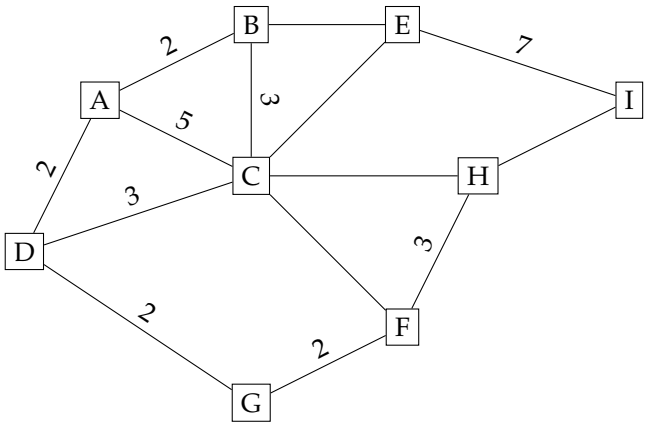
# Dijkstra-Algorithmus[1]

Führen Sie auf dem gegebenen Graphen die Suche nach der kürzesten Distanz aller Knoten zum Startknoten *A* mit dem Algorithmus von Dijkstra durch. Tragen Sie die Abarbeitungsreihenfolge, den unmittelbaren Vorgängerknoten, sowie die ermittelte kürzeste Distanz für jeden Knoten ein! Bei gleichen Distanzen arbeiten Sie die Knoten in lexikalischer Reihenfolge ab.




| Nr. | besucht | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | A | **0** | 2 | 5 | 2 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | B | \| | **2** | 5 | 2 | 3 | ∞ | ∞ | ∞ | ∞ |
| 3 | D | \| | \| | 5 | **2** | 3 | ∞ | 4 | ∞ | ∞ |
| 4 | E | \| | \| | 4 | \| | **3** | ∞ | 4 | ∞ | 10 |
| 5 | C | \| | \| | **4** | \| | \| | 5 | 4 | 5 | 10 |
| 6 | G | \| | \| | \| | \| | \| | 5 | **4** | 5 | 10 |
| 7 | F | \| | \| | \| | \| | \| | **5** | \| | 5 | 10 |
| 8 | H | \| | \| | \| | \| | \| | \| | \| | **5** | 6 |
| 9 | I | \| | \| | \| | \| | \| | \| | \| | \| | **6** |

[1] **aud:e-klausur**.

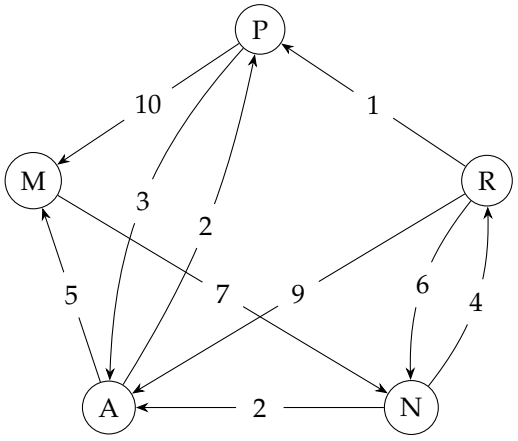
| nach | Entfernung | Reihenfolge | Pfad |
|---|---|---|---|
| A → A | 0 | 0 | |
| A → B | 2 | 2 | A → B |
| A → C | 4 | 5 | A → B → E → C |
| A → D | 2 | 3 | A → D |
| A → E | 3 | 4 | A → B → E |
| A → F | 5 | 7 | A → B → E → C → F |
| A → G | 4 | 6 | A → D → G |
| A → H | 5 | 8 | A → B → E → C → H |
| A → I | 6 | 9 | A → B → E → C → H → I |

## gerichteter Distanzgraph angegeben durch Adjazenzmatrix[2]

Ein gerichteter Distanzgraph sei durch seine Adjazenzmatrix gegeben (in einer Zeile stehen die Längen der von dem Zeilenkopf ausgehenden Wege.)[3]

$$\begin{array}{c} \\ M \\ A \\ P \\ R \\ N \end{array} \begin{array}{c} \begin{array}{ccccc} M & A & P & R & N \end{array} \\ \begin{pmatrix} - & 5 & 10 & - & - \\ - & - & 3 & 9 & 2 \\ - & 2 & - & 1 & - \\ - & - & - & - & 4 \\ 7 & - & - & 6 & - \end{pmatrix} \end{array}$$

(a) Stellen Sie den Graph in der üblichen Form dar.




[2] **aud:ab:6**.
[3] **aud:ab:6**.

(b) Bestimmen Sie mit dem Algorithmus von Dijkstra ausgehend von $M$ die kürzeste Wege zu allen anderen Knoten.

