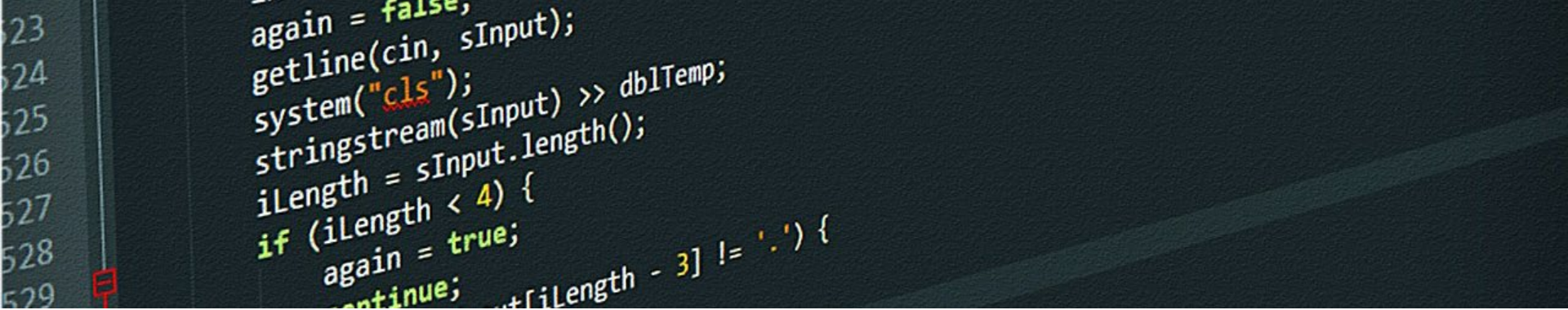


Thomas

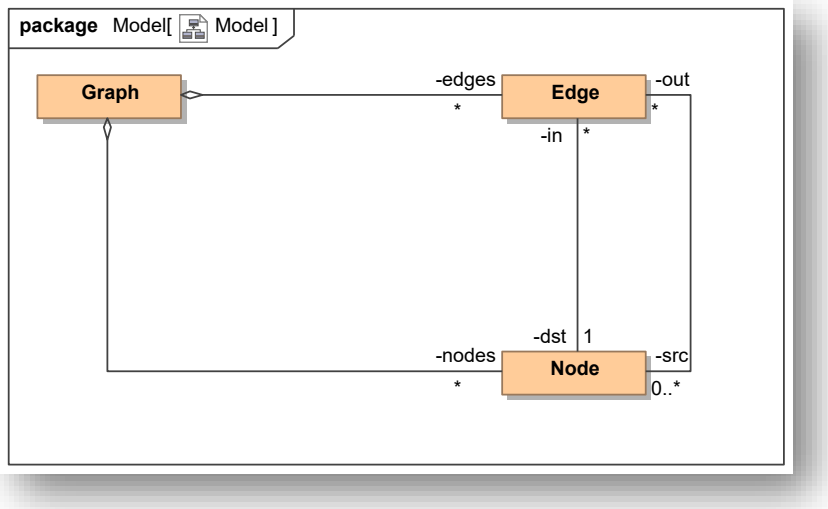
C23-11.1 Graphen, Dijkstra

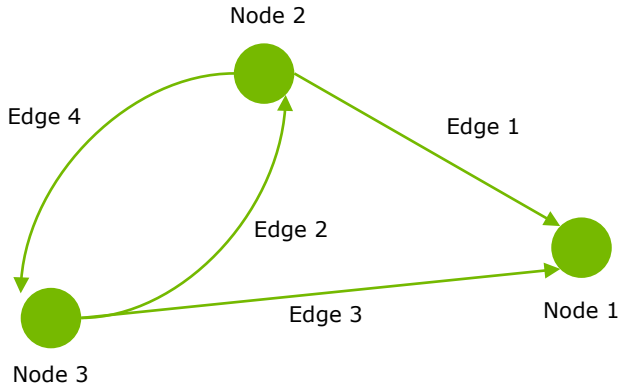
Fortgeschrittene Algorithmen und Programmierung

