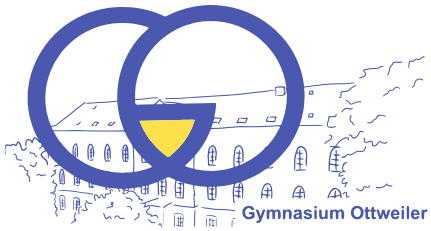
Kapitel 9

Graphentheorie

Skript Informatik · Grundkurs



Ein Navigationsgerät sucht den kürzesten Weg. Ein Netzwerkplaner sucht die billigste Verkabelung. Eine Suchmaschine ordnet Seiten nach ihren Verweisen, ein soziales Netz schlägt Bekannte vor, ein Übersetzer zerlegt Sätze in Bäume. So verschieden diese Aufgaben klingen, sie haben alle dieselbe Form: *Dinge* und *Verbindungen zwischen ihnen*.

Für diese Form gibt es ein eigenes Modell — den **Graphen**. Wer ein Problem erst einmal als Graph aufgeschrieben hat, kann auf fertige Algorithmen zurückgreifen, ohne sich noch um Straßen, Kabel oder Freundschaften zu kümmern. Genau darin liegt die Stärke: Ein einziger Algorithmus löst den kürzesten Weg für alle diese Fälle auf einmal.

> LK
>
> Das ganze Kapitel gehört zum Themenfeld *Graphentheorie* und steht damit nur im Lehrplan des Leistungskurses (10 % der Unterrichtszeit). Für den Grundkurs ist nichts davon verbindlich. Die Abschnitte 9.1 bis 9.3 lohnen sich trotzdem, weil sie zeigen, was Modellbildung in der Informatik eigentlich heißt.

Lernziele

Am Ende dieses Kapitels können Sie

- einen Graphen als Paar aus Knoten- und Kantenmenge definieren,
- den Zusammenhang zwischen gerichteten, ungerichteten und gewichteten Graphen erklären,
- die Begriffe Knotengrad, schlicht, vollständig und zusammenhängend sicher verwenden,
- Bäume als Spezialfall von Graphen einordnen,
- Alltagssituationen und informatische Aufgaben als Graphen modellieren,
- einen Graphen mengentheoretisch, als Adjazenzmatrix und als Adjazenzliste darstellen und begründen, welche Darstellung wann die bessere ist,
- Breiten- und Tiefensuche auf einem gegebenen Graphen ausführen und ihre Unterschiede erläutern,
- das Kürzeste-Wege-Problem benennen und mit dem Algorithmus von Dijkstra lösen,
- einen minimalen Spannbaum mit dem Algorithmus von Kruskal konstruieren,
- Hamiltonkreise finden, das Problem des Handlungsreisenden benennen und begründen, warum es praktisch schwer ist.

9.1 Die sieben Brücken von Königsberg

LK

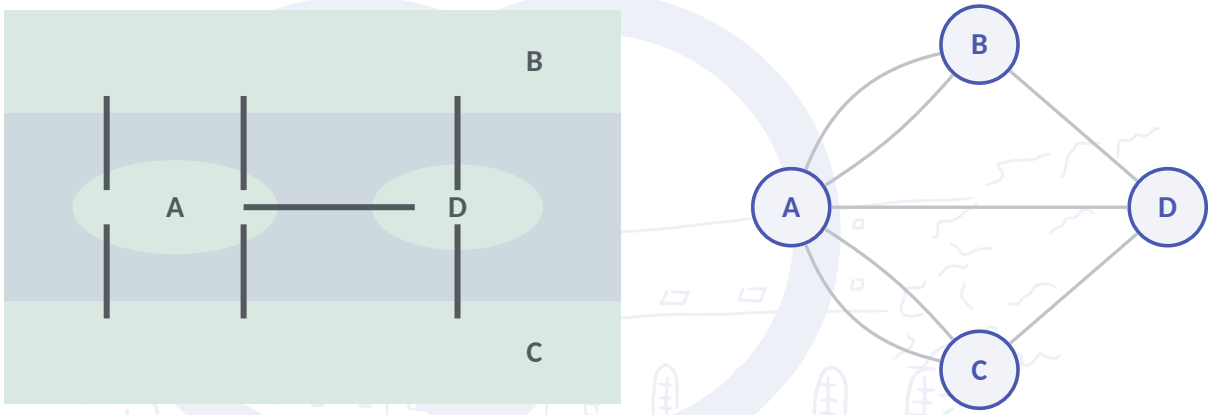
In Königsberg — dem heutigen Kaliningrad — teilte sich der Fluss Pregel um zwei Inseln. Sieben Brücken verbanden die Inseln mit den beiden Ufern. Im 18. Jahrhundert ging in der Stadt eine Frage um, über die man beim Sonntagsspaziergang nachdachte:

Leitidee: Die Frage der Königsberger

Gibt es einen Rundgang durch die Stadt, bei dem man **jede der sieben Brücken genau einmal** überquert?

Niemand fand einen solchen Weg — aber niemand konnte auch beweisen, dass es keinen gibt. 1736 beantwortete der Mathematiker **Leonhard Euler** die Frage in einer Arbeit mit dem Titel *Solutio problematis ad geometriam situs pertinentis*. Seine entscheidende Idee war nicht eine clevere Suche, sondern ein **Weglassen**: Für die Frage ist es völlig gleichgültig, wie breit der Fluss ist, wie lang die Brücken sind oder wo genau sie stehen. Wichtig ist allein, *welche* Brücke *welche* Landstücke verbindet.

Also schrumpfte Euler jedes Landstück auf einen Punkt und jede Brücke auf eine Linie zwischen zwei Punkten:



Pregel, Inseln und die sieben Brücken — vier Knoten, sieben Kanten

Links der Lageplan, rechts dasselbe als Graph. Die vier Landstücke werden zu vier Knoten, die sieben Brücken zu sieben Kanten.

Übrig bleibt ein Gebilde aus vier Punkten und sieben Linien. Mehr braucht man nicht — und genau dieses Gebilde heißt heute **Graph**. Die Antwort auf die Königsberger Frage holen wir in Abschnitt 9.2.2 nach; dafür fehlt uns noch ein einziger Begriff.

Merke

Modellieren heißt weglassen. Ein Graph behält von einer Situation nur zwei Dinge: *welche Dinge es gibt* und *welche davon miteinander verbunden sind*. Alles andere — Längen, Richtungen, Farben, Namen — kommt nur dazu, wenn die Frage es verlangt.

Arbeitsauftrag 1

1. Nennen Sie drei Situationen aus Ihrem Alltag, die sich als Graph beschreiben lassen. Geben Sie jeweils an, was die Knoten und was die Kanten sind.
2. Zeichnen Sie den Graphen zu einer U-Bahn-Linie mit sechs Stationen. Was sind hier die Knoten, was die Kanten?
3. Erklären Sie, warum die Zeichnung eines Graphen nicht eindeutig ist: Zeichnen Sie den Königsberger Graphen ein zweites Mal mit anders angeordneten Knoten.

Gymnasium Ottweiler

9.2 Was ein Graph ist

LK

Formal ist ein Graph nichts weiter als zwei Mengen: die Dinge und die Verbindungen.

Definition: Gerichteter Graph

Ein **gerichteter Graph** ist ein Paar $G = (V, E)$ aus

- einer endlichen Menge V von **Knoten** (englisch *vertices*) und
- einer Menge $E \subseteq V \times V$ von **Kanten** (englisch *edges*).

