Sommersemester 2020

CARL VON OSSIETZKY universität OLDENBURG

Objektorientierte Modellierung und Programmierung

Dr. Christian Schönberg

CARL VON OSSIETZKY universität OLDENBURG

Lösungsstrategien I

Lernziele

- Best Practices
- Optimierungsprobleme
- Dynamische Programmierung

Best Practices

DRY (Don‘t Repeat Yourself)

- Vermeidung von Wiederholungen
  - doppelten Programmcode in eine Methode auslagern
  - doppelte Attribute oder Methoden vererben
  - doppelte Datenstrukturen generisch machen
  - Programmbibliotheken verwenden

CARL VON OSSIETZKY universität OLDENBURG

KISS (Keep It Simple, Stupid)

- Code und Architektur nicht unnötig kompliziert machen
  - entwickelt 1960 bei der US-Marine
  - ➢ Konzepte, Strukturen und Algorithmen so einfach wie möglich halten, aber so komplex wie nötig
  - ➢ keine komplexen Strukturen „auf Vorrat“ entwerfen, sondern Struktur anpassen wenn nötig (refactoring)

Separation of Concerns / Single Responsibility Principle

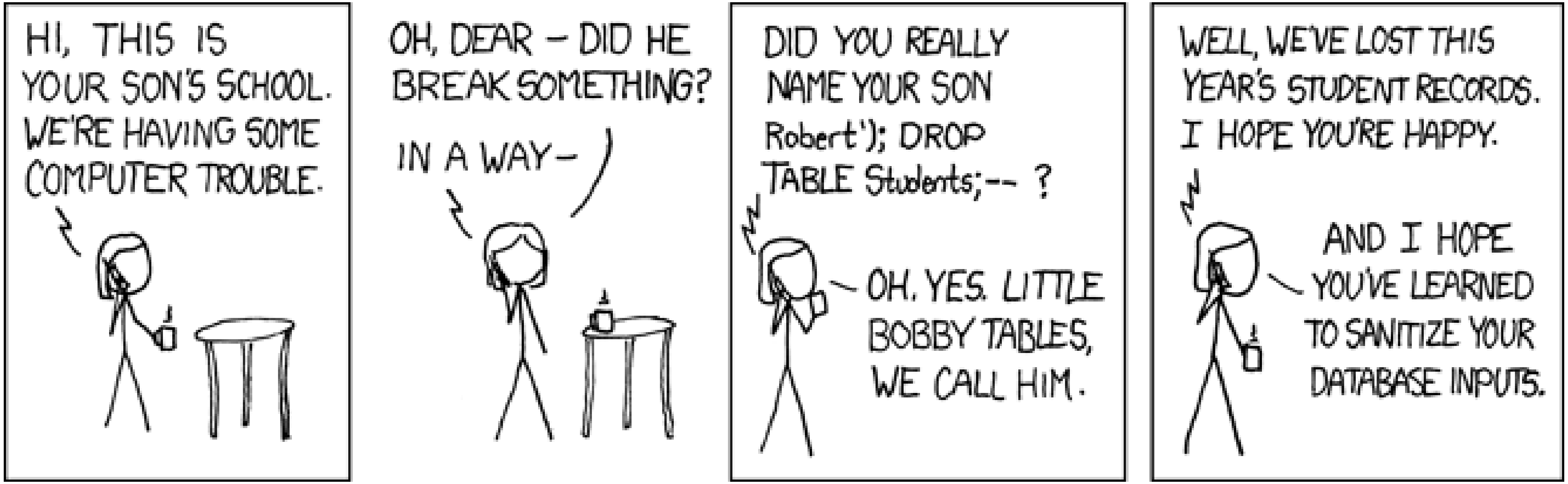
- Jede Klasse hat genau eine Aufgabe
  - ➢ nicht unterschiedliche Belange in eine einzige Klasse packen
  - Beispiel: MVC-Pattern
    - eine Klasse (Model) ist für die Datenhaltung zuständig
    - eine Klasse (View) ist für das Anzeigen der Daten zuständig
    - eine Klasse (Control) ist für die Programmlogik zuständig
  - Analog: HTML / CSS / JS

SPOT (Single Point Of Truth)

- Jedes Datum soll nur einmal gespeichert werden
  - DRY-Prinzip für Daten
  - Duplikate vermeiden, Referenzierung verwenden
  - Konstanten/enums verwenden
  - Datenbanken normalisieren

trust no1

- Parameter und Nutzereingaben immer prüfen




```sql
INSERT INTO Students (name) VALUES ('#1');
```

Robert

```sql
INSERT INTO Students (name) VALUES ('Robert');
```

Robert‘); DROP TABLE Students; --

```sql
INSERT INTO Students (name) VALUES ('Robert');
DROP TABLE Students; --');
```

Motivation

Motivation

- Wozu Lösungsstrategien?
  - viele Probleme → viele Algorithmen
  - neues Problem → neuer Algorithmus?
- Strategien
  - Gemeinsamkeiten in der Struktur des Problems
    - ➢ Gemeinsamkeiten in der Lösungsidee
    - ➢ Wiederverwendung von bewährten Methoden

Probleme

- Bisher
  - gegeben: ein Problem
  - gesucht: eine (ggf. einzig für dieses Problem erstellte) Lösung
  - Methode:
    - effiziente Umsetzung
    - maximal polynomialer Aufwand
- Beispiele
  - sortieren: Quicksort in $\mathrm{O}(\mathrm{n} \log \mathrm{n})$ oder $\mathrm{O}(\mathrm{n}^2)$
  - MST: Prim in $\mathrm{O}(|\mathrm{E}| \log |\mathrm{V}|)$
- Was ist mit Problemen, die nicht in polynomialer Zeit gelöst werden können?

Effizienz

- Was können wir tun, wenn eine bestehende Lösung zu ineffizient (d.h. zu langsam oder zu speicherplatzhungrig) für ein gegebenes Problem ist?
  - effiziente Algorithmen
  - effiziente Datenstrukturen
  - bessere Hardware
  - Parallelisierung

Lösungsstrategien

- Übertragbar
  - nicht nur für ein spezielles Problem zugeschnitten
- Effektiv
  - erzeugen eine brauchbare Lösung
- Effizient
  - erzeugen eine Lösung „schnell genug“

Strategien bekannter Algorithmen

- Gemeinsamkeiten von
  - Binäre Suche, Quicksort
    - unterteilen das Problem in kleinere Teilprobleme
    - Divide & Conquer Strategie
  - Binäre Suche, Quicksort
    - Rekursion
  - Quicksort, Tripel-Algorithmus
    - (Optimale) Lösung wird aus (optimalen) Teillösungen zusammengesetzt
    - Dynamische Programmierung
  - Algorithmus von Prim
    - Lösung wird aus ausschließlich lokal optimalen Teillösungen zusammengesetzt, einmal gewählte Teillösungen werden nicht wieder verworfen
    - Greedy-Verfahren
    - Heuristik

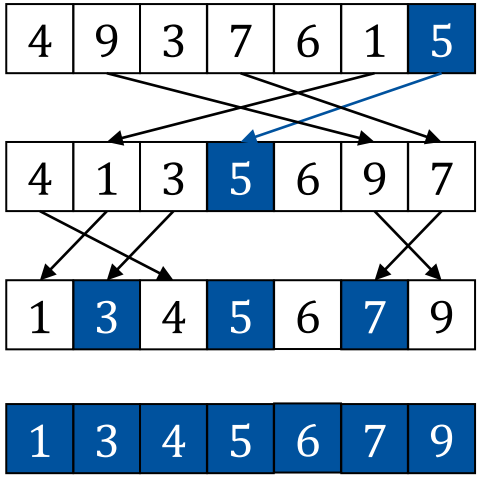
Divide & Conquer Strategien

- Divide & Conquer (teile und herrsche, divide et impera)
  - teile das Problem in kleinere, einfachere Teilprobleme
  - die Teilprobleme haben die gleiche Struktur wie das Gesamtproblem
  - setze die Lösung aus den Teillösungen zusammen
  - Anwendung
    - Algorithmen
    - Softwaretechnik (top-down Design)
    - Ingenieurwissenschaft
    - …

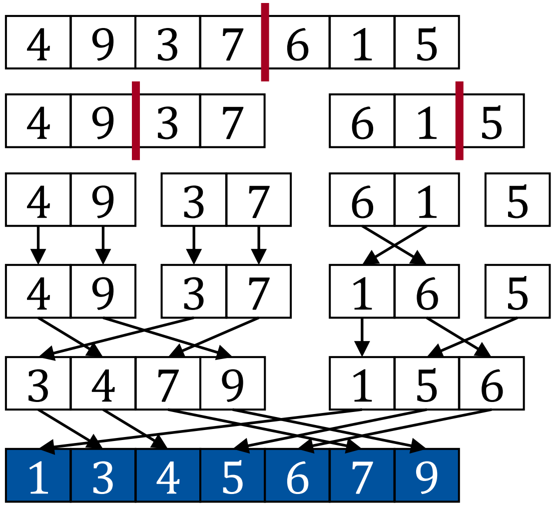
Beispiel: Divide & Conquer

- Kosten für das Aufteilen des Problems oder für das Zusammensetzen der Teillösungen?
  - Quicksort
    - hard split: Aufteilung in Teilfelder mit einem Pivot-Element
    - easy join: sortierte Teilfelder sind automatisch in der korrekten Reihenfolge
  - Mergesort
    - easy split: Feld wird in der Mitte geteilt
    - hard join: sortierte Teilfelder werden zusammengemischt

Quicksort:



Mergesort:



Rekursion

- Top-Down Strategie
- In der Definition eines Objekts (Datenstruktur, Algorithmus) wird das zu definierende Objekt verwendet
- Beispiel: Fakultätsberechnung

```
public int factorial(int number) {
    if (number < 0) {
        return -1;
    } else if (number == 0) {
        return 1;
    } else {
        return number * factorial(number - 1);
    }
}
```

Greedy-Verfahren

- Lösungsprozess: Treffen von Entscheidungen
- Eine getroffene Entscheidung wird nie revidiert
  - Beispiel: Algorithmus von Prim
- Vorteil: schnell
- Aber: Für viele Optimierungsverfahren ist kein Greedy-Verfahren bekannt, das die optimale Lösung liefert
  - Greedy-Verfahren liefern dann zwar gültige, aber nur suboptimale Lösungen

Optimierungsprobleme

Optimierungsprobleme

- Manche Probleme haben eine eindeutige Lösung
  - z.B. das Sortieren von Daten oder
  - die Suche nach einem Schlüssel
- Andere Probleme haben mehrere mögliche Lösungen
  - z.B. das Suchen nach bestimmten Begriffen oder
  - das Suchen eines Weges von A nach B
- Oft kann man diese Lösungen nach ihrer Qualität bewerten
  - z.B. die Relevanz eines Suchergebnisses oder
  - die Länge des gefundenen Weges

➢ Ziel ist es, die beste (oder zumindest eine ausreichend gute) Lösung zu finden

Optimierungsprobleme (formal)

- Optimierungsprobleme
  - gegeben: ein Problem O
  - gesucht: die beste (optimale) Lösung für O
- Formal:
  - Lösungsraum S: Menge aller Lösungen für O
  - Gütefunktion $q: S \rightarrow R^{+}$ ($q(x)$ = Qualität der Lösung x aus S)
  - Finde die Lösung $x \in S$, so dass $q(x)$ optimal (maximal oder minimal) ist
    - $\max\{ q(x) \mid x \in S \}$ oder $\min\{ q(x) \mid x \in S \}$
- Für viele Optimierungsprobleme gibt es **keine** polynomiale Lösung
- Beispiel: Route des Paketlieferanten, die an allen Adressen vorbeikommt und im Lager anfängt und aufhört

Entscheidungsfolge

- Betrachte den Lösungsweg als Folge von Entscheidungen $(d_n, d_{n-1}, \dots, d_1)$
- Gliedere das Optimierungsproblem durch die Entscheidungsfolge in $n + 1$ Zustände $(s_n, s_{n-1}, \dots, s_1, s_0)$
  - P: Menge der (Teil-)Lösungen, $\{ s_n, s_{n-1}, \dots, s_0 \} \subseteq P$
  - D: Menge zulässiger Entscheidungen, $\{ d_n, d_{n-1}, \dots, d_1 \} \subseteq D$
  - Starte mit Teillösung s_n (Startzustand)
  - Zustand s_i: noch i Entscheidungen zu treffen
  - Gewählte Entscheidung $d_i \in D$ liefert neue Teillösung $s_{i-1} = t(d_i, s_i) \in P$
    - t: Zustandsübergangsfunktion

$$\boxed{s_n} \xrightarrow{t(d_n,s_n)} \boxed{s_{n-1}} \longrightarrow \dots \xrightarrow{t(d_{i+1},s_{i+1})} \boxed{s_i} \xrightarrow{t(d_i,s_i)} \dots \xrightarrow{t(d_2,s_2)} \boxed{s_1} \xrightarrow{t(d_1,s_1)} \boxed{s_0}$$

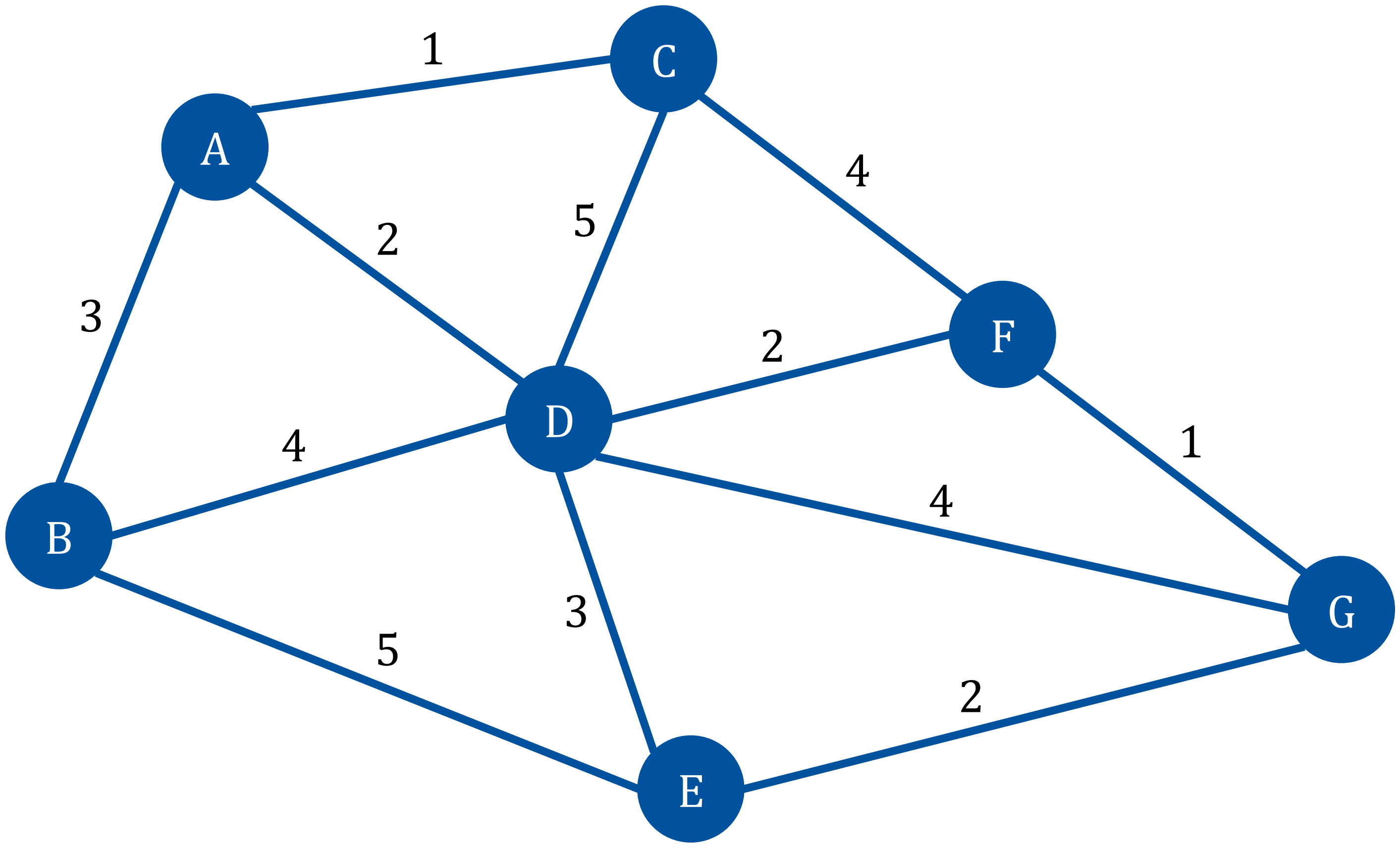
Entscheidungsfolge (2)

