Termine Abnahme Hauptprojekt

Die Abnahmen des Hauptprojekts finden statt:

Dienstag 12.02.19 und *Mittwoch 13.02.19*

in Raum L109 an der Oettingenstr.67

Die Prüfungen dauern jeweils 90min, bitte Foliensatz 16.5 "'Abnahme Gruppenprojekte"' beachten!

	Von	Bis
1.Prüfung	09:00	10:30
2.Prüfung	10:30	12:00
3.Prüfung	13:00	14:30
4.Prüfung	14:30	16:00
5.Prüfung	16:15	17:45

Gruppenbetreuer bis inklusive 4.2.19 mitteilen, welche Termine zwingend ausgeschlossen werden müssen; dabei ist der Grund anzugeben (z.B. Name der Klausur).

Mögliche Erweiterungen

Erweiterungen sind keine Pflicht zum Bestehen, erleichtern aber die Rechtfertigung sehr guter Noten.

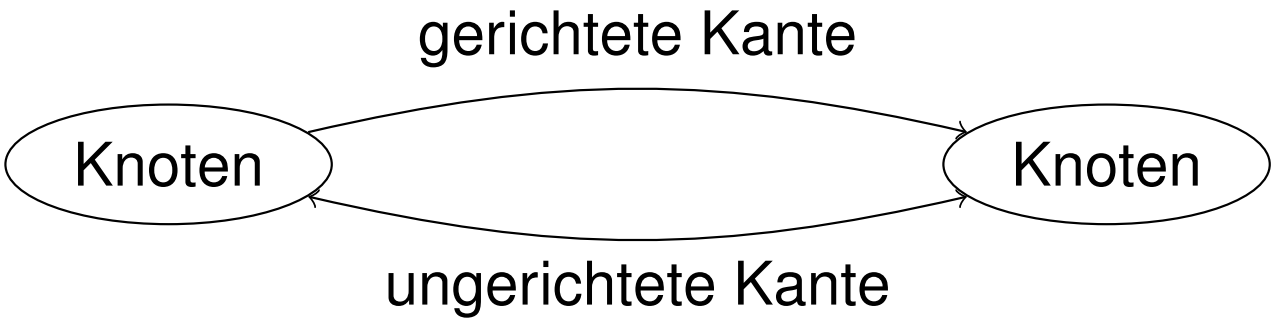
Mögliche Ideen:

- Rückspulen der Simulation
- gekrümmte Rollwege erlauben
- KFZ-Vorfeldverkehr einbeziehen
- eigener Flughafen mit interessanten Features JSON
- Verbesserte Wegsuche, welche Distanzen berücksichtigt

Sie dürfen sich auch gerne selbst sinnvolle Erweiterungen ausdenken, bitte mit dem Gruppenbetreuer absprechen!

Breitensuche

Graphen



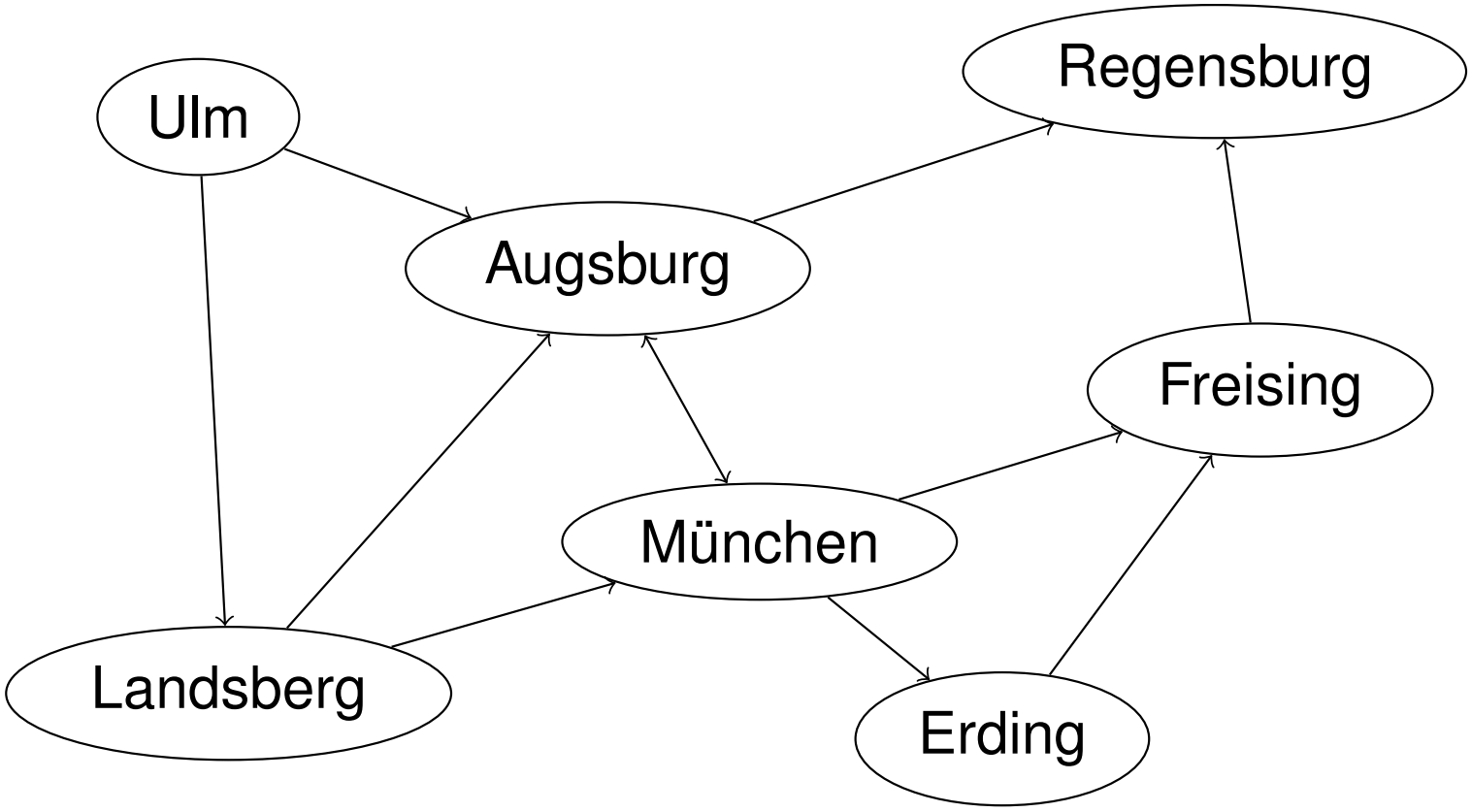
Graphen sind eine Abstraktion von sehr unterschiedlichen Situationen, beispielsweise

- Knoten sind Felder bei Minesweeper, Kanten gehen von jedem Feld zu allen Nachbarfeldern
- Knoten sind Kombination auf Zeit und Ort auf einer Landkarte; Kanten gehen zu den Orten, die in einem Schritt erreichbar sind, kombiniert mit einem Zeitschritt höher

Grundlegende Algorithmen sind in allen Fällen gleich

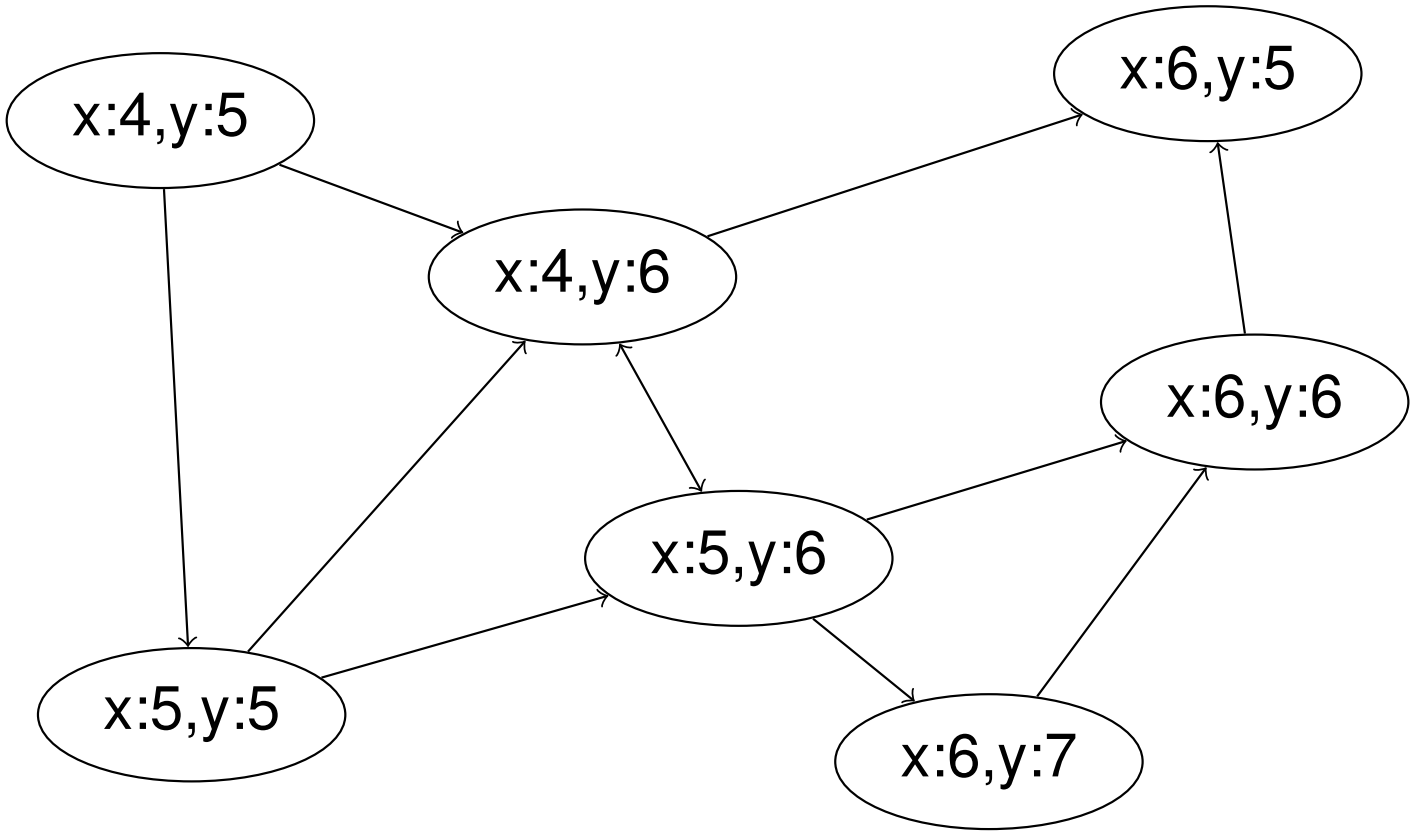
Graphenbeispiele

Graphalgorithmen sind unabhängig von der tatsächlichen Bedeutung des Graphens



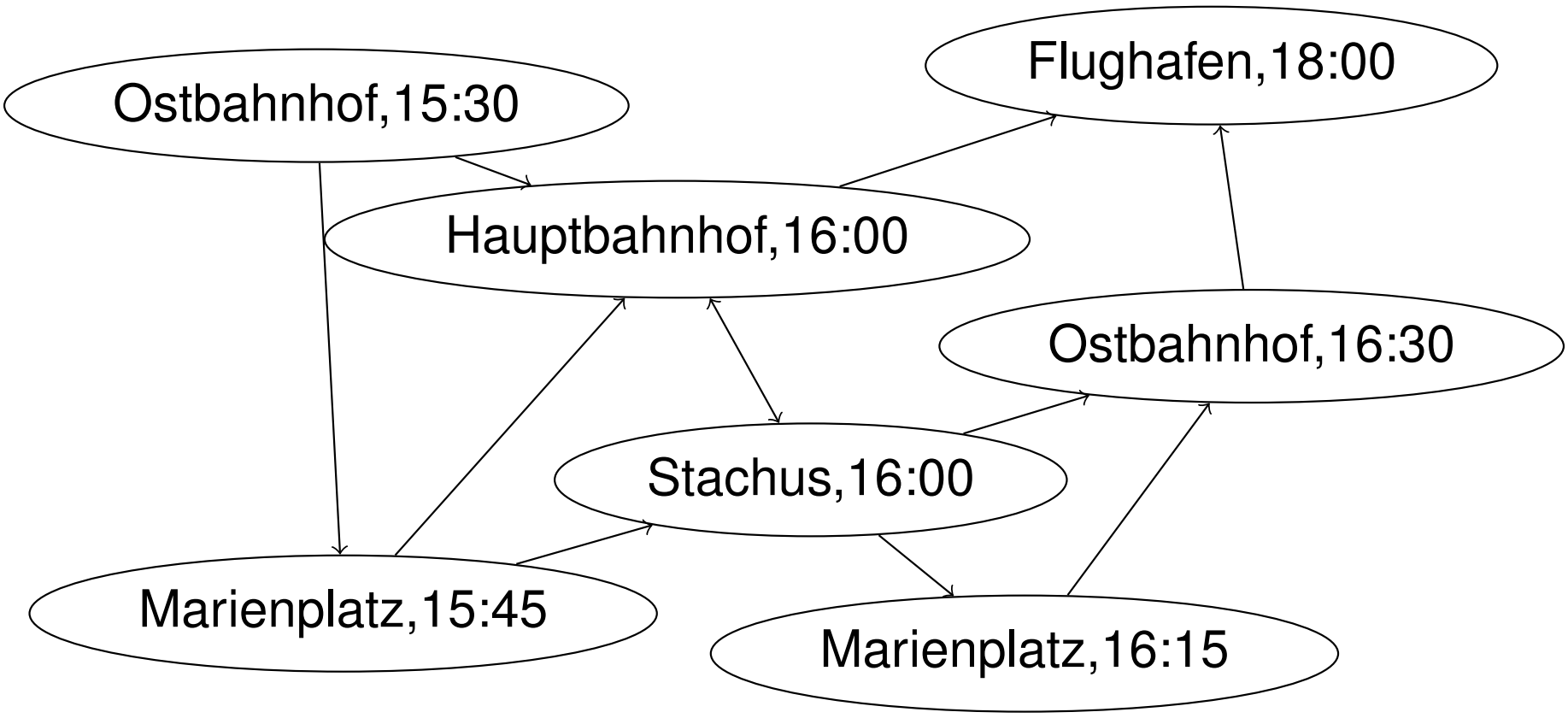
Graphenbeispiele

Graphalgorithmen sind unabhängig von der tatsächlichen Bedeutung des Graphens



Graphenbeispiele

Graphalgorithmen sind unabhängig von der tatsächlichen Bedeutung des Graphens



Suche

Suche deckt eine Reihe von Problemstellungen ab

- Wie deckt man Felder bei Minesweeper auf, die in einem zusammenhängenden Bereich ohne Minen liegen?
- Wie färbt man den einfarbigen Bereich eines Bildes in einer neuen Farbe?
- Wie findet man irgendeinen Weg von einem Start zu einem Ziel?

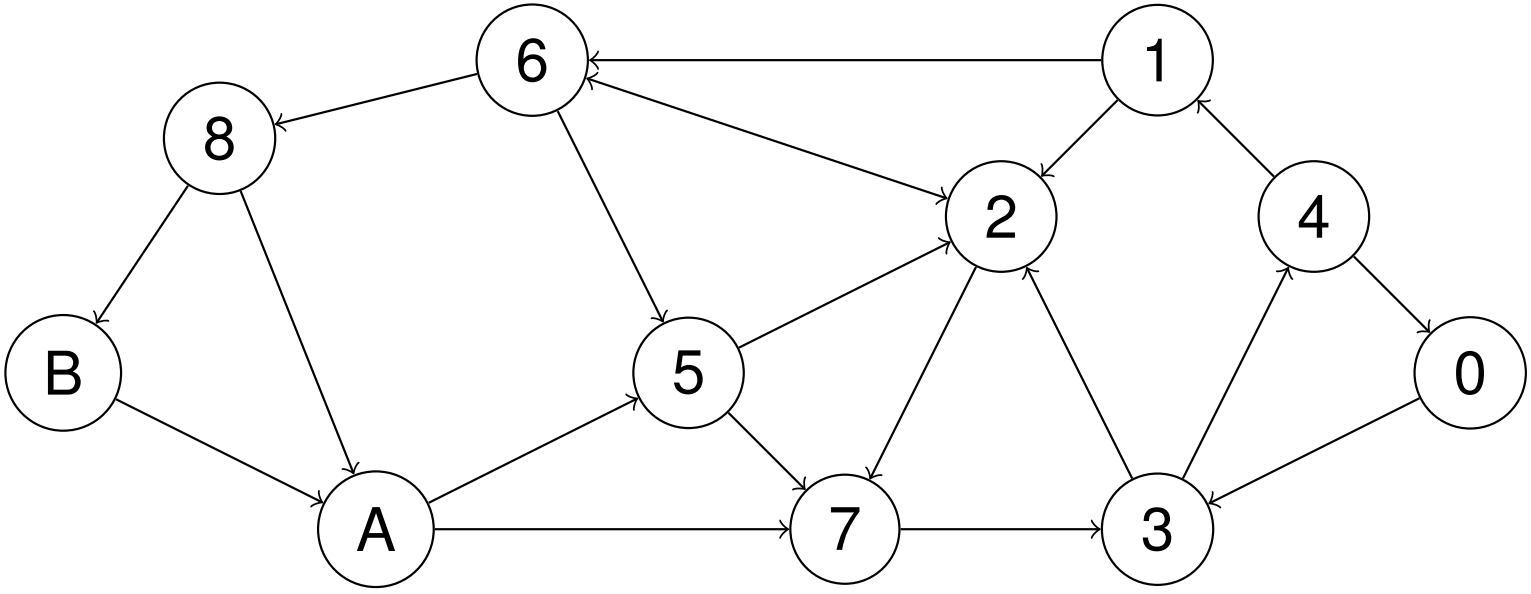
Arbeitsprinzip: Alle erreichbaren Punkte erreichen, dabei speichern, wo man schon war

Will man den Weg dann auch haben: Zusätzlich speichern, von wo man zu einem Knoten hergekommen gekommen ist

Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

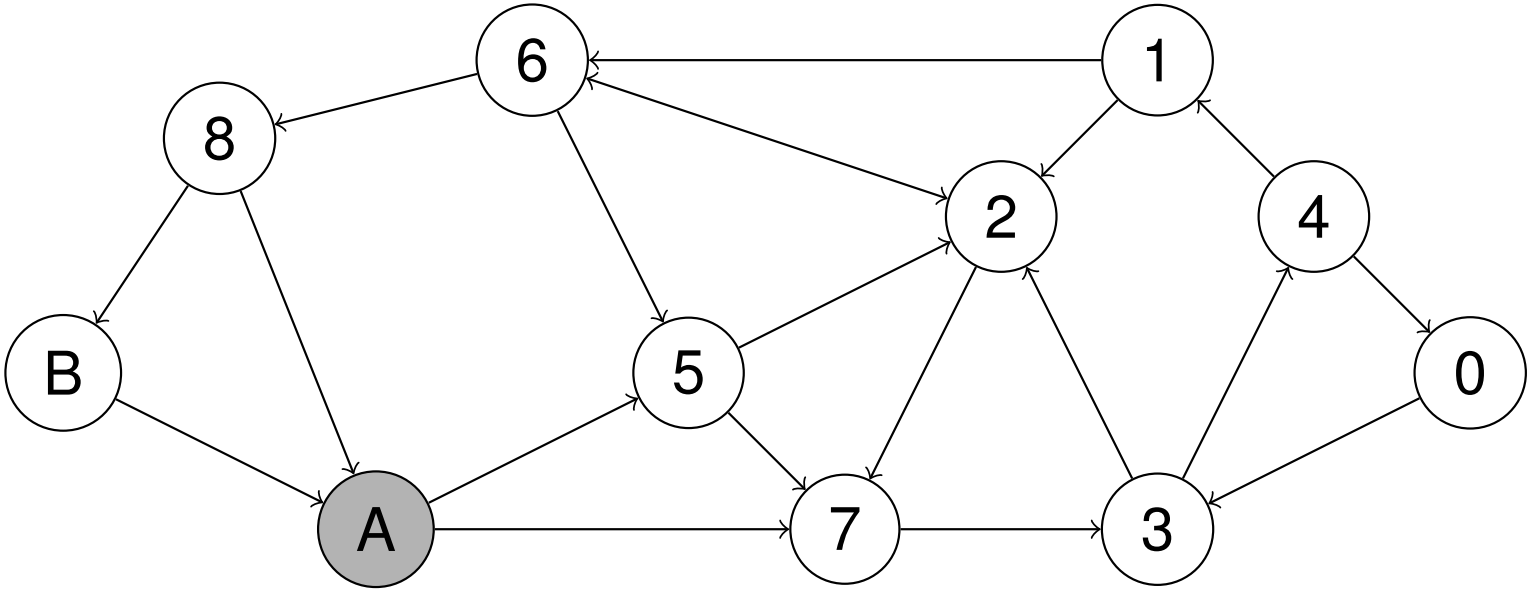
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

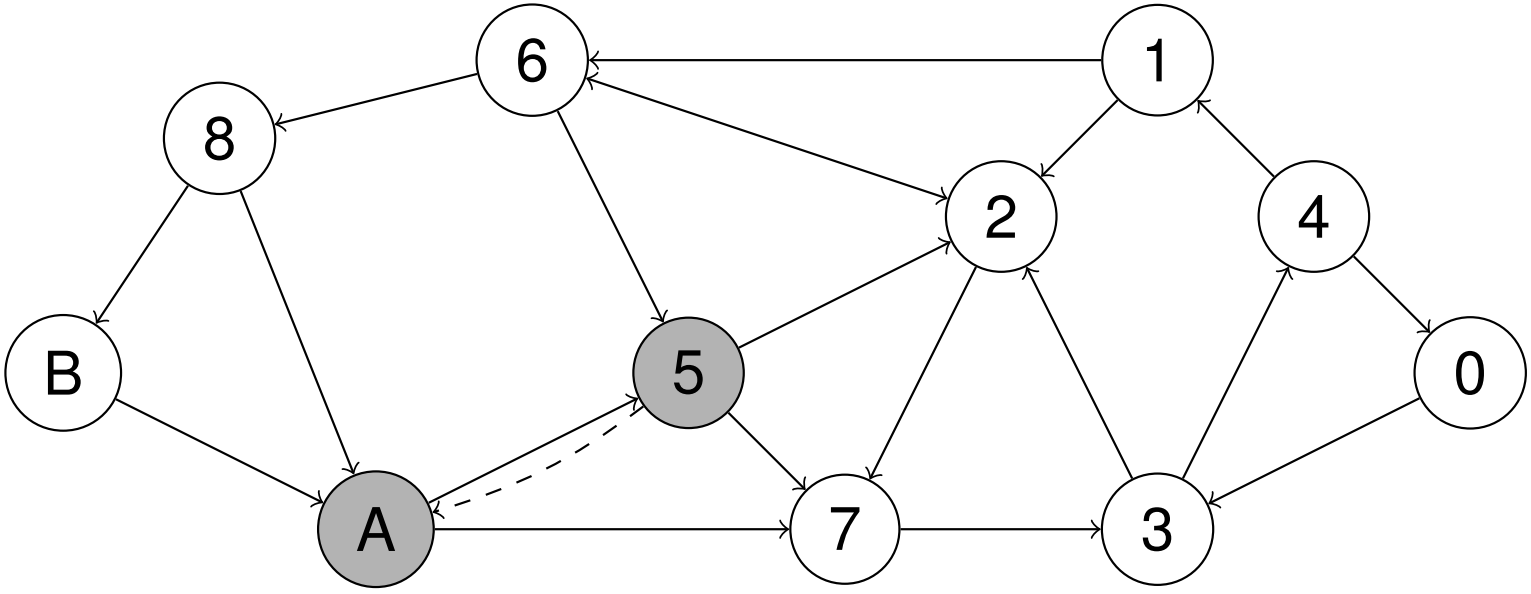
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

