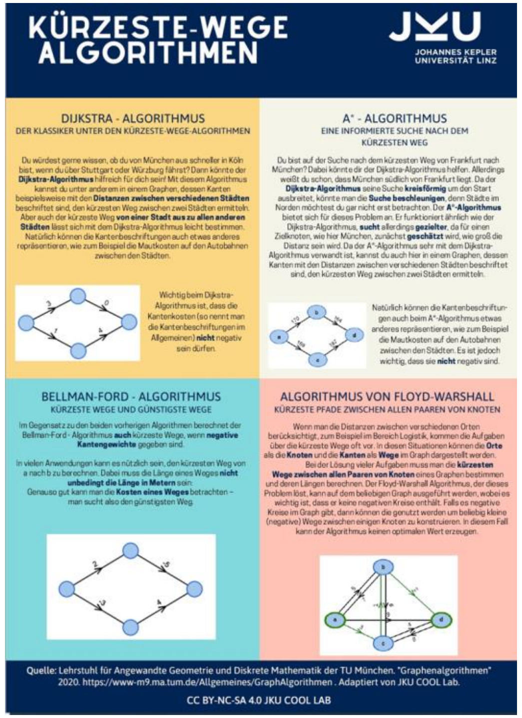
JOHANNES KEPLER
UNIVERSITÄT LINZ


KÜRZESTE-WEGE ALGORITHMEN

JOHANNES KEPLER UNIVERSITÄT LINZ

DIJKSTRA - ALGORITHMUS
DER KLASSIKER UNTER DEN KÜRZESTE-WEGE-ALGORITHMEN

Du würdest gerne wissen, ob du von München aus schneller in Köln bist, wenn du über Stuttgart oder Würzburg fährst? Dann könnte der **Dijkstra-Algorithmus** hilfreich für dich sein! Mit diesem Algorithmus kannst du unter anderem in einem Graphen, dessen Kanten beispielsweise mit den **Distanzen zwischen verschiedenen Städten** beschriftet sind, den kürzesten Weg zwischen zwei Städten ermitteln. Aber auch der kürzeste Weg **von einer Stadt aus zu allen anderen Städten** lässt sich mit dem Dijkstra-Algorithmus leicht bestimmen. Natürlich können die Kantenbeschriftungen auch etwas anderes repräsentieren, wie zum Beispiel die Mautkosten auf den Autobahnen zwischen den Städten.

Wichtig beim Dijkstra-Algorithmus ist, dass die Kantenkosten (so nennt man die Kantenbeschriftungen im Allgemeinen) **nicht** negativ sein dürfen.

A* - ALGORITHMUS
EINE INFORMIERTE SUCHE NACH DEM KÜRZESTEN WEG

Du bist auf der Suche nach dem kürzesten Weg von Frankfurt nach München? Dabei könnte dir der Dijkstra-Algorithmus helfen. Allerdings weißt du schon, dass München südlich von Frankfurt liegt. Da der **Dijkstra-Algorithmus** seine Suche **kreisförmig** um den Start ausbreitet, könnte man die **Suche beschleunigen**, denn Städte im Norden möchtest du gar nicht erst betrachten. Der **A*-Algorithmus** bietet sich für dieses Problem an. Er funktioniert ähnlich wie der Dijkstra-Algorithmus, **sucht** allerdings **gezielter**, da für einen Zielknoten, wie hier München, zunächst **geschätzt** wird, wie groß die Distanz sein wird. Da der A*-Algorithmus sehr mit dem Dijkstra-Algorithmus verwandt ist, kannst du auch hier in einem Graphen, dessen Kanten mit den Distanzen zwischen verschiedenen Städten beschriftet sind, den kürzesten Weg zwischen zwei Städten ermitteln.

Natürlich können die Kantenbeschriftungen auch beim A*-Algorithmus etwas anderes repräsentieren, wie zum Beispiel die Mautkosten auf den Autobahnen zwischen den Städten. Es ist jedoch wichtig, dass sie **nicht** negativ sind.

BELLMAN-FORD - ALGORITHMUS
KÜRZESTE WEGE UND GÜNSTIGSTE WEGE

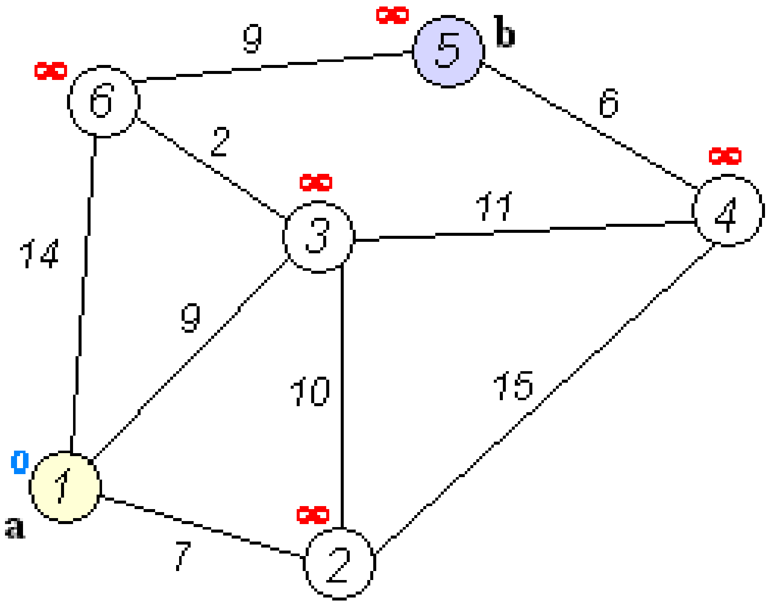
Im Gegensatz zu den beiden vorherigen Algorithmen berechnet der Bellman-Ford - Algorithmus **auch** kürzeste Wege, wenn **negative Kantengewichte** gegeben sind.

In vielen Anwendungen kann es nützlich sein, den kürzesten Weg von a nach b zu berechnen. Dabei muss die Länge eines Weges **nicht unbedingt die Länge in Metern** sein.
Genauso gut kann man die **Kosten eines Weges** betrachten – man sucht also den günstigsten Weg.

ALGORITHMUS VON FLOYD-WARSHALL
KÜRZESTE PFADE ZWISCHEN ALLEN PAAREN VON KNOTEN

Wenn man die Distanzen zwischen verschiedenen Orten berücksichtigt, zum Beispiel im Bereich Logistik, kommen die Aufgaben über die kürzeste Wege oft vor. In diesen Situationen können die **Orte** als die **Knoten** und die **Kanten** als **Wege** im Graph dargestellt werden. Bei der Lösung vieler Aufgaben muss man die **kürzesten Wege zwischen allen Paaren von Knoten** eines Graphen bestimmen und deren Längen berechnen. Der Floyd-Warshall Algorithmus, der dieses Problem löst, kann auf dem beliebigen Graph ausgeführt werden, wobei es wichtig ist, dass er keine negativen Kreise enthält. Falls es negative Kreise im Graph gibt, dann können die genutzt werden um beliebig kleine (negative) Wege zwischen einigen Knoten zu konstruieren. In diesem Fall kann der Algorithmus keinen optimalen Wert erzeugen.

Quelle: Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik der TU München. "Graphenalgorithmen" 2020. https://www-m9.ma.tum.de/Allgemeines/GraphAlgorithmen . Adaptiert von JKU COOL Lab.

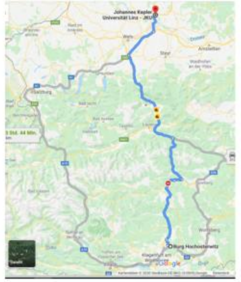


Kürzeste Wege Algorithmen im Informatikunterricht an Schulen

Univ.-Prof. MMag. Dr. Barbara Sabitzer
MINT-Didaktik & COOL Lab
Johannes Kepler Universität Linz
https://www.jku.at/schule/cool-lab/
barbara.sabitzer@jku.at

Kontext & Einbettung: Kürzeste Wege für (zukünftige) LehrerInnen

- LVA
  - Schulinformatik I & II
  - **Special Topics: Digitale Bildung & Computational Thinking**
- COOL Lab
  - LehrerInnenfortbildung
  - Thementage
  - COOL Lab Workshops



JKU Barbara Sabitzer Computational Thinking & Digitale Grundbildung integrativ https://www.google.at/maps 3

Stationen – COOL-IT

1. Kürzeste Wege sind überall
2. Hands-on & Bewegung
3. Kürzeste Wege entdecken
4. Roboter & Programmieren
5. Infos & mehr
6. Kürzeste Wege für Profis

1. Discovery
- Einstieg über verschiedene Fächer
- Animitation / Simulation / Video

2. Cooperation
- Teams & Wettbewerb
- Pair Programming
- Peer TutorInnen

3. Individuality
- Infostation & Zusatzmaterialien
- Eigene Ideen für kürzeste Wege

4. Activity
- Hands-on & Bewegungsspiele
- Programmieren & Drohnen fliegen

(Sabitzer 2014)

JKU Barbara Sabitzer Computational Thinking & Digitale Grundbildung integrativ 8

Tipps aus der Praxis

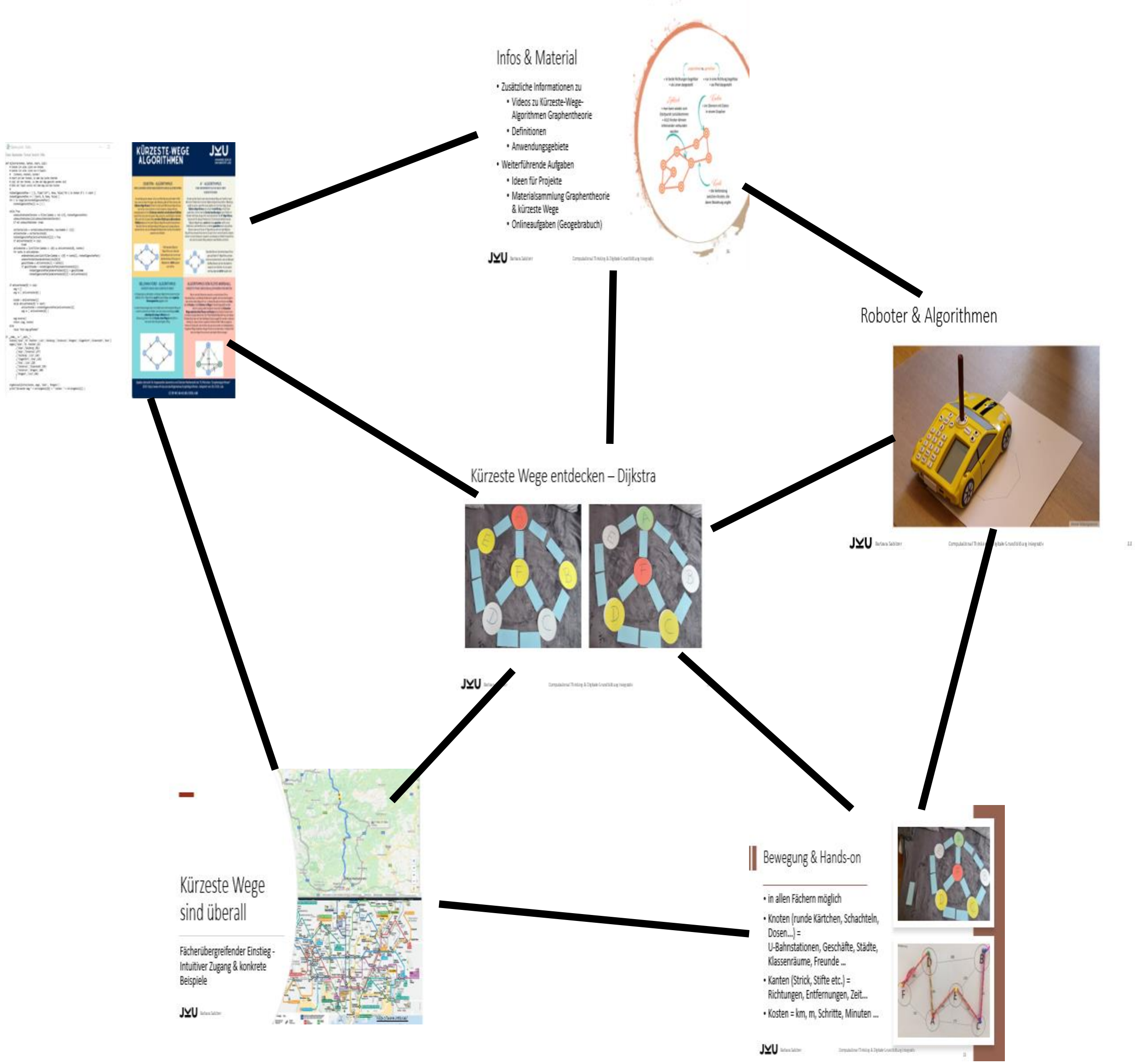
Schwierige & Komplexe Inhalte

Verschiedene Zugänge
- Fächer
- Perspektiven
- Aspekte
- Bezug zum Alltag bzw. bekannten Themen

