Algorithmentheorie

Daniel Neuen (Universität Bremen)

WiSe 2023/24

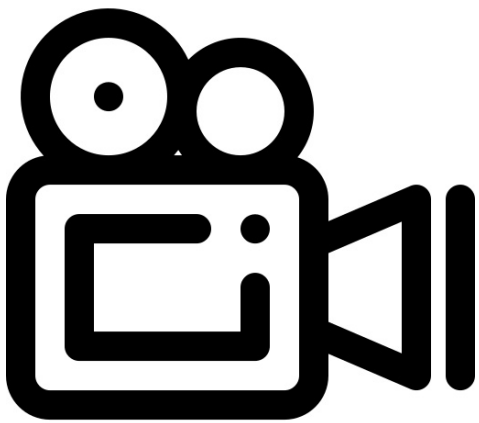
Dynamische Programmierung (Teil 2)

10. Vorlesung

Aufzeichnung der Vorlesung

Diese Vorlesung wird aufgezeichnet und live gestreamt.

- Aufzeichnungen nur der Lehrenden durch sich selbst.
- Bei Rückfragen aus dem Auditorium und Diskussion bitte deutlich anzeigen, falls das Mikro stumm geschaltet werden soll.



Organisatorisches

Aufgaben zur Klausurvorbereitung:

- separates Blatt mit Aufgaben zur Klausurvorbereitung
- seit Montag im StudIP verfügbar
- Aufgaben decken Stoff der gesamten Vorlesung ab
- Musterlösungen zum Ende der Vorlesungszeit
- Keine Abgabe bzw. Korrektur!

Wiederholung: Dynamische Programmierung (DP)

Designprinzip (Paradigma) für den Entwurf von effizienten Algorithmen

- **Andere Paradigmen:** Greedy, Divide-and-Conquer, Brute-Force

Designprinzip: Dynamische Programmierung

- Teile Probleminstanz in kleinere Teilprobleme auf.
- Löse kleine Teilprobleme durch erschöpfende Suche.
- Löse größere (Teil)probleme mit Hilfe von Lösungen für kleinere Teilprobleme.
- Speichere Teillösungen um mehrfache Berechung zu vermeiden.

  $\rightarrow$ zentraler Unterschied zu Divide-and-Conquer

Wiederholung: Fibonacci-Folge via Divide-and-Conquer

Fibonacci-Folge:

Die Fibonacci-Folge $f_1, f_2, f_3, \ldots$ ist die unendliche Folge von natürlichen Zahlen, die durch das rekursive Bildungsgesetz

- $f_n = f_{n-1} + f_{n-2}$ für $n > 2$

mit den Anfangswerten

- $f_1 = 1$ und $f_2 = 1$

definiert ist.

Fibonacci mit Divide-and-Conquer

```
1 Function Fib(n):
2     if n ≤ 2 then
3         return 1
4     else
5         return Fib(n − 1) + Fib(n − 2)
```

Wiederholung: Fibonacci-Folge via Divide-and-Conquer

Fibonacci mit Divide-and-Conquer

```
1 Function Fib(n):
2     if n ≤ 2 then
3         return 1
4     else
5         return Fib(n − 1) + Fib(n − 2)
```

Laufzeit (Anzahl Funktionsaufrufe):

- $T(1) = T(2) = 1$.
- $T(n) = 1 + T(n-1) + T(n-2)$.
- $T(n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right] \approx 1.45 \cdot 1.62^n$

Wiederholung: Dynamische Programmierung (DP)

Vermeide mehrfache Berechnung gleicher Funktionswerte durch Speichern bereits berechneter Werte:

- Verwende DP-Tabelle M um Zwischenergebnisse zu speichern.

  Fibonacci:

  $M[i] :=$ Zwischenergebnis f_i

- Berechne und speichere zunächst die Basisfälle.

  Fibonacci:

  ```
  M[]=new array;      M[1]=1; M[2]=1;
  ```

- Berechne den Rest der Tabelle entweder rekursiv (Memoization) oder iterativ (Tabularization).
- Größter Eintrag der DP-Tabelle enthält am Ende das Gesamtergebnis (**Fibonacci**: $M[n]$).

Wiederholung: Memoization

Entwurfsprinzip Memoization:

- Rekursives Programm
- Speichere die Lösungen der Teilprobleme
- Teste bei Funktionsaufruf zunächst, ob der Wert der Funktion bereits berechnet wurde und gebe im positiven Fall den gespeicherten Wert zurück. Sonst setze die Rekursion zur Berechnung fort.

Fibonacci mit Memoization

```
1 Initialisiere Array M[·] und setze M[1] := 1 und M[2] := 1
2 Function Fib(n):
3     if M[n] undefiniert then
4         M[n] := Fib(n − 1) + Fib(n − 2)
5     return M[n]
```

Wiederholung: Tabularization

Entwurfsprinzip Tabularization:

- Iteratives Programm
- Berechne bottom-up die Werte der Teillösungen und speichere sie in einer Tabelle. Wenn ein Teilproblem gelöst werden soll sind bereits die benötigten Teillösungen berechnet.

Fibonacci mit Tabularization

```
1 Initialisiere Array M[·] und setze M[1] := 1 und M[2] := 1
2 Function Fib(n):
3     for i = 3, ..., n do
4         M[i] := M[i − 1] + M[i − 2]
5     return M[n]
```

Wiederholung: Dynamische Programmierung

- Wir betrachten eine Reihe von Beispiel-Problemen
- Dabei benutzen wir Tabularization, d.h., wir erstellen eine DP-Tabelle und berechnen alle Einträge
- Beachte: Häufig brauchen wir Tabellen mit Dimension 2 oder 3
- Schwierigkeit: Welche Teilergebnisse sollen in der Tabelle gespeichert werden?
- In der Tabelle berechnen wir zunächst Basisfälle. Die anderen Einträge werden sukzessive aus den "vorherigen" Einträgen berechnet

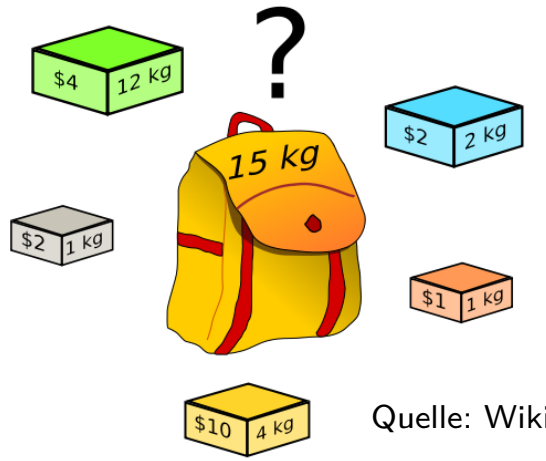
Das (ganzzahlige) Rucksackproblem

Das Rucksackproblem (Knapsack Problem)

Gegeben: Rucksack mit Gewichtsschranke $B \in \mathbb{N}$ (budget)
Menge $I = \{1, \ldots, n\}$ von Gegenständen mit

- Gewicht $w_i \in \mathbb{N}, i \in I$ (weight)
- Wert/Profit $v_i \in \mathbb{N}, i \in I$ (value)

Gesucht: Teilmenge $K \subseteq I$ sodass $\sum_{i \in K} w_i \leq B$ und $\sum_{i \in K} v_i$ maximal.



Quelle: Wikipedia

Ein Greedy Algorithmus

1. Entferne alle Gegenstände, die nicht passen ($w_i > B$).
2. Indiziere Gegenstände absteigend nach Dichte $d_i = \frac{v_i}{w_i}$.
3. **For** $i = 1, \ldots, n$ **do**
   Füge Gegenstand dem Rucksack hinzu, wenn er passt.

Beispiel mit Rucksackkapazität $B = 6$ und Gegenständen nach Dichte absteigend sortiert (w_i, v_i): $(1,1), (14,7), (9,4), (7,3), (5,2)$

- Greedy Algorithmus ist optimal für die fraktionale Variante (VL 4)
- Greedy Algorithmus nicht optimal für die ganzzahlige Variante
- Greedy Algorithmus beliebig schlecht
  Beispiel: $(1,1), (B, B-\epsilon)$ für ein kleines $\epsilon > 0$

DP-Tabelle für das Rucksack Problem

DP-Tabelle (Teilprobleme):
Sei $M[i, b]$ der Wert der optimalen Rucksacklösung aus Gegenständen in $\{1, 2, \ldots, i\}$ für Kapazität b.

Basisfälle:

$$M[i, 0] = 0, \qquad \forall i \in I$$
$$M[0, b] = 0, \qquad b \in \{1, 2, \ldots, B\}$$

Wie kann man $M[i, b]$ rekursiv berechnen? (aus Teillösungen)

1. Fall Wenn i nicht in Lösung zu $M[i, b]$ enthalten, dann $M[i, b] = M[i - 1, b]$.
2. Fall Wenn i in Lösung enthalten (nur möglich wenn $w_i \leq b$), dann $M[i, b] = v_i + M[i - 1, b - w_i]$.