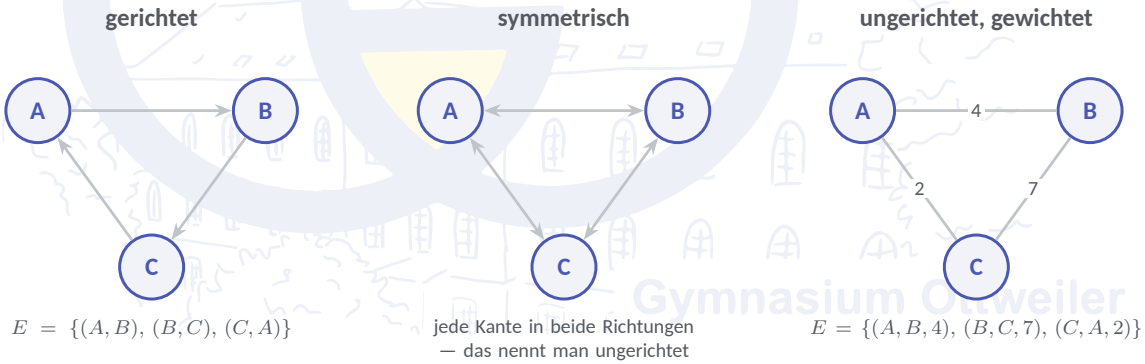
Eine Kante ist also ein *geordnetes Paar* (a, b) und zeigt von a nach b. Die Kantenmenge ist damit eine **Relation** auf der Knotenmenge.

Dass E eine Relation ist, ist keine Spitzfindigkeit, sondern das wichtigste Werkzeug dieses Kapitels: Fast jede Eigenschaft eines Graphen lässt sich an dieser Relation ablesen.

9.2.1 Gerichtet, ungerichtet, gewichtet

Bei manchen Beziehungen kommt es auf die Richtung an: Eine Webseite verlinkt eine andere, ohne dass zurückverlinkt wird. Bei anderen nicht: Wenn eine Straße Ottweiler mit Neunkirchen verbindet, dann verbindet sie auch Neunkirchen mit Ottweiler.

Man braucht dafür aber keine zweite Definition. Ein **ungerichteter** Graph ist einfach ein gerichteter, dessen Relation **symmetrisch** ist: Zu jeder Kante (a, b) gehört auch die Kante (b, a). Gezeichnet spart man sich dann die Pfeilspitzen und schreibt die Kante als Menge $\{a, b\}$.



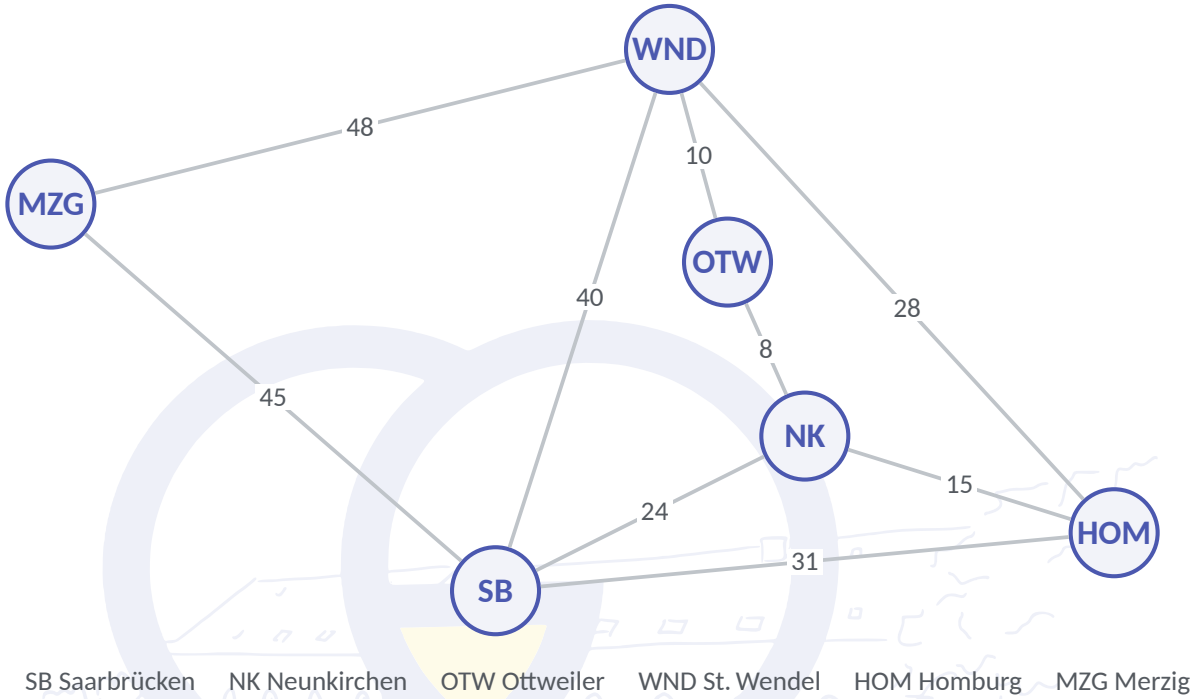
Drei Sichtweisen auf dieselbe Sache.