Vielfältige Materialien
- Aussagekräftige Beispiele
- Musteraufgaben – Worked Examples
- Für alle Sinne
- Zum BeGREIFen und ErLeben

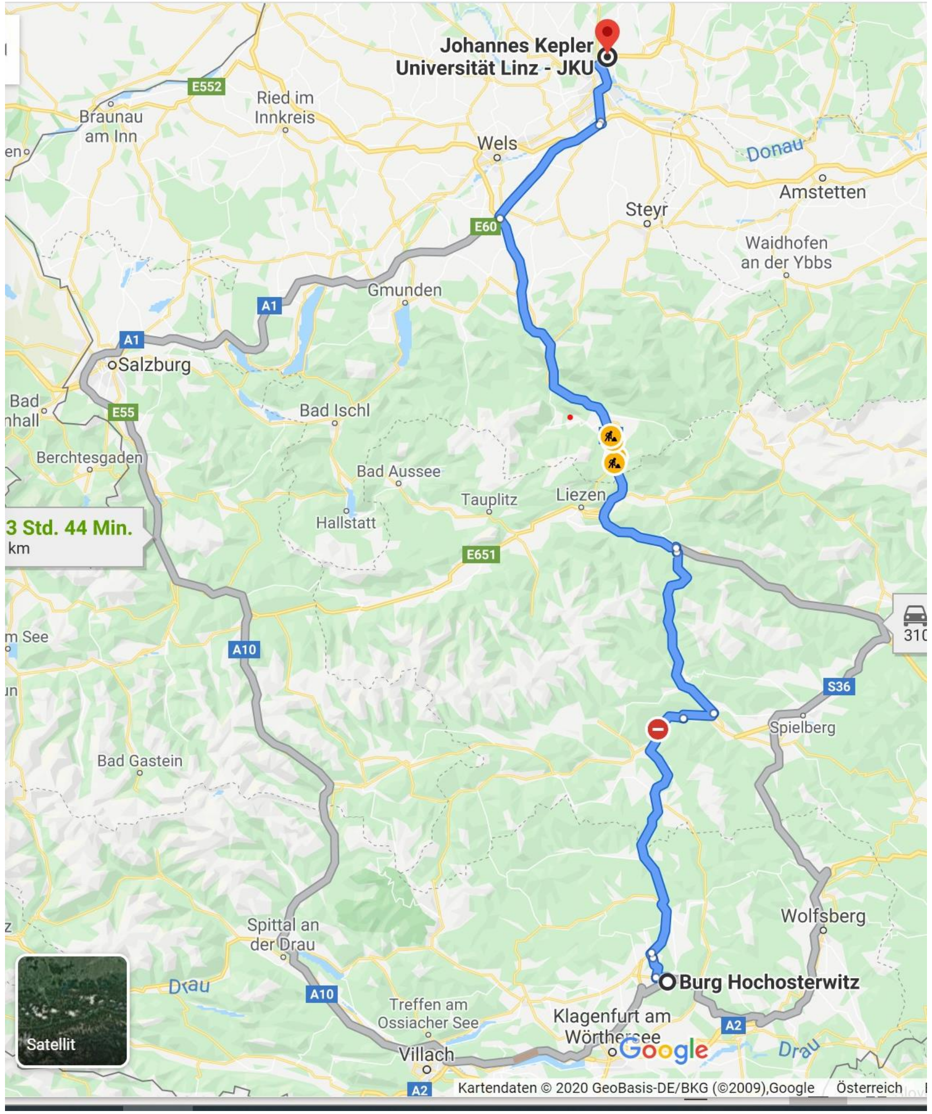
Spiralförmiger Aufbau
- Minimalwissen
- Aufbau
- Ausnahmen

JKU Barbara Sabitzer Computational Thinking & Digitale Grundbildung integrativ 42



Kontext & Einbettung: Kürzeste Wege für (zukünftige) LehrerInnen

- LVA
  - Schulinformatik I & II
  - **Special Topics: Digitale Bildung & Computational Thinking**
- COOL Lab
  - LehrerInnenfortbildung
  - Thementage
  - COOL Lab Workshops



Einbettung:
Kürzeste Wege in der Schule

Lehrpläne

- Informatik: Algorithmen & Datenstrukturen
- Digitale Grundbildung: Algorithmen
- Mathematik: Graphentheorie

Integrativ

- Sprachen, Geographie, Sport ...
- Fächerübergreifende Projekte
- COOL Lab Workshop

Materialien

- Unterrichtspakete
  - Kürzeste Wege
  - Aufgabensammlung zur Graphentheorie
- Geogebrabuch Graphen
  - Theorie & Aufgaben
  - Linksammlung & Videos
- Hands-on Materialien
  - Roboter & Matten
  - Kärtchen, Schnüre & Schere
  - Stifte & Farben etc.

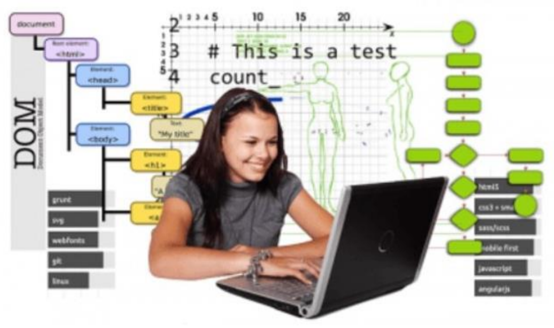
Zusatzmaterialien – Didaktik & Informatik

GeoGebra

- Special Topics: Digitale Bildung und Co...
- Infos & Arbeitsaufträge
- Methoden
- Digitale Bildung
- Computational Thinking
- Digitale Tools
- Erklärvideos erstellen
- Algorithmen
- Modellierung
- Codierung
- Programmierung
- Verschiedene Beiträge & Materialien

Special Topics: Digitale Bildung und Computational Thinking

Autor: Barbara Sabitzer



Inhaltsverzeichnis

1. COOL Informatics - COOL-IT
17. März 2020 - 09:24
Barbara Sabitzer
Dieses Material ist nicht öffentlich sichtbar

2. Problem-based Learning - Üb...
21. März 2020 - 07:50
Barbara Sabitzer
Dieses Material ist nicht öffentlich sichtbar

3. PBL - 7 Schritte
24. März 2020 - 07:00
Barbara Sabitzer
Dieses Material ist nicht öffentlich sichtbar

4. Flipped Classroom für Lehrp...
17. März 2020 - 09:07
Barbara Sabitzer
Dieses Material ist nicht öffentlich sichtbar

5. Projektbasiertes Lernen & P...
24. März 2020 - 09:41
Barbara Sabitzer
Dieses Material ist nicht öffentlich sichtbar

6. Personalisiertes Lernen
22. März 2020 - 07:05
Barbara Sabitzer
Dieses Material ist nicht öffentlich sichtbar

7. Weitere Methoden - Links
24. März 2020 - 10:12
Barbara Sabitzer
Dieses Material ist nicht öffentlich sichtbar

Stationen – COOL-IT

1. Kürzeste Wege sind überall
2. Hands-on & Bewegung
3. Kürzeste Wege entdecken
4. Roboter & Programmieren
5. Infos & mehr
6. Kürzeste Wege für Profis

- Einstieg über verschiedene Fächer
- Animitation / Simulation / Video

- Teams & Wettbewerb
- Pair Programming
- Peer TutorInnen

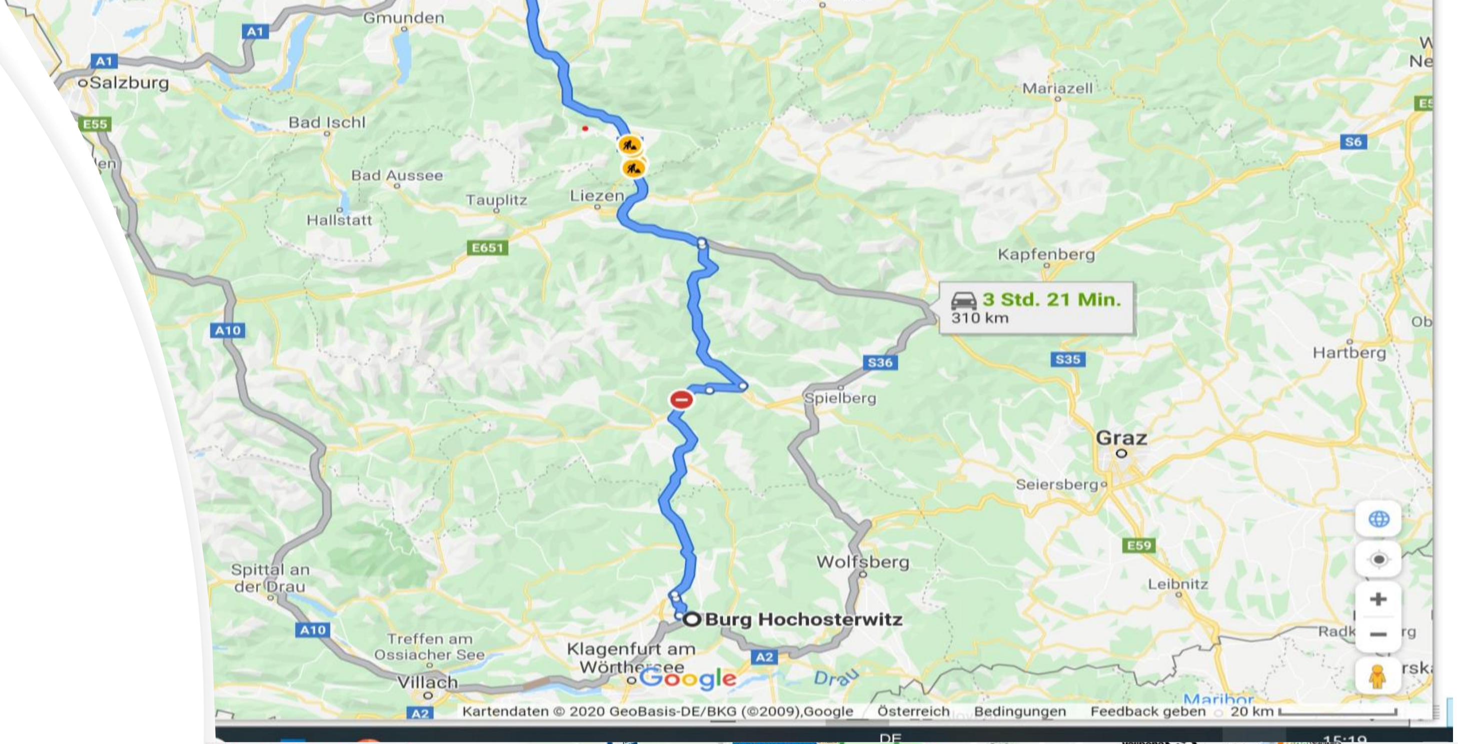
- Infostation & Zusatzmaterialien
- Eigene Ideen für kürzeste Wege

