Praxis: Dijkstra-Algorithmus

Philipp Hanisch, Valentin Roland

6. Oktober 2022

Python-Grundlagen

Gliederung

1. Problemstellung

2. Der Dijkstra-Algorithmus

3. Aufgaben

Problemstellung

Das Problem

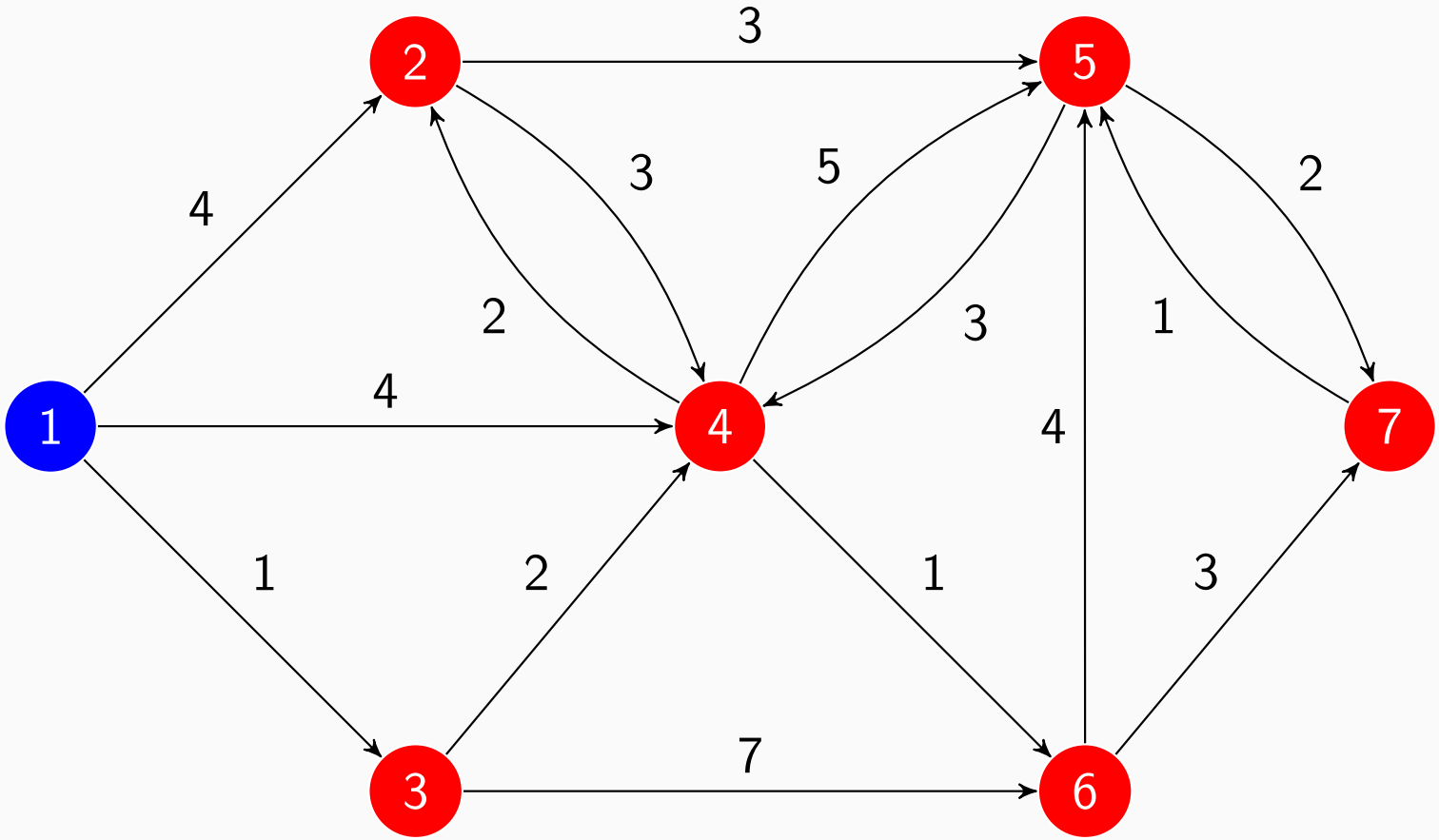
gegeben:

- ein gerichteter, kantenbewerteter Graph $G = (V, E, d)$
- ein Startknoten S

gesucht:

- kürzeste Wege zu den anderen Knoten

Das Problem



Modellierung

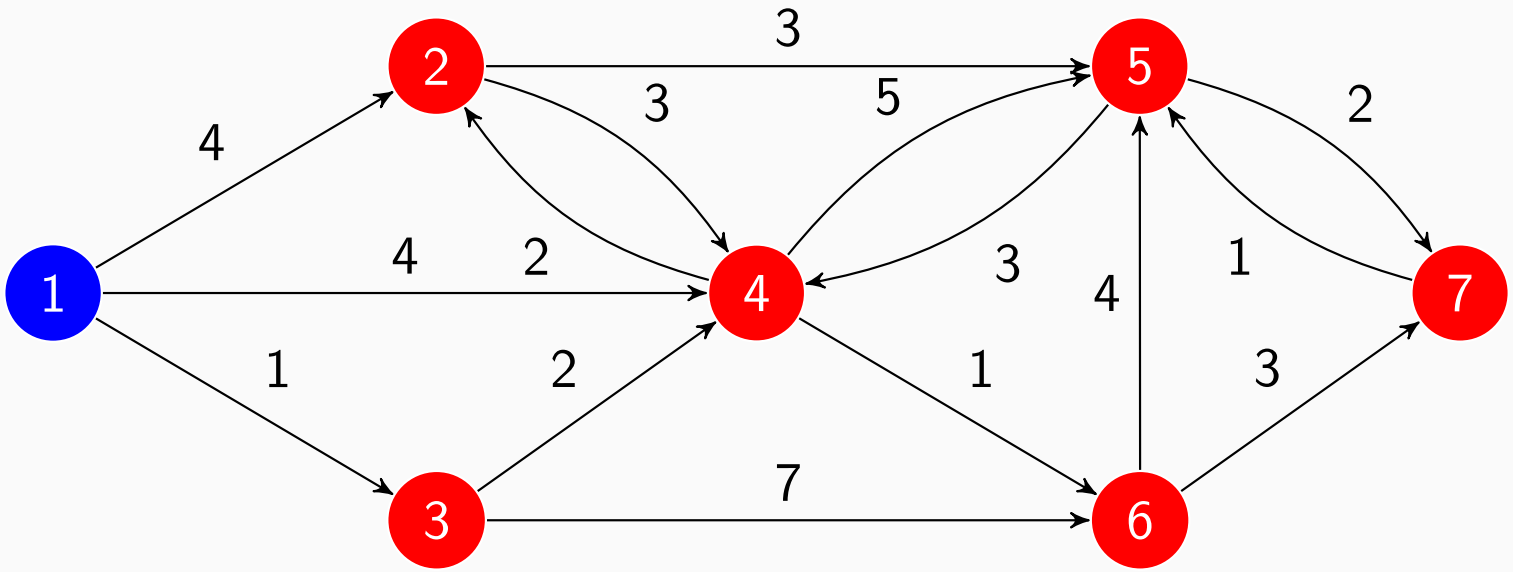
- Knotenmenge als Liste
  hier: `[1, 2, 3, 4, 5, 6, 7]`
- Kantenbewertung als Dictonary: (Knoten, Knoten) $\rightarrow$ Zahl hier:
  `{(1,2):4, (1,3):1, (1,4):4, (2,4):3, (2,5):3, (3,4):2, ...}`

