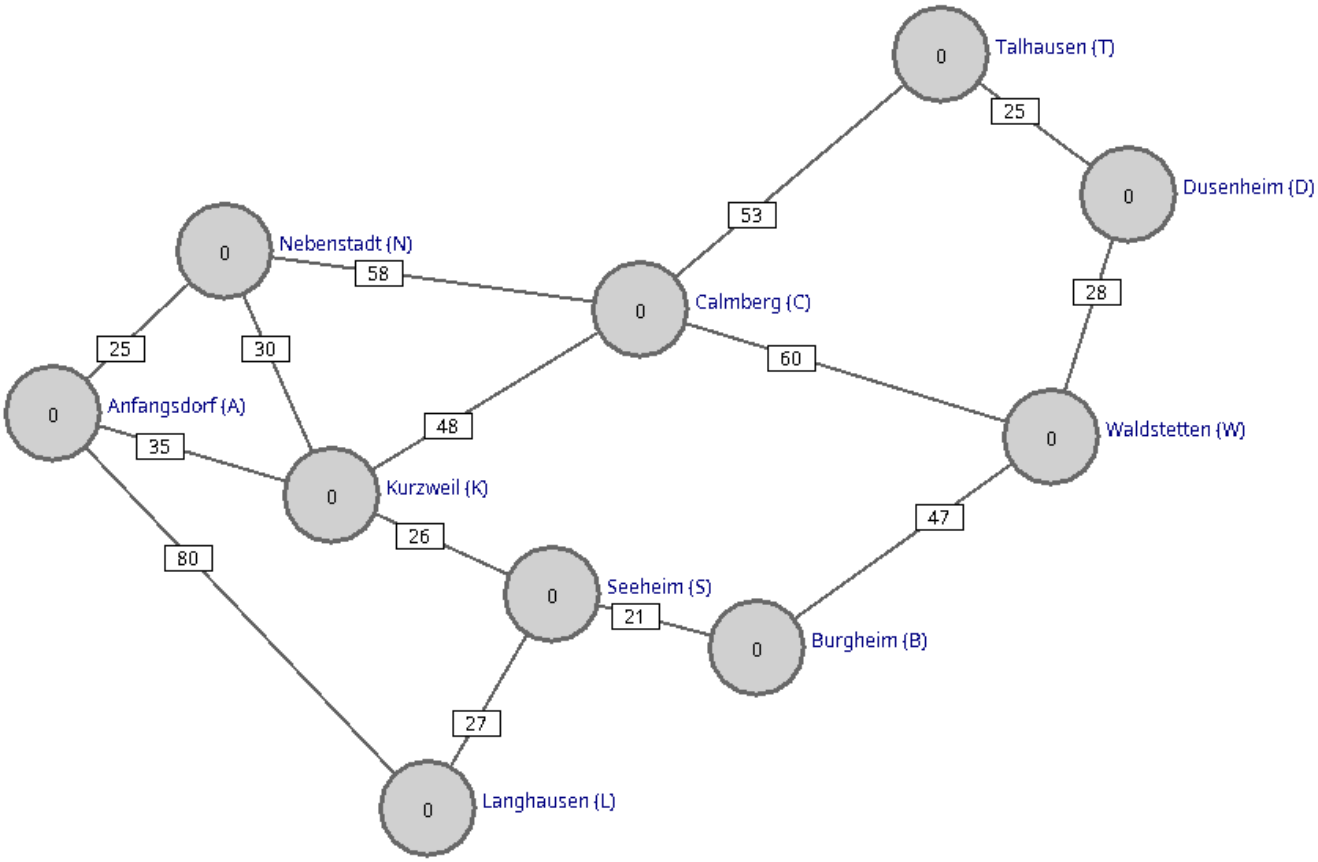
Der Dijkstra Algorithmus

Häufig möchte man kürzeste Pfade in gewichteten Graphen bestimmen. Die Länge des Pfads entspricht in diesem Fall nicht einfch der Zahl der Kanten, sondern der Summe der Kantengewichte entlang des Pfads.

Ein wichtiges Beispiel für diese Situation sind **Navigationssysteme**. Die Kantengewichte hängen von der genauen Problemstellung ab: Wenn man die kürzeste Strecke zwischen zwei Knoten im Graphen sucht, trägt man an den Kanten den Abstand zwischen benachbarten Knoten ein, sucht man den schnellsten Weg, repräsentieren die Kantengewichte die Fahrzeiten.

In diesem Sinne können die Zahlen an den Kanten im folgenden Beispielgraphen Entfernungen oder Zeiten repräsentieren.



