Programmierung 2
Studiengang MI / WI

Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm

Hochschule Harz

FB Automatisierung und Informatik

mwilhelm@hs-harz.de

Raum 2.202

Tel. 03943 / 659 338

Inhalt der Vorlesung

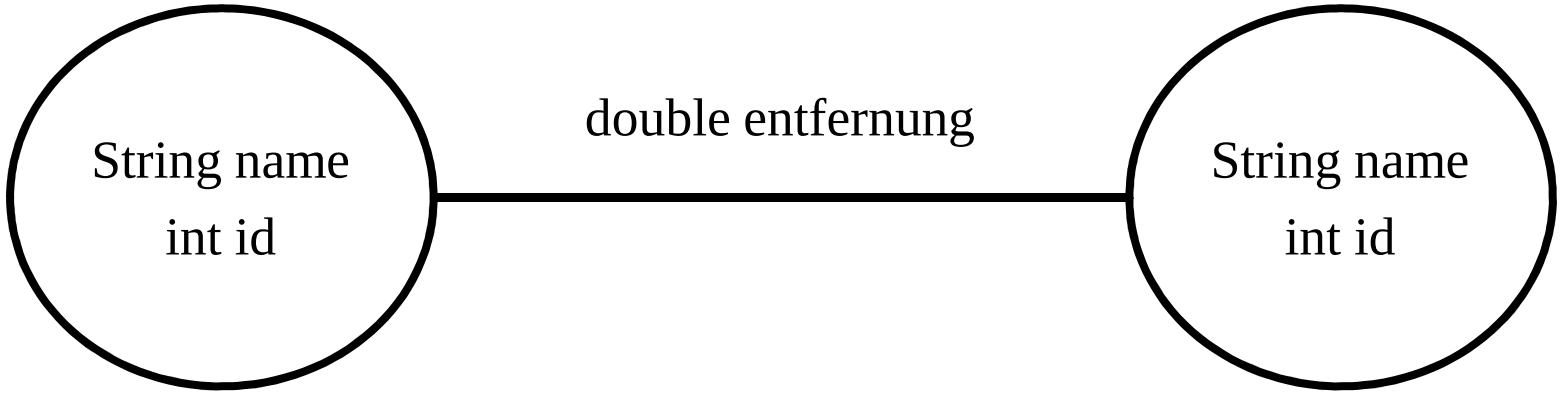
Überblick:

- Objekte und Methoden
- Swing
- Exception
- I/O-Klassen / Methoden
- Threads
- Algorithmen (Das Collections-Framework)
- Design Pattern
- **Graphentheorie**
- JUnit

Graph-Algorithmus

Was ist ein Graph?

- Ein Graph besteht aus Vertices, Plätze, und Kanten, welche jeweils zwei Vertices verbinden.
- Sowohl die Vertices als auch die Kanten können eigene Attribute besitzen.



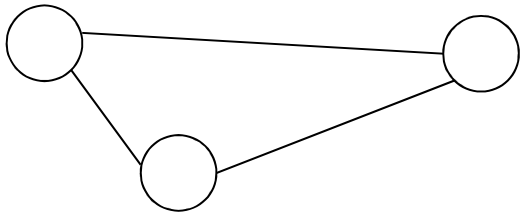
Teilimplementierung:

- Vertices sind explizit durch Kanten verbunden.
- Ein Vertex kann durch viele Kanten mit ebenso vielen Vertices verbunden sein.
- Kanten haben einen Anfangs- und einen Endvertex.

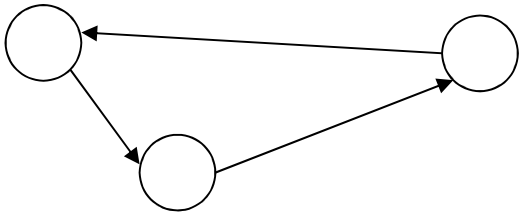
```
class Vertex {
    private Kante[] kanten;
    .....
}
class Kante {
    private double abstand;
    private Vertex anfang;
    private Vertex ende;
    ......
}
```

Arten der Graphen

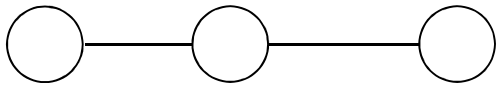
- Kanten ohne Richtung, zyklisch



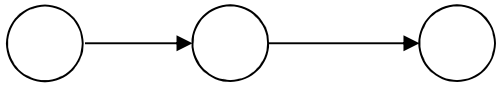
- Kanten haben Richtung, zyklisch



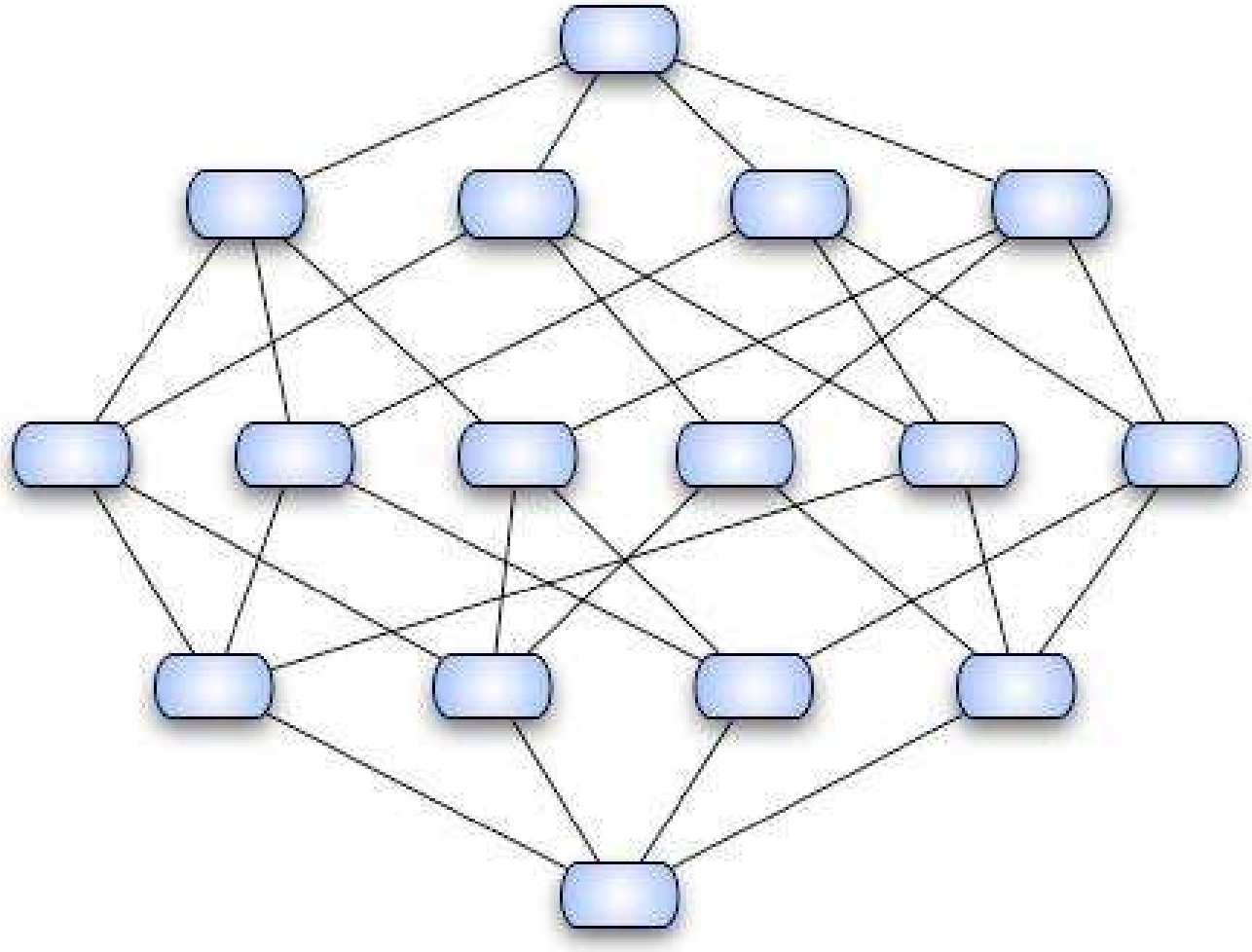
- Kanten ohne Richtung, keine Zyklen



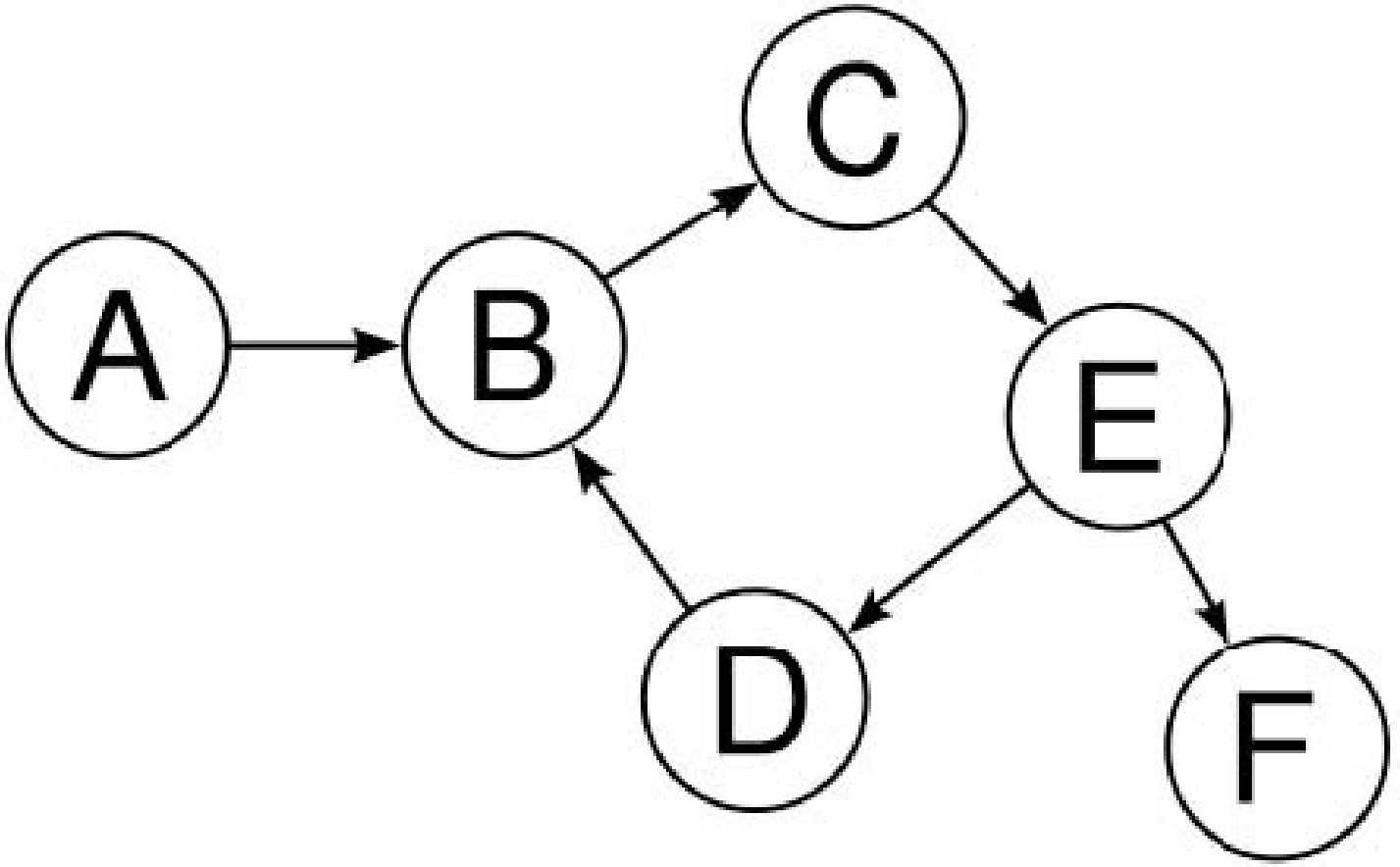
- Kanten haben Richtung, keine Zyklen



Allgemeiner Graph (Kanten ungerichtet, zyklisch)