- Betrachte den Lösungsweg als Folge x von Entscheidungen $(d_n, d_{n-1}, \ldots, d_1)$
- Bewerte die Gesamtgüte $q(x)$ der Entscheidungsfolge $x = (d_n, \ldots, d_1)$
- $q(x) = q(d_n, s_n) + \ldots + q(d_1, s_1)$, mit $d_i \in D$, $s_i \in P$, $i = 1, \ldots, n$
- $q(d_k, s_k)$: Güte der Entscheidung d_k im Zustand s_k

- Gesucht: Entscheidungsfolge $x = (d_n, \ldots, d_1)$ mit optimaler (o.B.d.A. maximaler) Gesamtgüte $q(x)$
- Gesucht: $\max\{\, q(x) \mid x \in S \,\}$

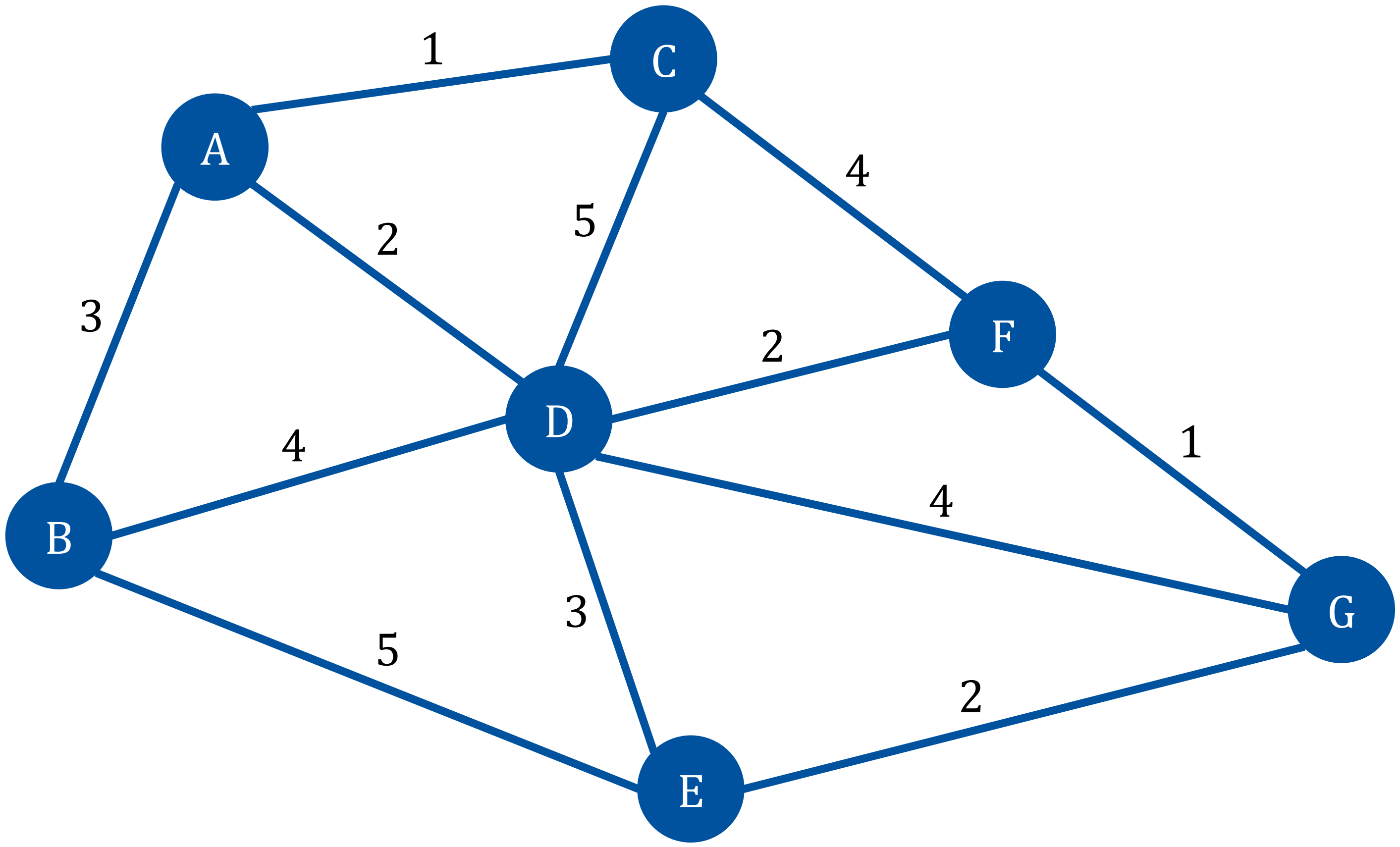
Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)



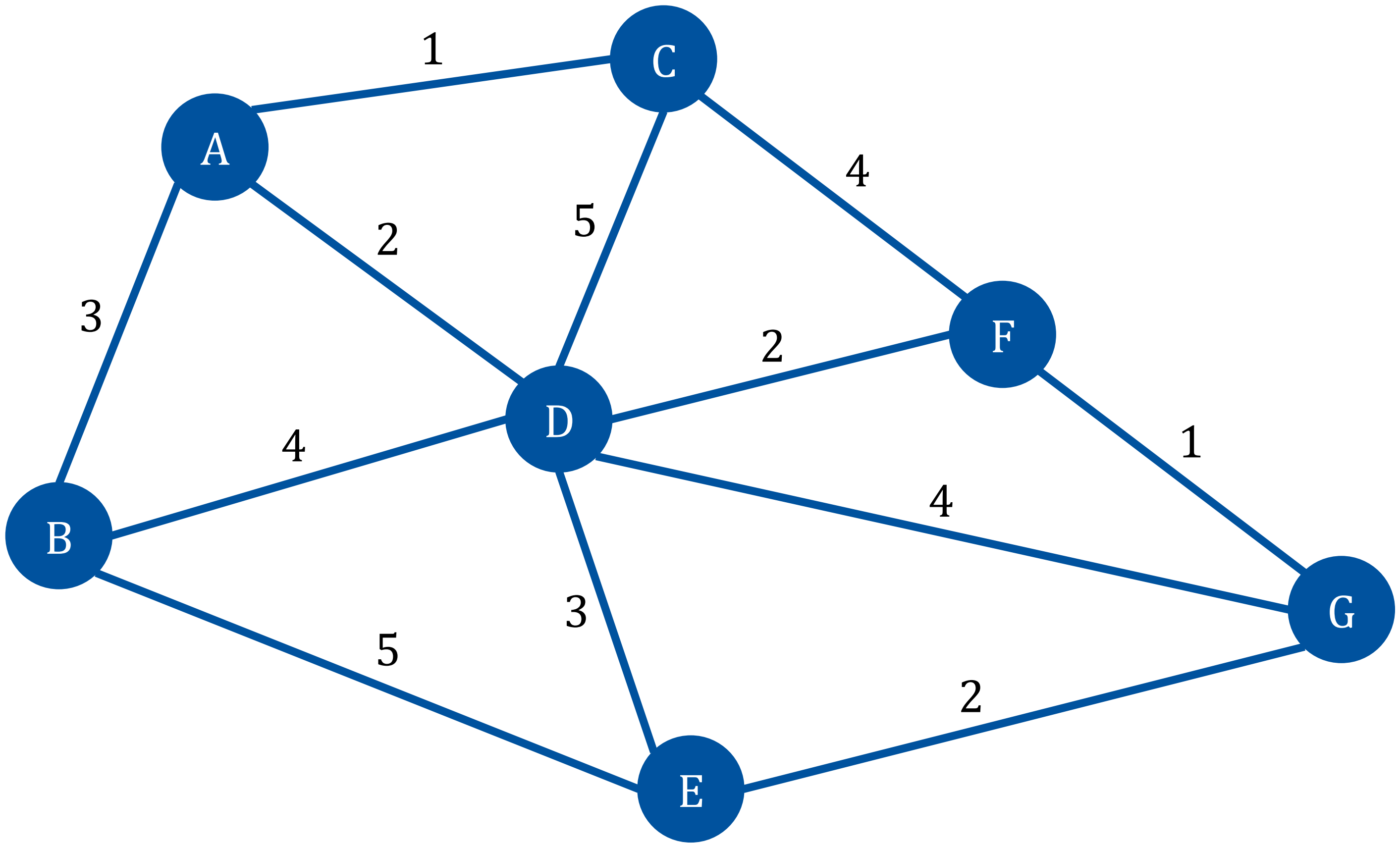
Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G



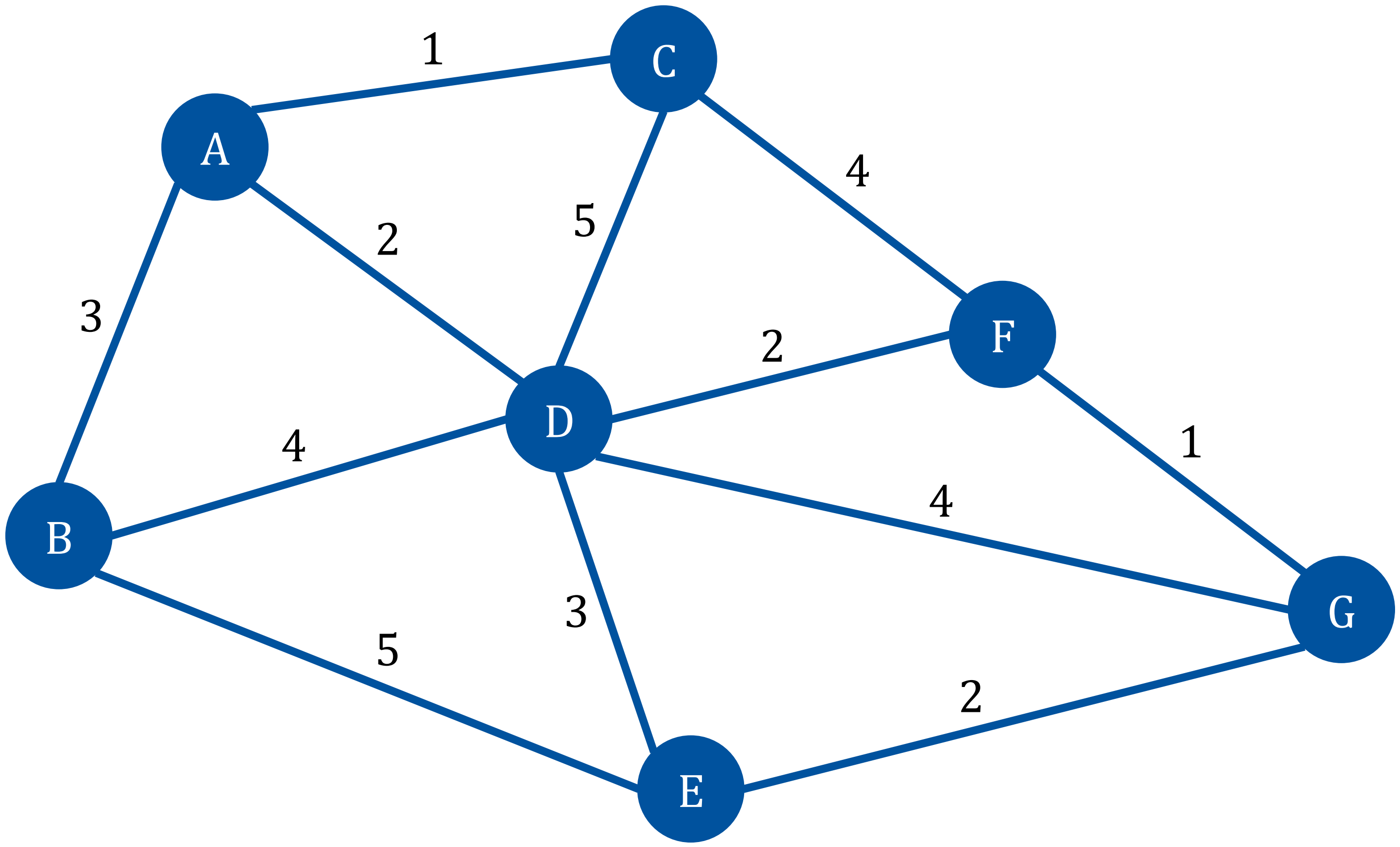
Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten



Beispiel: Entscheidungsfolge

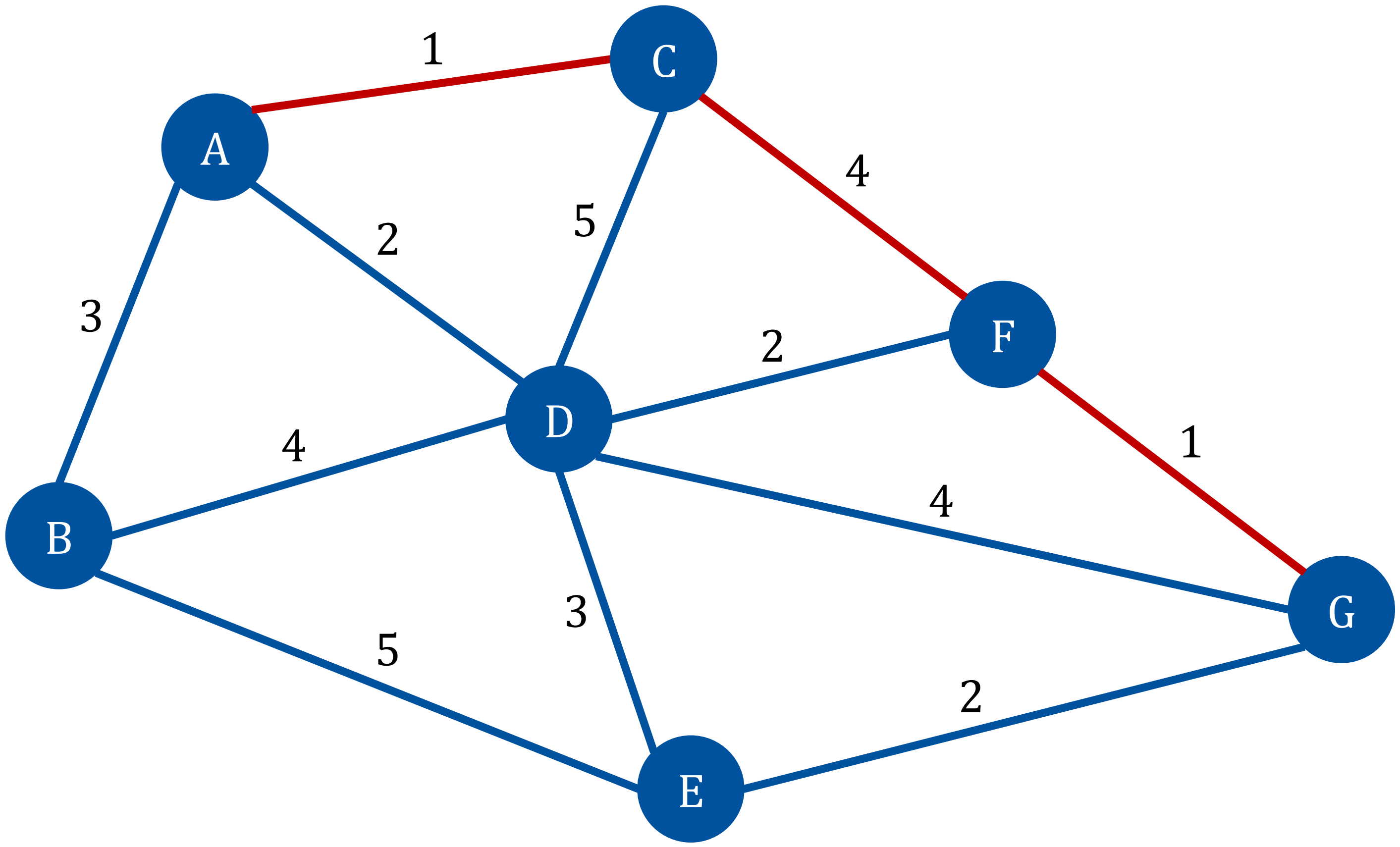
Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht



Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht

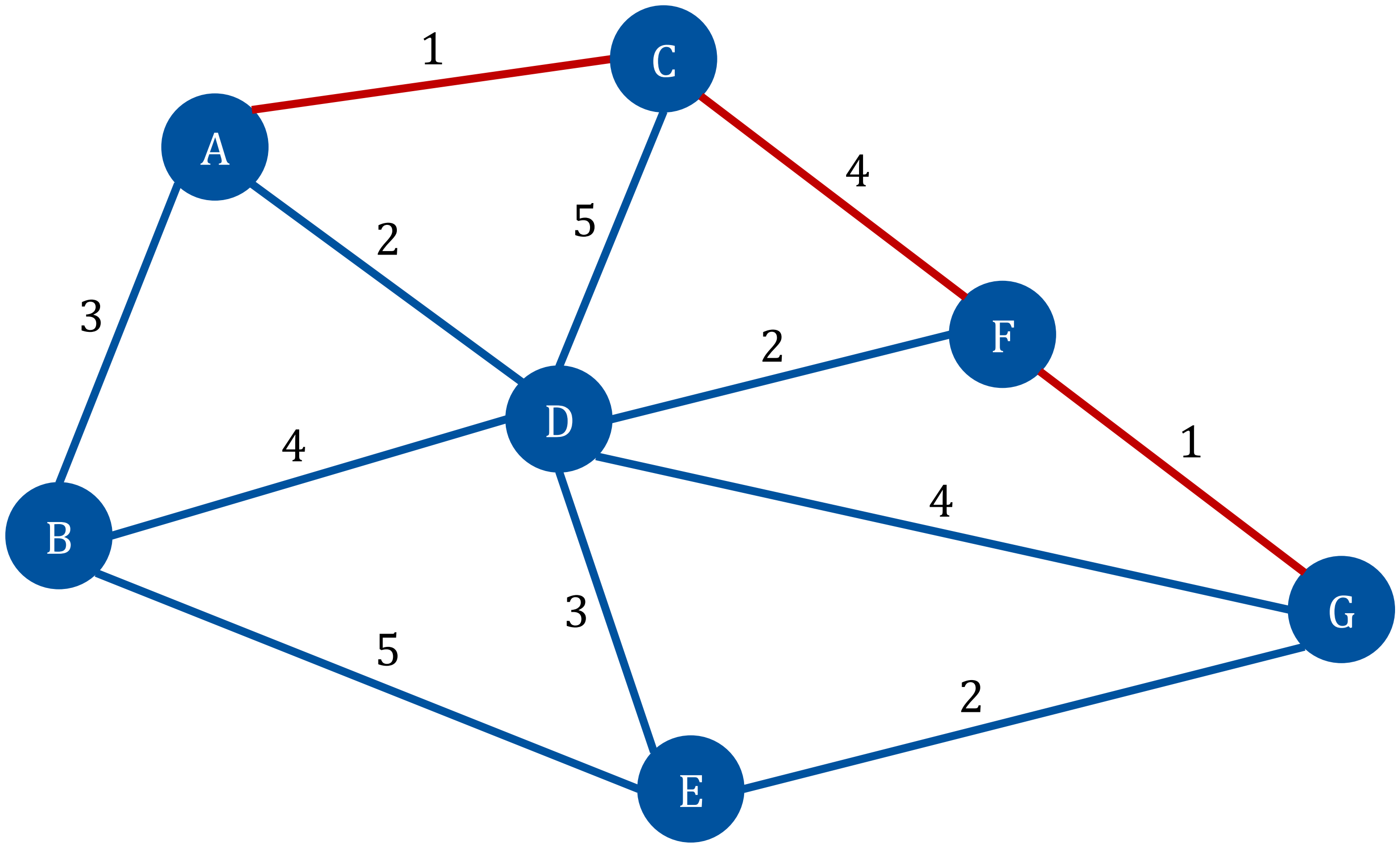
Entscheidungsfolge $x = (1, 4, 1)$



Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht

Entscheidungsfolge $x = (1, 4, 1)$
$q(x) = q(1, A) + q(4, C) + q(1, F) = 6$

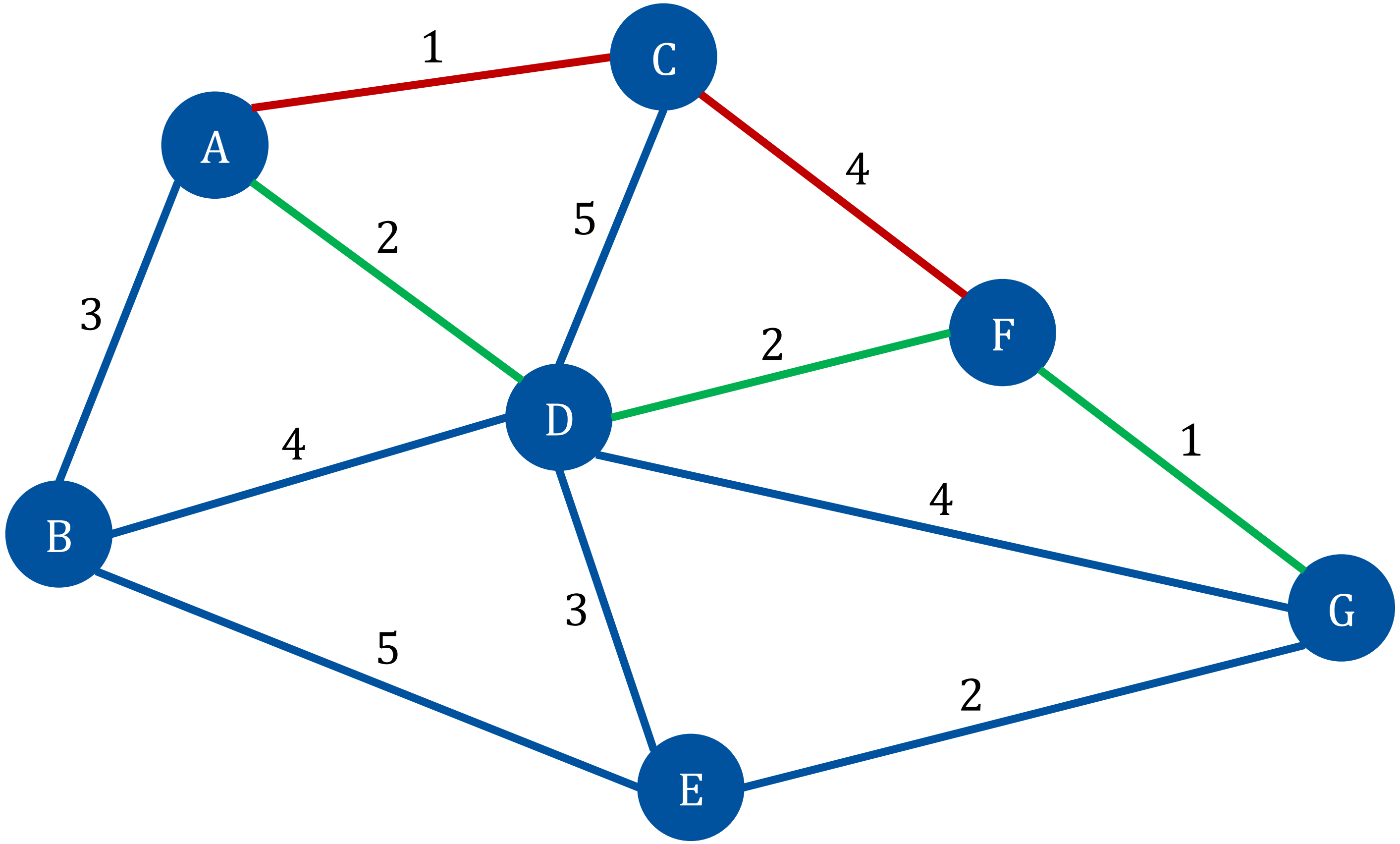


Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht

Entscheidungsfolge $x = (1, 4, 1)$
$q(x) = q(1, A) + q(4, C) + q(1, F) = 6$

Entscheidungsfolge $y = (2, 2, 1)$
$q(y) = q(2, A) + q(2, D) + q(1, F) = 5$



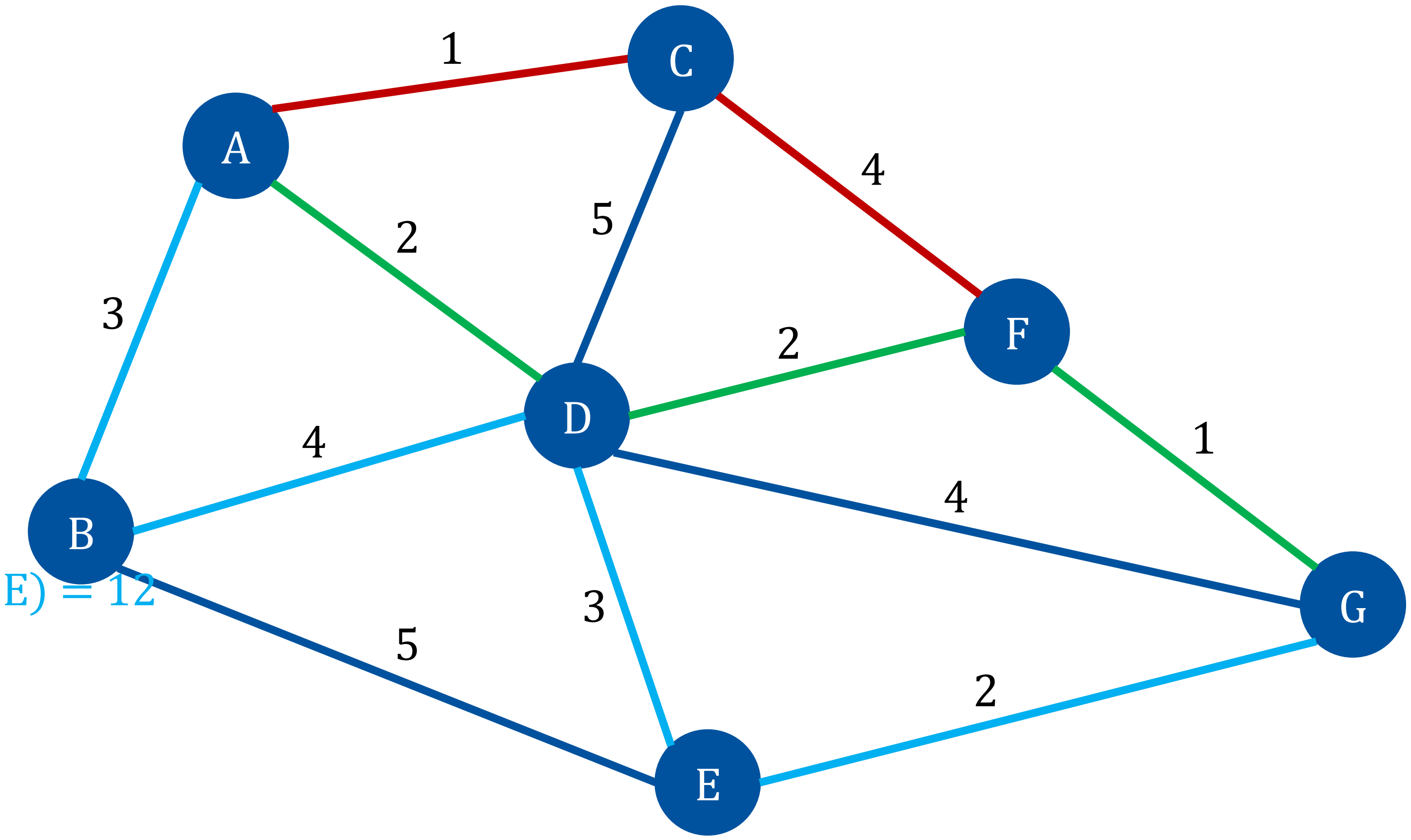
Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht

Entscheidungsfolge $x = (1, 4, 1)$
$q(x) = q(1, A) + q(4, C) + q(1, F) = 6$

Entscheidungsfolge $y = (2, 2, 1)$
$q(y) = q(2, A) + q(2, D) + q(1, F) = 5$

Entscheidungsfolge $z = (3, 4, 3, 2)$
$q(z) = q(3, A) + q(4, B) + q(3, D) + q(2, E) = 12$



