Shortest Paths

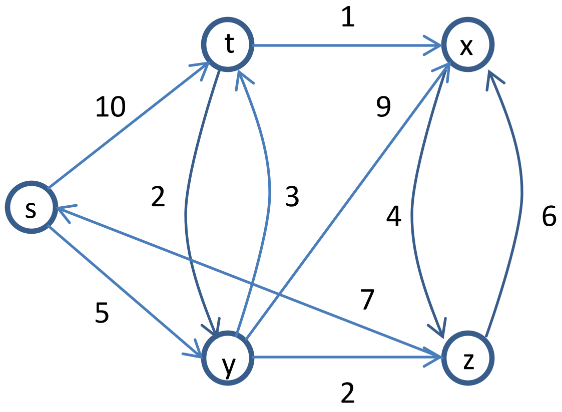
Summary

- Main problems
  - Shortest paths from a source to all nodes/shortest paths from all nodes to a destination
  - Shortest path from a source to a destination
  - Shortest paths from all sources to all destinations
- Main algorithms
  - Dijkstra’s algorithm and Bellman-Ford algorithm
  - Johnson’s algorithm
  - Floyd-Warshall algorithm

Shortest paths

- Every link in a network is assigned a **length**
- The length of a path is the **sum** of the lengths of its links
- A path is **preferred** to another if its length is smaller than that of the other
  - Distance is the length of a shortest path

Number of paths in a network can be super-exponential in the number of nodes: it is not a feasible to enumerate all paths and then select the ones of minimum length



Shortest paths: structure

Proposition SP1: if all cycles in the network have nonnegative length, then there is a shortest simple path from s to u, not longer than any walk from s to u.

Proof idea:
Any walk from s to u containing a circuit is not shorter than the walk with the circuit removed. Thus, a not-longer path can be obtained from a every walk. The number of simple paths is finite.

Proposition SP2: if all cycles in the network have nonnegative length, then every sub-path of a shortest-path is a shortest path.

Proof idea (contradiction):
Assume that there is a sub-path that is not shortest and substitute it by a shortest path to obtain a shorter walk from the source to the destination. Then use SP1.

How would you define "shortest paths" with cycles of negative length?