Welcher Fall? Max Wert!

$$M[i, b] = \max\{M[i - 1, b], v_i + M[i - 1, b - w_i]\}$$

Dynamische Programmierung (DP) für Knapsack

Knapsack Algorithmus (DP)

Input : n Gegenstände mit Gewicht w_i und Wert v_i, B Kapazität vom Rucksack

Output : Wert einer optimalen Lösung

```
1  Initialisiere Array M[·, ·] mit Dimension n × B
2  Setze M[i, 0] := 0        ∀i ∈ {1, ..., n}
3  Setze M[0, b] := 0        ∀b ∈ {1, 2, ..., B}
4  for i = 1, ..., n do
5      for b = 1, 2, ..., B do
6          if w_i ≤ b then
7              M[i, b] := max{M[i − 1, b], v_i + M[i − 1, b − w_i]}
8          else
9              M[i, b] := M[i − 1, b]
10 return M[n, B]
```

Analyse

Satz

Der DP-Algorithmus findet eine optimale Rucksacklösung mit Laufzeit $\mathcal{O}(n \cdot B)$.

Dieser Algorithmus läuft nicht in polynomieller Zeit, da die Kodierungslänge der Zahl B in Binärdarstellung $1 + \log B$ ist.

$\longrightarrow$ Pseudopolynomielle Laufzeit

Berechnung der eigentlichen Lösung

- Nach Ausführung des DP enthält $M[n, B]$ den optimalen Zielfunktionswert.
- Wie kann aus der DP-Tabelle M die zugehörige Lösung $S \subseteq \{1, \ldots, n\}$ berechnet werden?

Rekonstruktion der Lösung aus M via Backtracking

```
1 i := n und b := B
2 S := ∅
3 while i ≥ 0 do
4     if M[i, b] ≠ M[i − 1, b] then
5         S := S ∪ {i}
6         i := i − 1 und b := b − w_i
7     else
8         i := i − 1
```

Levenshtein Distanz für Strings

Strings

Sei Σ eine endliche Menge. Ein String (Wort) über Σ ist eine Folge $w \in \Sigma^*$.

Beispiel:

- $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$
- $w = \mathtt{bittes}$

In vielen Anwendungen wollen dir die Distanz zwischen Strings bestimmen.

Beispiel:

- Rechtschreibprüfung
- Korrekturvorschläge (engl.): `buttes`, `bitted`, `bitts`, `bites`, `bitters`, `bitten`, `bitter`, ...
- Wir geben korrekt geschriebene Wörter aus, die möglichst kleine Distanz zu w haben

Levenshtein Distanz für Strings

Es gibt verschiedene Möglichkeiten, die Distanz zwischen zwei Wörtern zu messen.

Wir betrachten die Levenshtein Distanz.

Elementare Operationen:

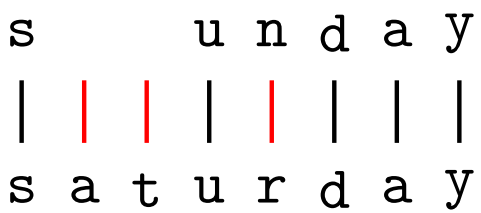
- Einen Buchstaben an beliebiger Stelle einfügen (z.B. `bittes` $\rightarrow$ `bitters`)
- Einen Buchstaben an beliebiger Stelle löschen (z.B. `bittes` $\rightarrow$ `bites`)
- Einen Buchstaben an beliebiger Stelle ersetzen (z.B. `bittes` $\rightarrow$ `buttes`)

Levenshtein Distanz für Strings

Definition

Seien $x, y \in \Sigma^*$ zwei Strings. Die Levenshtein Distanz $lev(x, y)$ ist die minimale Anzahl an elementaren Operationen, um x in y zu überführen.

Bsp: $lev(\texttt{sunday}, \texttt{saturday}) = 3$



Wir können die Levenshtein Distanz von zwei Strings $x, y \in \Sigma^*$ mit dynamischer Programmierung berechnen.

DP-Tabelle für die Levenshtein Distanz

Sei $x = x_1 x_2 \dots x_n$ and $y = y_1 y_2 \dots y_m$ wobei $x_i, y_j \in \Sigma$.

DP-Tabelle (Teilprobleme):
Sei $M[i,j] := lev(x_1 \dots x_i, y_1 \dots y_j)$ die Levenshtein Distanz zwischen den Teilwörtern $x_1 \dots x_i$ und $y_1 \dots y_j$.

Basisfälle:

$$M[i,0] = i, \qquad \forall i \in \{1, \dots, n\}$$
$$M[0,j] = j, \qquad \forall j \in \{1, \dots, m\}$$

Wie kann man $M[i,j]$ rekursiv berechnen? (aus Teillösungen)

1. Fall Der Buchstabe x_i wird gelöscht. Dann ist $M[i,j] = 1 + M[i-1,j]$.
2. Fall Der Buchstabe y_j wird eingefügt. Dann ist $M[i,j] = 1 + M[i,j-1]$.

DP-Tabelle für die Levenshtein Distanz

Wie kann man $M[i,j]$ rekursiv berechnen? (aus Teillösungen)

3. Fall Beide Buchstaben bleiben erhalten. Dann ist

$$M[i,j] = M[i-1,j-1] + d(x_i, y_j)$$

wobei $d(x_i, y_j) = 0$, falls $x_i = y_j$, und $d(x_i, y_j) = 1$ sonst.

Welcher Fall? Min Wert!

$$M[i,j] = \min\{1 + M[i-1,j], 1 + M[i,j-1], M[i-1,j-1] + d(x_i, y_j)\}$$

DP für Levenshtein Distanz

Levenshtein Distanz (DP)

Input : Zwei Strings $x = x_1 \dots x_n$ und $y = y_1 \dots y_m$
Output : $lev(x, y)$

```
1  Initialisiere Array M[·,·] mit Dimension (n+1) × (m+1)
2  Setze M[i,0] := i        ∀i ∈ {1,...,n}
3  Setze M[0,j] := j        ∀j ∈ {1,...,m}
4  for i = 1,...,n do
5      for j = 1,...,m do
6          if x_i = y_j then
7              d := 0
8          else
9              d := 1
10         M[i,j] := min{1 + M[i−1,j], 1 + M[i,j−1], d + M[i−1,j−1]}
11 return M[n,m]
```

Beispiel

$x =$ `sunday` und $y =$ `saturday`

$M[\cdot,\cdot]$		s	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
s	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Beispiel

$x = \texttt{sunday}$ und $y = \texttt{saturday}$

$M[\cdot,\cdot]$		s	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
s	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

$$2 = \min\{2 + 1, 3 + 1, 2 + 0\}$$

Beispiel

$x =$ `sunday` und $y =$ `saturday`

$M[\cdot,\cdot]$		s	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
s	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

$$4 = \min\{3 + 1, 5 + 1, 4 + 1\}$$

Analyse

Satz

Der DP-Algorithmus berechnet die Levenshtein Distanz von zwei Strings x, y in Zeit $\mathcal{O}(n \cdot m)$, wobei n die Länge von x und m die Länge von y ist.

Wie üblich können wir die elementaren Operationen, die x in y überführen, mit Backtracking berechnen.

Kürzeste Wege

zwischen allen Paaren von Knoten:

Floyd-Warshall Algorithmus

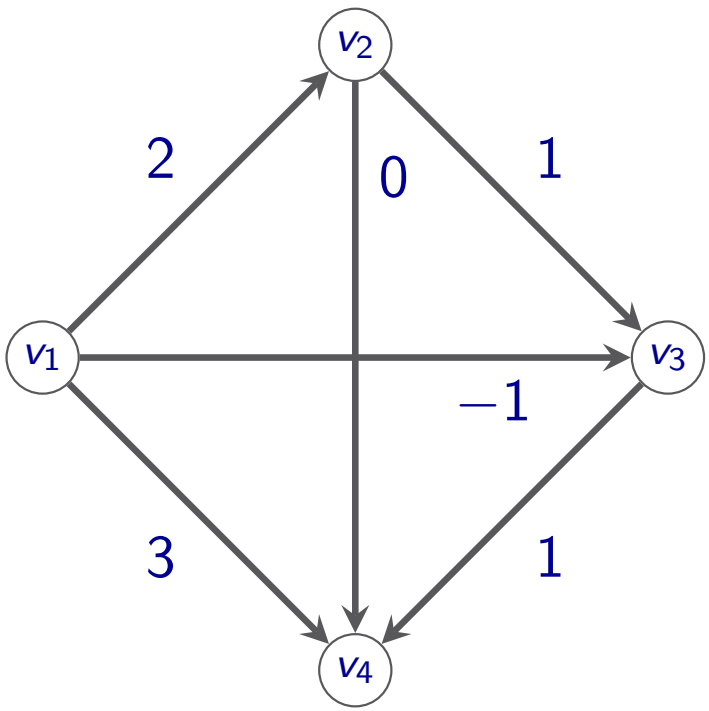
All-Pairs Shortest Paths Problem

- Finde kürzesten Weg zwischen Knoten u und v für alle $u, v \in V$.
- Wir könnten den Algorithmus von Bellman-Ford n mal anwenden (jeden Knoten $v \in V$ einmal als Startknoten s): Laufzeit $\mathcal{O}(n^2 m)$.
- Besser: **dynamische Programmierung**

Rekursion über Distanzen

Die Knotenmenge sei $V = \{v_1, v_2, \ldots, v_n\}$ (durchnummeriert).

Matrix $D_k[i, j]$: Länge eines kürzesten v_i-v_j-Weges, der nur Zwischenknoten aus $\{v_1, \ldots, v_k\}$ benutzt, bzw. ∞ falls kein solcher Weg existiert.



$$D_0[1,4] = 3$$
$$D_1[1,4] = 3$$
$$D_2[1,4] = 2$$
$$D_3[1,4] = 0$$

Bellmann Gleichungen

Die Knotenmenge sei $V = \{v_1, v_2, \ldots, v_n\}$.