Kommt zu jeder Kante noch eine Zahl hinzu — eine Entfernung, eine Fahrzeit, eine Bandbreite, ein Preis —, heißt der Graph **gewichtet**. Aus den Paaren der Kantenmenge werden dann Tripel (a, b, w) mit dem **Gewicht** $w \in \mathbb{R}$.

Merke

Ungerichtet und gewichtet sind keine anderen Dinge, sondern Zusätze: *ungerichtet* heißt „die Kantenrelation ist symmetrisch", *gewichtet* heißt „jede Kante trägt zusätzlich eine Zahl". Jeder ungerichtete Graph lässt sich als gerichteter schreiben, indem man jede Kante durch zwei gegenläufige ersetzt.

Für den Rest des Kapitels arbeiten wir mit einem Beispiel aus der Nachbarschaft: sechs Orte im Saarland und die Straßen zwischen ihnen, mit den Entfernungen in Kilometern.



Der Saarland-Graph. Entfernungen in Kilometern, auf ganze Zahlen gerundet.

Mengentheoretisch aufgeschrieben ist das

$$V = \{\mathsf{SB}, \mathsf{NK}, \mathsf{OTW}, \mathsf{WND}, \mathsf{HOM}, \mathsf{MZG}\}$$

und, weil der Graph ungerichtet und gewichtet ist,

$$E = \{(\mathsf{SB}, \mathsf{NK}, 24),\ (\mathsf{SB}, \mathsf{HOM}, 31),\ (\mathsf{SB}, \mathsf{WND}, 40),\ (\mathsf{SB}, \mathsf{MZG}, 45),\ (\mathsf{NK}, \mathsf{OTW}, 8),$$
$$(\mathsf{NK}, \mathsf{HOM}, 15),\ (\mathsf{OTW}, \mathsf{WND}, 10),\ (\mathsf{HOM}, \mathsf{WND}, 28),\ (\mathsf{WND}, \mathsf{MZG}, 48)\}.$$

Neun Kanten für sechs Knoten.

Stolperstein

Zwischen Neunkirchen und St. Wendel steht keine Kante — obwohl man natürlich von einem Ort zum anderen fahren kann. Der Grund: Die Straße dorthin ist die B 41, und die führt **durch Ottweiler**. Eine Kante bedeutet „direkte Verbindung, ohne einen anderen Knoten des Modells zu berühren"; die 18 km von Neunkirchen nach St. Wendel stecken als $8 + 10$ bereits im Graphen. Nimmt man solche Durchfahrten zusätzlich als Kanten auf, zählt man Strecken doppelt und der Graph beschreibt ein Straßennetz, das es nicht gibt. Welche Verbindungen eine Kante verdienen, ist die erste und wichtigste Entscheidung beim Modellieren.

