Gallenbachalgorithmus - Programm

Umsetzung des
Gallenbacher-Algorithmusses
zu Dijkstra im Programm

Gallenbachalgorithmus - Programm

0. Markiere die **Startstadt** rot, weise ihr die **Kennzahl** 0 zu. Bezeichne diese als **aktuelle Stadt**.

1. Gehe aus von der **aktuellen Stadt** zu allen direkt erreichbaren **Nachbarstädten**

... und führe das Folgende für jede **Nachbarstadt** durch:

Errechne die **Summe** aus der Kennzahl an der **aktuellen Stadt** und der Länge der Strecke dorthin.

- Ist die **Nachbarstadt** bereits rot markiert, mache nichts.
- Hat die **Nachbarstadt** keine **Kennzahl**, weise ihr die **Summe** als **Kennzahl** zu. markiere die Strecke zur **aktuellen Stadt**.
- Hat die **Nachbarstadt** eine **Kennzahl** kleiner der **Summe**, mache nichts.
- Hat die **Nachbarstadt** eine **Kennzahl** größer der **Summe**, streiche die dortige **Kennzahl** sowie die Markierung. Weise ihr danach die **Summe** als neue **Kennzahl** zu. Markiere die Strecke zur **aktuellen Stadt**.

2. Betrachte alle Städte, die zwar eine **Kennzahl** haben, aber noch nicht rot markiert sind. Suche die Stadt mit der kleinsten **Kennzahl**.

3. Bezeichne diese als **aktuelle Stadt**. Weisen mehrere Städte die kleinste **Kennzahl** auf, wähle eine beliebige davon als **aktuelle Stadt**.

4. Markiere die **aktuelle Stadt** rot, zeichne die dort markierte Strecke in rot ein.

5. Falls es noch Städte gibt, die nicht rot markiert sind, weiter bei (1.)

Aus: Gallenbacher, *Abenteuer Informatik, 2. Aufl.*

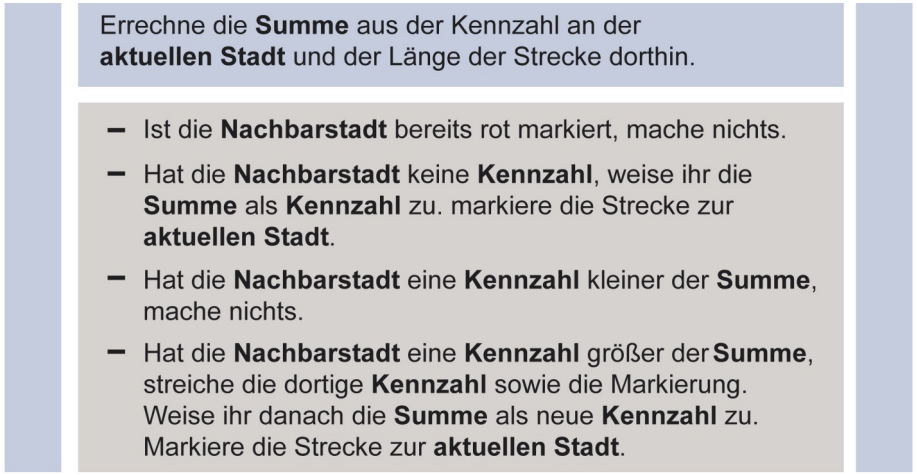
Gallenbachalgorithmus - Programm

0.	Markiere die **Startstadt** rot, weise ihr die **Kennzahl** 0 zu. Bezeichne diese als **aktuelle Stadt**.
1.	Gehe aus von der **aktuellen Stadt** zu allen direkt erreichbaren **Nachbarstädten**

```
def dijkstra(rot, wege, nachfolgeKanten, ziel):
    ...

def aufruf(start,ziel):
    print('kürzester Weg gesucht von '+start+' nach '+ziel)
    weg=dijkstra([start],
                 prioSchlange([[start,0]]),
                 gallenbacherGraph[start],
                 ziel)
```

Gallenbachalgorithmus - Programm




```
# zunaechst muss getestet werden, ob der Ort schon enthalten ist.
        for weg in self.wege:
            if neuerWeg[0]==weg[0]:
                # es gibt schon einen Weg zum Knoten
                if self.wegLaenge(neuerWeg)>=self.wegLaenge(weg):
                    # dann ist nichts zu tun
                    return
                else:
                    # der alte Weg muss entfernt werden
                    self.wege.remove(weg)
```

(Ausschnitt aus der Bearbeitung in der Proritätswarteschlange)

Gallenbachalgorithmus - Programm

2.	Betrachte alle Städte, die zwar eine **Kennzahl** haben, aber noch nicht rot markiert sind. Suche die Stadt mit der kleinsten **Kennzahl**.
3.	Bezeichne diese als **aktuelle Stadt**. Weisen mehrere Städte die kleinste **Kennzahl** auf, wähle eine beliebige davon als **aktuelle Stadt**.
4.	Markiere die **aktuelle Stadt** rot, zeichne die dort markierte Strecke in rot ein.
5.	Falls es noch Städte gibt, die nicht rot markiert sind, weiter bei (1.)

Aus: Gallenbacher, *Abenteuer Informatik, 2. Aufl.*

```
def dijkstra(rot, wege, nachfolgeKanten, ziel):
    while len(wege.gibWege())>0:
        aktuellerWeg=wege.ersterRaus()
        if ziel==aktuellerWeg[0][0]: return aktuellerWeg
        for kante in nachfolgeKanten:
            if not (kante[0] in rot):
                wege.fuegeEin(aktuellerWeg,kante)
        neuer = wege.gibWege()[0][0][0]
        if not neuer in rot:
            rot.append(neuer)
        nachfolgeKanten=gallenbacherGraph[neuer]
    return False
```