Example 1: no negative-weight cycles

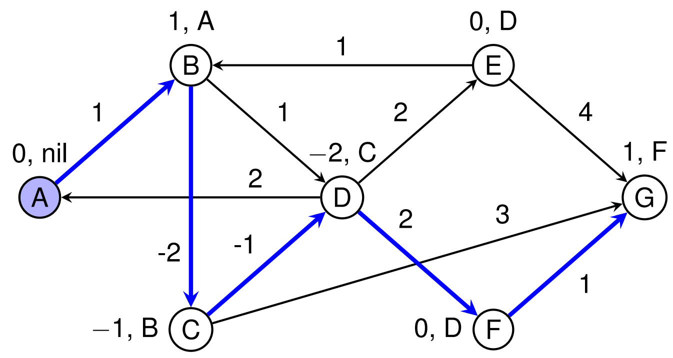
The second iteration of the for loop:



- Note that the edges are picked in arbitrary order.

Example 1: no negative-weight cycles

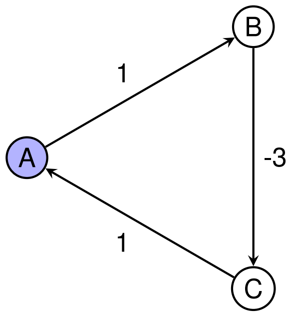
The 4 iterations of the for loop that follow do not update any distance or predecessor values, so the final state is:



- So the shortest path from A to G (for example) has weight 1.
- To output a shortest path itself, we can trace back the predecessor values from G.

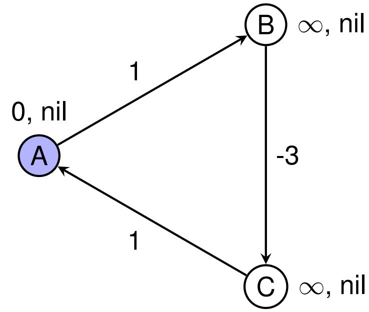
Example 2: negative-weight cycle

We now consider an input graph that has a negative-weight cycle.



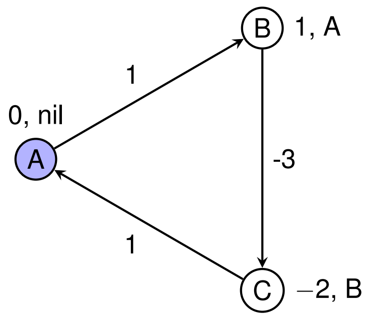
Example 2: negative-weight cycle

At the start of the algorithm:



Example 2: negative-weight cycle

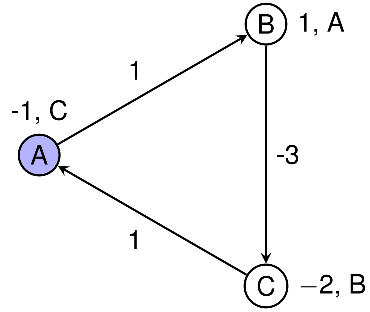
The first iteration of the for loop:



- As before, the order in which we consider the edges is arbitrary (here we use the order A $\to$ B, C $\to$ A, B $\to$ C).

Example 2: negative-weight cycle

The second iteration of the for loop:



- At the end of the algorithm, $B.d > A.d + w(A, B)$.
- So the algorithm terminates with "Negative-weight cycle detected".

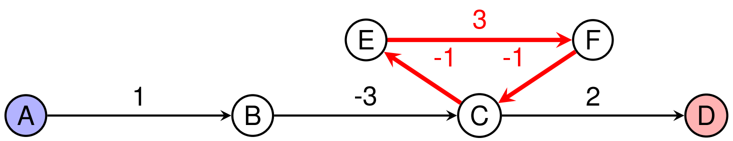
Proof of correctness: Preliminaries

Claim (cycles)

If G does not contain any negative-weight cycles reachable from s, a shortest path from s to t cannot contain a cycle.