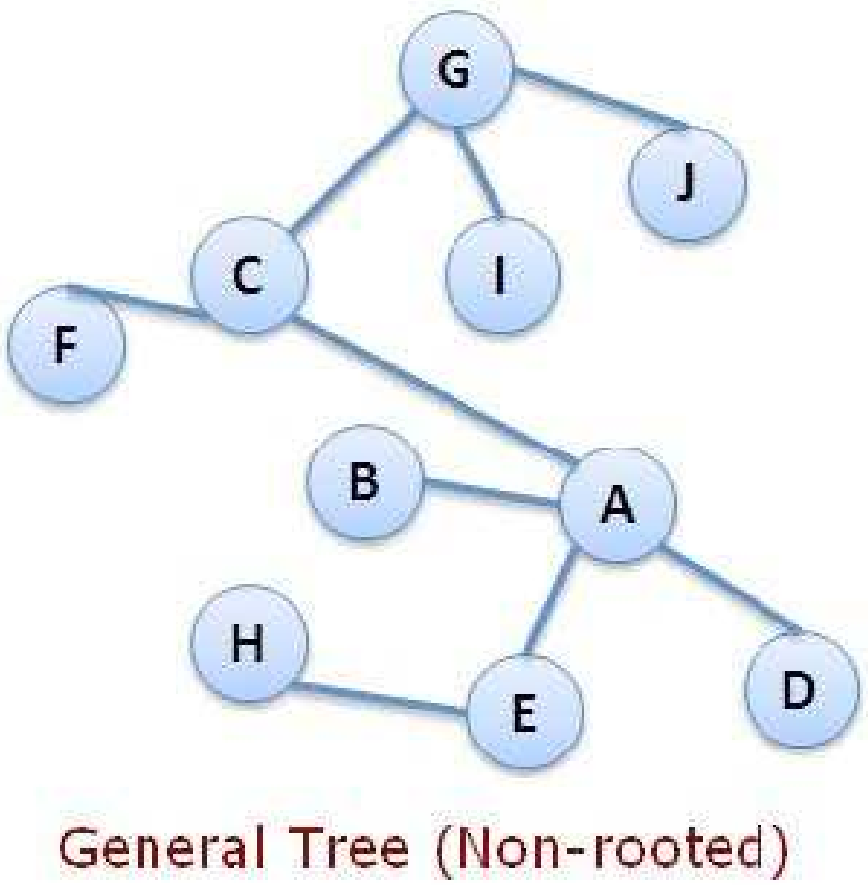


Gerichteter Graph (Kanten haben Richtung, zyklisch)



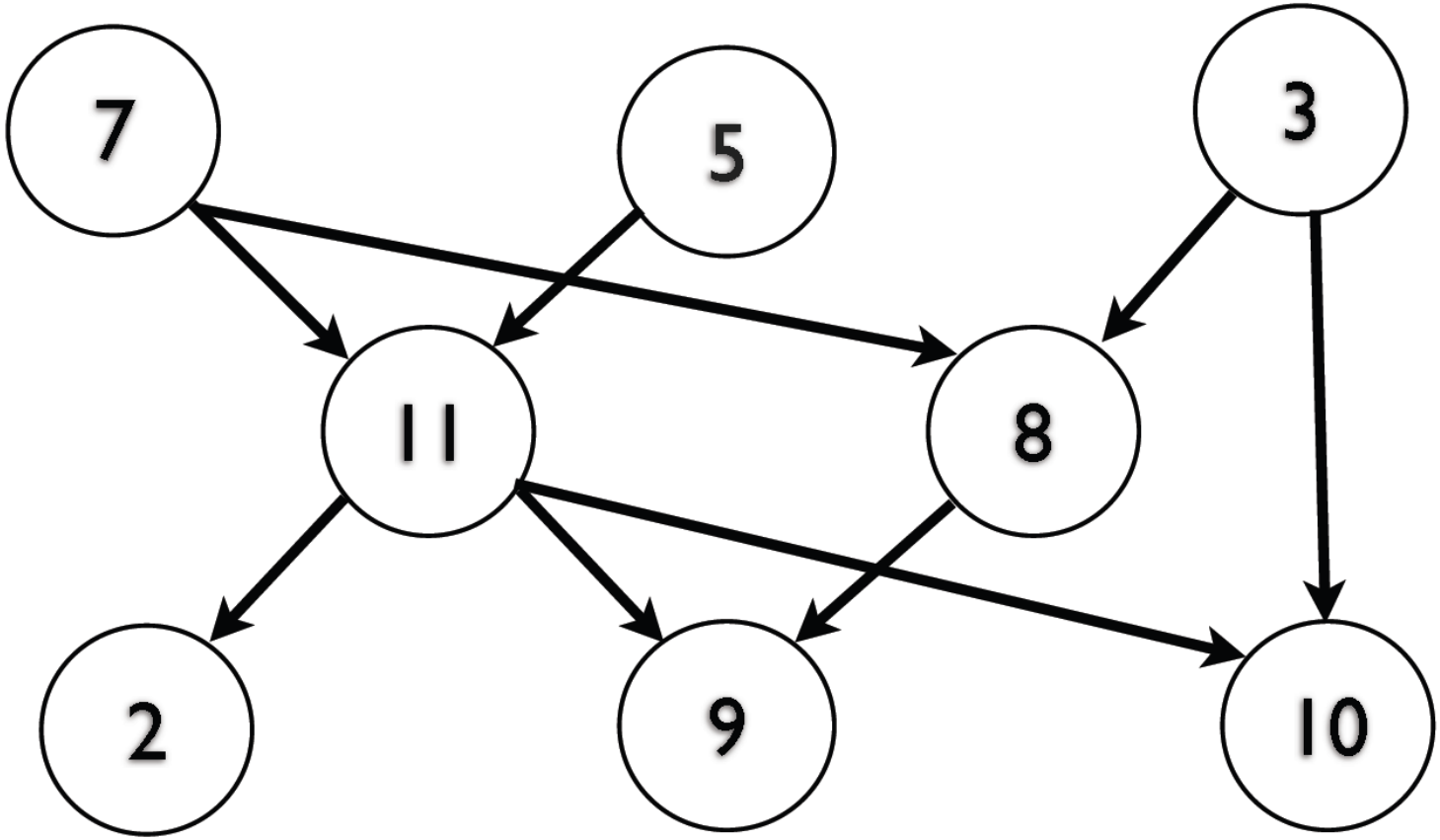
Azyklischer Graph

- **Kanten haben keine Richtung**
- **Es treten keine Zyklen auf**

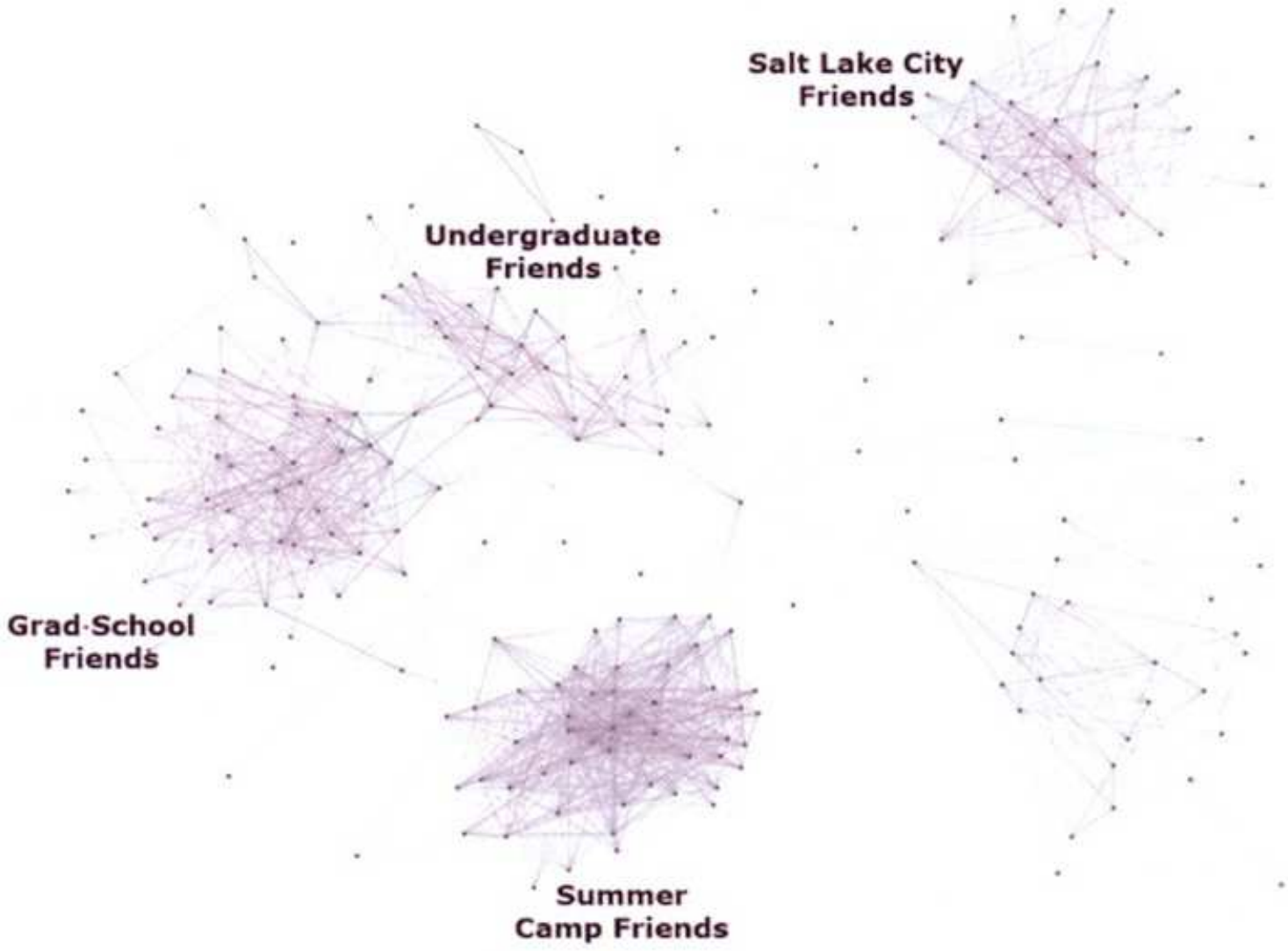


Gerichteter azyklischer Graph (keine Zyklen)

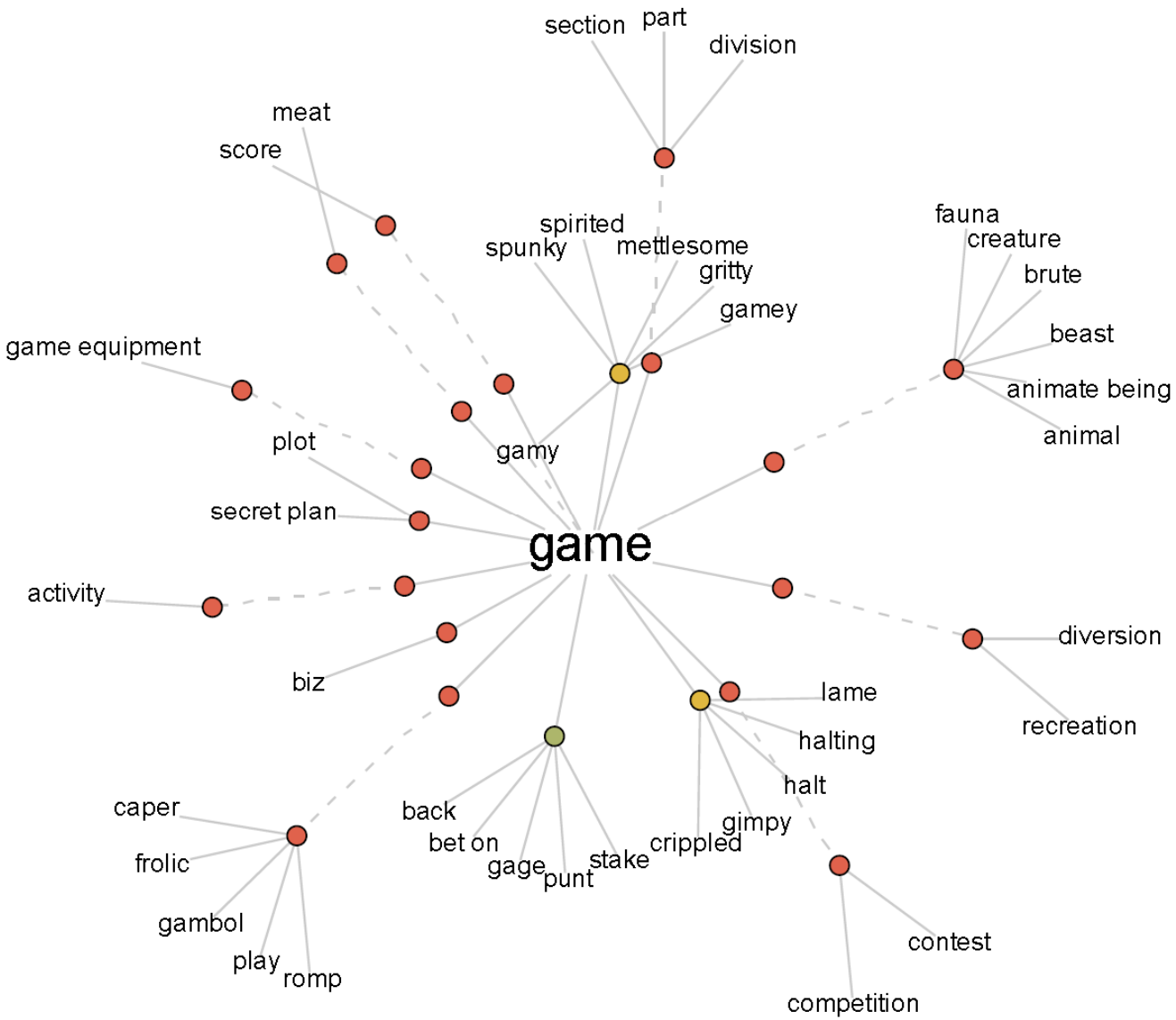
- **DAG = directed graphic acyclic graph**



