DP 3:

Daniel Rutschmann

September 30, 2017

Swiss Olympiad in Informatics

All-pairs shortest path

Gewichteter Graph V, E mit $|V| = N$

Gewichte ensprechen Längen ≥ 0.

Was ist der kürzeste Weg von u nach v? (Für alle $u, v \in V$)

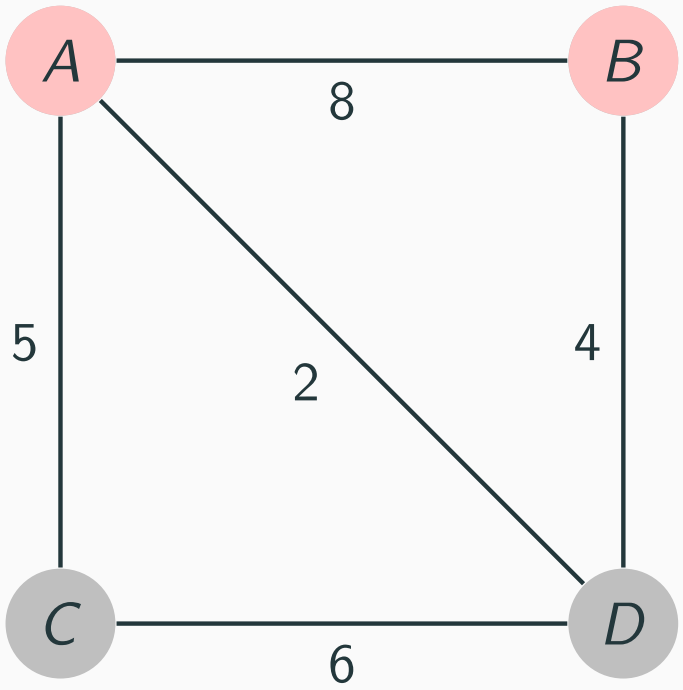
All-pairs shortest path

Adjazenzmatrix

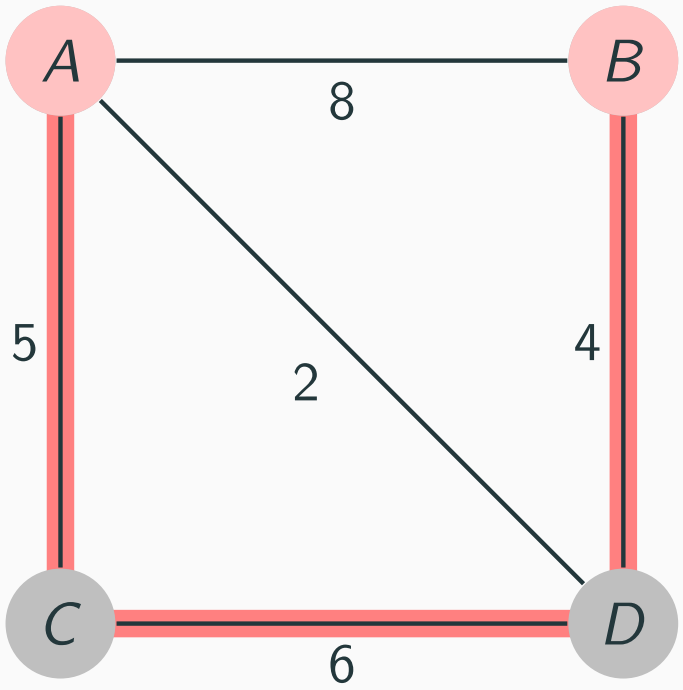
$G[i][j]$: Länge der Kante von i nach j.
$G[i][j] = \infty$: keine Kante von i nach j.
$G[i][i] = 0$

Die Kantengewichte sollten nicht negativ sein, da Zyklen negativer Länge problematisch sind.