Beispiel: Entscheidungsfolge

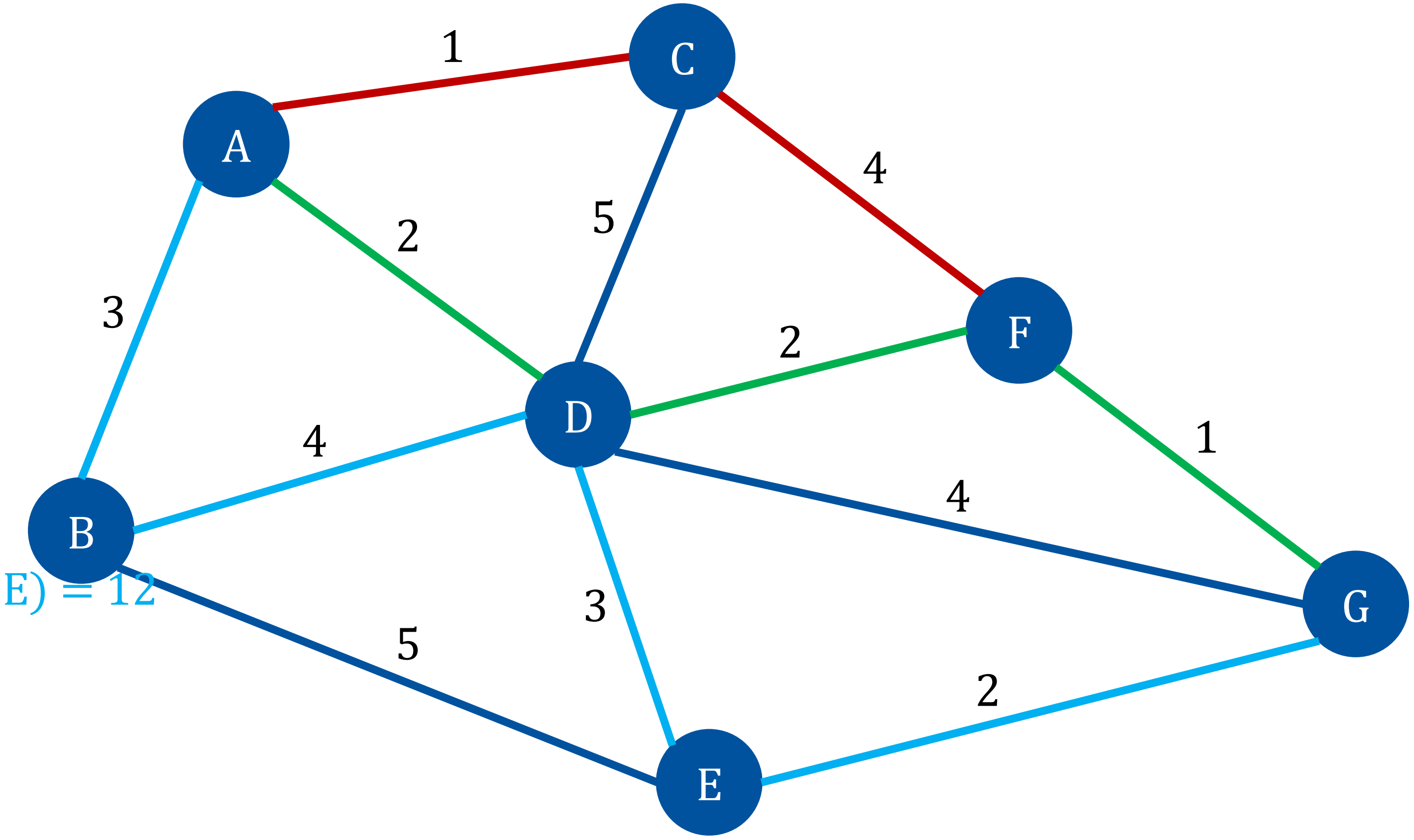
Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht

Entscheidungsfolge $x = (1, 4, 1)$
$q(x) = q(1, A) + q(4, C) + q(1, F) = 6$

Entscheidungsfolge $y = (2, 2, 1)$
$q(y) = q(2, A) + q(2, D) + q(1, F) = 5$

Entscheidungsfolge $z = (3, 4, 3, 2)$
$q(z) = q(3, A) + q(4, B) + q(3, D) + q(2, E) = 12$

Entscheidungsfolge …



Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht

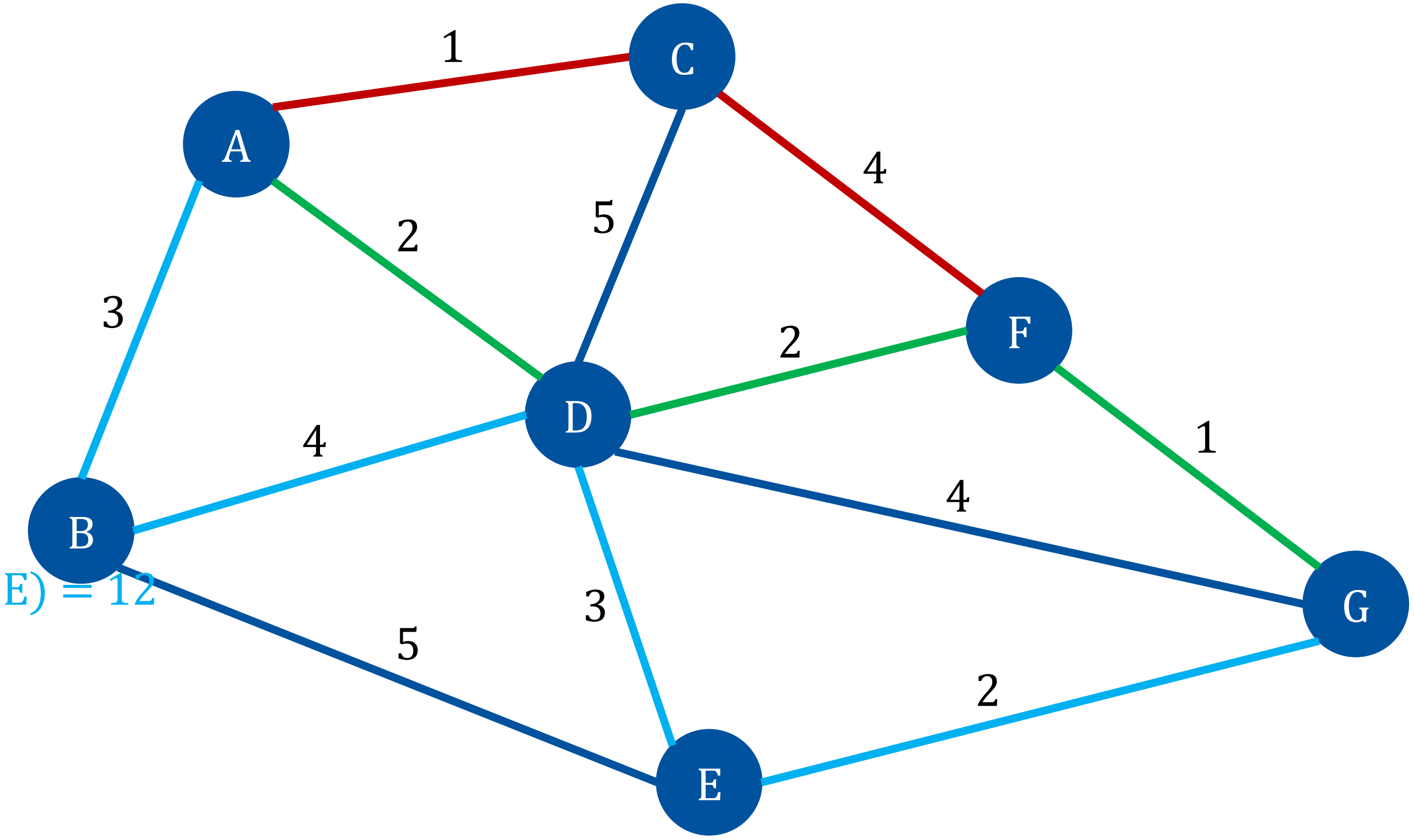
Entscheidungsfolge $x = (1, 4, 1)$
$q(x) = q(1, A) + q(4, C) + q(1, F) = 6$

Entscheidungsfolge $y = (2, 2, 1)$
$q(y) = q(2, A) + q(2, D) + q(1, F) = 5$

Entscheidungsfolge $z = (3, 4, 3, 2)$
$q(z) = q(3, A) + q(4, B) + q(3, D) + q(2, E) = 12$

Entscheidungsfolge …

$\min\{ q(x), q(y), q(z), \ldots \} = 5$



Beispiel: Entscheidungsfolge

Gesucht: Minimaler Weg von A nach G (bzgl. der Kantengewichte)
Lösungsraum S: Alle möglichen Wege von A nach G
Zustände s_k der Entscheidungsfolge: Knoten
Entscheidungen d_k: Kanten
Güte $q(d_k, s_k)$: Kantengewicht

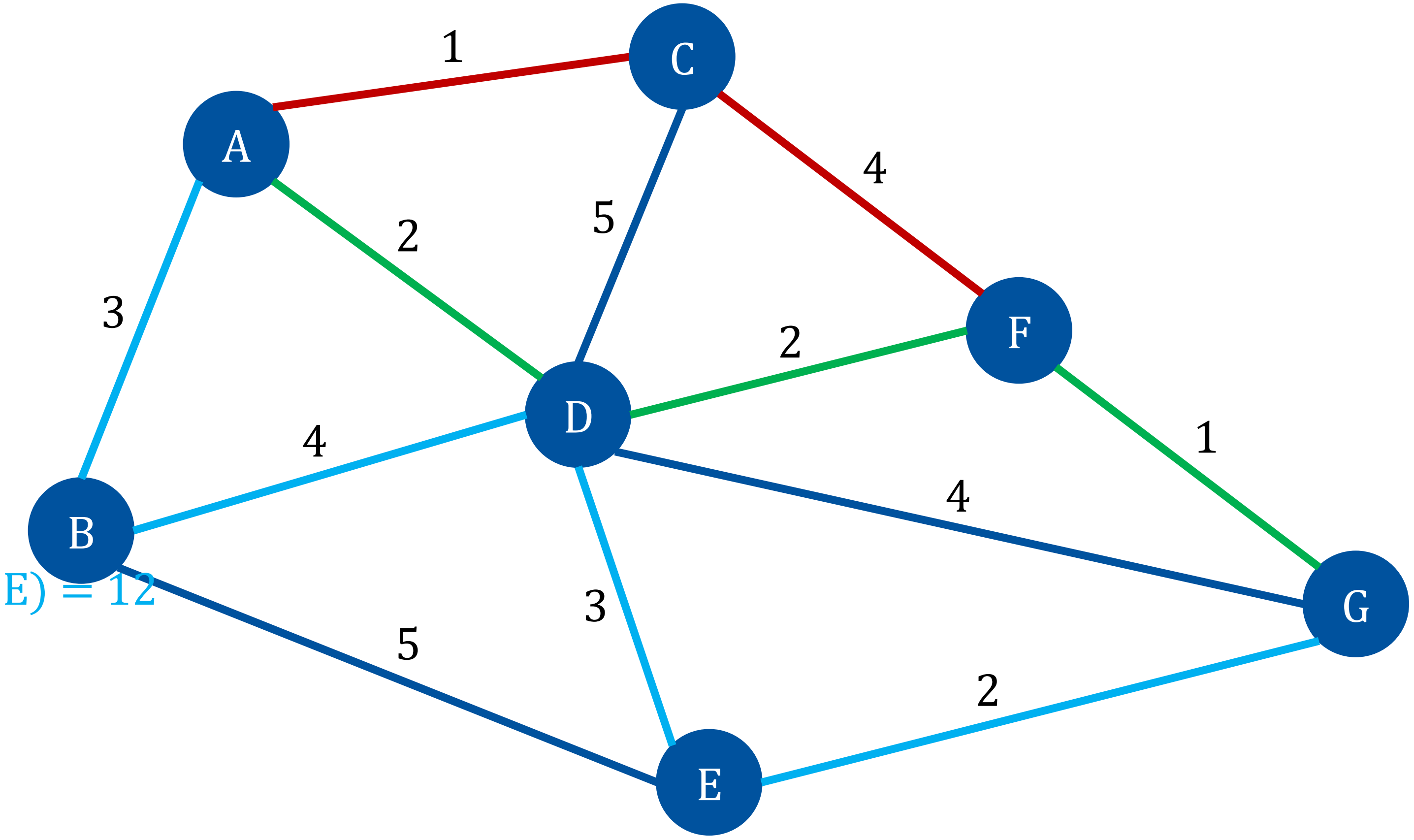
Entscheidungsfolge $x = (1, 4, 1)$
$q(x) = q(1, A) + q(4, C) + q(1, F) = 6$

Entscheidungsfolge $y = (2, 2, 1)$
$q(y) = q(2, A) + q(2, D) + q(1, F) = 5$

Entscheidungsfolge $z = (3, 4, 3, 2)$
$q(z) = q(3, A) + q(4, B) + q(3, D) + q(2, E) = 12$

Entscheidungsfolge …

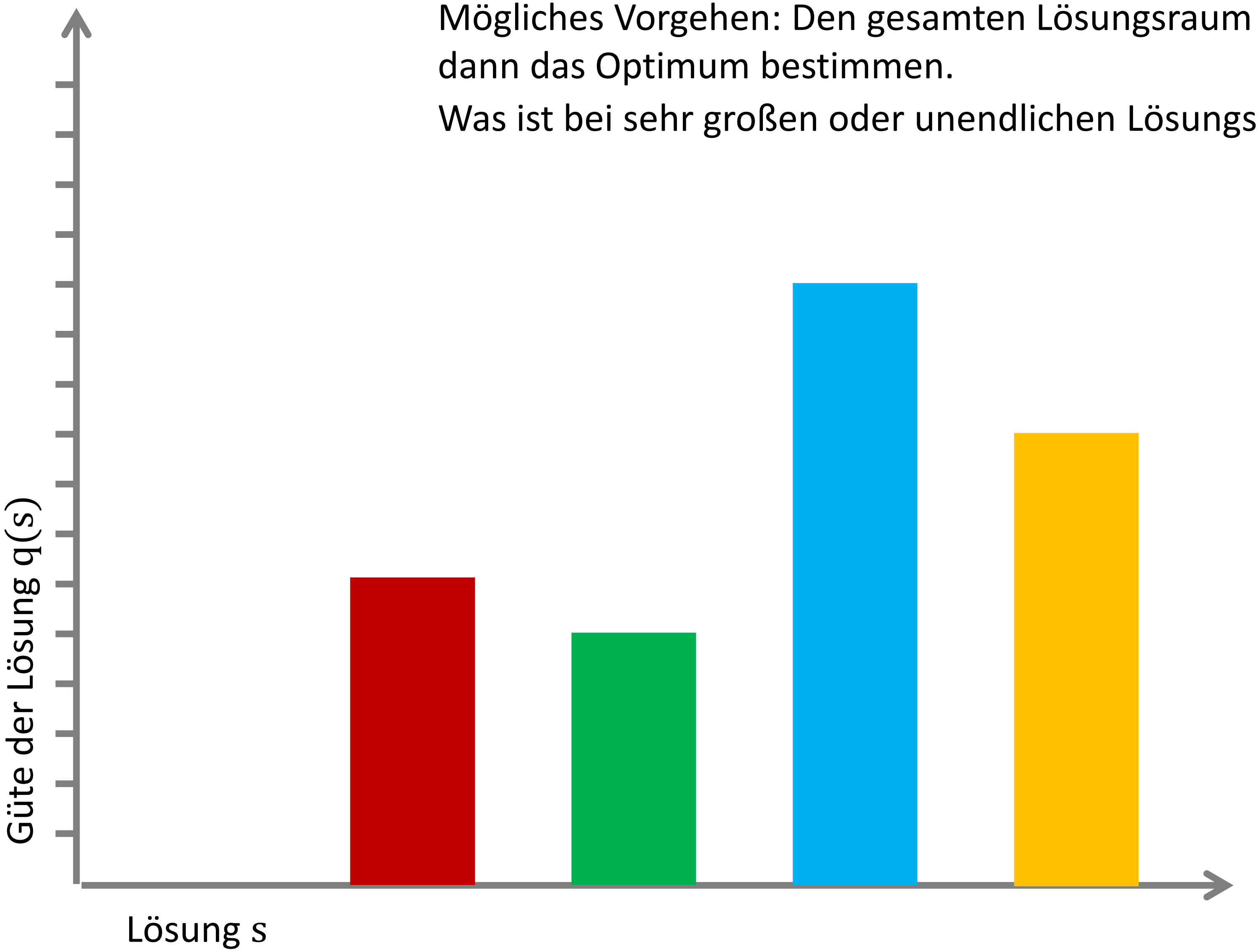
$\min\{ q(x), q(y), q(z), \dots \} = 5$
Minimaler Weg: A, D, F, G