Beispiel: Freunde auf Facebook



Visuelle Darstellung der Synonyme des Wortes „game“



Visuelle Darstellung der Synonyme des Wortes „game“

Collins

game [1]

n

1 amusement, distraction, diversion, entertainment, frolic, fun, jest, joke, lark, merriment, pastime, play, recreation, romp, sport
2 competition, contest, event, head-to-head, match, meeting, round, tournament
3 adventure, business, enterprise, line, occupation, plan, proceeding, scheme, undertaking
4 chase, prey, quarry, wild animals
5 (informal) design, device, plan, plot, ploy, scheme, stratagem, strategy, tactic, trick
6 make (a) game of deride, make a fool of, make a laughing stock, make a monkey of, make fun of, make sport of, mock, poke fun at, ridicule, send up (Brit. informal)

Antonyms

business, chore, duty, job, labour, toil, work

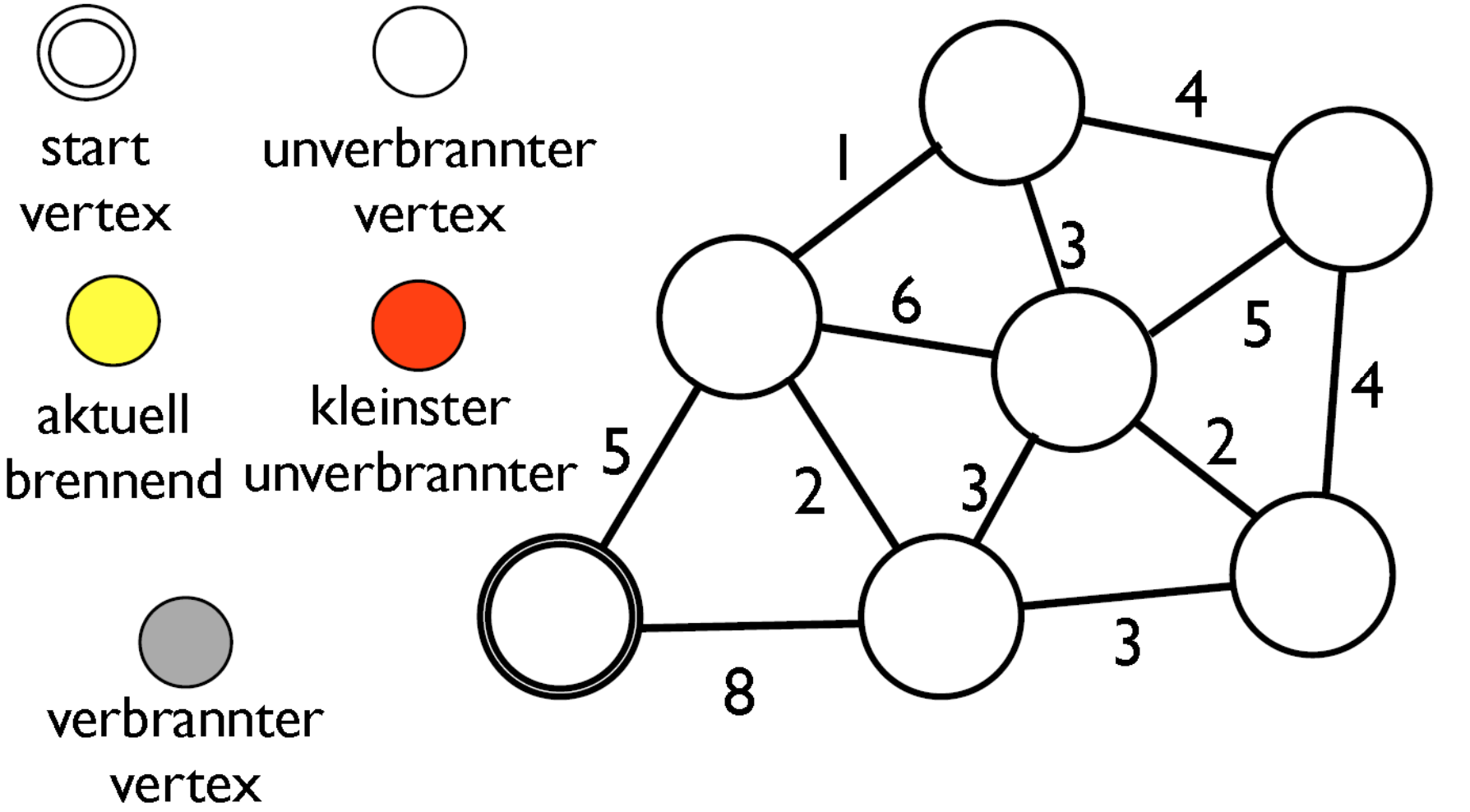
Dictionnaire anglais Collins English synonyme-Thesaurus

Implementierung:Kanten haben keine Attribute, daher nur implizit

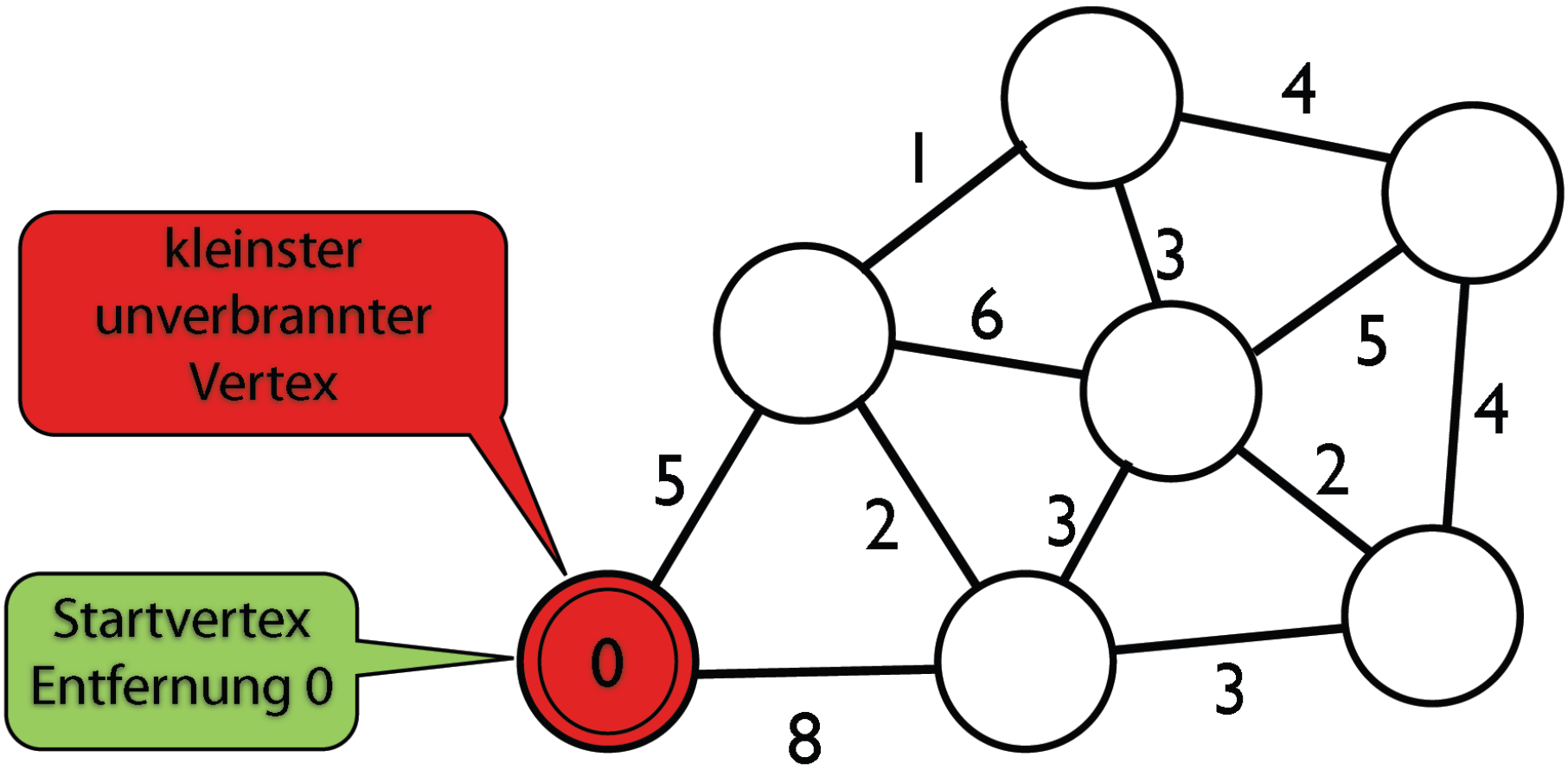
```
class Vertex {
   private static HashSet<Vertex> graphVertices; // merken der bearbeiteten Vertices
   private Vertex[] neighbor; // Referenzen auf Nachbarn
   private double attribut; // weitere Eigenschaften des Vertex
.....
   // Aufsuchen aller Vertices: Depth First Search (DFS), naechstes Semester: Breadth First Search
   private void alleVerticesRekursiv() {
      if( ! graphVertices.contains(this)) {
         graphVertices.add(this);
         this.doSomething(); // erledige die notwendigen Arbeiten
         for(int n; n < neigbor.length; n++) {
            Vertex v = neighbor[n];
            if(v != null) v.alleVerticesRekursiv();
         }
      }  // noch nicht bearbeitet
}
public void alleVerbundenenVertices() {
   graphVertices.clear();
   alleVerticesRekursiv();
   graphVertices.clear();
}
```

Gewichtete Graphen

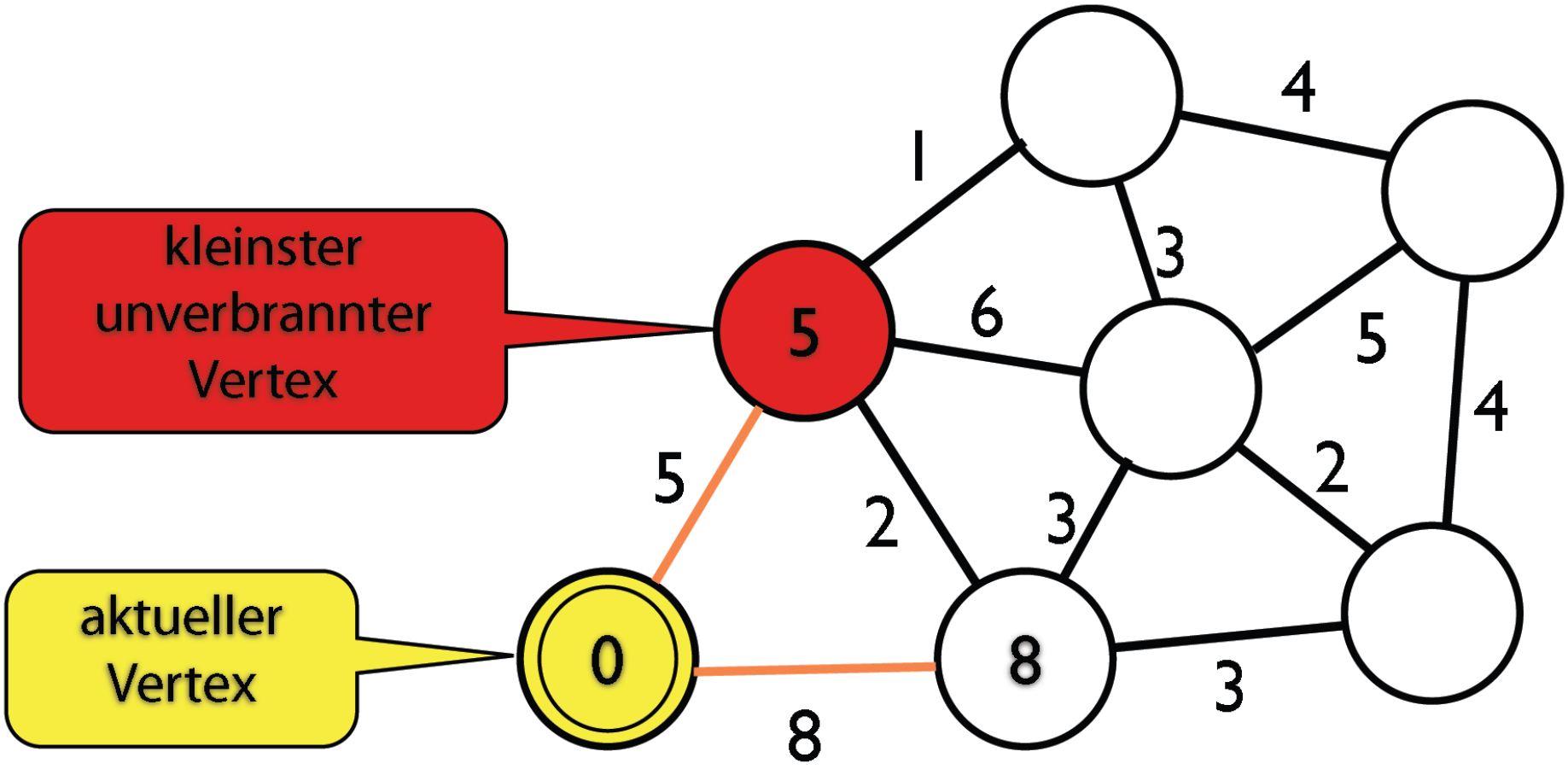
Entfernungen zwischen zwei verbundenen Vertices bilden Gewicht



