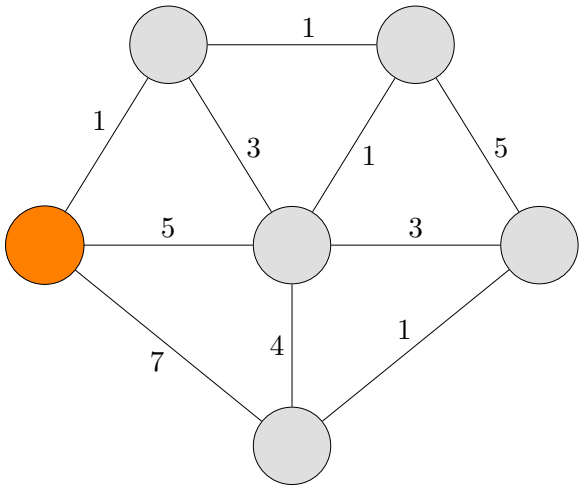
Dijkstra-Algorithmus 1959, Dijkstra (1930-2002), sprich „Deiks-tra“



Vom Startknoten (orange) suchen wir die kürzesten Pfade zu den übrigen Knoten.

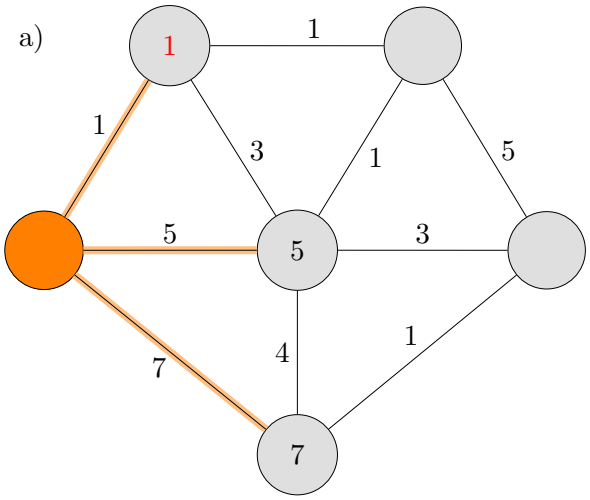
1. Veranschaulichung:
Die Pfadangaben stellen Fadenlängen dar.
Der Startknoten wird langsam gehoben, bis sich ein Faden bis zum nächst liegenden Knoten spannt.
Der Startknoten wird weiter gehoben, bis sich ein Faden bis zum zweitnächst liegenden Knoten spannt, usw.

2. Veranschaulichung:
Die Pfadangaben sind die Längen (in *LE*) von Zündschnüren, die durch die Knoten verbunden sind.
Wir entzünden den Startknoten.
Die Zündschnüre brennen mit der konstanten Geschwindigkeit 1 *LE* pro Sekunde ab.

Erläutere den weiteren Ablauf.

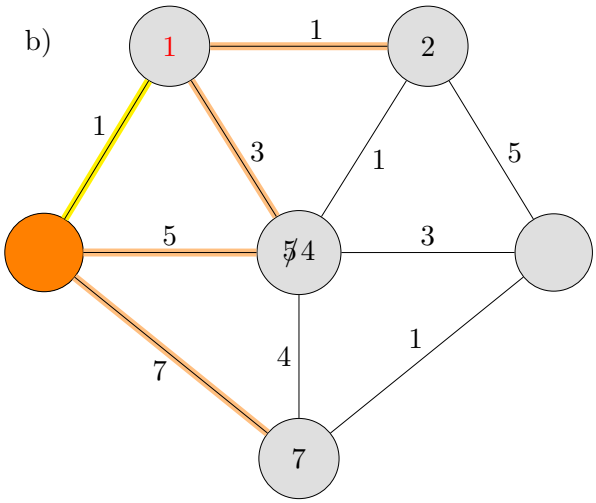
Dijkstra-Algorithmus

a)



Das Feuer läuft entlang der Zündschnüre, die vom Startknoten ausgehen.
Der Knoten mit der kleinsten Entfernung 1 wird zuerst erreicht.

