# DP 3: Kürzeste Wege

Daniel Rutschmann

September 3, 2019

Swiss Olympiad in Informatics

# All-pairs shortest path

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

Gewichte ensprechen Längen $\geq 0$.

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

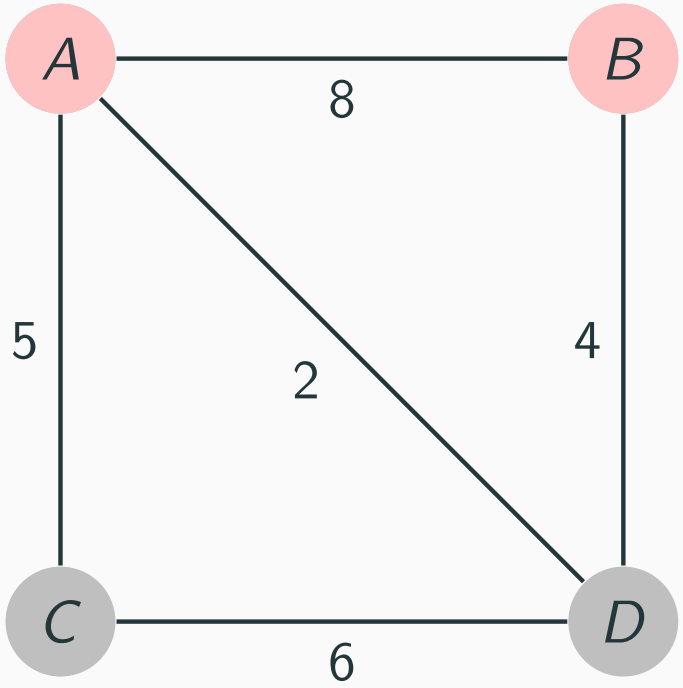
# All-pairs shortest path

**Adjazenzmatrix**

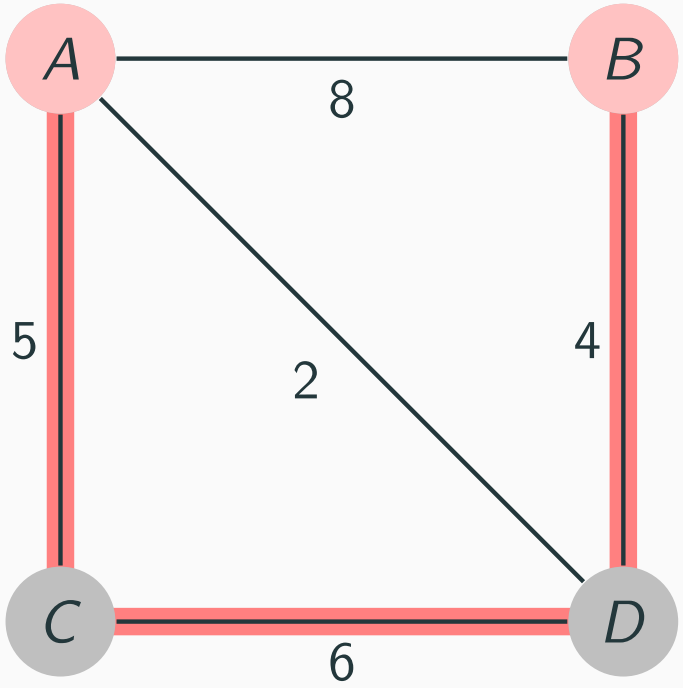
$G[i][j]$ : Länge der Kante von $i$ nach $j$.
$G[i][j] = \infty$ : keine Kante von $i$ nach $j$.
$G[i][i] = 0$