Matrix $D_k[i,j]$: Länge eines kürzesten v_i-v_j-Weges, der nur Zwischenknoten aus $\{v_1, \ldots, v_k\}$ benutzt, bzw. ∞ falls kein solcher Weg exist.

Bellman Gleichungen

$$D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$$

mit den Basisfällen

$$D_0[i,j] = \begin{cases} c(v_i, v_j), & \text{für alle } (v_i, v_j) \in E \\ \infty & \text{sonst.} \end{cases}$$

Iterative Berechnung über Dynamisches Programm.

Algorithmus von Floyd und Warshall

Floyd-Warshall Algorithmus

Input : gewichteter Digraph $G = (V, E, c)$
Output: für alle $v_i, v_j \in V$ die Distanz $d(v_i, v_j)$

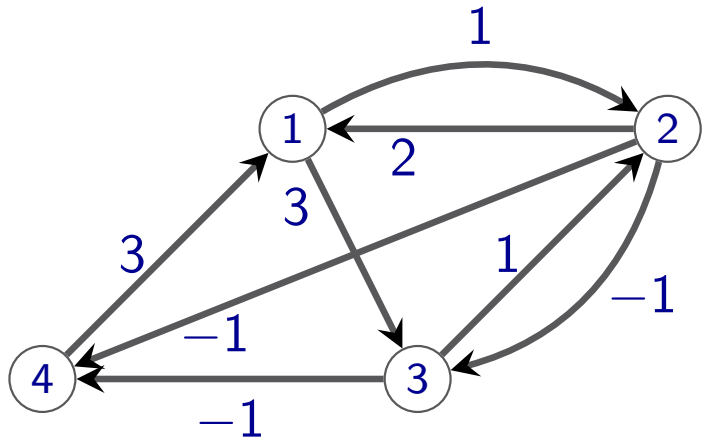
1 Initialisiere $D_0[i,j] := c(v_i, v_j)$, falls $(v_i, v_j) \in E$, und $D_0[i,j] := \infty$
  sonst
2 **for** $k = 1, \ldots, n$ **do**
3 **for** $i = 1, \ldots, n$ **do**
4 **for** $j = 1, \ldots, m$ **do**
5 $D_k[i,j] := \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
6 **return** $D_n[]$

Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



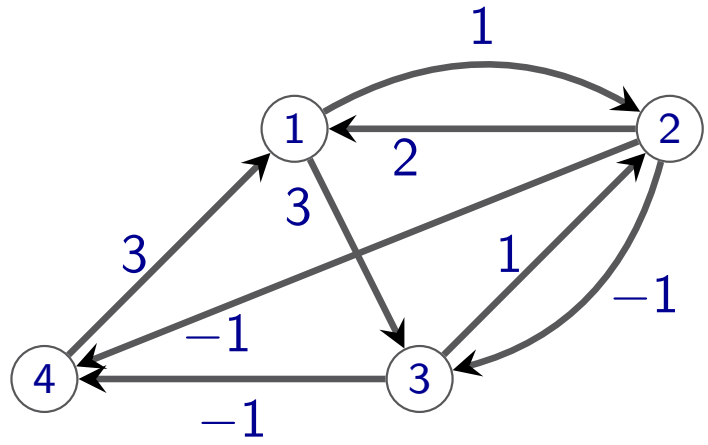
Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & & & \\ \infty & & & \\ 3 & & & \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

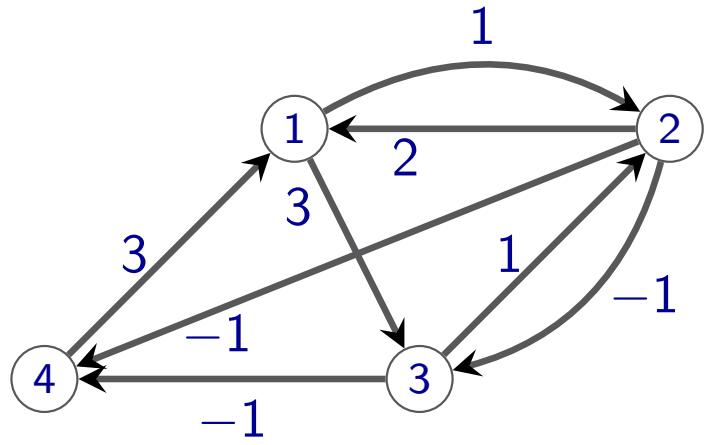
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & & \\ \infty & & & \\ 3 & & & \end{bmatrix}$$

$$0 = \min\{0, 2 + 1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

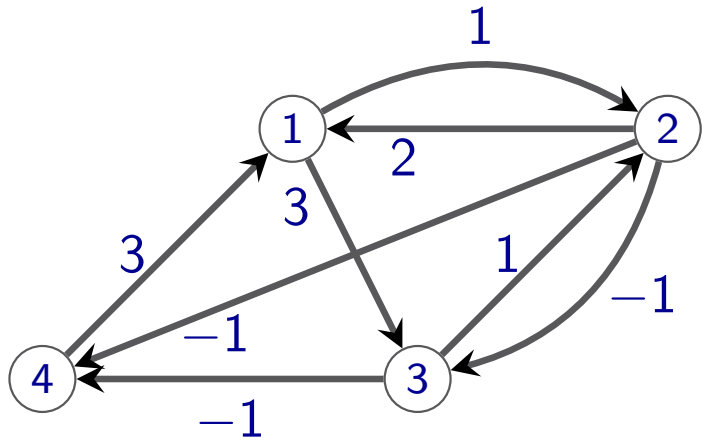
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & \\ \infty & & & \\ 3 & & & \end{bmatrix}$$

$$-1 = \min\{-1, 2 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

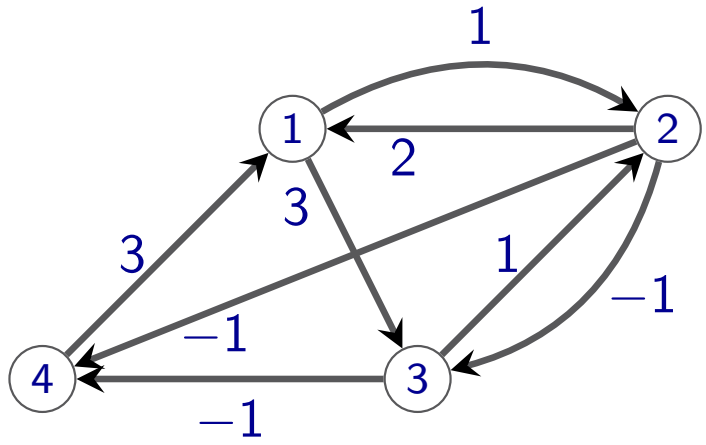
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & & & \\ 3 & & & \end{bmatrix}$$

$$-1 = \min\{-1, 2 + \infty\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

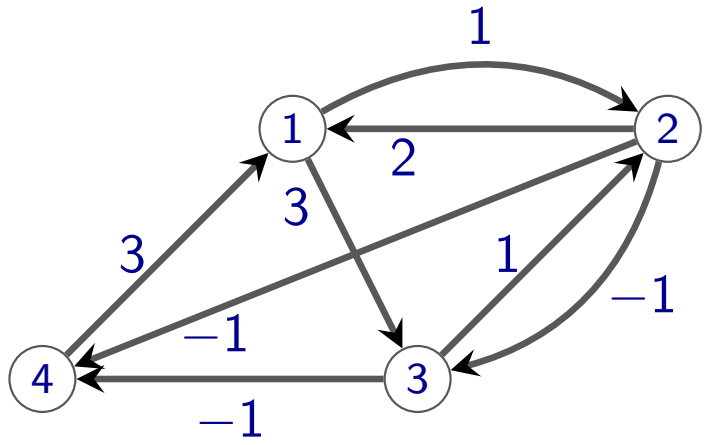
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & & \\ 3 & & & \end{bmatrix}$$

$$1 = \min\{1, \infty + 1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

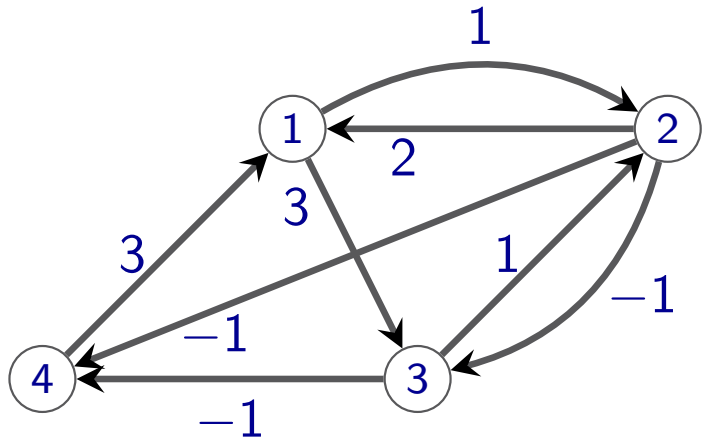
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & \\ 3 & & & \end{bmatrix}$$

$$0 = \min\{0, \infty + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

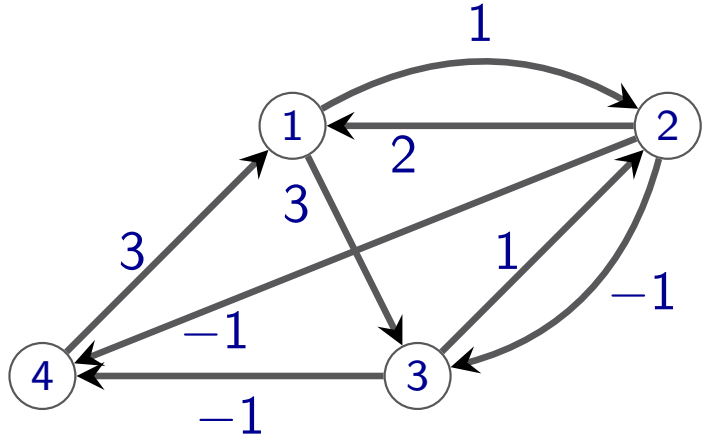
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & & & \end{bmatrix}$$

$$-1 = \min\{-1, \infty + \infty\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

