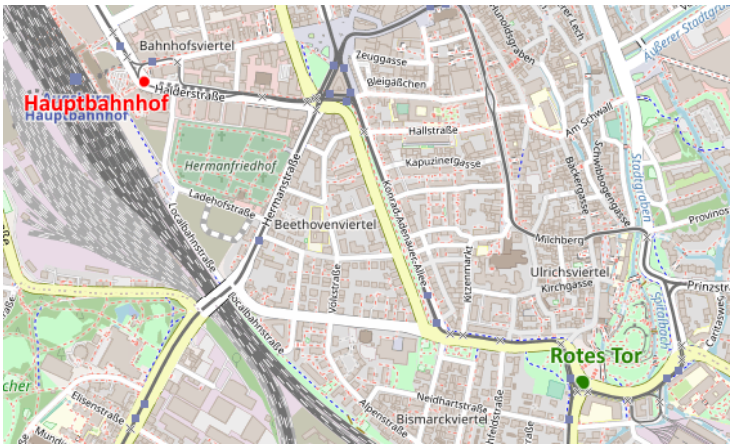
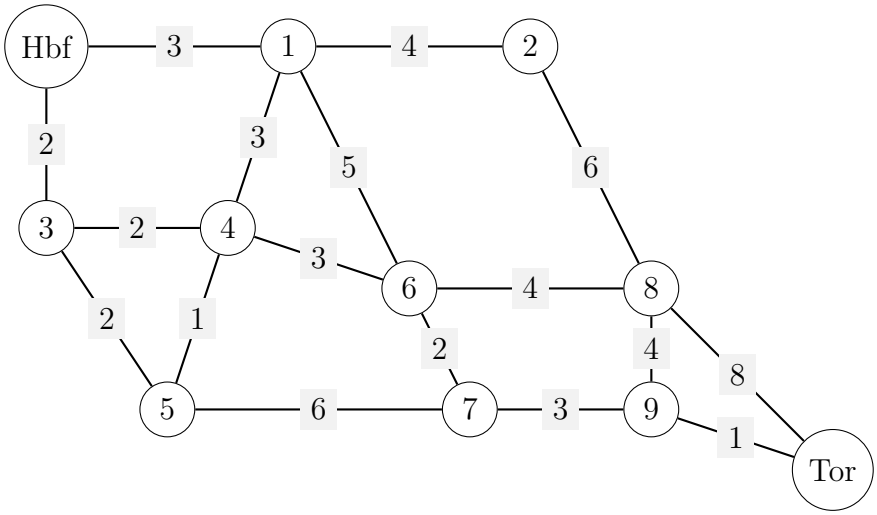
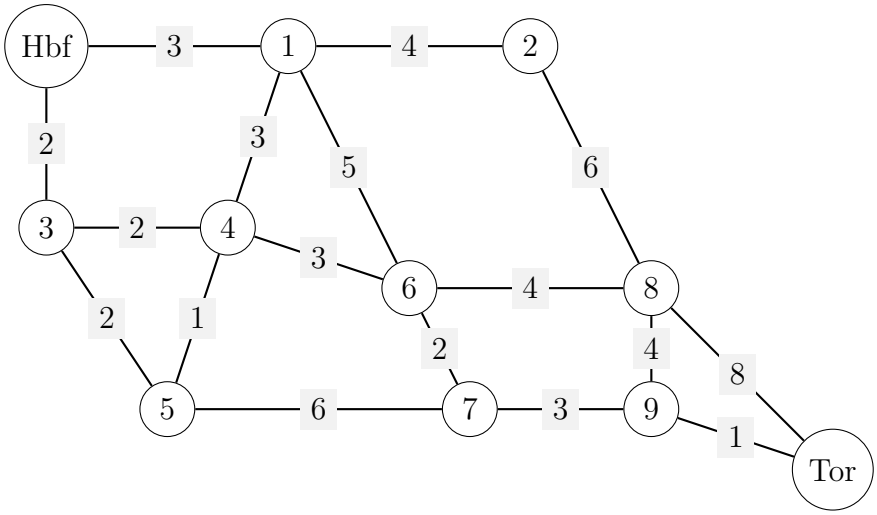


Ein Ausschnitt aus dem Augsburger Stadtplan © OpenStreetMap-Mitwirkende





Aufgabe 1.