**Nach der Methode von Prof. Dr. Oliver Lazar**

| Schritt | Betrachteter Knoten | Kosten A | Kosten P | Kosten R | Kosten N |
|---|---|---|---|---|---|
| Initial | M | | | | 7 |
| 1 | N | 9 | | 11 | 7 |
| 2 | A | 9 | 11 | 11 | 7 |
| 3 | P | 9 | 11 | 11 | 7 |
| 4 | R | 9 | 11 | 11 | 7 |

**Nach der Methode aus der Vorlesung**

**Besuchte Knoten:** M

| Knoten-Name | M | A | P | R | N |
|---|---|---|---|---|---|
| Distanz | 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| Vorgänger | null | null | null | null | null |

**Besuchte Knoten:** M, N

| Knoten-Name | M | A | P | R | N |
|---|---|---|---|---|---|
| Distanz | 0 | $\infty$ | $\infty$ | $\infty$ | 7 |
| Vorgänger | null | null | null | null | M |

**Besuchte Knoten:** M, N, A

| Knoten-Name | M | A | P | R | N |
|---|---|---|---|---|---|
| Distanz | 0 | 9 | $\infty$ | $\infty$ | 7 |
| Vorgänger | null | N | null | null | M |

**Besuchte Knoten:** M, N, A, P

| Knoten-Name | M | A | P | R | N |
|---|---|---|---|---|---|
| Distanz | 0 | 9 | 11 | $\infty$ | 7 |
| Vorgänger | null | N | A | null | M |

**Besuchte Knoten:** M, N, A, P, R

| Knoten-Name | M | A | P | R | N |
|---|---|---|---|---|---|
| Distanz | 0 | 9 | 11 | 11 | 7 |
| Vorgänger | null | N | A | N | M |

**Ergebnis**

$M \to N = 7$

$M \to N \to A = 9$

$M \to N \to A \to P = 11$

$M \to N \to R = 11$

(c) Beschreiben Sie wie ein Heap als Prioritätswarteschlange in diesem Algorithmus verwendet werden kann.

> Die Prioritätswarteschlange kann dazu verwendet werden, den Knoten mit der kürzesten Distanz schnell zu finden. Eine Prioritätswarteschlange kann zum Beispiel durch eine Min-Heap realisiert werden. Wenn eine Min-Heap aufgebaut wird, ist das Minium immer das Wurzelelement. Es kann sehr einfach und schnell entnommen werden. Der Aufbau einer Min-Heap geht mit linearem Zeitaufwand $\mathcal{O}(n)$ vonstatten. Die Entnahme des Miniums schlägt im schlechtesten Fall mit einem Aufwand von $\mathcal{O}(\log n)$ zu Buche schlagen.

(d) Geben Sie die Operation *„Entfernen des Minimums“* für einen Heap an. Dazu gehört selbstverständlich die Restrukturierung des Heaps.

> Bei der Entnahme des Minimums wird an dessen Stelle das am Ende der Halde sich befindende Element gesetzt.
>
> Das neue Minimum verletzt unter Umständen die Heap-Eigenschaften, wenn eines oder beide seiner Kind-Knoten kleiner sind. Es muss mit dem kleinsten Kind-Knoten getauscht werden. Diese Prozedur wird rekursiv so lange ausgeführt, bis die Heap-Eigenschaften wieder hergestellt sind. Man nennt diesen Vorgang auch `heapify`.
>
> Es kann aber auch sein das das verschobene Element kleiner ist. Dann muss die gegenteilige Operation von `heapify` ausführt werden, die `decrease` genannt wird.

## Flugverbindung zwischen sieben Städten[4]

Nehmen Sie an, es gibt sieben Städte A, B, C, D, E, F und G. Sie wohnen in der Stadt A und möchten zu jeder der anderen Städte die preiswerteste Flugverbindung finden (einfach ohne Rückflug). Sie sind dazu bereit, beliebig oft umzusteigen. Folgende Direktflüge stehen Ihnen zur Verfügung:

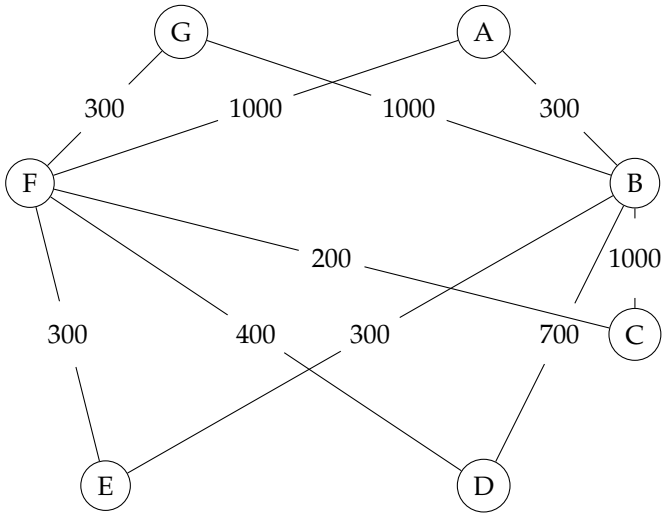
[4] **aud:ab:6**.