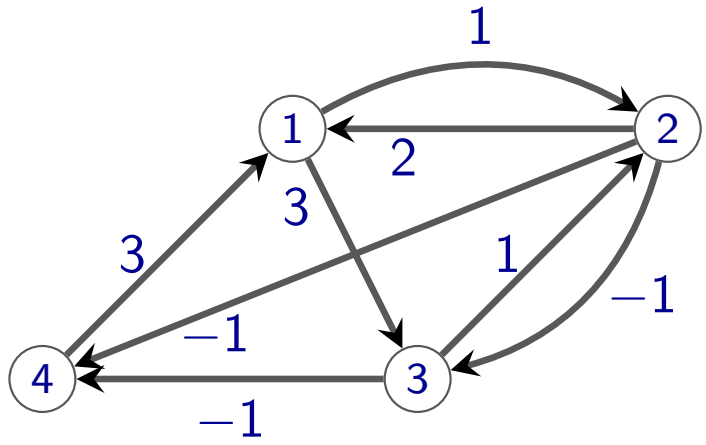
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & & \end{bmatrix}$$

$$4 = \min\{\infty, 3 + 1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

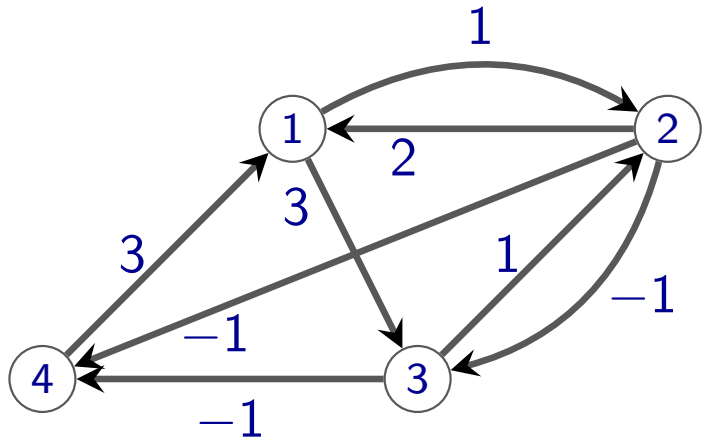
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & \end{bmatrix}$$

$$6 = \min\{\infty, 3 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

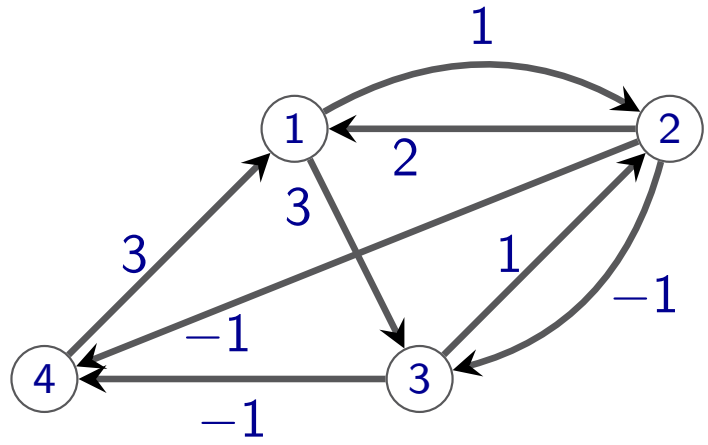
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$0 = \min\{0, 3 + \infty\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

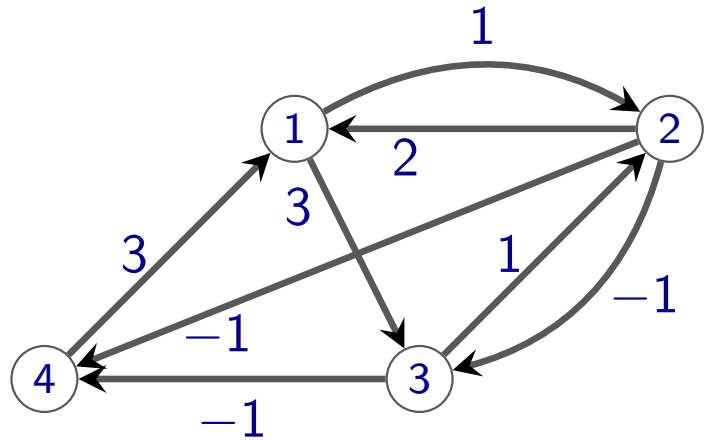
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

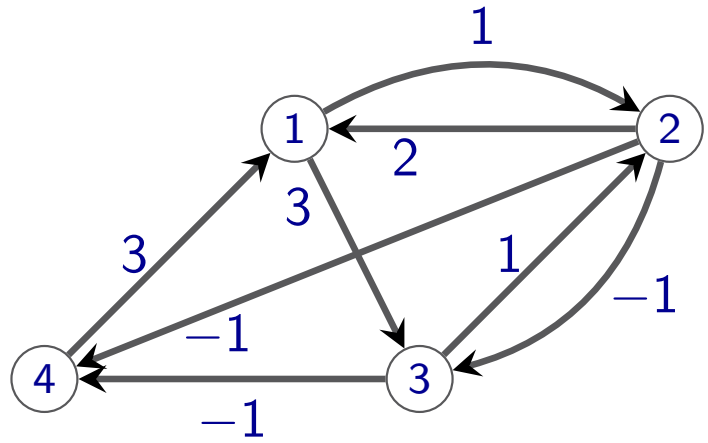
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} & 1 & & \\ 2 & 0 & -1 & -1 \\ & 1 & & \\ & 4 & & \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

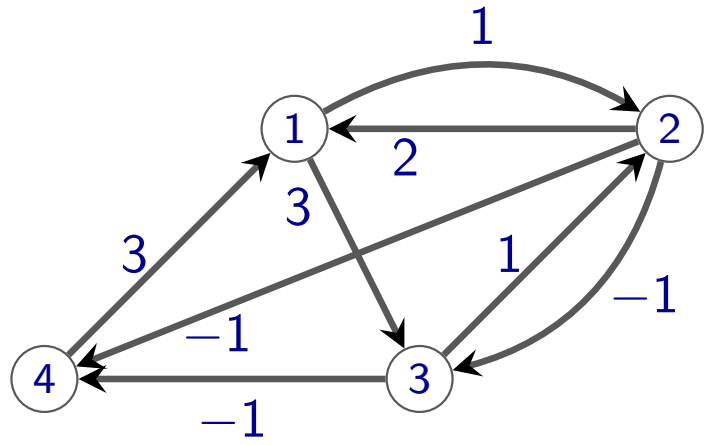
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & & \\ 2 & 0 & -1 & -1 \\ & 1 & & \\ & 4 & & \end{bmatrix}$$

$$0 = \min\{0, 1 + 2\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

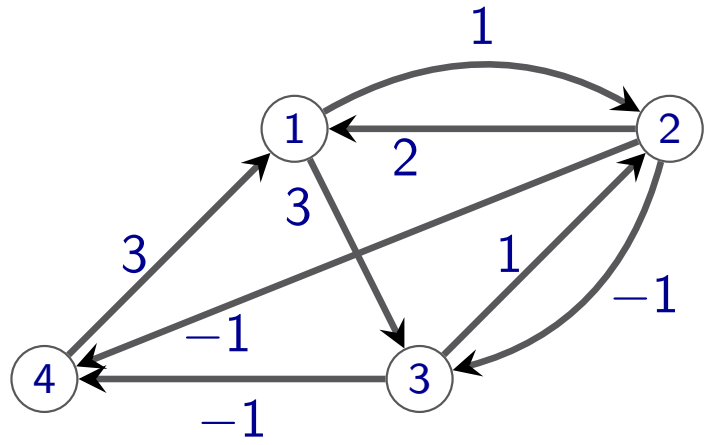
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & \\ 2 & 0 & -1 & -1 \\ & 1 & & \\ & 4 & & \end{bmatrix}$$