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

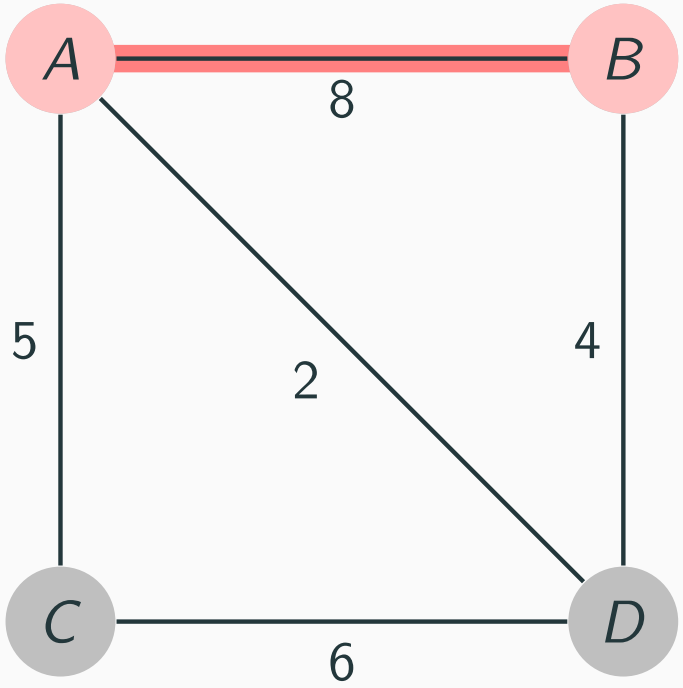
# All-pairs shortest path – Beispiel



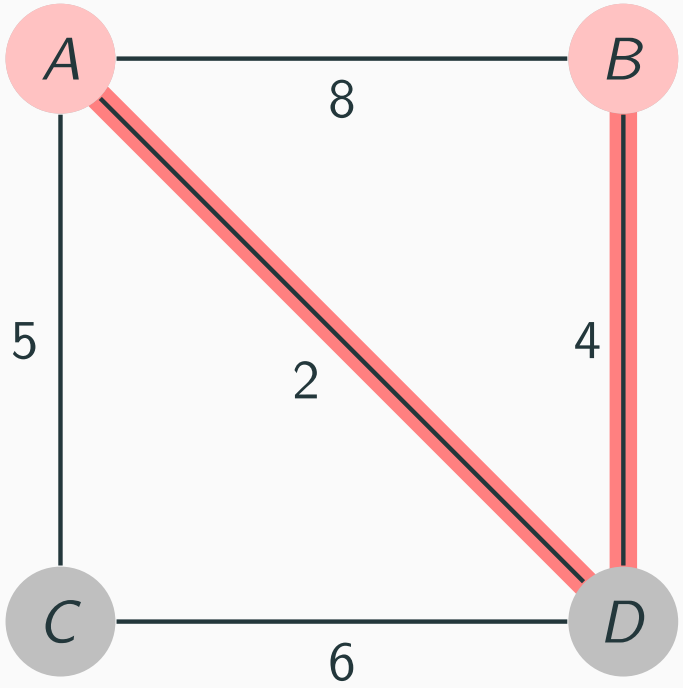
# All-pairs shortest path – Beispiel



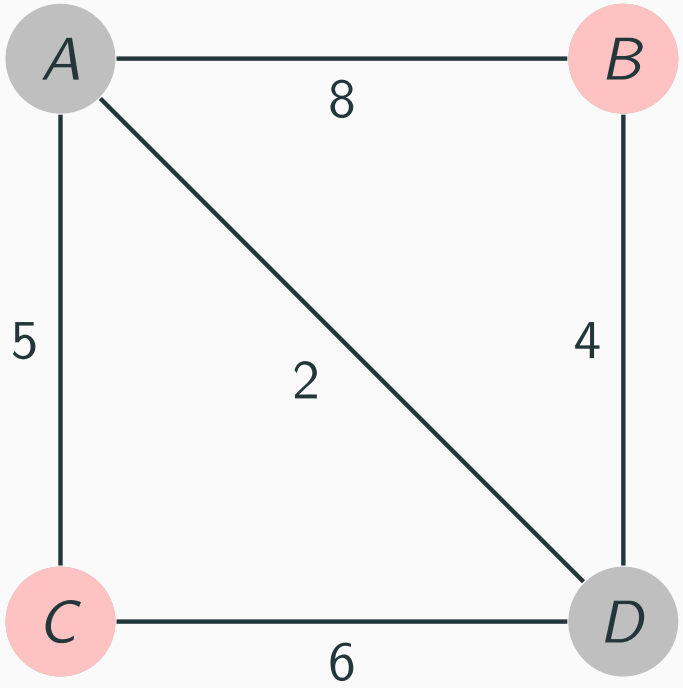
# All-pairs shortest path – Beispiel



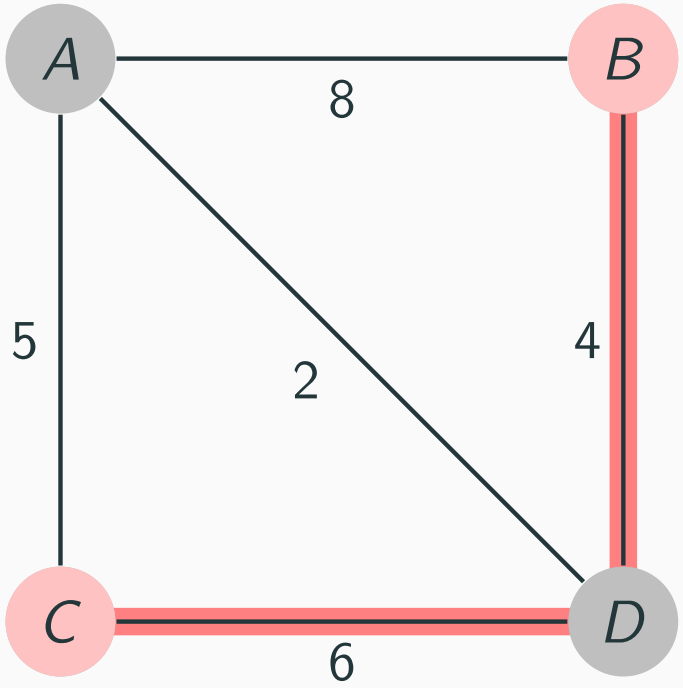
# All-pairs shortest path – Beispiel



# All-pairs shortest path – Beispiel



# All-pairs shortest path – Beispiel



# Wichtige Eigenschaften von kürzesten Wegen

Ein kürzerster Weg enthält keine Kreise.

# Wichtige Eigenschaften von kürzesten Wegen

Ein kürzerster Weg enthält keine Kreise.

Jeder Teilweg eines kürzesten Weges ist ein kürzester Weg.

# Einfacher DP Ansatz

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

# Reparierter DP Ansatz

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

**Optimale Teilstruktur**

Die ersten $l-1$ Kanten eines kürzesten Weges bilden einen kürzesten Weg.

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

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

$\Rightarrow$ keine zyklischen Abhängigkeiten mehr.

# Reparierter DP Ansatz – Laufzeit und Speicherverbrauch

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

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

# Reparierter DP Ansatz – Laufzeit und Speicherverbrauch

$\mathrm{DP}[l][i][j]$ : Länge eines kürzesten Weges 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 benötigt $\Theta(N)$ Zeit.
$\Theta(N^4)$ Laufzeit und $\Theta(N^3)$ Speicher.

# Reparierter DP Ansatz – top-down Implementation

```
1  vector<vector<vector<int> > > cache (n+1,
   ↪  vector<vector<int> > (n, vector<int>(n, -1)));
2  const int inf = 1e9;
3  int dp(int l, int i, int j){
4      if (l == 1) return G[i][j];
5      int &c = cache[i][j];
6      if(c != -1) return c;
7      c = inf;
8      for(int k = 0; k < n; ++k){
9          c = min(c, dp(l-1, i, k) + G[k][j]);
10     }
11     return c;
12 }
