

Course «Algorithms» · Lecture 9

All-Pairs Shortest Paths: Floyd’s Algorithm

Textbook: A. Levitin, Section 8.4

Today’s question

How far is it from every vertex to every other vertex?

- Warshall’s idea with two operations changed
- a full example, stage by stage
- recovering the actual paths
- negative weights and negative cycles

The problem and the weight matrix

Find a shortest path length for **every ordered pair** of vertices, ideally with the paths.

Case	**W[i, j]**
i = j	0
edge i → j exists	its weight
no edge	∞

Floyd’s algorithm is short, works on the matrix, and allows negative edges.

Recap: Warshall’s idea

Allow only vertices 1, …, k in the middle of a path. At stage k ask: **does the path use k?**

$$R^{(k)}[i,j] = R^{(k-1)}[i,j] \ \vee \ \left(R^{(k-1)}[i,k] \ \wedge \ R^{(k-1)}[k,j]\right)$$

Today we keep this structure and ask **how short** instead of **whether**.

The subproblem

$$D^{(k)}[i, j] = \text{length of a shortest path } i \to j \text{ with intermediates in } \{1, \ldots, k\}$$

$$D^{(0)} = W, \qquad D^{(n)} = \text{matrix of shortest distances}$$

A shortest path either avoids k or passes through it once. With no negative cycles, repeating a vertex never helps.

Floyd’s recurrence

Avoid k:
$D^{(k-1)}[i, j]$

Go through k: both halves must be shortest
$D^{(k-1)}[i, k] + D^{(k-1)}[k, j]$

$$D^{(k)}[i, j] = \min \left\{ D^{(k-1)}[i, j], \quad D^{(k-1)}[i, k] + D^{(k-1)}[k, j] \right\}$$

Keep the best path we had, or go through k, whichever is shorter.

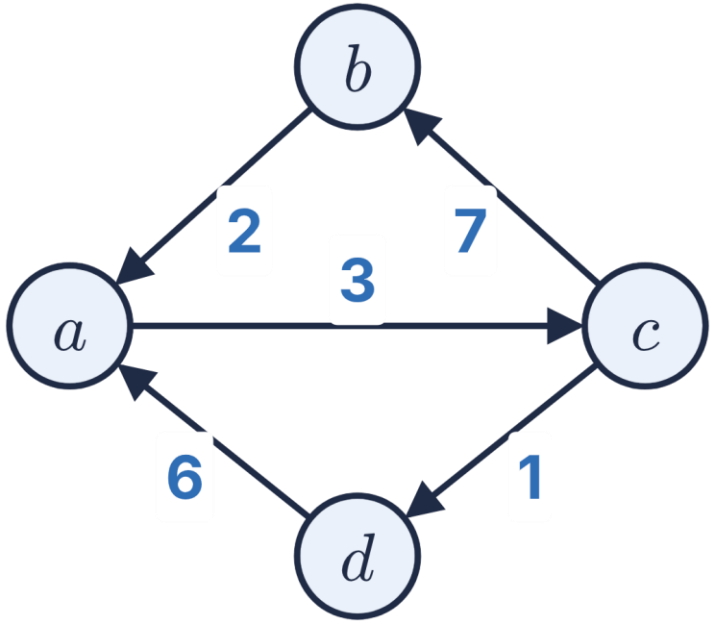
The algorithm: Warshall with two swaps

```
Floyd(W[1..n, 1..n])
  D ← W
  for k ← 1 to n do
    for i ← 1 to n do
      for j ← 1 to n do
        D[i, j] ← min(D[i, j], D[i, k] + D[k, j])
  return D
```

$$\{\vee,\ \wedge\} \quad \Rightarrow \quad \{\min,\ +\}$$

The same loop with (max, ×) gives the most reliable path, with (max, min) the widest path. **k must be outermost.**

Example: the graph and $D^{(0)} = W$



	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Stage k = a

Into a: from b and d. Out of a: to c.

$$b \to a \to c: \; 2+3=5, \qquad d \to a \to c: \; 6+3=9$$

$D^{(1)}$	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	0

Stage k = b

Only c has an edge into b.

$$c \to b \to a: \;\; 7 + 2 = 9$$

c → b → c costs 12: no better than 0.