v4

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

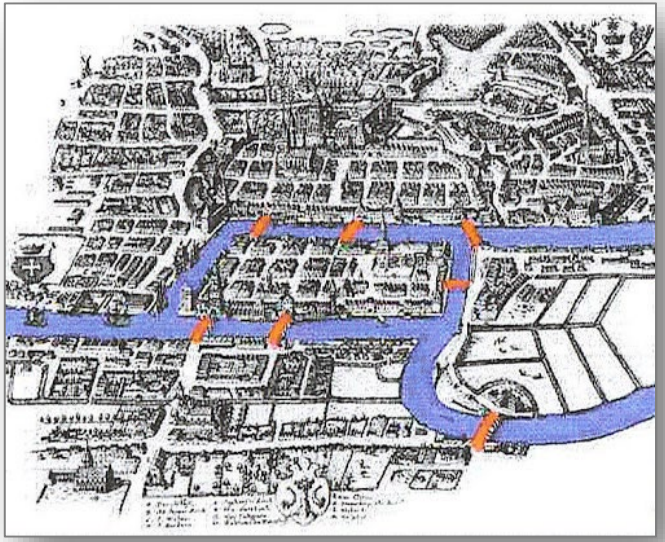
Graphen
Übersicht



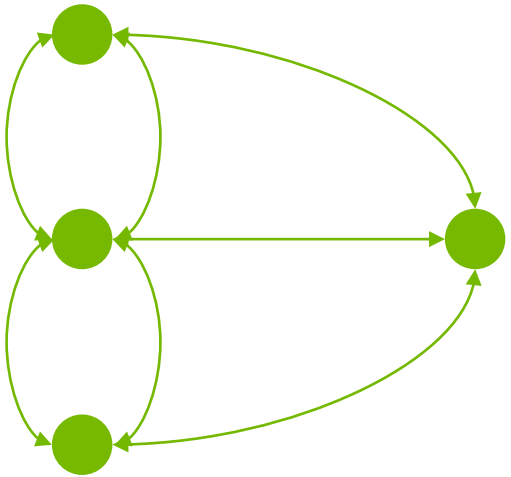


Graphen

Historie: Königsberger Brückenproblem



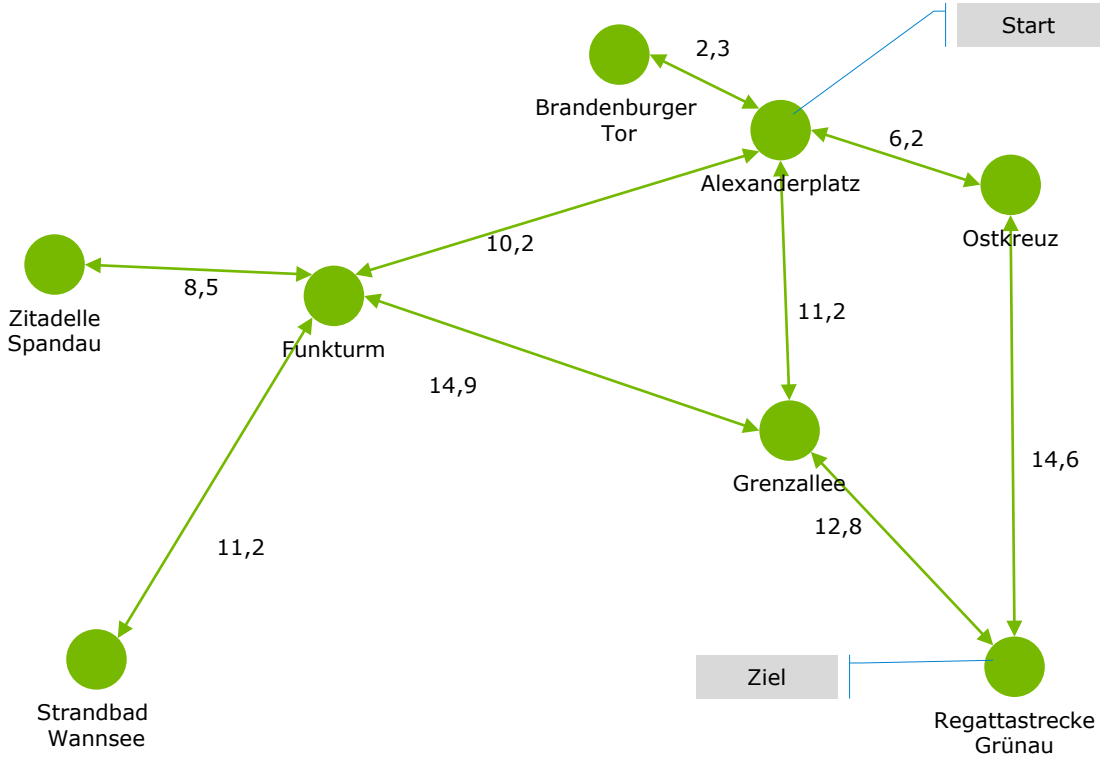
[friderizianer.de]



