



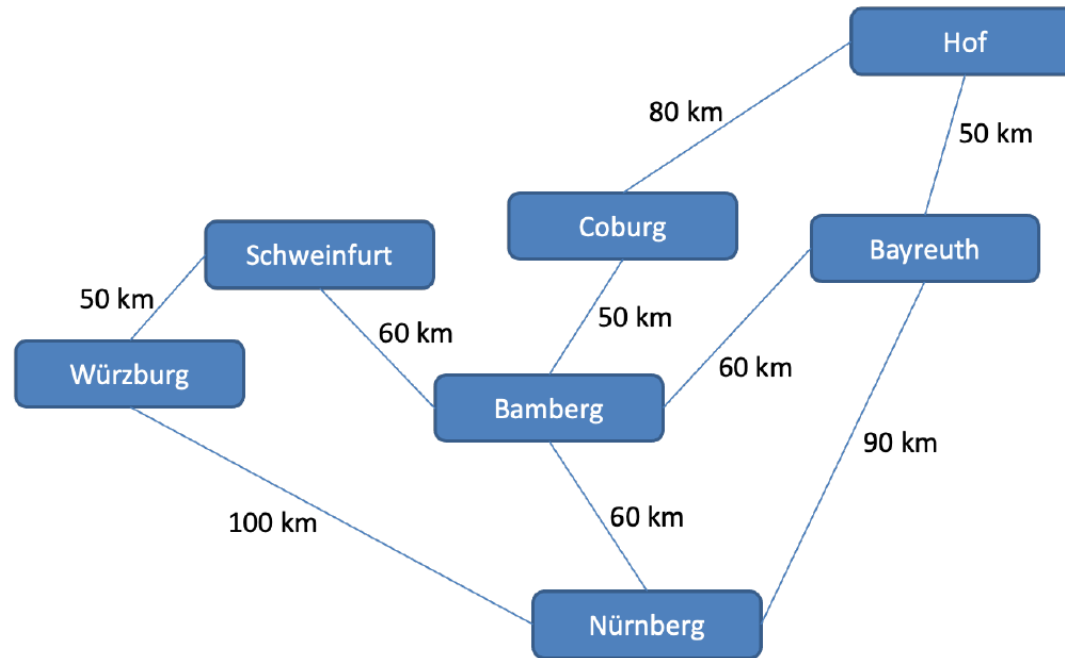
# Praktikumsaufgabe 6 (Dictionaries und Dijkstra)

## Software-Entwicklung

Prof. Dr.-Ing. Jan Paulus

# Praktikumsaufgabe 6 – Dictionaries und Dijkstra

- Folgende Beispielkarte ist gegeben:



- Die Lösung soll für eine beliebige Verbindung aus der Karte den kürzesten Weg finden.
- Die Lösung soll für beliebige Karten der gezeigten Form funktionieren.

# 1. Extrahiere alle Knoten

```
connections = [('Würzburg', 'Schweinfurt', 50),  
               ('Würzburg', 'Nürnberg', 100),  
               ('Schweinfurt', 'Bamberg', 60),  
               ('Bamberg', 'Coburg', 50),  
               ('Bamberg', 'Nürnberg', 60),  
               ('Bamberg', 'Bayreuth', 60),  
               ('Nürnberg', 'Bayreuth', 90),  
               ('Bayreuth', 'Hof', 50),  
               ('Coburg', 'Hof', 80),  
               ]
```

```
nodes = allNodes(connections)  
print(nodes)
```

```
{'Würzburg', 'Coburg', 'Bamberg', 'Nürnberg',  
 'Schweinfurt', 'Hof', 'Bayreuth'}
```

# Dictionaries

---

- Um die Verbindungen zu berechnen, müssen die Knoten mit zusätzlichen Informationen gespeichert werden.
- Dabei müssen auch temporäre Ergebnisse zwischengespeichert werden.
- Ein Dictionary bietet eine bequeme Möglichkeit, über den Knotennamen als Schlüssel auf die Knoten mit allen Informationen zuzugreifen.