Proof

If a path p contains a cycle $v_0 \to v_1 \to \cdots \to v_0$ such that the sum of the weights of the edges is non-negative, deleting this cycle from p cannot increase p's total weight.



Proof of correctness: Preliminaries

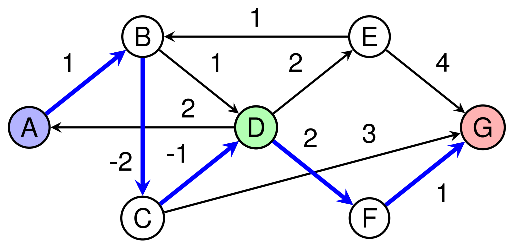
Claim (triangle inequality)

For any vertices a, b, c, $\delta(a,c) \leq \delta(a,b) + \delta(b,c)$.

Proof

Given a shortest path from a to b and a shortest path from b to c, combining these two paths gives a path from a to c with total weight $\delta(a,b) + \delta(b,c)$.

Note that this holds even if some edge weights are negative.



Proof of correctness: Preliminaries

Finally, an important property of relaxation, which can be proven by induction and using the triangle inequality, is called path-relaxation:

Claim (path-relaxation)

Assume that:

- $p = s \to v_1 \to \cdots \to v_k \to v$ is a shortest path from s to v;
- $s.d$ is initially set to 0 and $u.d$ is initially set to ∞ for all $u \neq s$;
- the edges in p are relaxed in the order they appear in p (possibly with other edges relaxed in between).

Then, at the end of this process, $v.d = \delta(s,v)$.

Proof: exercise.

Proof of correctness

Claim

If G does not contain a negative-weight cycle reachable from s, then at the completion of BellmanFord, $v.d = \delta(s,v)$ for all vertices v.

Proof

- Write $v_0 = s$, $v_m = v$. If v is reachable from s, there must exist a shortest path $v_0 \to v_1 \to \cdots \to v_m$.
- A shortest path cannot contain a cycle, so $m \leq V - 1$.
- In the i'th iteration of the for loop, the edge $v_{i-1} \to v_i$ is relaxed (among others).
- By the path-relaxation property, after $V - 1$ iterations, $v.d = \delta(s,v)$.
- So $V - 1$ iterations suffice to set $v.d$ correctly for all v.

□

Proof of correctness

Claim

If G does not contain a negative-weight cycle reachable from s, then BellmanFord does not exit with an error.

Proof

- By the triangle inequality, for all edges $u \to v$, $\delta(s, v) \le \delta(s, u) + w(u, v)$.
- By the claim on the previous slide, $v.d = \delta(s, v)$ for all vertices v.
- So, for all edges $u \to v$, $v.d \le u.d + w(u, v)$.
- So the check in step (7) of the algorithm never fails.

□

Proof of correctness

Claim

If G contains a negative-weight cycle reachable from s, then BellmanFord exits with an error.

Proof

- We will assume that G contains a negative-weight cycle reachable from s, and that BellmanFord does not exit with an error, and prove that this implies a contradiction.
- Let $v_0, \dots, v_k$ be a negative-weight cycle, where $v_k = v_0$.
- Then by definition $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$.
- As BellmanFord does not exit with an error, for all $1 \le i \le k$,

$$v_i.d \le v_{i-1}.d + w(v_{i-1}, v_i).$$

...

Proof of correctness

Claim

If G contains a negative-weight cycle reachable from s, then BellmanFord exits with an error.

Proof

- Summing this inequality over i between 1 and k,

$$\sum_{i=1}^{k} v_i.d \le \sum_{i=1}^{k} v_{i-1}.d + w(v_{i-1}, v_i) = \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$
$$< \sum_{i=1}^{k} v_{i-1}.d = \sum_{i=0}^{k-1} v_i.d.$$

- Subtracting $\sum_{i=1}^{k-1} v_i.d$ from both sides, we get $v_k.d < v_0.d$.
- But $v_0 = v_k$, so we have a contradiction.