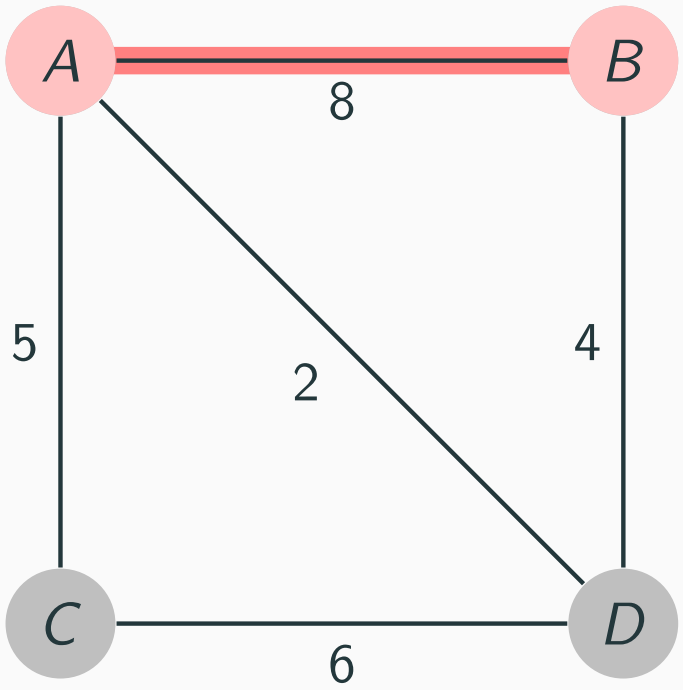
All-pairs shortest path – Beispiel



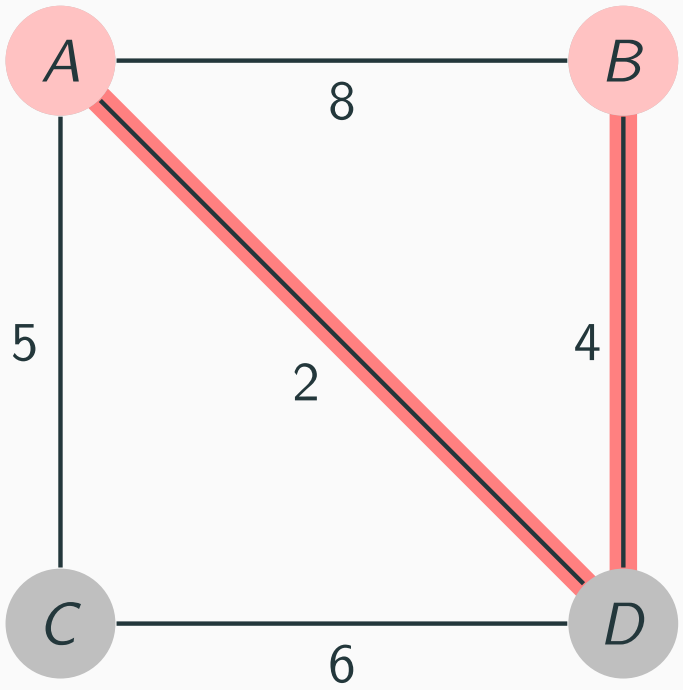
All-pairs shortest path – Beispiel



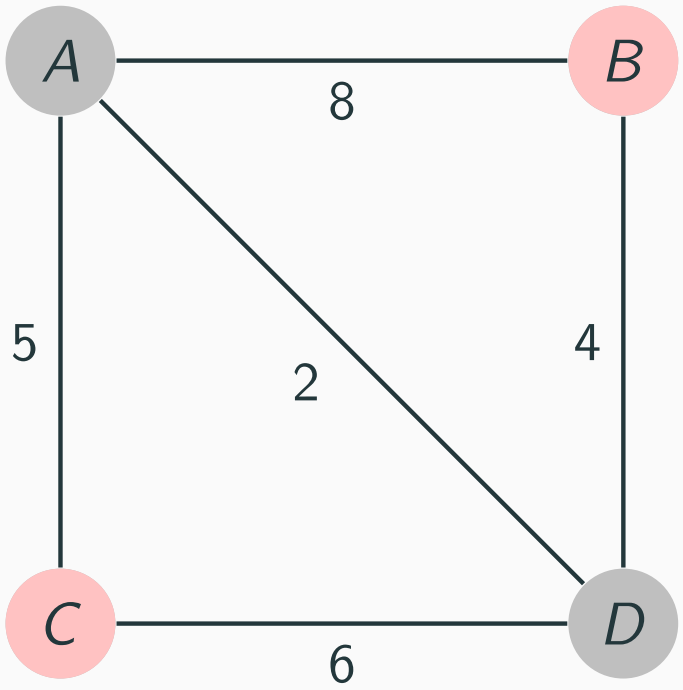
All-pairs shortest path – Beispiel



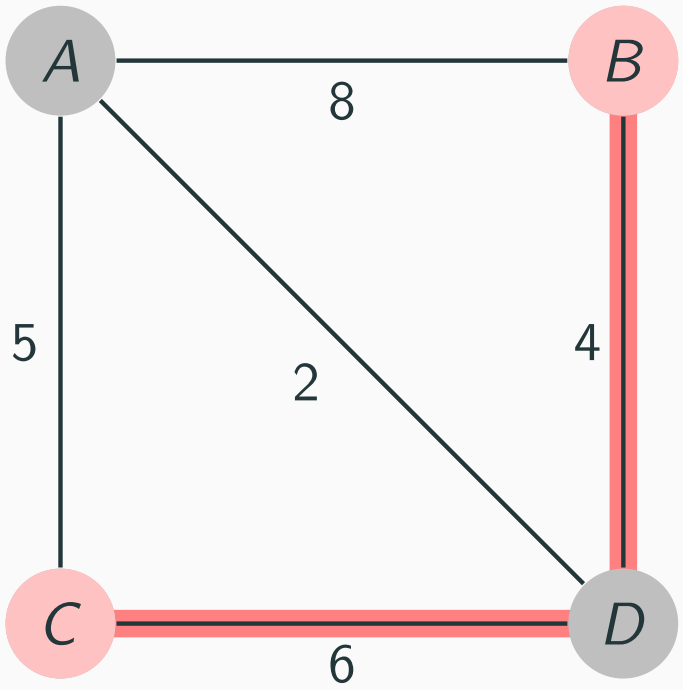
All-pairs shortest path – Beispiel



All-pairs shortest path – Beispiel



All-pairs shortest path – Beispiel



Wichtige Eigenschaften von kürzesten Pfaden

Ein kürzerster Pfad enthält keine Zyklen.

Wichtige Eigenschaften von kürzesten Pfaden

Ein kürzerster Pfad enthält keine Zyklen.

Jeder Teilpfad einen kürzesten Pfades ist ein kürzester Pfad.

Einfacher DP Ansatz

$\mathrm{DP}[i][j]$: Länge des kürzesten Pfades von i nach j.

Optimale Teilstruktur

Falls k auf dem kürzesten Weg liegt:

$$\mathrm{DP}[i][j] = \mathrm{DP}[i][k] + \mathrm{DP}[k][i]$$

Einfacher DP Ansatz

$\mathrm{DP}[i][j]$: Länge des kürzesten Pfades von i nach j.

Optimale Teilstruktur