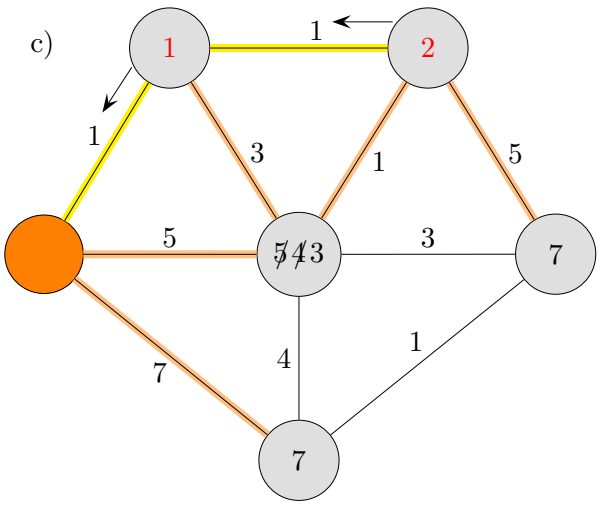
b)



Das Feuer läuft weiter entlang der Zündschnüre, die vom nächstliegenden Knoten ausgehen.
Die gelbe Zündschnur ist abgebrannt.
Der mittlere Knoten mit der Entfernung 5 kann nun nach 4 Sekunden erreicht werden.

Erneut ist der Knoten zu ermitteln, der als Nächster erreicht wird.

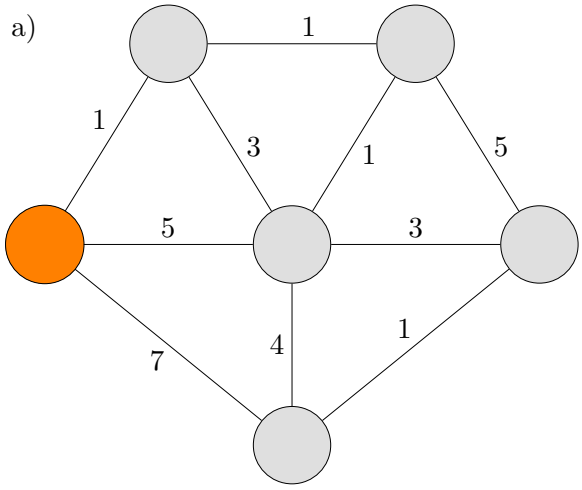
c)



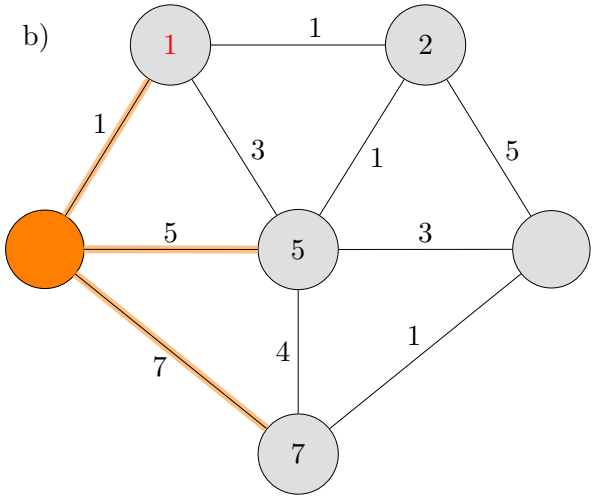
Mit diesem Verfahren ergeben sich die Knoten in aufsteigender, minimaler Entfernung zum Startknoten. Nachdem ein minimaler Knoten ermittelt wurde, sind die Entfernungsangaben der Randknoten, zu denen brennende Zündschnüre führen, zu aktualisieren.
Knoten können wiederholt aus verschiedenen Richtungen erreicht werden. Das ist aber ohne Belang.
Für das Aufsuchen der kürzesten Wege sind für die minimalen Knoten deren Vorgänger zu notieren. Sie können in der Grafik durch einen Pfeil kenntlich gemacht werden.

Dijkstras-Algorithmus

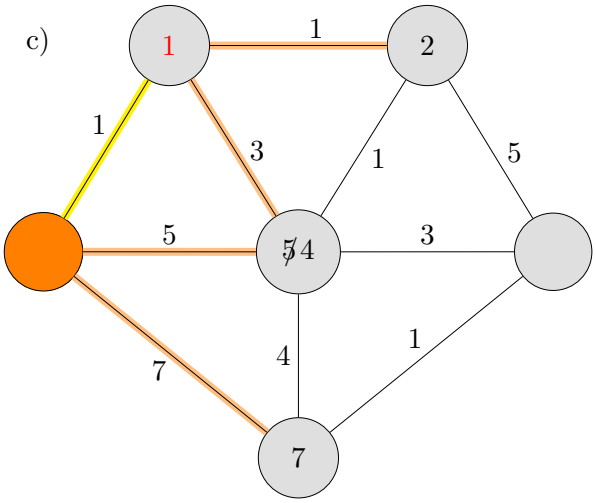
a)



