# H2.1 Dijkstra-Algorithmus nachvollziehen

2.1 a)

|  |  |  |
| --- | --- | --- |
|  | **Liste besuchter Knoten** | **To-Do Liste** |
| Zeile 28 | [] | [(0.0, R2)] |
| Iteration 1 | [R2] | [(1.0, R5), (2.0, R1), (6.0, R3)] |
| Iteration 2 | [R2, R5] | [(2.0, R1), (6.0, R3), (7.0, R4)] |
| Iteration 3 | [R2, R5, R1] | [(6.0, R3), (6.0, R4)] |
| Iteration 4 | [R2, R5, R1, R3] | [(6.0, R4)] |

* 6.0 ist die berechnete Länge des kürzesten Weges von **R2** nach **R4**.
* In Iteration 3 findet ein update statt, da der Eintrag an der Stelle todo\_list[2][0] größer ist als new\_dist. Statt dist = 7.0 wird new\_dist = 6.0 für den kürzesten Weg nach **R4** eingefügt.

b) Kombinationen aus Start- und Zielknoten bei denen während der Ausführung des Dijkstra-Algorithmus die Situation auftritt, dass **ein Knoten in der ToDo-Liste aktualisiert wird** und dabei einen anderen Eintrag in der Liste überholt.

|  |  |  |
| --- | --- | --- |
| Startknoten | Zielknoten | Fälle, in denen ein Knoten aktualisiert wird |
| R4 | R2 | old distance: 9.0 , update: (6.0, 'R2') |
| R5 | R1 | old distance: 7.0, update: (3.0, 'R1') |
| R5 | R3 | old distance: 7.0, update: (3.0, 'R1') |
| R5 | R4 | old distance: 7.0, update: (3.0, 'R1') |

# H2.2 Dijkstra-Algorithmus erweitern

Start: 10905268916

Ziel: 821857536

# H2.3 Sortierte Listen vereinigen

Gegeben sind zwei sortierte Listen von Zahlen *s* = [1,4,4,5,8,9,9] und *t* = [2,5,6]. Diese Liste sollen zu einer neuen Liste zusammengefügt werden, die ebenfalls sortiert ist.

1. Beschreibt zunächst (in wenigen Sätzen, aber möglichst präzise) eure Strategie, wie ihr dieses Problem in linearer Laufzeit bzgl. der Länge der Ergebnisliste (also linear in *len*(*s*)+*len*(*t*)) lösen könnt.

* Man sieht sich die Liste s am Index 0 an.
* Dann sieht man sich die Liste t am Index 0 an.
* Man vergleicht die beiden Zahl am Index 0 der jeweiligen Listen s und t.
  + Ist t[0] kleiner als s[0], dann wird t[0] an die neue Liste angefügt.
  + Ist s[0] kleiner als t[0], dann wird s[0] an die neue Liste angefügt.
* Das soll so lange gemacht werden, bis jede Position von t[0] und jede Position von s[0] betrachtet wurde. Dafür wird zu Beginn der Zähler für die Indices i und j auf 0 gesetzt, also i = 0 und j = 0.
* Gib die Liste „neue Liste“ als Ergebnis zurück.

**H2.4 Listen rotieren (nur EinfAlg)**

Definiert in Python eine Prozedur rotate\_list (also ein Unterprogramm ohne Rückgabewert), die eine Liste l und eine Ganzzahl s als Parameter übergeben bekommt und die Liste um s Schritte "rotiert", wobei s>0 nach rechts rotiert und s<0 nach links. Eine Rotation um einen Schritt nach rechts bedeutet, dass alle Elemente um eine Position nach rechts verschoben werden und das letzte Element zum ersten wird. Analog verschiebt eine Rotation nach links alle Elemente um eine Position nach links und das erste wird zum letzten. Die Prozedur soll die übergebene Liste dabei mutieren.