Falls k auf dem kürzesten Weg liegt:

$$\mathrm{DP}[i][j] = \mathrm{DP}[i][k] + \mathrm{DP}[k][i]$$

Berechnung?

$$\mathrm{DP}[i][j] = \min\left(G[i][j], \min_{k \in V}\left(\mathrm{DP}[i][k] + \mathrm{DP}[k][j]\right)\right)$$

Einfacher DP Ansatz – Schwierigkeiten

Berechnung?

$$\mathrm{DP}[i][j] = \min\left(G[i][j], \min_{k \in V}\left(\mathrm{DP}[i][k] + \mathrm{DP}[k][j]\right)\right)$$

Einfacher DP Ansatz – Schwierigkeiten

Berechnung?

$$\mathrm{DP}[i][j] = \min\left(G[i][j], \min_{k \in V}\left(\mathrm{DP}[i][k] + \mathrm{DP}[k][j]\right)\right)$$

Zyklische Abhängigkeiten

$$\mathrm{DP}[1][2] \Leftarrow \mathrm{DP}[1][3] \Leftarrow \mathrm{DP}[1][2] \Leftarrow \ldots$$

DP state zu klein!

Reparierter DP Ansatz

$\mathrm{DP}[l][i][j]$: Länge der kürzesten Pfades von i nach j mit höchstens l Kanten.

Reparierter DP Ansatz

$\mathrm{DP}[l][i][j]$: Länge der kürzesten Pfades von i nach j mit höchstens l Kanten.