| Städte | Preis |
|---|---|
| A ↔ B | 300 € |
| A ↔ F | 1000 € |
| B ↔ C | 1000 € |
| B ↔ D | 700 € |
| B ↔ E | 300 € |
| B ↔ G | 1000 € |
| C ↔ F | 200 € |
| D ↔ F | 400 € |
| E ↔ F | 300 € |
| F ↔ G | 300 € |

Der Preis p in einer Zeile

| Städte | Preis |
|---|---|
| x ↔ y | p |

gilt dabei sowohl für einen einfachen Flug von x nach y als auch für einen einfachen Flug von y nach x. Bestimmen Sie mit dem Algorithmus von Dijkstra (führen Sie den Algorithmus händisch durch!) die Routen und die Preise für die preiswertesten Flugverbindungen von der Stadt A zu jeder der anderen Städte.



| Schritt | besuchte Knoten | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|
| Init | | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | **A** | 0 | **300,A** | ∞ | ∞ | ∞ | 1000,F | ∞ |
| 2 | A,**B** | 0 | \| | 1300,B | 1000,B | **600,B** | 1000,F | 1300,B |
| 3 | A,B,**E** | 0 | \| | 1300,B | 1000,B | \| | **900,E** | 1300,B |
| 4 | A,B,E,**F** | 0 | \| | 1100,F | **1000,B** | \| | \| | 1200,F |
| 5 | A,B,E,F,**D** | 0 | \| | **1100,F** | \| | \| | \| | 1200,F |
| 6 | A,B,E,F,D,**C** | 0 | \| | \| | \| | \| | \| | **1200,F** |
| 7 | A,B,E,F,D,C,**G** | 0 | \| | \| | \| | \| | \| | \| |

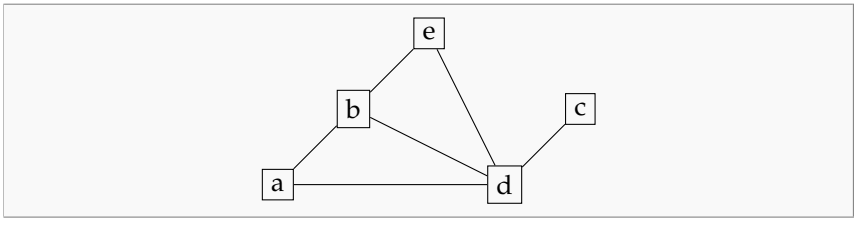
| Städte | Preis |
|---|---|
| A → B | 300 |
| A → B → E → F → C | 1100 |
| A → B → D | 1000 |
| A → B → E | 600 |
| A → B → E → F | 900 |
| A → B → E → F → G | 1200 |

## Aufgabe 2[5]

Gegeben sei folgender Graph:

V: {a, b, c, d, e}
E: a → a, b
b → b, d, e
c → c, d
d → a, e

(a) Stellen Sie den Graphen grafisch dar!




(b) Berechnen Sie mit dem Algorithmus von Dijkstra schrittweise die Länge der kürzesten Pfade ab dem Knoten a! Nehmen Sie dazu an, dass alle Kantengewichte 1 sind. Erstellen Sie eine Tabelle gemäß folgendem Muster:

[5] **46114:2008:09**.

ausgewählt |a| b c d e

Ergebnis:

Hinweis: Nur mit Angabe der jeweiligen Zwischenschritte gibt es Punkte. Es reicht also nicht, nur das Endergebnis hinzuschreiben.

| Nr. | ausgewählt | a | b | c | d | e |
|---|---|---|---|---|---|---|
| 1 | a | **0** | 1 | ∞ | 1 | ∞ |
| 2 | b | \| | **1** | ∞ | 1 | 2 |
| 3 | d | \| | \| | 2 | **1** | 2 |
| 4 | c | \| | \| | **2** | \| | 2 |
| 5 | e | \| | \| | \| | \| | **2** |

(c) Welchen Aufwand hat der Algorithmus von Dijkstra bei Graphen mit |V| Knoten und |E| Kanten,

- wenn die Kantengewichte alle 1 sind? Mit welcher Datenstruktur und welchem Vorgehen lässt sich der Aufwand in diesem Fall reduzieren (mit kurzer Begründung)?
- wenn die Kantengewichte beliebig sind und als Datenstruktur eine Halde verwendet wird (mit kurzer Begründung)?