Der Dijkstra-Algorithmus

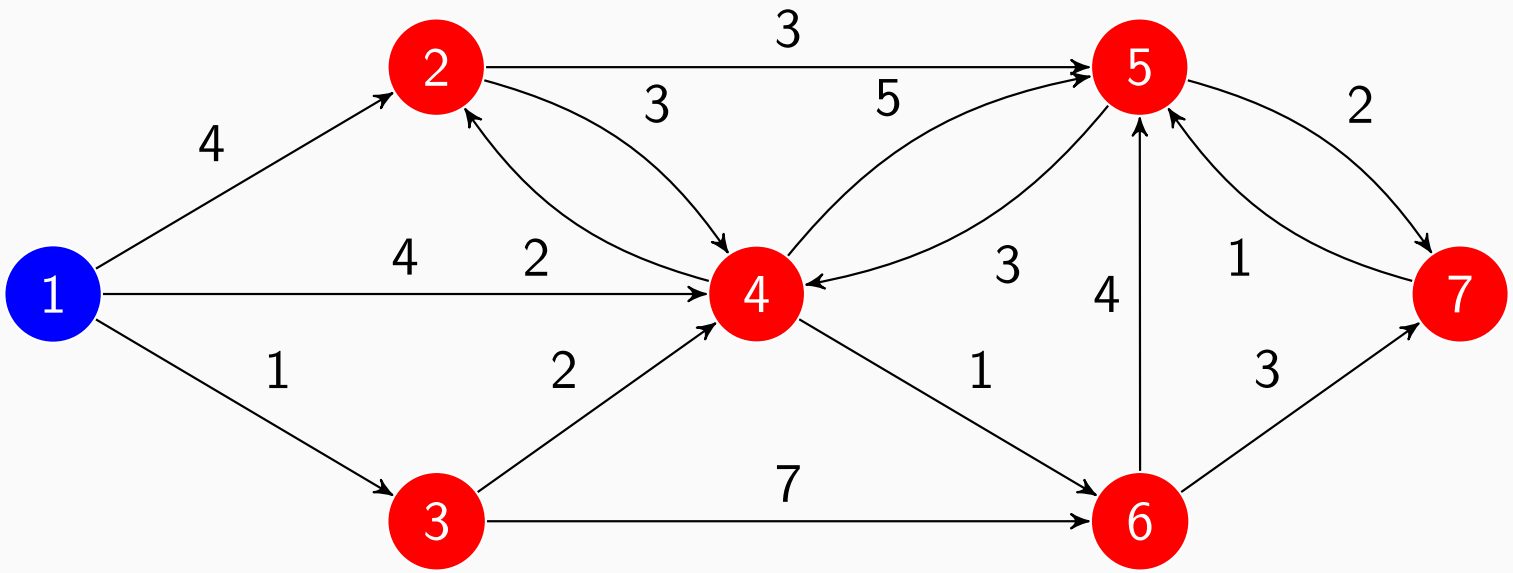
Die Idee

1. **Initialisierung:**
2. **Schleife:**
   - Wähle erreichbaren Knoten mit minimaler Entfernung
   - Markiere Knoten als besucht
   - Aktualisiere die Information der anderen Knoten
3. **Ausgabe**