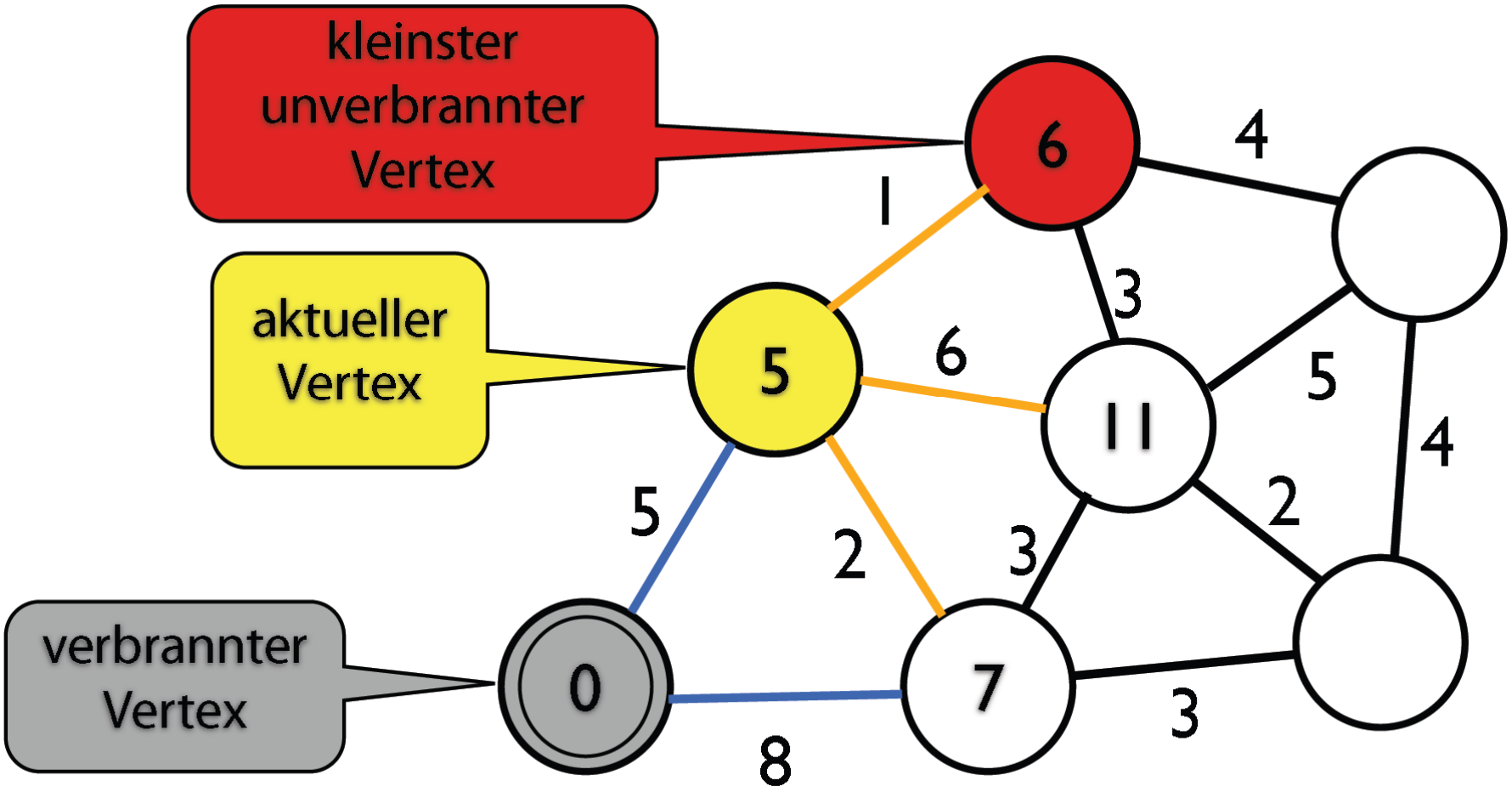
Aufgabe: Finde den kürzesten Weg zu jedem Vertex



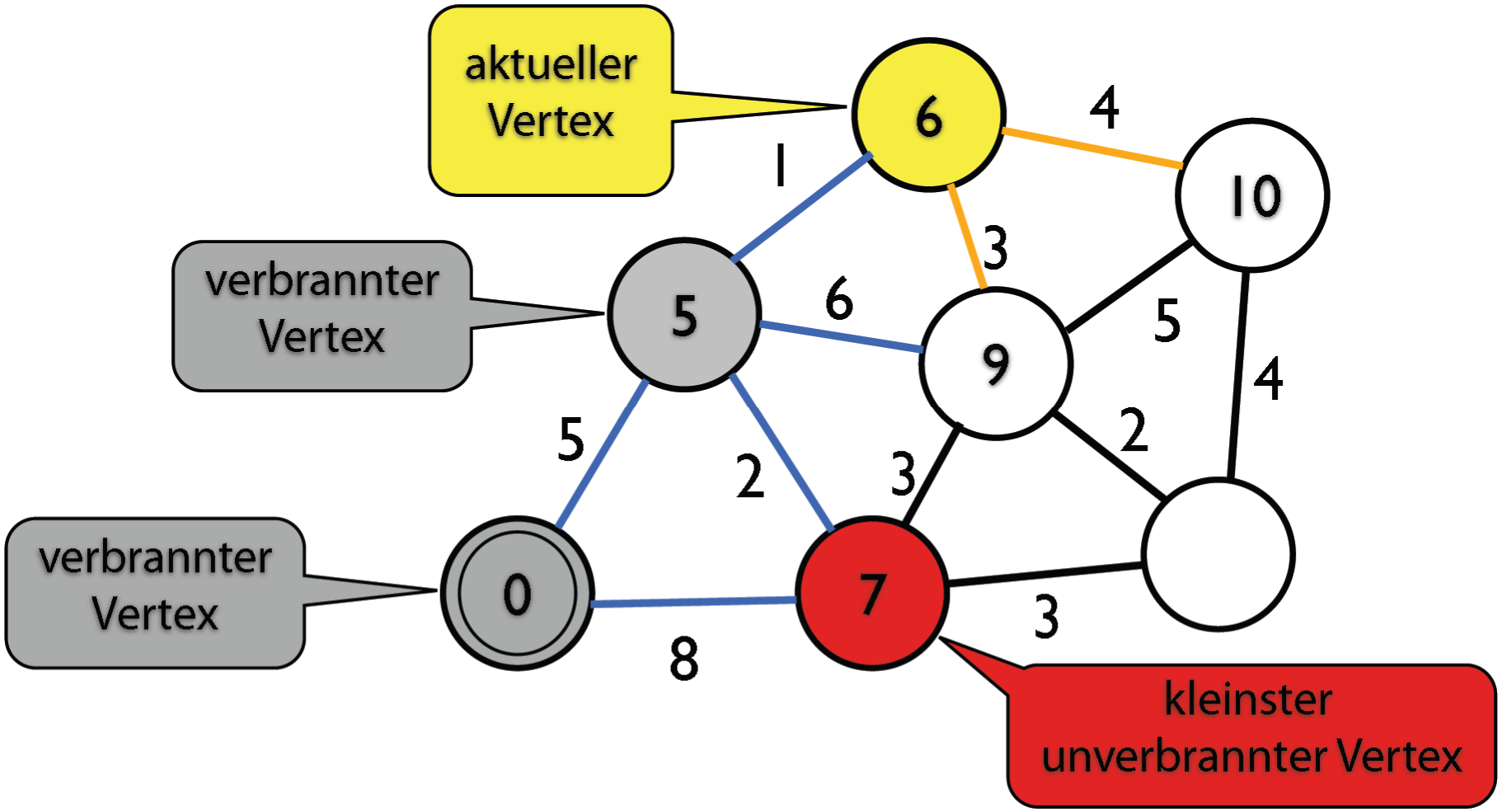
Aufgabe: Finde den kürzesten Weg zu jedem Vertex



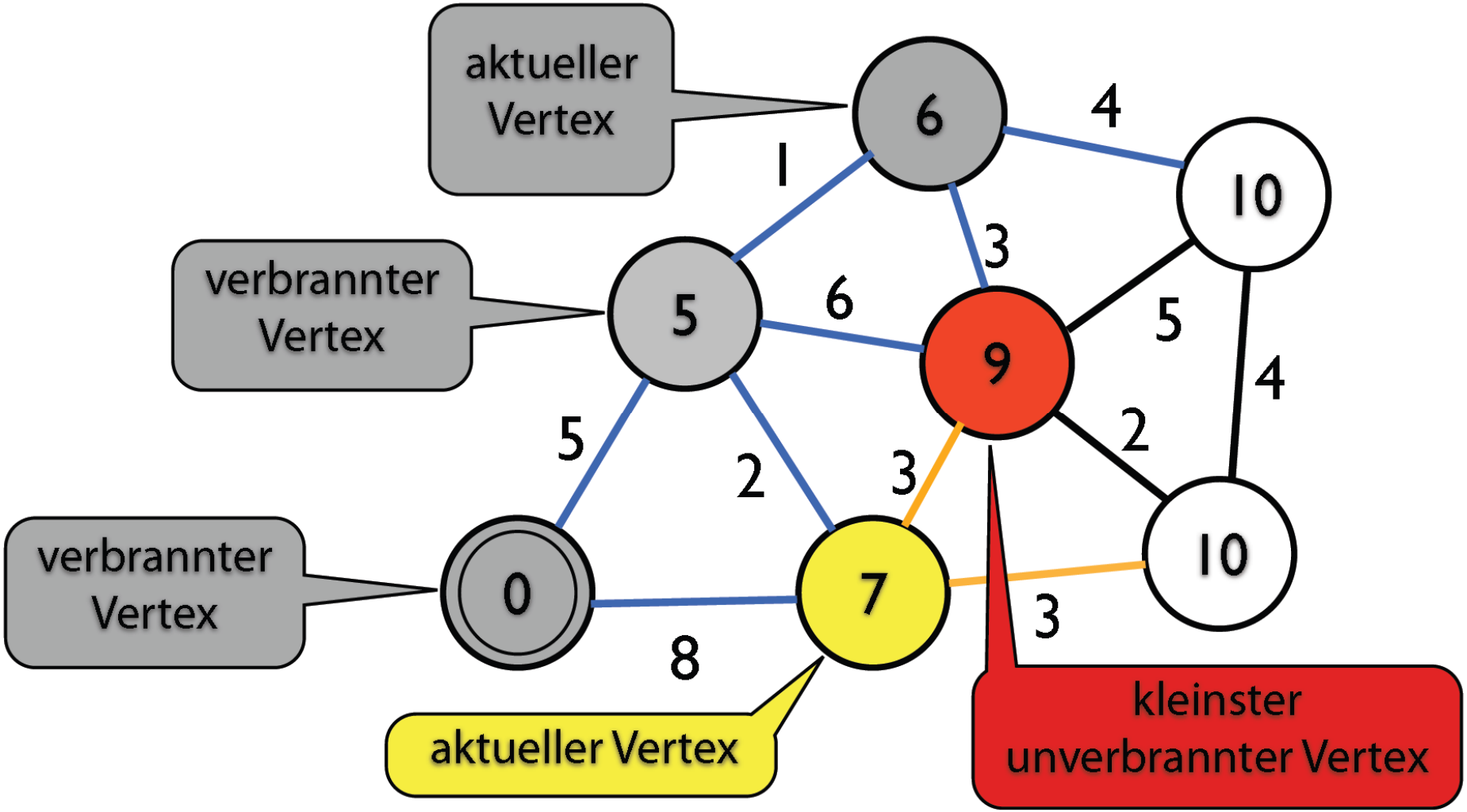
Aufgabe: Finde den kürzesten Weg zu jedem Vertex