Wie viele mögliche Wege gibt es in unserem Graphen?

a) Was schätzt du: Wie viele verschiedene Wege vom Hauptbahnhof zum Roten Tor gibt es in unserem Graphen?

b) Nehmen wir einmal kurz an, jeder der 12 Knoten wäre mit jedem anderen Knoten durch eine direkte Kante verbunden (das nennt man dann einen *vollständigen Graphen*). Wie viele Wege vom Hauptbahnhof zum Roten Tor gäbe es dann? Das ist eine obere Schranke an die Anzahl der Wege in unserem tatsächlichen Graphen. Warum?
Hinweis: Du darfst hier annehmen, dass ein Weg sinnvollerweise nie zweimal über die gleiche Kreuzung läuft. Bestimme jetzt wie viele Wege aus 1 Kante es gibt. Wie viele Wege aus 2 Kanten. Wie viele Wege aus 3 Kanten. Usw.

c) Nehmen wir nun stattdessen an die 12 Knoten wären in einem 3×4-Gitter angeordnet, wobei unser Startknoten ganz links oben und unser Zielknoten ganz rechts unten ist. Wie viele Wege gibt es jetzt, bei denen wir immer nur entweder nach rechts oder nach unten laufen? Das ist eine untere Schranke an die Gesamtzahl der Wege in unserem Gittergraphen. Warum?

d) Stell dir nun vor wir hätten einen Gittergraphen mit 10×10 Knoten. Was sind die untere und obere Schranke nun? Was ist mit einem $n \times n$ Gitter?

Satz 1. *Sei $s{-}v_1{-}v_2{-}v_3{-}\ldots{-}v_k$ ein kürzester Weg von s nach v_k. Wenn wir (in dieser Reihenfolge, aber möglicherweise noch mit anderen Schritten dazwischen) die Kanten $s{-}v_1, v_1{-}v_2, v_2{-}v_3, \ldots, v_{k-1}{-}v_k$ betrachten (und gegebenenfalls abkürzen), dann gilt danach $\tilde{d}(v_k) = d(v_k)$.*

Aufgabe 2.

Zeige, dass dieser Satz richtig ist.

Tipp: Zeige zunächst die folgende (einfachere) Aussage: Gilt $\tilde{d}(v) = d(v)$ und wir machen einen Abkürzungsschritt zur Kante $v{-}w$, dann gilt danach auch $\tilde{d}(w) = d(w)$. Überlege dir dann wie daraus der obige Satz folgt.

Input : Ein Graph mit Längenangaben auf den Kanten und ein Startknoten s
Output : Für jeden Knoten v dessen Distant $d(v)$ von s aus
Setze $\tilde{d}(s) = 0$ und $\tilde{d}(v) = \infty$ für alle knoten $v \neq s$
for $i = 1, 2, \ldots, n-1$ **do**
  foreach *Kante* $v - w$ **do**
    if $\tilde{d}(v) +$ *Länge von* $v{-}w < \tilde{d}(w)$ **then**
      Setze $\tilde{d}(w) = \tilde{d}(v) +$ Länge von $v{-}w$
    end
    if $\tilde{d}(w) +$ *Länge von* $w{-}v < \tilde{d}(v)$ **then**
      Setze $\tilde{d}(v) = \tilde{d}(w) +$ Länge von $w{-}v$
    end
  end
end

Aufgabe 3.

Führe den Algorithmus von Bellman-Ford auf dem Beispiel-Graphen vom Anfang aus.

Aufgabe 4.

Wie oft musst du für unseren Beispiel-Graphen einen Abkürzungsschritt durchführen?

Wie oft müsstest du das bei einem Graphen aus n Knoten (höchstens) machen? Vergleiche dein Ergebnis mit dem aus Aufgabe 1.

Satz 2. *Sei v ein Knoten, für den $\tilde{d}(v) = d(v)$ gilt (wir kennen also bereits seine wahre Distanz vom Startknoten). Führen wir noch einen Abkürzungsschritt mit der Kante $v-w$ durch, so werden wir danach nie wieder eine Abkürzung über diese Kante finden.*

Aufgabe 5.

Zeige, dass der obige Satz wirklich wahr ist.

Aufgabe 6.

Überzeuge dich, dass der Algorithmus von Dijkstra wirklich korrekt ist.