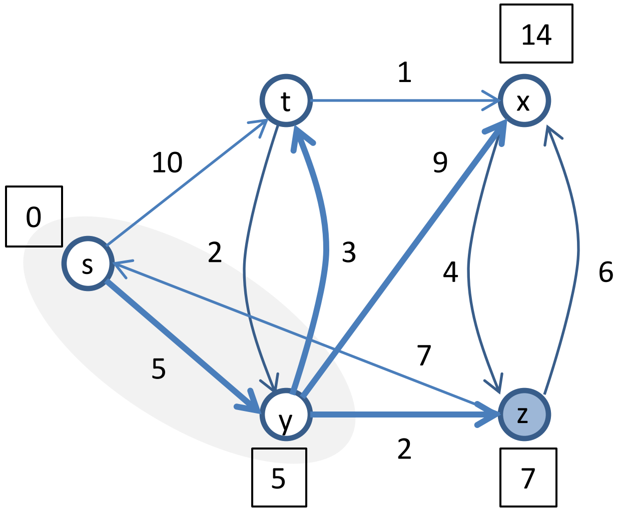
Dijkstra's algorithm: source to all

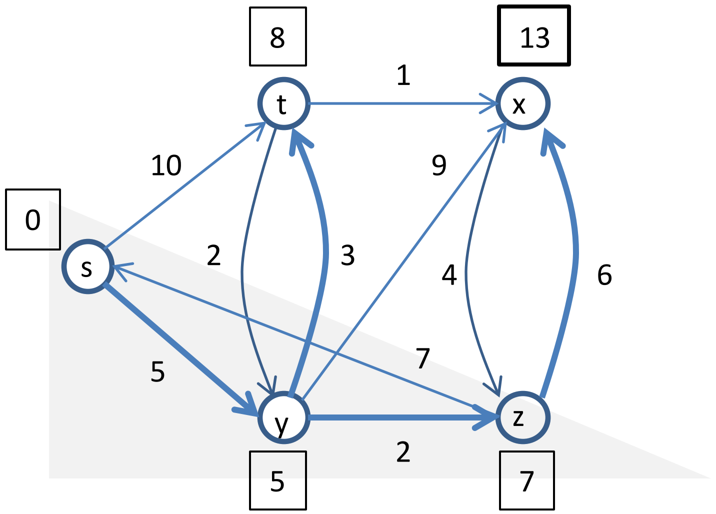
$l(u, v)$ – length of link uv
$d[u]$ – distance estimate from s to u

```
Dijkstra(G, l, s)
for each v
   d[v] := +∞; pred[v] := NIL
d[s] := 0; Q := V
while Q not empty
   extract from Q node u for which d[u] is smallest
   for each out-neighbor v of u
      if d[v] > d[u] + l(u,v)                    /* relaxation of link uv */
         d[v] := d[u] + l(u,v); pred[v] := u
```

1. *Does the algorithm really compute distances?*
2. *How extraction from Q is implemented?*
3. *What is the complexity of the algorithm?*

Dijkstra’s algorithm: iteration

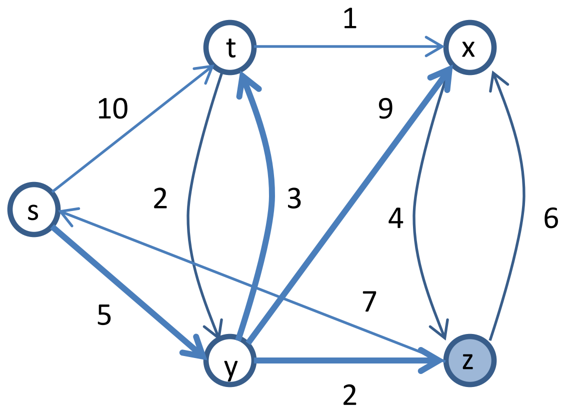




Nodes already extracted from *Q*

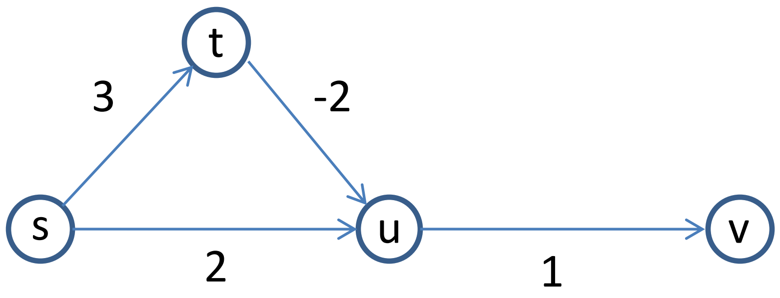
Estimated distances

Dijkstra’s algorithm: example



while loop	Q	d[s]	d[t]	d[x]	d[y]	d[z]
0	s,y,z,t,x	0	+∞	+∞	+∞	+∞
1	y,z,t,x	0	10	+∞	5	+∞
2	z,t,x	0	8	14	5	7
3	t,x	0	8	13	5	7
4	x	0	8	9	5	7
5		0	8	9	5	7

Lengths must be nonnegative



At the end of Dijkstra’s computation:

$d[s] \ = 0$
$d[t] \ = 3$
$d[u] = 1$
$d[v] = 3 \neq 2 = \delta(s, v)$

$\delta(s, v)$ – distance from s to v

Lower bound at all times

$l(u, v)$ - length of link uv
$\delta(s, v)$ - distance from s to v

Proposition LB: $d[v] \geq \delta(s, v)$ for all nodes v at all times during execution.