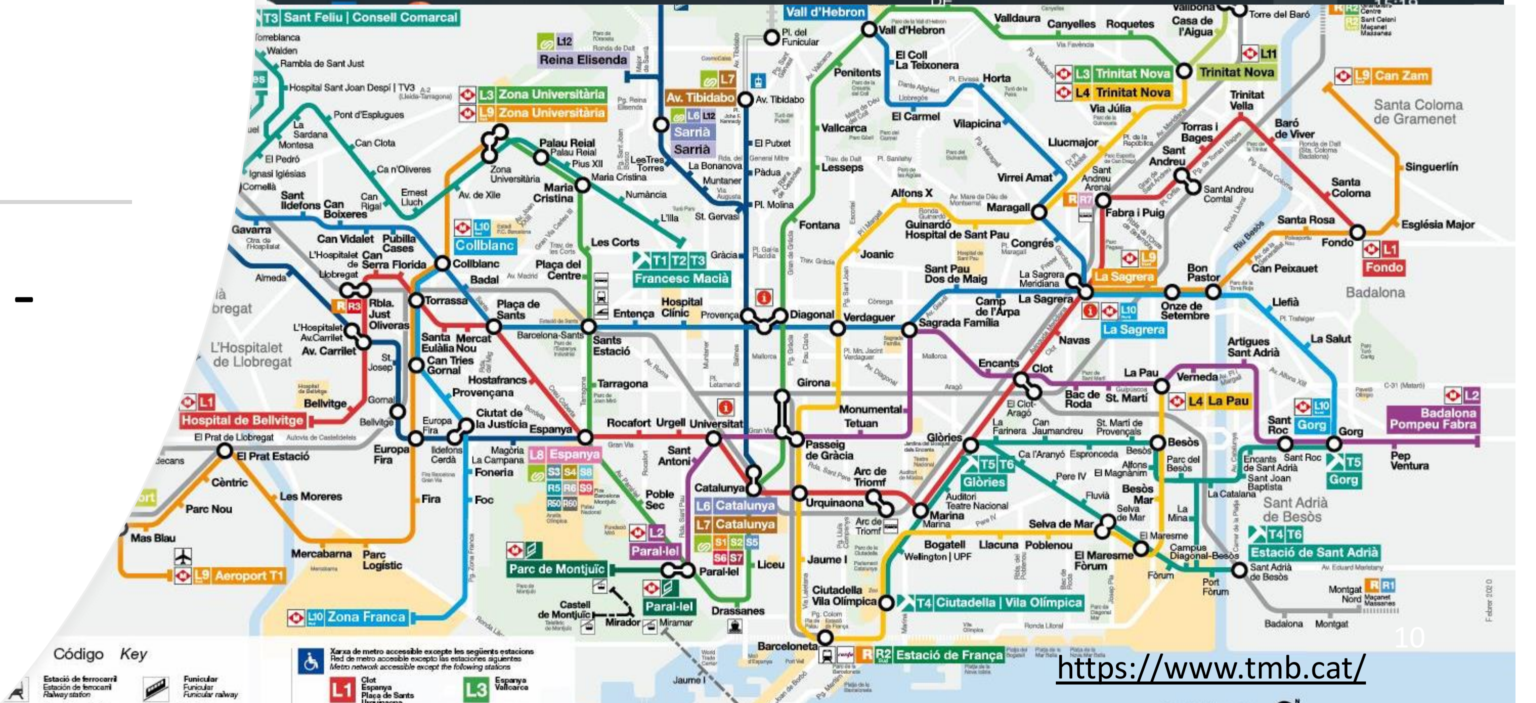
- Hands-on & Bewegungsspiele
- Programmieren & Drohnen fliegen

(Sabitzer 2014)

Kürzeste Wege sind überall

Fächerübergreifender Einstieg - Intuitiver Zugang & konkrete Beispiele





https://www.tmb.cat/

Barbara Sabitzer

Sprachen & Pseudocode

https://www.tmb.cat/

El viaje más rápido a …

Parto desde …

-> Startknoten

¿Cómo se llega a …?

-> Endknoten

¿Qué linea se toma?

-> Kanten

¿Cuántas paradas hay hasta …?

-> Kosten

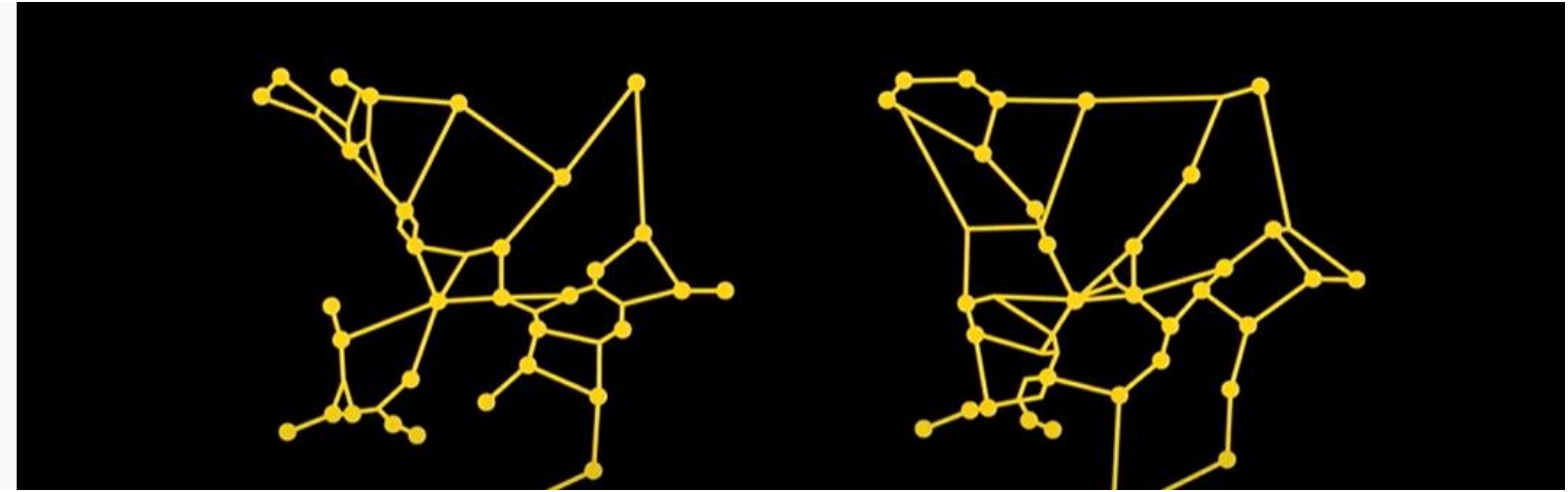
Tomo linea 1, despues linea 3…

-> Algorithmus

Biologie

Was Schleimpilze (Einzeller) und U-Bahnfahrer gemeinsam haben:

Sie suchen den kürzesten Weg



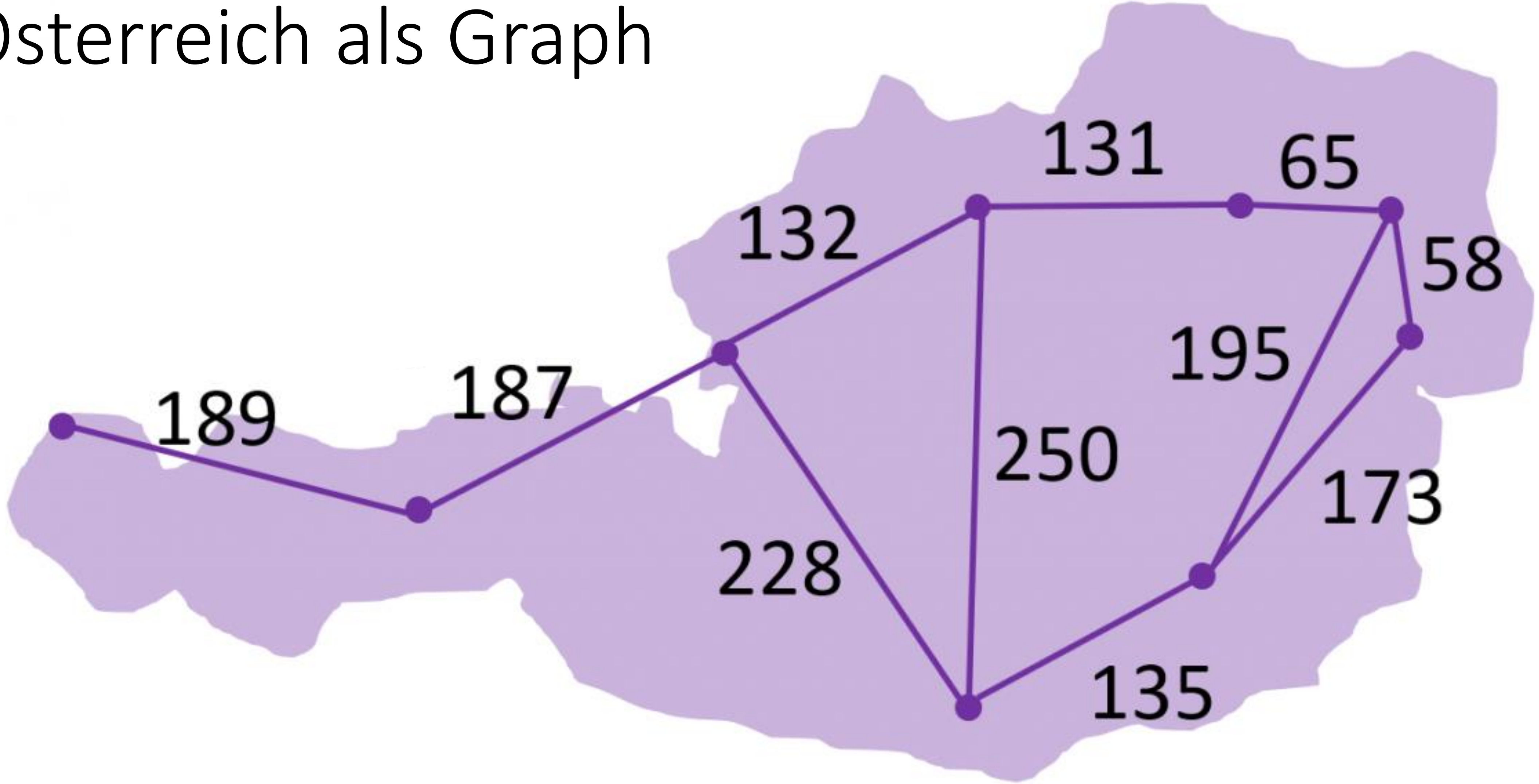
https://www.youtube.com/watch?v=f2cSz14Y_wY&t=20s

Geographie & Sport: Kennst du Österreich?

- Kürzester Weg zwischen Hauptstädten
- Berechnen & Messen
- Laufen
  - Riesengraph im Turnsaal
  - Matten = Knoten = Städte
  - Seile = Kanten = Straßen
  - Kreide = Grenzen

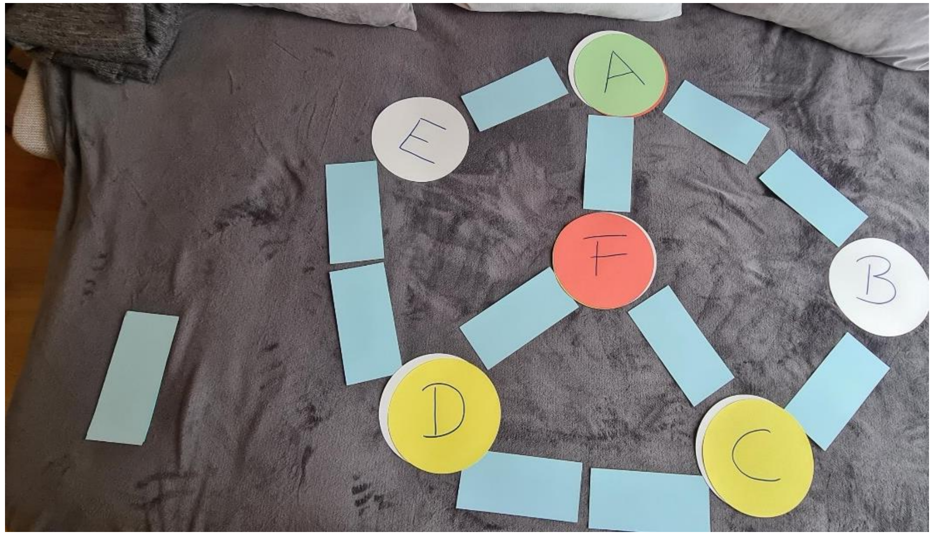
https://commons.wikimedia.org/wiki/File:%C3%96sterreich_-_Bundesl%C3%A4nder_-_Hauptst%C3%A4dte.jpg

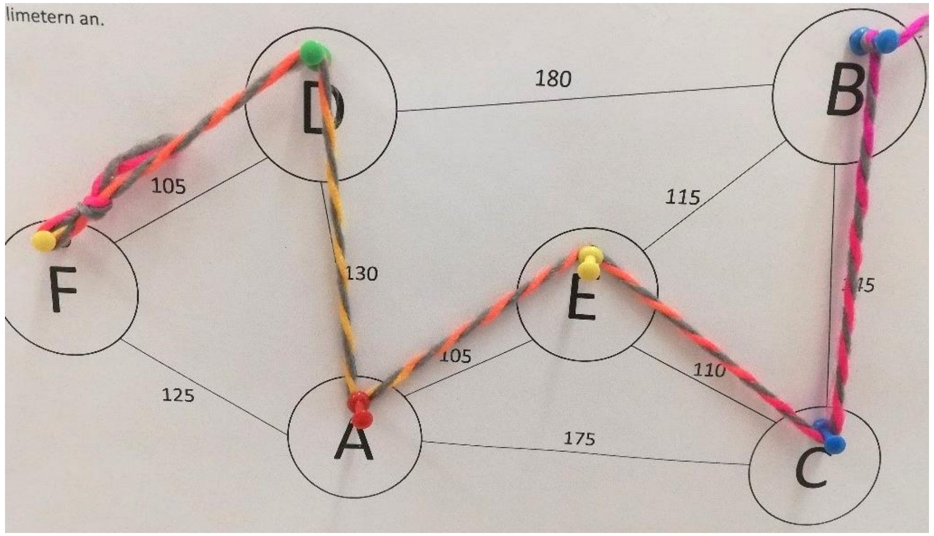
Österreich als Graph



Bewegung & Hands-on

- in allen Fächern möglich
- Knoten (runde Kärtchen, Schachteln, Dosen...) = U-Bahnstationen, Geschäfte, Städte, Klassenräume, Freunde ...
- Kanten (Strick, Stifte etc.) = Richtungen, Entfernungen, Zeit...
- Kosten = km, m, Schritte, Minuten ...