Hinweis: Einen vollständigen Beweis hierfür zu finden, ist nicht ganz einfach. Aber vielleicht bist du ja schon davon überzeugt, dass der Algorithmus korrekt ist, wenn du die folgenden Fragen beantworten kannst:

- Warum kann sich der Wert $\tilde{d}(v)$ eines Knotens v nicht mehr ändern, nachdem wir ihn als fertig bezeichnet haben.
- Warum gilt für alle fertigen Knoten v, dass wir bereits ihre wahre Distanz von s kennen also ($\tilde{d}(v) = d(v)$)?
- Ist ein Knoten v wirklich fertig, wenn wir ihn so nennen? Anders gefragt: Warum hat ein solcher Knoten die Eigenschaften, die in Satz 2 gefordert werden.

Input : Ein Graph mit Längenangaben auf den Kanten und ein Startknoten s
Output : Für jeden Knoten v dessen Distant $d(v)$ von s aus
Setze $\tilde{d}(s) = 0$ und $\tilde{d}(v) = \infty$ für alle knoten $v \neq s$
Nenne alle Knoten „unfertig“
while *es gibt noch unfertige Knoten* **do**
  Wähle einen unfertigen Knoten v mit geringstem $\tilde{d}(v)$ unter allen unfertigen Knoten.
  foreach *Kante* $v - w$*, die von* v *ausgeht* **do**
    if $\tilde{d}(v) +$ *Länge von* $v{-}w < \tilde{d}(w)$ **then**
      Setze $\tilde{d}(w) = \tilde{d}(v) +$ Länge von $v{-}w$
    end
    if $\tilde{d}(w) +$ *Länge von* $w{-}v < \tilde{d}(v)$ **then**
      Setze $\tilde{d}(v) = \tilde{d}(w) +$ Länge von $w{-}v$
    end
  end
  Nenne den Knoten v „fertig“
end

Aufgabe 7.

Führe den Algorithmus von Dijkstra auf dem Beispiel-Graphen vom Anfang aus.

Aufgabe 8.

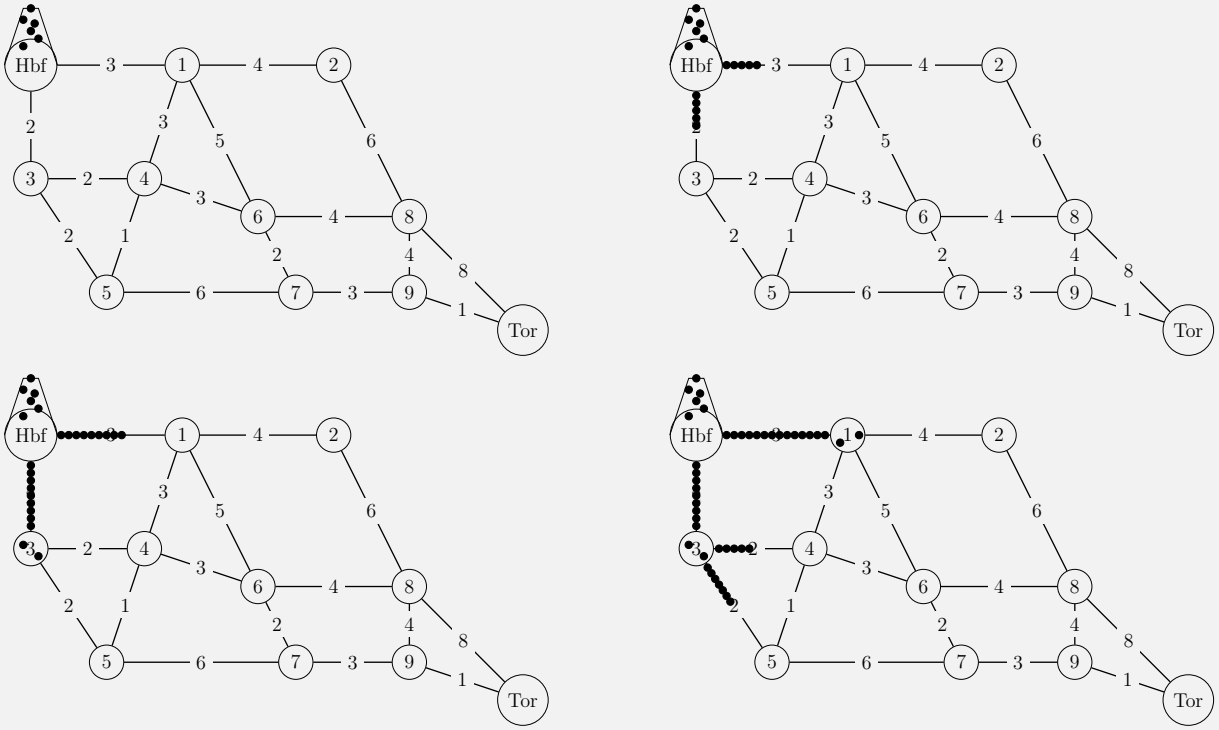
Wie oft musst du für unseren Beispiel-Graphen einen Abkürzungsschritt durchführen?