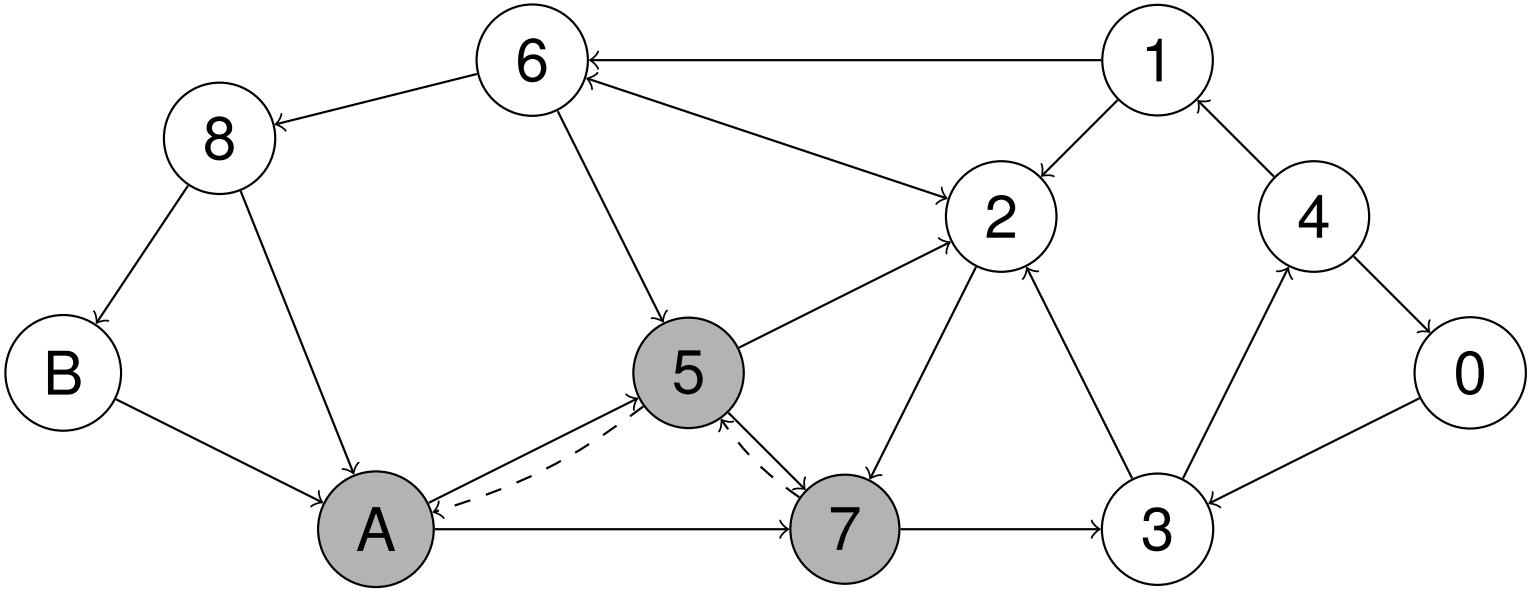
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

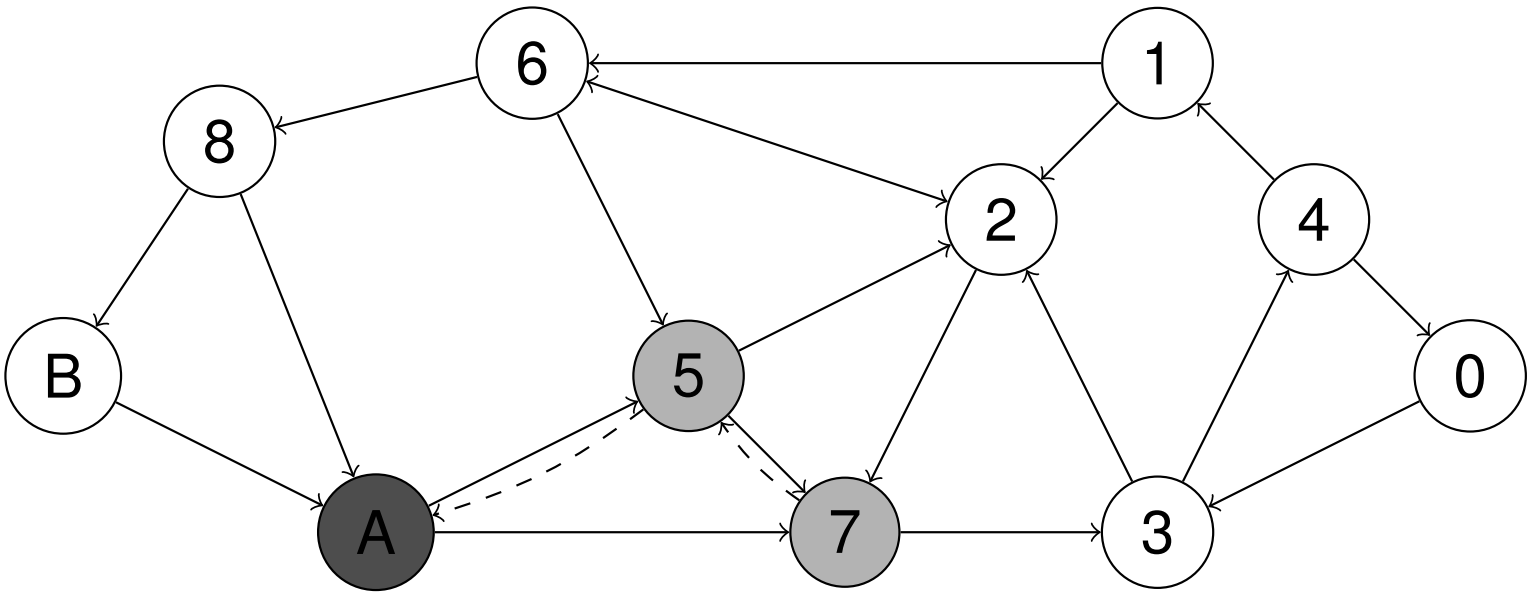
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

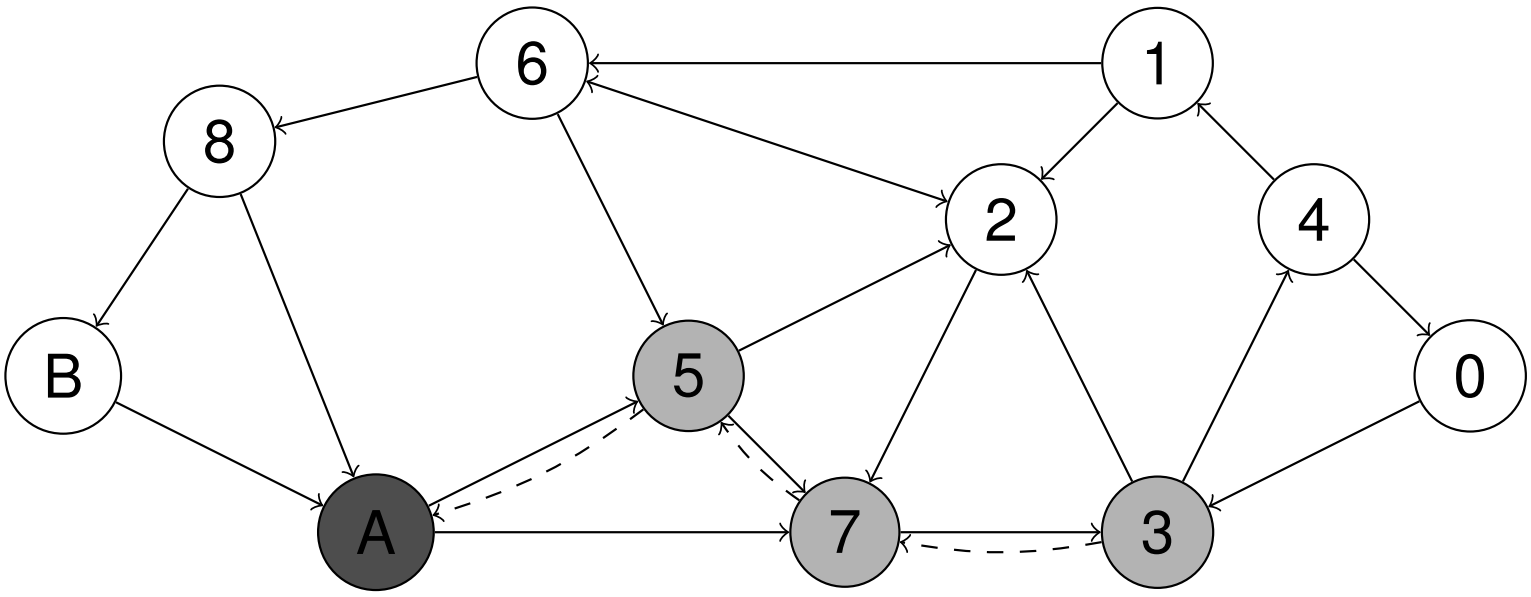
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

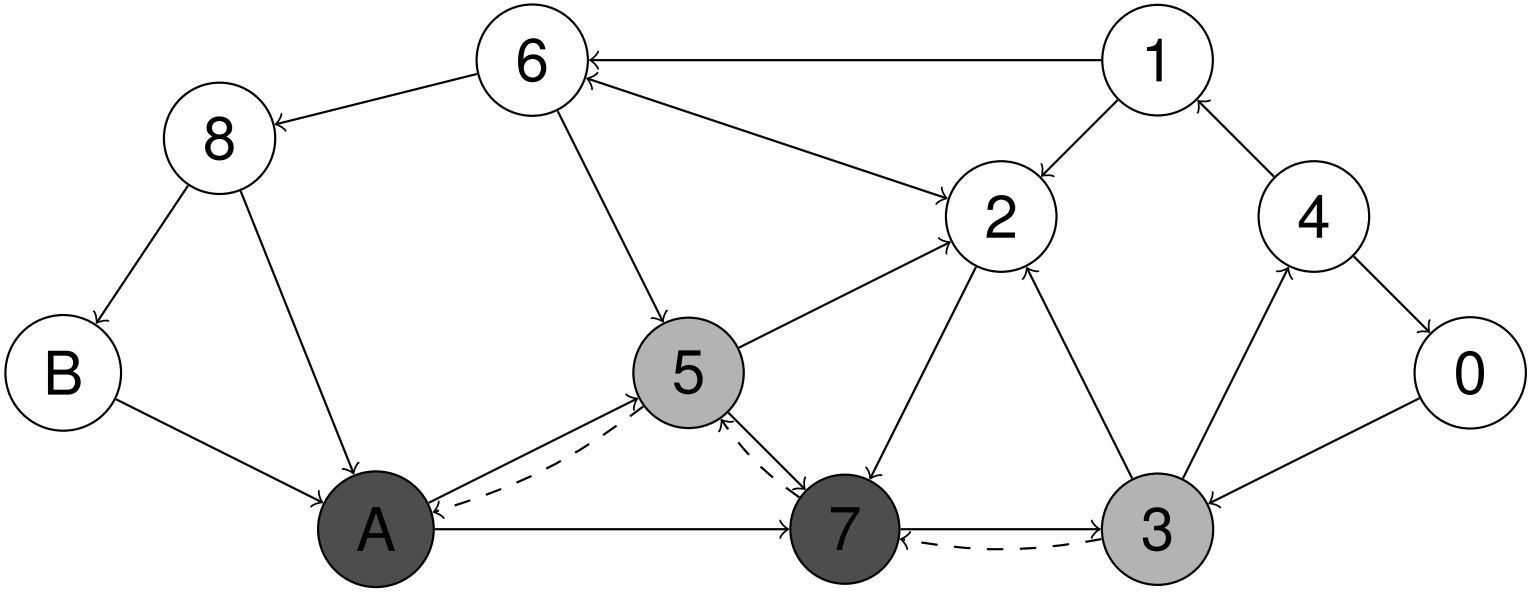
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

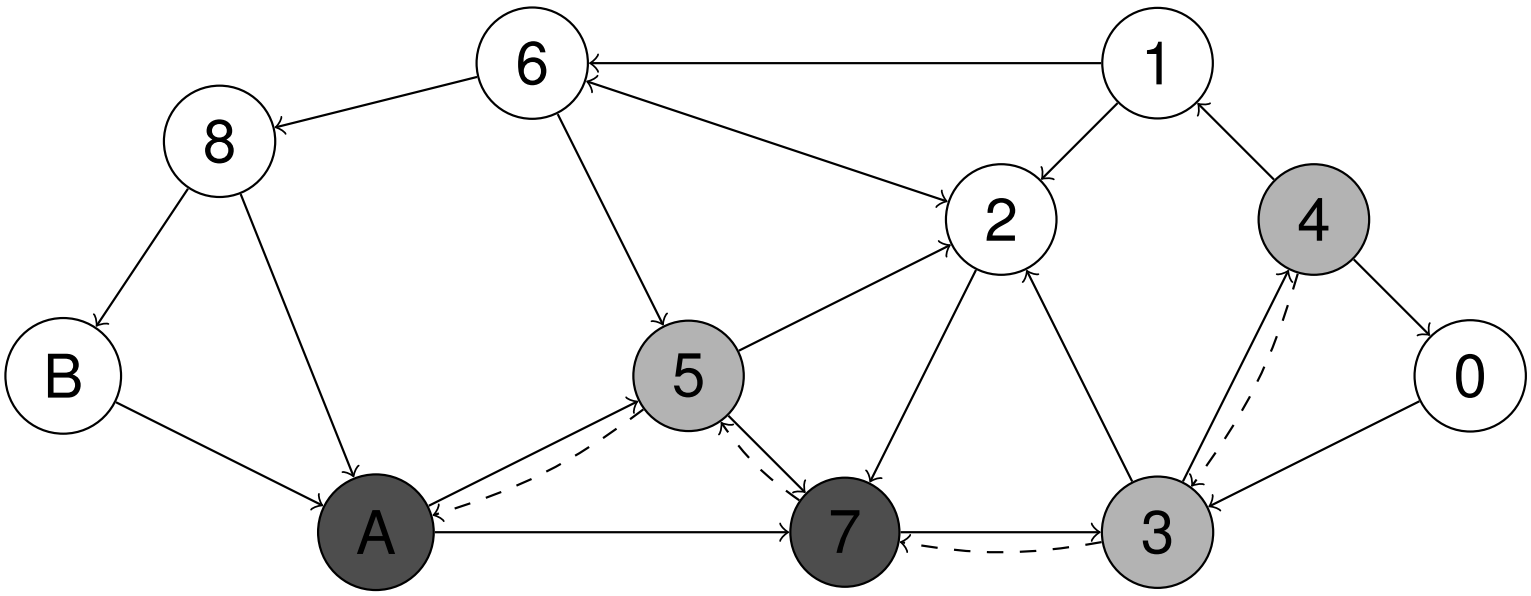
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

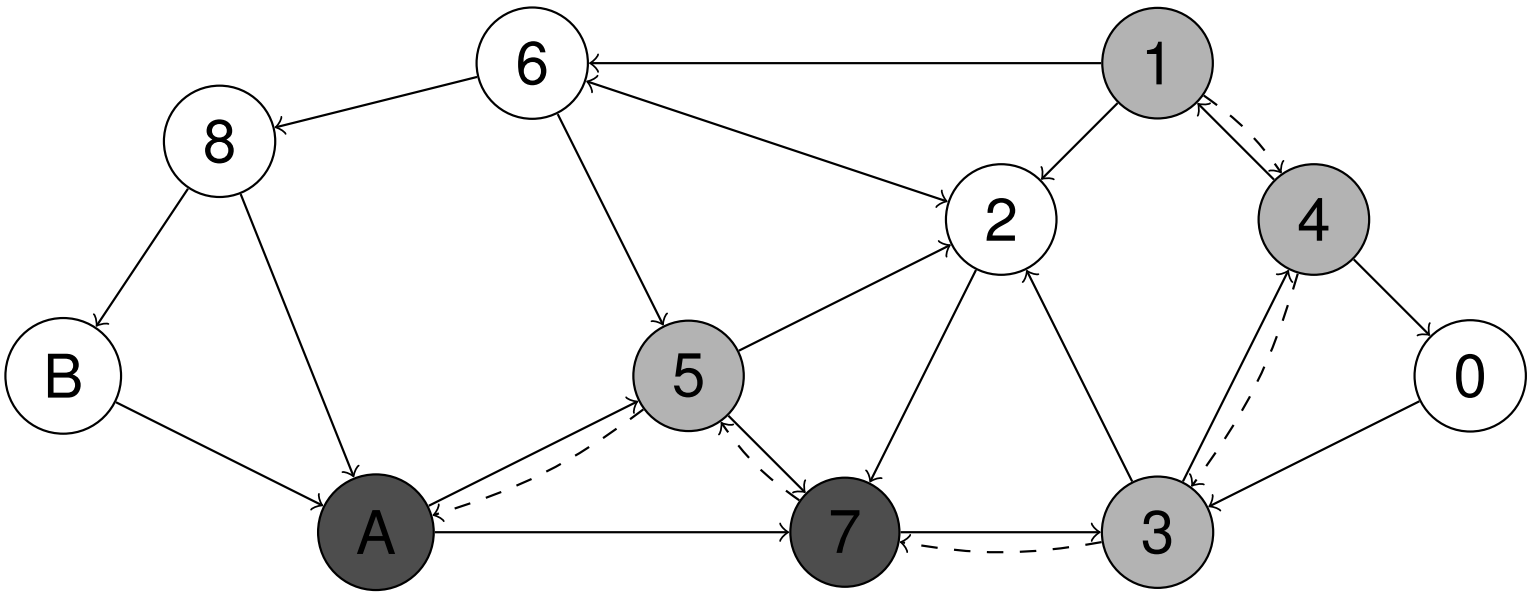
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

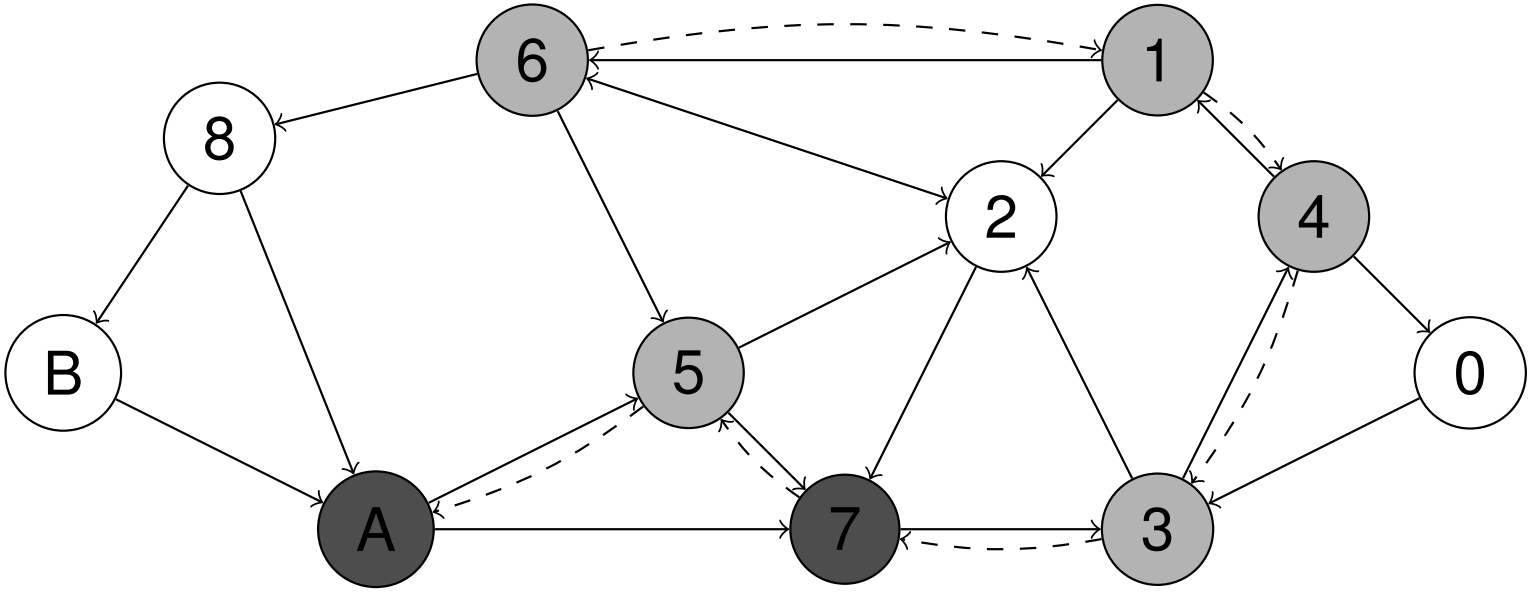
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

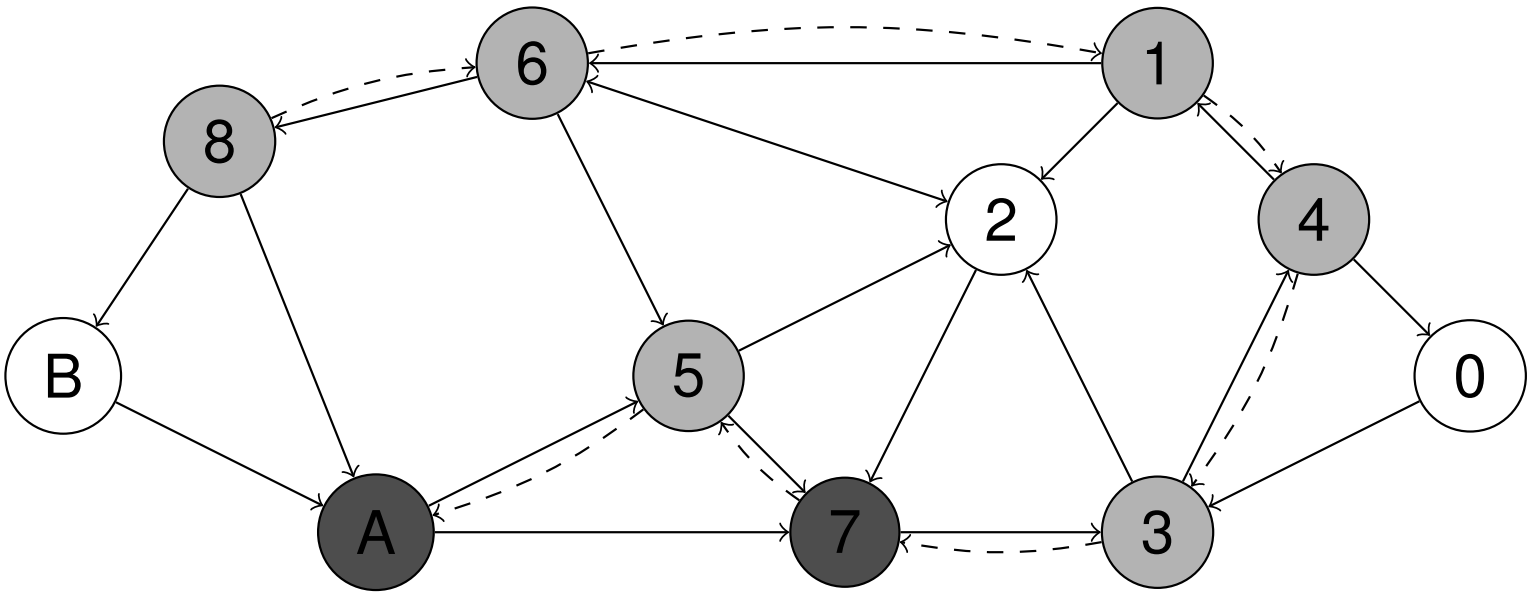
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

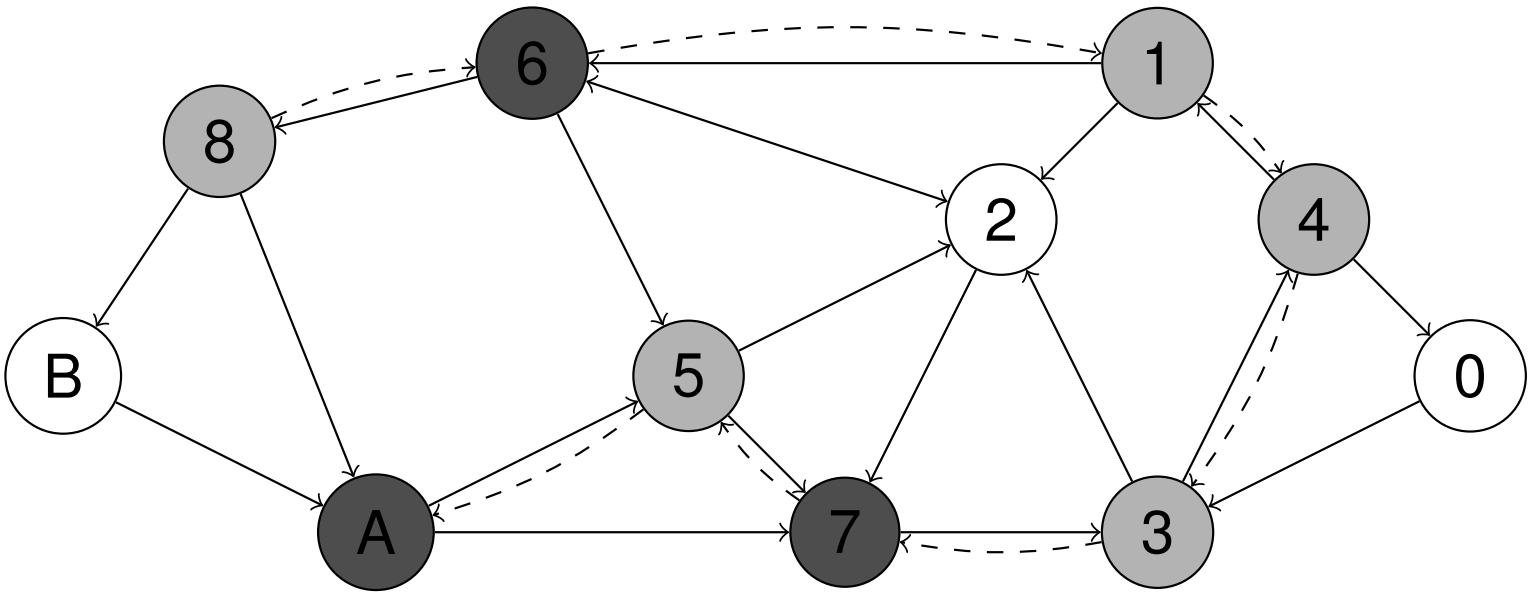
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