Proof (induction on the number of link relaxations):

- Initially, $d[x] \geq \delta(s, x)$ for all nodes x. (Prop. SP1)
- Link uv is relaxed and $d[v]$ is updated:
  $d[v] = d[u] + l(u, v)$
  $\geq \delta(s, u) + l(u, v)$ (induction)
  $\geq \delta(s, v)$. (triangle inequality)

1. $d[v]$ *is always the length of some path from* s *to* v
2. $d[v]$ *never increases during an execution*
3. *The proof does not make use of links having nonnegative length*

Upper bound after extraction

Proposition UB: If all links in the network are nonnegative, then $\delta(s,v) \geq d[v]$ when v is extracted from Q.

Proof (induction on the number of extractions from Q):

- Initially, $d[s] = 0 = \delta(s,s)$.
- Node v is extracted from Q**:**

  Let $PxyP'$ be a shortest path from s to v (Prop. SP1),
  with x the last node of $PxyP'$ not in Q when v is extracted:

$\delta(s,v) = l(PxyQ) = l(P) + l(x,y) + l(P')$

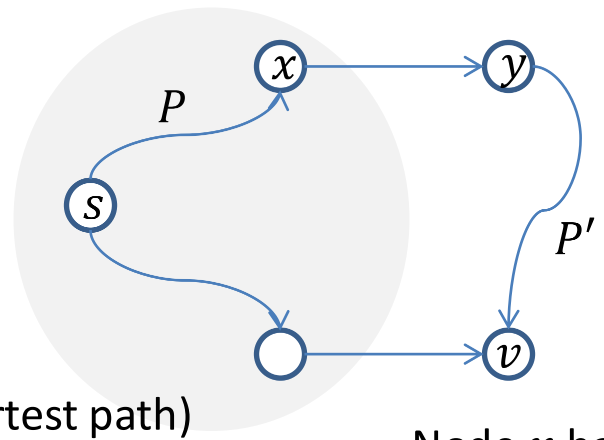
$\geq \delta(s,x) + l(x,y) + l(P')$ (P is not shorter than shortest path)

$\geq \delta(s,x) + l(x,y)$ (All links have nonnegative length)

$\geq d[x] + l(x,y)$ (Induction hypothesis)

$\geq d[y]$ (Relaxation of xy)

$\geq d[v]$ (v is extracted from Q before y)



Node v has just been extracted from Q

Dijkstra's algorithm: correctness

Proposition: If all links in the network are nonnegative, then Dijkstra's algorithm computes distances from source s to every other node.

Proof:
From Propositions LB and UP, $\delta(s, v) = d[v]$ when v is extracted from Q. Since, from the code, $d[v]$ never increases during execution and, from Proposition LB, $d[v] \geq \delta(s, v)$ at all times, $\delta(s, v) = d[v]$ at all times after v is extracted from Q.

Dijkstra's algorithm: complexity

Max n nodes in the heap: requires handles between nodes and heap positions

heap (priority queue)

Max m node replicas in the heap: discard replicas whose estimate is longer than distance

array

one **extractMin** per node:
O(n log(n)) total

max one **decreaseDist** per link:
O(m log(n)) total

O((n+m) log(n)) total

max m **extractMin** total:
O(m log(m)) total

max one **insert** per link:
O(m log(m)) total

O((n+m) log(n)) total

one **extractMin** per node:
n^2 total

O(n^2) total

Dijkstra's algorithm: summary

n – number of nodes
m – number of links

```
Dijkstra(G, l, s)
for each v
  d[v] := +∞; pred[v] := NIL
d[s] := 0; Q := V
while Q not empty
  extract from Q node u for which d[u] is smallest
  for each out-neighbor v of u
    if d[v] > d[u] + l(u,v)                /* relaxation of link uv */
      d[v] := d[u] + l(u,v); pred[v] := u
```

1. *Lengths must be nonnegative*
2. *Complexity:* $O(m \times \log n)$ *(binary heap);* $O(n^2)$ *(array)*