Die Abbildung zeigt die Stadt Königsberg im 18. Jahrhundert. Die beiden Arme des Flusses Pregel umfließen die Insel Kneiphof. Es gibt insgesamt sieben Brücken über den Fluss.
Frage: Gibt es einen Rundweg, der jede Brücke genau einmal benutzt?

Dijkstra-Algorithmus (Kürzester Pfad)

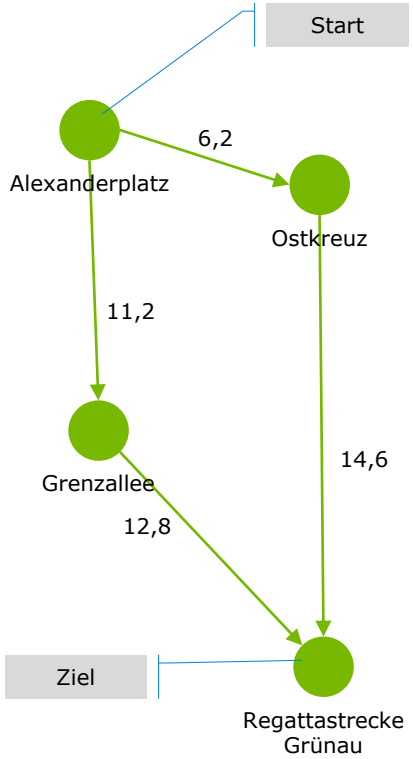
Start




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Dijkstra-Algorithmus (Kürzester Pfad)