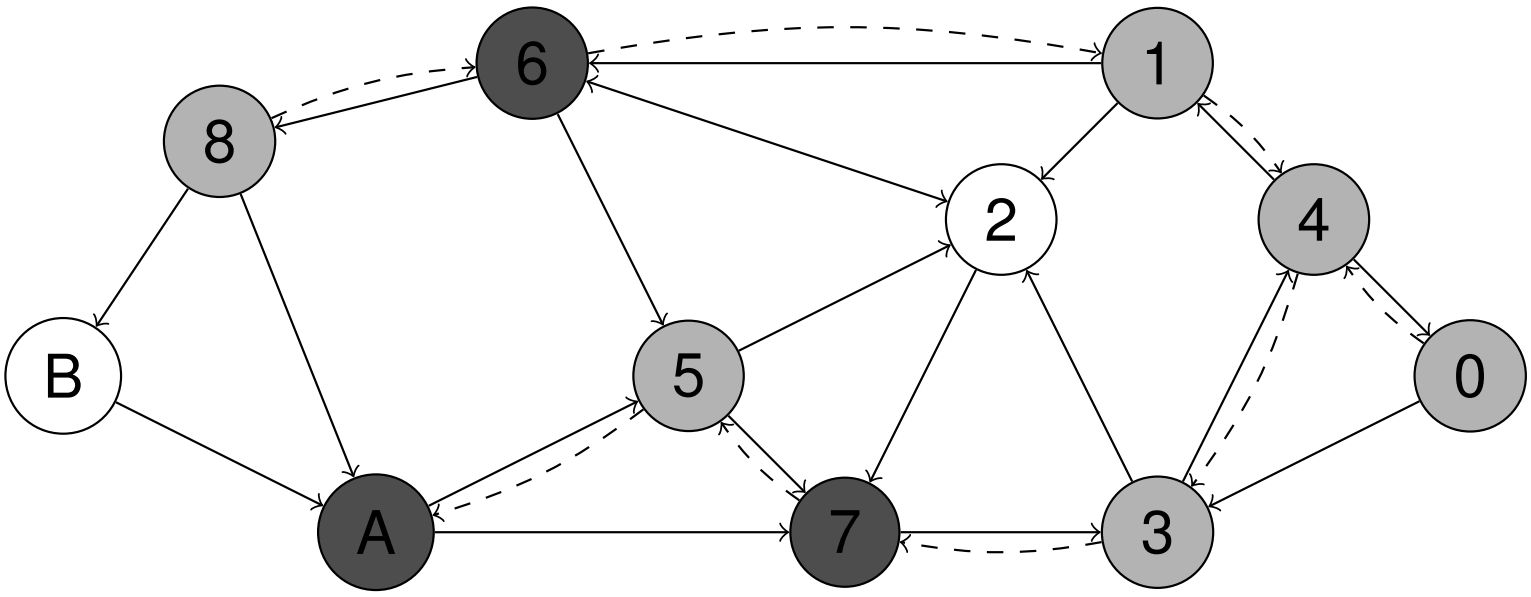
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

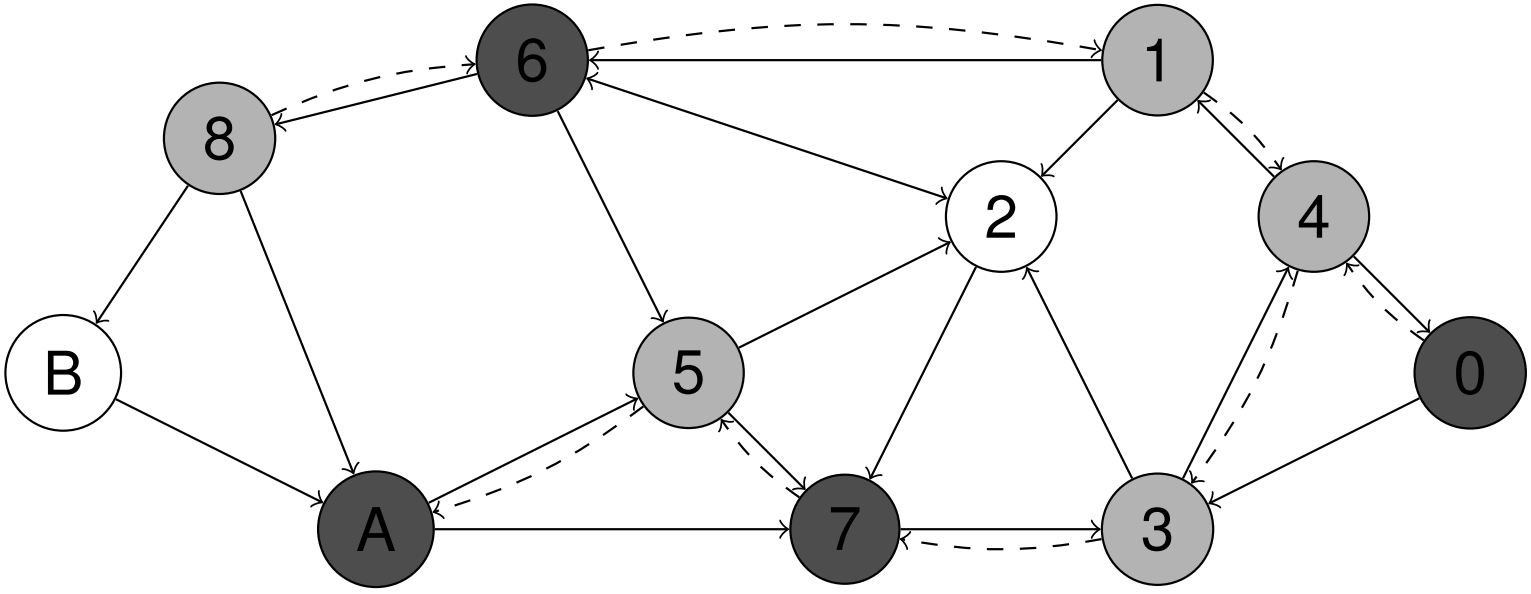
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

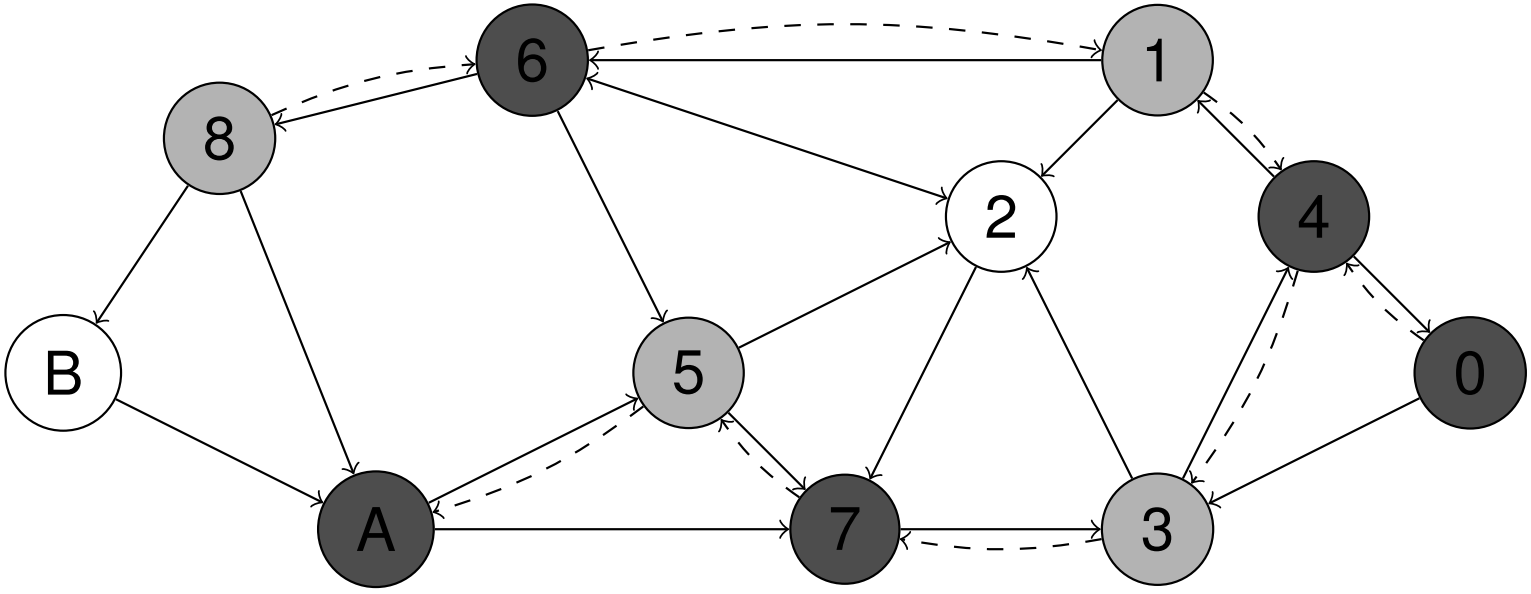
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

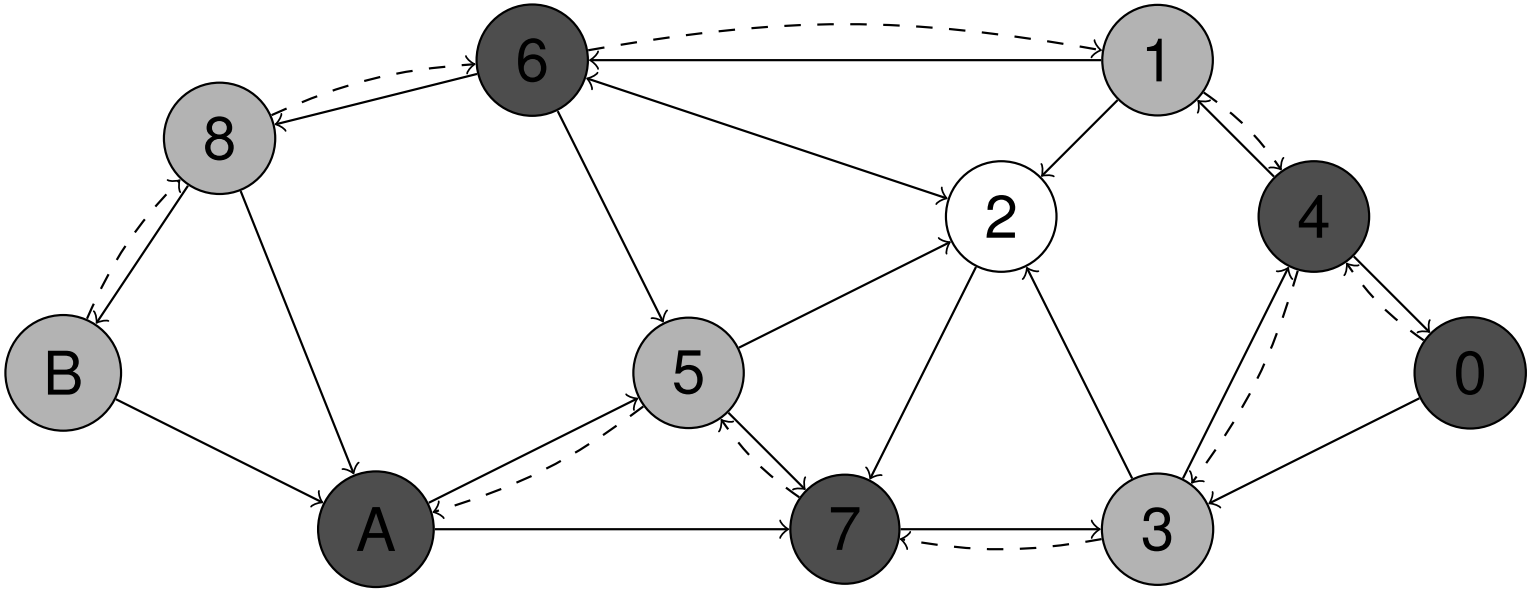
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

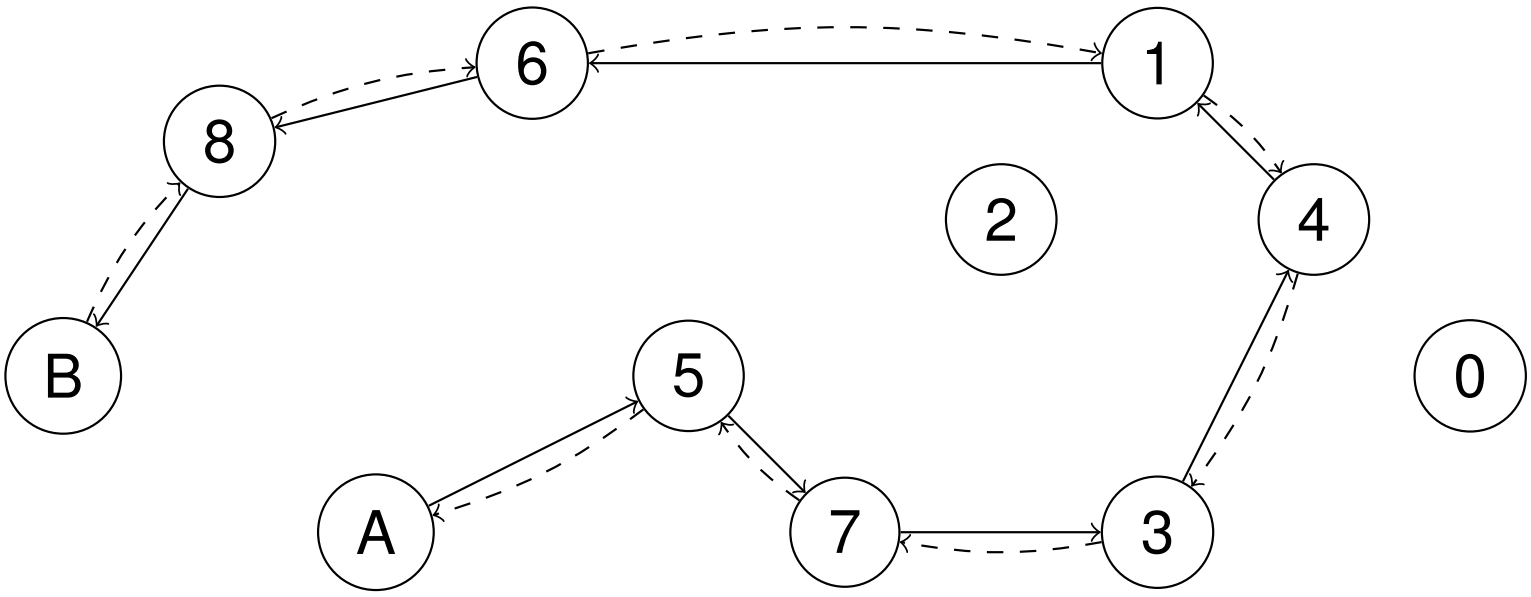
- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Beispiel: Zufällige Suche ohne Wiederholung

Vom Punkt A soll ein Weg zum Punkt B gefunden werden.
Dazu markieren wir:

- weiß: Noch kein Weg gesucht
- hellgrau: Punkt entdeckt, aber nicht abgearbeitet
- dunkelgrau: Alle Nachfolger abgearbeitet
- Bei entdeckten Knoten: Von wo entdeckt?



Breitensuche

Im Suchbeispiel wird nicht der kürzeste Weg gefunden

Breitensuche ist eine Suche mit geschickter Suchreihenfolge

- Dabei wird ein kürzester Weg gefunden
- Die entdeckten, aber nicht abgearbeiteten Knoten stehen in einer Warteschlange: Wer als erstes entdeckt wurde, kommt als erstes dran
- Geeignete Klassen für eine Warteschlange in Java: Interface Queue, Deque; Implementation ArrayDeque

Interface Deque, Implementierung ArrayDeque

Eine `Deque` ist eine zweiseitige Warteschlange

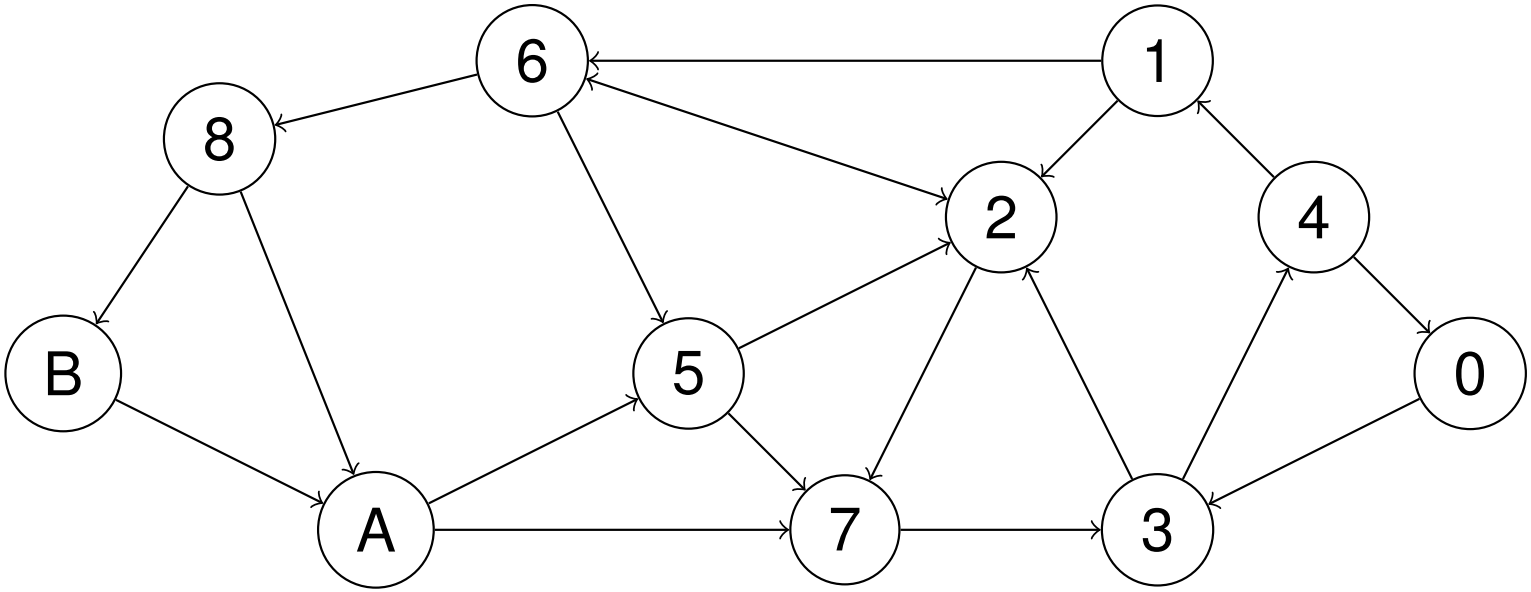
- Enthält eine Reihe an Elementen
- An beiden Seiten kann ein Element entfernt oder eingefügt werden

Beispiel: Deque<String> deq = | A | B | C | D |

- deq.size() $\rightarrow$ 4; deq unverändert
- deq.peekFirst() $\rightarrow$ A; deq unverändert
- deq.peekLast() $\rightarrow$ D; deq unverändert
- deq.addFirst(X) $\rightarrow$ deq verändert zu | X | A | B | C | D |
- deq.addLast(X) $\rightarrow$ deq verändert zu | A | B | C | D | X |
- deq.removeFirst() $\rightarrow$ A; deq verändert zu | B | C | D |
- deq.removeLast() $\rightarrow$ D; deq verändert zu | A | B | C |

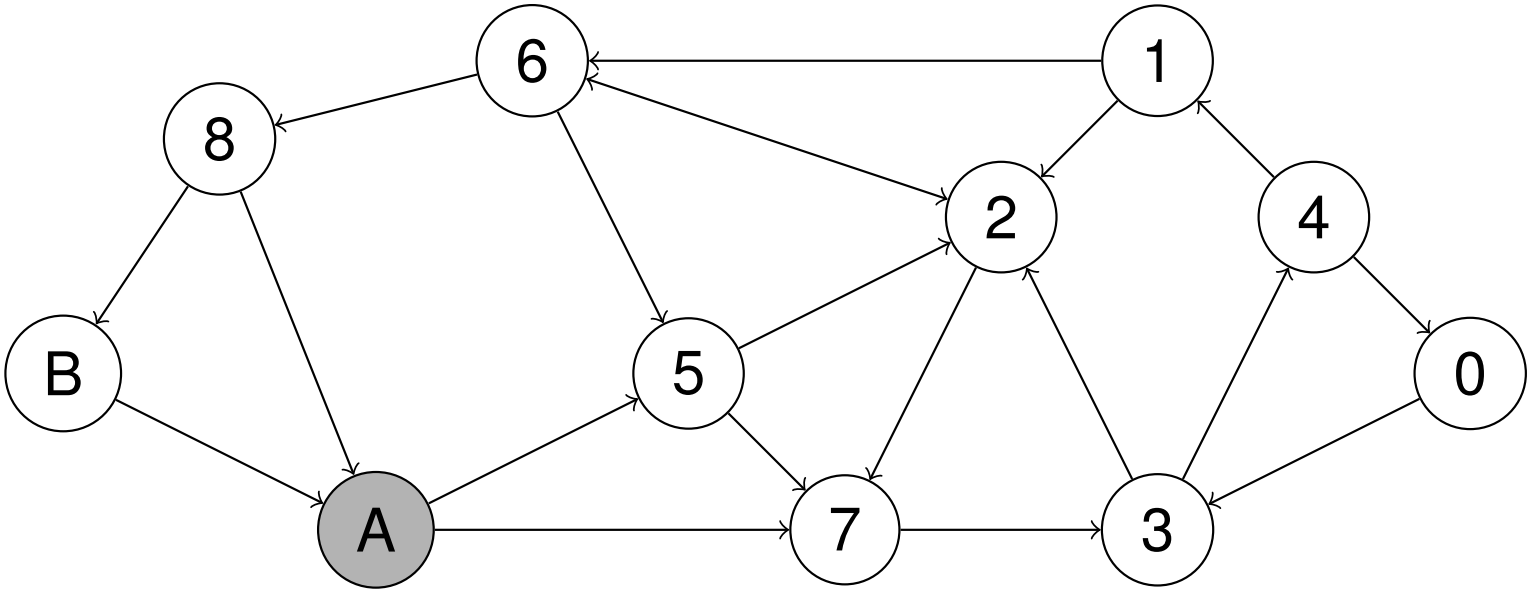
Beispiel: Breitensuche

Gleiches Beispiel nochmal, diesmal mit Breitensuche



Beispiel: Breitensuche

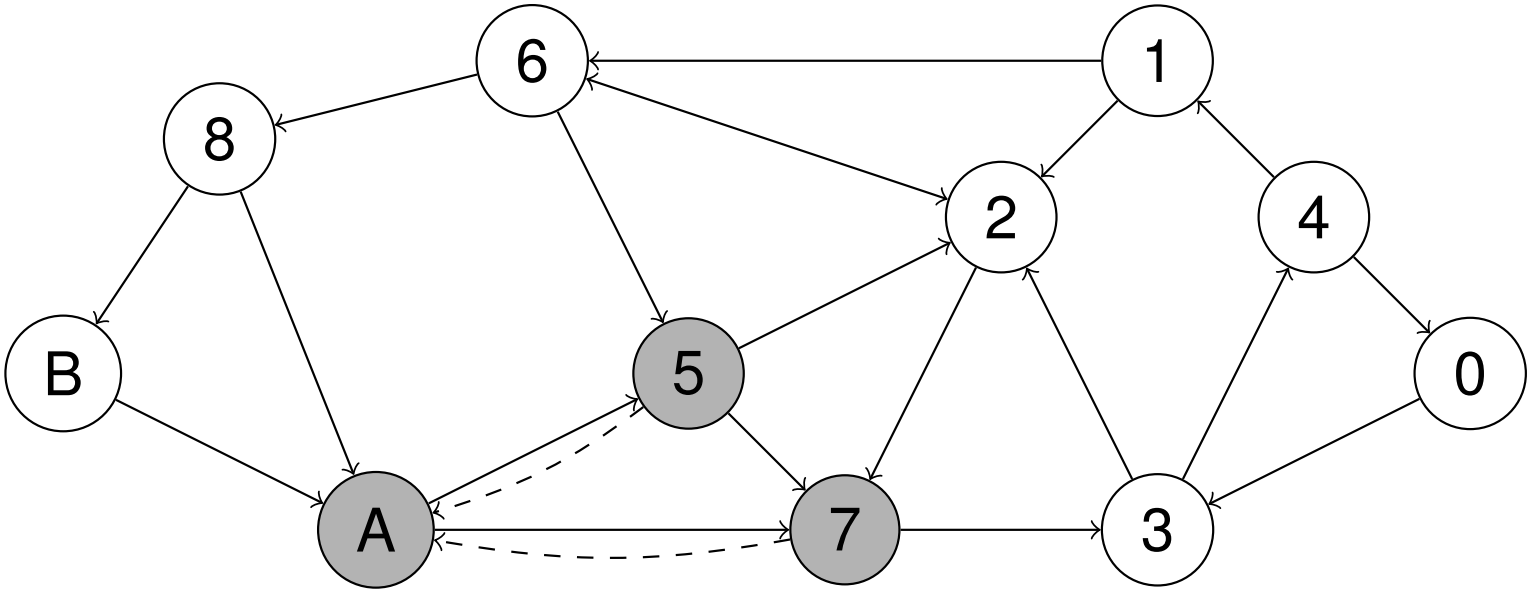
Gleiches Beispiel nochmal, diesmal mit Breitensuche