Beispiel: Entscheidungsfolge (Lösungsraum)

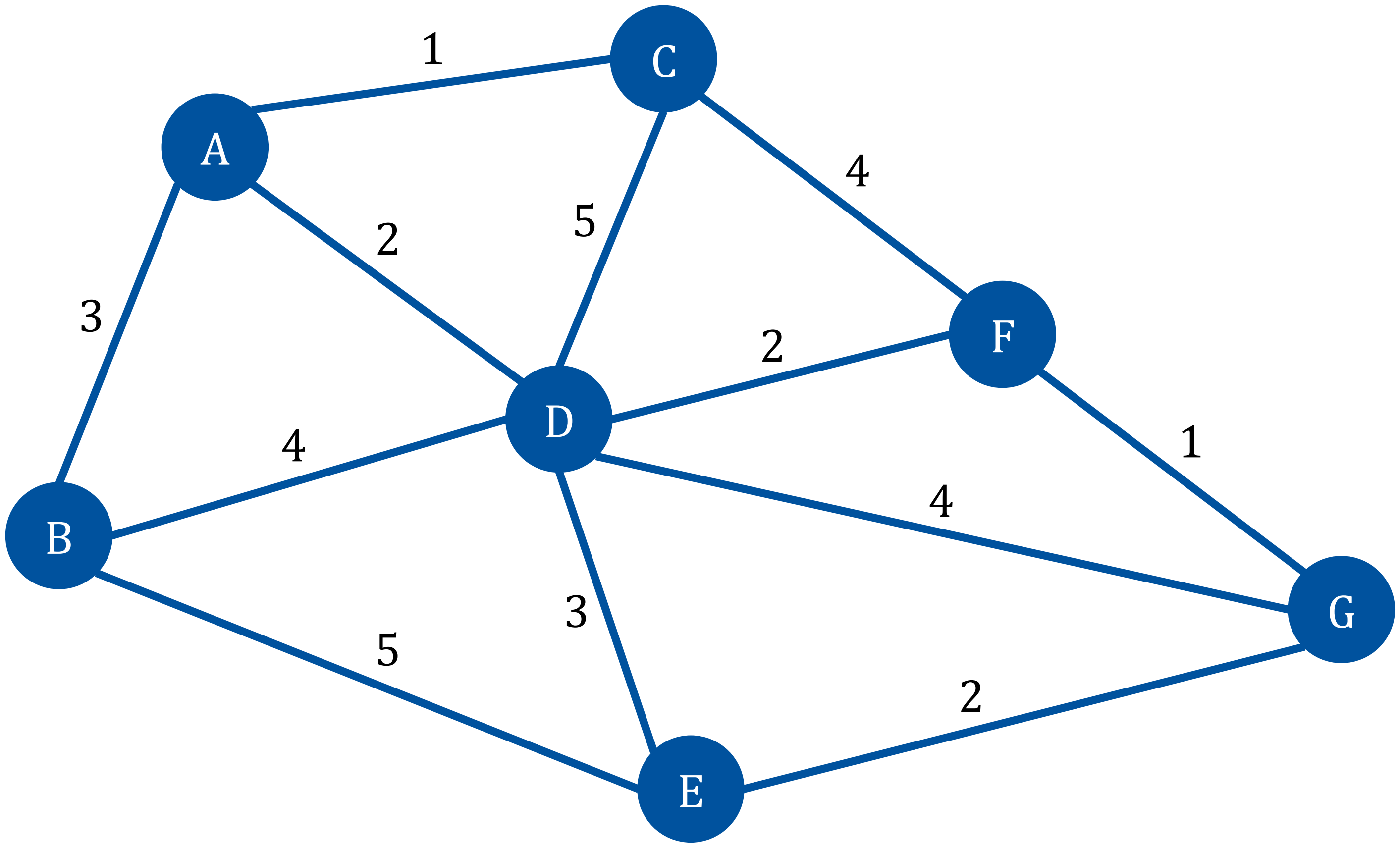
Mögliches Vorgehen: Den gesamten Lösungsraum „ausmessen“, dann das Optimum bestimmen.

Was ist bei sehr großen oder unendlichen Lösungsräumen?



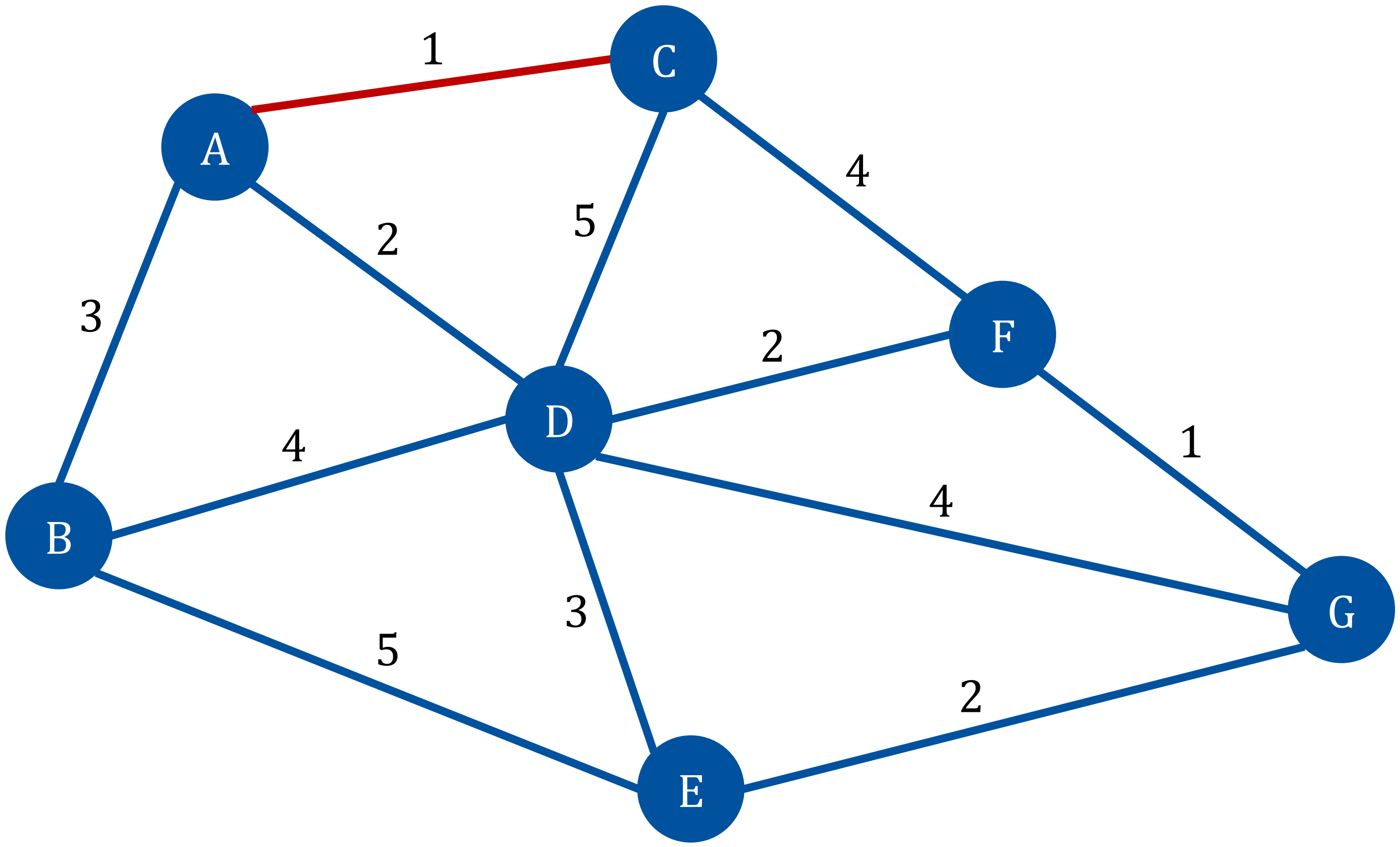
Beispiel: Greedy-Vorgehen

Wähle immer die im aktuellen Zustand beste Kante.
Revidiere diese Entscheidung nicht mehr.



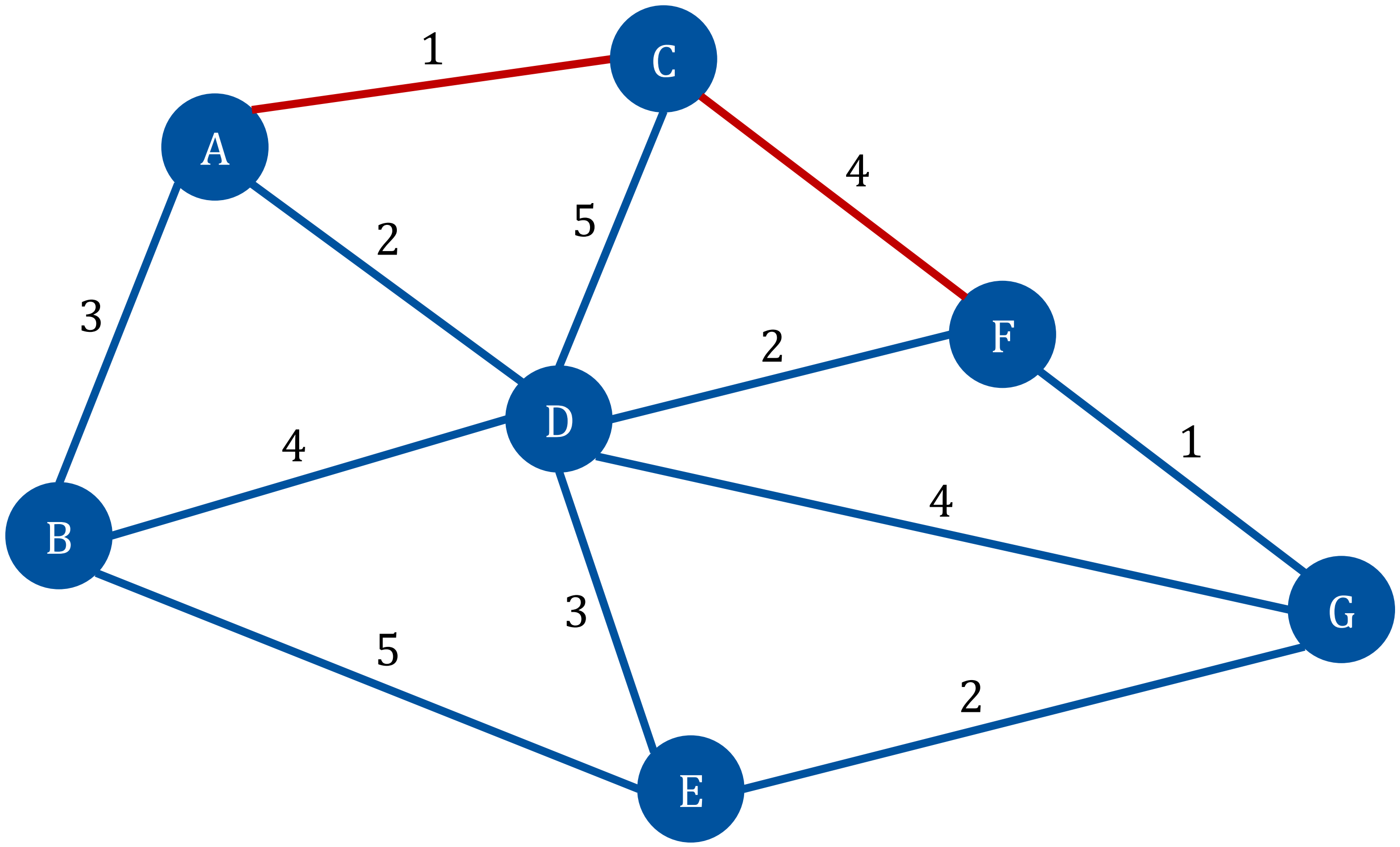
Beispiel: Greedy-Vorgehen

Wähle immer die im aktuellen Zustand beste Kante.
Revidiere diese Entscheidung nicht mehr.



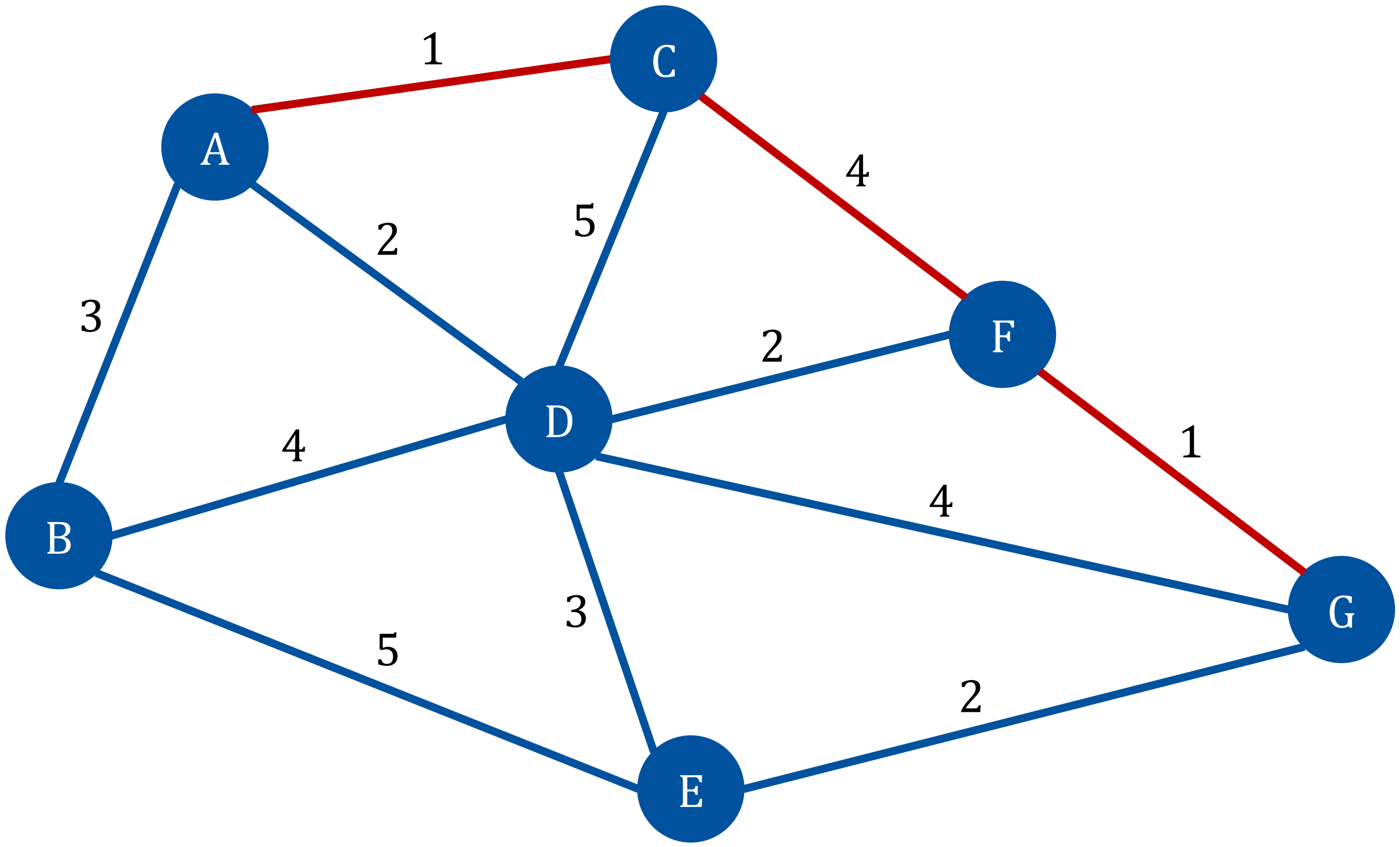
Beispiel: Greedy-Vorgehen

Wähle immer die im aktuellen Zustand beste Kante.
Revidiere diese Entscheidung nicht mehr.