$$0 = \min\{3, 1 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

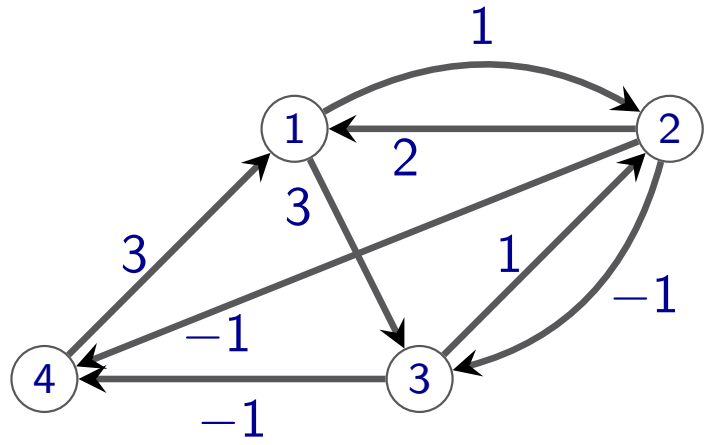
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ & 1 & & \\ & 4 & & \end{bmatrix}$$

$$0 = \min\{\infty, 1 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

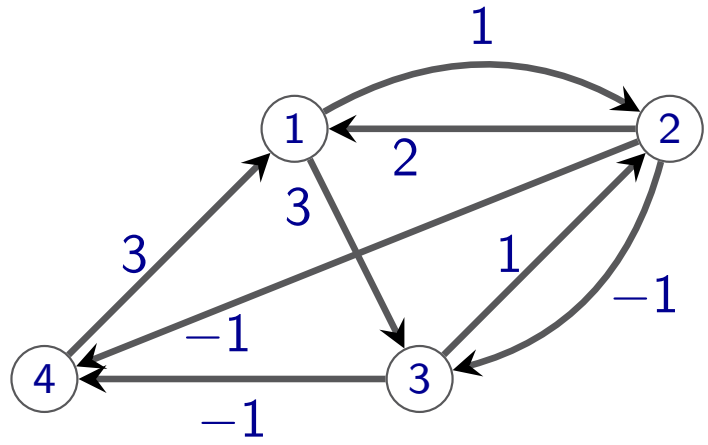
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & & \\ & 4 & & \end{bmatrix}$$

$$3 = \min\{\infty, 1 + 2\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

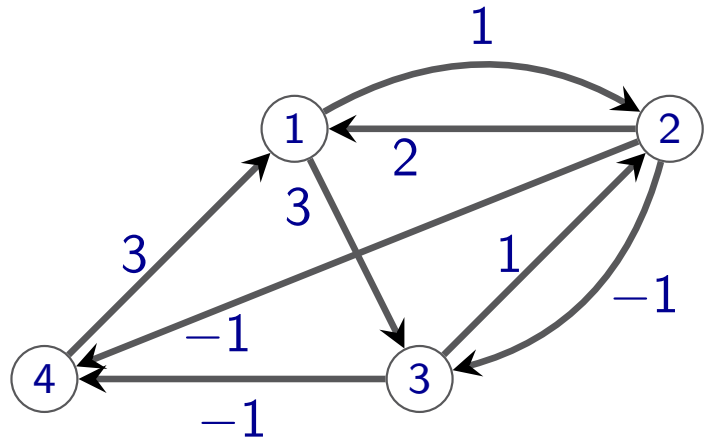
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & \\ & 4 & & \end{bmatrix}$$

$$0 = \min\{0, 1 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

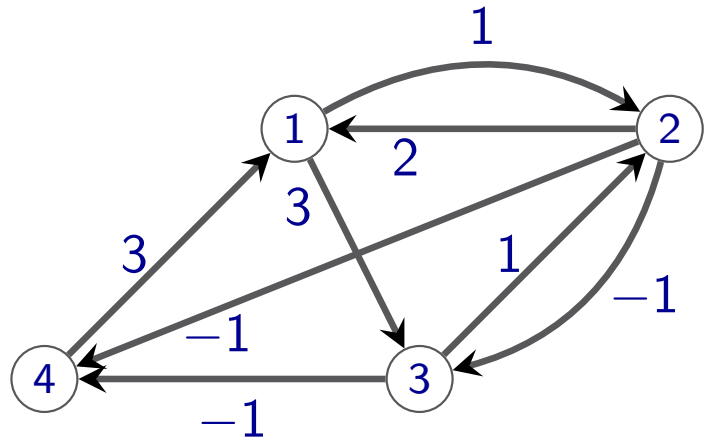
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ & 4 & & \end{bmatrix}$$

$$-1 = \min\{-1, 1 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

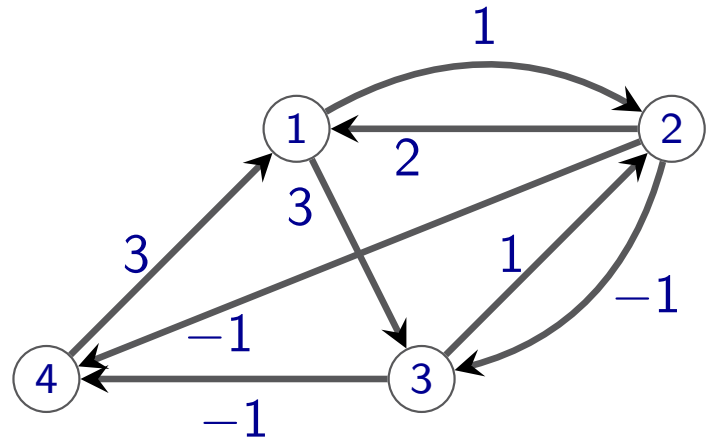
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & & \end{bmatrix}$$

$$3 = \min\{3, 4 + 2\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

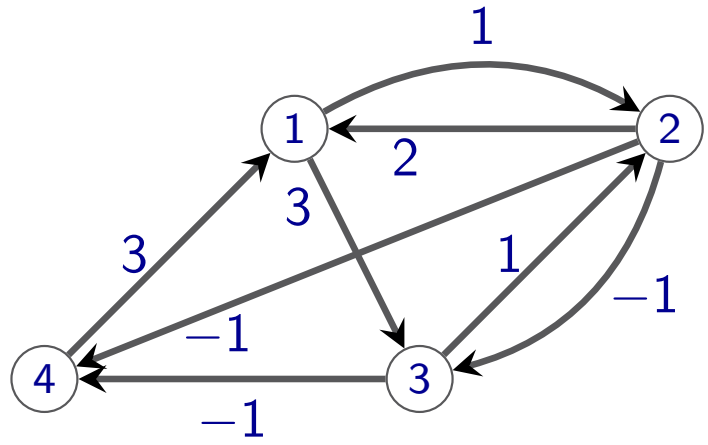
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & \end{bmatrix}$$

$$3 = \min\{6, 4 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

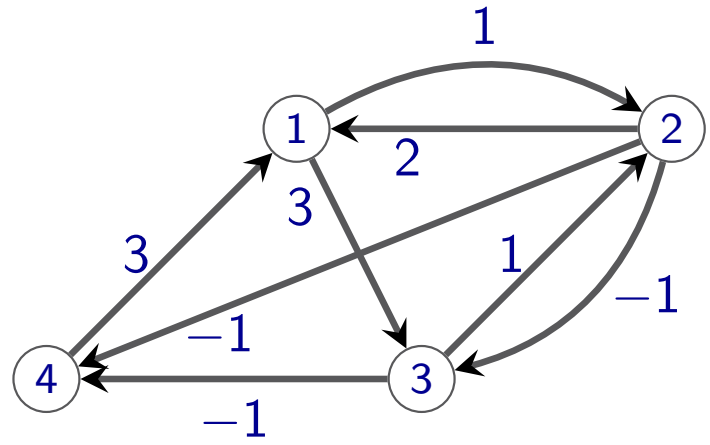
$$D_1 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & 4 & 6 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$0 = \min\{0, 4 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



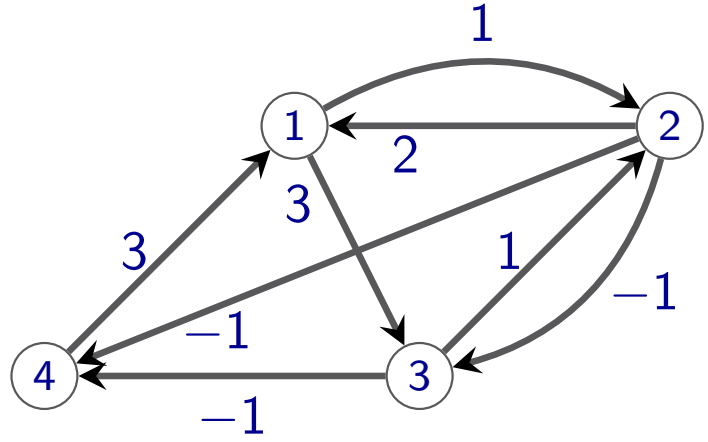
Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

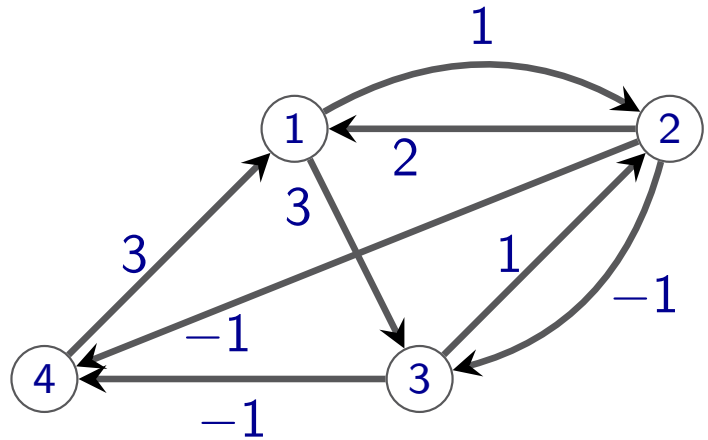
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