A

Beispiel: Breitensuche

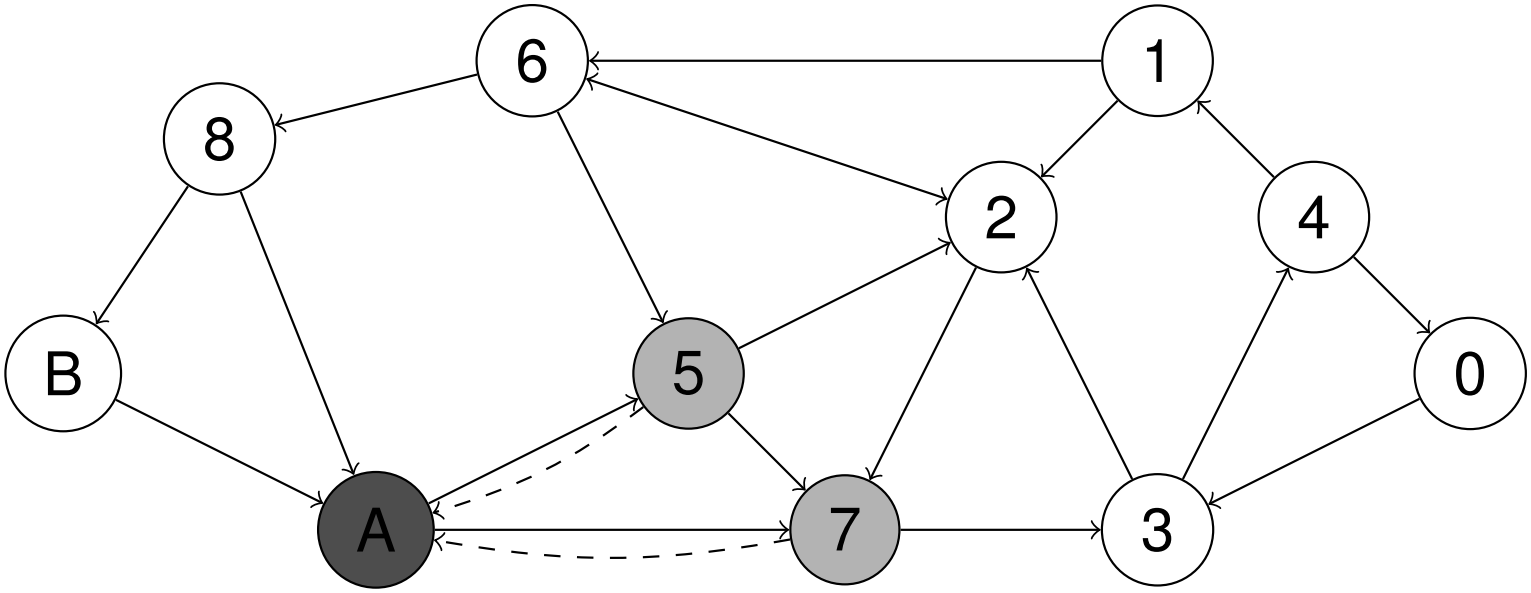
Gleiches Beispiel nochmal, diesmal mit Breitensuche



A	5	7

Beispiel: Breitensuche

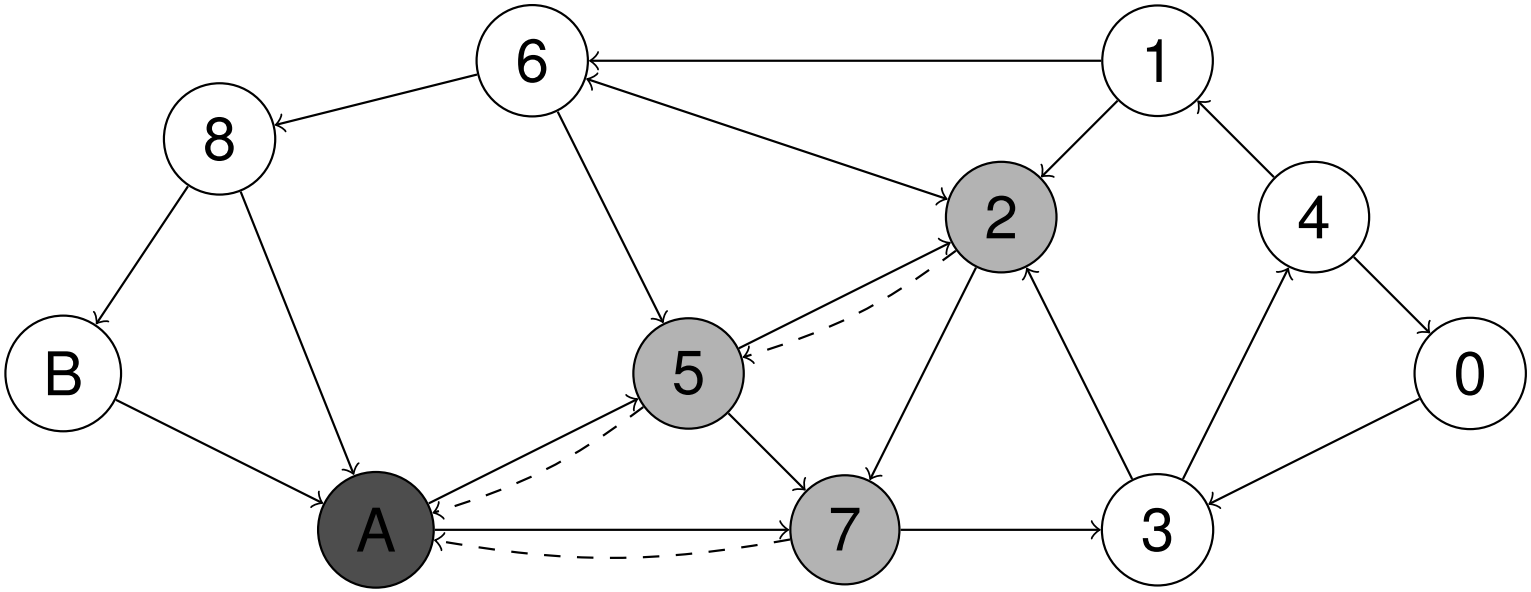
Gleiches Beispiel nochmal, diesmal mit Breitensuche



5	7

Beispiel: Breitensuche

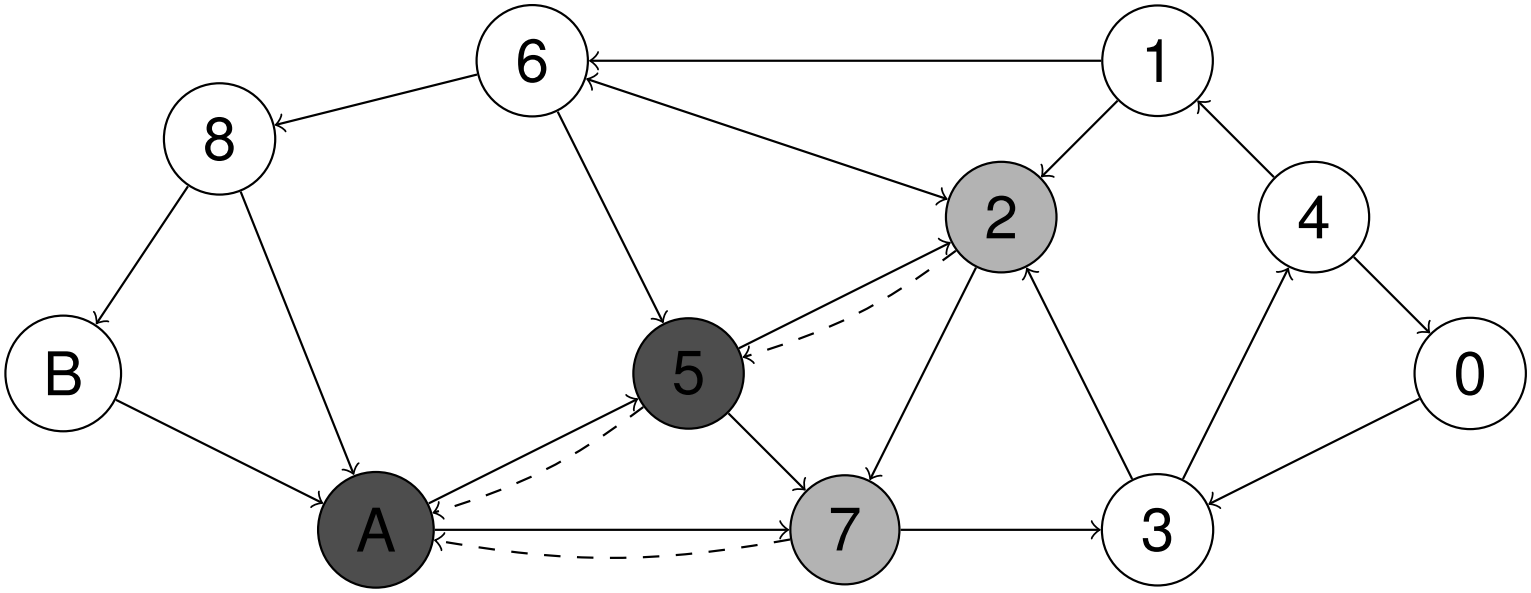
Gleiches Beispiel nochmal, diesmal mit Breitensuche



5	7	2

Beispiel: Breitensuche

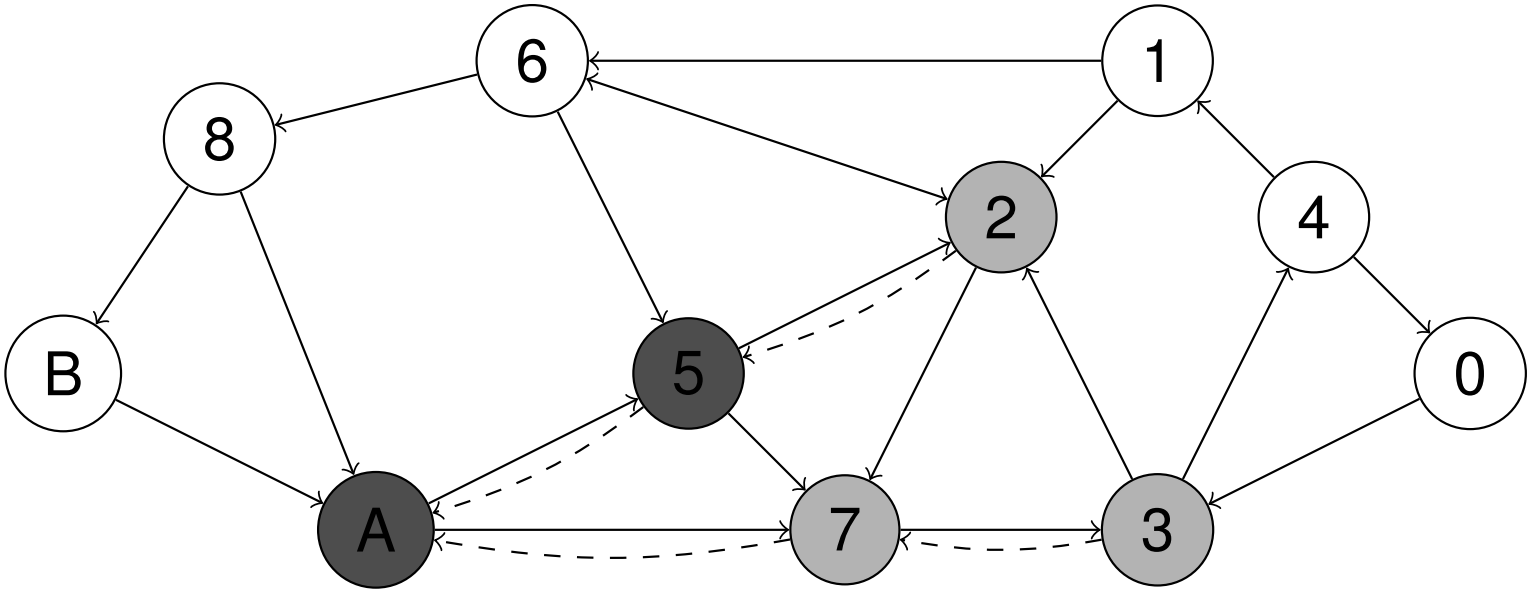
Gleiches Beispiel nochmal, diesmal mit Breitensuche



7	2

Beispiel: Breitensuche

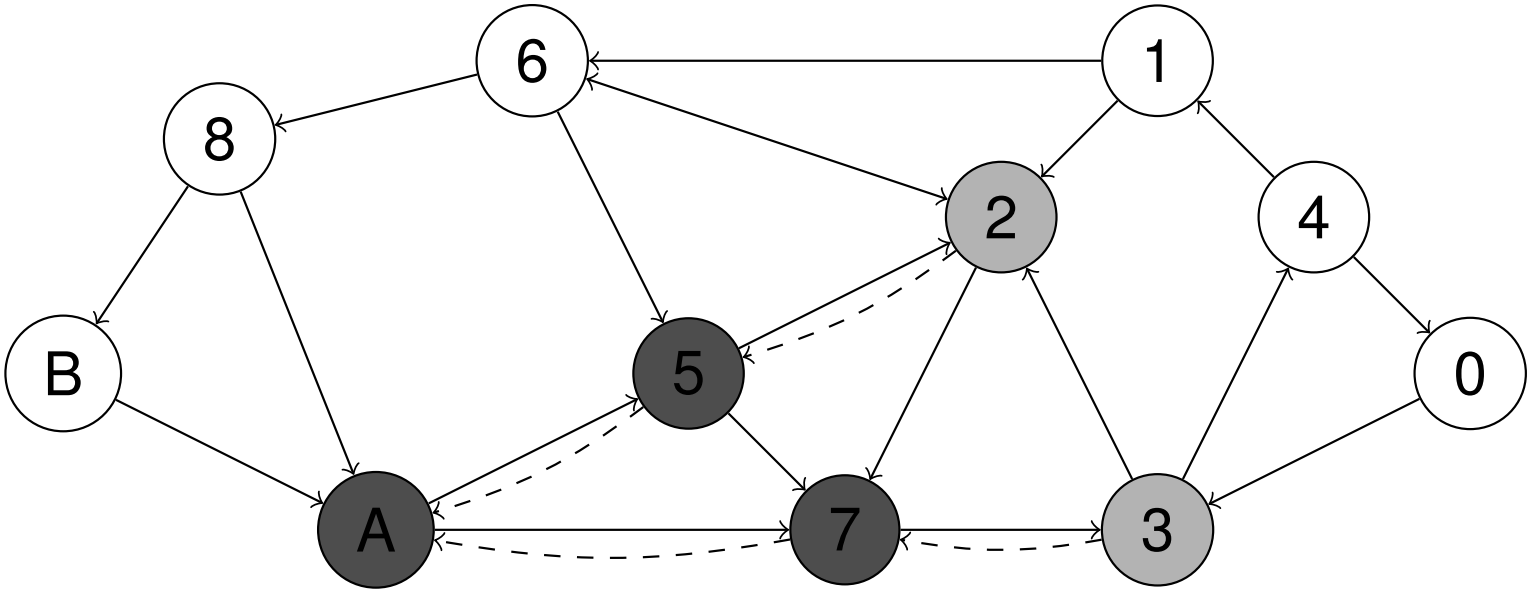
Gleiches Beispiel nochmal, diesmal mit Breitensuche



7	2	3

Beispiel: Breitensuche

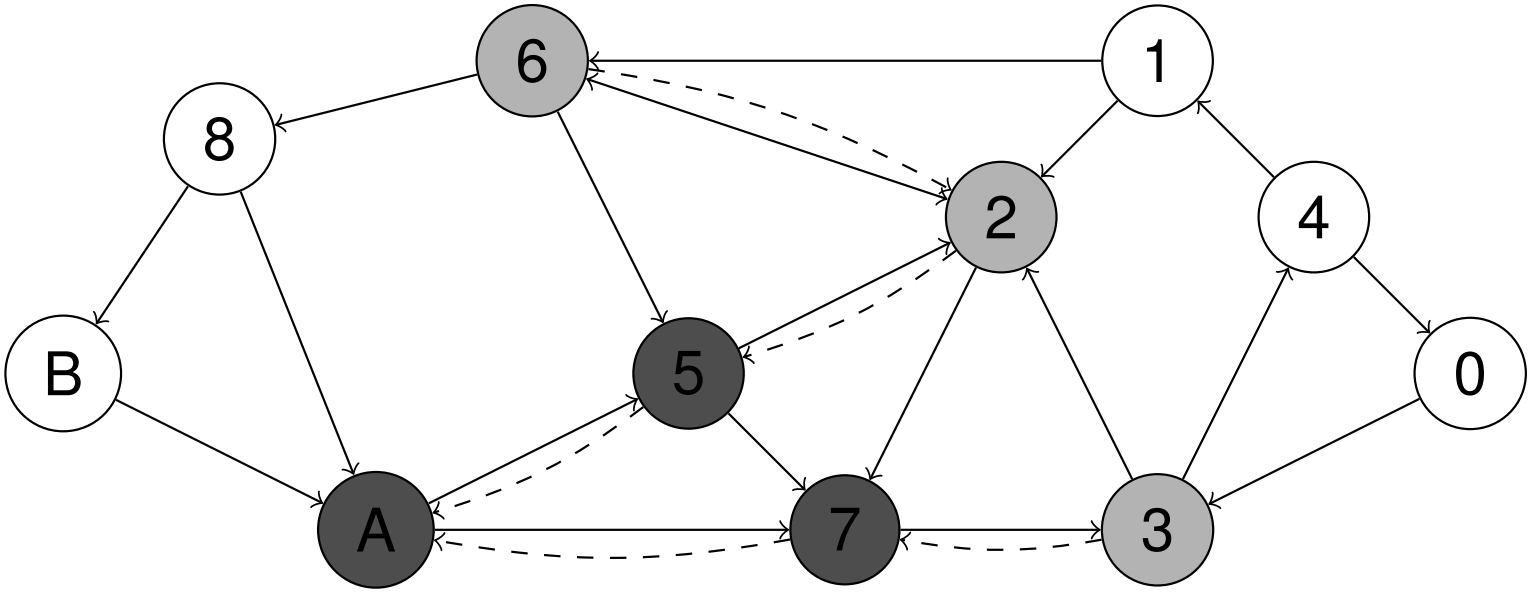
Gleiches Beispiel nochmal, diesmal mit Breitensuche



2	3

Beispiel: Breitensuche

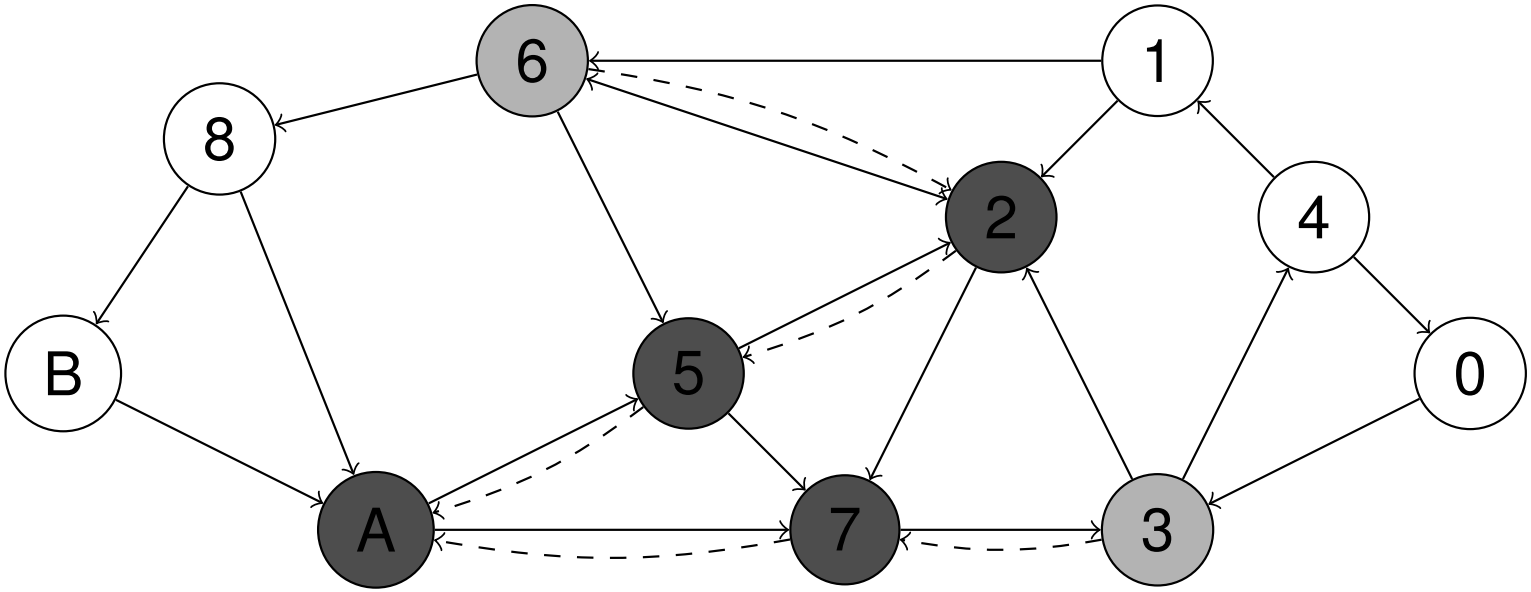
Gleiches Beispiel nochmal, diesmal mit Breitensuche