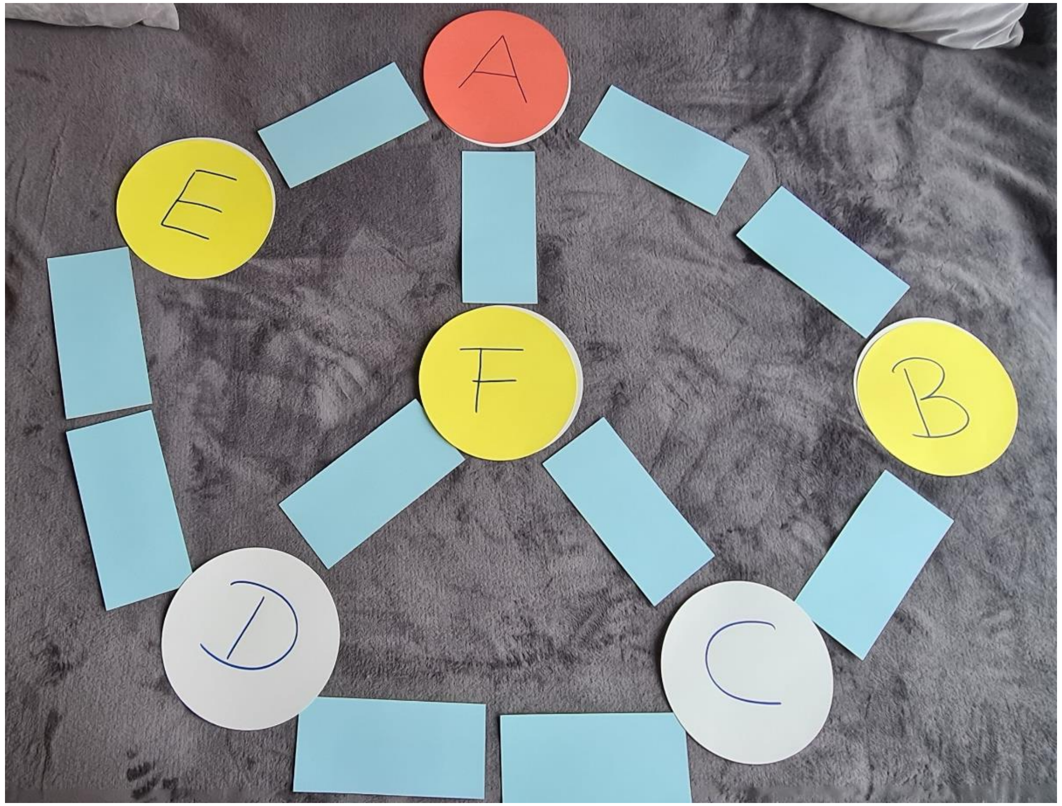


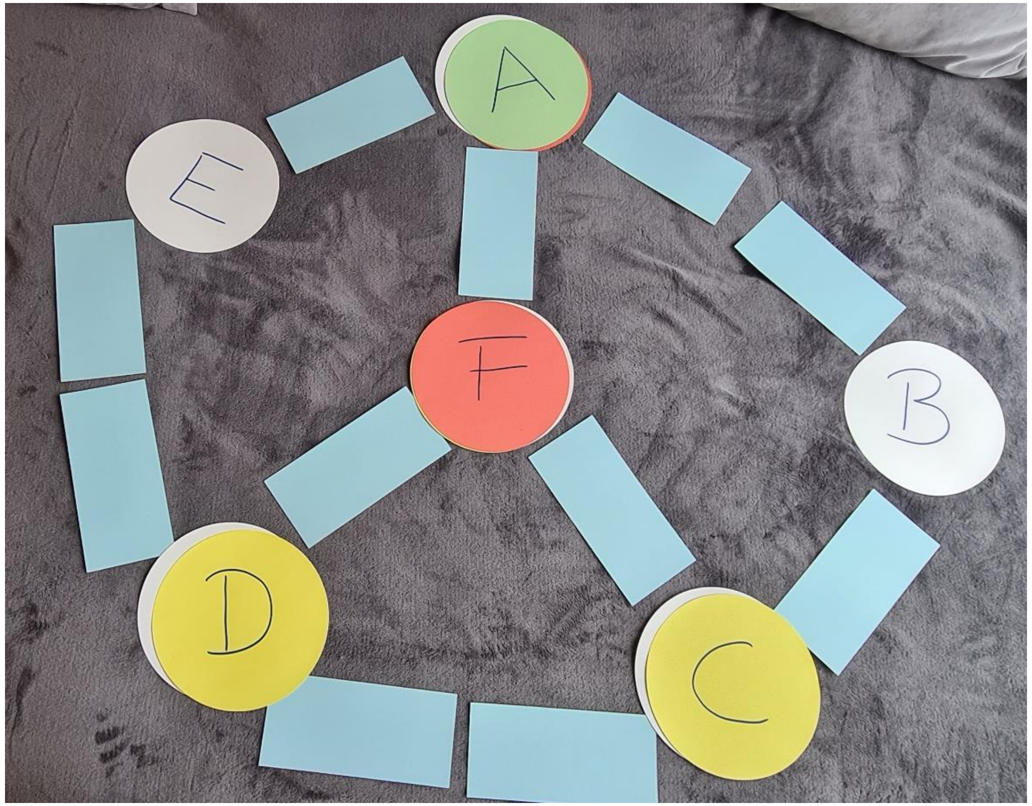
Brettspiel –
Kennst du die Welt?

- 4 Gruppen
  - Europa
  - Asien
  - Nordamerika
  - Südamerika
- Kürzeste Wege einzeichnen zwischen
  - Ländern
  - Städten
  - Meeren

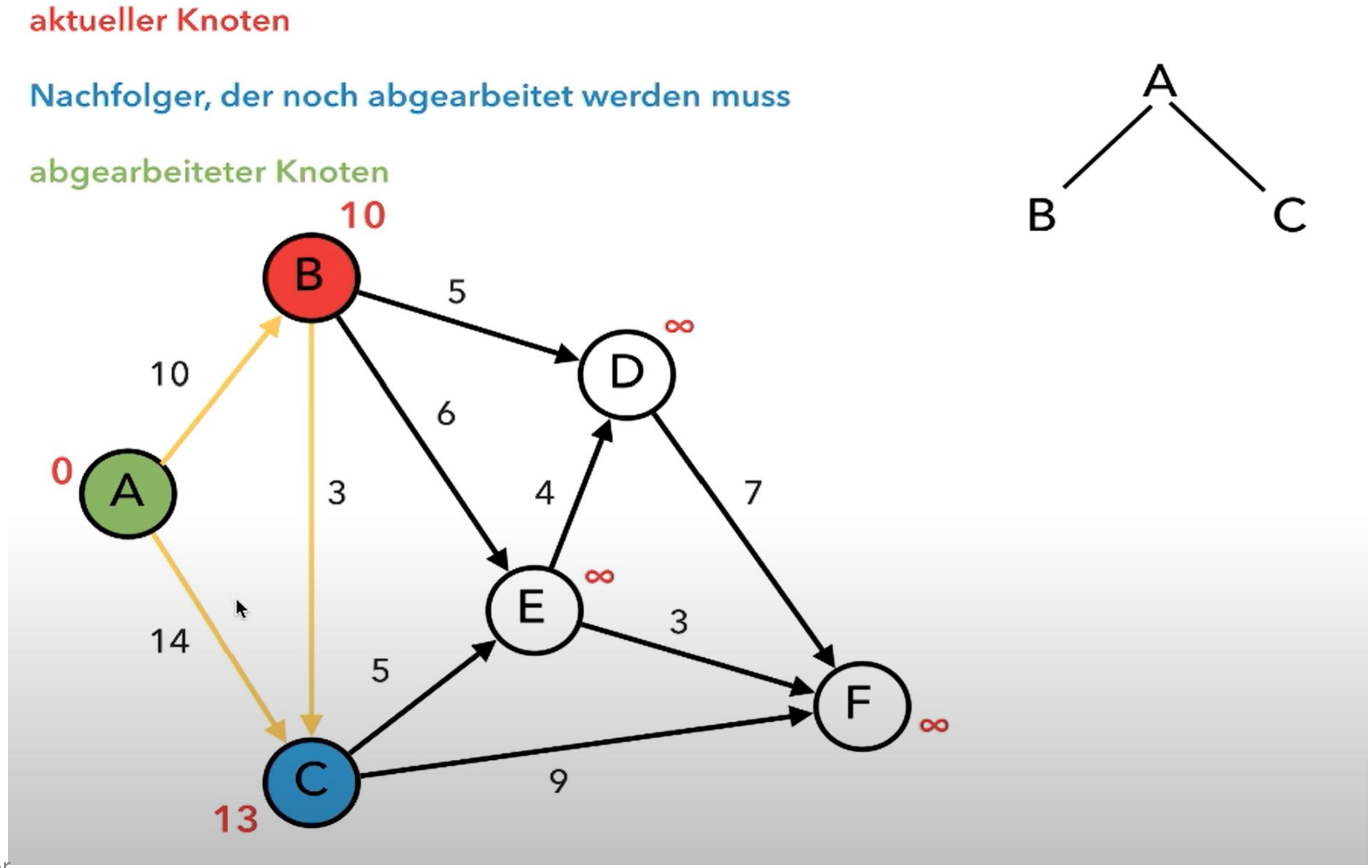
https://static.springer.com/sgw/documents/1360762/image/jpeg/w_37_3577.jpg

Kürzeste Wege entdecken – Dijkstra





Kürzeste Wege entdecken
Dijkstra Algorithmus mit Farben (Video)