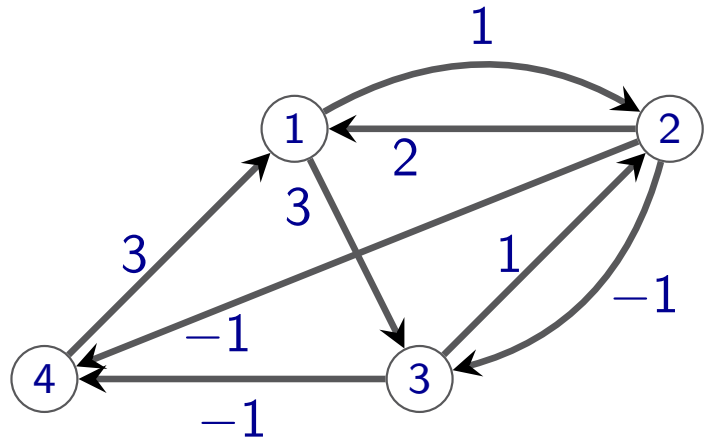
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} & & 0 & \\ & & -1 & \\ 3 & 1 & 0 & -1 \\ & & 3 & \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

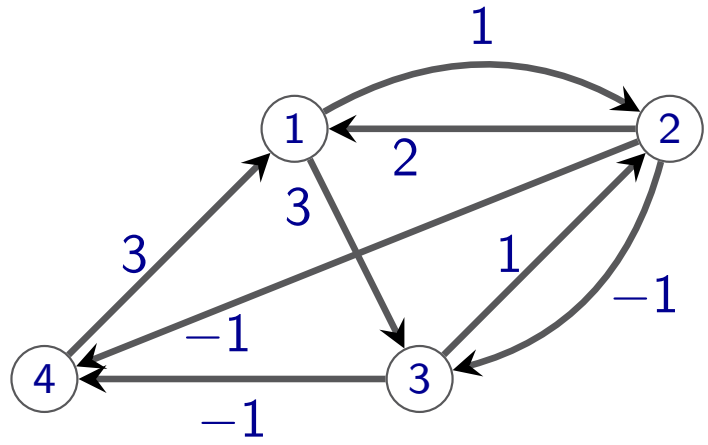
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & & 0 & \\ & & -1 & \\ 3 & 1 & 0 & -1 \\ & & 3 & \end{bmatrix}$$

$$0 = \min\{0, 0 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

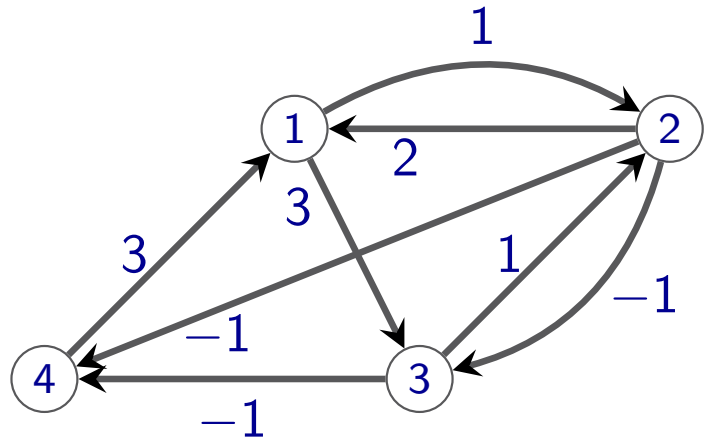
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & \\ & & -1 & \\ 3 & 1 & 0 & -1 \\ & & 3 & \end{bmatrix}$$

$$1 = \min\{1, 0 + 1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

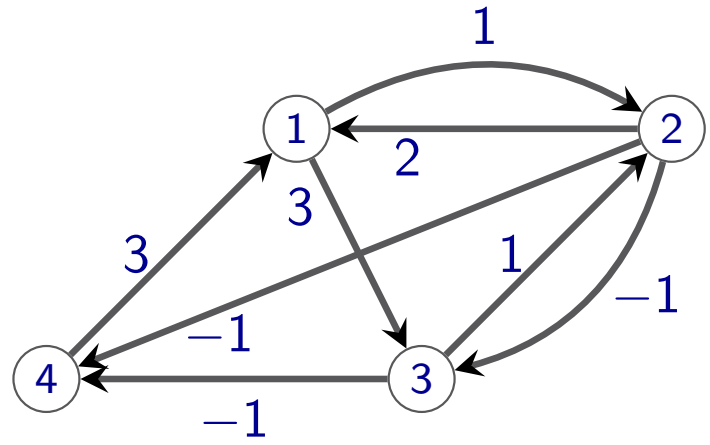
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ & & -1 & \\ 3 & 1 & 0 & -1 \\ & & 3 & \end{bmatrix}$$

$$-1 = \min\{0, 0 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

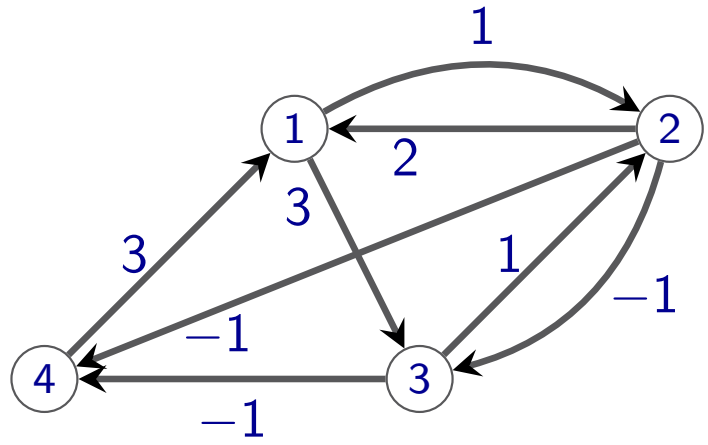
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & & -1 & \\ 3 & 1 & 0 & -1 \\ & & 3 & \end{bmatrix}$$

$$2 = \min\{2, -1 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

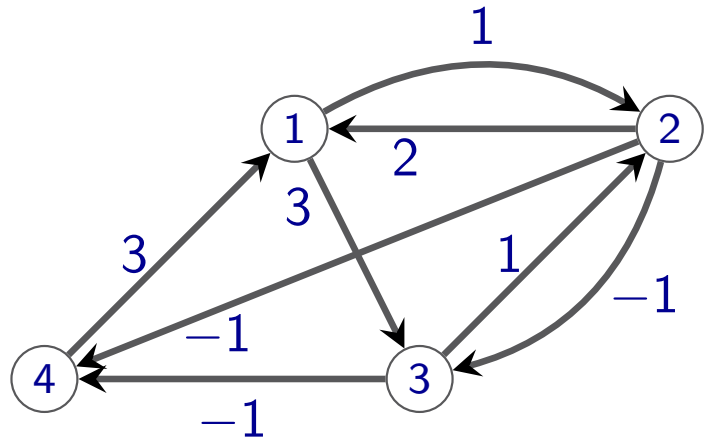
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & \\ 3 & 1 & 0 & -1 \\ & & 3 & \end{bmatrix}$$

$$0 = \min\{0, -1 + 1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

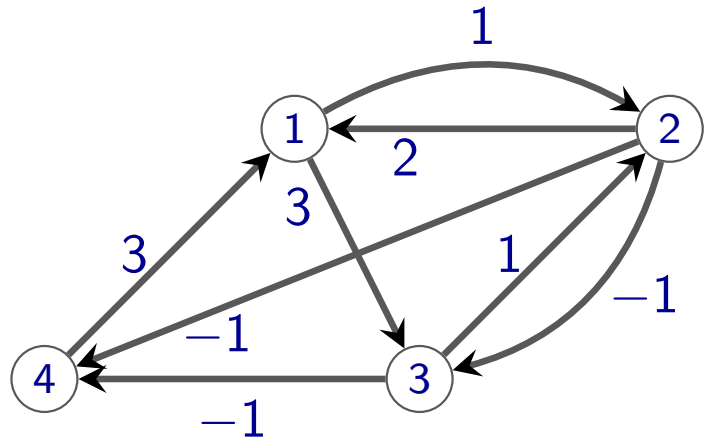
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ & & 3 & \end{bmatrix}$$

$$-2 = \min\{-1, -1 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

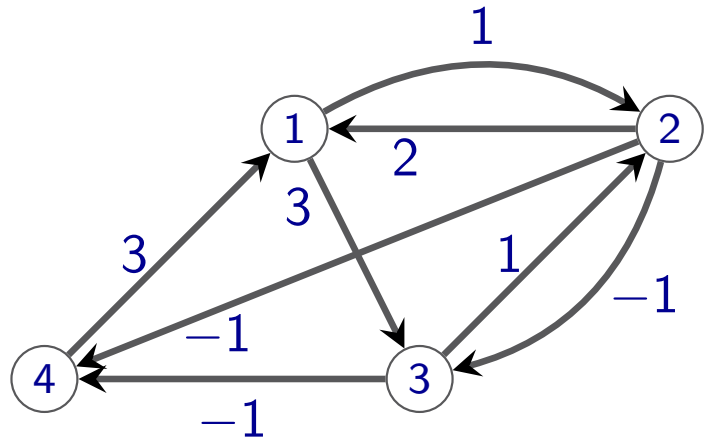
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & & 3 & \end{bmatrix}$$