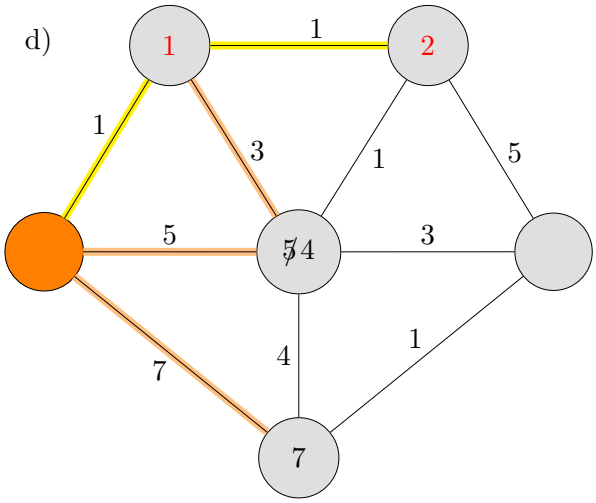
b)



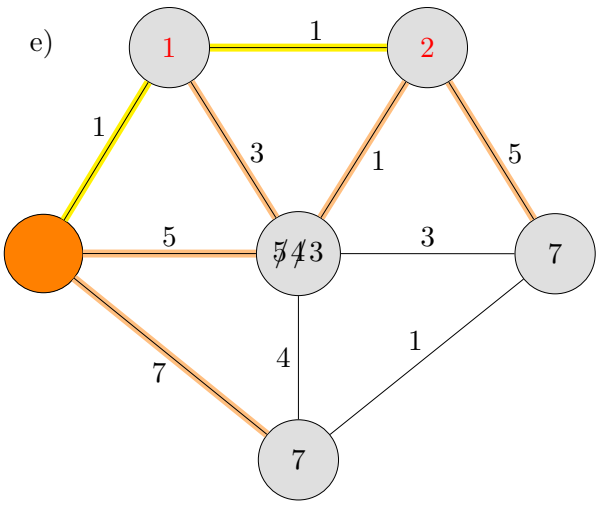
c)



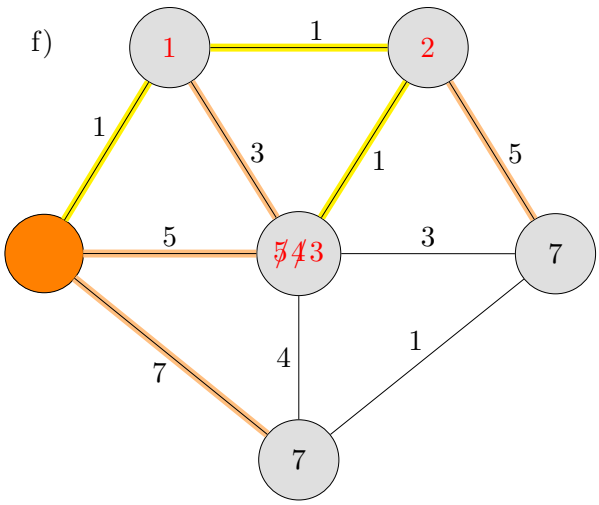
d)

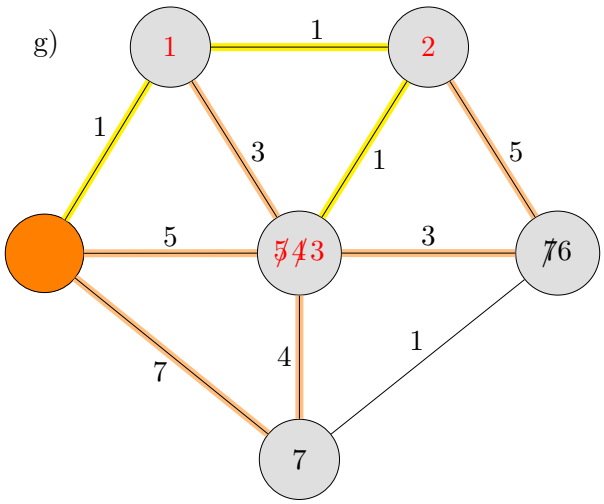


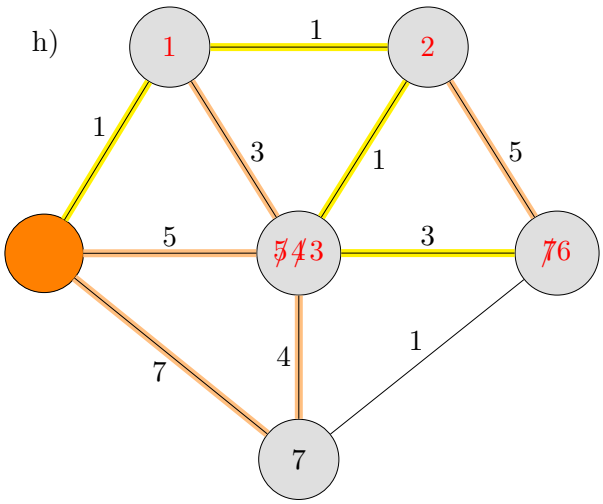
e)



f)