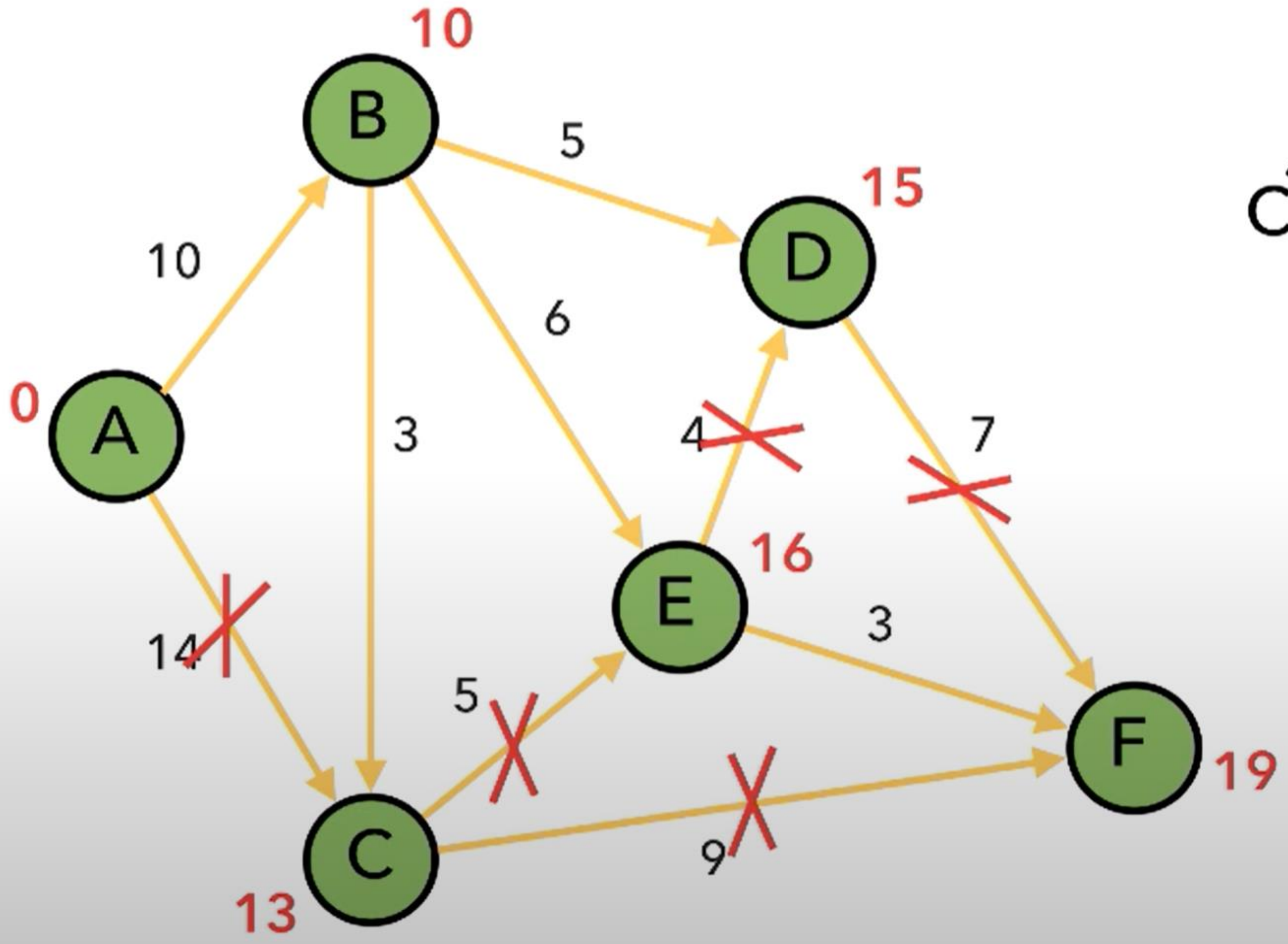


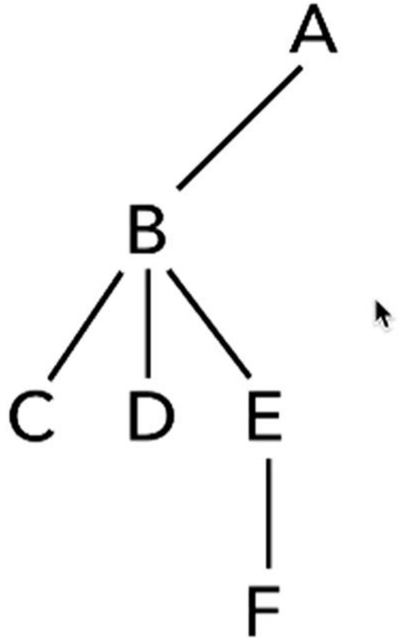
https://www.youtube.com/watch?v=AGE87wMrS20&feature=youtu.be&t=21

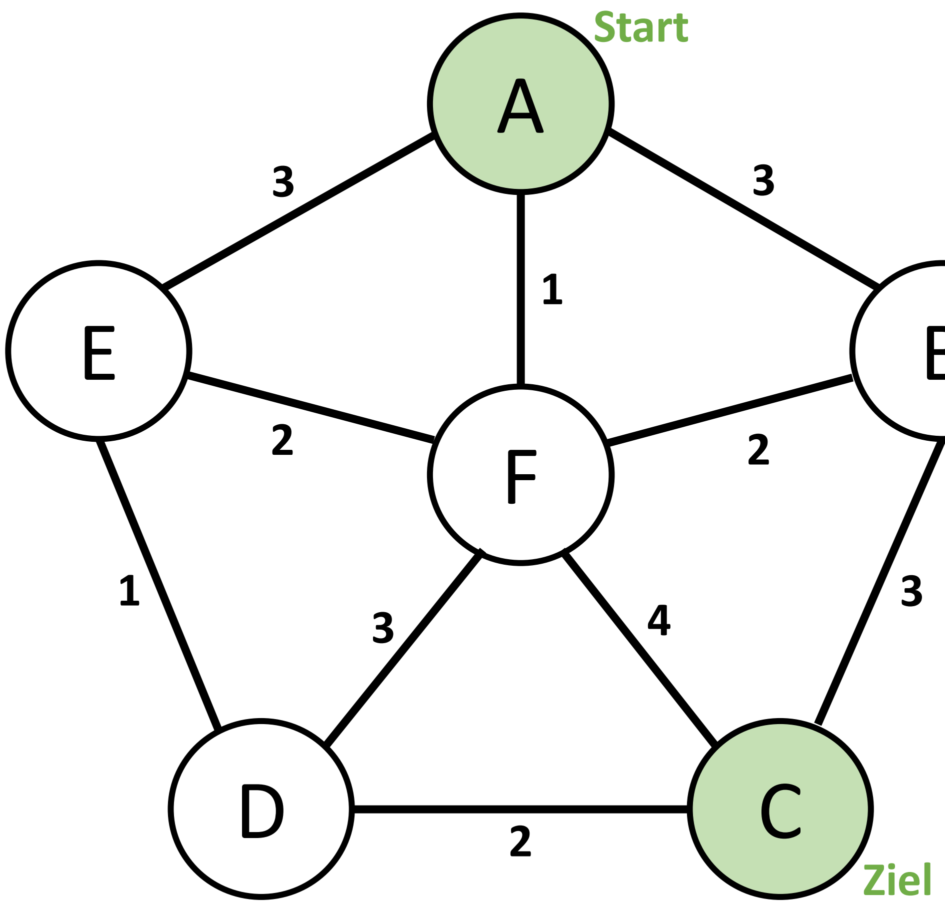
aktueller Knoten

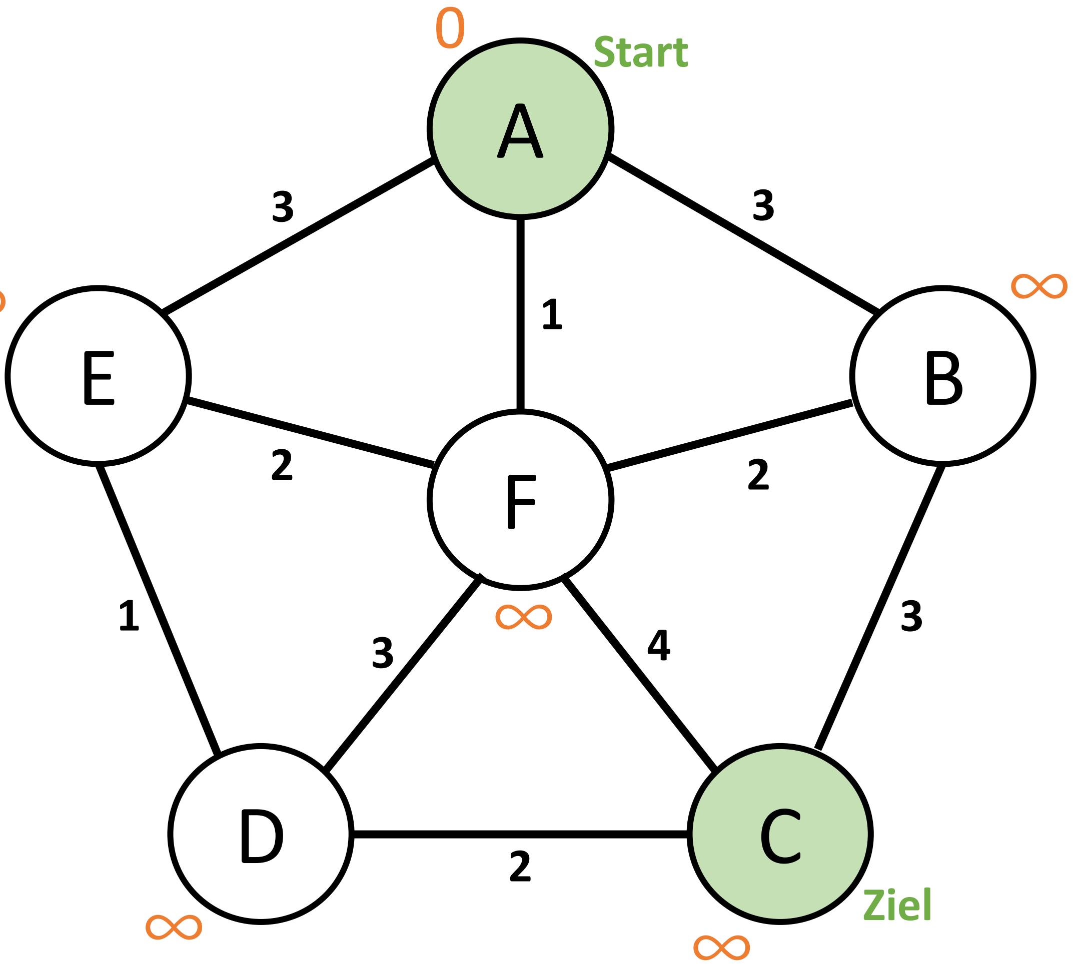
Nachfolger, der noch abgearbeitet werden muss