9.2.2 Grundbegriffe

Definition: Knotengrad

Der **Grad** $\deg(v)$ eines Knotens v in einem ungerichteten Graphen ist die Anzahl der Kanten, die an ihm hängen.

In einem gerichteten Graphen unterscheidet man den **Eingangsgrad** (wie viele Kanten enden hier) vom **Ausgangsgrad** (wie viele beginnen hier).

Im Saarland-Graphen ist $\deg(\text{SB}) = 4$ und $\deg(\text{MZG}) = 2$. Zählt man alle Grade zusammen, kommt $4 + 3 + 2 + 4 + 3 + 2 = 18$ heraus — genau das Doppelte der neun Kanten. Das ist kein Zufall: Jede Kante hat zwei Enden und wird deshalb bei genau zwei Knoten mitgezählt.

Merke

Handschlaglemma. In jedem ungerichteten Graphen gilt

$$\sum_{v \in V} \deg(v) = 2 \cdot |E|.$$

Die Summe aller Knotengrade ist also immer gerade. Daraus folgt unmittelbar: Die Anzahl der Knoten mit *ungeradem* Grad ist in jedem Graphen gerade.

Drei weitere Eigenschaften braucht man ständig:

- Ein Graph heißt **schlicht**, wenn er keine Schlingen (Kanten von einem Knoten zu sich selbst) und keine Mehrfachkanten enthält. Der Saarland-Graph ist schlicht, der Königsberger Graph nicht — dort gibt es ja Doppelbrücken.
- Ein Graph heißt **vollständig**, wenn jeder Knoten mit jedem anderen direkt verbunden ist. Ein vollständiger schlichter Graph mit n Knoten hat $\frac{n(n-1)}{2}$ Kanten; bei sechs Knoten wären das 15. Unser Graph hat nur 9 Kanten, ist also nicht vollständig.
- Ein Graph heißt **zusammenhängend**, wenn es zwischen je zwei Knoten einen Weg gibt. Der Saarland-Graph ist zusammenhängend — man kommt von überall überallhin. Nähme man die beiden Kanten an Merzig weg, zerfiele er in zwei Teile.

Beispiel: Zurück nach Königsberg

Jetzt lässt sich Eulers Antwort in zwei Sätzen geben. Stellen Sie sich vor, Sie laufen den gesuchten Rundgang. Jedes Mal, wenn Sie ein Landstück betreten, müssen Sie es über eine *andere* Brücke wieder verlassen — Brücken verbrauchen sich ja. An jedem Landstück, das weder Start noch Ziel ist, werden die Brücken also **paarweise** benutzt, der Knotengrad muss dort gerade sein.

Im Königsberger Graphen haben *alle vier* Knoten ungeraden Grad: $\deg(A) = 5$, $\deg(B) = \deg(C) = \deg(D) = 3$. Start und Ziel könnten zwei davon entschuldigen, die anderen beiden nicht. **Es gibt keinen solchen Rundgang** — und bei einem geschlossenen Rundgang, der wieder am Ausgangspunkt endet, dürfte es sogar gar keinen ungeraden Knoten geben.

Ein Weg, der jede Kante genau einmal benutzt, heißt heute **Eulerweg**; ist er geschlossen, heißt er **Eulerkreis**. Eulers Ergebnis ist bemerkenswert, weil es die Frage nicht durch Probieren beantwortet, sondern durch eine Eigenschaft, die man in zehn Sekunden ablesen kann.

9.2.3 Bäume sind auch nur Graphen

Die Bäume aus Kapitel 3 lassen sich jetzt sauber einordnen: Ein **Baum** ist ein zusammenhängender Graph ohne Kreise. Richtet man alle Kanten von der Wurzel weg, erhält man einen gerichteten, **azyklischen** Graphen, bei dem jeder Knoten außer der Wurzel genau einen Eingangsgrad 1 hat.