Beispiel

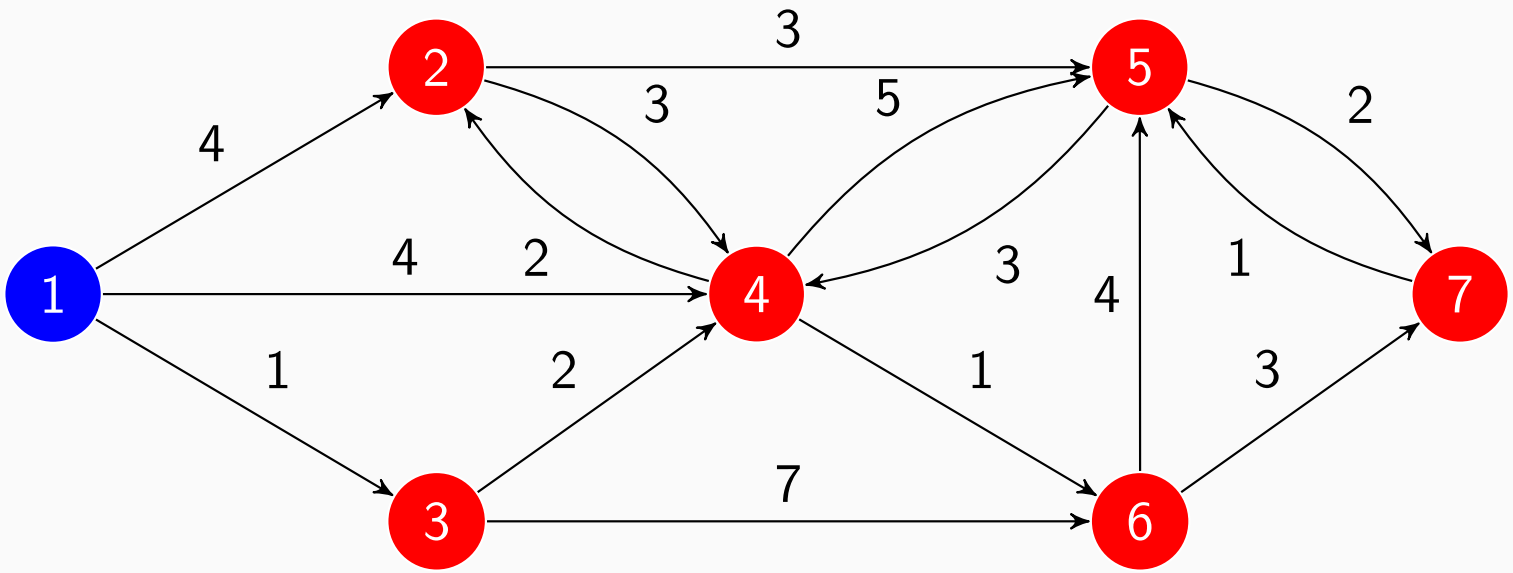


Beispiel



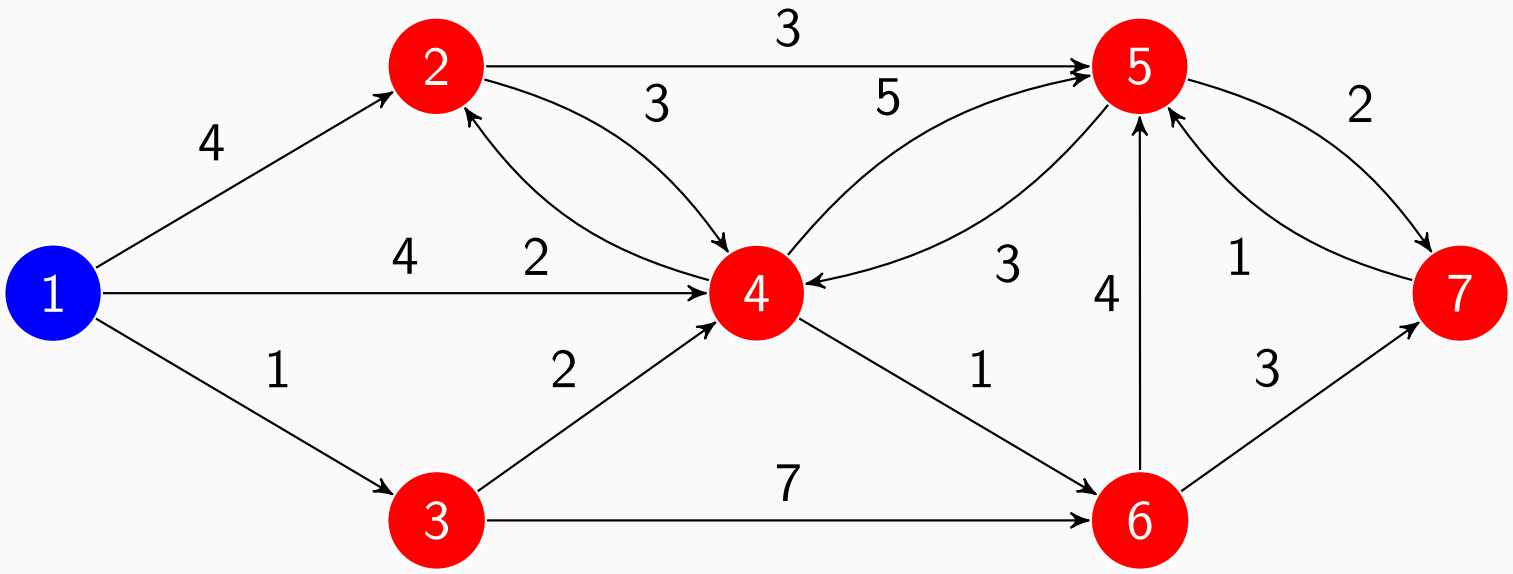
l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7

Beispiel



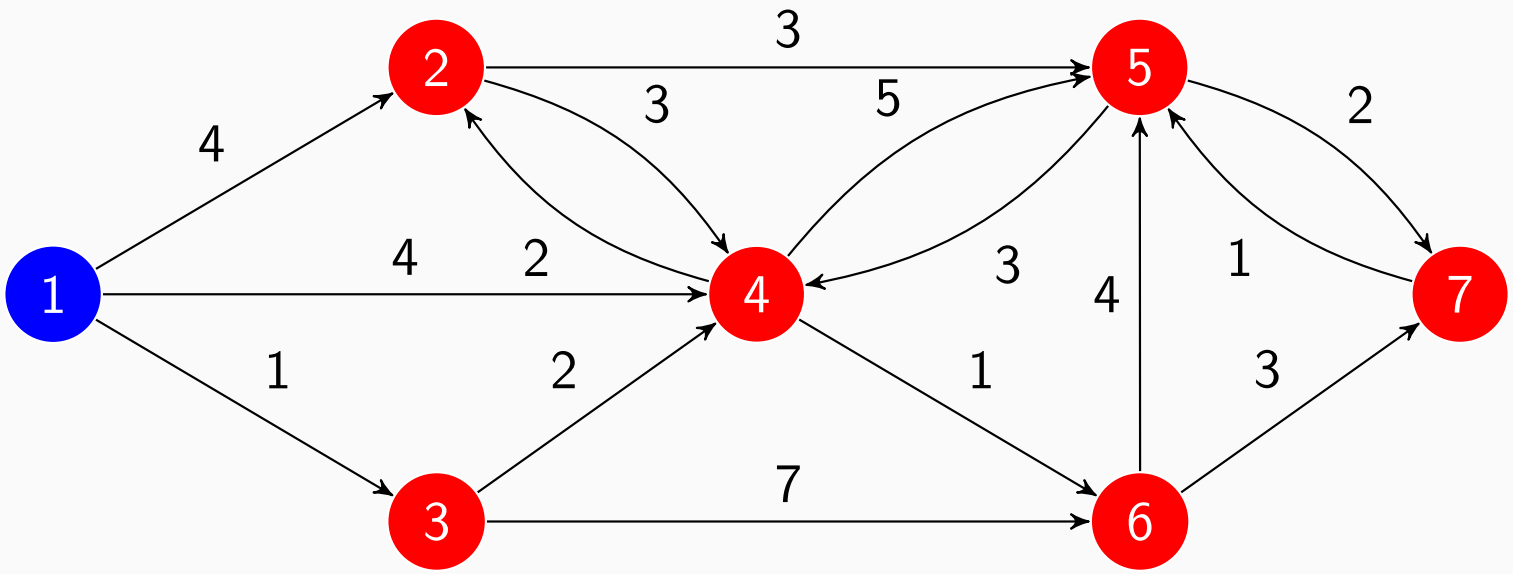
l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-