abgearbeiteter Knoten



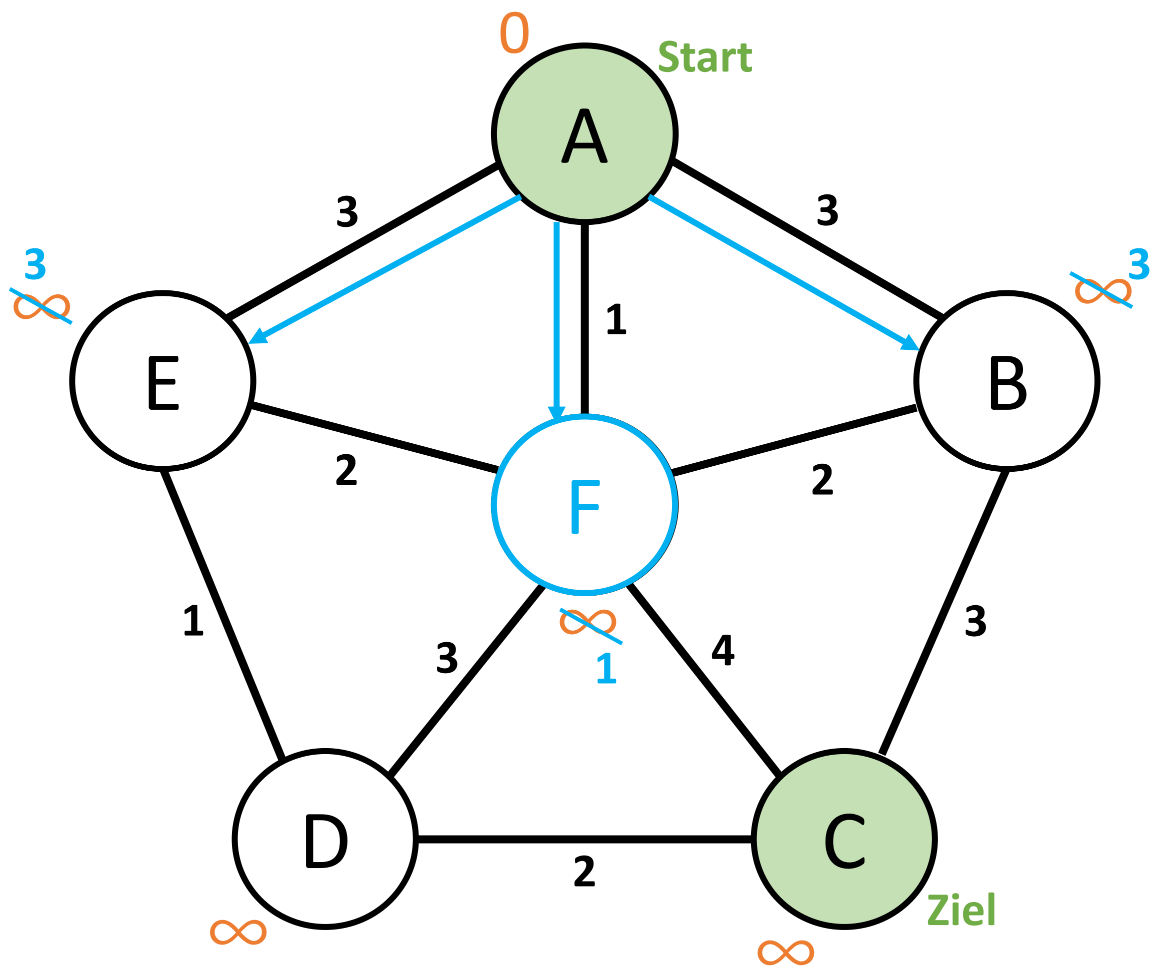



Start
A
3
3
1
E
B
2
F
2
1
3
4
3
D
2
C
Ziel



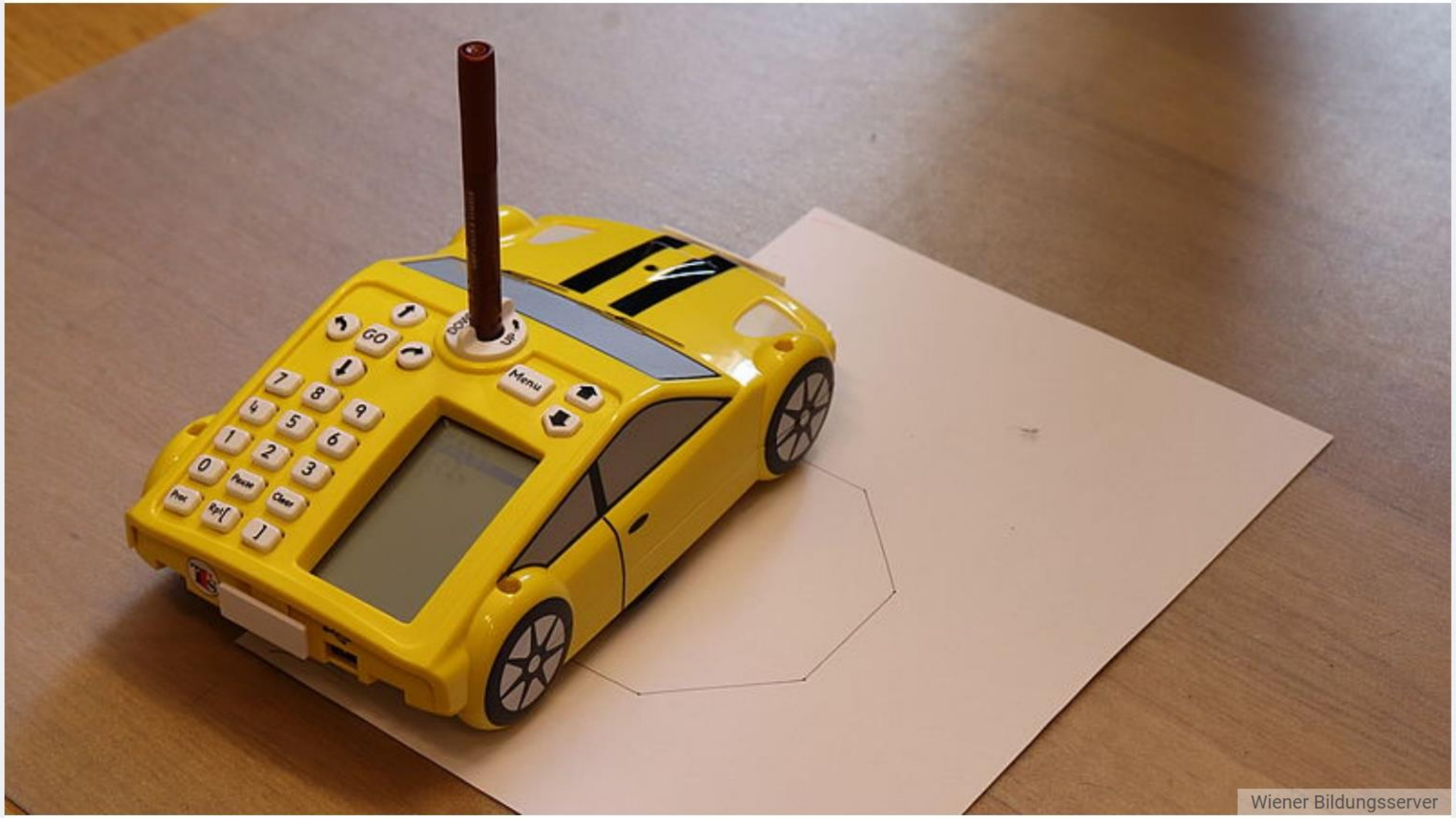
∞, da die Kosten zum Knoten noch unbekannt sind!