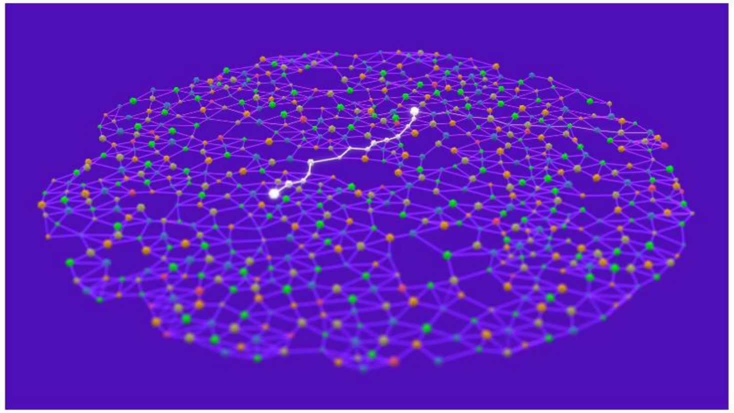
Still improving on Dijkstra’s algorithms

New Method Is the Fastest Way To Find the Best Routes

By Ben Brubaker

August 6, 2025

A canonical problem in computer science is to find the shortest route to every point in a network. A new approach beats the classic algorithm taught in textbooks.



DVDP for *Quanta Magazine*

If you want to solve a tricky problem, it often helps to get organized. You might, for example, break the problem into pieces and tackle the easiest pieces first. But this kind of sorting has a cost. You may end up spending too much time putting the pieces in order.

https://www.quantamagazine.org/new-method-is-the-fastest-way-to-find-the-best-routes-20250806/

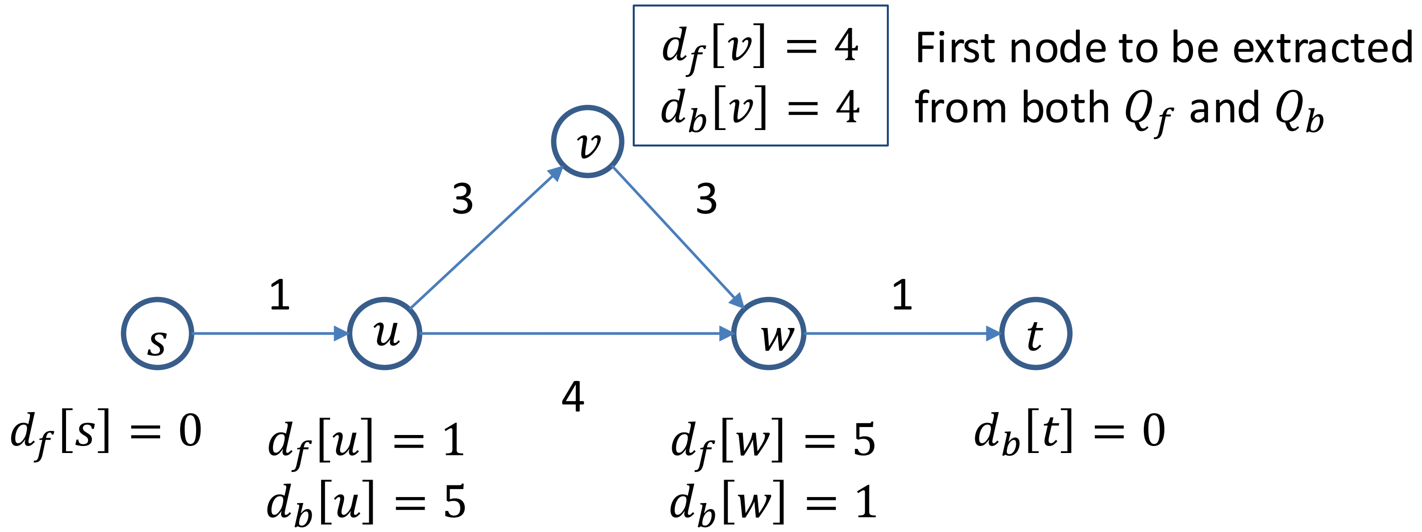
Shortest–path from source to destination: bidirectional Dijkstra's algorithm