Die Länge des direkten Pfads von Anfangsdorf nach Langhausen ist 80, die Länge des Pfads von Anfangsdorf nach Langhausen über Kurzweil und Seeheim ist 88. der direkte Pfad ist also kürzer.

Die Idee des Dijkstra Algorithmus

Gegeben ist:

- Ein gewichteter Graph G(V,E).[1)]

- Ein Startknoten S
- Ein Zielknoten Z

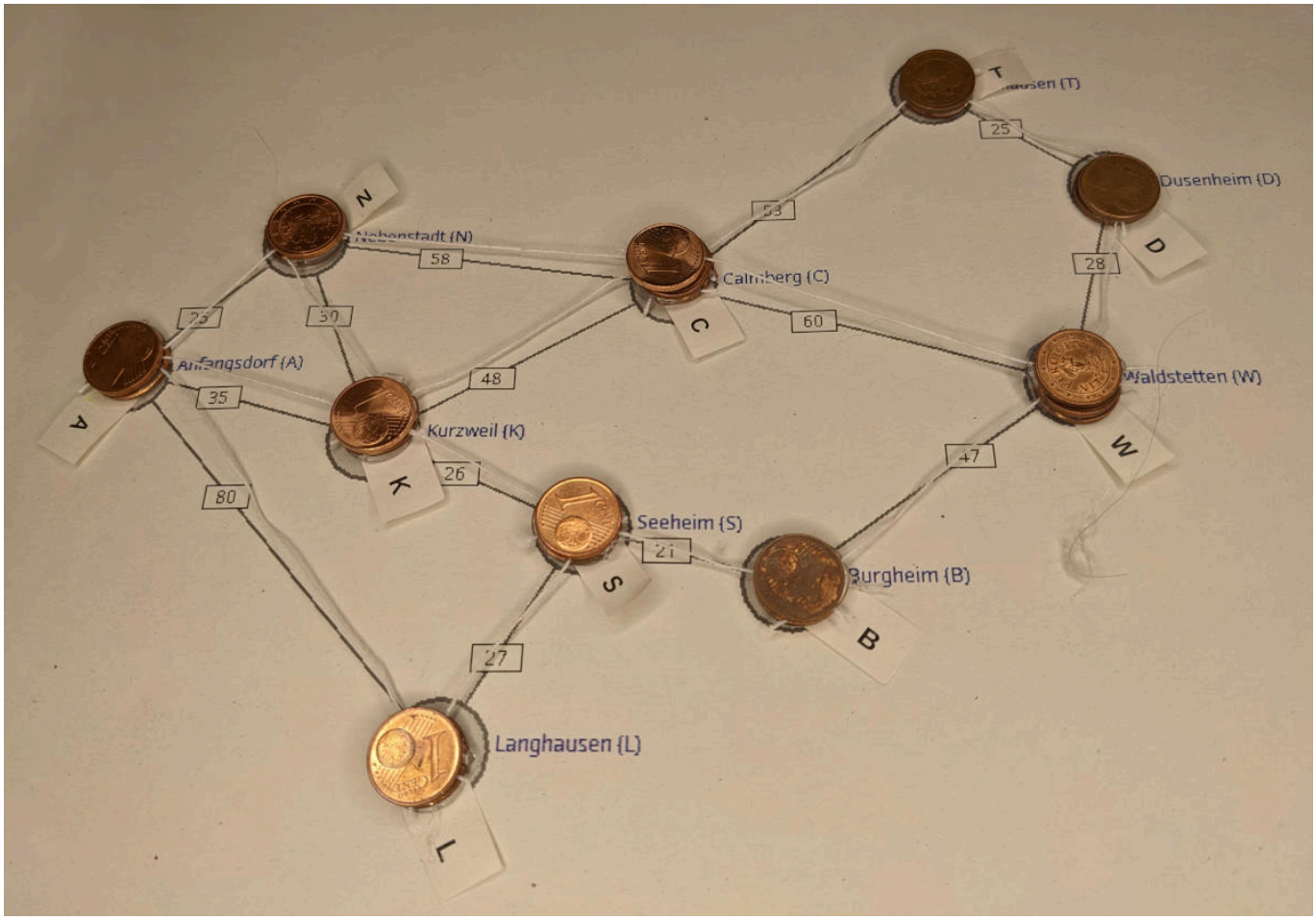
Gesucht: Der kürzeste Weg durch den Graphen, der von S nach Z führt.

Eine andere Formulierung des Problems ist, dass zu einem gegebenen Graphen mit Startknoten S die Pfadlänge von S zu allen anderen Knoten im Graphen gefunden werden soll.

Lösungsansatz mit Pfiff

Der folgende Lösungsansatz führt direkt zum 1959 von Edsger Wybe Dijkstra gefundenen Algorithmus zur Lösung dieses Problems.