2	3	6

Beispiel: Breitensuche

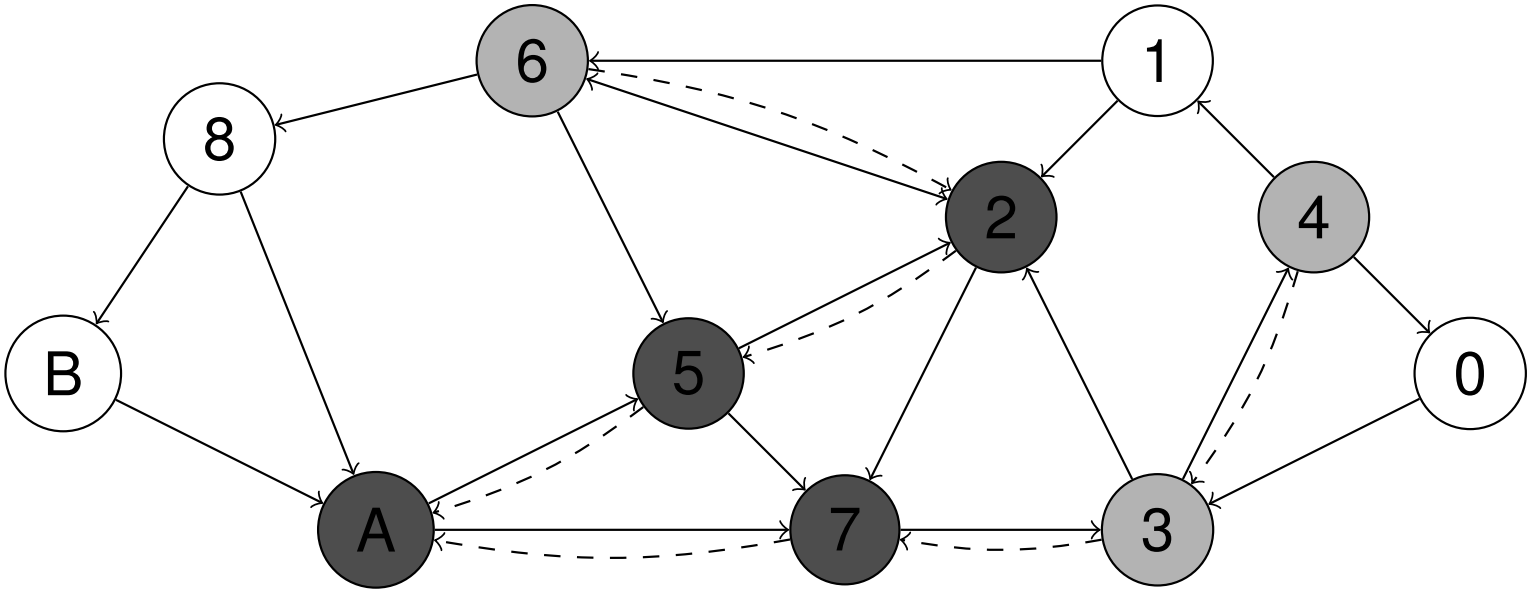
Gleiches Beispiel nochmal, diesmal mit Breitensuche



3	6

Beispiel: Breitensuche

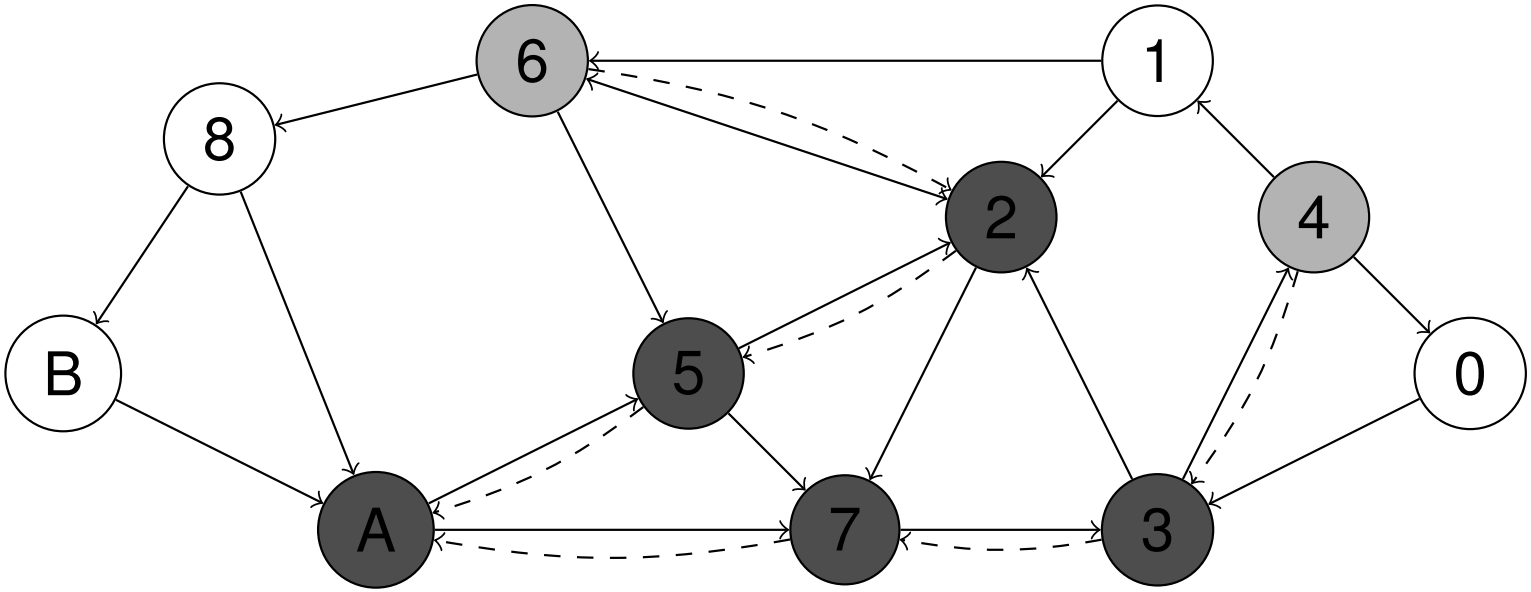
Gleiches Beispiel nochmal, diesmal mit Breitensuche



3	6	4

Beispiel: Breitensuche

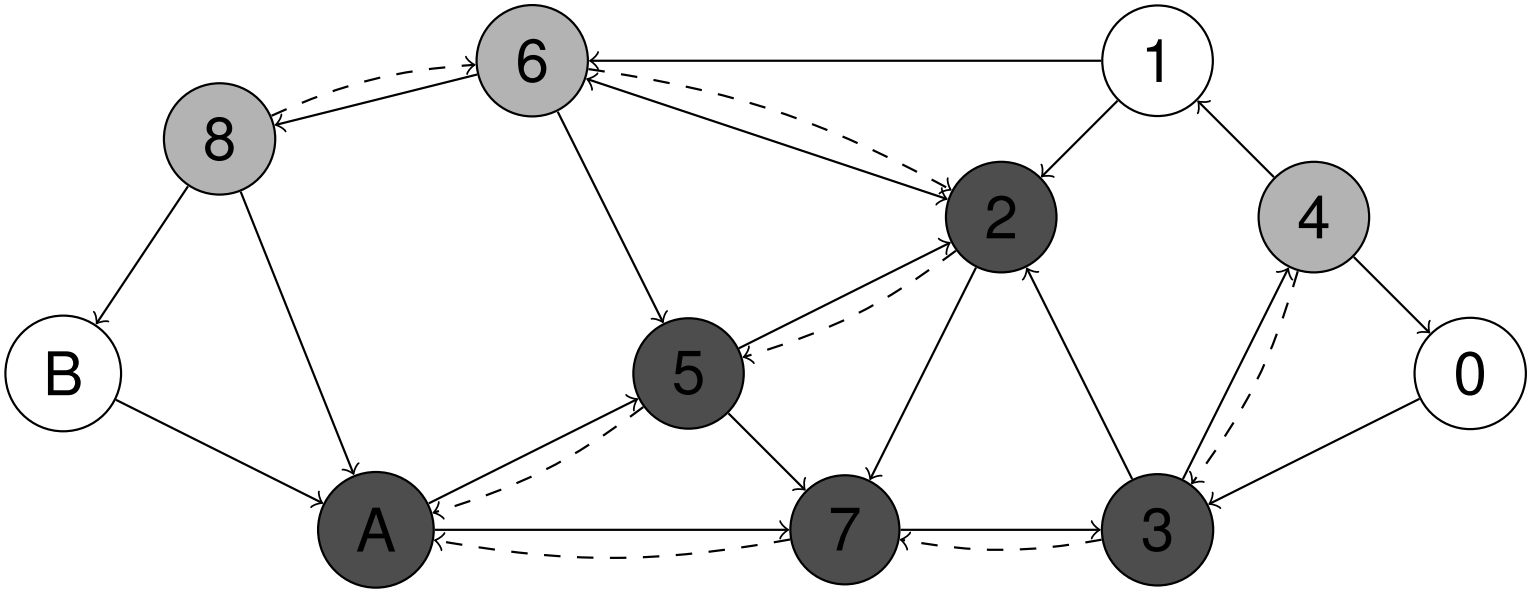
Gleiches Beispiel nochmal, diesmal mit Breitensuche



6	4

Beispiel: Breitensuche

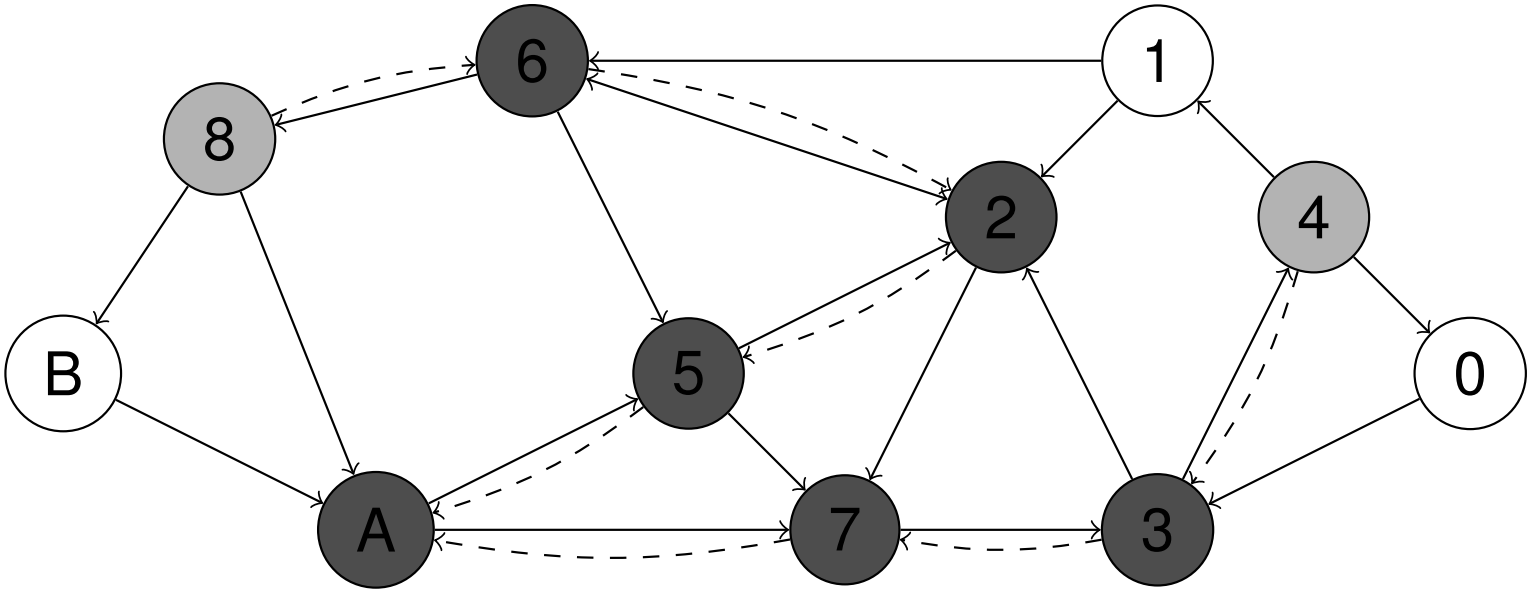
Gleiches Beispiel nochmal, diesmal mit Breitensuche



6	4	8

Beispiel: Breitensuche

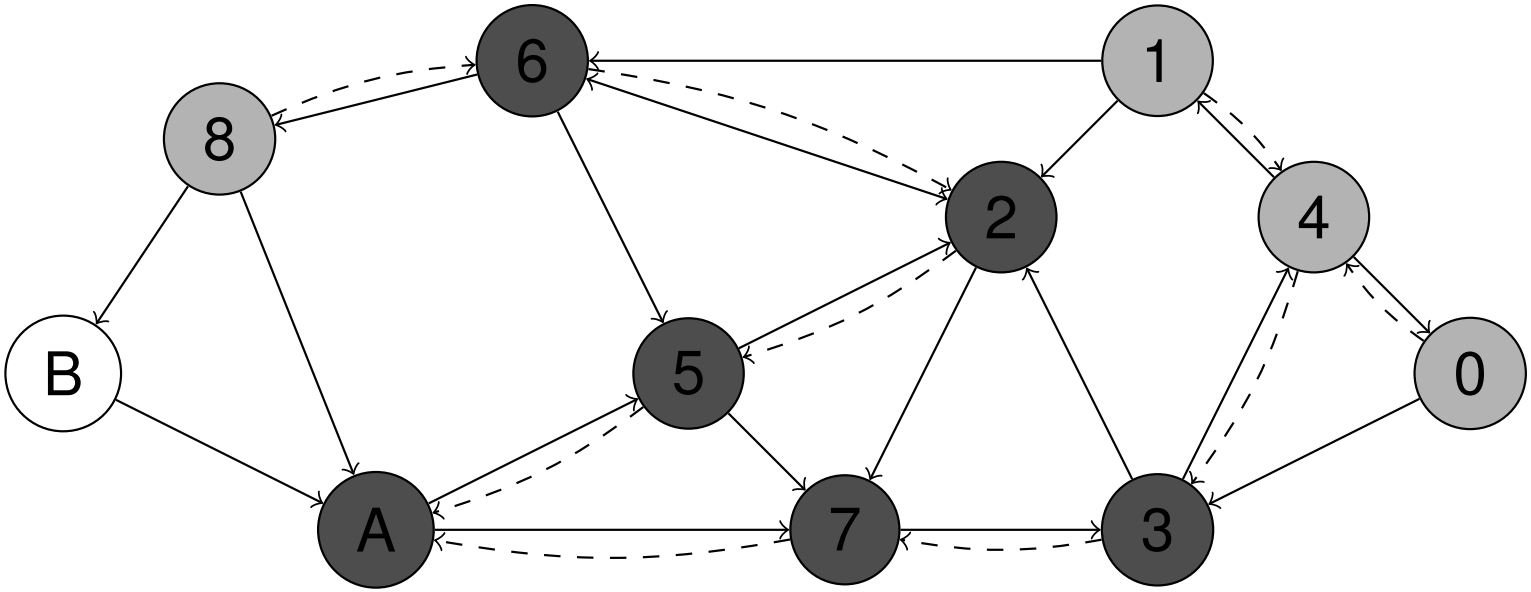
Gleiches Beispiel nochmal, diesmal mit Breitensuche



4	8

Beispiel: Breitensuche

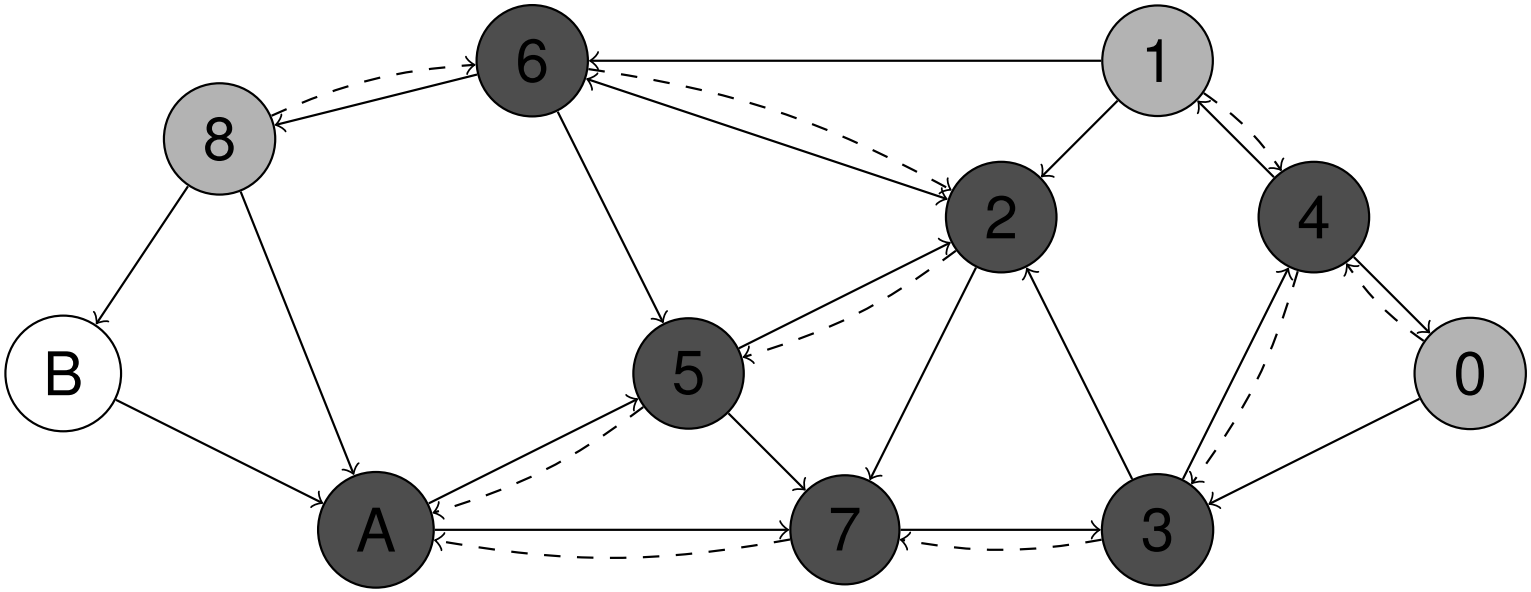
Gleiches Beispiel nochmal, diesmal mit Breitensuche



4	8	1	0

Beispiel: Breitensuche

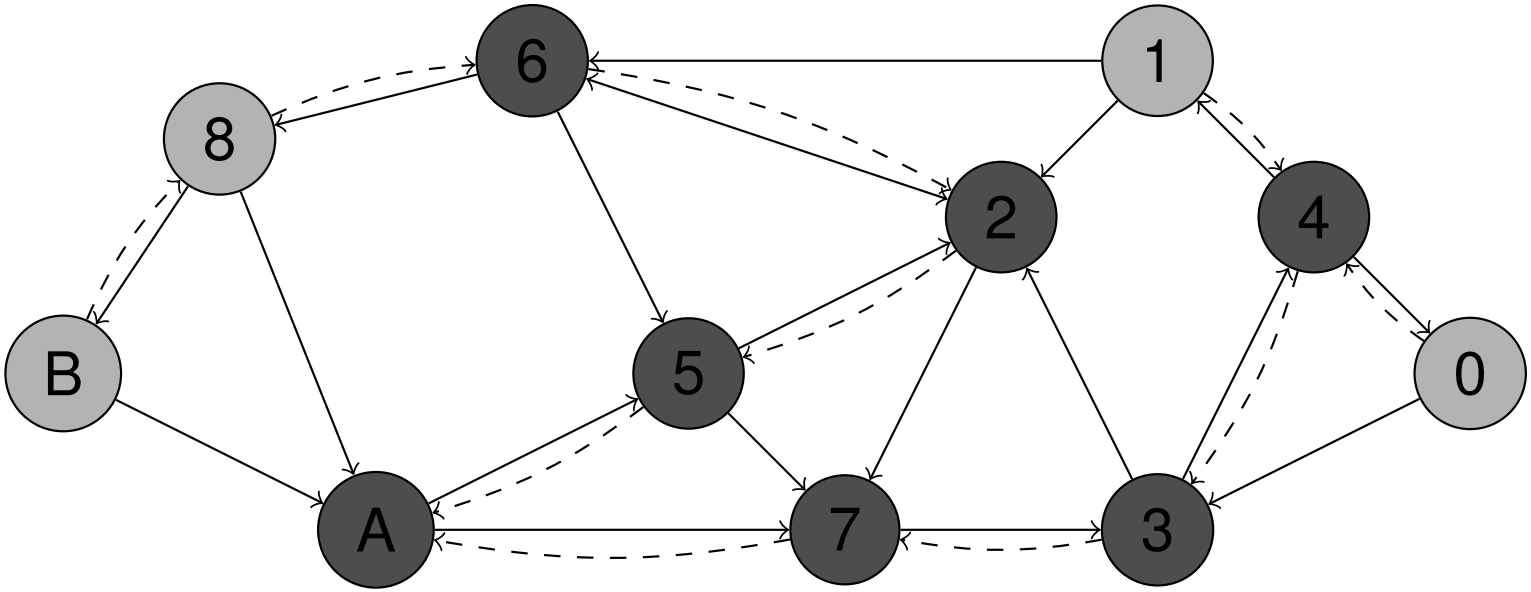
Gleiches Beispiel nochmal, diesmal mit Breitensuche



8	1	0

Beispiel: Breitensuche

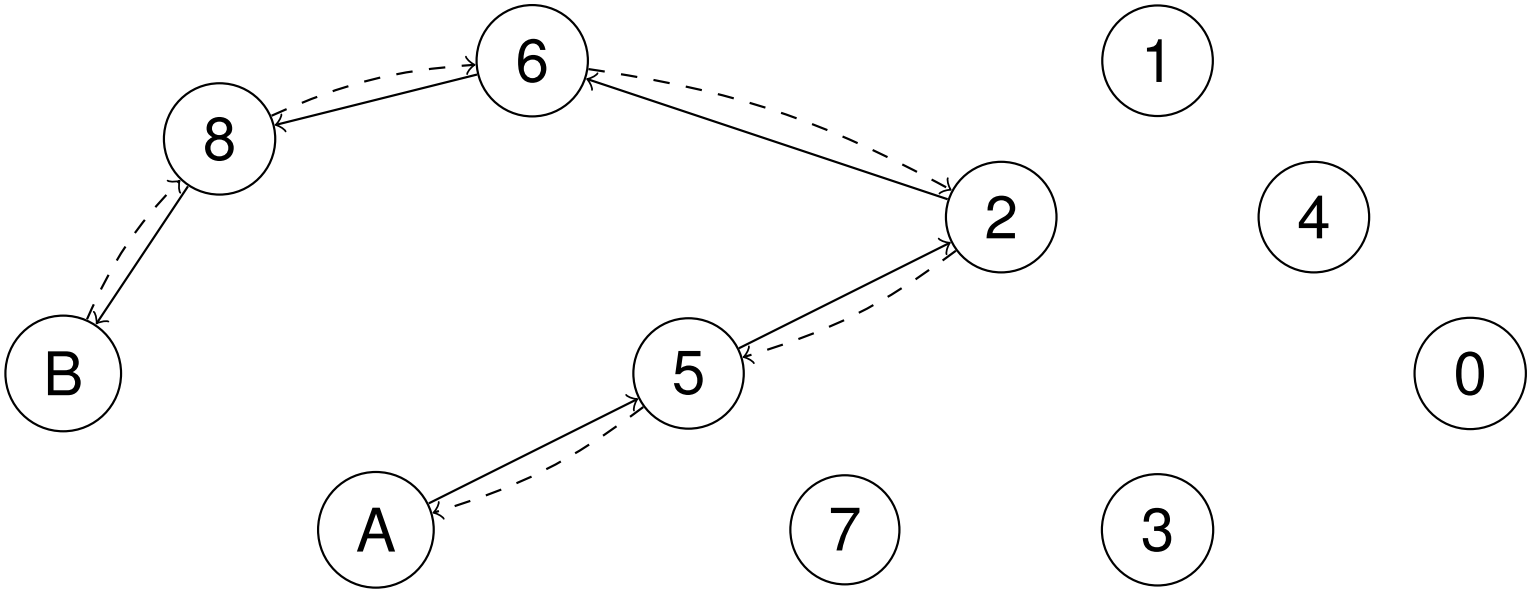
Gleiches Beispiel nochmal, diesmal mit Breitensuche



8	1	0	B

Beispiel: Breitensuche

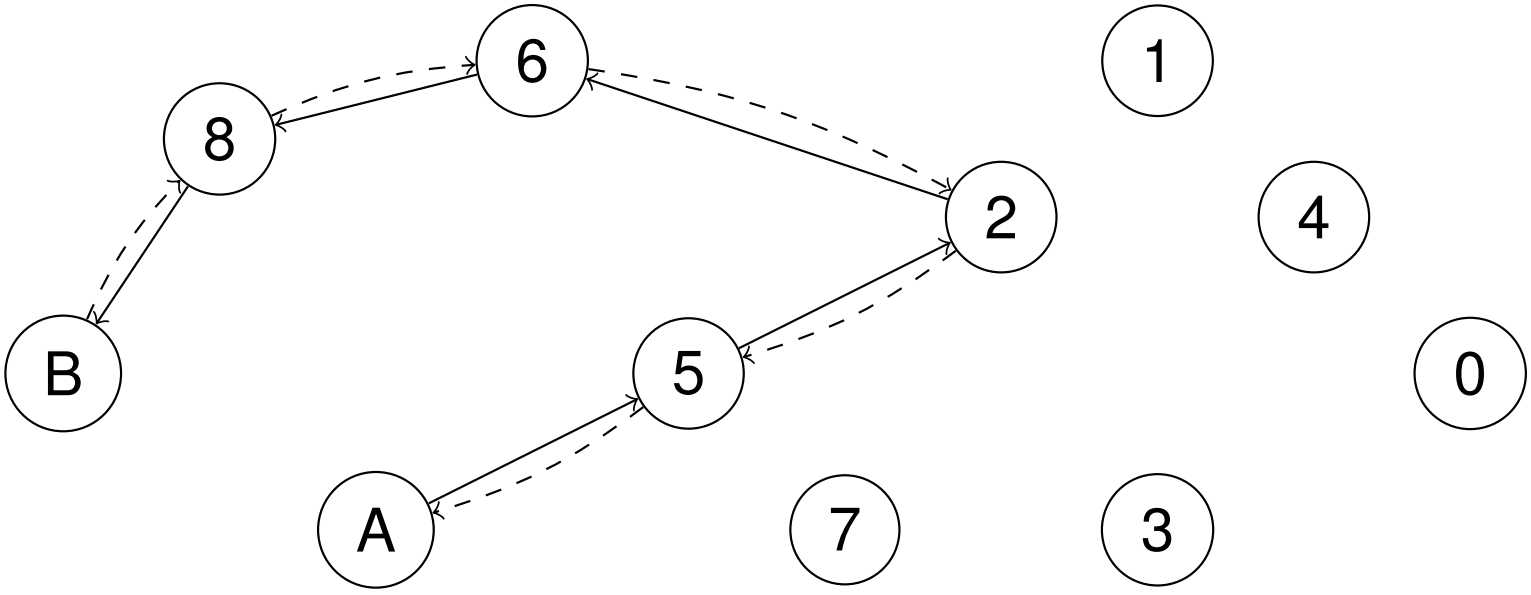
Gleiches Beispiel nochmal, diesmal mit Breitensuche



Weg mit der geringstmöglichen Anzahl an Knoten gefunden.