Beispiel



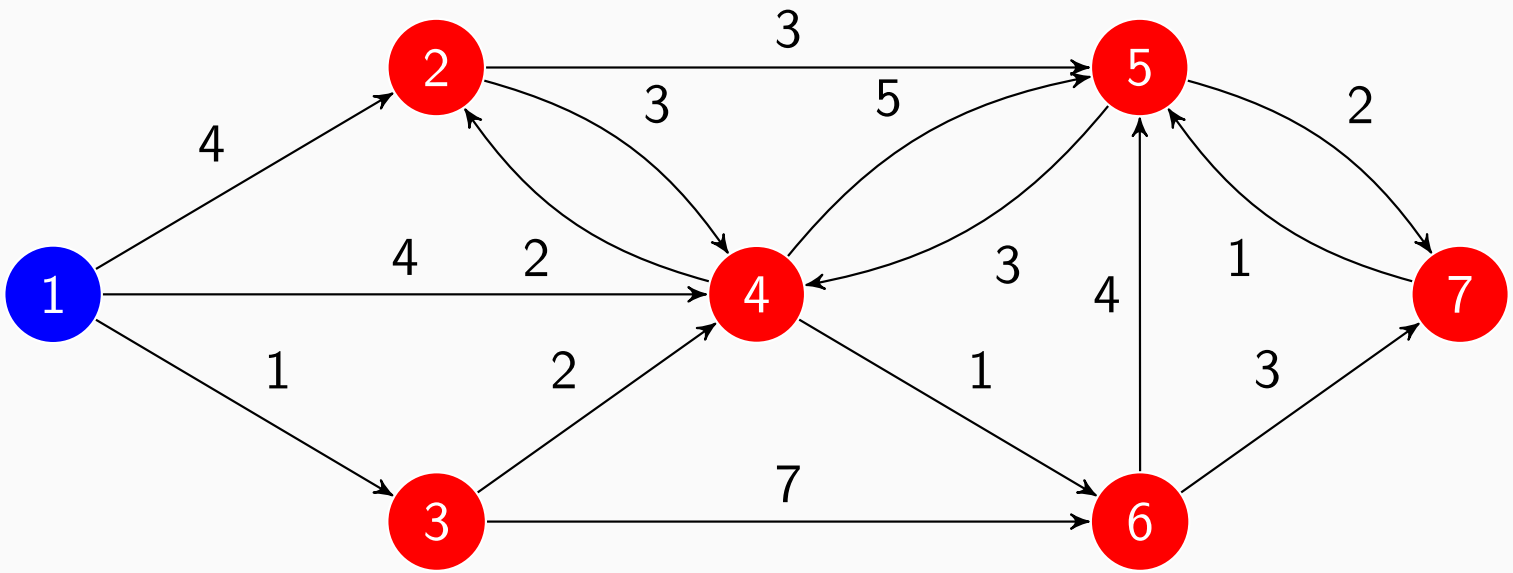
l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-
*	4	1	4	∞	∞	∞	-	1	1	1	-	-	-