Es wurden nicht alle abgebrannten Zündschnüre gelb gefärbt,
da dies nicht wichtig ist.

Zum Algorithmus:

Sei K_1, K_2, K_3, …, K_{n-1}, K_n der kürzeste Pfad von K_1 zu K_n.
Dann ist für je 2 Zwischenknoten, z. B. K_2 und K_{n-1}, der kürzeste Pfad K_2, K_3, …, K_{n-1}.
Gäbe es einen Kürzeren von K_2 nach K_{n-1}, so könnte dieser in K_1, K_2, K_3, …, K_{n-1}, K_n
eingesetzt werden und wir erhielten einen kürzesten Pfad von K_1 zu K_n.
Zwischen K_1, K_2 ist die Entfernung daher minimal. Dies begründet den Anfang des Verfahrens.

Der Algorithmus besteht aus n Schritten.
In jedem Schritt wird ein Zwischenknoten ermittelt.

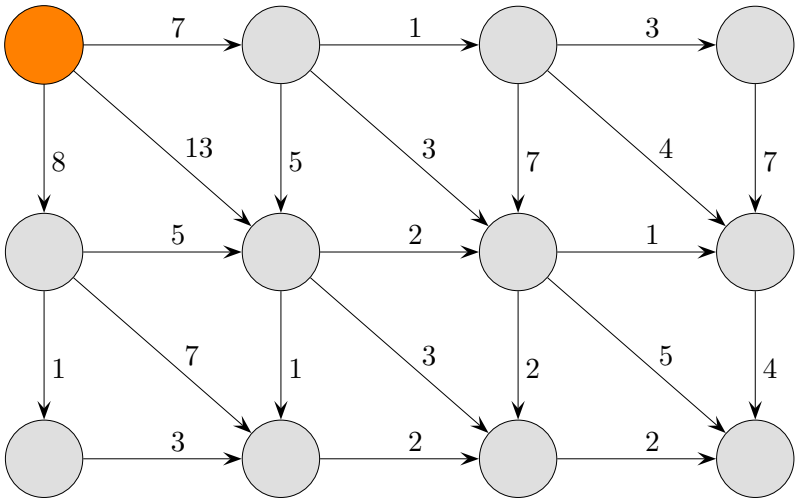
Wir beginnen mit dem Startknoten K_1,
legen die Entfernung $L(Q) = d(K_1, Q)$ (Kantenbelegung) der Randknoten Q fest
und ermitteln den minimalen Randknoten K_2.
*
Mit der Entfernung $L[K_2]$ von K_2 werden die Entfernungen der übrigen Randknoten Q aktualisiert:

$$L[Q] = \mathit{Minimum}\ \{\, L[Q],\ L[K_2] + d[K_2, Q] \,\}.$$

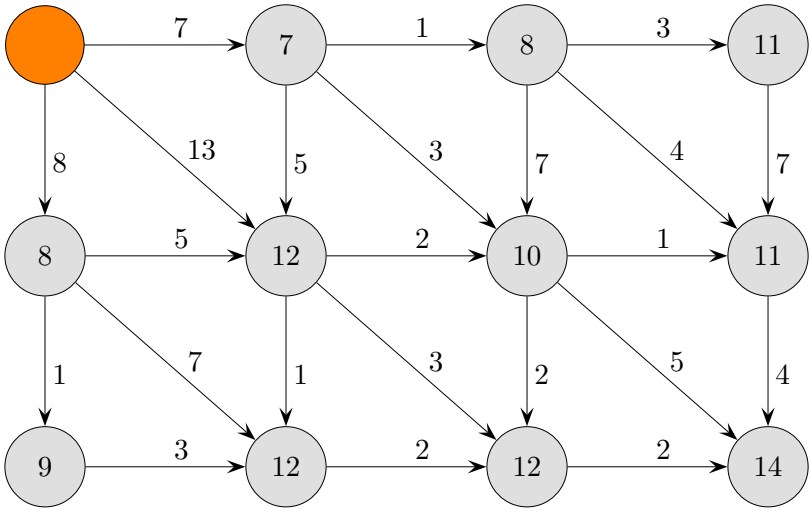
Den Endknoten P der von K_2 ausgehenden Kanten, die noch ohne Entfernungsangabe L sind,
werden durch Addition der Kantenbelegung eine Entfernung zugeordnet, $L[P] = L[K_2] + d[K_2, P]$.
Diese Endknoten P werden der Menge der Randpunkte hinzugefügt, K_2 wird der Menge entnommen.
Der minimale Randknoten K_3 wird ermittelt.