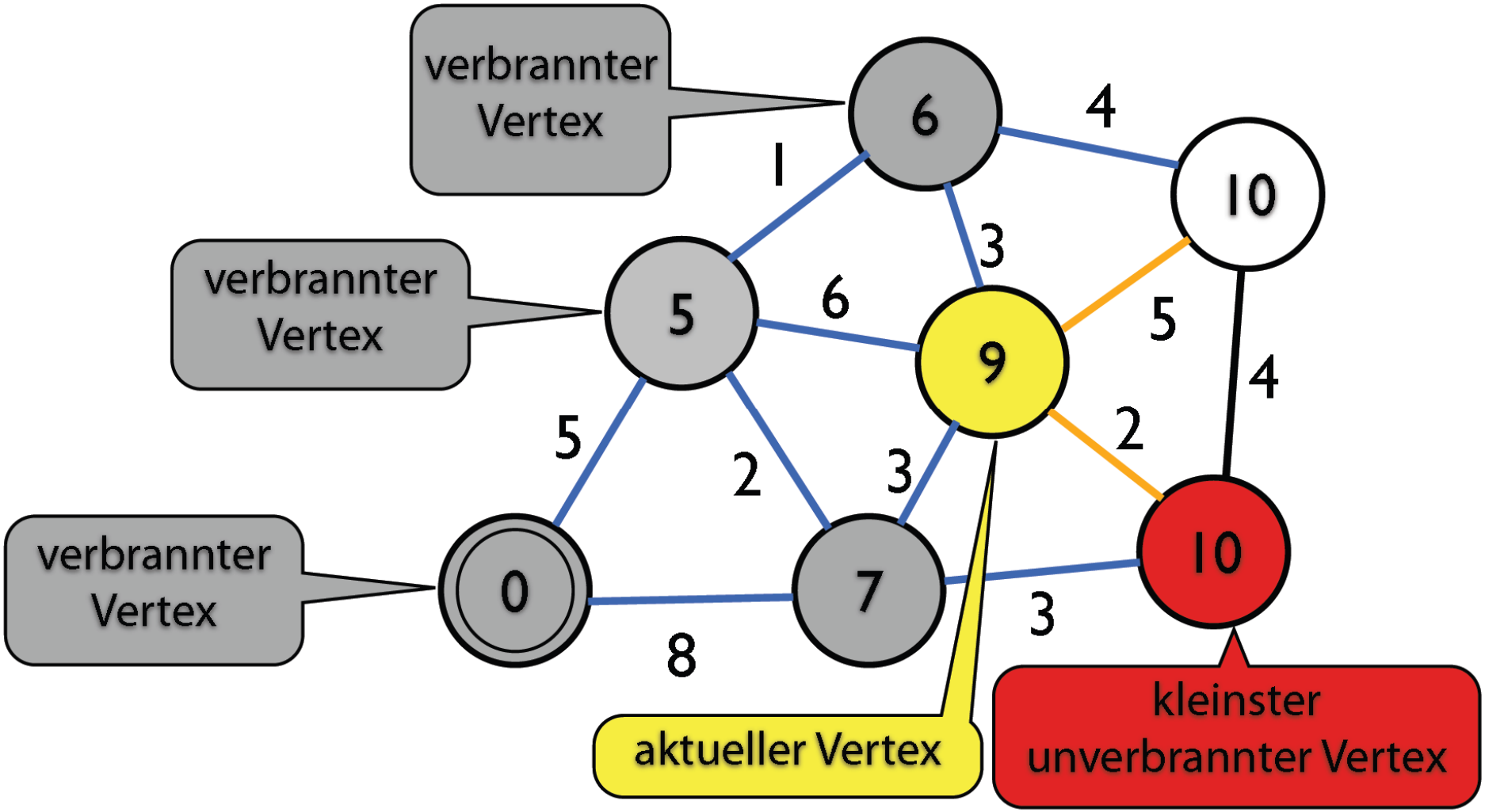
Aufgabe: Finde den kürzesten Weg zu jedem Vertex



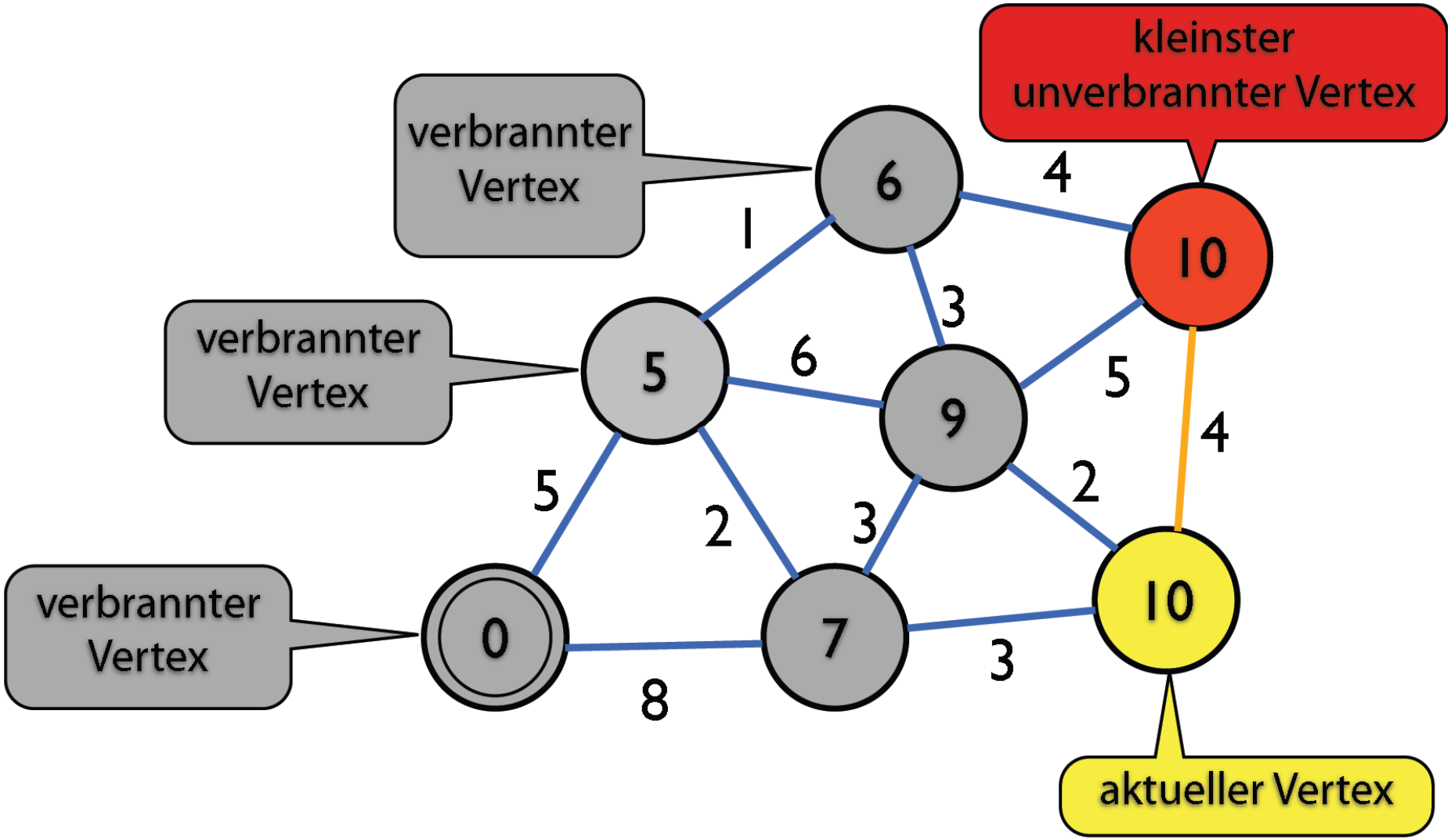
Aufgabe: Finde den kürzesten Weg zu jedem Vertex



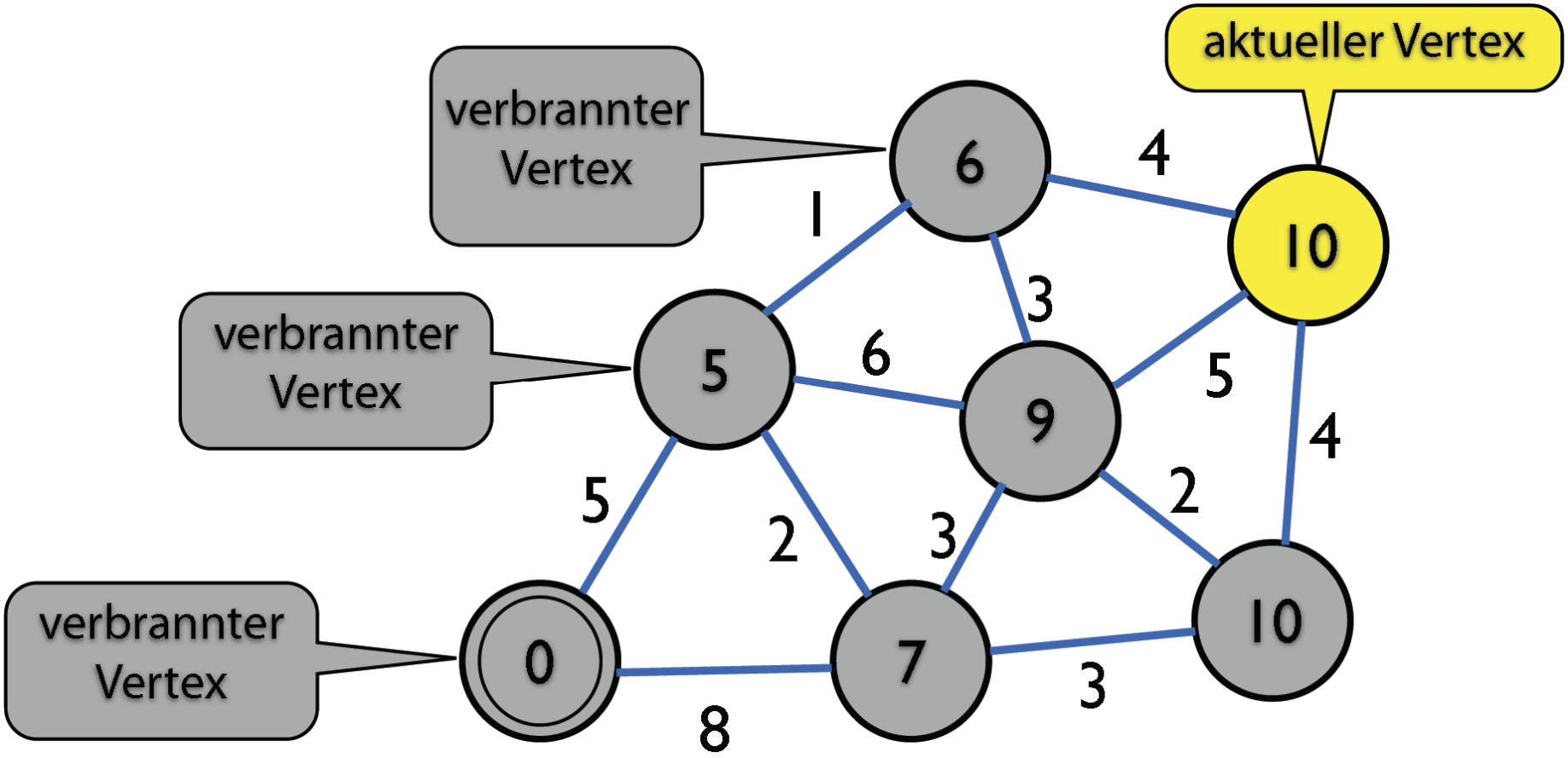
Aufgabe: Finde den kürzesten Weg zu jedem Vertex



Aufgabe: Finde den kürzesten Weg zu jedem Vertex

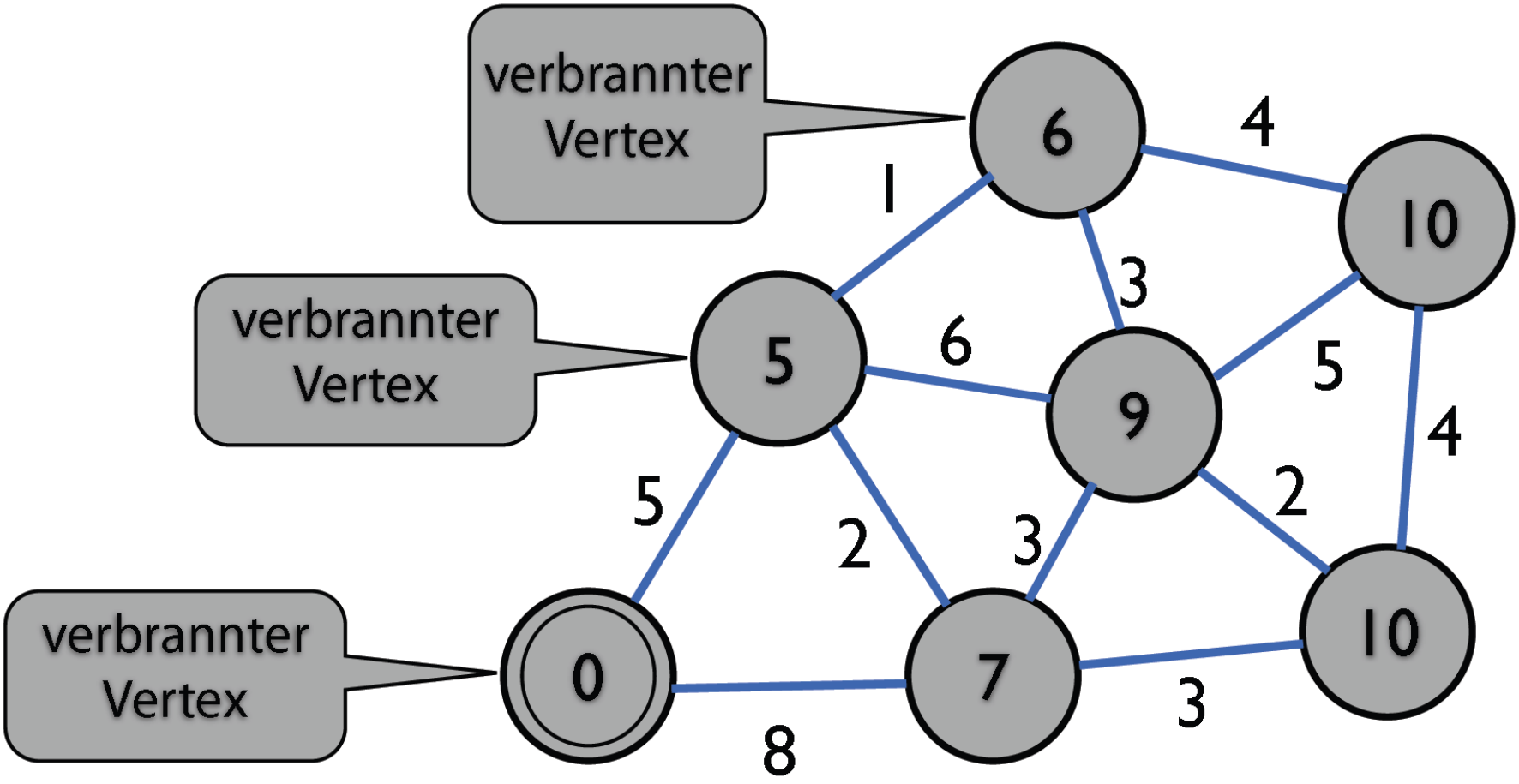


Aufgabe: Finde den kürzesten Weg zu jedem Vertex



Aufgabe: Finde den kürzesten Weg zu jedem Vertex

In jedem Vertex steht die Entfernung (vom Start aus)