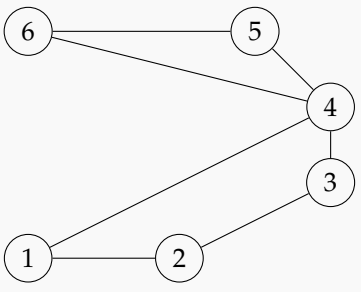
## Aufgabe 10: Graphen I[6]

Gegeben seien folgende ungerichtete Graphen in textueller Notation, wobei die erste Menge die Menge der Knoten und die zweite Menge die Menge der Kanten ist:[7]

$G_1 = (\{1,2,3,4,5,6\}, \{[1,2],[1,4],[2,3],[3,4],[4,5],[4,6],[5,6]\})$
$G_2 = (\{1,2,3,4,5,6\}, \{[1,2],[1,3],[2,3],[4,5],[4,6],[5,6]\})$
$G_3 = (\{1,2,3,4,5,6\}, \{[1,2],[1,6],[2,3],[2,5],[3,4],[4,5],[5,6]\})$

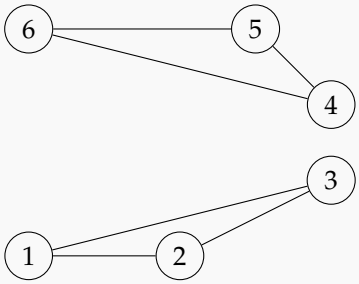
(a) Zeichnen Sie die obigen Graphen.

- $G_1$

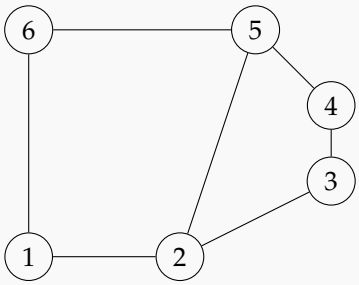



[6] **aud:pu:7**.
[7] **examen:46115:2013:03**.

- $G_2$




- $G_3$




(b) Erstellen Sie zu jedem Graphen die zugehörige Adjazenzmatrix mit $X$ als Symbol für eine eingetragene Kante.

- $G_1$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} - & X & & X & & \\ X & - & X & & & \\ & X & - & X & & \\ X & & X & - & X & X \\ & & & X & - & X \\ & & & X & X & - \end{pmatrix} \end{array}$$

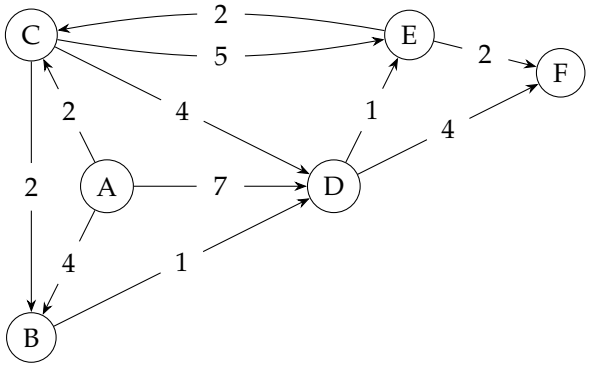
- $G_2$

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{pmatrix}
- & X & X & & & \\
X & - & X & & & \\
X & X & - & & & \\
 & & & - & X & X \\
 & & & X & - & X \\
 & & & X & X & -
\end{pmatrix}
\end{array}
$$

- $G_3$

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{pmatrix}
- & X & & & X & X \\
X & - & X & & & \\
 & X & - & X & & \\
 & & X & - & X & \\
 & X & & X & - & X \\
X & & & & X & -
\end{pmatrix}
\end{array}
$$

(c) Betrachten Sie nun folgenden gerichteten Graphen $G_4$:



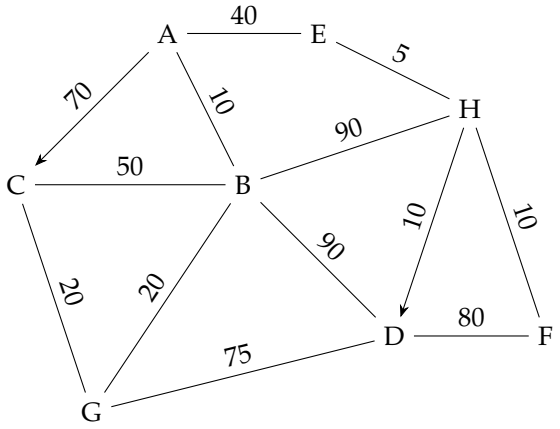

Bestimmen Sie die kürzeste Entfernung von Knoten A zu jedem anderen Knoten des Graphen. Verwenden Sie dazu den Algorithmus von Dijkstra und tragen Sie Ihre einzelnen Rechenschritte in eine Tabelle folgender Form ein (schreiben Sie neben jede Zeile die Prioritätswarteschlange der noch zu bearbeitenden Knoten, priorisiert nach ihren Wegkosten):

Hinweis: Mit den „Wegkosten“ eines Knotens ist die gegenwärtige Entfernung dieses Knotens vom Startknoten gemeint.