Start – Vereinfachung zur Demonstration



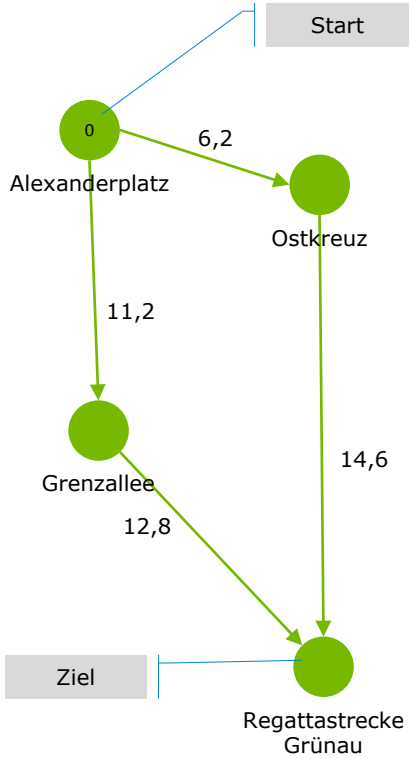

```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 1a – Initialisierung

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Der Abstand vom Startknoten zu sich selbst wird auf 0 gesetzt.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Dijkstra-Algorithmus (Kürzester Pfad)

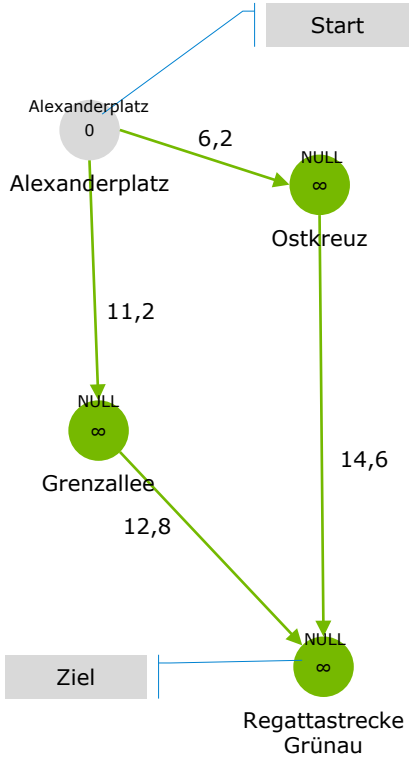
Schritt 1b – Initialisierung

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Als Abstand zu allen anderen Knoten wird ein Maximalabstand von unendlich angenommen. Der Algorithmus versucht, diese Abstände zu verringern.

Der Vorgängerknoten des Startknotens ist er selbst und für alle anderen Knoten „NULL", also ein unbekannter Wert.

Der Startknoten wird in eine Warteschlange eingefügt, dies ist eine Liste von Knoten, wobei die Knoten, deren Abstände zum Startknoten geringer sind, weiter vorne stehen.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

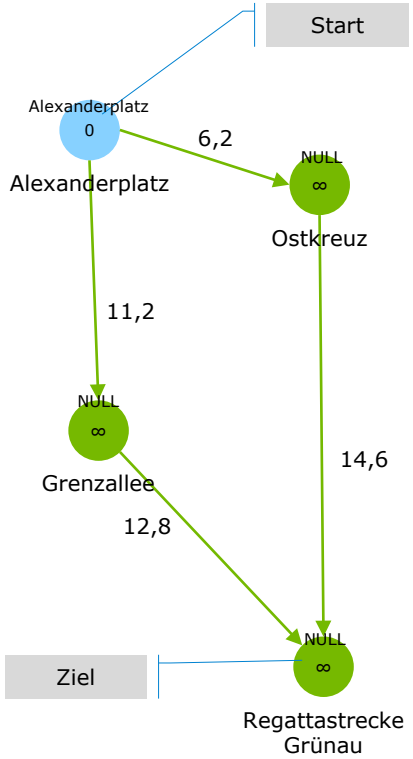
- Alexanderplatz

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 2a – Zum nächsten Knoten vorrücken

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Der Knoten mit dem geringsten Abstand wird aus der Warteschlange entnommen und wird nun bearbeitet, ist also blau markiert.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