## 2. Initialisiere das Dictionary

a) Dictionary nur mit Knotennamen als Keys.

Leeres Dictionary, das später Daten zum jeweiligen Knoten enthalten soll.

```
map = initMap(nodes)
print(map)
```

```
{'Nürnberg': {}, 'Hof': {}, 'Würzburg': {}, 'Bamberg': {},
'Schweinfurt': {}, 'Bayreuth': {}, 'Coburg': {}}
```

b) Füge Verbindungen eines jeden Knotens hinzu.

```
for con in connections:
    addConnection(map, con)
print(map)
```

```
{ 'Bamberg': {'connections':
    [('Schweinfurt', 60), ('Coburg', 50),
    ('Nürnberg', 60), ('Bayreuth', 60)] },
'Hof': {'connections':
    [('Bayreuth', 50), ('Coburg', 80)] },
...
}
```

Mit  
einmal übergeben.

```
for con in connections:  
    addConnection(map, con)
```

wird jeder Tupel aus **connections**

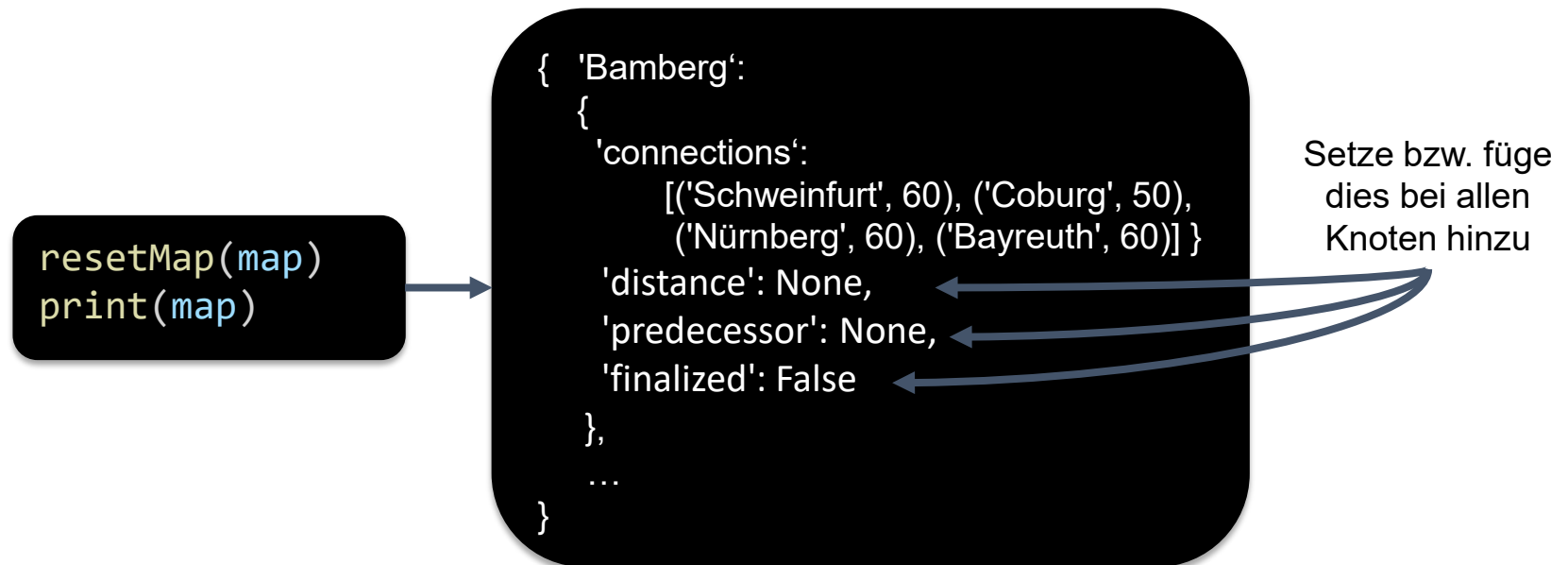
```
[('Würzburg', 'Schweinfurt', 50),  
 ('Würzburg', 'Nürnberg', 100),  
 ('Schweinfurt', 'Bamberg', 60),  
 ('Bamberg', 'Coburg', 50),  
 ('Bamberg', 'Nürnberg', 60),  
 ('Bamberg', 'Bayreuth', 60),  
 ('Nürnberg', 'Bayreuth', 90),  
 ('Bayreuth', 'Hof', 50),  
 ('Coburg', 'Hof', 80),  
 ]
```