Beispiel



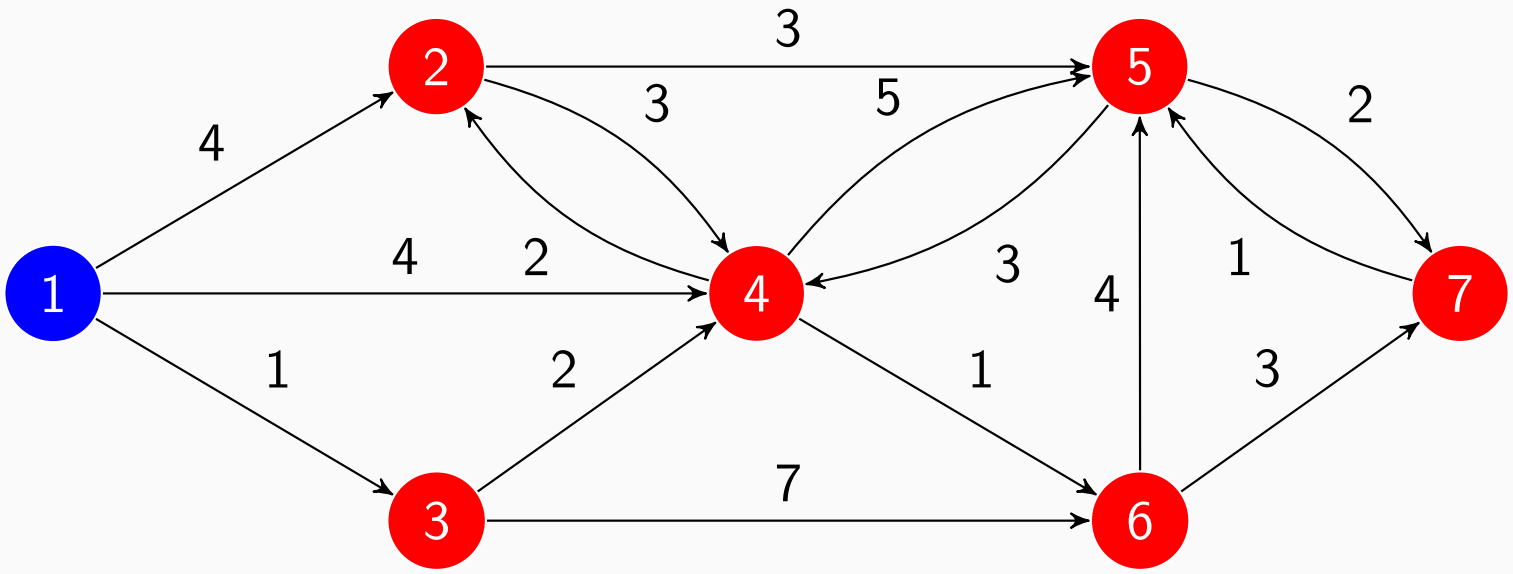
l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-
*	4	1	4	∞	∞	∞	-	1	1	1	-	-	-
*	4	*	3	∞	8	∞	-	1	1	3	-	3	-

Beispiel



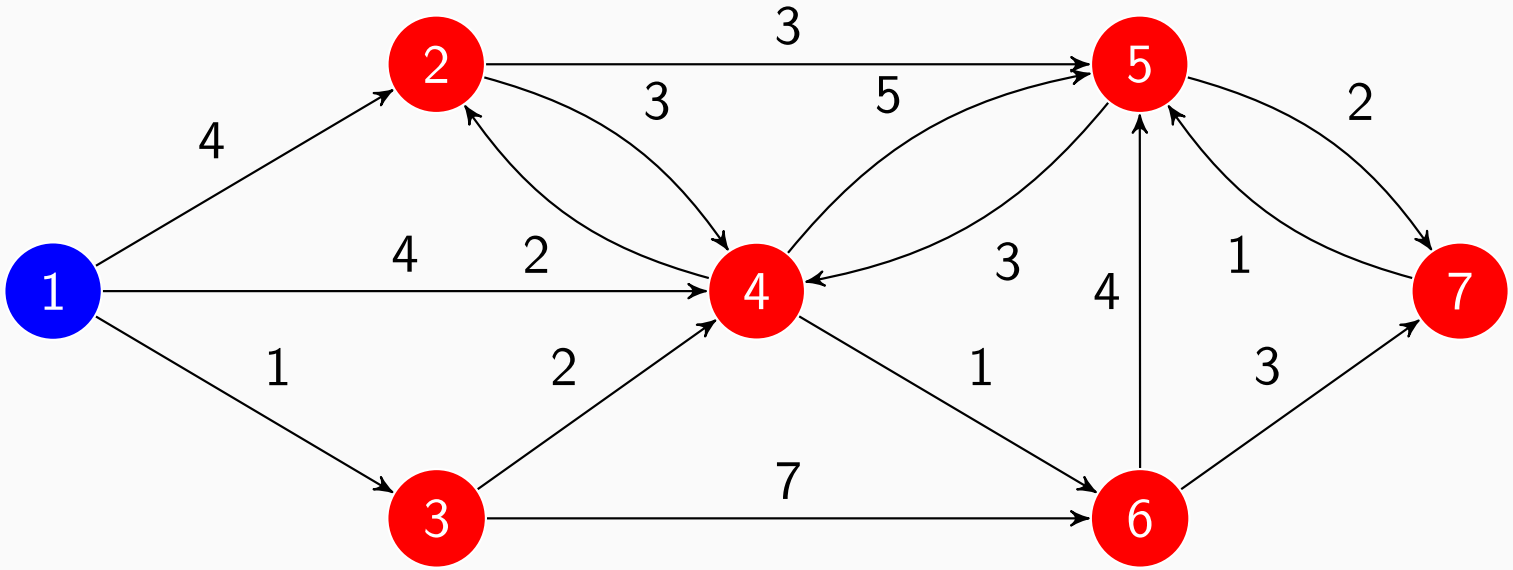
l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-
*	4	1	4	∞	∞	∞	-	1	1	1	-	-	-
*	4	*	3	∞	8	∞	-	1	1	3	-	3	-
*	4	*	*	8	4	∞	-	1	1	3	4	4	-