Beispiel: Greedy-Vorgehen

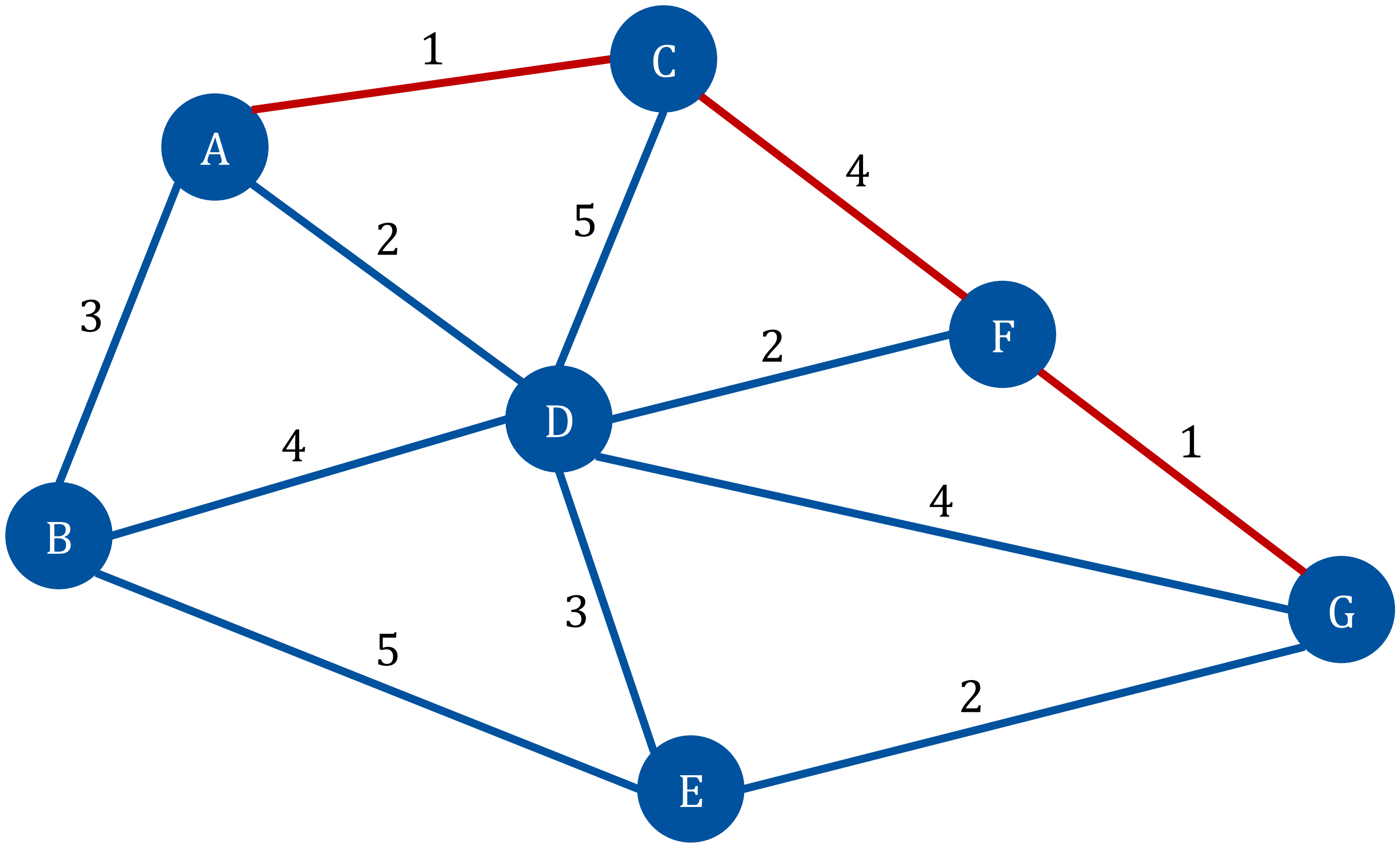
Wähle immer die im aktuellen Zustand beste Kante.
Revidiere diese Entscheidung nicht mehr.



Beispiel: Greedy-Vorgehen

Wähle immer die im aktuellen Zustand beste Kante.
Revidiere diese Entscheidung nicht mehr.

Vorteil: Schnell eine gute Lösung gefunden.
Nachteil: Nicht die beste Lösung gefunden.



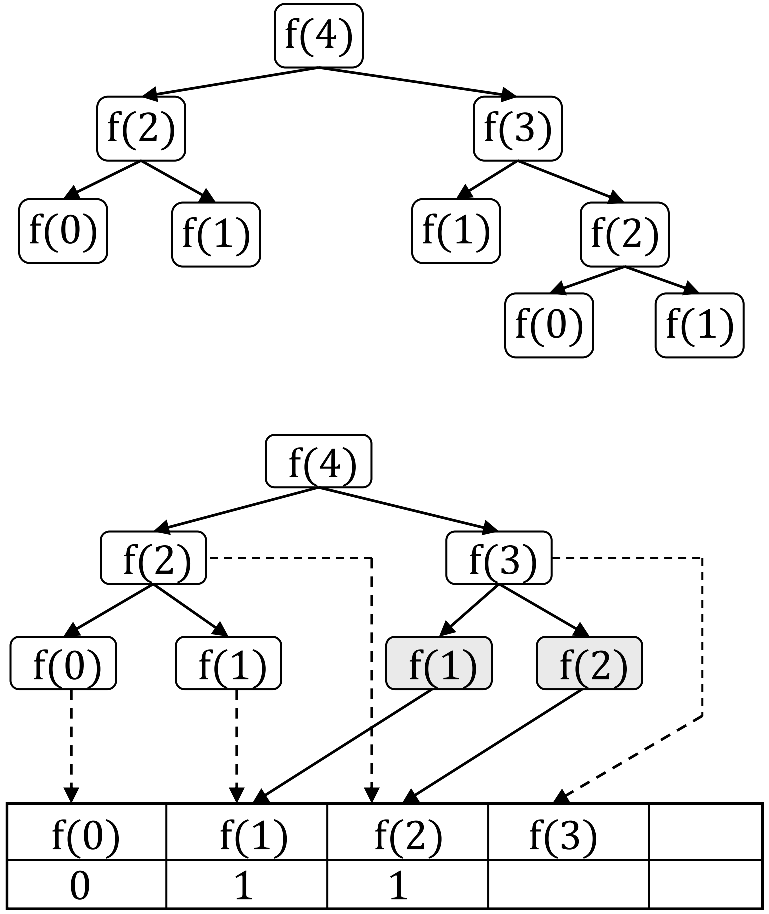
Dynamische Programmierung

Dynamische Programmierung

- Strategie zum Lösen von Optimierungsproblemen
- Voraussetzung: Das Optimierungsproblem
  - lässt sich in homogene Teilprobleme zerlegen
  - (und einige dieser Teilprobleme überlappen)
  - erfüllt das Optimalitätsprinzip von Bellmann

Beispiel: Überlappende Teilprobleme

- Rekursive Berechnung der Fibonacci-Zahlen
  - $f(n) = f(n\text{-}1) + f(n\text{-}2)$
    - $f(0) = 0$
    - $f(1) = 1$
  - Überlappende Teilprobleme: Mehrfacher Aufruf der Funktion für gleiche Argumente
    → unnötiger Rechenaufwand
- Besser: Memoisation
  - speichere bereits berechnete Teilprobleme



Bellmannsches Optimalitätskriterium

- Sei $(d_n, d_{n-1}, \ldots, d_1)$ eine optimale Entscheidungsfolge
- Das Bellmannsche Optimalitätskriterium gilt, wenn sich jede Teilfolge $(d_j, \ldots, d_i)$ optimaler Entscheidungen zu einer optimalen Gesamtfolge fortsetzen lässt

$$s_n \xrightarrow{t(d_n,s_n)} s_{n-1} \longrightarrow \ldots \xrightarrow{t(d_{i+1},s_{i+1})} s_i \xrightarrow{t(d_i,s_i)} \ldots \xrightarrow{t(d_2,s_2)} s_1 \xrightarrow{t(d_1,s_1)} s_0$$

- Jedes $s_{i-1} = t(d_i, s_i)$ ist eine optimale Teillösung
- Teilfolgen optimaler Entscheidungen sind selber wieder optimal

Bellmannsches Optimalitätskriterium (2)

- Beispiel:
  - kürzester Weg von A nach G ist (A, D, F, G)
  - dann ist der kürzeste Weg von A nach F: (A, D, F)
  - wenn es einen kürzeren Weg von A nach F gäbe, z.B. (A, H, F),
    dann wäre (A, H, F, G) auch ein kürzerer Weg von A nach G

- Gegenbeispiel:
  - minimaler Weg von A nach G
  - aber an einigen Knoten gibt es Abbiegebeschränkungen (wie z.B. in Bremen)
  - minimaler Weg von A nach F führt über C,
    aber bei F darf man von C kommend nicht nach G abbiegen
  - ➢ minimaler Weg von A nach G über F führt über D

Dynamische Programmierung (2)

- Ziel: Entscheidungsfolge $x = (d_n, d_{n-1}, \ldots, d_1)$ mit optimaler (maximaler) Gesamtgüte $q(x)$
- Bellmannsches Optimalitätskriterium erlaubt rekursive Zerlegung
  - suche optimale Entscheidung d_n und
  - suche optimale Entscheidungsfolge $(d_{n-1}, \ldots, d_1)$

- $\max\{\, q(x) \,\} = \max\{\, q(d_n, s_n) + \ldots + q(d_1, s_1) \mid d_i \in D \,\}$

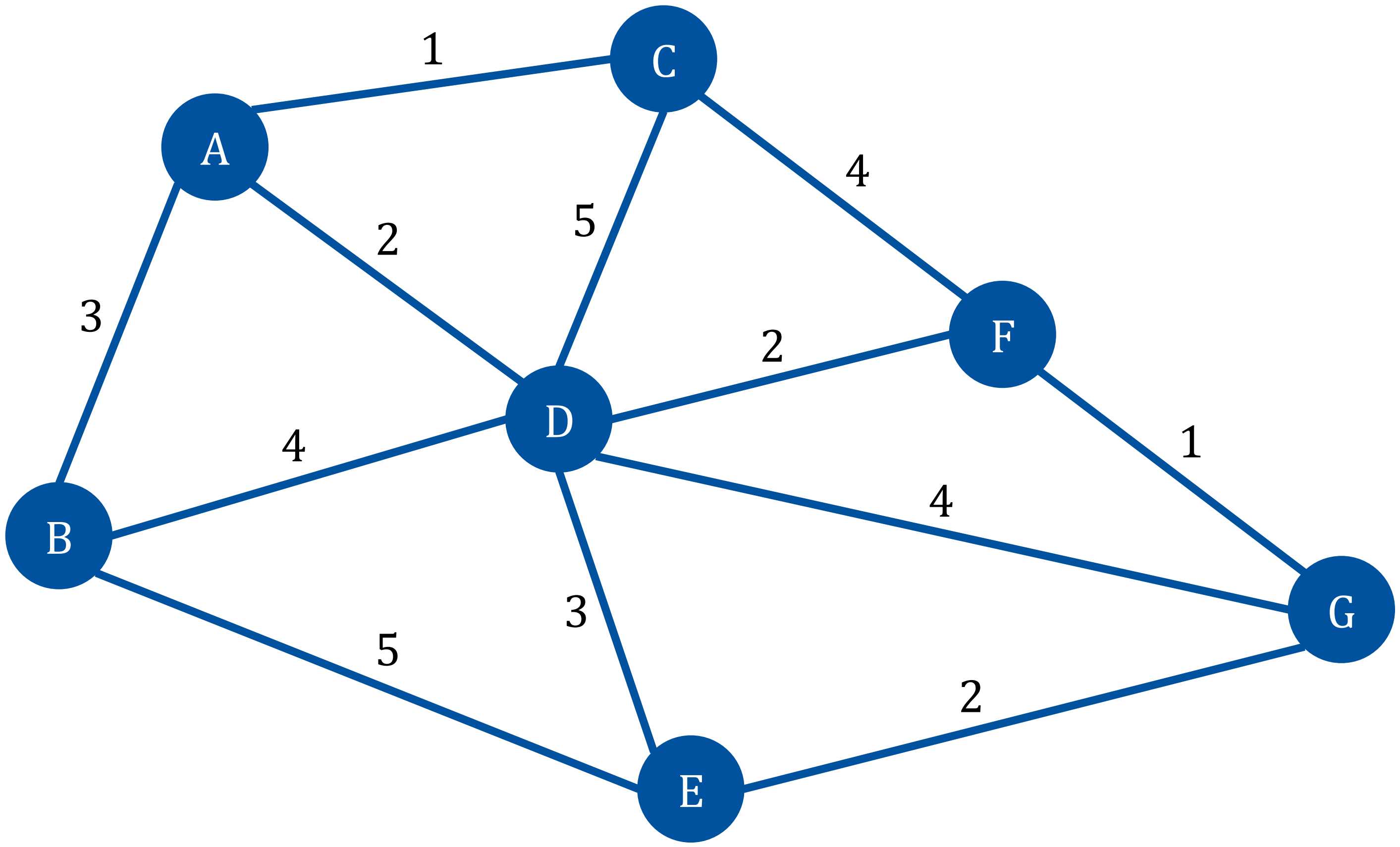
$$= \max\{\, q(d_n, s_n) \,\} + \max\{\, q(d_{n-1}, s_{n-1}) + \ldots + q(d_1, s_1) \,\}$$

$$= \max\{\, q(d_n, s_n) \,\} + \max\{\, q(d_{n-1}, s_{n-1}) \,\} + \max\{\, \ldots \,\}$$

Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

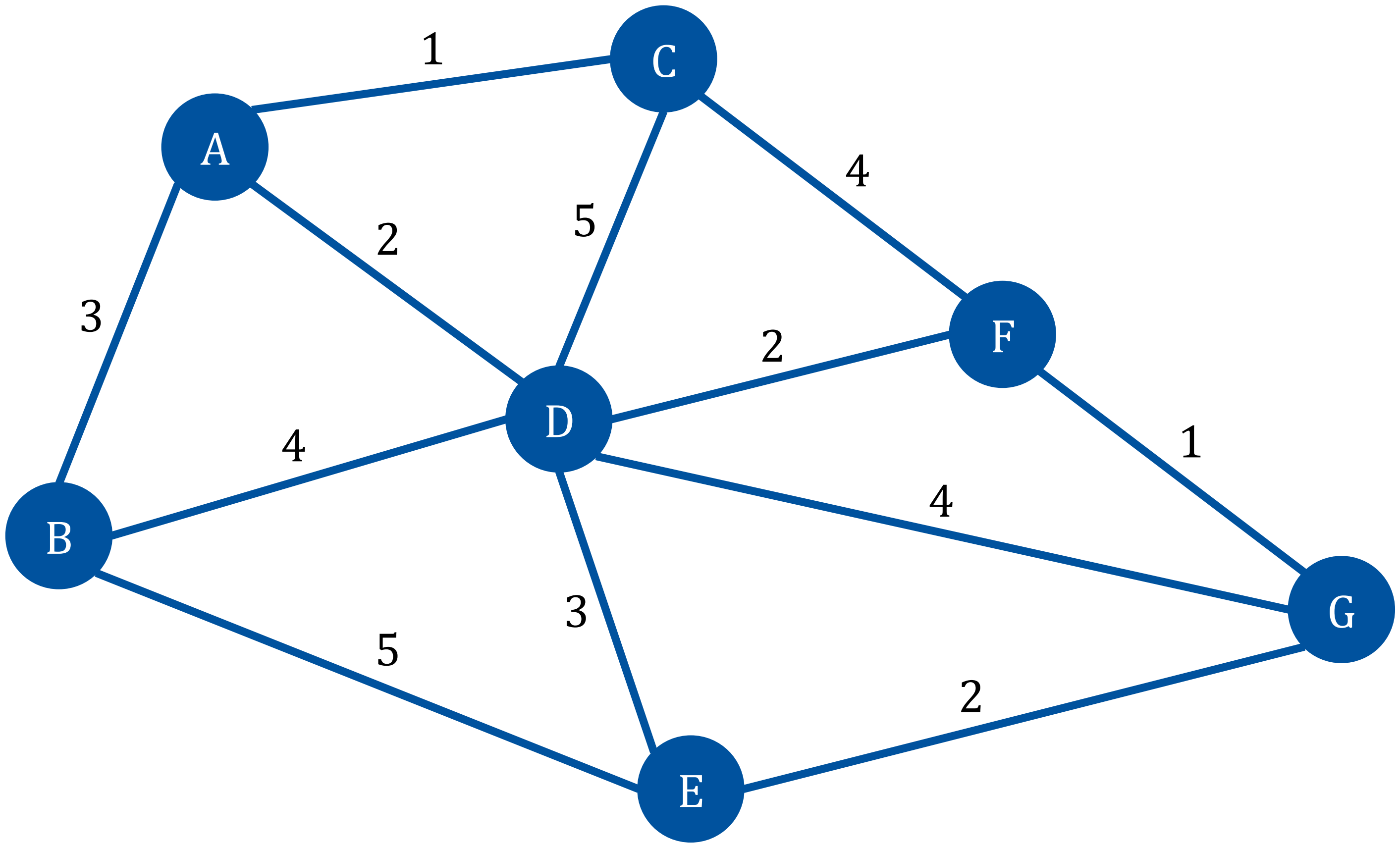
d_0	A	B	C	D	E	F	G
A	0	3	1	2	∞	∞	∞
B	3	0	∞	4	5	∞	∞
C	1	∞	0	5	∞	4	∞
D	2	4	5	0	3	2	4
E	∞	5	∞	3	0	∞	2
F	∞	∞	4	2	∞	0	1
G	∞	∞	∞	4	2	1	0



Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

d_0	A	B	C	D	E	F	G
A	0	3	1	2	∞	∞	∞
B		0	∞	4	5	∞	∞
C			0	5	∞	4	∞
D				0	3	2	4
E					0	∞	2
F						0	1
G							0

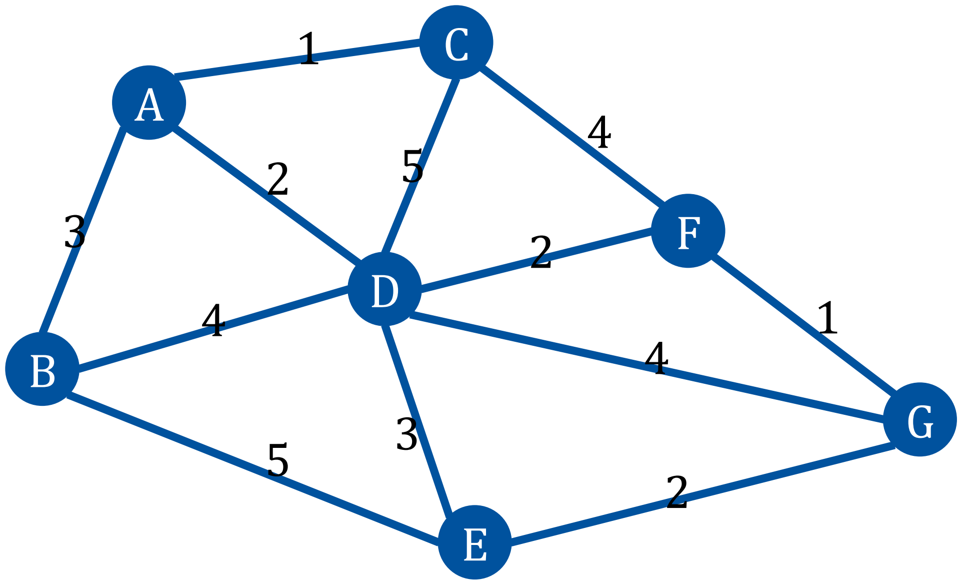


Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)
Beobachtung: Der minimale Weg von u nach w kostet entweder

- das bekannte $d(u, w)$ (siehe Tabelle), oder
- $d(u, v) + d(v, w)$ für einen Zwischenknoten v

d_0	A	B	C	D	E	F	G
A	0	3	1	2	∞	∞	∞
B		0	∞	4	5	∞	∞
C			0	5	∞	4	∞
D				0	3	2	4
E					0	∞	2
F						0	1
G							0



Beispiel: Dynamische Programmierung

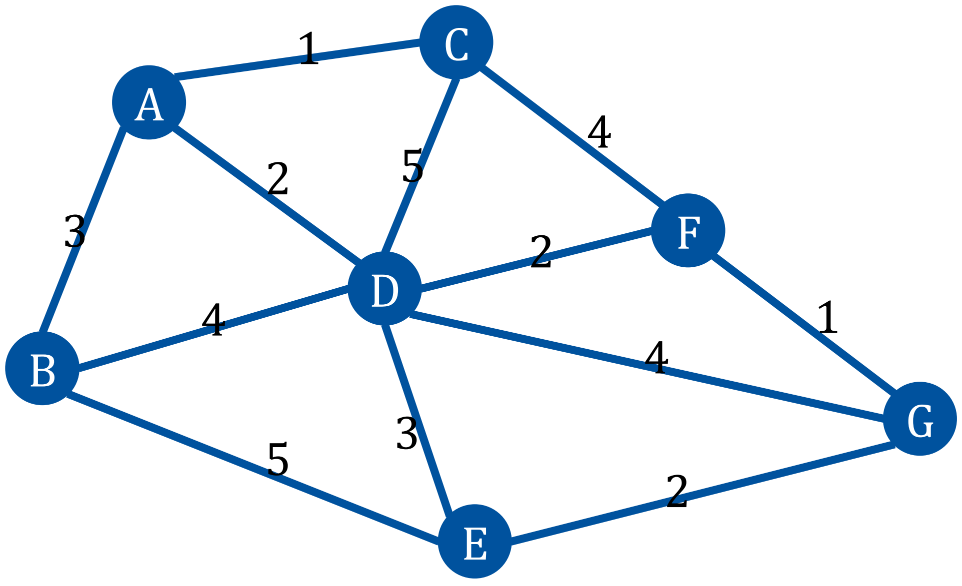
Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

Beobachtung: Der minimale Weg von u nach w kostet entweder

- das bekannte $d(u, w)$ (siehe Tabelle), oder
- $d(u, v) + d(v, w)$ für einen Zwischenknoten v

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

d_0	A	B	C	D	E	F	G
A	0	3	1	2	∞	∞	∞
B		0	∞	4	5	∞	∞
C			0	5	∞	4	∞
D				0	3	2	4
E					0	∞	2
F						0	1
G							0



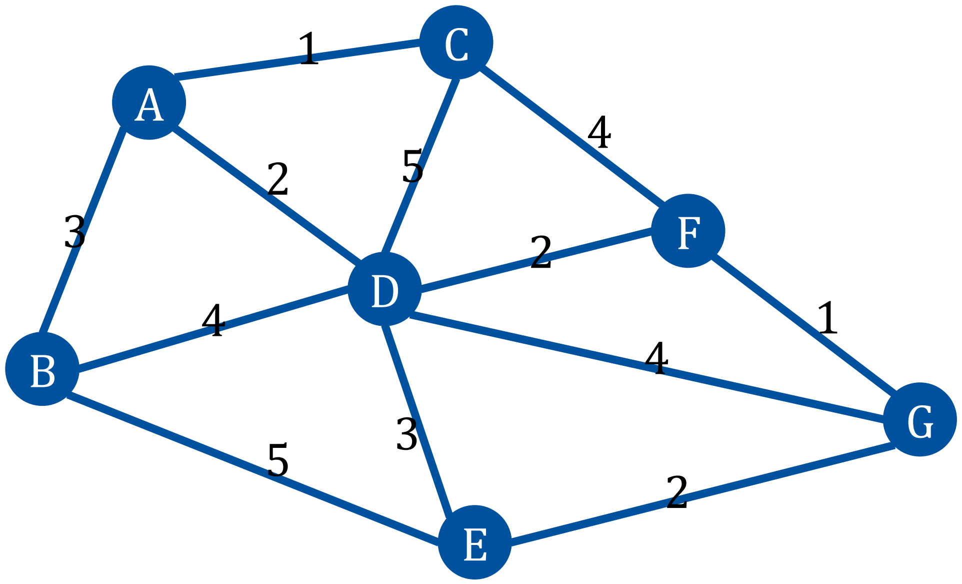
Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Rekursive Umsetzung für $d(A, G)$:

$d(A, G)$



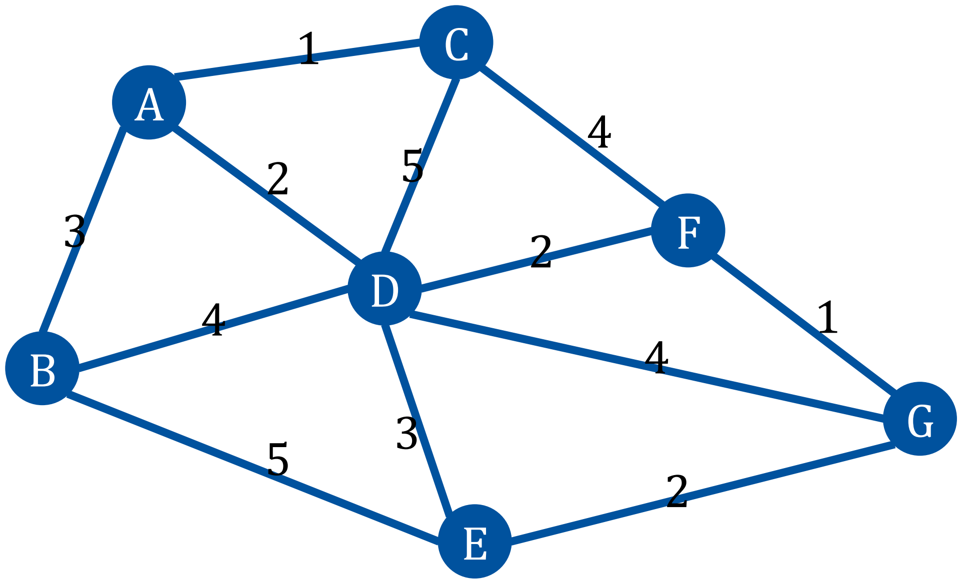
Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Rekursive Umsetzung für $d(A, G)$:

$d(A, G) = \min \{ d(A, G), d(A, D) + d(D, G), d(A, E) + d(E, G), d(A, F) + d(F, G) \}$



Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Rekursive Umsetzung für d(A, G):

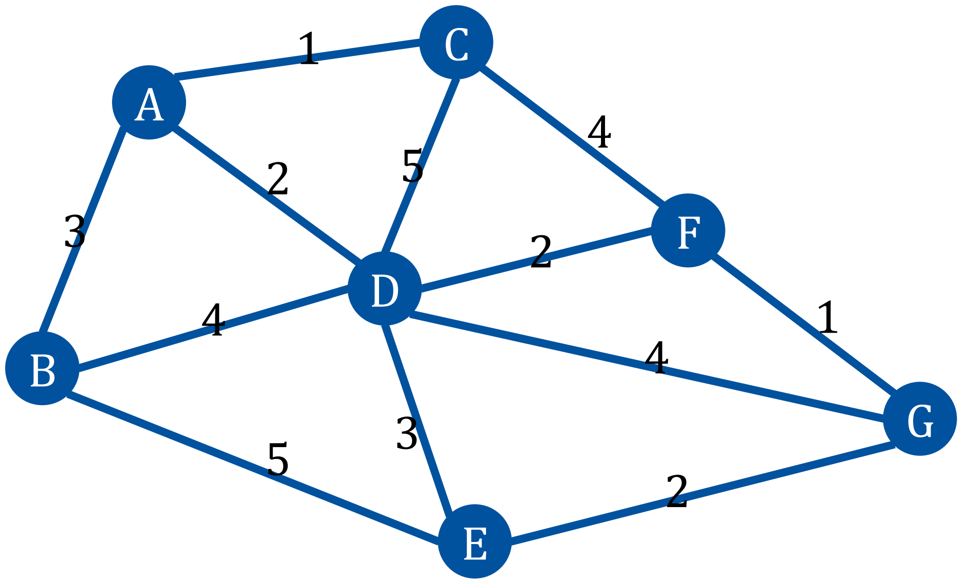
$d(A, G) = \min \{ d(A, G), d(A, D) + d(D, G), d(A, E) + d(E, G), d(A, F) + d(F, G) \}$

$= \min \{ d(A, G),$

$\min \{ d(A, D), d(A, B) + d(B, D), d(A, C) + d(C, D), d(A, E) + d(E, D), d(A, F) + d(F, D), d(A, G) + d(G, D) \} + 4,$

$\min \{ d(A, E), d(A, B) + d(B, E), d(A, D) + d(D, E), d(A, G) + d(G, E) \} + 2,$

$\min \{ d(A, F), d(A, C) + d(C, F), d(A, D) + d(D, F), d(A, G) + d(G, F) \} + 1 \}$



Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Rekursive Umsetzung für $d(A, G)$:

$d(A, G) = \min \{ d(A, G), d(A, D) + d(D, G), d(A, E) + d(E, G), d(A, F) + d(F, G) \}$

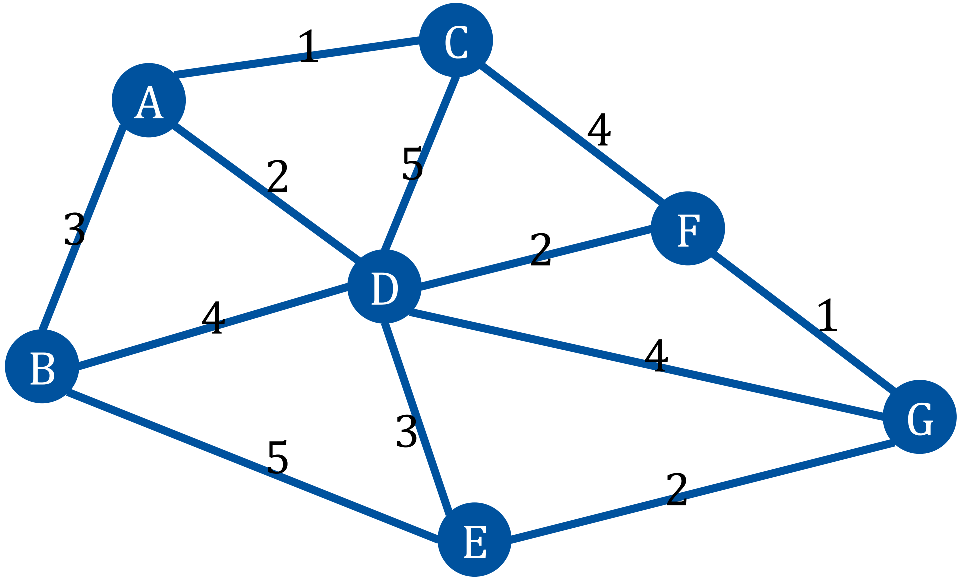
$= \min \{ d(A, G),$

$\quad \min \{ d(A, D), d(A, B) + d(B, D), d(A, C) + d(C, D), d(A, E) + d(E, D), d(A, F) + d(F, D), d(A, G) + d(G, D) \} + 4,$

