9.5 Kürzeste Wege: der Algorithmus von Dijkstra

LK

Jetzt zählen die Gewichte. Die Aufgabe heißt **Kürzeste-Wege-Problem**: Gesucht ist zu einem Startknoten der Weg mit der kleinsten Summe der Kantengewichte zu jedem anderen Knoten.

Naheliegend wäre, immer die gerade kürzeste Kante zu nehmen. Das geht schief — die Abkürzung von heute kann morgen in eine Sackgasse führen. Edsger **Dijkstra** fand 1956 das Verfahren, das es richtig macht; er entwarf es nach eigener Erzählung in zwanzig Minuten in einem Amsterdamer Café, ohne Papier.

Definition: Algorithmus von Dijkstra

Jeder Knoten bekommt eine vorläufige Entfernung: der Start 0, alle anderen ∞. Dann wiederhole, bis alle Knoten abgehakt sind:

1. Wähle unter den noch nicht abgehakten Knoten den mit der **kleinsten** vorläufigen Entfernung. Hake ihn ab — seine Entfernung ist jetzt endgültig.
2. Prüfe für jeden seiner noch nicht abgehakten Nachbarn: Ist der Weg über den gerade abgehakten Knoten *kürzer* als der bisher notierte? Dann trage die kleinere Entfernung ein und merke dir den Vorgänger.

Der zweite Schritt heißt **Relaxation** — man „entspannt" eine zu groß geschätzte Entfernung auf einen besseren Wert.

Beispiel: Von Merzig aus, Runde für Runde

Gesucht sind die kürzesten Wege von **Merzig** zu allen anderen Orten. In jeder Zeile steht der Stand *nach* dem Abhaken; der abgehakte Knoten ist fett.

Runde	abgehakt	SB	NK	OTW	WND	HOM	MZG
Start	—	∞	∞	∞	∞	∞	0
1	**MZG** (0)	45	∞	∞	48	∞	—
2	**SB** (45)	—	69	∞	48	76	—
3	**WND** (48)	—	69	58	—	76	—
4	**OTW** (58)	—	**66**	—	—	76	—
5	**NK** (66)	—	—	—	—	76	—
6	**HOM** (76)	—	—	—	—	—	—

Zwei Stellen lohnen einen zweiten Blick.

Runde 4. Nach Runde 2 stand bei Neunkirchen die 69, über Saarbrücken: $45 + 24$. In Runde 4 wird Ottweiler abgehakt, und über Ottweiler kostet Neunkirchen nur $58 + 8 = 66$. Der alte Wert wird überschrieben. Genau dafür ist Schritt 2 da — und genau deshalb darf man einen Knoten erst dann für endgültig erklären, wenn er der kleinste aller offenen ist.

Runde 3. Homburg steht auf 76, über Saarbrücken: $45 + 31$. Über St. Wendel wären es $48 + 28 = 76$ — exakt gleich viel. Gleich ist nicht kleiner, also bleibt der Eintrag stehen und Homburg behält Saarbrücken als Vorgänger. Es gibt hier schlicht zwei kürzeste Wege, und der Algorithmus notiert denjenigen, den er zuerst gefunden hat.

Die Entfernungen stehen damit fest, die Wege noch nicht — die stecken in den gemerkten Vorgängern. Man liest sie rückwärts ab: Neunkirchens Vorgänger ist Ottweiler, dessen Vorgänger St. Wendel, dessen Vorgänger Merzig. Umgedreht ergibt das den Weg.

Ziel	Entfernung	Weg
Saarbrücken	45 km	MZG → SB
St. Wendel	48 km	MZG → WND
Ottweiler	58 km	MZG → WND → OTW
Neunkirchen	66 km	MZG → WND → OTW → NK
Homburg	76 km	MZG → SB → HOM

In Python wird daraus ein knappes Programm. Die Funktion liefert zwei Wörterbücher: die Entfernungen und die Vorgänger, aus denen sich die Wege zusammensetzen lassen.

```python
def dijkstra(graph, start):
    entfernung = {knoten: float("inf") for knoten in graph}
    vorgaenger = {}
    entfernung[start] = 0
    offen = set(graph)
    while offen:
        knoten = min(offen, key=lambda k: entfernung[k])
        offen.remove(knoten)
        for nachbar, laenge in graph[knoten].items():
            neu = entfernung[knoten] + laenge
            if neu < entfernung[nachbar]:
                entfernung[nachbar] = neu
                vorgaenger[nachbar] = knoten
    return entfernung, vorgaenger


def weg(vorgaenger, start, ziel):
    strecke = [ziel]
    while strecke[-1] != start:
        strecke.append(vorgaenger[strecke[-1]])
    strecke.reverse()
    return strecke
```

```
>>> entfernung, vorgaenger = dijkstra(GRAPH, "MZG")
>>> entfernung["NK"]
66
>>> weg(vorgaenger, "MZG", "NK")
['MZG', 'WND', 'OTW', 'NK']
```