Beispiel: Breitensuche

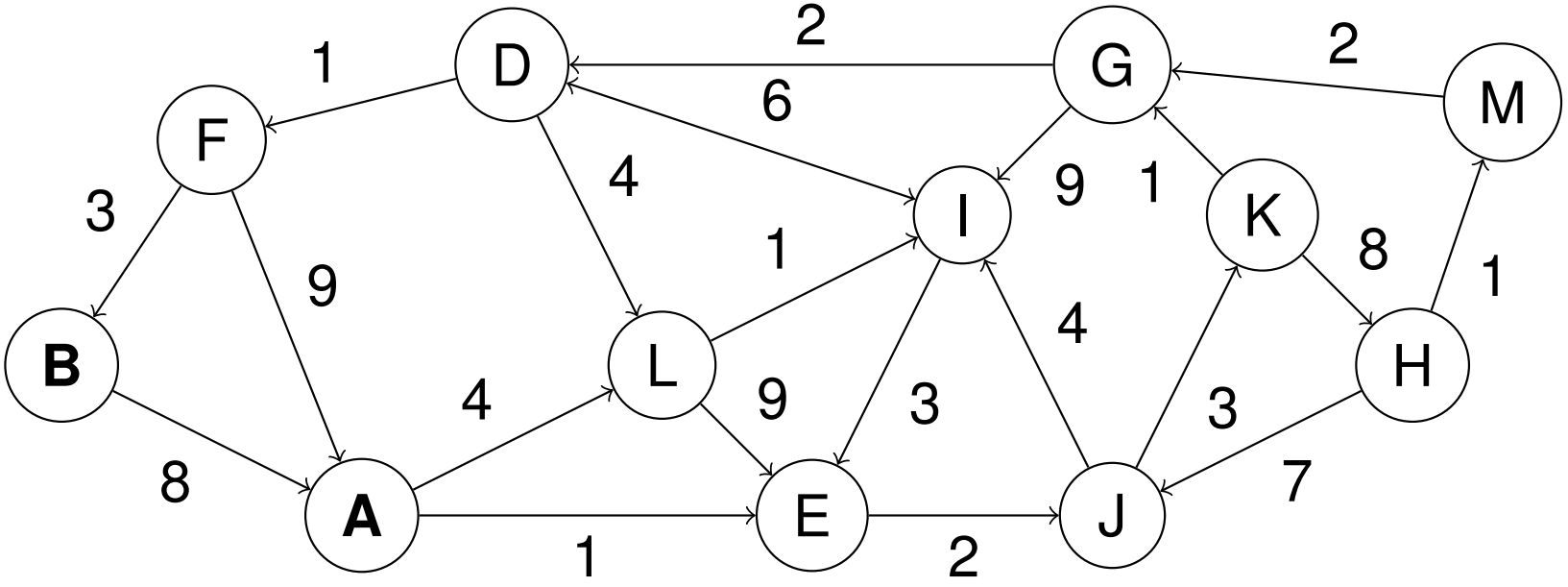
Gleiches Beispiel nochmal, diesmal mit Breitensuche



Weg mit der geringstmöglichen Anzahl an Knoten gefunden.
Frage: Was passiert bei unterschiedlichen Weglängen?

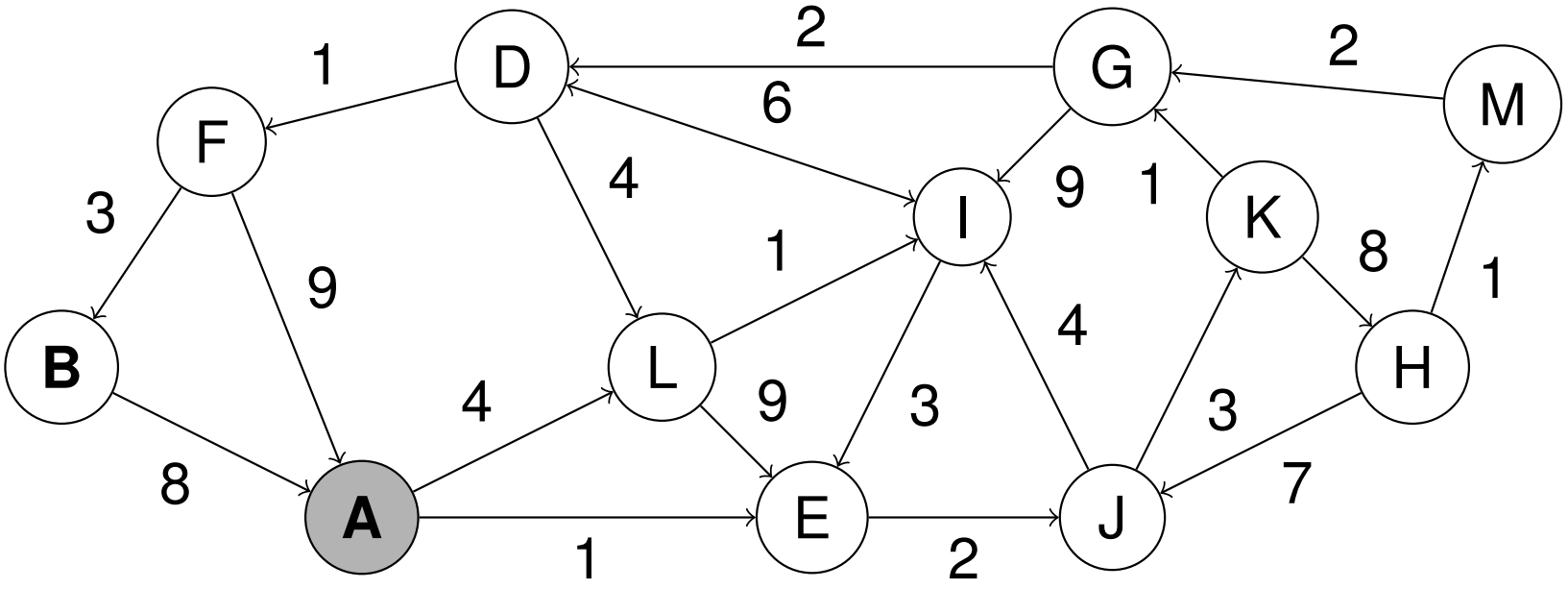
Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



Beispiel: Breitensuche mit Kantengewichten

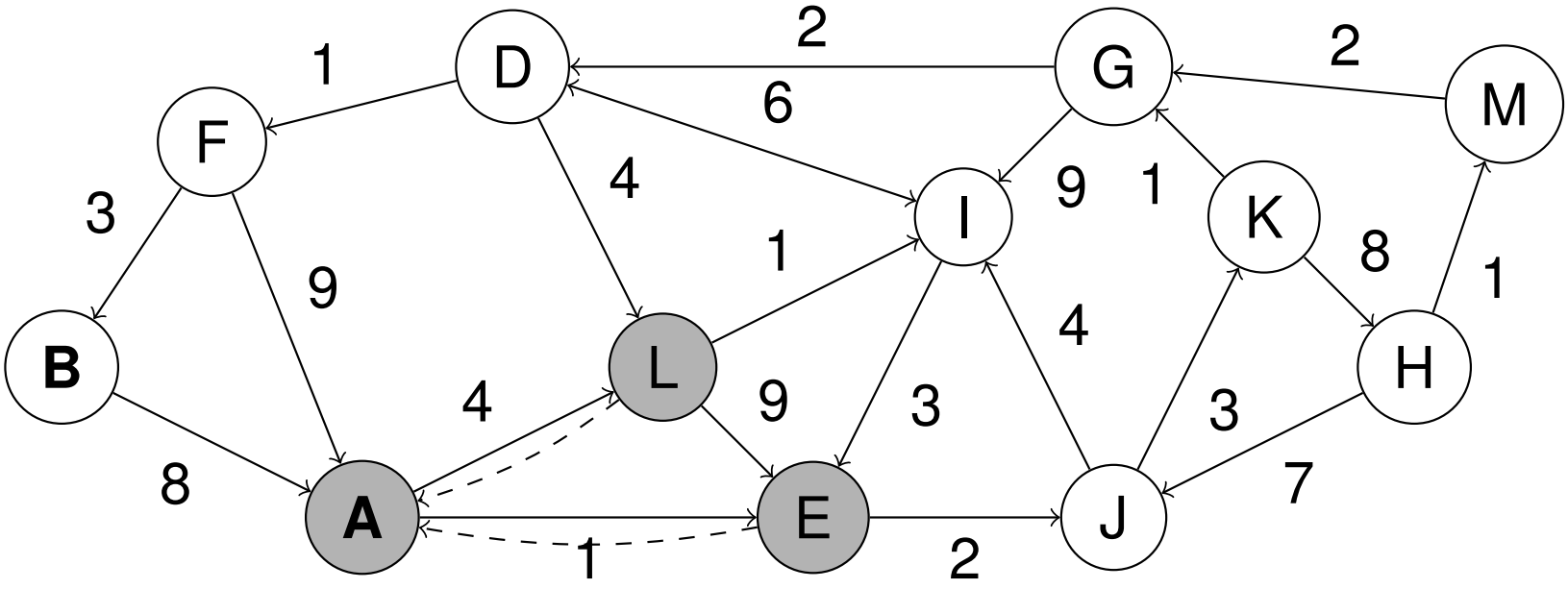
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



A,0

Beispiel: Breitensuche mit Kantengewichten

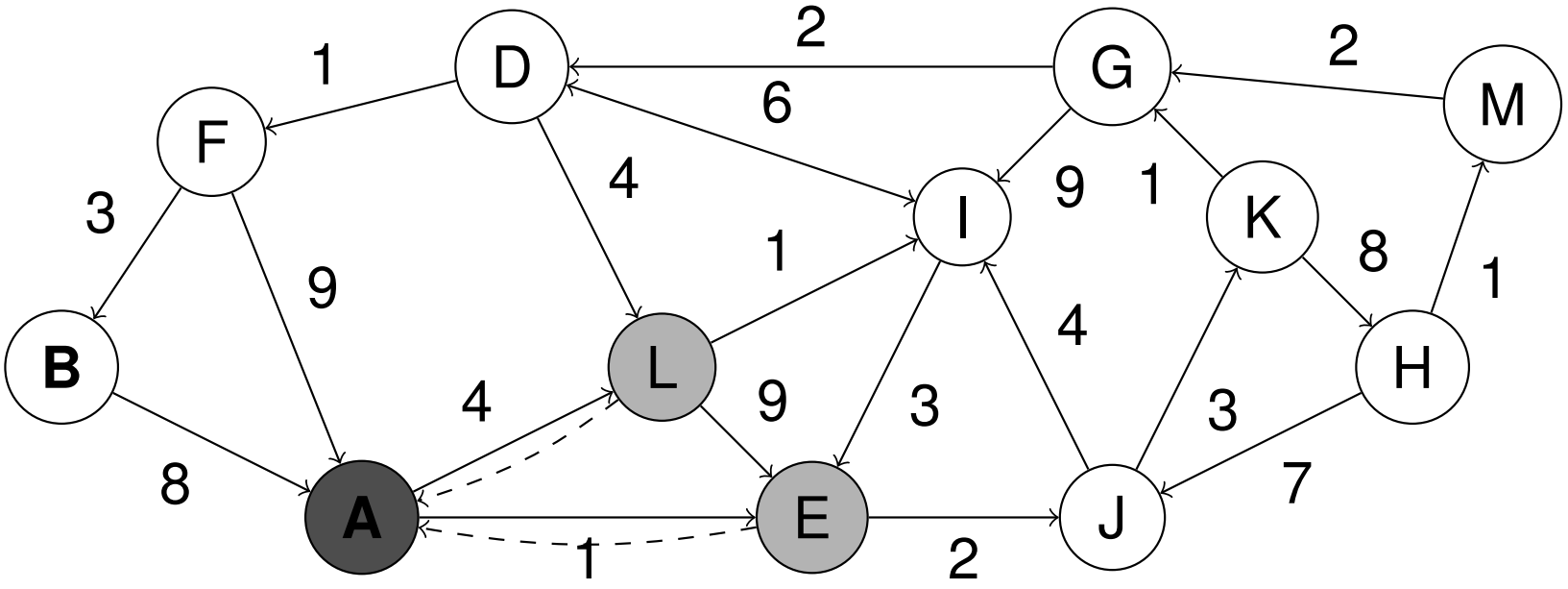
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



A,0	E,1	L,4

Beispiel: Breitensuche mit Kantengewichten

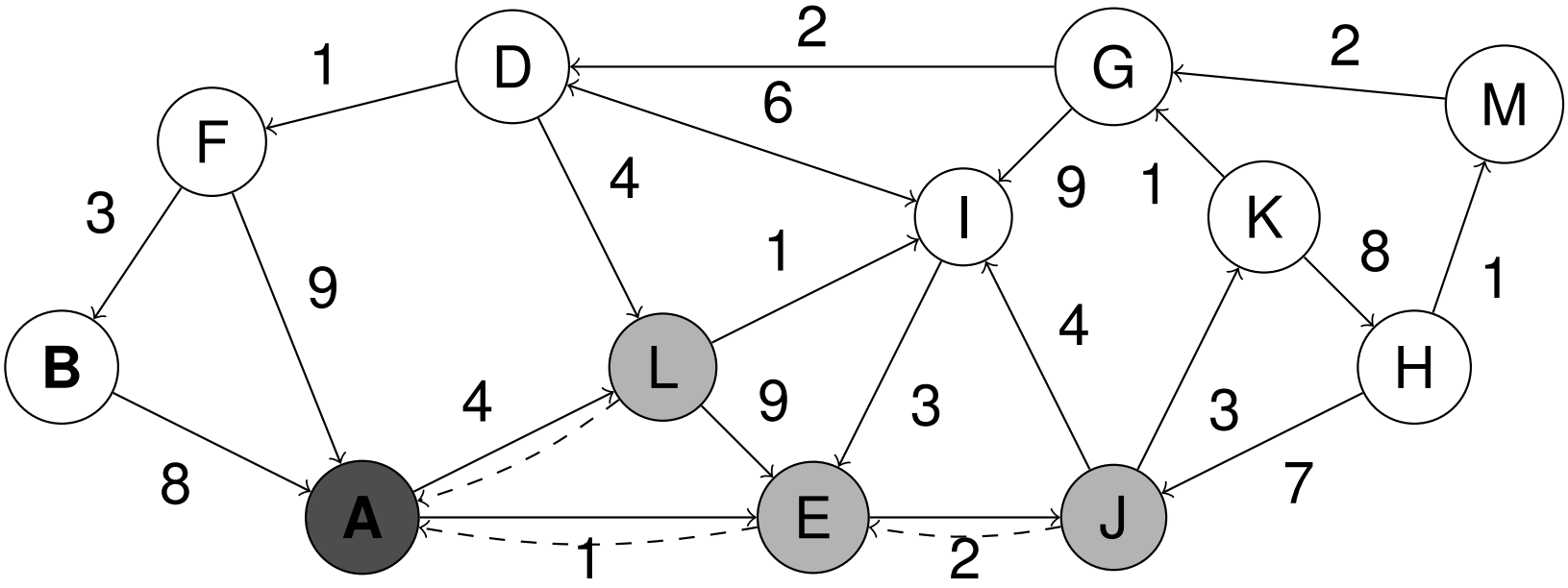
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



E,1	L,4

Beispiel: Breitensuche mit Kantengewichten

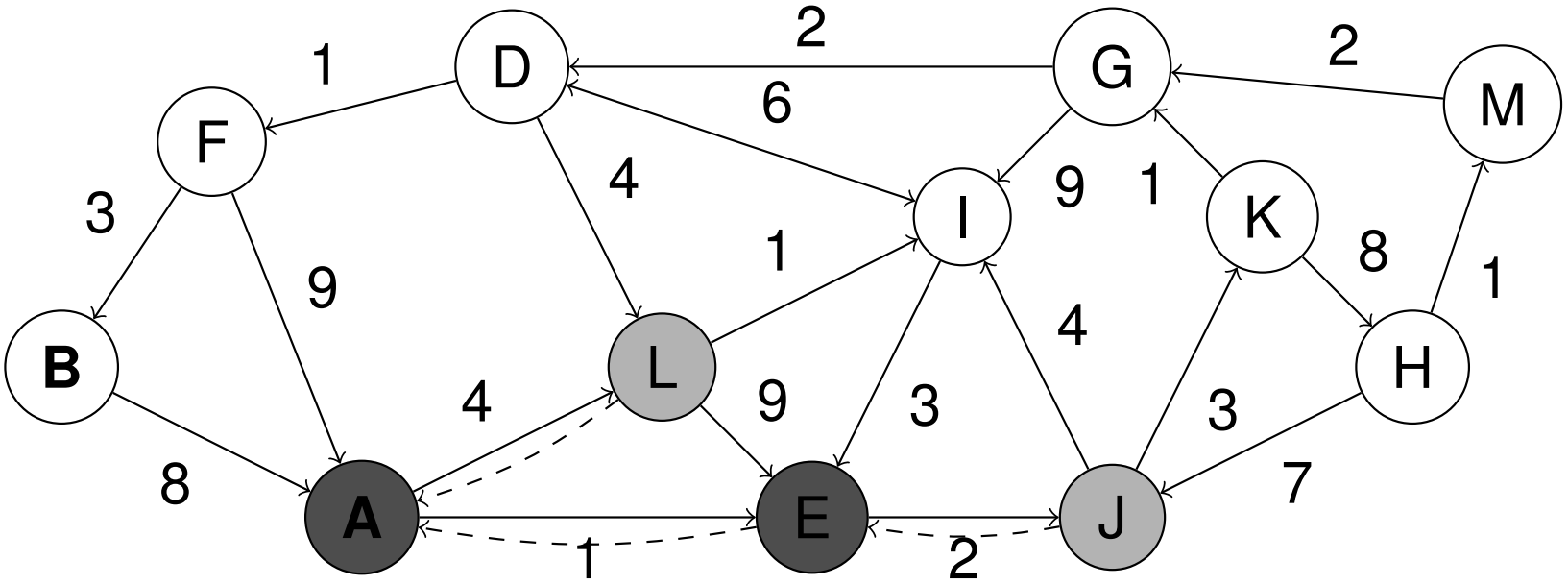
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



E,1	J,3	L,4

Beispiel: Breitensuche mit Kantengewichten

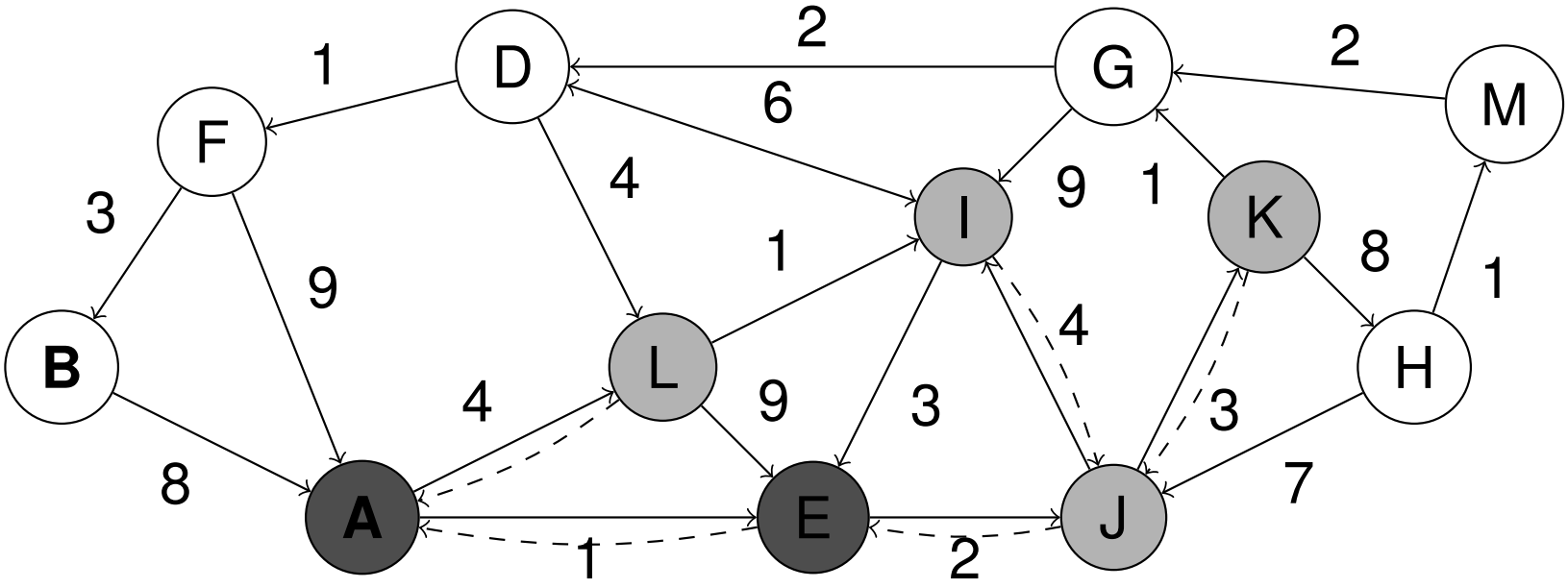
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



J,3	L,4

Beispiel: Breitensuche mit Kantengewichten

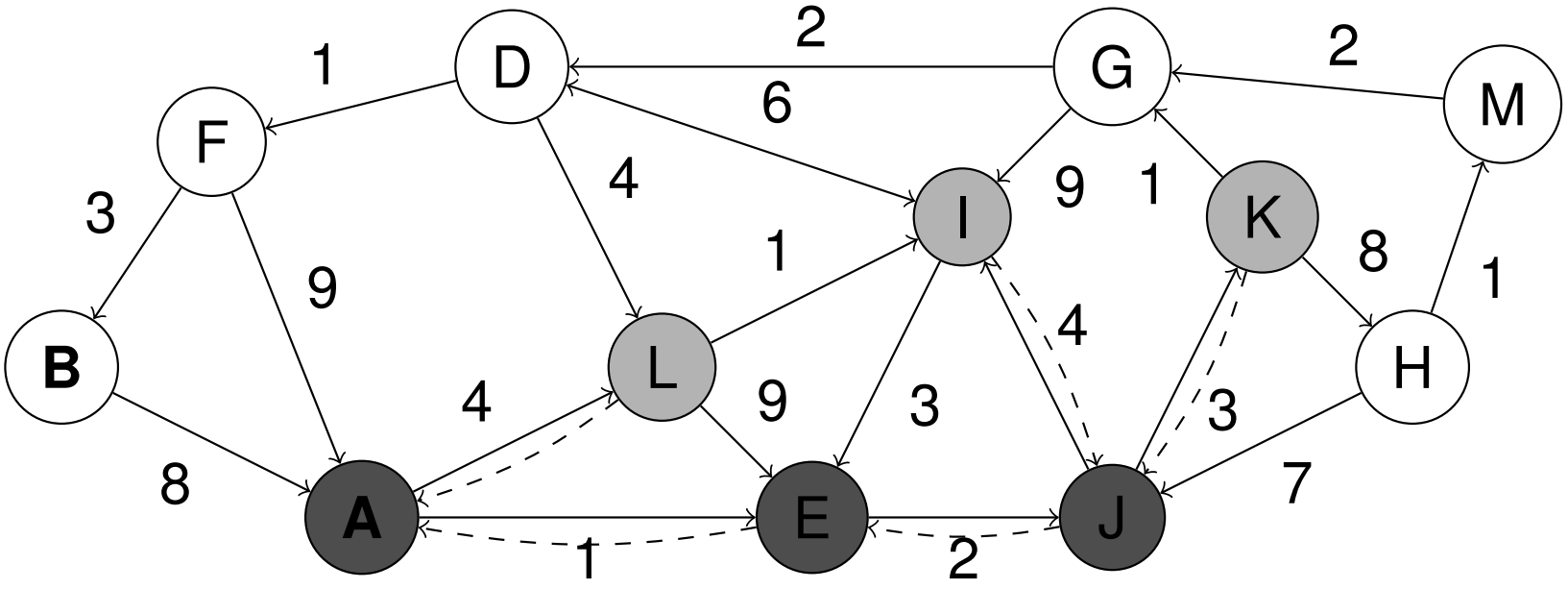
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