Schritt 1 - A

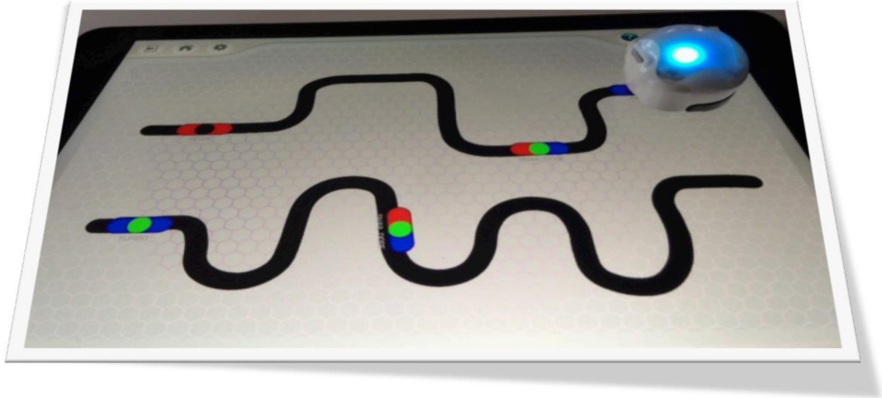


0
Start
A
3
3
1
3
$\cancel{\infty}$
E
B
$\cancel{\infty}$ 3
2
F
2
$\cancel{\infty}$
1
1
3
4
3
D
2
C
Ziel
∞
∞

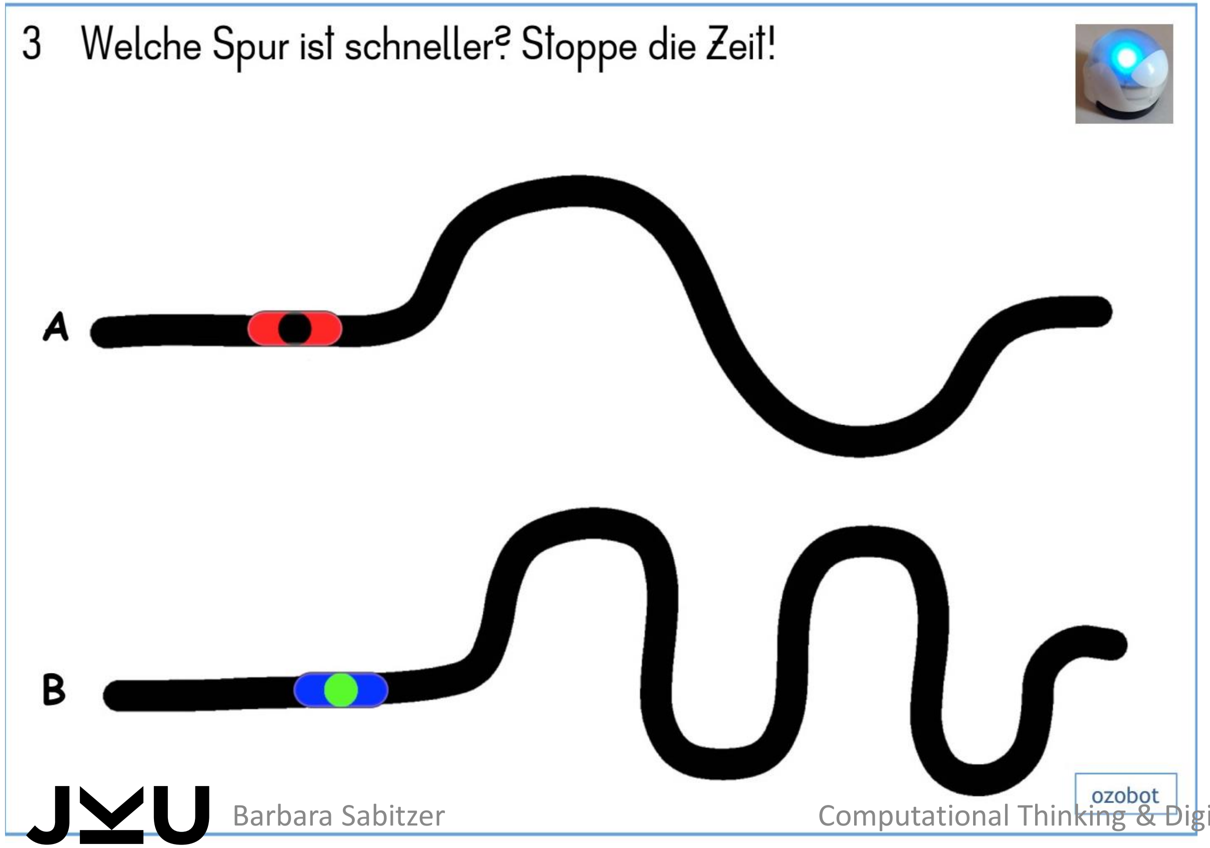
Roboter & Algorithmen

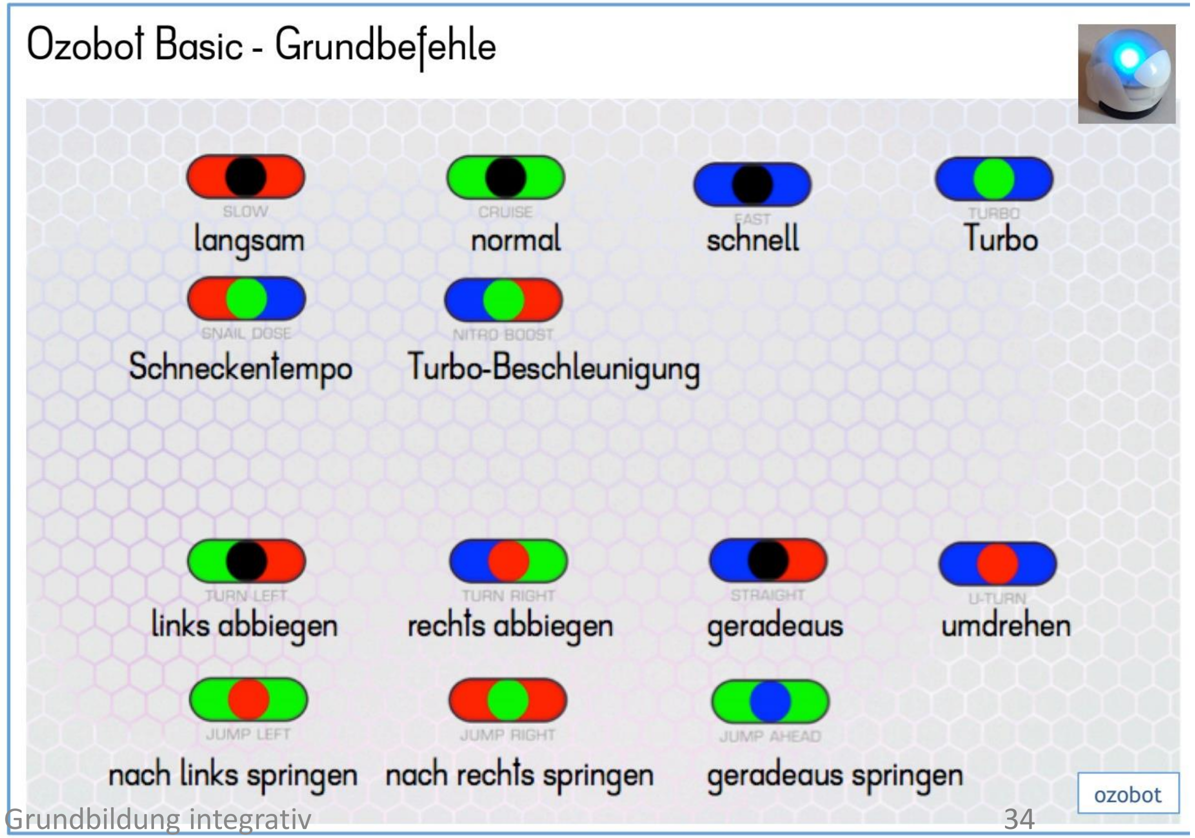


Ozobots
Kürzeste Wege
mit Farben

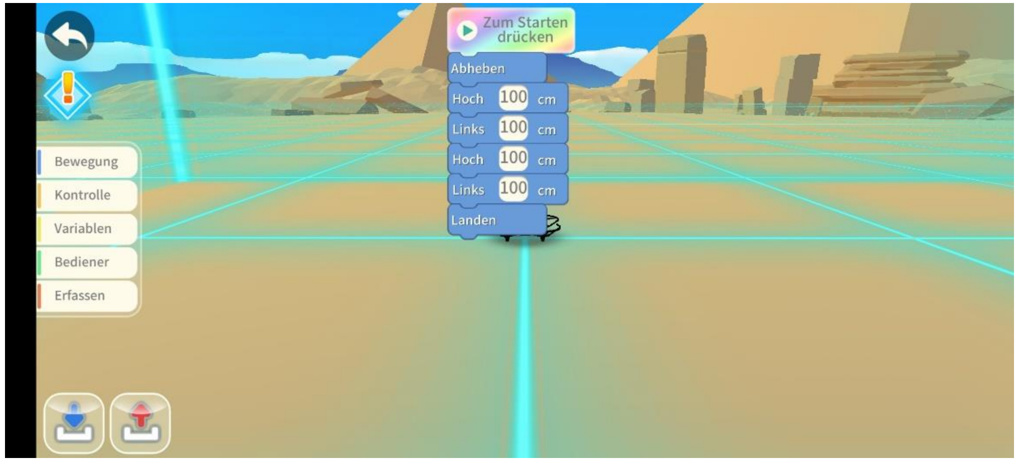


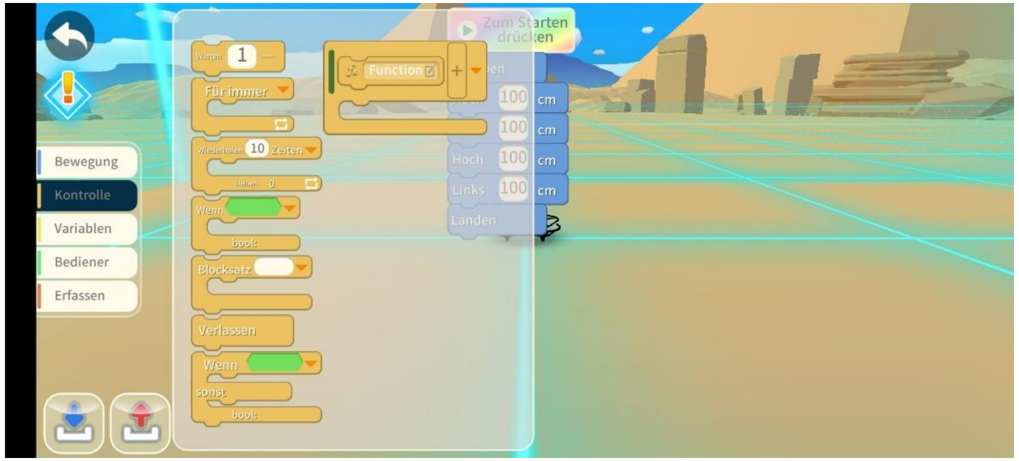
ozobot





Drohnen steuern





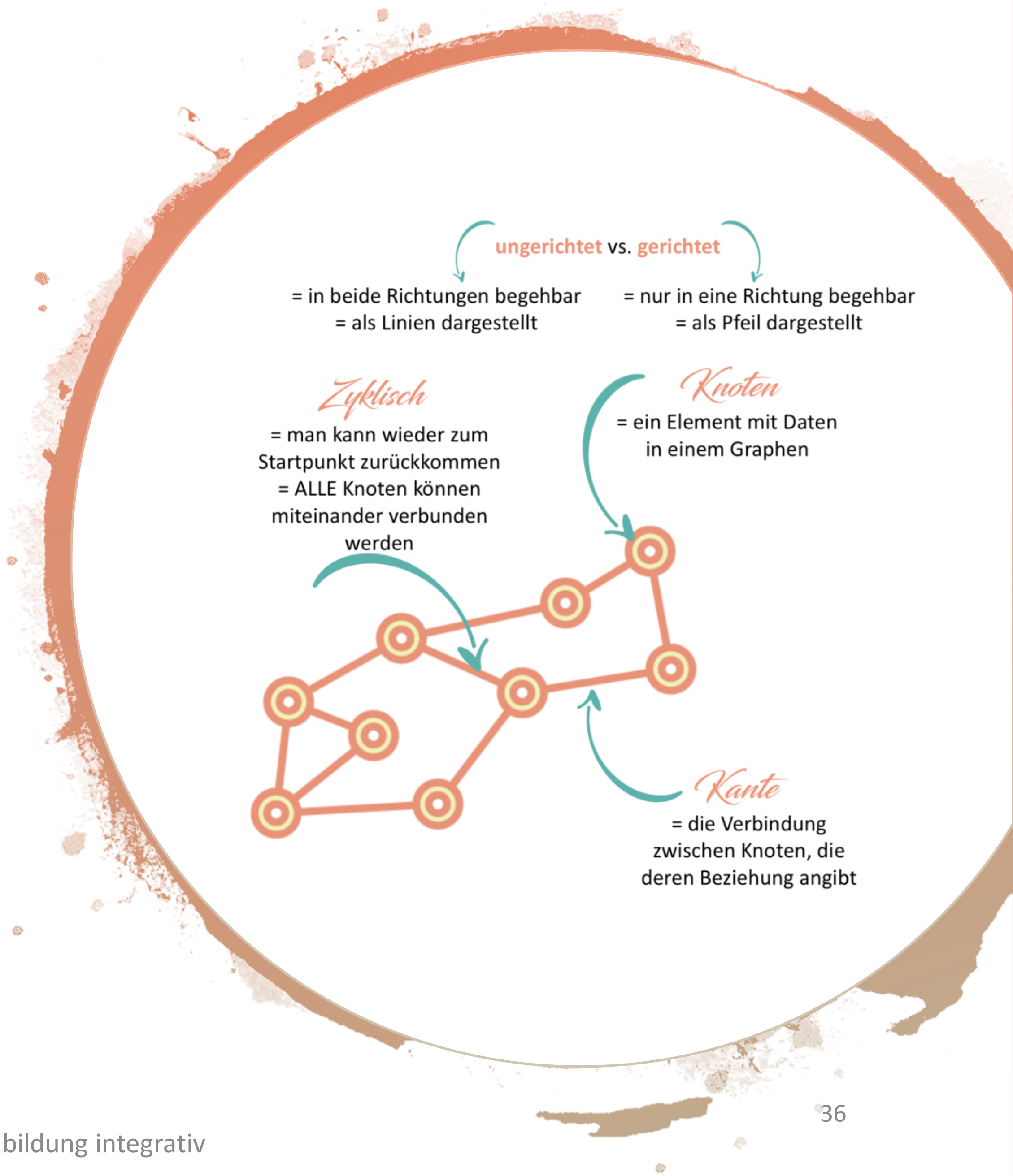
Quelle: eigene Screenshots

Burg Hochosterwitz

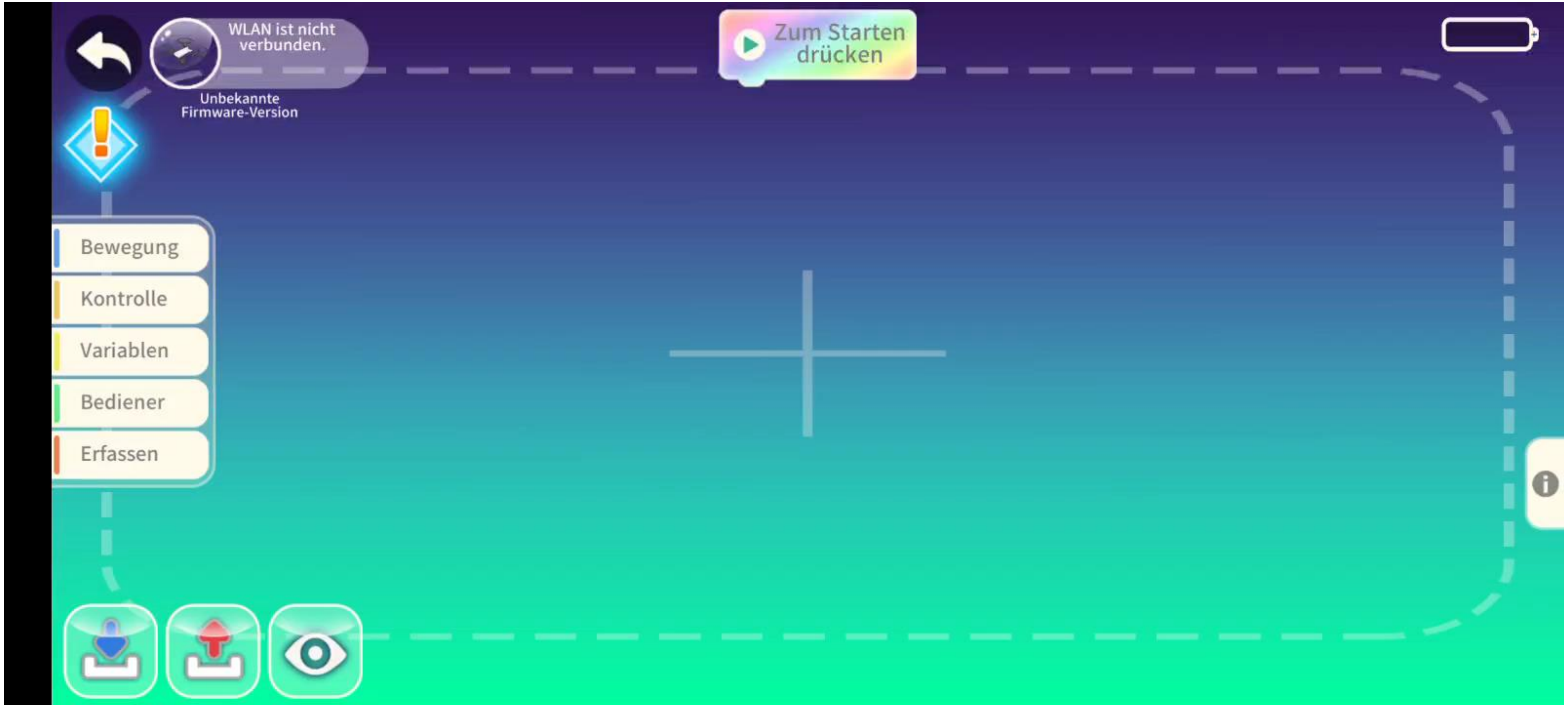
https://www.ryzerobotics.com/de/ironman

Infos & Material

- Zusätzliche Informationen zu
  - Videos zu Kürzeste-Wege-Algorithmen Graphentheorie
  - Definitionen
  - Anwendungsgebiete
- Weiterführende Aufgaben
  - Ideen für Projekte
  - Materialsammlung Graphentheorie & kürzeste Wege
  - Onlineaufgaben (Geogebrabuch)