- Alternate in any way
  - Forward Dijkstra's from s with distance estimates d_f
  - Backward Dijkstra's to t with distance estimates d_b
- Stopping condition
  - Same node x has been extracted from both Q_f and Q_b

Not as simple as it looks! Node x may not be on a shortest path from s to t

Searching for the shortest path

Alternate between forward search from s and backward search from t



Node v is the first node to be extracted from both Q_f and Q_b and is not on the shortest path from s to t

When a node has been extracted from both Q_f and Q_b find the node u^* with minimum $d_f[u] + d_b[u]$

Potential transformation

- Network $G = (V, E, l)$
- Potential h from nodes to real numbers
- Relabel link lengths from l to l' such that $l'(uv) = l(uv) + h(u) - h(v)$
- Shortest paths in $G = (V, E, l)$ coincide with shortest paths in $G' = (V, E, l')$

A judicious choice of potential h guides Dijkstra's algorithm from source to destination; for example, in a plane, $h(v)$ could be the Euclidean distance from v to the destination.

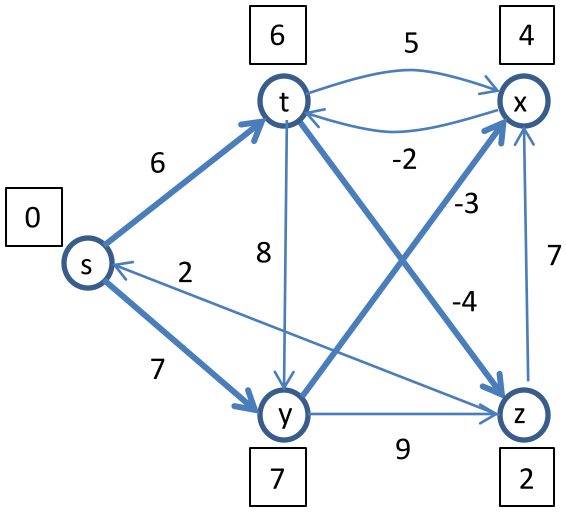
Bellman-Ford algorithm: source to all

n – number of nodes
m – number of links

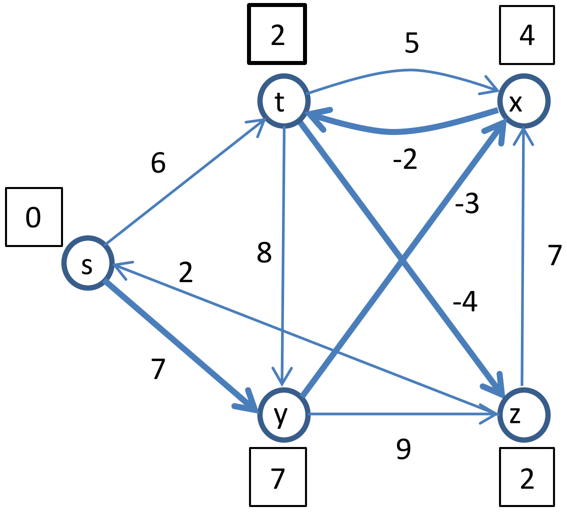
```
BellmanFord(G, l, s)
for each v
   d[v] := +∞; pred[v] := NIL
d[s] := 0
repeat n - 1 times
   for each uv
      if d[v] > d[u] + l(u,v)                /* relaxation of link uv */
         d[v] := d[u] + l(u,v); pred[v] := u
```

1. *Computes distances if all cycles have nonnegative length*
2. *Can it be extended to detect negative cycles?*
3. *Complexity:* $O(m \times n)$