Wie oft müsstest du das bei einem Graphen aus n Knoten (höchstens) machen? Vergleiche dein Ergebnis mit dem aus Aufgabe 1 und Aufgabe 4.

Aufgabe 9.

Ich habe mal gelesen, dass der Dijkstra-Algorithmus auch als „Ameisen-Erkundungs-Algorithmus“ bezeichnet wird. Dieser Name kommt daher, dass man sich den Ablauf des Algorithmus auch so vorstellen kann:
Am Startknoten befindet sich ein großer Ameisenhügel, von dem aus sich die Ameisen zum Zeitpunkt 0 auf Erkundungstour machen und dazu gleichzeitig in alle Richtungen ausschwärmen. Die Ameisen bewegen sich dabei mit einer gleichmäßigen Geschwindigkeit (für eine Kante der Länge 6 brauchen sie also beispielsweise doppelt so lang wie für eine Kante der Länge 3 um vom einen Ende zum anderen zu kommen). Wann immer die Ameisen zu einem Knoten kommen, an dem noch keine anderen Ameisen sind, teilen sie sich auf und strömen in alle ausgehenden Kanten um weiter zu erkunden. Kommen sie an einen Knoten, an dem bereits andere Ameisen sind, hören sie auf in diese Richtung zu suchen (weil das ja schon die Ameisen übernommen haben, die diesen Knoten früher

erreicht haben). Im Folgenden habe ich mal ein paar Beispiele wie das ganze nach Zeit 0, 1, 2 und 3 in unserem Beispiel-Graphen aussieht (die Punkte sollen die Ameisen sein):



Wenn wir jetzt die Ameisen von oben beobachten und uns die Zeiten aufschreiben, an denen die Ameisen zum ersten mal einen Knoten erreichen, dann erhalten wir genau unsere Distanzen vom Startknoten. Kannst du erkennen, warum das so ist? Lasse die Ameisen weiterlaufen und versuche so die Distanzen zu bestimmen.

Siehst du den Zusammenhang zum Algorithmus von Dijkstra?

Aufgabe 10.

Eigentlich wollten wir doch kürzeste Wege bestimmen – unsere beiden Algorithmen berechnen aber nur kürzeste *Distanzen* (die Werte $d(v)$)! Hilft uns das überhaupt weiter?

Tatsächlich ist es nicht besonders schwierig kürzeste Wege zu bestimmen, wenn man erstmal die kürzesten Distanzen kennt. Kannst du herausfinden wie?

Hinweis: Es gibt (mindestens) zwei Wege das zu tun:

- Entweder du versuchst die Algorithmen so anzupassen, dass wir uns dabei schon irgendwie *merken* wie wir jeweils zu den einzelnen Knoten gekommen sind.
- Oder du nutzt die Distanzen, die wir am Ende bestimmt haben um daraus *rückwärts* kürzeste Wege zu den einzelnen Knoten zu konstruieren.

Aufgabe 11.

Denke dir selbst einen Graphen aus und bestimme darin kürzeste Wege.

Aufgabe 12.

Finde Graphen, auf denen unsere Algorithmen möglichst schlecht sind. Also zum Beispiel einen Graphen, auf dem der Wert $\tilde{d}(v)$ ein und desselben Knotens v möglichst oft geändert wird. Oder ein Graph mit einer Kante $v{-}w$, die möglichst oft zu einer neuen Abkürzung führt.

Was ist die höchste Zahl die du erreichen kannst?

Aufgabe 13.

Bisher waren die Längen der Kanten immer positive Zahlen – was aber passiert, wenn wir auch negative Zahlen zulassen? [1] Funktionieren unsere Algorithmen dann immer noch, oder kannst du dir einen Graphen überlegen, in dem die Algorithmen dann etwas falsches machen?

Hinweis: Das ist eine sehr offene Frage, die nicht eine eindeutige richtige Antwort hat, aber zu vielen interessanten neuen Fragen führen kann :-)