Algorithmus:

Wiederhole bis kein unverbrannter Knoten vorhanden:

- Der aktuelle Knoten ist der “unverbrannte” mit der geringsten Entfernung
- Markiere den aktuellen Knoten als “verbrannt”
- Besuche alle Nachbarn des aktuellen Knotens
- Falls der Nachbar n eine größere Entfernung als die Entfernung des „aktuellen Knotens + Länge“ der Verbindungskante hat, ersetze mit dem niedrigeren Wert und vermerke den aktuellen Knoten als Teil des Pfads

Algorithmus:Edsger W. Dijkstra, 1930 - 2002

- Für jeden Vertex v im Graphen G
  - Markiere v als unverbrannt
  - Setze die Enfernung E(v) auf unendlich
  - Setze Elter(v) als undefiniert
- Setze E(Startvertex) = 0
- Solange es noch unverbrannte Vertices gibt:
  - der unverbrannte Vertex mit der kleinsten Entfernung heist u
  - Markiere u als verbrannt
  - Für jeden Nachbarn n von u
    - Falls E(u) + Abstand(u, n) < E(n)
      - Setze E(n) = E(u) + Abstand(u, n)
      - Setze Elter(n) = u
- Ende

Zusammenfassung

- Graphen sind eine wichtige Datenstruktur.
- Graphen bestehen aus Vertices (Knoten) und Kanten
- Vier Arten von Graphen aus zwei Eigenschaften:
  - zyklisch oder nicht
  - gerichtet oder nicht

Beispiel für einen Algorithmus auf Graphen:

- Dijkstra:
  - berechnet die kürzeste Entfernung und den zugehörigen Pfad zu jedem erreichbaren Vertex von einem Startvertex aus.

Alternative Algorithmen: A* (a star)

- Der A*-Algorithmus (auch A*-Suche) gehört zur Klasse der in-formierten Suchalgorithmen. Er dient in der Informatik der Berechnung eines kürzesten Pfades zwischen zwei Knoten in einem Graphen mit positiven Kantengewichten. Er wurde das erste Mal 1968 von Peter Hart, Nils J. Nilsson und Bertram Raphael beschrieben. Der Algorithmus gilt als Verallgemeinerung und Erweiterung des Dijkstra-Algorithmus, in vielen Fällen kann aber umgekehrt A* auch auf Dijkstra reduziert werden.
- Im Gegensatz zu uninformierten Suchalgorithmen verwendet der A*-Algorithmus eine Schätzfunktion (Heuristik), um zielgerichtet zu suchen und damit die Laufzeit zu verringern. Der Algorithmus ist optimal. Das heißt, dass immer die optimale Lösung gefunden wird, falls eine existiert.

Alternative Algorithmen: Prim

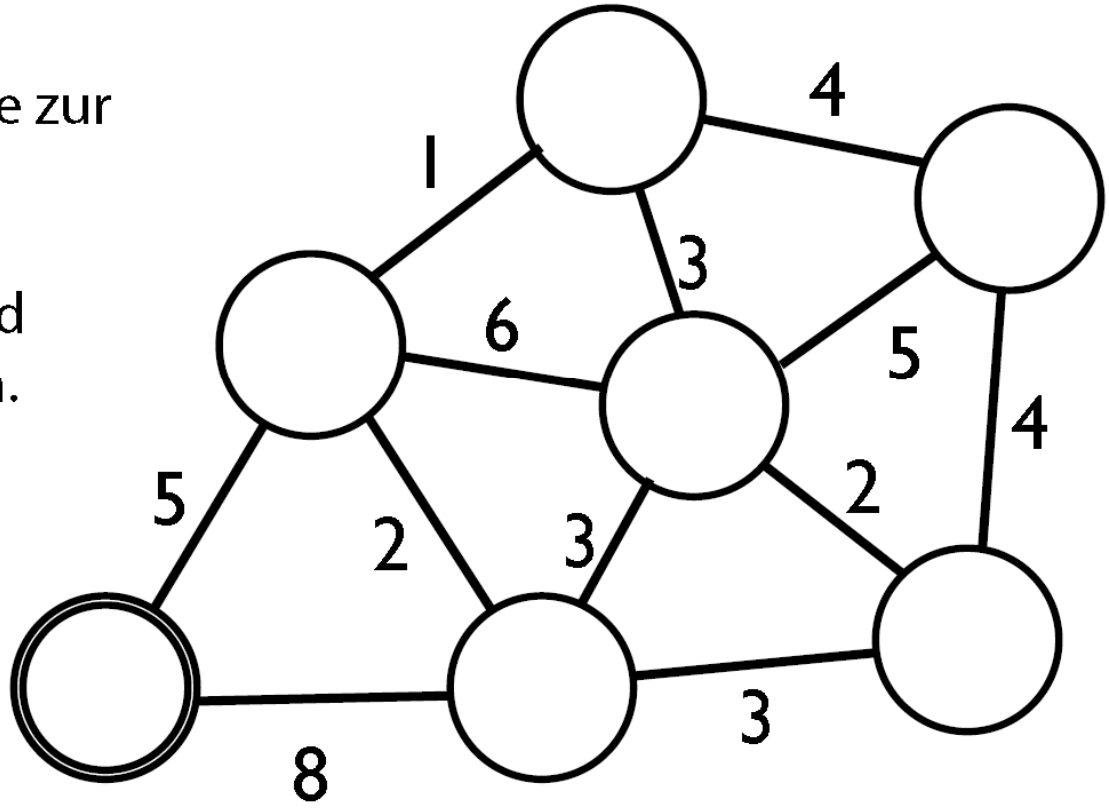
Der Algorithmus beginnt mit einem trivialen Graphen T, der aus einem beliebigen Knoten des gegebenen Graphen besteht. In jedem Schritt wird nun eine Kante mit minimalem Gewicht gesucht, die einen weiteren Knoten mit T verbindet. Diese und der entsprechende Knoten werden zu T hinzugefügt. Das Ganze wird solange wiederholt, bis alle Knoten in T vorhanden sind; dann ist T ein minimaler Spannbaum:

- Wähle einen beliebigen Knoten als Startgraph T.
- Solange T noch nicht alle Knoten enthält:
  - Wähle eine Kante e minimalen Gewichts aus, die einen noch nicht in T enthaltenen Knoten v mit T verbindet.
  - Füge e und v dem Graphen T hinzu.

Aufgabe 25:

Implementieren Sie Klassen zur Repräsentation von Vertices und Kanten.
Bauen Sie aus diesen Komponenten den Graphen aus dem Beispiel.

Die Klassen sollen alle zur Durchführung des Dijkstra-Algorithmus nötigen Attribute und Methoden aufweisen.

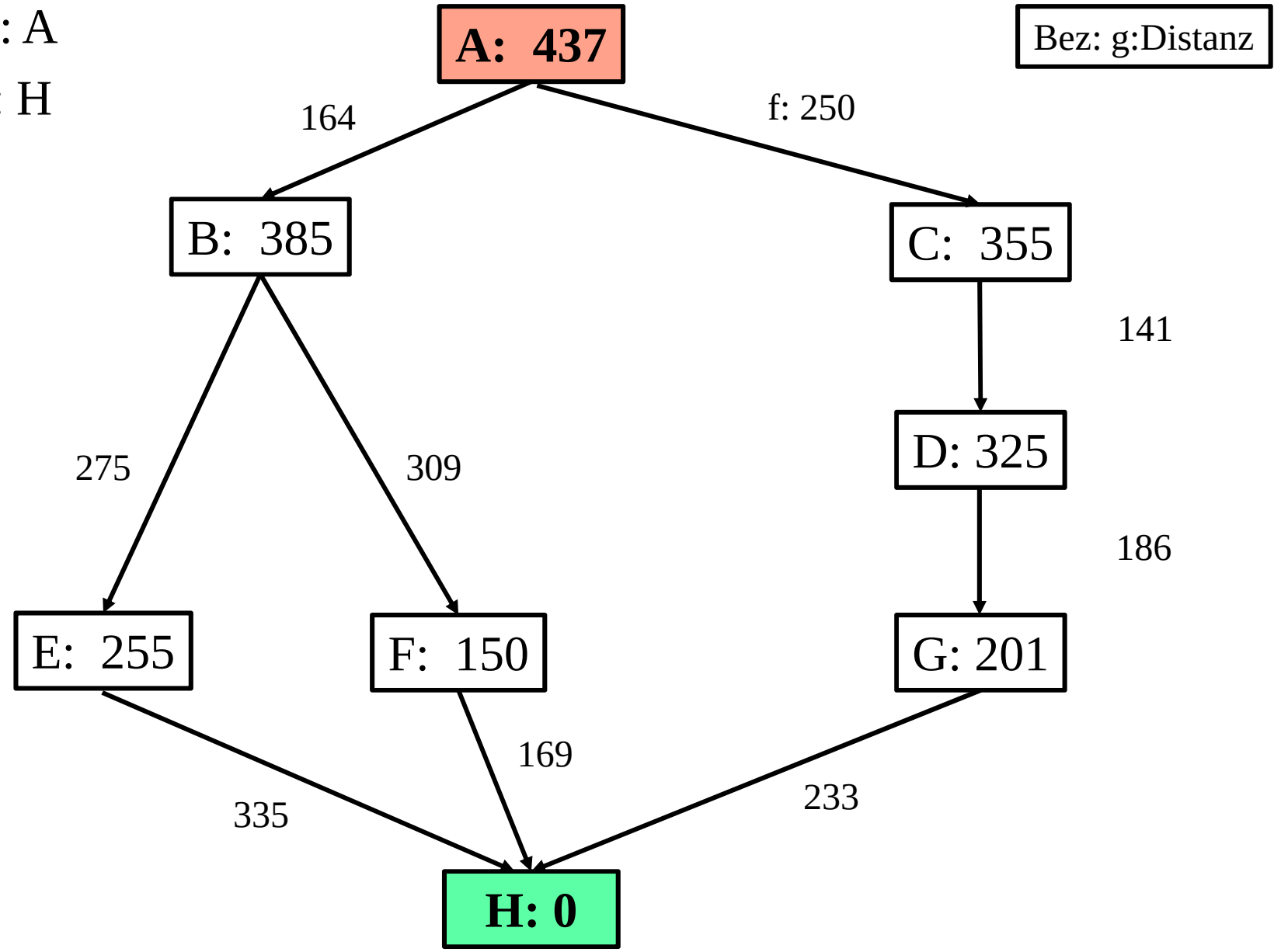


Aufgabe 26:

Implementieren Sie den Dijkstra-Algorithmus.
Testen Sie Ihre Implementierung mit dem Graphen aus Aufgabe 25.