1. Füge `'Würzburg'` den Tupel `('Schweinfurt', 50)` hinzu.
2. Füge `'Schweinfurt'` den Tupel `('Würzburg', 50)` hinzu.

Die Funktion **addConnection** soll alle  
anderen Tupel genauso behandeln.

### 3. Zurücksetzen der Daten

- Im Dijkstra-Algorithmus wird folgendes als Nutzdaten hinterlegt
  - Wurde der Knoten bereits fertig verarbeitet?
  - Wie groß ist die (aktuell) kürzeste Distanz zu einem festen Startpunkt?
  - Welcher Knoten ist der Vorgängerknoten, um diese Distanz zu erreichen?
- Da der Algorithmus immer mit einem festen Startpunkt arbeitet, muss er für neue Startpunkte neu initialisiert werden.



## 4. Setzen eines Startpunkts

---

- Später wird von einem Startpunkt iterativ der kürzeste Abstand zu jedem anderen Knoten (ggf. über weitere Knoten) berechnet.
- Der Startpunkt muss vor der Abfrage einer kürzesten Distanz einmal festgelegt werden.
- Mit der Funktion `setStart` setzen wir für den Eintrag, der als Start dienen soll, die Distanz auf 0 und den Vorgänger auf sich selbst.

## 5. Der Dijkstra-Algorithmus

---

- Für einen Startknoten wird nun die kürzeste Distanz zu allen anderen Knoten in der Karte mithilfe `calculateAllDistances` ermittelt.
- Wir brauchen zunächst eine Schleife, die prüft, ob es noch Knoten gibt, die noch nicht fertig verarbeitet wurden (`finalized==False`).
  - Welcher Knoten hat die kleinste Entfernung zum Startknoten? Wähle diesen Knoten.
  - Setze Knoten auf fertig bearbeitet (`finalized=True`)
  - Prüfe alle möglichen (direkte) Verbindungsknoten (`connections`)
    - Ist die Verbindungsdistanz über den aktuellen Knoten kleiner?  
→ Aktualisiere Vorgänger und Distanz des Verbindungsknoten

## 6. Berechne konkrete Verbindung

---

- Setze Karte zurück mittels `resetMap`
- Setze Startknoten mit `setStart`
- Berechne kürzeste Verbindungen vom Startknoten zu allen Knoten mittels `calculateAllDistances`
- Ermittle nun mit `pathFromTo` die eigentliche konkrete Strecke. Laufe dabei vom Zielknoten aus „rückwärts“, indem solange auf den Vorgänger zugegriffen wird, bis der Startknoten erreicht ist. Alle besuchten Knoten werden in einer Liste hinterlegt. Die umgedrehte Liste ist das Ergebnis.

- **Autoren**

Prof. Dr.-Ing. Jan Paulus

- **Impressum**

Prof. Dr.-Ing. Jan Paulus

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik,

Wassertorstraße 10

904489 Nürnberg, Germany

Tel: +49-911-5880-1098

E-mail : [jan.paulus@th-nuernberg.de](mailto:jan.paulus@th-nuernberg.de)

Dieses Skriptum ist nur für den eigenen Gebrauch im Studium gedacht. Eine Weitergabe ist nur mit Zustimmung des Autors gestattet.