J,3	L,4	K,6	I,7

Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten

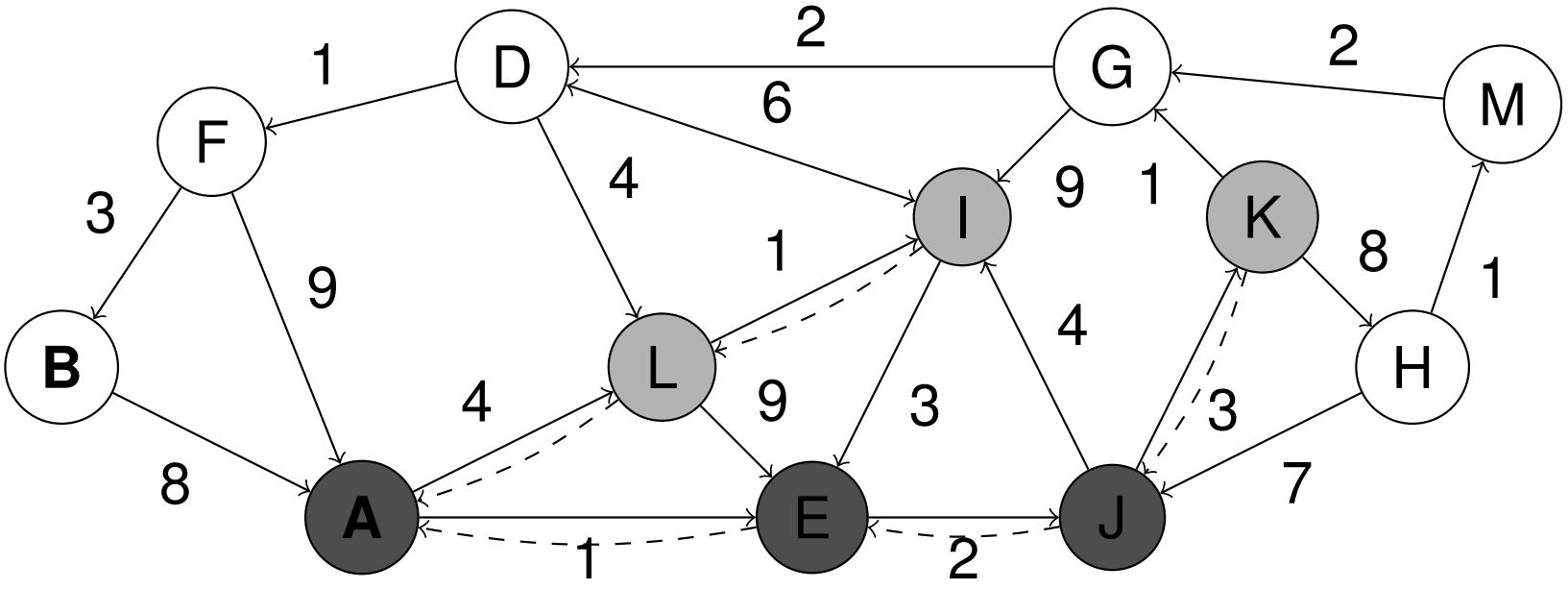


L,4	K,6	I,7

Kürzerer Weg nach I gefunden!
Herkunftsknoten für I muss aktualisiert werden.

Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten

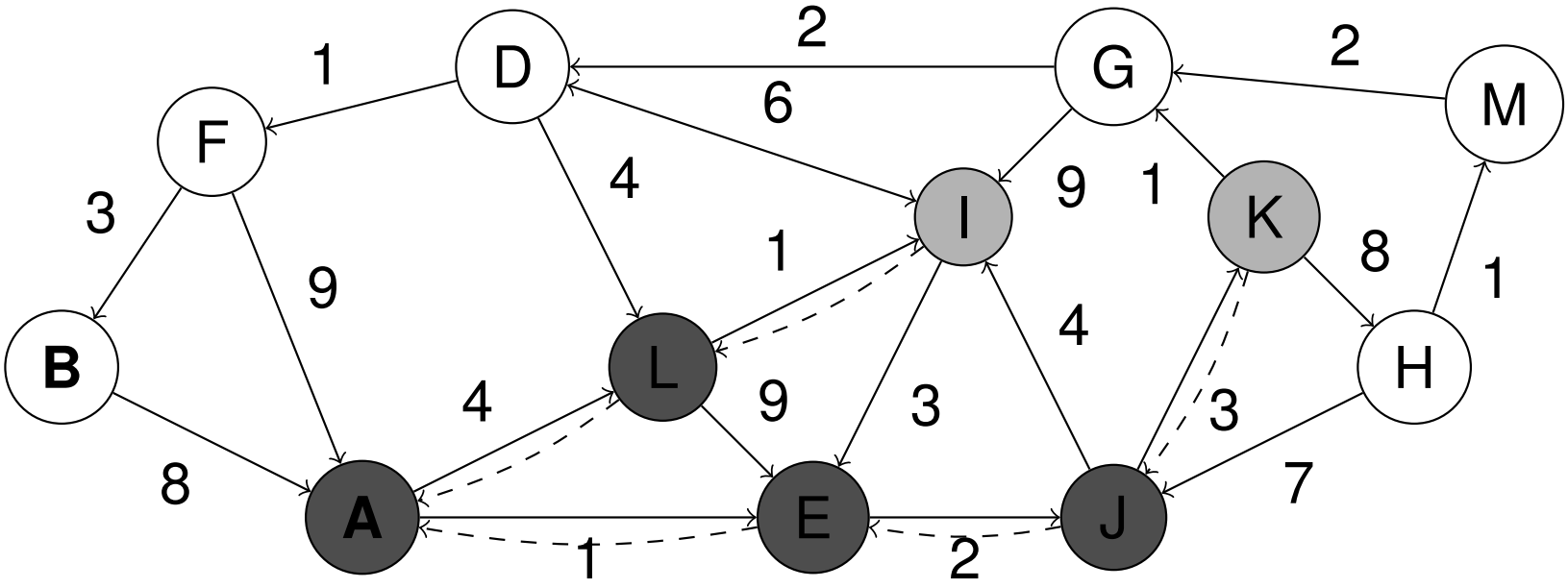


L,4	I,5	K,6

Kürzerer Weg nach I gefunden!
Herkunftsknoten für I muss aktualisiert werden.

Beispiel: Breitensuche mit Kantengewichten

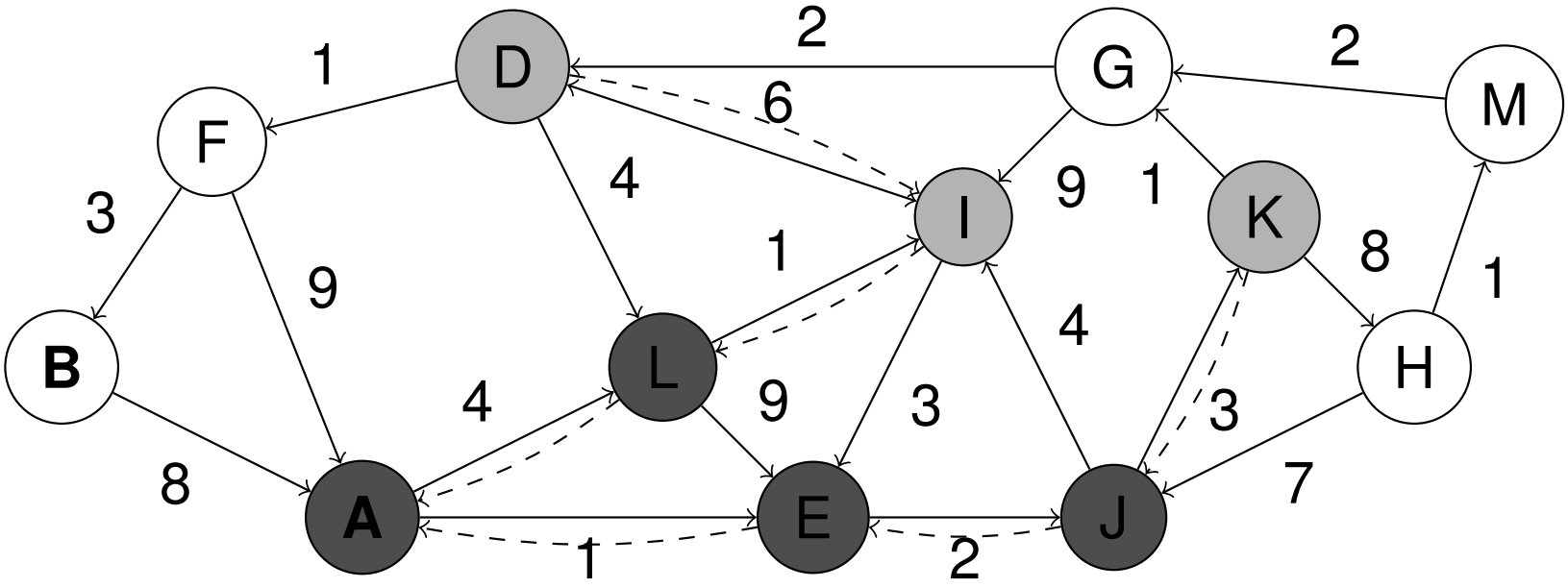
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



I,5	K,6

Beispiel: Breitensuche mit Kantengewichten

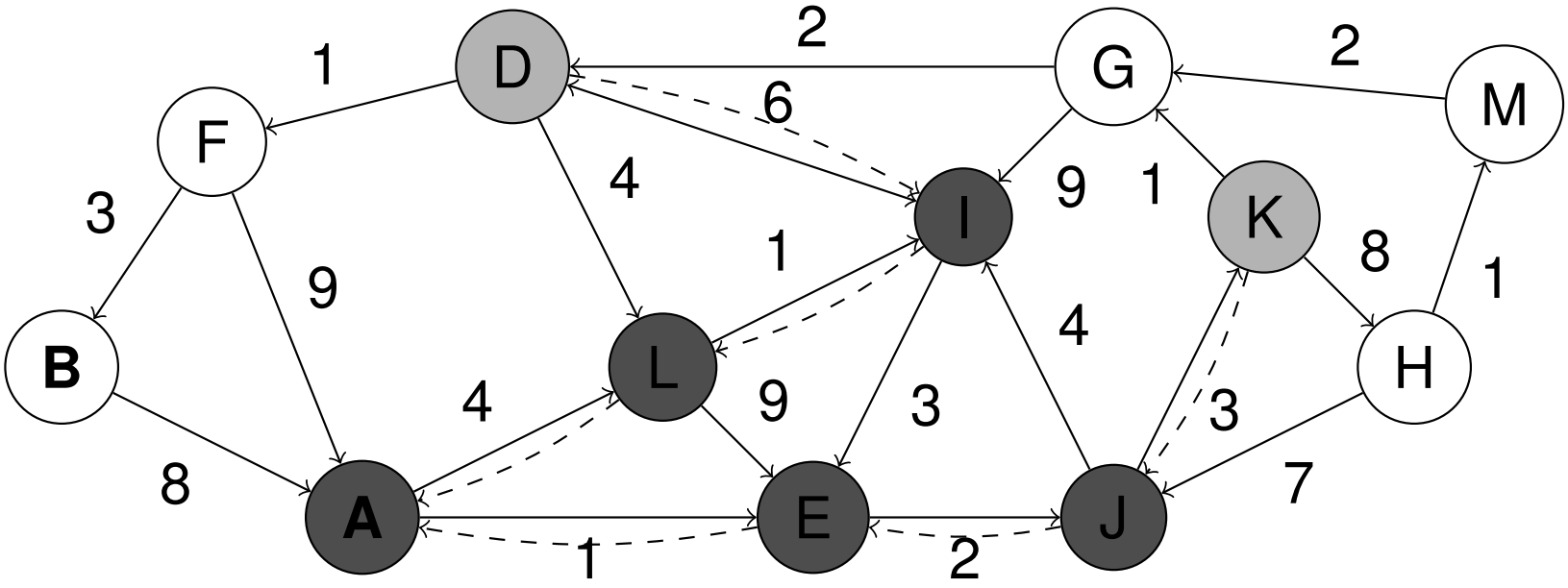
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



I,5	K,6	D,11

Beispiel: Breitensuche mit Kantengewichten

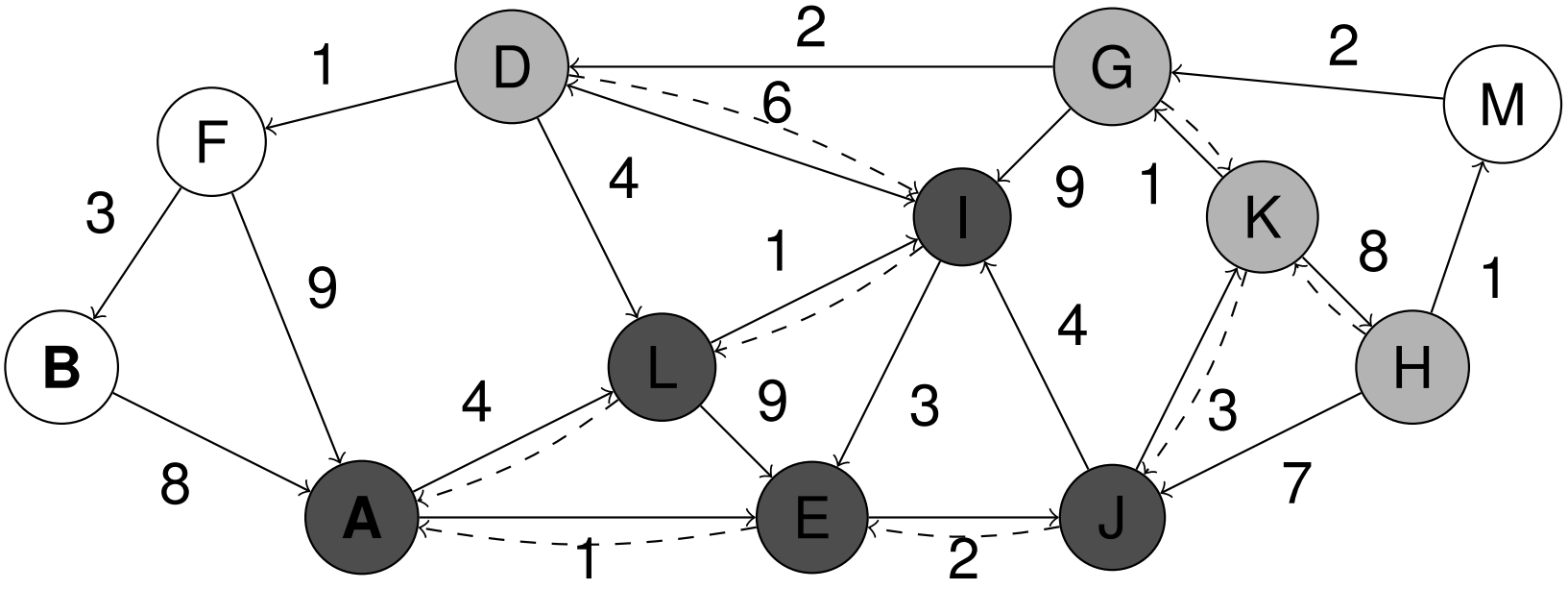
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



K,6	D,11

Beispiel: Breitensuche mit Kantengewichten

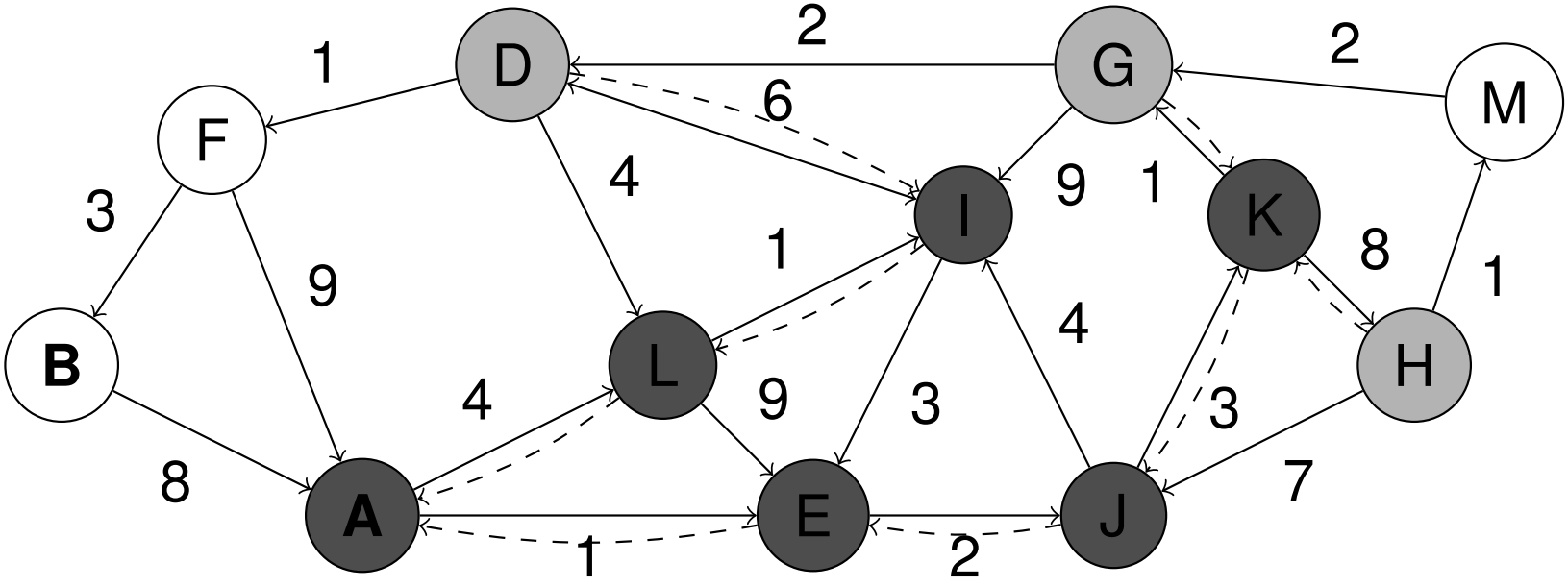
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



K,6	G,7	D,11	H,14

Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten

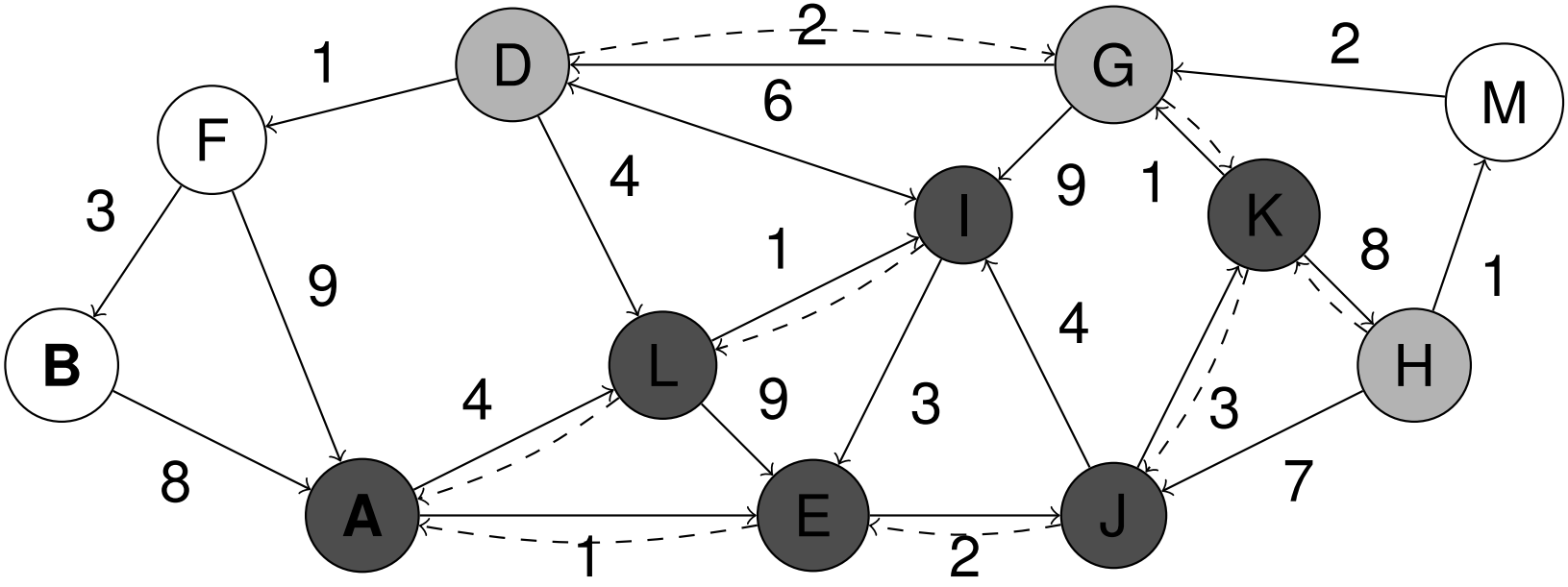


G,7	D,11	H,14

Kürzerer Weg nach D gefunden!
Herkunftsknoten für D muss aktualisiert werden.

Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten

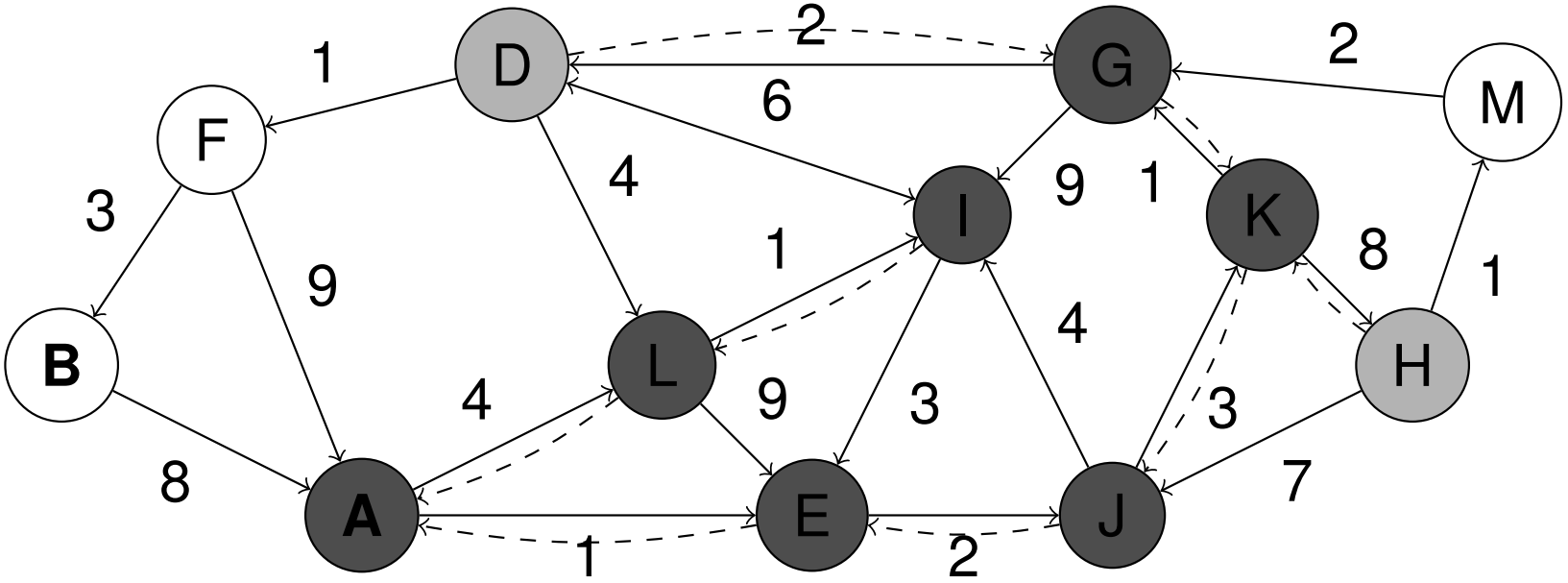


G,7	D,9	H,14

Kürzerer Weg nach D gefunden!
Herkunftsknoten für D muss aktualisiert werden.