Bellman-Ford: iteration



Second iteration

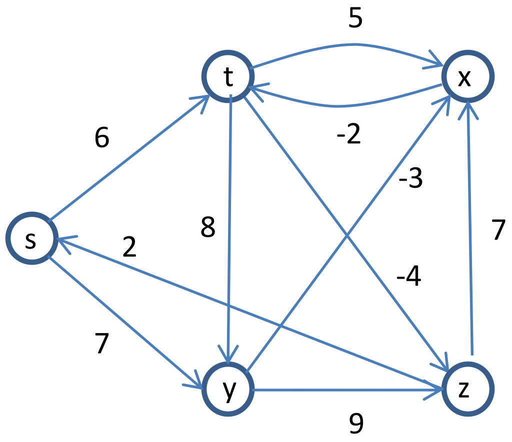


Third iteration

Order by which links are relaxed

tx
ty
tz
<u>xt</u>
yx
yz
zx
zs
st
sy

Bellman-Ford: example

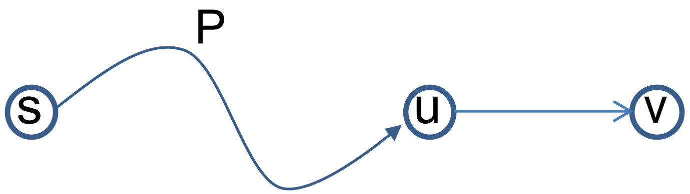


tx
ty
tz
xt
yx
yz
zx
zs
st
sy

repeat loop	d[s]	d[t]	d[x]	d[y]	d[z]
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	6	$+\infty$	7	$+\infty$
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	-2

Upper bound after *i*-th iteration

Proposition UP: If all cycles in the network are nonnegative, then if there is a shortest path with i links from s to v, then $\delta(s, v) \geq d[v]$ after iteration i.



Proof (induction on the number of iterations):

- Initially, $d[s] = 0 = \delta(s, s)$.
- Let Puv be a shortest path from s to v with i links (Prop. SP1)

$\delta(s, v) = l(Puv) = l(P) + l(u, v)$

$\geq \delta(s, u) + l(u, v)$ (P is not shorter than shortest path)

$\geq d[u] + l(u, v)$ (Induction hypothesis; P has $i - 1$ links)

$\geq d[v]$ (Relaxation of link uv)

Bellman-Ford algorithm: optimizations

```
BellmanFord(G, l, s)
for each v
  d[v] := +∞; pred[v] := NIL
d[s] := 0; done := FALSE; i := 1
while not done and i < n
  done := TRUE
  for each uv
    if d[v] > d[u] + l(uv)
      d[v] := d[u] + l(uv); pred[v] := u
      done := FALSE
  i := i +1
```

Terminate as soon as estimates no longer improve

```
BellmanFord(G, l, s)
for each v
  d[v] := +∞; pred[v] := NIL; scan[v] := FALSE
d[s] := 0; scan[s] := TRUE; done := FALSE; i := 1
while not done and i < n
  done := TRUE
  for each u
    if scan[u]
      for each v out-neighbor of u
        if d[v] > d[u] + l(uv)
          d[v] := d[u] + l(uv); pred[v] := u
          scan[v] := TRUE; done := FALSE
      scan[u] := FALSE
  i := i +1
```

Only scan nodes that have been updated in the previous iteration

Shortest-paths in a DAG

```
DAG-distance(G,l,s)
TopologicalSort(G)
for each v
    d[v] := +∞; pred[v] := NIL
endfor
d[s] := 0
for each u according to the topological order
  for each out-neighbor v of u
      if d[v] > d[u] + l(uv)
        d[v] := d[u] + l(uv), pred[v] := u
      endif
  endfor
endfor
```

Complexity O(n+m)

1. *What if the "lengths" are inside the nodes?*
2. *What if we want to find the longest path in a DAG?*