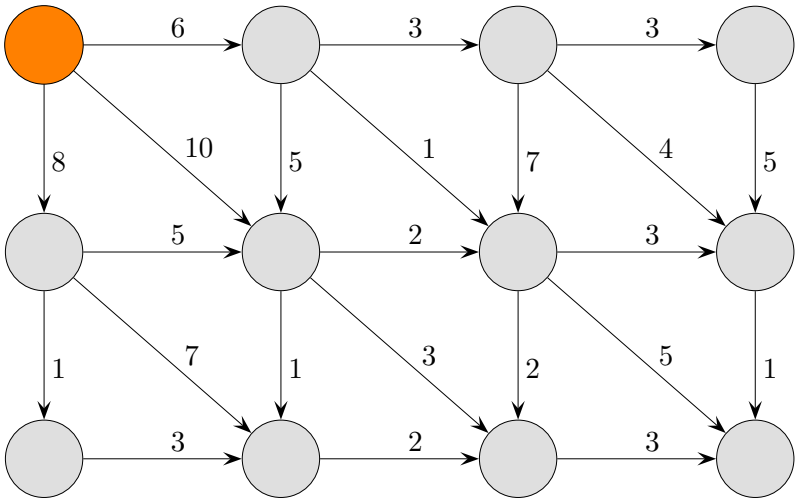
Wenn K_3 nicht schon K_n ist, wird das Verfahren bei $*$ mit K_3 statt K_2 fortgesetzt, usw.

In der Literatur wird häufig $L[K_1] = 0$ und $L[K] = \infty$ für Knoten K gesetzt, denen noch keine (vorläufige) Entfernung zugeordnet wurde. Die zu bearbeitenden Randknoten werden in einer Vorrangwarteschlange *(engl. priority queue, heap)* gespeichert. Diese Datenstruktur ermöglicht das Einfügen von Elementen, das Entfernen des Elements mit kleinstem Schlüssel (daher mit höchster Priorität) und das Verändern von Schlüsselwerten (hier Entfernungen).

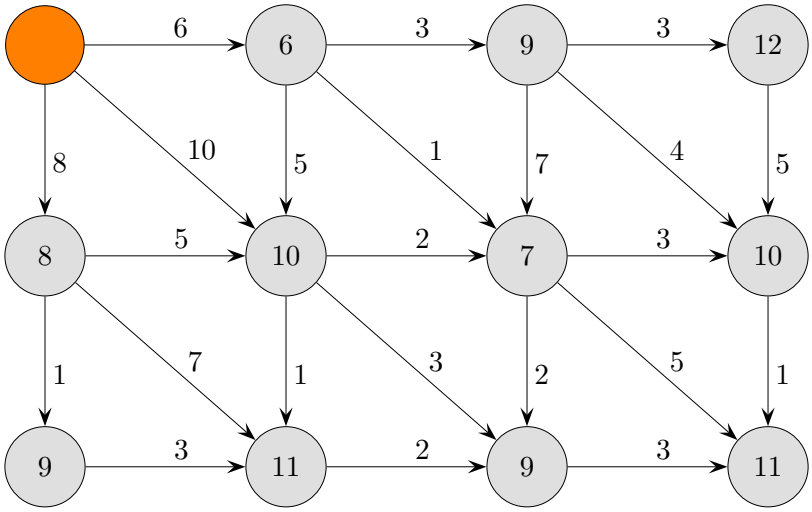


Ermittle zu jedem Knoten die kürzeste Entfernung zum Startknoten.


7
1
3
8
13
5
3
7
4
7
8
5
12
2
10
1
11
1
7
1
3
2
5
4
9
3
12
2
12
2
14



Ermittle zu jedem Knoten die kürzeste Entfernung zum Startknoten.


6
6
3
9
3
12
8
10
5
1
7
4
5
8
5
10
2
7
3
10
1
7
1
3
2
5
1
9
3
11
2
9
3
11