```

# Verbesserter DP Ansatz

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

# Verbesserter DP Ansatz

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

**Optimale Teilstruktur**

Der kürzeste Weg 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  vector<vector<vector<int> > > cache(n+1,
    ↪  vector<vector<int> > (n, vector<int>(n, -1)));
2  int dp(int k, int i, int j){
3      if(k == 0) return G[i][j];
4      if(cache[i][j] != -1) return cache[i][j];
5      cache[i][j] = min(DP(k-1, i, k-1) + DP(k-1, k-1,
    ↪  j), DP(k-1, i, j));
6      return cache[i][j];
7  }
```

# Verbesserter DP Ansatz – top-down Implementation

```
1  vector<vector<vector<int> > > cache(n+1,
    ↪  vector<vector<int> > (n, vector<int>(n, -1)));
2  int dp(int k, int i, int j){
3      if(k == 0) return G[i][j];
4      if(cache[i][j] != -1) return cache[i][j];
5      cache[i][j] = min(DP(k-1, i, k-1) + DP(k-1, k-1,
    ↪  j), DP(k-1, i, j));
6      return cache[i][j];
7  }
```

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

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

# Verbesserter DP Ansatz – bottom up

```
1  vector<vector<vector<int> > > DP(n+1,
   ↪  vector<vector<int> >(n, vector<int>(n, inf)));
2  for(int i=0;i<n;++i)
3      for(int j=0;j<n;++j)
4          DP[0][i][j] = G[i][j];
5  for(int k = 0; k < n; ++k){
6      for(int i = 0; i < n; ++i){
7          for(int j = 0; j < n; ++j){
8              DP[k+1][i][j] = min(DP[k][i][k] +
   ↪  DP[k][k][j], DP[k][i][j]);
9          }
10     }
11 }
```

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

# Verbesserter DP Ansatz – Weniger Speicher

```
1  vector<vector<int> > DP = G;
2  for(int k = 0; k < n; ++k){ // k ganz aussen!
3      for(int i = 0; i < n; ++i){
4          for(int j = 0; j < n; ++j){
5              DP[i][j] = min(DP[i][k] + DP[k][j],
   ↪  DP[i][j]);
6          }
7      }
8  }
```

$\Theta(N^3)$ 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.