All pairs shortest-paths: Johnson's

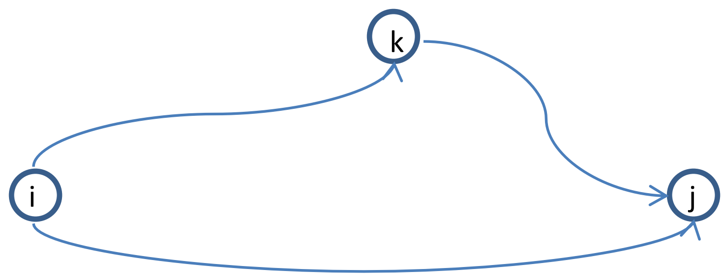
- All cycles of nonnegative length
- Insert a node q with a link of length 0 to every other node
- Run Bellman-Ford from q; let $\delta(q, u)$ be the distance from q to u
- Remove q; assign new length $l'(uv) \; = \; l(uv) + \delta(q, u) - \delta(q, v)$ to link uv (triangle inequality implies new lengths are all nonnegative)
- Run Dijkstra's algorithm once at each node
- Complexity $O(n \times m \log n)$

All pairs shortest paths: Floyd-Warshall

$V = \{1, \ldots, n\}$

$d^{(k)}(i,j)$ Distance from *i* to *j* with intermediate nodes only from set of nodes {1, …, *k*}

$$d^{(k)}(i,j) = \begin{cases} l(i,j), & k = 0 \\ \min\left(d^{(k-1)}(i,j), d^{(k-1)}(i,k) + d^{(k-1)}(k,j)\right), & k > 0 \end{cases}$$



Either a shortest path from i to j restricted to intermediate nodes {1, …, k} does not include pivot node k, or it does

Floyd-Warshal algorithm

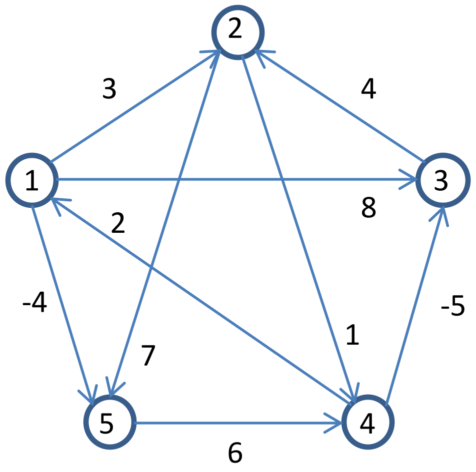
the length of every cycle
is non-negative

only one matrix d[,] is needed;
distances updated in place

```
FloydWarshall(G, l)
for i := 1 to n
  for j := 1 to n
    if i=j
      d[i,j] := 0
    else
      d[i,j] := l(i,j)
for k := 1 to n
  for i := 1 to n
    for j := 1 to n
      d[i,j] := min(d[i,j], d[i,k] + d[k,j])
```

complexity is $O(n^3)$

Floyd-Warshall: example



1

$$\begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

2

3

$$\begin{matrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

4

5

$$\begin{matrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{matrix}$$

Pivot node

Update from previous iteration

Recent fast algorithm for graphs with negative lengths

Finally, a Fast Algorithm for Shortest Paths on Negative Graphs

By Ben Brubaker

January 18, 2023

Researchers can now find the shortest route through a network nearly as fast as theoretically possible, even when some steps can cancel out others.

Samuel Velasco/Quanta Magazine

In algorithms, as in life, negativity can be a drag.

Consider the problem of finding the shortest path between two points on a graph — a network of nodes connected by links, or edges. Often, these edges aren't interchangeable: A graph could represent a road map on which some roads are slower than others or have higher tolls. Computer scientists account for these differences by pairing each edge with a "weight" that quantifies the cost of moving across that segment — whether that cost represents time, money or something else. Since the 1970s, they've known how to find shortest paths essentially as fast as theoretically possible, assuming all weights are positive numbers.

https://www.quantamagazine.org/finally-a-fast-algorithm-for-shortest-paths-on-negative-graphs-20230118/