| A | B | C | D | E | F | Warteschlange |
|---|---|---|---|---|---|---|
| 0 | ∞ | ∞ | ∞ | ∞ | ∞ | A |
| \| | **4** (A) | **2** (A) | **7** (A) | ∞ | ∞ | C, B, D |
| \| | \| | \| | **6** (C) | **7** (C) | ∞ | B, D, E |
| \| | \| | \| | **5** (B) | 7 (C) | ∞ | D, E |
| \| | \| | \| | \| | **6** (D) | 9 (D) | E, F |
| \| | \| | \| | \| | \| | **8** (E) | F |
| \| | \| | \| | \| | \| | \| | |

## Städte gemischt gerichtet / ungerichtet[8]

Ein wichtiges Problem im Bereich der Graphalgorithmen ist die Berechnung kürzester Wege. Gegeben sei der folgende Graph, in dem Städte durch Kanten verbunden sind. Die Kantengewichte geben Fahrzeiten an. Außer den durch Pfeile als nur in eine Richtung befahrbar gekennzeichneten Straßen sind alle Straßen in beiden Richtungen befahrbar.[9]




(a) Geben Sie zu dem obigen Graphen zunächst eine Darstellung als Adjazenzmatix an.

[8] **examen:66112:2004:03**.
[9] **aud:pu:6**.

$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array}
\begin{array}{c}
\begin{array}{cccccccc} A & B & C & D & E & F & G & H \end{array} \\
\begin{pmatrix}
0 & 10 & 70 & 0 & 40 & 0 & 0 & 0 \\
10 & 0 & 50 & 90 & 0 & 0 & 20 & 90 \\
0 & 50 & 0 & 0 & 0 & 0 & 20 & 0 \\
0 & 90 & 0 & 0 & 0 & 80 & 75 & 0 \\
40 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
0 & 0 & 0 & 80 & 0 & 0 & 0 & 10 \\
0 & 20 & 20 & 75 & 0 & 0 & 0 & 0 \\
0 & 90 & 0 & 10 & 5 & 10 & 0 & 0
\end{pmatrix}
\end{array}
$$

(b) Berechnen Sie nun mit Hilfe des Algorithmus von Dijkstra die kürzesten Wege vom Knoten A zu allen anderen Knoten.

## 9. Aufgabe: Dijkstra[10]

Gegeben sei der unten stehende gerichtete Graph $G = (V, E)$ mit positiven Kantenlingen $l(e)$ für jede Kante $e \in E$. Kanten mit Doppelspitzen können in beide Richtungen durchlaufen werden.

(a) In welcher Reihenfolge werden die Knoten von $G$ ab dem Knoten $a$ durch den Dijkstra-Algorithmus bei der Berechnung der kürzesten Wege endgültig bearbeitet?

(b) Berechnen Sie die Länge des kürzesten Weges von $a$ zu jedem Knoten.

(c) Geben Sie einen kürzesten Weg von $a$ nach $c$ an.

## Aufgabe 7[11]

Auf folgendem ungerichteten, gewichteten Graphen wurde der Dijkstra-Algorithmus (wie auf der nächsten Seite beschrieben) ausgeführt, doch wir wissen lediglich, welcher Knoten als letztes schwarz (black) wurde (Nr. 8) und was seine Distanz zum Startknoten (Nr. 1) ist. Die Gewichte der Kanten sind angegeben.

Finden Sie zunächst den Startknoten, nummerieren Sie anschließend die Knoten in der Reihenfolge, in der sie schwarz wurden, und geben Sie in jedem Knoten die Distanz zum Startknoten an.

Hinweis: Der Startknoten ist eindeutig.

Dijkstra(WeightedGraph G, Vertex s)

```
1 Initialize(G, s);
2 S=ß;
3 Q = new PriorityQueue(V, d) ;
```

[10] **66115:2013:09**.

[11] **66115:2015:03**.

```
4  while not Q.Empty() do
5    u = Q.ExtractMin() ;
6    S = S U {u};
7    foreach v € Adj[u| do
8      Relax(u, v; w);
9
10   u.color = black;
```

Initialize(Graph G, Vertex s)

```
1  foreach u € V do
2    u.color = white;
3    u.d = 00;
4  s.color = gray;
5  s.d = 0;
```

Relax(u, v; w)

```
1  if v.d> u.d+ w(u,v) then
2    v.color = gray;
3    v.d = u.d + w(u,v);
4    Q.DecreaseKey(v, v.d);
```

## Dijkstra Algorithmus[12]

(a) Berechnen Sie für folgenden Graphen den kürzesten Weg von Karlsruhe nach Kassel und dokumentieren Sie den Berechnungsweg:

**Verwendete Abkürzungen:**

**A** Augsburg

**EF** Erfurt

**F** Frankfurt

**KA** Karlsruhe

**KS** Kassel

**M** München

**MA** Mannheim

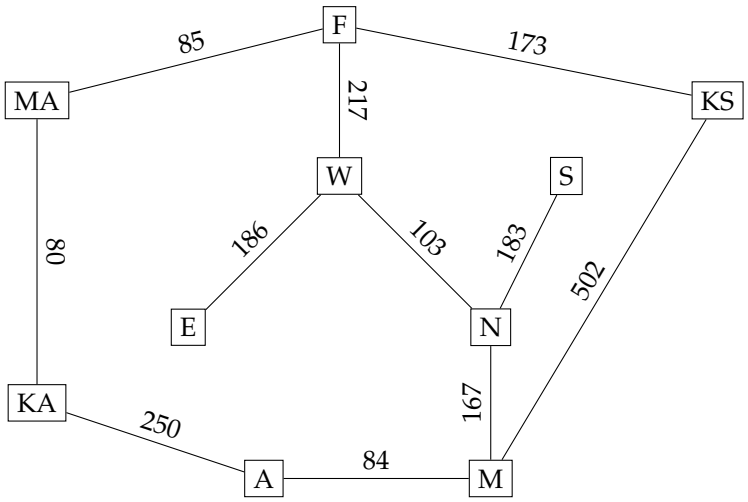
**N** Nürnberg

**S** Stuttgart

**WÜ** Würzburg

Zahl = Zahl in Kilometern

[12] **examen:66115:2016:03**.




| Nr. | besucht | A | E | F | KA | KS | M | MA | N | S | W |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | KA | **250** | ∞ | ∞ | **0** | ∞ | ∞ | 80 | ∞ | ∞ | ∞ |
| 2 | MA | \| | ∞ | 165 | \| | ∞ | ∞ | **80** | ∞ | ∞ | ∞ |
| 3 | F | \| | ∞ | **165** | \| | 338 | ∞ | \| | ∞ | ∞ | 382 |
| 4 | A | \| | ∞ | \| | \| | 338 | 334 | \| | ∞ | ∞ | 382 |
| 5 | M | \| | ∞ | \| | \| | 338 | **334** | \| | 501 | ∞ | 382 |
| 6 | KS | \| | ∞ | \| | \| | **338** | \| | \| | 501 | ∞ | 382 |
| 7 | W | \| | 568 | \| | \| | \| | \| | \| | 485 | ∞ | **382** |
| 8 | N | \| | 568 | \| | \| | \| | \| | \| | **485** | 668 | \| |
| 9 | E | \| | **568** | \| | \| | \| | \| | \| | \| | 668 | \| |
| 10 | S | \| | \| | \| | \| | \| | \| | \| | \| | **668** | \| |

| nach | Entfernung | Reihenfolge | Pfad |
|---|---|---|---|
| KA → A | 250 | 0 | KA → A |
| KA → E | 568 | 9 | KA → MA → F → W → E |
| KA → F | 165 | 3 | KA → MA → F |
| KA → KA | 0 | 1 | |
| KA → KS | 338 | 6 | KA → MA → F → KS |
| KA → M | 334 | 5 | KA → A → M |
| KA → MA | 80 | 2 | KA → MA |
| KA → N | 485 | 8 | KA → MA → F → W → N |
| KA → S | 668 | 10 | KA → MA → F → W → N → S |
| KA → W | 382 | 7 | KA → MA → F → W |

(b) Könnte man den Dijkstra Algorithmus auch benutzen, um das Travelling-Salesman Problem zu lösen?

## Aufgabe 1 (Graphalgorithmen)[13]

Die folgende Abbildung zeigt die wichtigsten bayerischen Autobahnen zusammen mit einigen anliegenden Orten und die Entfernungen zwischen diesen.

Entfernungstabelle

| von | nach | km |
|---|---|---|
| Würzburg | Nürnberg | 115 |
| Nürnberg | Regensburg | 105 |
| Regensburg | AK Deggendorf | 70 |
| AK Deggendorf | Passau | 50 |
| Hof | Nürnberg | 135 |
| Nürnberg | Ingolstadt | 90 |
| Ingolstadt | AD Holledau | 20 |
| AD Holledau | München | 50 |
| München | AK Deggendorf | 140 |
| Hof | Regensburg | 170 |
| Regensburg | AD Holledau | 70 |

(a) Bestimmen Sie mit dem Algorithmus von Dijkstra den kürzesten Weg von Ingolstadt zu allen anderen Orten. Verwenden Sie zur Lösung eine Tabelle gemäß folgendem Muster und markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Ort. Setzen Sie für die noch zu bearbeitenden

[13] **66115:2017:03**.

Orte eine Prioritätswarteschlange ein, ðbei gleicher Entfernung wird der ältere Knoten gewählt.