$$3 = \min\{3, 3 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

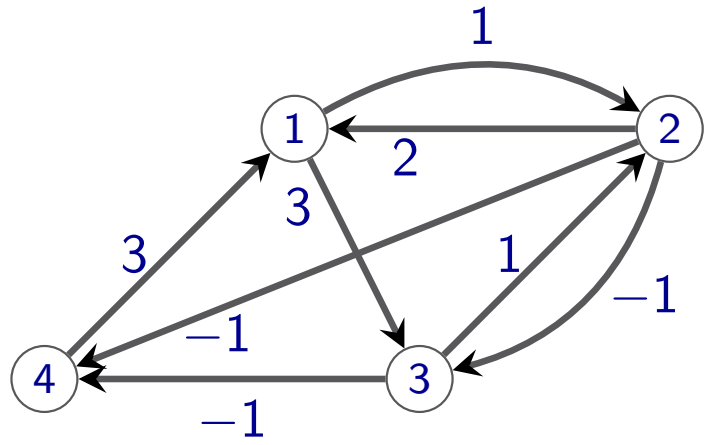
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & \end{bmatrix}$$

$$4 = \min\{4, 3 + 1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

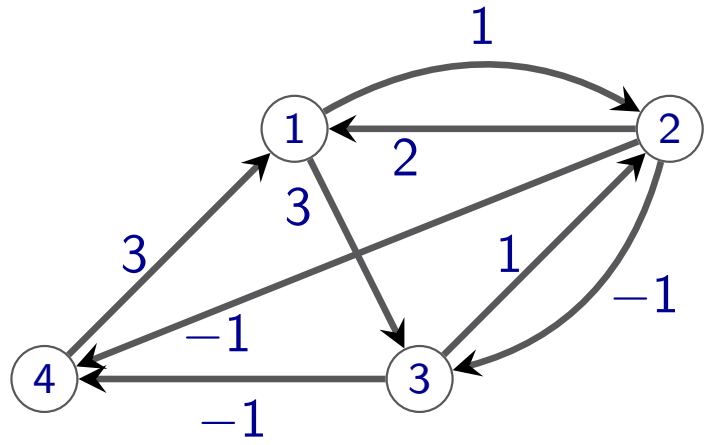
$$D_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & -1 & -1 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$0 = \min\{0, 3 + -1\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



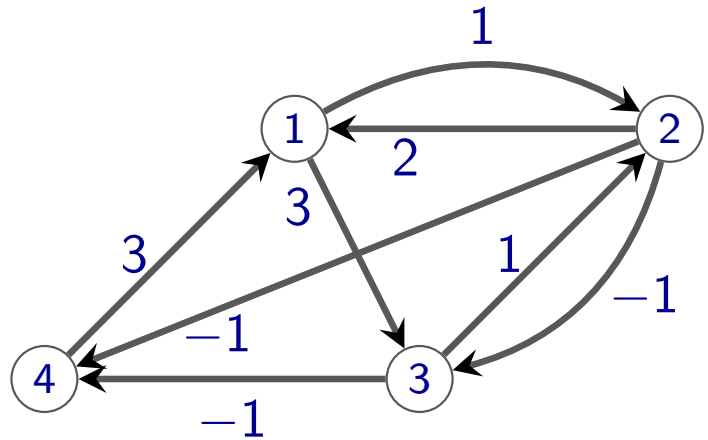
Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

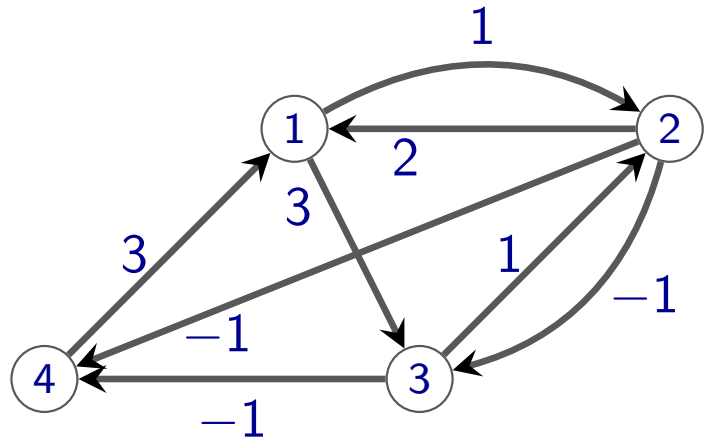
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

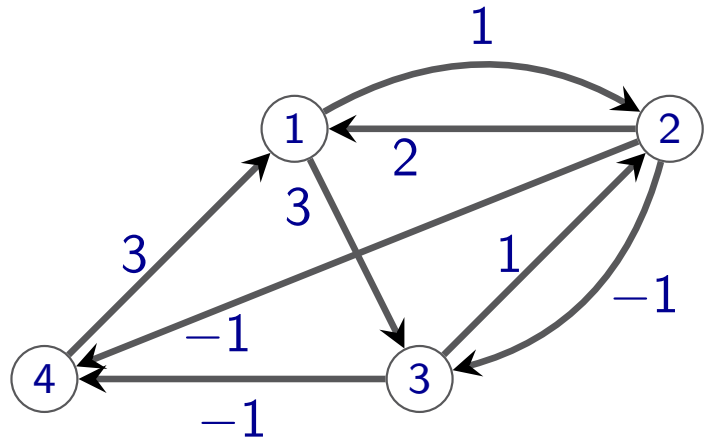
$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} & & & -1 \\ & & & -2 \\ & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

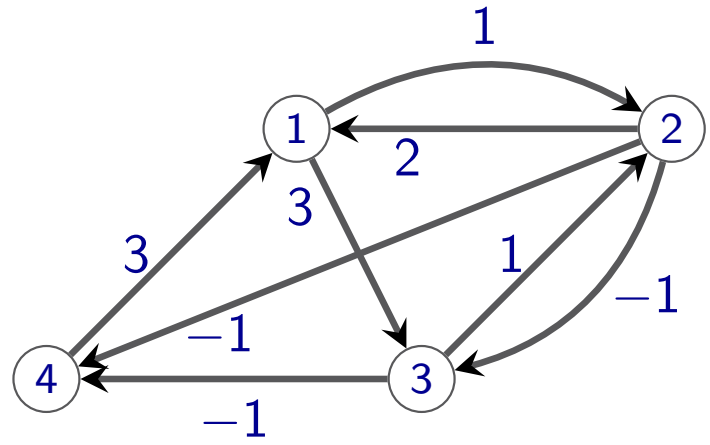
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & & & -1 \\ & & & -2 \\ & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$0 = \min\{0, -1 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

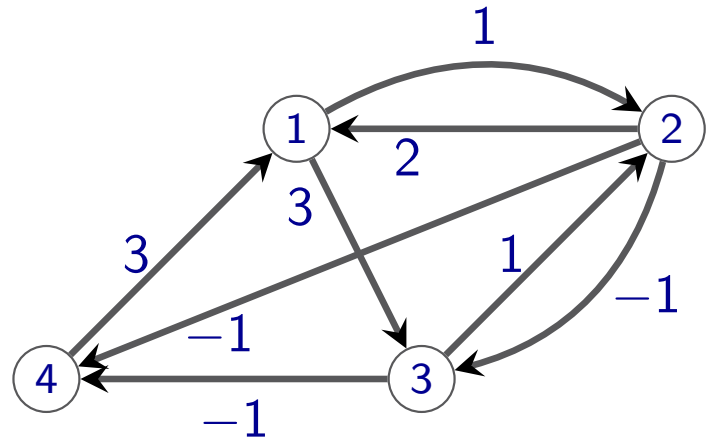
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & & -1 \\ & & & -2 \\ & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$1 = \min\{1, -1 + 4\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

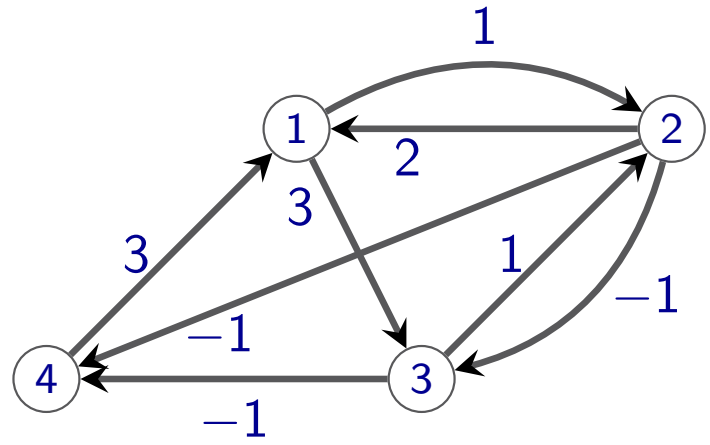
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ & & & -2 \\ & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$0 = \min\{0, -1 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