Und mit der Matrix in Pascal:

```pascal
procedure Dijkstra(start: Integer);
var
  entfernung, vorgaenger: array [1..N] of Integer;
  fertig: array [1..N] of Boolean;
  i, n, kleinste, naechster: Integer;
begin
  for n := 1 to N do
  begin
    entfernung[n] := UNENDLICH;
```

```
10      vorgaenger[n] := 0;
11      fertig[n] := False;
12    end;
13    entfernung[start] := 0;
14
15    for i := 1 to N do
16    begin
17      kleinste := UNENDLICH;
18      naechster := 0;
19      for n := 1 to N do
20        if (not fertig[n]) and (entfernung[n] < kleinste) then
21        begin
22          kleinste := entfernung[n];
23          naechster := n;
24        end;
25      if naechster = 0 then
26        Break;                                    { Rest nicht erreichbar }
27      fertig[naechster] := True;
28      for n := 1 to N do
29        if (ENTFERNUNG[naechster, n] > 0) and not fertig[n] then
30          if entfernung[naechster] + ENTFERNUNG[naechster, n] < entfernung[n] then
31          begin
32            entfernung[n] := entfernung[naechster] + ENTFERNUNG[naechster, n];
33            vorgaenger[n] := naechster;
34          end;
35    end;
36
37    for n := 1 to N do
38      WriteLn(ORTE[start], ' -> ', ORTE[n], ': ', entfernung[n], ' km');
39  end;
```

Der Aufwand: In jeder der n Runden wird einmal das Minimum unter den offenen Knoten gesucht und einmal über die Nachbarn gelaufen. In dieser einfachen Fassung sind das $\mathcal{O}(n^2)$ Schritte — für ein Navigationsgerät mit Millionen Kreuzungen zu viel, weshalb dort eine geschicktere Datenstruktur für den „kleinsten offenen Knoten" zum Einsatz kommt.

Stolperstein

Dijkstra setzt voraus, dass **alle Gewichte nicht negativ** sind. Mit negativen Kanten bricht die ganze Begründung zusammen: Man darf einen Knoten dann nicht mehr für endgültig erklären, nur weil er gerade der kleinste ist — eine spätere negative Kante könnte ihn noch billiger machen. Für solche Fälle gibt es andere Verfahren. Bei Entfernungen und Fahrzeiten stellt sich die Frage nicht, bei Gewinnen und Verlusten schon.

Arbeitsauftrag 5

24. Führen Sie Dijkstra von Saarbrücken aus durch und legen Sie die Tabelle wie im Beispiel an. Welcher Ort ist am weitesten entfernt?
25. Führen Sie Dijkstra von Ottweiler aus durch — also von der Schule. Wie weit ist es nach Merzig, über welche Orte führt der Weg, und warum taucht Saarbrücken darin nicht auf, obwohl es die Kante SB – MZG gibt?
26. Lassen Sie die Breitensuche ab Merzig laufen und lesen Sie aus dem Suchbaum den Weg nach Neunkirchen ab. Vergleichen Sie ihn mit dem Dijkstra-Weg aus dem Beispiel: Welcher hat weniger Stationen, welcher ist kürzer? Erklären Sie, warum beides zusammenpasst.
27. Begründen Sie, warum der Knoten mit der kleinsten vorläufigen Entfernung endgültig sein *muss*, solange alle Gewichte nicht negativ sind.
28. Geben Sie einen kleinen Graphen mit einer negativen Kante an, bei dem Dijkstra ein falsches Ergebnis liefert. Rechnen Sie es vor.
29. Erweitern Sie das Python-Programm so, dass es die Wege zu allen Knoten ausgibt.
30. Ersetzen Sie im Saarland-Graphen die Entfernungen durch geschätzte Fahrzeiten — die Autobahnstrecken SB – MZG und SB – WND fahren sich deutlich schneller als die Landstraßen. Rechnen Sie erneut. Ändert sich der kürzeste Weg von Merzig nach Neunkirchen?

Gymnasium Ottweiler