Optimale Teilstruktur

Die ersten $l-1$ Kanten eines kürzesten Pfades bilden einen kürzesten Pfad.

$$\mathrm{DP}[l][i][j] = \min_{k \in V} (\mathrm{DP}[l-1][i][k] + G[k][j])$$

$$\mathrm{DP}[1][i][j] = G[i][j]$$

$\Rightarrow$ keine zyklischen Abhängigkeiten mehr.

Reparierter DP Ansatz – Laufzeit und Speicherverbrauch

$\mathrm{DP}[l][i][j]$: Länge des kürzesten Pfades von i nach j mit höchstens l Kanten.

$$\mathrm{DP}[l][i][j] = \min_{k \in V} \left(\mathrm{DP}[l-1][i][k] + G[k][j]\right)$$

Reparierter DP Ansatz – Laufzeit und Speicherverbrauch

$\mathrm{DP}[l][i][j]$: Länge des kürzesten Pfades von i nach j mit höchstens l Kanten.

$$\mathrm{DP}[l][i][j] = \min_{k \in V} (\mathrm{DP}[l-1][i][k] + G[k][j])$$

$\Theta(N^3)$ States, jeder benötig $\Theta(N)$ Zeit.
$\Theta(N^4)$ Laufzeit und $\Theta(N^3)$ Speicher.

Reparierter DP Ansatz – top-down Implementation

```
from functools import lru_cache
@lru_cache(maxsize=None)
def DP(l, i, j):
  if l == 1: return G[i][j]
  return min(DP(l-1, i, k) + G[k][j] for k in
↪  range(n))
```

Verbesserter DP Ansatz

$\mathrm{DP}[k][i][j]$: Länge des kürzesten Pfades von i nach j durch Knoten mit Index $< k$.

Verbesserter DP Ansatz

$\mathrm{DP}[k][i][j]$: Länge des kürzesten Pfades von i nach j durch Knoten mit Index $< k$.

Optimale Teilstruktur

Der kürzeste Pfad kann durch den Knoten k gehen oder k meiden.

$$\mathrm{DP}[k+1][i][j] = \min\left(\mathrm{DP}[k][i][k] + \mathrm{DP}[k][k][j], \mathrm{DP}[k][i][j]\right)$$
$$\mathrm{DP}[0][i][j] = G[i][j]$$

Verbesserter DP Ansatz – top-down Implementation

```
1  @lru_cache(maxsize=None)
2  def DP(k, i, j):
3    if k==0: return G[i][j]
4    return min(DP(k-1,i,k-1)+DP(k-1,k-1,j),
 ↪   DP(k-1,i,j))
```

Verbesserter DP Ansatz – top-down Implementation

```
1  @lru_cache(maxsize=None)
2  def DP(k, i, j):
3    if k==0: return G[i][j]
4    return min(DP(k-1,i,k-1)+DP(k-1,k-1,j),
 ↪   DP(k-1,i,j))
```

$\Theta(N^3)$ States, jeder benötigt $\Theta(1)$ Zeit.

$\Theta(N^3)$ Laufzeit und Speicher.

Verbesserter DP Ansatz – bottom up

```
inf = 10**18
DP = [[[inf for _ in range(n)] for _ in range(n)]
 ↪  for _ in range(n+1)]
DP[0] = G
for k in range(n):
  for i in range(n):
    for j in range(n):
      DP[k+1][i][j] = min(DP[k][i][k]+DP[k][k][j],
 ↪  DP[k][i][j])
```

$\Theta(N^3)$ States, jeder benötigt $\Theta(1)$ Zeit.

$\Theta(N^3)$ Laufzeit und Speicher.

Verbesserter DP Ansatz – Weniger Speicher

```
DP = G
for k in range(n): # Schleife mit k ganz aussen!
  for i in range(n):
    for j in range(n):
      DP[i][j] = min(DP[i][j], DP[i][k] + DP[k][j])
```

$\Theta(N^3)$ Laufzeit, $\Theta(N^2)$ Speicher.

Anwendungen

Alles Mögliche, was auf kürzesten Distanzen aufbaut.

- Durchmesser eines Graphen.

Anwendungen

Alles Mögliche, was auf kürzesten Distanzen aufbaut.

- Durchmesser eines Graphen.

Der Algorithmus funktoniert auch auf gerichteten Graphen und auch mit negativen Gewichten.