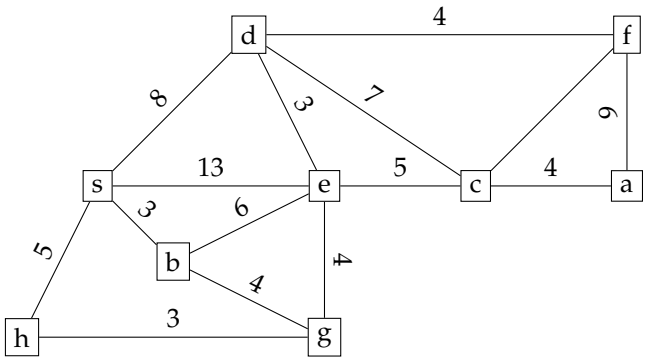
Ergebnis:

(b) Die bayerische Landesregierung hat beschlossen, die eben betrachteten Orte mit einem breitbandigen Glasfaser-Backbone entlang der Autobahnen zu verbinden. Dabei soll aus Kostengründen so wenig Glasfaser wie möglich verlegt werden. Identifizieren Sie mit dem Algorithmus von Kruskal diejenigen Strecken, entlang welcher Glasfaser verlegt werden muss. Geben Sie die Ortspaare (Autobahnsegmente) in der Reihenfolge an, in der Sie sie in Ihre Verkabelungsliste aufnehmen.

(c) Um Touristen den Besuch aller Orte so zu ermöglichen, dass sie dabei jeden Autobahnabschnitt genau einmal befahren müssen, bedarf es zumindest eines sogenannten offenen Eulerzugs. Zwischen welchen zwei Orten würden Sie eine Autobahn bauen, damit das bayerische Autobahnnetz mindestens einen Euler-Pfad enthält?

## Aufgabe 9:[14]

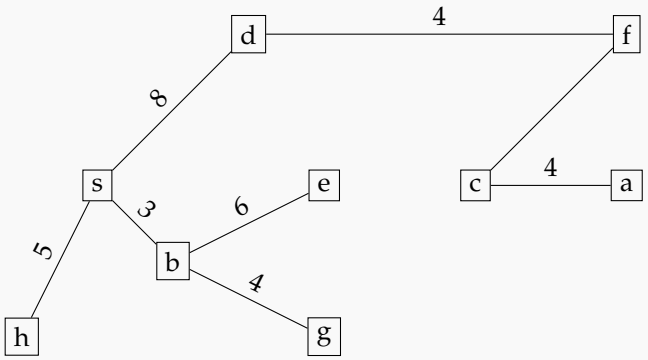
Gegeben sei folgender Graph $G$.




(a) Berechnen Sie mithilfe des Algorithmus von Dijkstra die kürzesten Wege vom Knoten $s$ zu allen anderen Knoten im Graphen $G$. Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte den jeweils als nächstes fertigzustellenden Knoten $v$ (wird sog. „schwarz“) als Tripel $(v, p, \delta)$ mit $v$ als Knotenname, $p$ als aktueller Vorgängerknoten und $\delta$ als aktuelle Distanz von $s$ zu $v$ über $p$ an. Führen Sie in der zweiten Spalten alle anderen bisher erreichten Knoten $v$ ebenfalls als Tripel $(v, p, \delta)$ auf, wobei diese sog. „grauen Randknoten“ in folgenden Durchgängen erneut betrachtet werden müssen. Zeichnen Sie anschließend den entstandenen Wegebaum, ðden Graphen $G$, in dem nur noch diejenigen Kanten vorkommen, die Teil der kürzesten Wege von $s$ zu allen anderen Knoten sind.

[14] **66115:2018:03**.

| Nr | „schwarze“ Knoten | „graue“ Randknoten |
|---|---|---|
| 1 | **(s, -, 0)** | [(b, s, 3)] (d, s, 8) (e, s, 13) (h, s, 5) |
| 2 | **(b, s, 3)** | (d, s, 8) (e, **b**, **9**) (g, b, 7) [(h, s, 5)] |
| 3 | **(h, s, 5)** | (d, s, 8) (e, b, 9) [(g, b, 7)] |
| 4 | **(g, b, 7)** | [(d, s, 8)] (e, b, 9) |
| 5 | **(d, s, 8)** | (c, d, 15) [(e, b, 9)] (f, d, 12) |
| 6 | **(e, b, 9)** | (c, **e**, **14**) [(f, d, 12)] |
| 7 | **(f, d, 12)** | (a, f, 21) [(c, **f**, **13**)] |
| 8 | **(c, f, 13)** | [(a, **c**, **17**)] |
| 9 | **(a, c, 17)** | |