Drohnen & Programmieren - Tello Edu App & BYOD



*Dijkstra_at.txt - Editor

Datei Bearbeiten Format Ansicht Hilfe

```
def dijkstra(knoten, kanten, start, ziel):
    # knoten ist eine Liste von Knoten
    # kanten ist eine Liste von 3-Tupeln:
    #   (knoten1, knoten2, kosten)
    # start ist der Knoten, in dem die Suche startet
    # ziel ist der Knoten, zu dem ein Weg gesucht werden soll
    # Gibt ein Tupel zurück mit dem Weg und den Kosten
    #
    knotenEigenschaften = [ [i, float('inf'), None, False] for i in knoten if i != start ]
    knotenEigenschaften += [ [start, 0, None, False] ]
    for i in range(len(knotenEigenschaften)):
        knotenEigenschaften[i] += [ i ]

    while True:
        unbesuchteKnotenIterator = filter(lambda x: not x[3], knotenEigenschaften)
        unbesuchteKnoten=list(unbesuchteKnotenIterator)
        if not unbesuchteKnoten: break

        sortierteListe = sorted(unbesuchteKnoten, key=lambda i: i[1])
        aktiverKnoten = sortierteListe[0]
        knotenEigenschaften[aktiverKnoten[4]][3] = True
        if aktiverKnoten[0] == ziel:
                break
        aktiveKanten = list(filter(lambda x: x[0] == aktiverKnoten[0], kanten))
        for kante in aktiveKanten:
                andereKnotenListe=list(filter(lambda x: x[0] == kante[1], knotenEigenschaften))
                andererKnotenId=andereKnotenListe[0][4]
                gewichtSumme = aktiverKnoten[1] + kante[2]
                if gewichtSumme < knotenEigenschaften[andererKnotenId][1]:
                        knotenEigenschaften[andererKnotenId][1] = gewichtSumme
                        knotenEigenschaften[andererKnotenId][2] = aktiverKnoten[4]


    if aktiverKnoten[0] == ziel:
        weg = []
        weg += [ aktiverKnoten[0] ]

        kosten = aktiverKnoten[1]
        while aktiverKnoten[0] != start:
                aktiverKnoten = knotenEigenschaften[aktiverKnoten[2]]
                weg += [ aktiverKnoten[0] ]

        weg.reverse()
        return (weg, kosten)
    else:
        raise "Kein Weg gefunden"

if __name__ == "__main__":
    knoten=['Wien','St. Poelten','Linz','Salzburg','Innsbruck','Bregenz','Klagenfurt','Eisenstadt','Graz']
    wege=[('Wien','St. Poelten',65)
         ,('Wien','Salzburg',301)
         ,('Wien','Innsbruck',477)
         ,('Salzburg','Linz',136)
         ,('Klagenfurt','Graz',136)
         ,('Graz','Linz',220)
         ,('Innsbruck','Eisenstadt',526)
         ,('Innsbruck','Bregenz',188)
         ,('Bregenz','Linz',350)
         ]

    ergebnis=dijkstra(knoten, wege, 'Wien', 'Bregenz')
    print("Kürzester Weg:" + str(ergebnis[0]) + " Kosten: " + str(ergebnis[1]) )
```

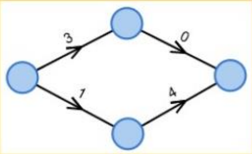
KÜRZESTE-WEGE ALGORITHMEN

JOHANNES KEPLER UNIVERSITÄT LINZ

DIJKSTRA - ALGORITHMUS

DER KLASSIKER UNTER DEN KÜRZESTE-WEGE-ALGORITHMEN

Du würdest gerne wissen, ob du von München aus schneller in Köln bist, wenn du über Stuttgart oder Würzburg fährst? Dann könnte der **Dijkstra-Algorithmus** hilfreich für dich sein! Mit diesem Algorithmus kannst du unter anderem in einem Graphen, dessen Kanten beispielsweise mit den **Distanzen zwischen verschiedenen Städten** beschriftet sind, den kürzesten Weg zwischen zwei Städten ermitteln. Aber auch der kürzeste Weg **von einer Stadt aus zu allen anderen Städten** lässt sich mit dem Dijkstra-Algorithmus leicht bestimmen. Natürlich können die Kantenbeschriftungen auch etwas anderes repräsentieren, wie zum Beispiel die Mautkosten auf den Autobahnen zwischen den Städten.

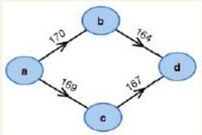


Wichtig beim Dijkstra-Algorithmus ist, dass die Kantenkosten (so nennt man die Kantenbeschriftungen im Allgemeinen) **nicht** negativ sein dürfen.

A* - ALGORITHMUS

EINE INFORMIERTE SUCHE NACH DEM KÜRZESTEN WEG

Du bist auf der Suche nach dem kürzesten Weg von Frankfurt nach München? Dabei könnte dir der Dijkstra-Algorithmus helfen. Allerdings weißt du schon, dass München südlich von Frankfurt liegt. Da der **Dijkstra-Algorithmus** seine Suche **kreisförmig** um den Start ausbreitet, könnte man die **Suche beschleunigen**, denn Städte im Norden möchtest du gar nicht erst betrachten. Der **A*-Algorithmus** bietet sich für dieses Problem an. Er funktioniert ähnlich wie der Dijkstra-Algorithmus, **sucht** allerdings **gezielter**, da für einen Zielknoten, wie hier München, zunächst **geschätzt** wird, wie groß die Distanz sein wird. Da der A*-Algorithmus sehr mit dem Dijkstra-Algorithmus verwandt ist, kannst du auch hier in einem Graphen, dessen Kanten mit den Distanzen zwischen verschiedenen Städten beschriftet sind, den kürzesten Weg zwischen zwei Städten ermitteln.



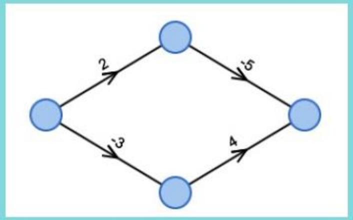
Natürlich können die Kantenbeschriftungen auch beim A*-Algorithmus etwas anderes repräsentieren, wie zum Beispiel die Mautkosten auf den Autobahnen zwischen den Städten. Es ist jedoch wichtig, dass sie **nicht** negativ sind.

BELLMAN-FORD - ALGORITHMUS

KÜRZESTE WEGE UND GÜNSTIGSTE WEGE

Im Gegensatz zu den beiden vorherigen Algorithmen berechnet der Bellman-Ford - Algorithmus **auch** kürzeste Wege, wenn **negative Kantengewichte** gegeben sind.

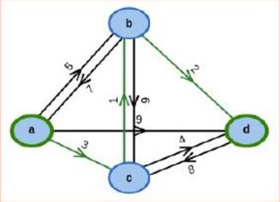
In vielen Anwendungen kann es nützlich sein, den kürzesten Weg von a nach b zu berechnen. Dabei muss die Länge eines Weges **nicht unbedingt die Länge in Metern** sein:
Genauso gut kann man die **Kosten eines Weges** betrachten – man sucht also den günstigsten Weg.



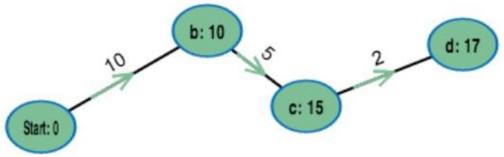
ALGORITHMUS VON FLOYD-WARSHALL

KÜRZESTE PFADE ZWISCHEN ALLEN PAAREN VON KNOTEN

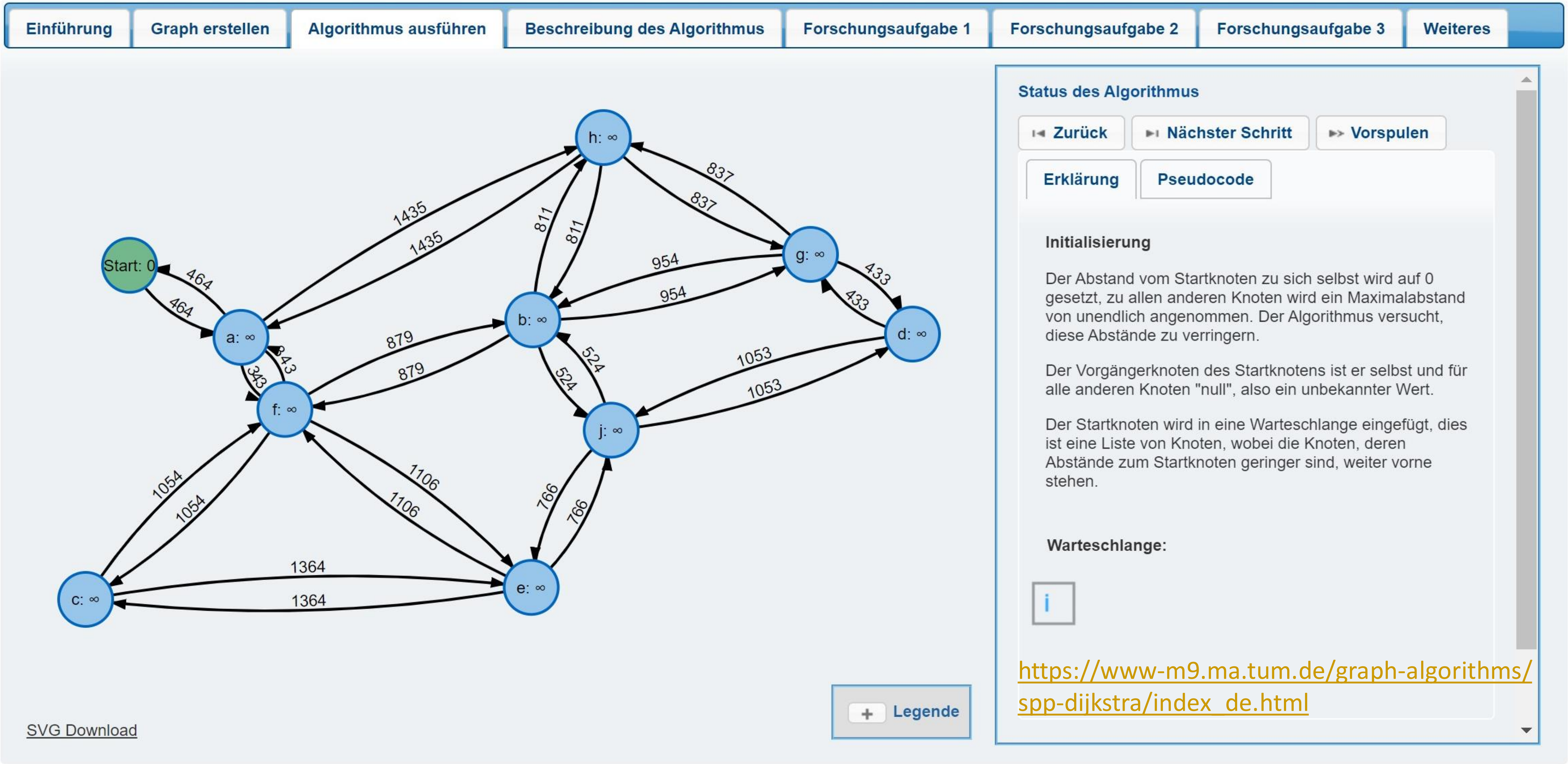
Wenn man die Distanzen zwischen verschiedenen Orten berücksichtigt, zum Beispiel im Bereich Logistik, kommen die Aufgaben über die kürzeste Wege oft vor. In diesen Situationen können die **Orte** als die **Knoten** und die **Kanten** als **Wege** im Graph dargestellt werden. Bei der Lösung vieler Aufgaben muss man die **kürzesten Wege zwischen allen Paaren von Knoten** eines Graphen bestimmen und deren Längen berechnen. Der Floyd-Warshall Algorithmus, der dieses Problem löst, kann auf dem beliebigen Graph ausgeführt werden, wobei es wichtig ist, dass er keine negativen Kreise enthält. Falls es negative Kreise im Graph gibt, dann können die genutzt werden um beliebig kleine (negative) Wege zwischen einigen Knoten zu konstruieren. In diesem Fall kann der Algorithmus keinen optimalen Wert erzeugen.



Quelle: Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik der TU München. "Graphenalgorithmen" 2020. https://www-m9.ma.tum.de/Allgemeines/GraphAlgorithmen . Adaptiert von JKU COOL Lab.



Der Dijkstra-Algorithmus



https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_de.html

Tipps aus der Praxis

Schwierige & Komplexe Inhalte

Verschiedene Zugänge

- Fächer
- Perspektiven
- Aspekte
- Bezug zum Alltag bzw. bekannten Themen

Vielfältige Materialien

- Aussagekräftige Beispiele
- Musteraufgaben – Worked Examples
- Für alle Sinne
- Zum BeGREIFen und ErLeben

Spiralförmiger Aufbau

- Minimalwissen
- Aufbau
- Ausnahmen

Literatur & Links

- Dijkstra-Animation Titelfolie: https://de.wikipedia.org/wiki/Dijkstra-Algorithmus#/media/Datei:Dijkstra_Animation.gif
- Sabitzer, B. (2014). A Neurodidactical Approach to Cooperative and Cross-curricular Open Learning: COOL Informatics. *Habilitation. Alpen-Adria-University Klagenfurt*.