Beispiel: Breitensuche mit Kantengewichten

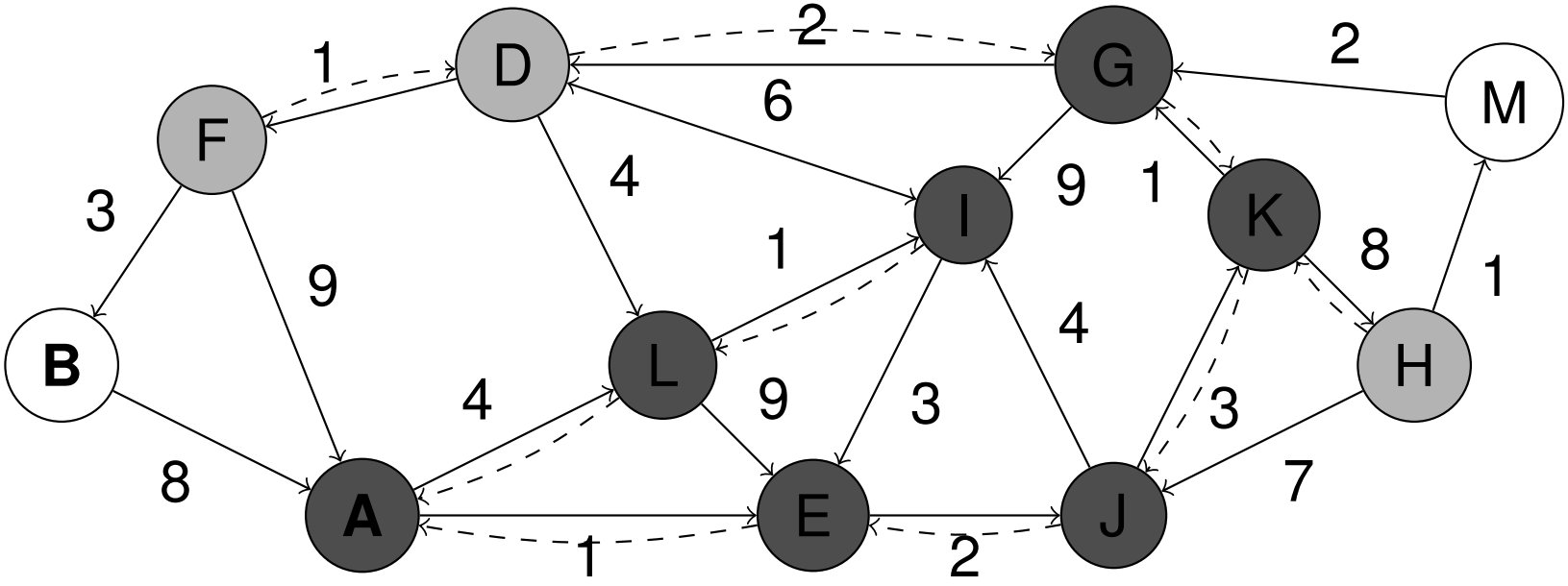
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



D,9	H,14

Beispiel: Breitensuche mit Kantengewichten

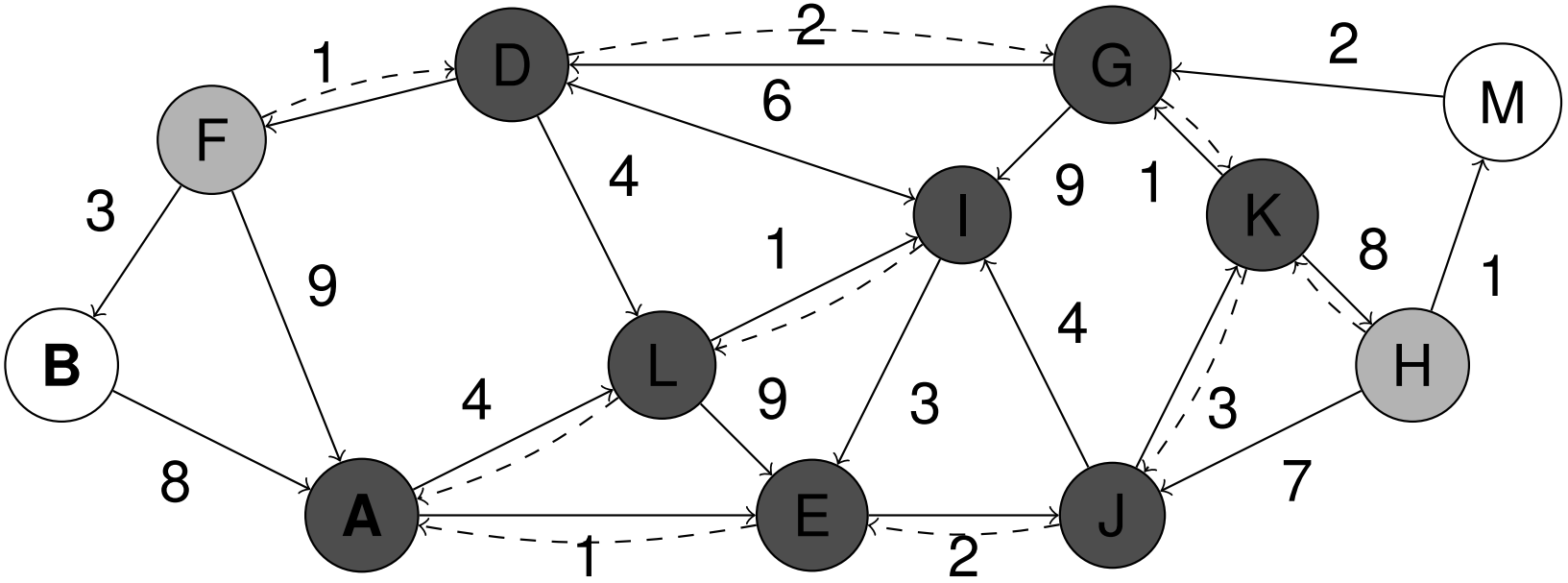
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



D,9	F,10	H,14

Beispiel: Breitensuche mit Kantengewichten

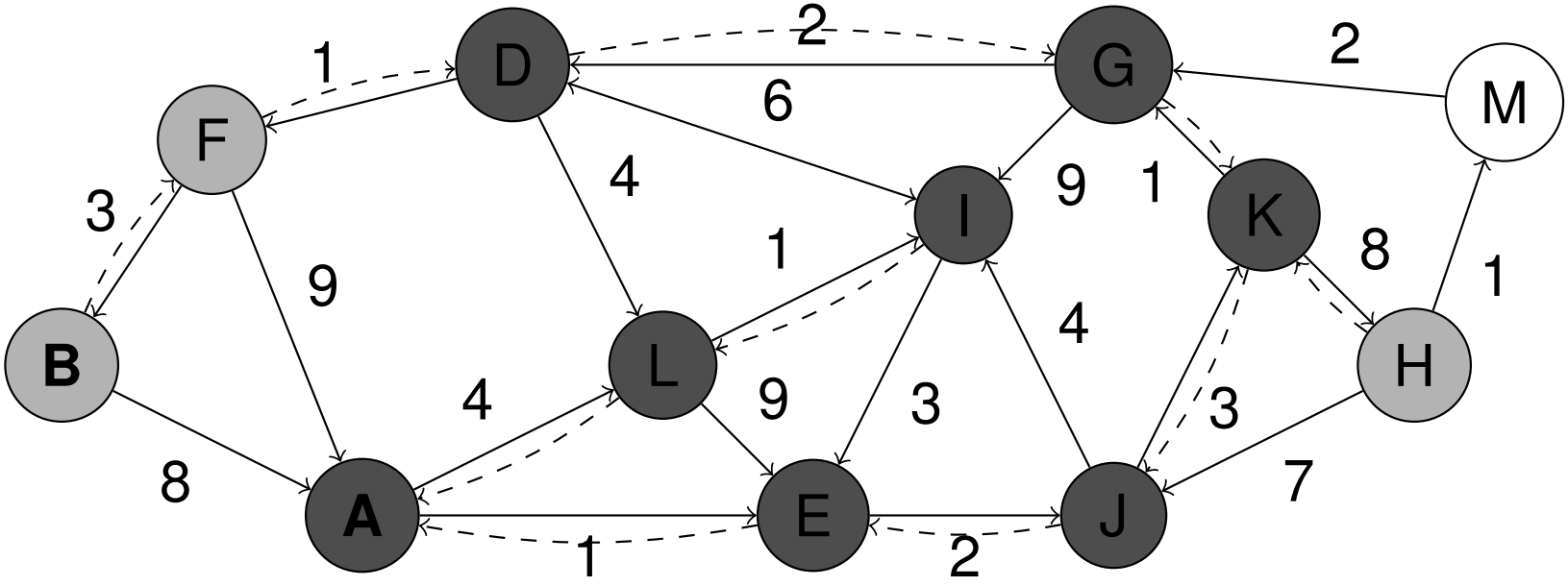
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



F,10	H,14

Beispiel: Breitensuche mit Kantengewichten

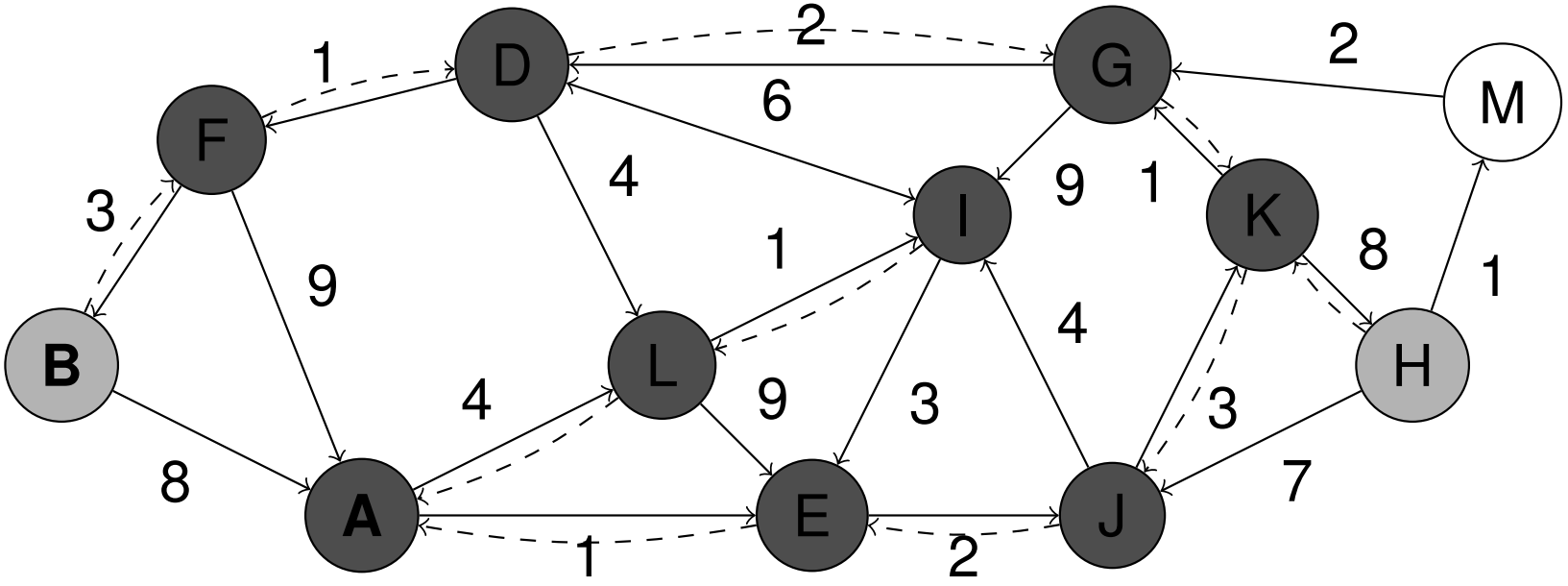
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



F,10	B,13	H,14

Beispiel: Breitensuche mit Kantengewichten

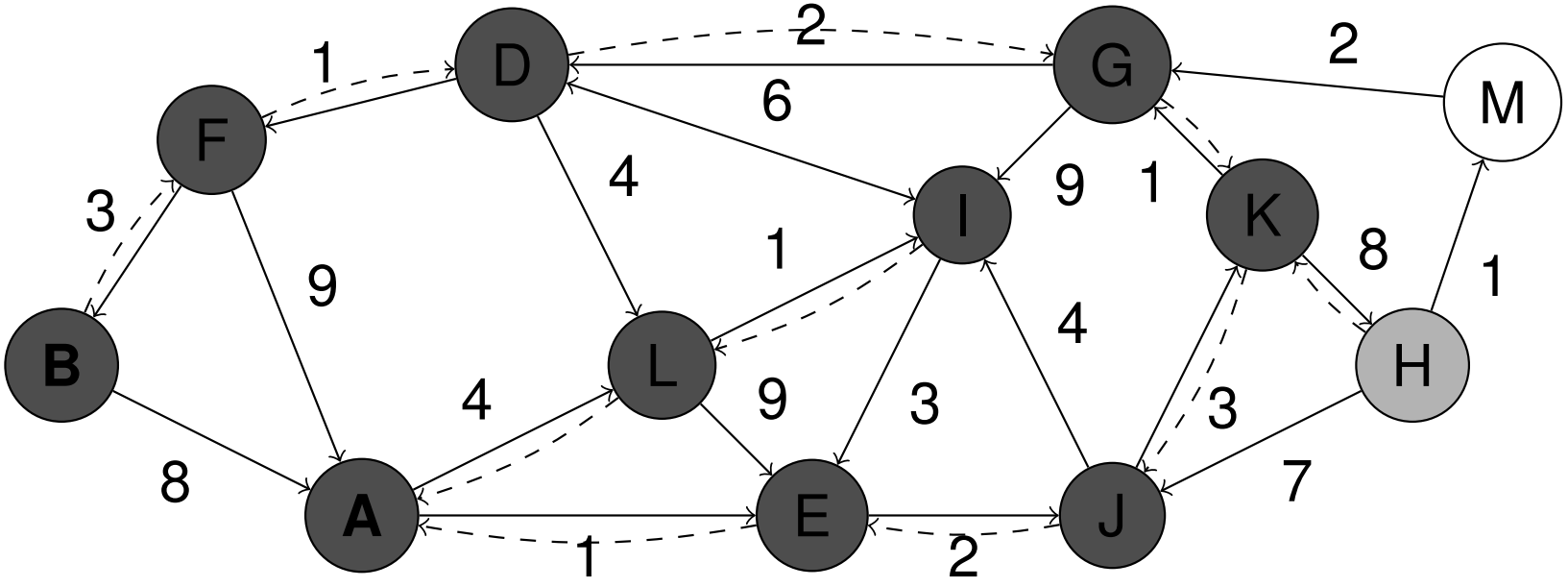
Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



B,13	H,14

Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



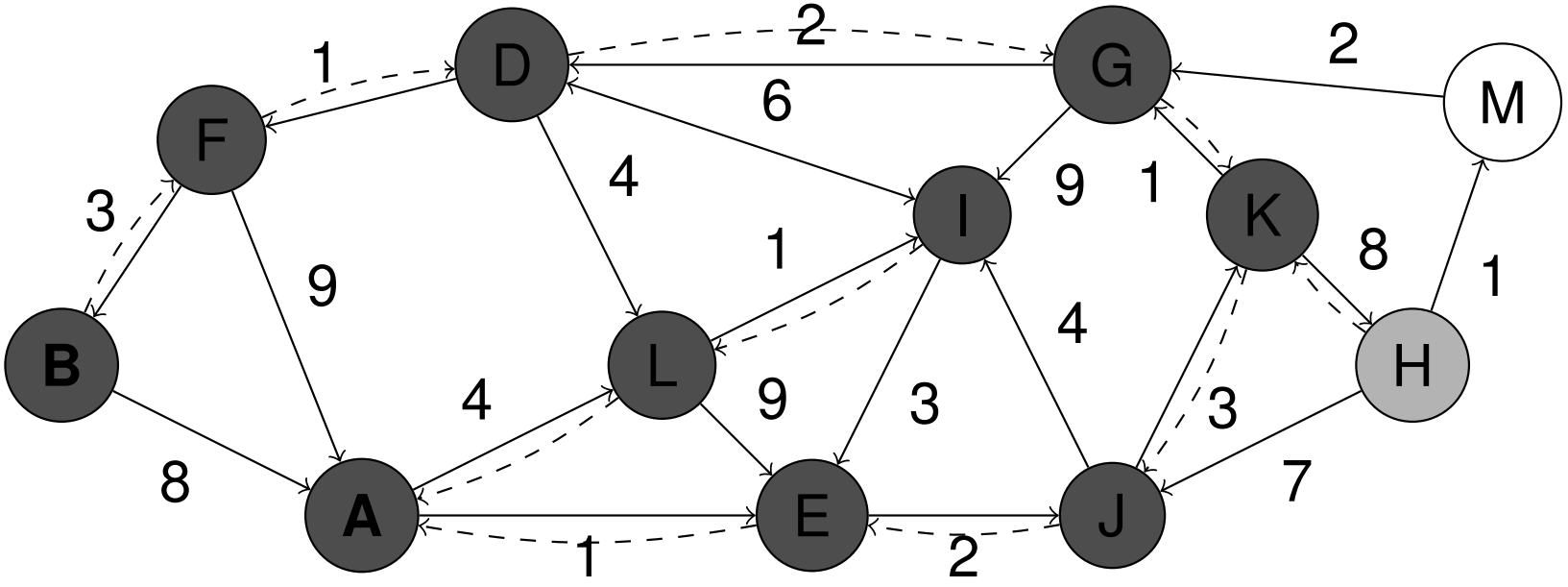
B,13	H,14

Ende jetzt erst, wenn der Zielknoten schwarz gefärbt ist!

D.h. Zielknoten B steht am Anfang der Warteschlange

Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten

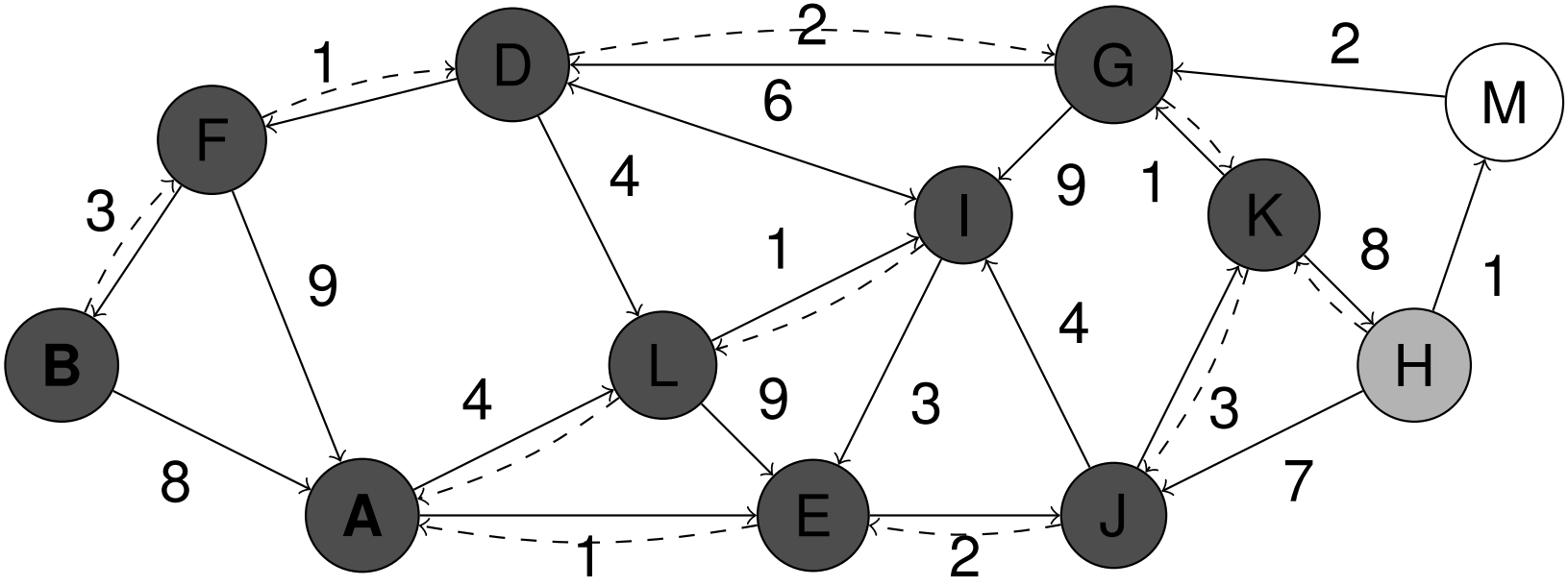


B,13	H,14

Ende jetzt erst, wenn der Zielknoten schwarz gefärbt ist!
D.h. Zielknoten B steht am Anfang der Warteschlange
Kürzester gewichteter Weg: A-E-J-K-G-D-F-B, Länge 13

Beispiel: Breitensuche mit Kantengewichten

Ähnliches Beispiel nochmal, diesmal mit gewichteten Kanten



B,13	H,14

Ende jetzt erst, wenn der Zielknoten schwarz gefärbt ist!
D.h. Zielknoten B steht am Anfang der Warteschlange
Kürzester gewichteter Weg: A-E-J-K-G-D-F-B, Länge 13

Änderung am Algorithmus:

In jedem Schritt muss die Warteschlange nach der Distanz des Knotens sortiert werden $\Rightarrow$ `PriorityQueue`

Interface Queue und Implementierung PriorityQueue

Eine `Queue` ist eine *einseitige* Warteschlange (FIFO), entsprechend ist `Queue` ist Superinterface von `Deque`

- Enthält eine Reihe an Elementen
- Nur am Ende kann ein Element eingefügt werden `add`
- Nur am Anfang kann ein Element entfernt werden `remove`

Eine `PriorityQueue` ist eine sehr spezielle Implementierung von `Queue`, da diese Elemente der Warteschlange stets gemäß einer Vergleichsoperation sortiert hält.

Die Elemente der `PriorityQueue` müssen dazu das Interface `Comparable` implementieren, um die Reihenfolge festzulegen.

Dazu ist es meist hilfreich, eine spezielle Wrapper-Klasse zu schreiben...

Beispiel: Paar-Wrapper für PriorityQueue

Wir bilden einfach ein Paar aus dem zu speichernden Knoten und seiner Priorität, was z.B. die Weglänge sein kein.

```
public class PrioPair implements Comparable<PrioPair>{
  final GraphNode node;
  final Integer priority;

  public PrioPair(GraphNode node, Integer priority) {
    this.node = node;
    this.priority = priority;
  }

  @Override
  public int compareTo(PrioPair other) {
    return (this.priority - other.priority);
  }
}
```

Je nachdem ob die `PriorityQueue` auf- oder absteigend sortiert sein soll, führt man die Subtraktion umgekehrt aus.

Beispiel: Generischer Paar-Wrapper für PriorityQueue

Wir bilden einfach ein Paar aus dem zu speichernden Objekt und seiner Priorität, was z.B. die Weglänge sein kein.

```
public class PrioPair<N> implements Comparable<PrioPair<N>>{
  final N node;
  final Integer priority;

  public PrioPair(N node, Integer priority) {
    this.node = node;
    this.priority = priority;
  }

  @Override
  public int compareTo(PrioPair<N> other) {
    return (this.priority - other.priority);
  }
}
```

Je nachdem ob die `PriorityQueue` auf- oder absteigend sortiert sein soll, führt man die Subtraktion umgekehrt aus.

Wegsuche mit A^*

Eine weitere Optimierung der Wegsuche bietet der weit verbreitete A^*-Algorithmus.

Benötigt eine **Heuristik**, welche für ein Wegstück die *minimale* Restdistanz zum Ziel abschätzt.

In unserem Fall ist diese Abschätzung einfach: Die Luftlinien-Distanz vom Ende des Wegstücks zum Ziel. Diese ist garantiert kürzer als der bestmöglich erreichbare Weg

Gilt dies nicht, dann braucht man eine andere monotone Abschätzung!

D.h. die Priorität eines Knotens entspricht dann einfach der bisher errechneten Distanz vom Startknoten aus *plus* der Luftliniendistanz zum Zielknoten.

Damit ist es wahrscheinlicher, dass viel mehr Knoten weiß bleiben (also nicht berechnet werden müssen).