**Alternativer Lösungsweg**

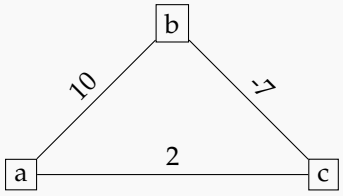
| **Nr.** | **besucht** | **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **s** |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 0 |
| 1 | s | ∞ | 3 | ∞ | 8 | 13 | ∞ | ∞ | 5 | **0** |
| 2 | b | ∞ | **3** | ∞ | 8 | 9 | ∞ | 7 | 5 | \| |
| 3 | h | ∞ | \| | ∞ | 8 | 9 | ∞ | 7 | **5** | \| |
| 4 | g | ∞ | \| | ∞ | 8 | 9 | ∞ | **7** | \| | \| |
| 5 | d | ∞ | \| | 15 | **8** | 9 | 12 | \| | \| | \| |
| 6 | e | ∞ | \| | 14 | \| | **9** | 12 | \| | \| | \| |
| 7 | f | 21 | \| | 13 | \| | \| | **12** | \| | \| | \| |
| 8 | c | 17 | \| | **13** | \| | \| | \| | \| | \| | \| |
| 9 | a | **17** | \| | \| | \| | \| | \| | \| | \| | \| |

| nach | Entfernung | Reihenfolge | Pfad |
|---|---|---|---|
| s → a | 17 | 9 | s → d → f → c → a |
| s → b | 3 | 2 | s → b |
| s → c | 13 | 8 | s → d → f → c |
| s → d | 8 | 5 | s → d |
| s → e | 9 | 6 | s → b → e |
| s → f | 12 | 7 | s → d → f |
| s → g | 7 | 4 | s → b → g |
| s → h | 5 | 3 | s → h |
| s → s | 0 | 1 | |

(b) Der Dijkstra-Algorithmus liefert bekanntlich auf Graphen mit negativen Kantengewichten unter Umständen ein falsches Ergebnis.

(i) Geben Sie einen Graphen mit negativen Kantengewichten an, sodass der Dijkstra-Algorithmus ausgehend von einem von Ihnen ausgezeichneten Startknoten ein korrektes Ergebnis liefert.

Startknoten a

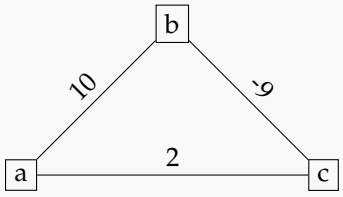



| Nr. | besucht | a | b | c |
|---|---|---|---|---|
| 0 | | 0 | ∞ | ∞ |
| 1 | a | **0** | 10 | 2 |
| 2 | c | \| | 10 | **2** |
| 3 | b | \| | **10** | \| |

Richtig: a - c: 2

(ii) Geben Sie einen Graphen mit negativen Kantengewichten an, sodass der Dijkstra-Algorithmus ausgehend von einem von Ihnen ausgezeichneten Startknoten ein falsches Ergebnis liefert.

Startknoten a

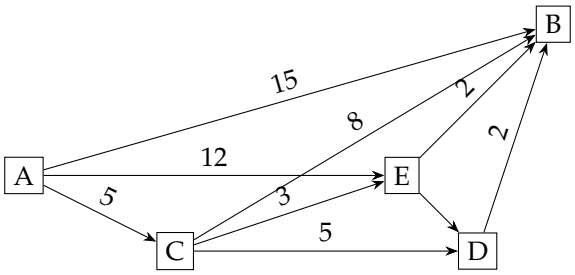


| Nr. | besucht | a | b | c |
|---|---|---|---|---|
| 0 | | 0 | ∞ | ∞ |
| 1 | a | **0** | 10 | 2 |
| 2 | c | \| | 10 | **2** |
| 3 | b | \| | **10** | \| |

falsch: a - c: müsste 1 (10 - 9) sein.

Ein Beweis oder eine Begründung ist jeweils nicht erforderlich.

## Aufgabe 3[15]




(a) Ermitteln Sie mit dem Algorithmus von Dijkstra den kürzesten Weg vom Knoten *A* zu allen erreichbaren Knoten in *G*. Verwenden Sie zur Lösung eine Tabelle der folgenden Form. Markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Knoten und führen Sie die Prioritätswarteschlange der noch zu betrachtenden Knoten (aufsteigend sortiert).

| Nr. | besucht | A | B | C | D | E |
|---|---|---|---|---|---|---|
| 0 | | 0 | ∞ | ∞ | ∞ | ∞ |

| Nr. | besucht | A | B | C | D | E |
|---|---|---|---|---|---|---|
| 0 | | 0 | ∞ | ∞ | ∞ | ∞ |
| 1 | A | **0** | 15 | 5 | ∞ | 12 |
| 2 | C | \| | 13 | **5** | 10 | 8 |
| 3 | E | \| | 10 | \| | 9 | **8** |
| 4 | D | \| | 10 | \| | **9** | \| |
| 5 | B | \| | **10** | \| | \| | \| |

(b) Geben Sie den kürzesten Pfad vom Knoten *A* zum Knoten *B* an.

[15] **examen:66115:2020:09**.

A → C → E → B: 10

| nach | Entfernung | Reihenfolge | Pfad |
|---|---|---|---|
| A → A | 0 | 0 | |
| A → B | 10 | 5 | A → C → E → B |
| A → C | 5 | 2 | A → C |
| A → D | 9 | 4 | A → C → E → D |
| A → E | 8 | 3 | A → C → E |