Merke

Ein Baum mit n Knoten hat immer genau $n - 1$ Kanten. Eine Kante weniger, und er zerfällt; eine Kante mehr, und er enthält einen Kreis.

Damit bekommt auch der Suchbaum aus Abschnitt 3.4 eine zweite Lesart: Er ist ein gerichteter azyklischer Graph, dessen Kanten zusätzlich sortiert sind.

Arbeitsauftrag 2

4. Geben Sie zum Saarland-Graphen V und E ohne Gewichte an, also als Menge von zweielementigen Mengen.
5. Bestimmen Sie alle sechs Knotengrade und prüfen Sie das Handschlaglemma nach.
6. Wie viele Kanten fehlen dem Saarland-Graphen zur Vollständigkeit? Welche sind das?
7. Begründen Sie, warum es keinen Graphen mit genau drei Knoten vom Grad 1 und sonst nur Knoten geraden Grades geben kann.
8. Zeichnen Sie einen zusammenhängenden schlichten Graphen mit fünf Knoten, der einen Eulerkreis besitzt. Geben Sie den Kreis an.
9. Hat der Saarland-Graph einen Eulerweg? Prüfen Sie mit dem Gradkriterium und begründen Sie Ihre Antwort.
10. Ein gerichteter Graph modelliert die Links zwischen fünf Webseiten. Welche Bedeutung haben Eingangs- und Ausgangsgrad eines Knotens für eine Suchmaschine?

9.3 Graphen im Rechner

LK

Ein Bild ist für Menschen gemacht. Ein Programm braucht eine Datenstruktur. Für Graphen gibt es zwei gebräuchliche, und die Wahl zwischen ihnen ist eine echte Entwurfsentscheidung.

9.3.1 Die Adjazenzmatrix

Man nummeriert die Knoten durch und legt eine quadratische Tabelle an. In Zeile i, Spalte j steht, ob — und bei gewichteten Graphen: wie teuer — die Kante von Knoten i nach Knoten j ist.

	SB	NK	OTW	WND	HOM	MZG
SB	–	24	–	40	31	45
NK	24	–	8	–	15	–
OTW	–	8	–	10	–	–
WND	40	–	10	–	28	48
HOM	31	15	–	28	–	–
MZG	45	–	–	48	–	–

Zwei Dinge lassen sich hier sofort ablesen. Erstens ist die Matrix **symmetrisch zur Hauptdiagonalen** — genau das heißt „ungerichtet". Zweitens ist die Zahl der Einträge in einer Zeile der Knotengrad.

Stolperstein

Im Programm schreibt man statt des Strichs meist eine 0. Das geht so lange gut, wie kein Gewicht 0 vorkommt — sonst kann man „keine Straße" nicht mehr von „Straße der Länge 0" unterscheiden. Sauberer ist ein großer Platzhalterwert wie `UNENDLICH = 9999` oder eine zweite, rein logische Matrix daneben. Es ist dieselbe Falle wie bei den fehlenden Einträgen in der Übergangstabelle aus Abschnitt 5.6.

9.3.2 Die Adjazenzliste

Statt einer Tabelle führt man zu jedem Knoten eine Liste seiner Nachbarn:

```
  SB: NK (24), WND (40), HOM (31), MZG (45)
  NK: SB (24), OTW (8), HOM (15)
 OTW: NK (8), WND (10)
 WND: SB (40), OTW (10), HOM (28), MZG (48)
 HOM: SB (31), NK (15), WND (28)
 MZG: SB (45), WND (48)
```

In Python ist das ein Wörterbuch von Wörterbüchern — Knotenname auf Nachbarn, Nachbar auf Gewicht:

```python
GRAPH = {
    "SB":  {"NK": 24, "WND": 40, "HOM": 31, "MZG": 45},
    "NK":  {"SB": 24, "OTW":  8, "HOM": 15},
    "OTW": {"NK":  8, "WND": 10},
    "WND": {"SB": 40, "OTW": 10, "HOM": 28, "MZG": 48},
    "HOM": {"SB": 31, "NK": 15, "WND": 28},
    "MZG": {"SB": 45, "WND": 48},
}
```