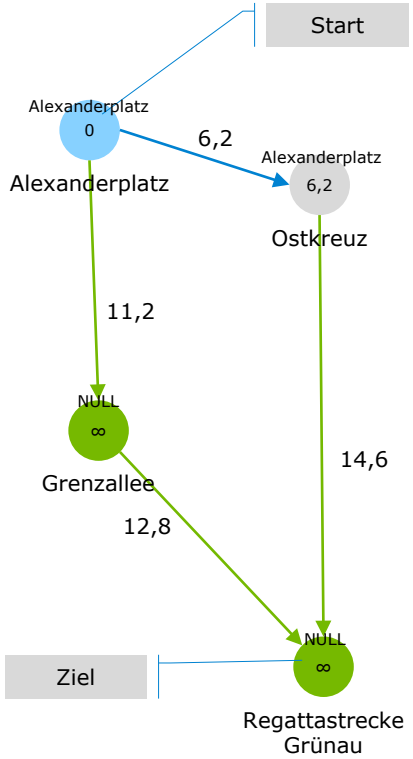
•

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 2b – Nachbarknoten betrachten

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Für jeden Nachbarknoten wird die Distanz zum Startknoten berechnet, wenn der Weg über die aktuelle Kante führt.
Die Distanz zum Startknoten ergibt sich aus dem Abstand des Vaterknotens plus dem Gewicht der Kante.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

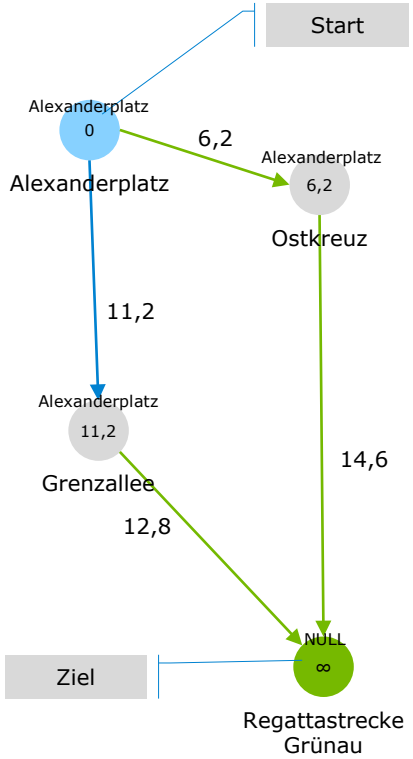
Warteschlange:
- Ostkreuz

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 2c – Nachbarknoten betrachten

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Für jeden Nachbarknoten wird die Distanz zum Startknoten berechnet, wenn der Weg über die aktuelle Kante führt.
Die Distanz zum Startknoten ergibt sich aus dem Abstand des Vaterknotens plus dem Gewicht der Kante.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

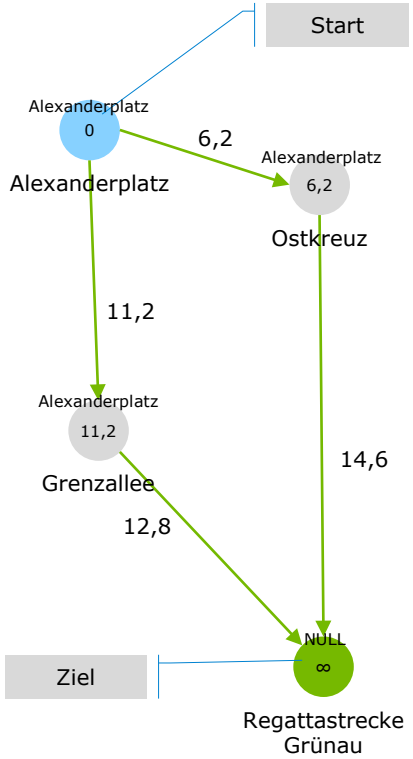
Warteschlange:
- Ostkreuz
- Grenzallee

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 2d – Nachbarknoten betrachten

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Alle ausgehenden Kanten des Knotens wurden betrachtet




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

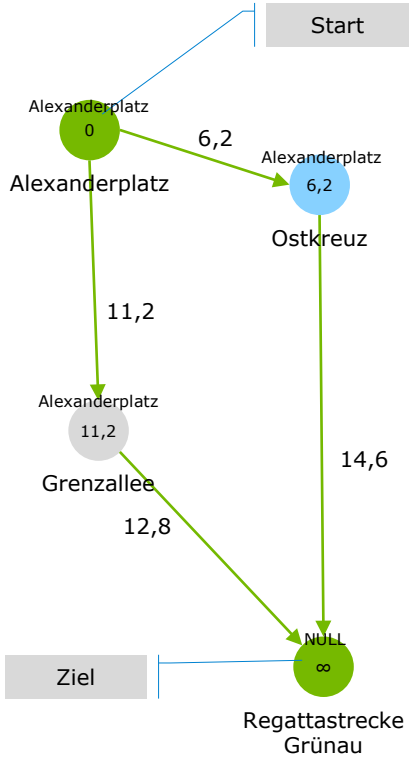
- Ostkreuz
- Grenzallee