Beispiel



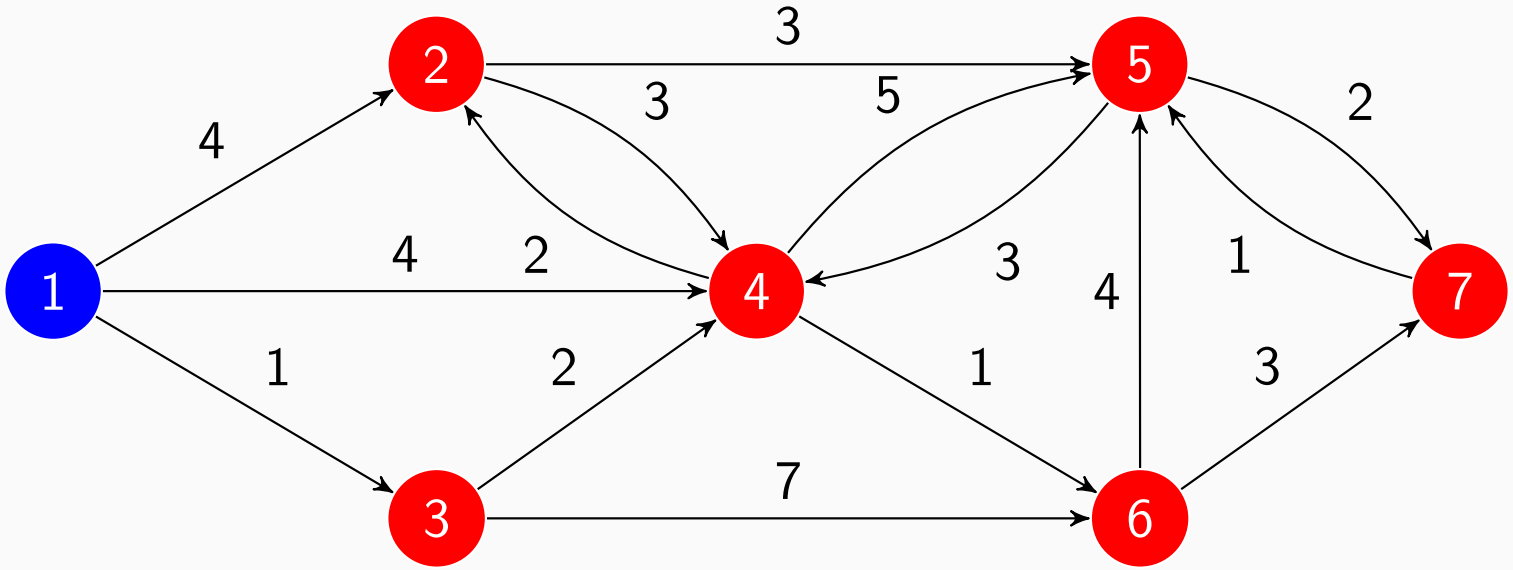
l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-
*	4	1	4	∞	∞	∞	-	1	1	1	-	-	-
*	4	*	3	∞	8	∞	-	1	1	3	-	3	-
*	4	*	*	8	4	∞	-	1	1	3	4	4	-
*	*	*	*	7	4	∞	-	1	1	3	2	4	-