In Pascal nimmt man die Matrix, weil sie sich als `array` direkt hinschreiben lässt:

```
const
  N = 6;
  UNENDLICH = 9999;
  ORTE: array [1..N] of String = ('SB', 'NK', 'OTW', 'WND', 'HOM', 'MZG');
  ENTFERNUNG: array [1..N, 1..N] of Integer = (
    {        SB   NK  OTW  WND  HOM  MZG }
    { SB }  ( 0,  24,   0,  40,  31,  45),
    { NK }  (24,   0,   8,   0,  15,   0),
    { OTW } ( 0,   8,   0,  10,   0,   0),
    { WND } (40,   0,  10,   0,  28,  48),
    { HOM } (31,  15,   0,  28,   0,   0),
    { MZG } (45,   0,   0,  48,   0,   0));
```

9.3.3 Welche Darstellung wann

Die Entscheidung hängt daran, wie **dicht** der Graph ist — also wie nah die Kantenzahl an der maximal möglichen liegt.

	Adjazenzmatrix	**Adjazenzliste**
Speicher	n^2 Plätze, unabhängig von der Kantenzahl	$n + 2m$ Plätze bei m Kanten
„Gibt es Kante $a \to b$?"	ein Zugriff	Liste von a durchsuchen
„Alle Nachbarn von a"	ganze Zeile prüfen, n Schritte	direkt da, $\deg(a)$ Schritte
geeignet für	dichte Graphen	dünne Graphen

Straßennetze, soziale Netze und das Internet sind extrem **dünn**: Kein Ort hat tausend Nachbarorte, kein Mensch Millionen Freunde. Für einen Graphen mit einer Million Knoten bräuchte die Matrix 10^{12} Plätze — undenkbar. Deshalb arbeitet man in der Praxis fast immer mit Adjazenzlisten.

Arbeitsauftrag 3

11. Geben Sie die Adjazenzmatrix des Königsberger Graphen an. Was steht dort, wo zwei Brücken dasselbe Paar verbinden?
12. Zeichnen Sie den Graphen zu dieser Adjazenzmatrix. Ist er gerichtet oder ungerichtet?

```
    A  B  C  D
A   0  1  1  0
B   0  0  1  1
C   0  0  0  1
D   0  0  0  0
```

13. Schreiben Sie eine Funktion, die zu einer Adjazenzmatrix prüft, ob der Graph ungerichtet ist.
14. Schreiben Sie eine Funktion, die aus einer Adjazenzmatrix die Adjazenzlisten erzeugt.
15. Ein sozialer Dienst hat 50 Millionen Mitglieder mit durchschnittlich 300 Kontakten. Rechnen Sie den Speicherbedarf beider Darstellungen aus, wenn ein Eintrag 4 Byte belegt.
16. Begründen Sie, warum die Adjazenzmatrix eines schlichten Graphen auf der Hauptdiagonalen nur Nullen enthält.

9.4 Breitensuche und Tiefensuche

LK

Die erste Frage an fast jeden Graphen lautet: *Welche Knoten erreiche ich von hier aus — und in welcher Reihenfolge schaue ich sie an?* Darauf gibt es zwei grundverschiedene Antworten.