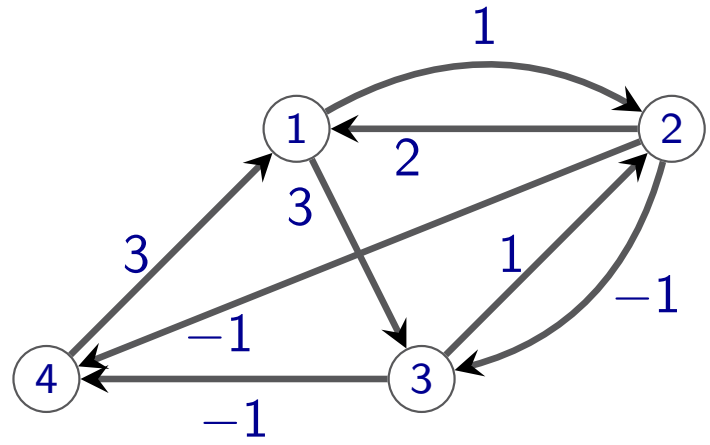
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & & & -2 \\ & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$1 = \min\{2, -2 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

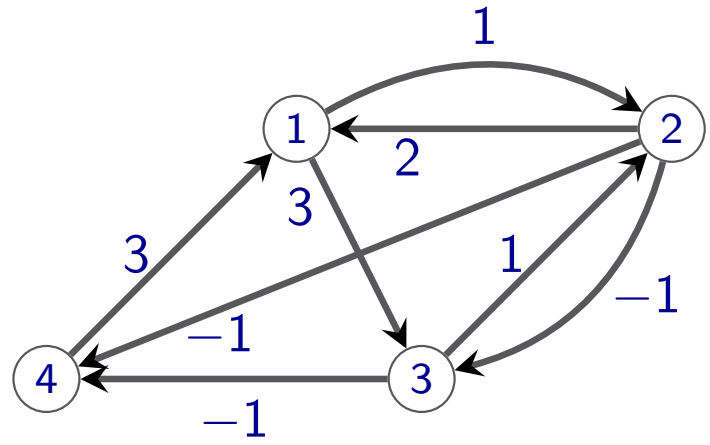
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & & -2 \\ & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$0 = \min\{0, -2 + 4\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

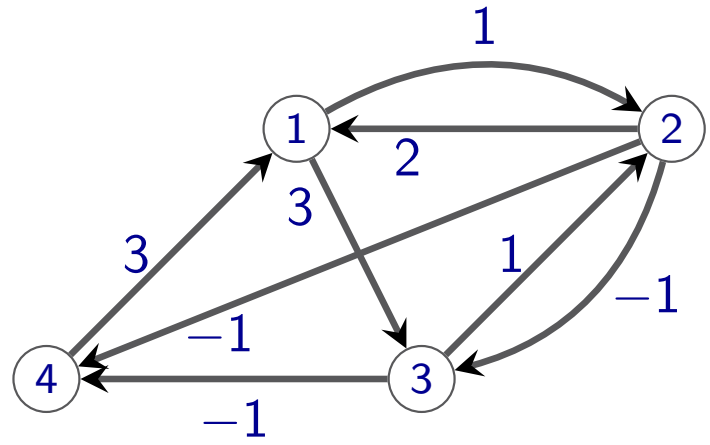
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & -2 \\ & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$-1 = \min\{-1, -2 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

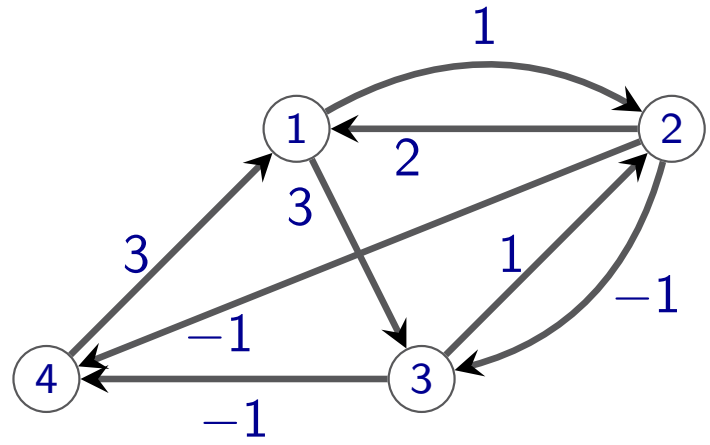
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & -2 \\ 2 & & & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$2 = \min\{3, -1 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

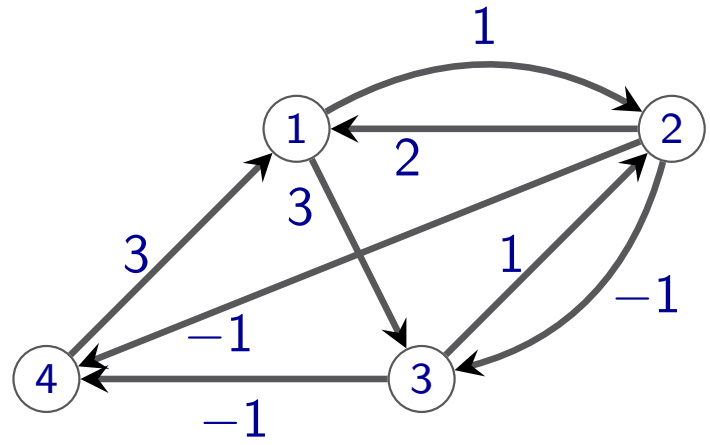
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$1 = \min\{1, -1 + 4\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall: Beispiel

$$D_0 = \begin{bmatrix} 0 & 1 & 3 & \infty \\ 2 & 0 & -1 & -1 \\ \infty & 1 & 0 & -1 \\ 3 & \infty & \infty & 0 \end{bmatrix}$$

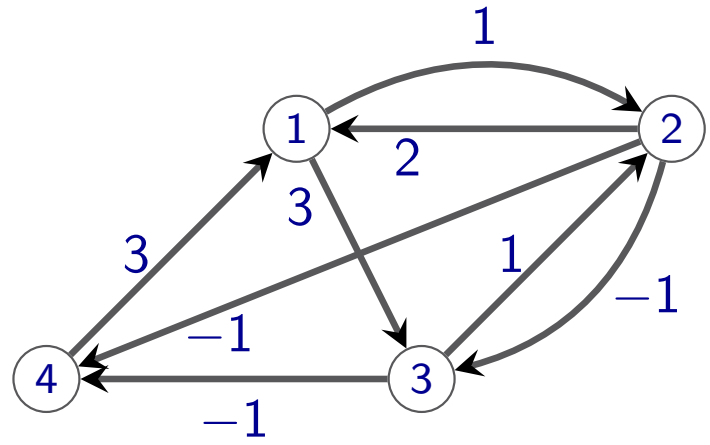
$$D_3 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 2 & 0 & -1 & -2 \\ 3 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

$$0 = \min\{0, -1 + 3\}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$

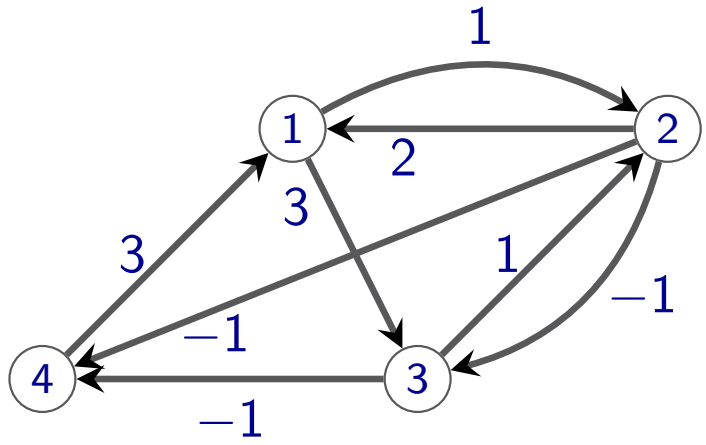


Algorithmus von Floyd und Warshall: Beispiel

$$D_4 = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \\ 3 & 4 & 3 & 0 \end{bmatrix}$$

Floyd-Warshall-Algorithmus

Init: $D_0[i,j] \leftarrow c(v_i, v_j), (v_i, v_j) \in E$
sonst $D_0[i,j] \leftarrow \infty$
For $k = 1, \ldots, n$ **do**
  For $i = 1, \ldots, n$ **do**
    For $j = 1, \ldots, n$ **do**
      $D_k[i,j] \leftarrow \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
Return $D_n[]$



Algorithmus von Floyd und Warshall

Satz

Wenn kein negativer Kreis existiert, dann berechnet der Algorithmus von Floyd und Warshall die kürzesten Wege-Distanzen zwischen allen Paaren von Knoten in Laufzeit $\mathcal{O}(n^3)$. Falls ein negativer Kreis existiert, so terminiert der Algorithmus mit $D_n[v, v] < 0$ für ein $v \in V$.

Korrektheit: Beweis per Induktion unter Ausnutzung der Bellman Gleichungen (Lemma über Teilwege).

Laufzeit: drei geschachtelte For-Schleifen.

Zusammenfassung

Designprinzip Dynamische Programmierung (DP)

- Lösung durch (rekursive) Aufteilung in kleinere Teilprobleme.
- Speichere Teillösungen um mehrfache Berechung zu vermeiden.
- Herzstück eines DP: DP-Tabelle.
- Berechnung der DP-Tabelle via Tabularization oder Memoization.
- Lösungen via Backtracking aus DP-Tabelle rekonstruieren.

Dynamische Programme für:

- Fibonacci-Folge,
- gewichtetes Interval Scheduling,
- Rucksackproblem,
- Levenshtein Distanz,
- All-Pairs Shortest Paths.


Gemütliche Feiertage
und alles Gute fürs
neue Jahr