Dazu stellt man sich zunächst vor, die Knoten seien durch Fäden miteinander verbunden.



Nun fasst man das so entstandene Netz am Startpunkt und hebt diesen an und zieht ihn langsam nach oben. Auf diese Weise heben sich nacheinander alle Knoten des Graphen von der Unterlage ab, und zwar in der Reihenfolge der Entfernung vom Startknoten: Der Knoten mit dem geringsten Abstand zuerst, dann der zweitnächste und so weiter.

Um den kürzesten Weg zu einem gegebenen Zielknoten Z zu finden, geht man also folgendermaßen vor: Man hebt den Startknoten an und zieht ihn nach oben. Knoten für Knoten löst sich von der

Tischplatte. Man stoppt, wenn sich der Zielknoten beginnt, sich abzuheben. Wenn man nun die straff gespannten Fäden zurückverfolgt, führen diese eindeutig zum Startpunkt - und dieser Weg von S zu Z ist der kürzeste Weg zwischen den Knoten.

(A1)

- Schaue dir den Film an, der den Vorgang zeigt und stelle anhand der Bewegung der Centstücke die Reihenfolge der Knotenentfernungen im Beispielgraphen auf. Überprüfe die so gefundene Reihenfolge anhand der Kantengewichte.

- Erläutere kurz, warum man so tatsächlich den kürzesten Weg zwischen Start- und Zielknoten findet.

Der Computer zieht die Fäden

Möchte man dieses Trickreiche Vorgehen in einem Algorithmus implementieren, muss man anstelle der Fäden andere Kriterien finden. Der Schlüssel zum Ersatz der Fäden ist die Antwort auf folgenden Frage:

Welcher Knoten ist stets der nächste, der von der Unterlage abgehoben wird?

Antwort

Es wird immer derjenige Knoten in der Liste der unbearbeiteten Knoten als nächstes von der Unterlage abgehoben, dessen Entfernung vom Startknoten die kleinste ist.

Wir benötigen also **zwei Markierungen**: Knoten die in der Liste sind und als nächstes bearbeitet werden können, weil sie Nachbarknoten von Knoten sind, die bereits von der Unterlage abgehoben wurden und die Knoten die sich bereits in den Luft befinden. Im Graphentester kann man z.B, folgende Vereinbarung treffen:

- Knoten **Besucht**: In der ToDo-Liste, mit einem Entfernungswert versehen.
- Knoten **Markiert**: In der Luft, Wert entspricht der kürzesten Entfernung vom Startknoten

Damit sieht ein Lückenhafter Pseudocode so aus:

```
hole den startknoten, markiere in als besucht, füge ihn in die ToDo-liste
ein.

solange die ToDo-liste nicht leer ist:
  // Hier fehlt etwas
  hole das element _aktuell_ aus der ToDo-liste // welches? - siehe Zeile
```

```
oberhalb und Zeile unterhalb
  hebe _aktuell_ von der unterlage ab // was muss dazu programmiert werden?
  für alle nachbarn n von _aktuell_ wiederhole:
    berechne die entfernung d des knotens n vom startknoten aus über den
knoten _aktuell_
    wenn d < wert von n ODER n nicht in der ToDo-liste
      // was muss dann geschehen
    wenn n nicht in der ToDo-Liste:
      // Was muss dann geschehen?
```

(A2)

Vervollständige den Pseudocode.

Lösungsvorschlag

```
hole den startknoten, markiere in als besucht, füge ihn in die ToDo-liste
ein.

solange die ToDo-liste nicht leer ist:
  sortiere die ToDo-liste aufsteigend
  hole das kleinste element _aktuell_ aus der ToDo-liste
  setze aktuell auf markiert
  für alle nachbarn n von _aktuell_, die nicht markiert sind, wiederhole:
    berechne die entfernung d des knotens n vom startknoten aus über den
knoten _aktuell_ // wert von aktuell + abstand zum nachbarn
    wenn d < wert von n ODER n nicht in der ToDo-liste
      wert von n auf d setzen
    wenn n nicht in der ToDo-Liste:
      markiere n als besucht
      fuege n der ToDo-liste hinzu
```

(A3)

Implementiere den Dijkstra Algorithmus im Graphentester, so dass in jedem Knoten die Entfernung des kürzesten Wegs vom Startknoten zu allen anderen Knoten des Graphen steht.

(A4)

Erweitere deinen Algorithmus so, dass die kürzesten Pfade farblich hervorgehoben werden.[2)]

(A5)

Passe den Algorithmus so an, dass

1)
Die Gewichte müssen eine Randbedingung erfüllen, dazu später mehr
2)
kante.setFarbe()