$\quad \min \{ d(A, E), d(A, B) + d(B, E), d(A, D) + d(D, E), d(A, G) + d(G, E) \} + 2,$

$\quad \min \{ d(A, F), d(A, C) + d(C, F), d(A, D) + d(D, F), d(A, G) + d(G, F) \} + 1 \}$

$= \ldots$



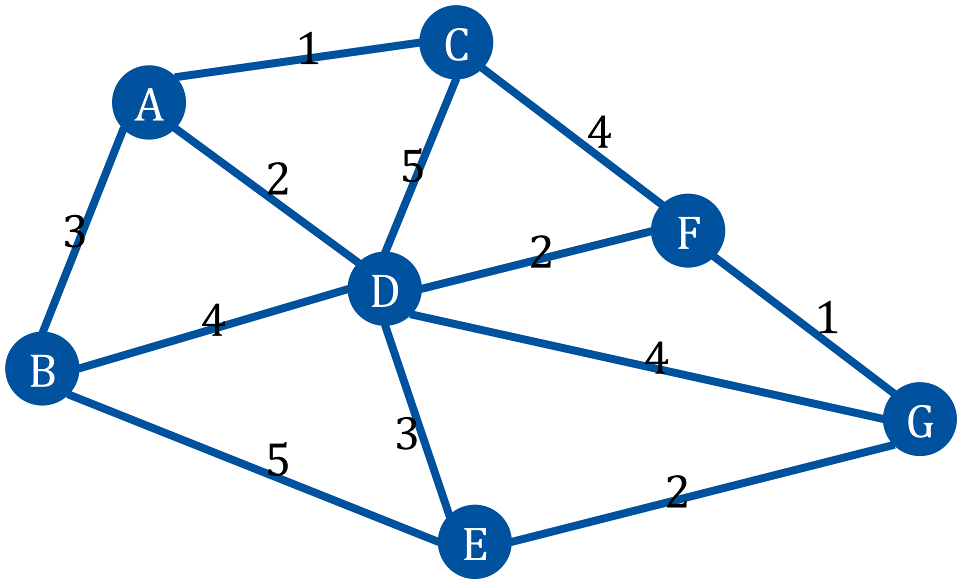
Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Umsetzung durch Dynamische Programmierung mit Memoisation:

- alte $d(u, w)$ merken,
- schrittweise Entfernungen aktualisieren



Beispiel: Dynamische Programmierung

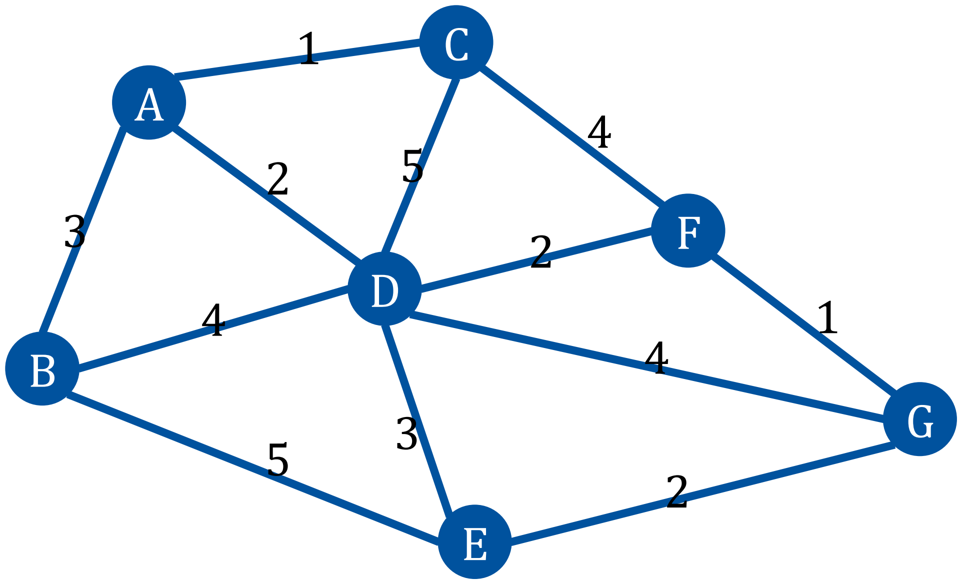
Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Umsetzung durch Dynamische Programmierung mit Memoisation:

- alte $d(u, w)$ merken,
- schrittweise Entfernungen aktualisieren

d_0	A	B	C	D	E	F	G
A	0	3	1	2	∞	∞	∞
B		0	∞	4	5	∞	∞
C			0	5	∞	4	∞
D				0	3	2	4
E					0	∞	2
F						0	1
G							0



Beispiel: Dynamische Programmierung

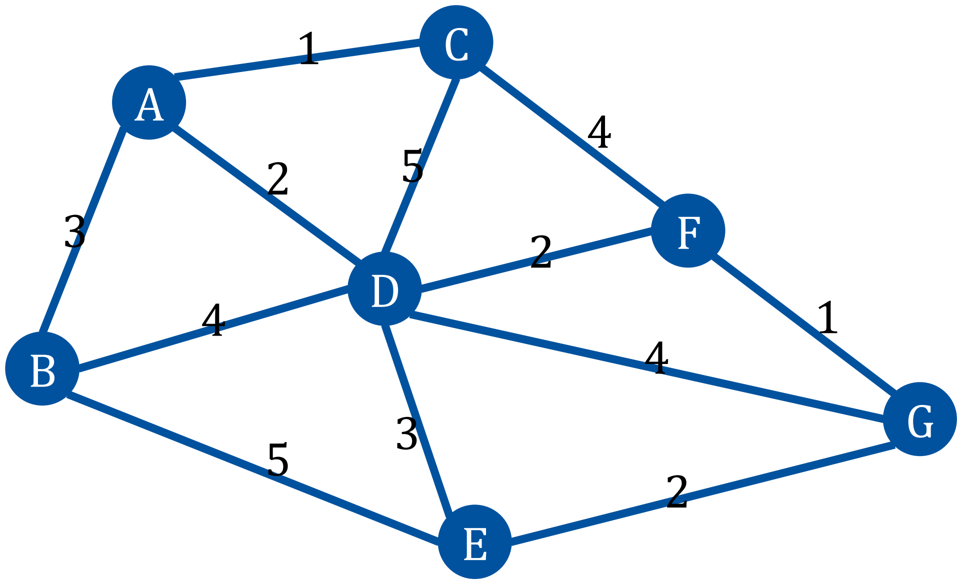
Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Umsetzung durch Dynamische Programmierung mit Memoisation:

- alte $d(u, w)$ merken,
- schrittweise Entfernungen aktualisieren

d_1	A	B	C	D	E	F	G
A	0	3	1	2	5	4	6
B		0	4	4	5	6	8
C			0	3	6	4	7
D				0	3	2	4
E					0	5	2
F						0	1
G							0



Beispiel: Dynamische Programmierung

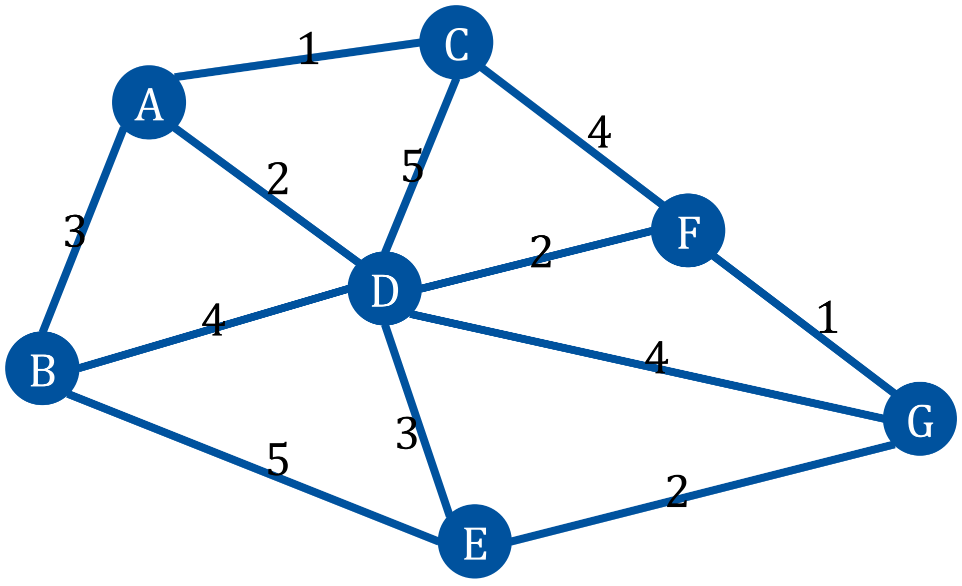
Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Umsetzung durch Dynamische Programmierung mit Memoisation:

- alte $d(u, w)$ merken,
- schrittweise Entfernungen aktualisieren

d_2	A	B	C	D	E	F	G
A	0	3	1	2	5	4	5
B		0	4	4	5	6	7
C			0	3	6	4	5
D				0	3	2	3
E					0	3	2
F						0	1
G							0



Beispiel: Dynamische Programmierung

Gesucht: Minimaler Weg zwischen **allen** Knotenpaaren (bzgl. der Kantengewichte)

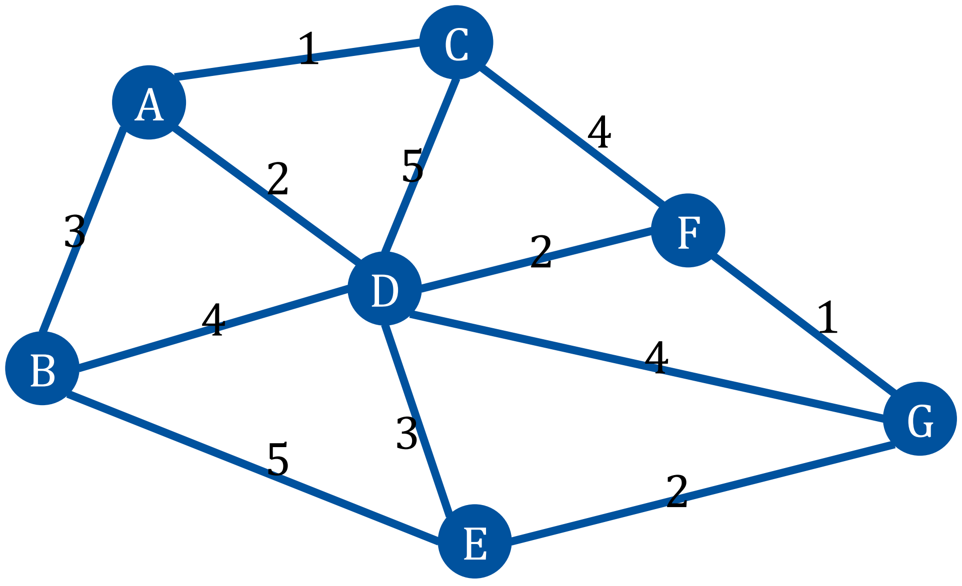
$d_{new}(u, w) = \min\{ d_{old}(u, w), d(u, v) + d(v, w) \}, v \in V$

Umsetzung durch Dynamische Programmierung mit Memoisation:

- alte $d(u, w)$ merken,
- schrittweise Entfernungen aktualisieren

d_2	A	B	C	D	E	F	G
A	0	3	1	2	5	4	5
B		0	4	4	5	6	7
C			0	3	6	4	5
D				0	3	2	3
E					0	3	2
F						0	1
G							0

→ Algorithmus von Floyd-Warshall



Algorithmus von Floyd-Warshall

```
int n = |V|;
double[][] d = new double[n][n];
initialise d with Double.POSITIVE_INFINITY;

for (v : V)
    d[v][v] = 0;
for ((u,v) : E)
    d[u][v] = directCost(u, v);

for (v : V)
    for (u : V)
        for (w : V)
            double newPath = d[u][v] + d[v][w];
            if (d[u][w] > newPath)
                d[u][w] = newPath;
```

Algorithmus von Floyd-Warshall (Java)

```java
public class FloydWarshall {

    public static void main(String[] args) {
        double[][] d = create(7);
        put(d, 0, 1, 3);
        put(d, 0, 2, 1);
        put(d, 0, 3, 2);
        put(d, 1, 3, 4);
        put(d, 1, 4, 5);
        put(d, 2, 3, 5);
        put(d, 2, 5, 4);
        put(d, 3, 4, 3);
        put(d, 3, 5, 2);
        put(d, 3, 6, 4);
        put(d, 4, 6, 2);
        put(d, 5, 6, 1);
        print(d);
        calculate(d);
        System.out.println();
        print(d);
    }
```

Algorithmus von Floyd-Warshall (Java)

```java
public static double[][] create(int size) {
    double[][] d = new double[size][size];
    for (int row = 0; row < d.length; row++) {
        for (int col = 0; col < d[row].length; col++) {
            if (row == col) {
                d[row][col] = 0.0;
            } else {
                d[row][col] = Double.POSITIVE_INFINITY;
            }
        }
    }
    return d;
}

public static void put(double[][] d, int x, int y, double value) {
    d[x][y] = value;
    d[y][x] = value;
}
```

Algorithmus von Floyd-Warshall (Java)

```java
    public static void calculate(double[][] d) {
        for (int v = 0; v < d.length; v++) {
            for (int u = 0; u < d.length; u++) {
                for (int w = 0; w < d.length; w++) {
                    double newPath = d[u][v] + d[v][w];
                    if (d[u][w] > newPath) {
                        d[u][w] = newPath;
                    }
                }
            }
        }
    }

    public static void print(double[][] d) { … }

}
```

Algorithmus von Floyd-Warshall (Java)

```
0 3 1 2 ? ? ?
3 0 ? 4 5 ? ?
1 ? 0 5 ? 4 ?
2 4 5 0 3 2 4
? 5 ? 3 0 ? 2
? ? 4 2 ? 0 1
? ? ? 4 2 1 0

0 3 1 2 5 4 5
3 0 4 4 5 6 7
1 4 0 3 6 4 5
2 4 3 0 3 2 3
5 5 6 3 0 3 2
4 6 4 2 3 0 1
5 7 5 3 2 1 0
```

Fazit: Dynamische Programmierung

- Einsatz bei komplexen Problemen, z.B. NP-schwere Probleme wie TSP, Rucksack, …
- Vorgehensweise: Überlappende Teilprobleme erkennen, lösen, speichern und wiederverwenden
  → senkt Berechnungskomplexität zu Lasten der Speicherkomplexität
- Nicht sinnvoll
  - bei zu großen Problemgrößen
  - bei zu tiefen Rekursionen
  - bei zu vielen Teilproblemen → zu großer Speicherbedarf
  - wenn effizientere Strategie bekannt ist, z.B. eine Greedy-Strategie

Lernziele

- Best Practices
- Optimierungsprobleme
- Dynamische Programmierung