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 3a – Zum nächsten Knoten vorrücken

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Der Knoten mit dem geringsten Abstand wird aus der Warteschlange entnommen.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

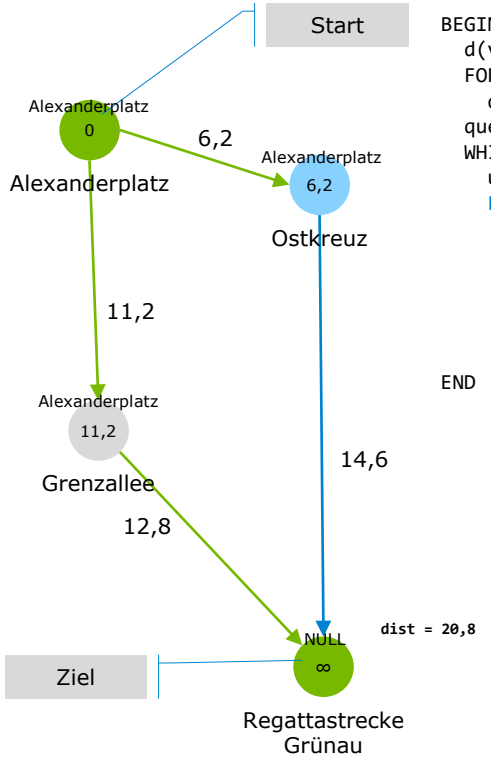
- Grenzallee

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 3b – Nachbarknoten betrachten

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Für jeden Nachbarknoten wird die Distanz zum Startknoten berechnet, wenn der Weg über die aktuelle Kante führt.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

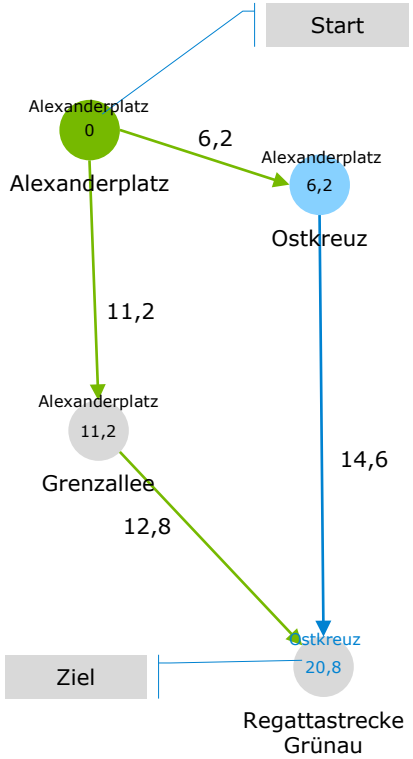
- Grenzallee

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 3c – Nachbarknoten betrachten

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Für jeden Nachbarknoten wird die Distanz zum Startknoten berechnet, wenn der Weg über die aktuelle Kante führt.
Die Distanz zum Startknoten ergibt sich aus dem Abstand des Vaterknotens plus dem Gewicht der Kante.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

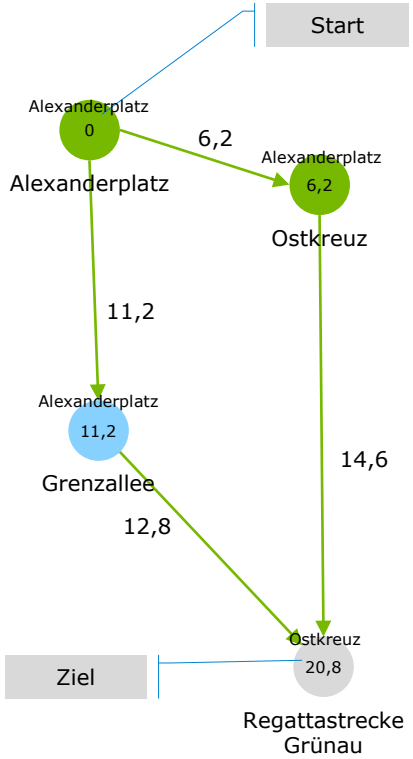
- Grenzallee
- Regattastrecke Grünau

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 4a – Zum nächsten Knoten vorrücken

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Der Knoten mit dem geringsten Abstand wird aus der Warteschlange entnommen.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