□

Application 1: difference constraints

- A system of difference constraints is a set of inequalities of the form $x_i - x_j \le b_{ij}$, where x_i and x_j are variables and b_{ij} is a real number.
- For example:

$$x_1 - x_2 \le 5, \quad x_2 - x_3 \le -2, \quad x_1 - x_4 \le 0.$$

- Given a system of m difference constraints in n variables, we would like to find an assignment of real numbers to the variables such that the constraints are all satisfied, if such an assignment exists.
- For example, the above system is satisfied by $x_1 = 0$, $x_2 = -1$, $x_3 = 1$, $x_4 = 7$ (among other solutions).
- We will show that this problem can be solved using Bellman-Ford in time $O(nm + n^2)$.

Graph representation of difference constraints

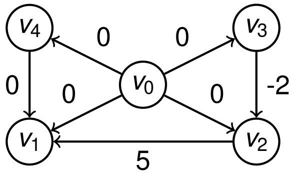
Given m difference constraints in n variables, we create a graph on $n+1$ vertices $v_0, \dots, v_n$ with $m+n$ edges where:

- for each constraint $x_i - x_j \leq b_{ij}$, we add an edge $v_j \to v_i$ with weight b_{ij}
- for all $1 \leq i \leq n$ there is an additional edge $v_0 \to v_i$ with weight 0.

For example:

$$x_1 - x_2 \leq 5, \quad x_2 - x_3 \leq -2, \quad x_1 - x_4 \leq 0$$

corresponds to



Claim

Let G be the graph corresponding to a system of difference constraints. If G does not contain a negative-weight cycle, the assignment $x_i = \delta(v_0, v_i)$, for all $1 \leq i \leq n$, is a valid solution to the system of constraints.

Proof

- We need to prove that

$$\delta(v_0, v_i) - \delta(v_0, v_j) \leq b_{ij}$$

  for all i, j in the list of constraints.
- This follows from the triangle inequality

$$\delta(v_0, v_i) \leq \delta(v_0, v_j) + \delta(v_j, v_i) \leq \delta(v_0, v_j) + w(v_j, v_i) = \delta(v_0, v_j) + b_{ij}$$

  and rearranging. □

Claim

Let G be the graph corresponding to a system of difference constraints. If G contains a negative-weight cycle, there is no valid solution to the system of constraints.

Proof (sketch)

- We prove the converse: if the system has a valid solution, there is no negative-weight cycle.
- Let $c = v_1, \dots, v_k, v_1$ be an arbitrary cycle on vertices $v_1, \dots, v_k$ (without loss of generality). This corresponds to the inequalities

$$x_2 - x_1 \leq b_{12}, \quad x_3 - x_2 \leq b_{23}, \quad \dots \quad , \quad x_1 - x_k \leq b_{k1}.$$

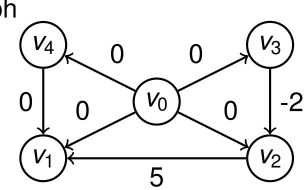
- If there is a valid solution x_i, then all the inequalities are satisfied.
- Summing the inequalities we get 0 for the left-hand side, and the weight of c for the right-hand side.
- So c has weight at least 0, and is not a negative-weight cycle. □

Example

The set of inequalities

$$x_1 - x_2 \leq 5, \quad x_2 - x_3 \leq -2, \quad x_1 - x_4 \leq 0$$

corresponds to the graph



with shortest paths

$$\delta(v_0, v_1) = 0, \quad \delta(v_0, v_2) = -2, \quad \delta(v_0, v_3) = 0, \quad \delta(v_0, v_4) = 0.$$

So

$$x_1 = 0, \quad x_2 = -2, \quad x_3 = 0, \quad x_4 = 0$$

is a solution to the constraints.