$D^{(2)}$	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

Stage k = c

Column c: 3 (a), 5 (b), 9 (d). Row c: (9, 7, 0, 1).

$$D[a,b] = 3 + 7 = 10, \qquad D[a,d] = 3 + 1 = 4$$

$$D[b,d] = 5 + 1 = 6, \qquad D[d,b] = 9 + 7 = 16$$

b → c → a = 14 and d → c → a = 18 are worse.

$D^{(3)}$	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	9	7	0	1
d	6	16	9	0

Stage k = d: the final distances

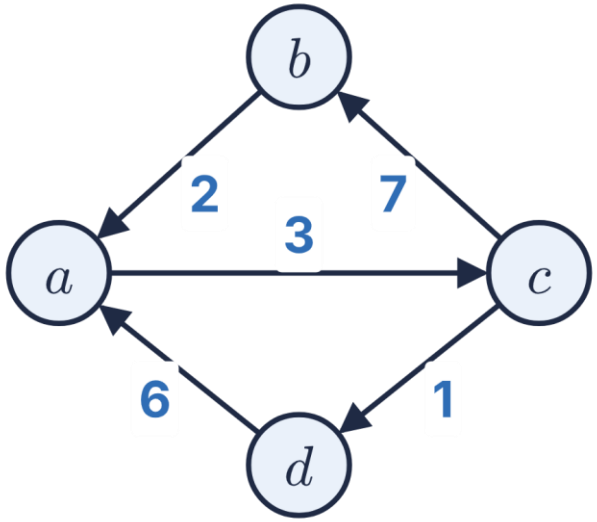
Column d: 4 (a), 6 (b), 1 (c). Row d: (6, 16, 9, 0).
One improvement:

$$D[c, a] = 1 + 6 = 7 < 9$$

c → d → b would be 17, worse than 7.

$D^{(4)}$	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

Sanity check against the picture



- c to a: c → d → a = 1 + 6 = **7**, better than c → b → a = 9
- d to b: d → a → c → b = 6 + 3 + 7 = **16**, the only way

Getting the actual paths

Keep a second matrix P: the vertex that last improved the pair.

$$D[i,k] + D[k,j] < D[i,j] \quad \Rightarrow \quad P[i,j] \leftarrow k$$

P	a	b	c	d
a	–	c	–	c
b	–	–	a	c
c	d	–	–	–
d	–	c	a	–

A dash means “no improvement”: the best path is the direct edge.
Extra memory $\Theta(n^2)$, same running time.

Recovering the path from d to b

1. P[d, b] = c → path(d, c), then path(c, b)
2. P[d, c] = a → path(d, a), then path(a, c): both direct edges
3. P[c, b] is empty → the edge c → b

$$d \to a \to c \to b, \qquad 6 + 3 + 7 = 16$$

Each path is printed in time proportional to its length.

Negative weights and negative cycles

- Negative edges are fine, as long as there is **no negative cycle**
- With a negative cycle, “shortest” has no meaning: go around again and again
- Normally D[i, i] = 0; on a negative cycle it drops below zero

$$\text{negative cycle} \iff D[i,i] < 0 \ \text{ for some } i$$

After the algorithm, just check the diagonal.

How does it compare?

$$\text{time} = \Theta(n^3), \qquad \text{space} = \Theta(n^2)$$

Method	Time	Remarks
Floyd	Θ(n³)	negative edges OK; tiny code
Dijkstra from each vertex	n × Dijkstra	non-negative weights; better for sparse graphs

Correctness invariant: after stage k, D holds shortest distances through {1, …, k}. Proof by induction on k.

Summary

$$D^{(k)}[i,j] = \min\left\{ D^{(k-1)}[i,j],\quad D^{(k-1)}[i,k] + D^{(k-1)}[k,j] \right\}$$

- Warshall's code with {∨, ∧} replaced by {min, +}; k outermost
- Θ(n³) time, one n × n matrix updated in place
- Matrix P recovers the paths
- Negative cycle ⇔ a negative diagonal entry
- Dense graphs: Floyd; sparse: Dijkstra from each vertex

Review questions

1. Derive Floyd’s recurrence. Where is the principle of optimality used?
2. Apply Floyd’s algorithm to edges a → b (4), a → c (1), c → b (2), b → d (5), c → d (8). Show all matrices.
3. What is a negative cycle, and how does Floyd’s algorithm detect one?