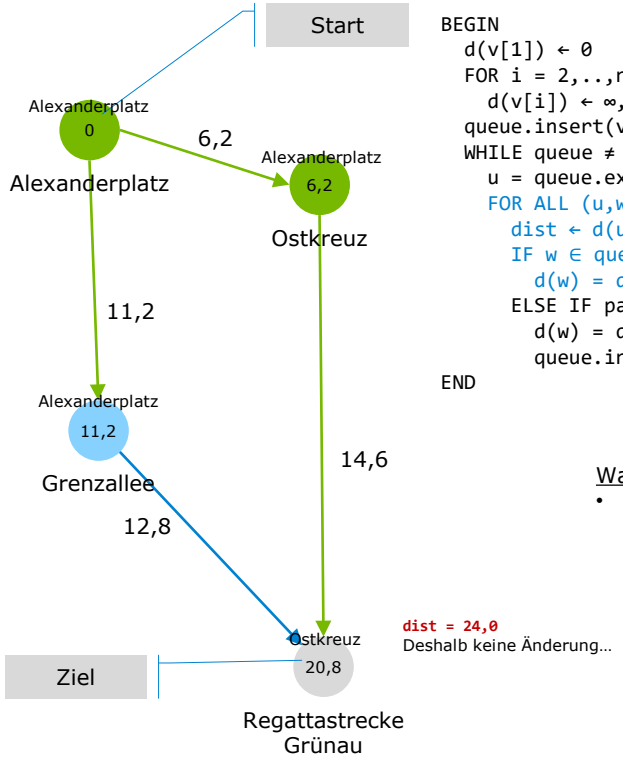
- Regattastrecke Grünau

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 4b – Nachbarknoten betrachten

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Der Knoten mit dem geringsten Abstand wird aus der Warteschlange entnommen.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

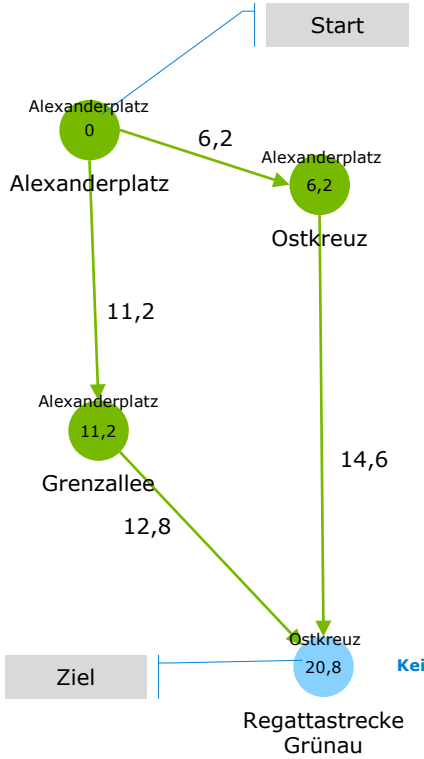
- Regattastrecke Grünau

Dijkstra-Algorithmus (Kürzester Pfad)

Schritt 5a – Zum nächsten Knoten vorrücken

Aktueller Knoten: u	Startknoten: v
Konten in der Warteschlange	Abstand vom Startknoten: d
	Nachbarknoten: w
Kante zum Nachbarknoten: u, w	Länge (Gewicht) der Kante zum Nachbarknoten: l(u,w)

Der Knoten mit dem geringsten Abstand wird aus der Warteschlange entnommen.




```
BEGIN
  d(v[1]) ← 0
  FOR i = 2,..,n DO
    d(v[i]) ← ∞, parent(v[i]) ← NULL
  queue.insert(v,0)
  WHILE queue ≠ ∅ DO
    u = queue.extractMin()
    FOR ALL (u,w) ∈ E DO
      dist ← d(u) + l(u,w)
      IF w ∈ queue AND d(w) > dist DO
        d(w) = dist, parent(w) = u
      ELSE IF parent(w) == NULL THEN
        d(w) = dist, parent(w) = u
        queue.insert(w,dist)
END
```

Warteschlange:

•

Keine Ausgangskanten → Algorithmus beendet

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de