Solving difference constraints

- We can run Bellman-Ford with v_0 as the source.
- If there is a negative-weight cycle, the algorithm detects it (and we output "no solution"); otherwise, we output $x_i = \delta(v_0, v_i)$ as the solution.
- For a solution to a system of m difference constraints on n variables, the graph produced has $n+1$ vertices and $m+n$ edges.
- The running time of Bellman-Ford is thus $O(VE) = O(mn + n^2)$.
- This can be improved to $O(mn)$ time (CLRS exercise 24.4-5).

Application 2: Currency exchange

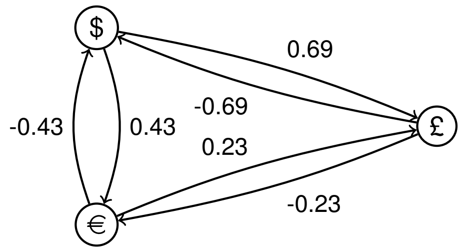
Imagine we have n different currencies, and a table T whose (i, j)'th entry T_{ij} represents the exchange rate we get when converting currency i to currency j. For example:

	£	$	€
£	1	1.61	1.18
$	0.62	1	0.74
€	0.85	1.35	1

- If we convert currency $i \to j \to k$, the rate we get is the product of the individual rates.
- If we convert $i \to j \to \cdots \to i$, and the product of the rates is greater than 1, we have made money by exploiting the exchange rates! This is called arbitrage.
- We can use Bellman-Ford to determine whether arbitrage is possible.

Application: Currency exchange

We produce a weighted graph G from the currency table, where the weight of edge $i \to j$ is $-\log_2 T_{ij}$. For example:



- Then the weight of a cycle $c_0 \to c_1 \to \cdots \to c_k$ (with $c_k = c_0$) is

$$-\sum_{j=1}^{k} \log_2 T_{c_j c_{j-1}} = -\log_2 \prod_{j=1}^{k} T_{c_j c_{j-1}}.$$

- This will be negative if and only if $\prod_j T_{c_j c_{j-1}} > 1$, i.e. the sequence of transactions corresponds to an arbitrage opportunity.
- So G has a negative-weight cycle if and only if arbitrage is possible.

Summary

- The Bellman-Ford algorithm solves the single-source shortest paths problem in time $O(VE)$.
- It works if the input graph has negative-weight edges, and can detect negative-weight cycles.
- Although the proof of correctness is a bit technical, the algorithm is easy to implement and doesn't use any complicated data structures.
- It can be used to solve a system of difference constraints and to determine whether arbitrage is possible.

Further Reading

- Introduction to Algorithms
  T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein.
  MIT Press/McGraw-Hill, ISBN: 0-262-03293-7.
  - Chapter 24 – Single-Source Shortest Paths
- Algorithms
  S. Dasgupta, C.H. Papadimitriou and U.V. Vazirani
  `http://www.cse.ucsd.edu/users/dasgupta/mcgrawhill/`
  - Chapter 4, Section 4.6 – Shortest paths in the presence of negative edges
- Algorithms lecture notes, University of Illinois
  Jeff Erickson
  `http://www.cs.uiuc.edu/~jeffe/teaching/algorithms/`
  - Lecture 19 – Single-source shortest paths

Biographical notes

Richard E. Bellman (1920–1984)

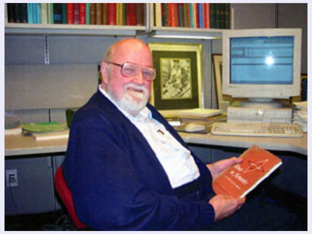
- American mathematician who worked at Princeton, Stanford, the RAND Corporation and the University of Southern California.
- Author of at least 621 papers and 41 books, including 100 papers after the removal of a brain tumour left him severely disabled.
- Winner of the IEEE Medal of Honor in 1979 for his invention of dynamic programming.


Pic: IEEE Global History Network

Biographical notes

Lester Ford, Jr. (1927–)

- Another American mathematician whose other contributions include the Ford-Fulkerson algorithm for maximum flow problems.
- His father was also a mathematician and, at one point, President of the Mathematical Association of America.


Pic: tangrammit.com