Beispiel



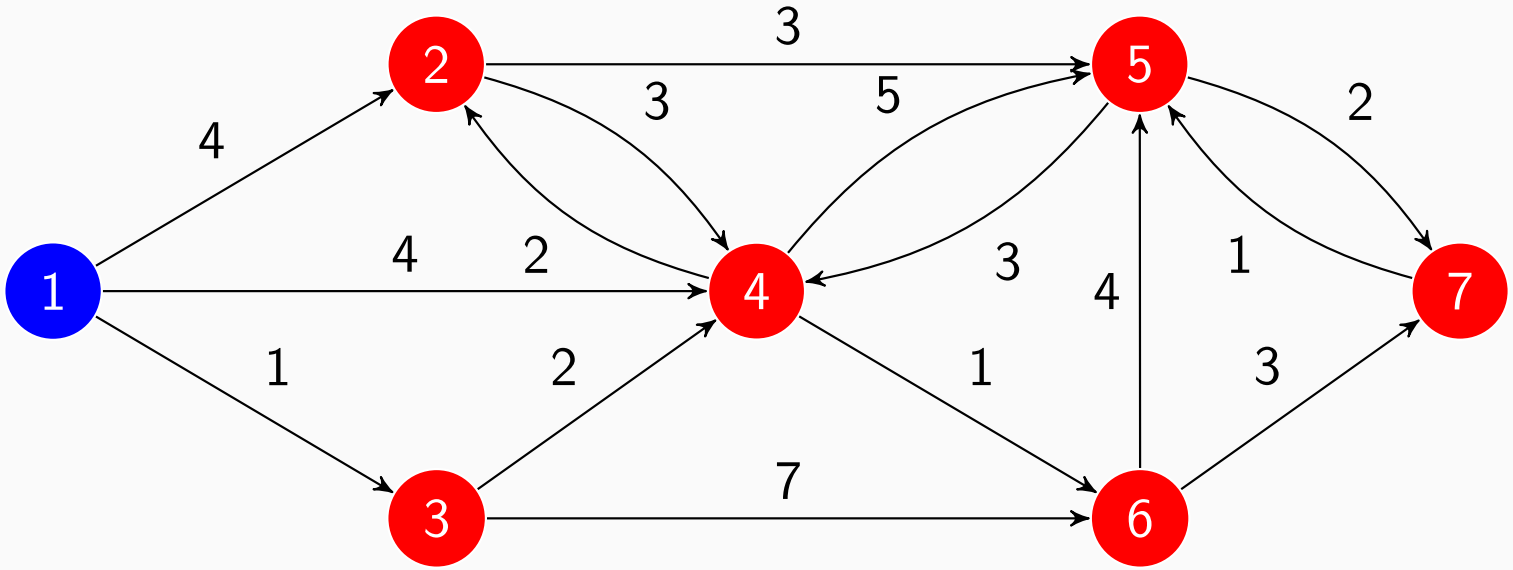
l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-
*	4	1	4	∞	∞	∞	-	1	1	1	-	-	-
*	4	*	3	∞	8	∞	-	1	1	3	-	3	-
*	4	*	*	8	4	∞	-	1	1	3	4	4	-
*	*	*	*	7	4	∞	-	1	1	3	2	4	-
*	*	*	*	7	*	7	-	1	1	3	2	4	6

Beispiel



l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-
*	4	1	4	∞	∞	∞	-	1	1	1	-	-	-
*	4	*	3	∞	8	∞	-	1	1	3	-	3	-
*	4	*	*	8	4	∞	-	1	1	3	4	4	-
*	*	*	*	7	4	∞	-	1	1	3	2	4	-
*	*	*	*	7	*	7	-	1	1	3	2	4	6
*	*	*	*	*	*	7	-	1	1	3	2	4	6

Beispiel



l_1	l_2	l_3	l_4	l_5	l_6	l_7	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	∞	∞	∞	∞	∞	∞	-	-	-	-	-	-	-
*	4	1	4	∞	∞	∞	-	1	1	1	-	-	-
*	4	*	3	∞	8	∞	-	1	1	3	-	3	-
*	4	*	*	8	4	∞	-	1	1	3	4	4	-
*	*	*	*	7	4	∞	-	1	1	3	2	4	-
*	*	*	*	7	*	7	-	1	1	3	2	4	6
*	*	*	*	*	*	7	-	1	1	3	2	4	6
*	*	*	*	*	*	*	-	1	1	3	2	4	6

Bezeichner

- Graph $G = (V, E, d)$
- Menge K an erreichbaren Knoten
- Startknoten S
- bisher kürzeste Entfernungen $l(k)$ für Knoten $k \in V$
- derzeitiger Vorgänger $p(k)$ für Knoten $k \in V$

Der Dijkstra-Algorithmus

1. **Initialisierung:**
   - $l(k) = \infty$ für alle Knoten $k \in V \setminus \{S\}$
   - $l(S) = 0$
   - $p(k) =$ `None` für alle Knoten $k \in V$
   - $K = \{S\}$
2. **Schleife:** Solange $K \neq \varnothing$ ...
   - Wähle einen erreichbaren Knoten v mit minimaler Entfernung:
     $v \in K$ mit $l(v) = min\{l(k) : k \in K\}$
   - Markiere Knoten v als besucht:
     $K := K \setminus \{v\}$
   - Aktualisiere die Information der anderen Knoten $k \in V$:
     Wenn $l(v) + d(v, k) < l(k)$:
     dann setze $l(k) := l(v) + d(v, k)$ und $p(k) := v$
     Füge k ggf. zu K hinzu
3. **Ausgabe**

Aufgaben

Aufgaben

Implementiert den Dijkstra-Algorithmus!