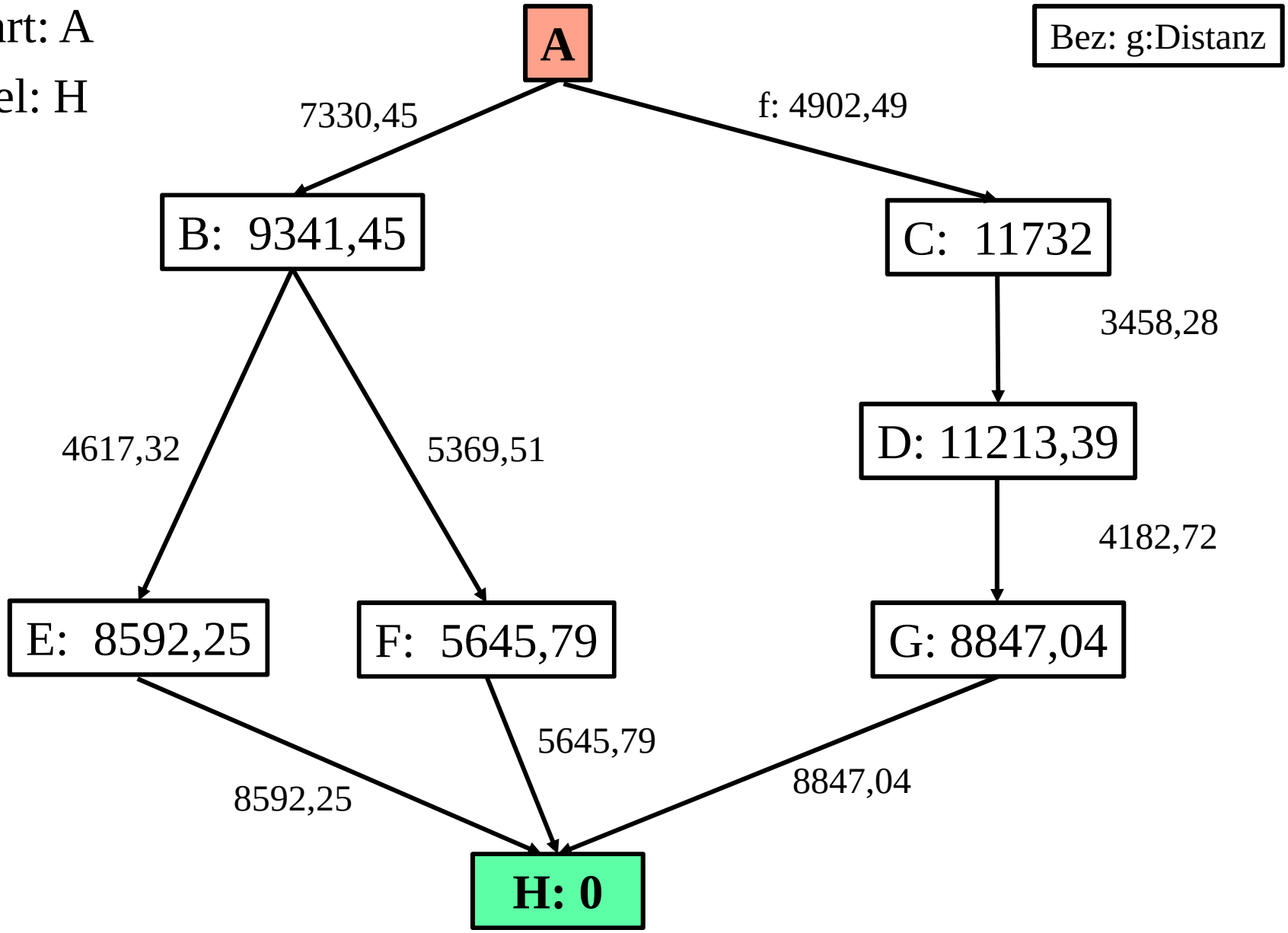
Start: A

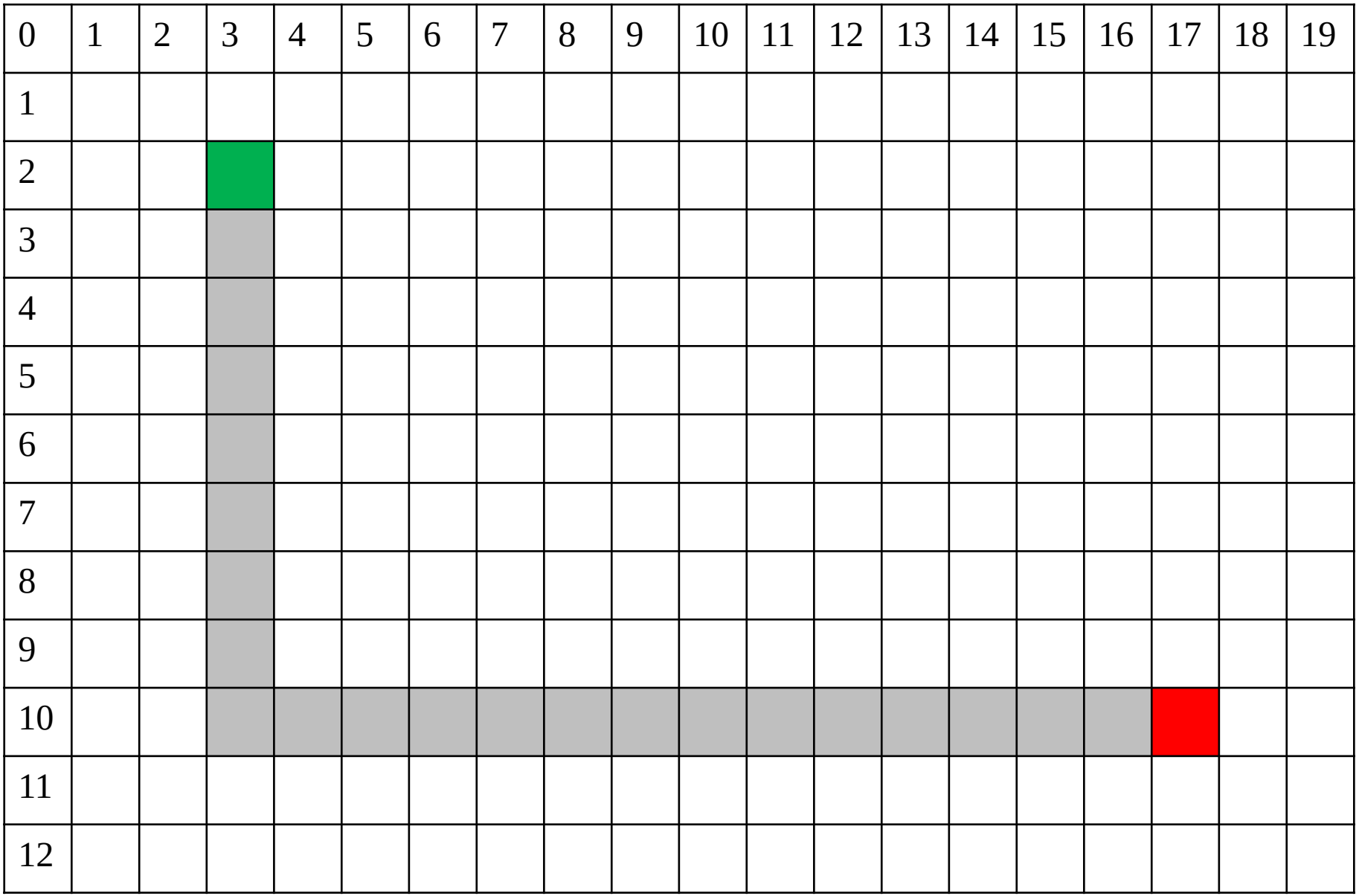
Ziel: H



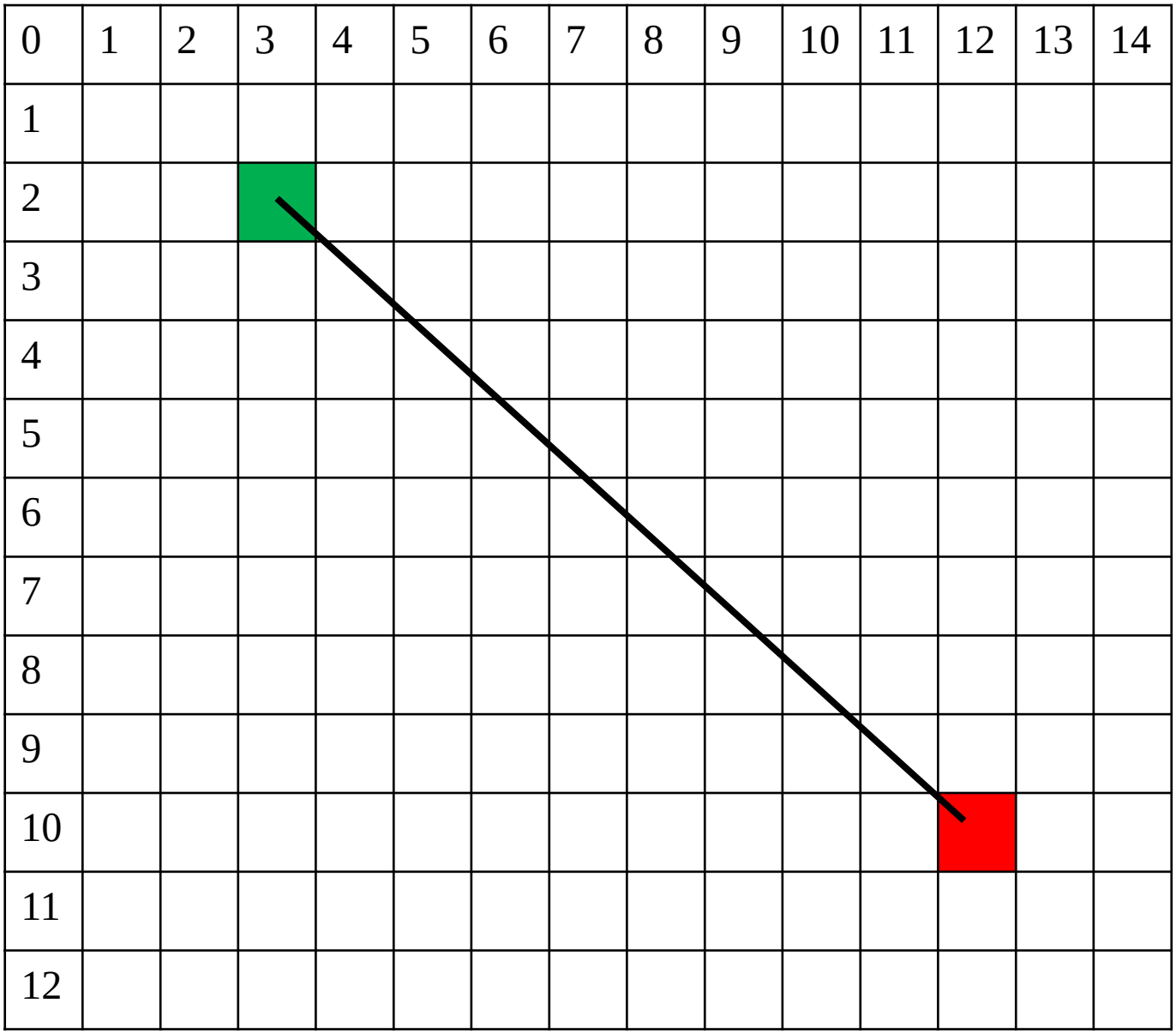
Start: A

Ziel: H




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
1
2
3
4
5
6
7
8
9
10
11
12

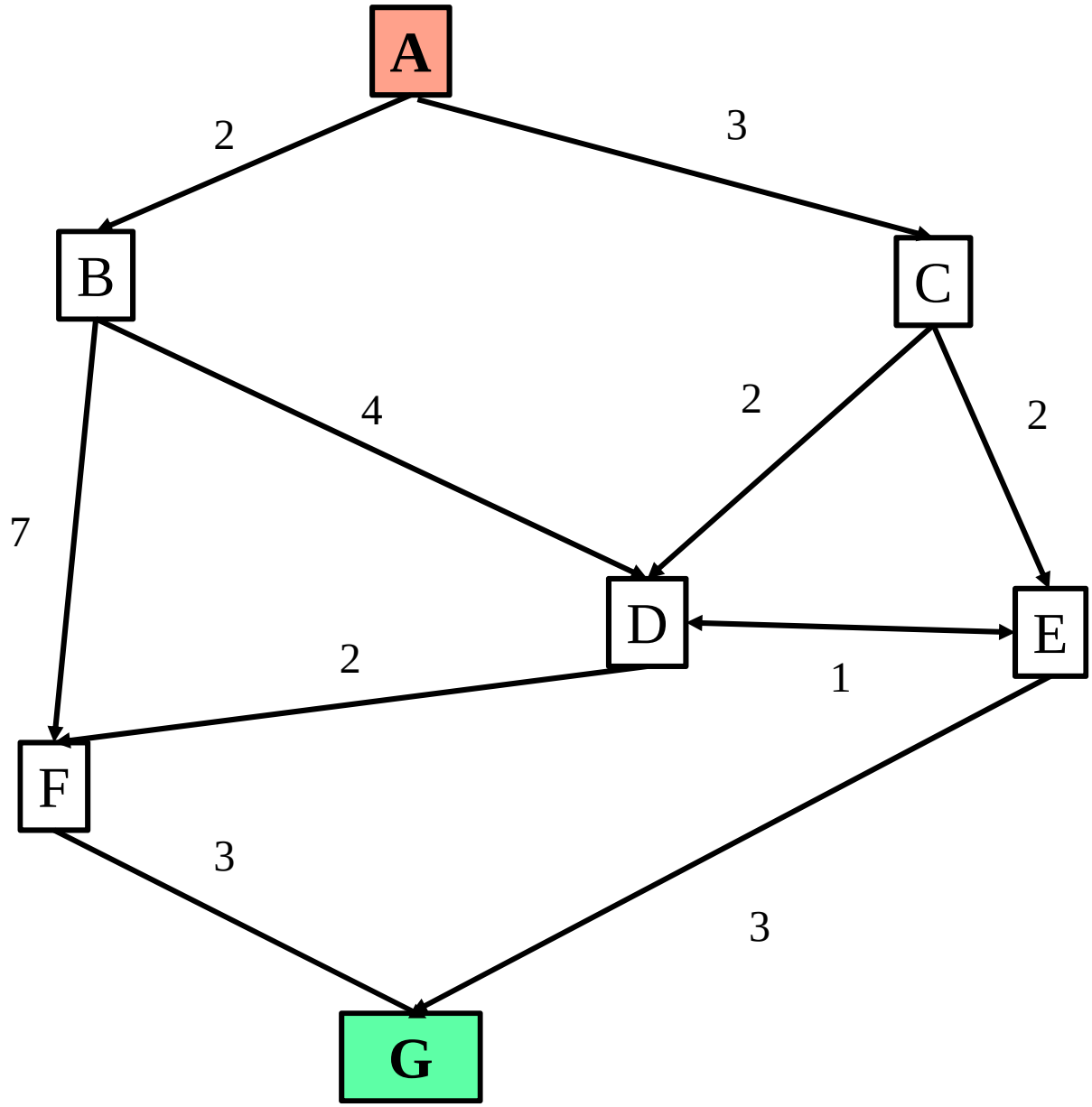
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1
2
3
4
5
6
7
8
9
10
11
12

Beispiel 1

Start: A

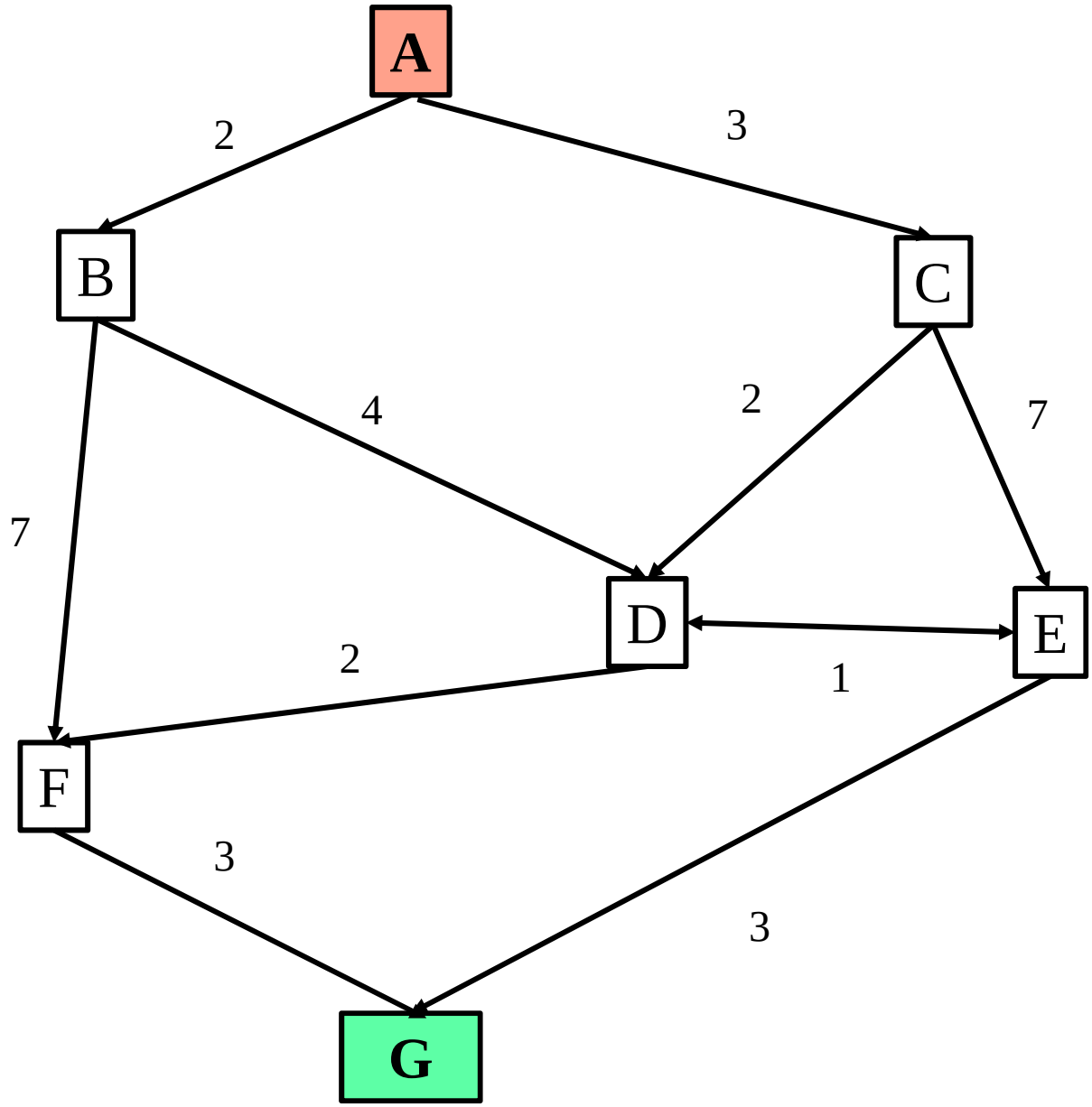
Ziel: G



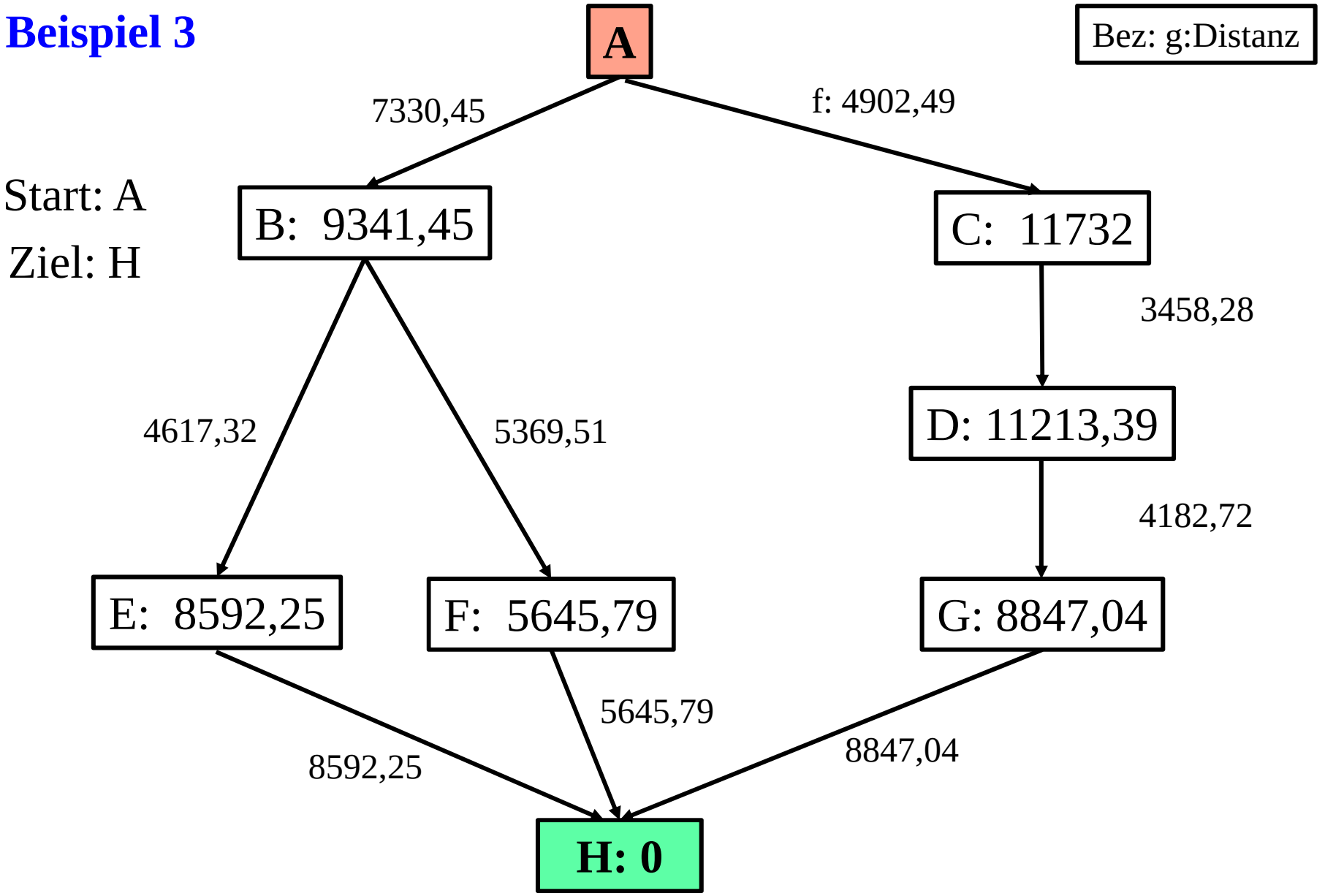
Beispiel 2

Start: A

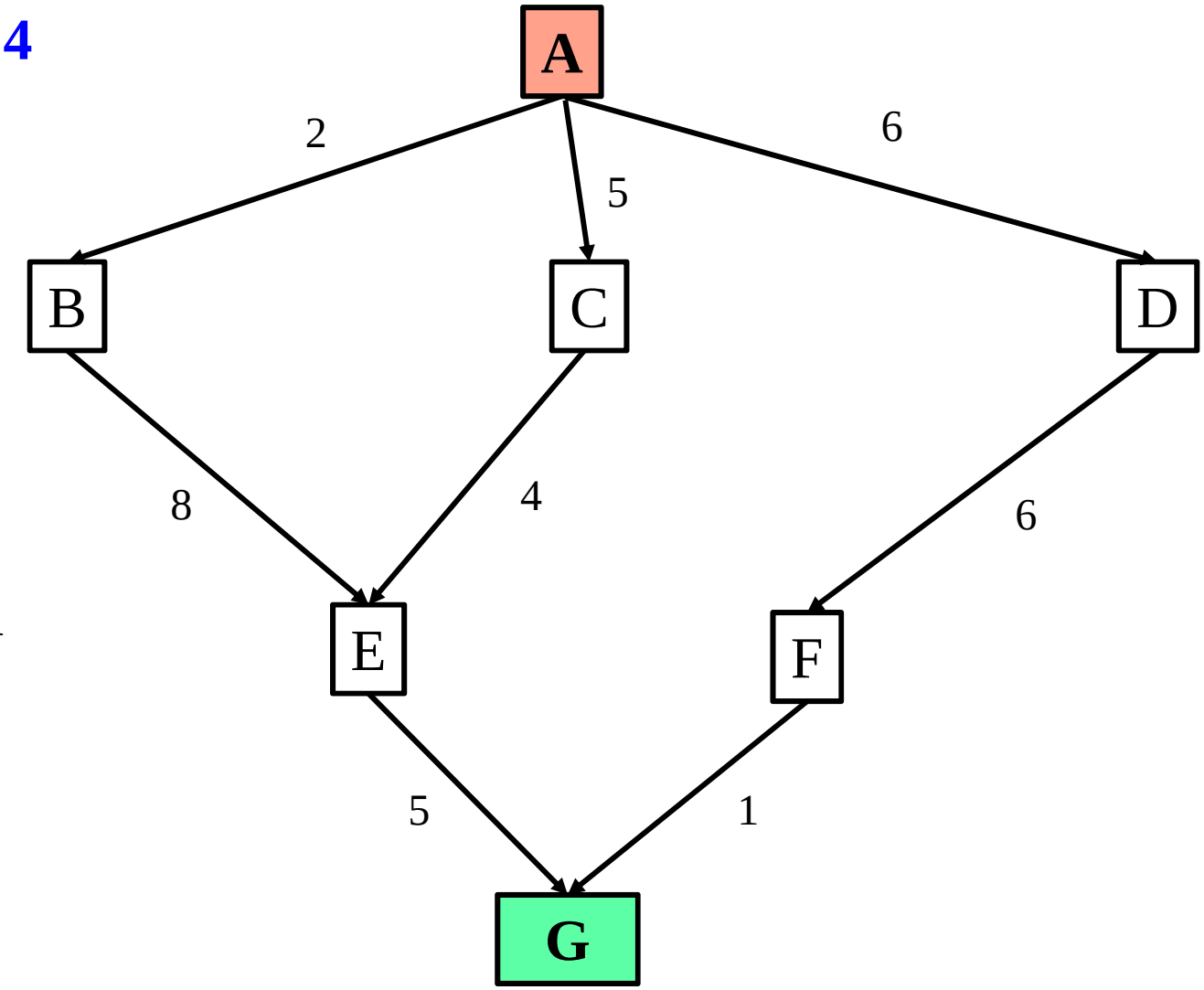
Ziel: G



Beispiel 3


A
Bez: g:Distanz
7330,45
f: 4902,49
Start: A
Ziel: H
B: 9341,45
C: 11732
3458,28
4617,32
5369,51
D: 11213,39
4182,72
E: 8592,25
F: 5645,79
G: 8847,04
5645,79
8592,25
8847,04
H: 0

Beispiel 4



Start: A

Ziel: G