9.4.1 Die Breitensuche

Die **Breitensuche** (englisch *breadth-first search*, BFS) arbeitet in Ringen: erst alle direkten Nachbarn, dann deren Nachbarn, dann die nächste Schale. Dafür braucht sie eine **Warteschlange** (Queue): Neu entdeckte Knoten werden hinten angehängt und vorne wieder entnommen — wer zuerst kommt, mahlt zuerst.

```python
def breitensuche(graph, start):
    besucht = [start]
    schlange = [start]
    while schlange:
        knoten = schlange.pop(0)          # vorne entnehmen: Queue
        for nachbar in graph[knoten]:
            if nachbar not in besucht:
                besucht.append(nachbar)
                schlange.append(nachbar)
    return besucht
```

```pascal
procedure Breitensuche(start: Integer);
var
  besucht: array [1..N] of Boolean;
  schlange: array [1..N] of Integer;
  kopf, ende, knoten, n: Integer;
begin
  for n := 1 to N do
    besucht[n] := False;
  kopf := 1;
  ende := 1;
  schlange[1] := start;
  besucht[start] := True;
  while kopf <= ende do
  begin
    knoten := schlange[kopf];
    Inc(kopf);
    Write(ORTE[knoten], ' ');
    for n := 1 to N do
      if (ENTFERNUNG[knoten, n] > 0) and not besucht[n] then
      begin
        besucht[n] := True;
        Inc(ende);
        schlange[ende] := n;
      end;
  end;
  WriteLn;
end;
```

9.4.2 Die Tiefensuche

Die **Tiefensuche** (*depth-first search*, DFS) macht das Gegenteil: Sie folgt einem Weg, so weit es geht, und kehrt erst um, wenn es nicht mehr weitergeht. Dafür braucht sie einen **Stapel** (Stack) — oder, viel bequemer, die Rekursion aus Kapitel 3, die ihren Stapel selbst mitbringt:

```python
def tiefensuche(graph, knoten, besucht=None):
    if besucht is None:
        besucht = []
    besucht.append(knoten)
    for nachbar in graph[knoten]:
        if nachbar not in besucht:
            tiefensuche(graph, nachbar, besucht)
    return besucht
```

```
var
  besucht: array [1..N] of Boolean;

procedure Tiefensuche(knoten: Integer);
var
  n: Integer;
begin
  besucht[knoten] := True;
  Write(ORTE[knoten], ' ');
  for n := 1 to N do
    if (ENTFERNUNG[knoten, n] > 0) and not besucht[n] then
      Tiefensuche(n);
end;
```

Gymnasium Ottweiler

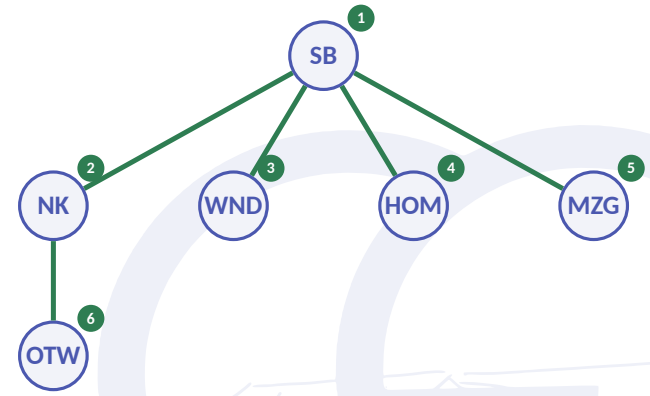
Stolperstein

Der Haken `if nachbar not in besucht` ist nicht Beiwerk, sondern lebenswichtig. Ohne ihn läuft die Suche in jedem Graphen mit einem Kreis — also in fast jedem — endlos im Kreis herum. Bei Bäumen aus Kapitel 3 brauchte man ihn nicht, weil ein Baum per Definition keine Kreise hat. Das ist der eine wesentliche Unterschied zwischen dem Durchlaufen eines Baums und dem eines Graphen.

9.4.3 Was die beiden unterscheidet

Beide Verfahren besuchen am Ende dieselben Knoten — alle erreichbaren. Die Reihenfolge ist aber eine völlig andere, und die Knoten, über die sie dabei zum ersten Mal entdeckt werden, bilden jeweils einen Baum, den **Suchbaum**:

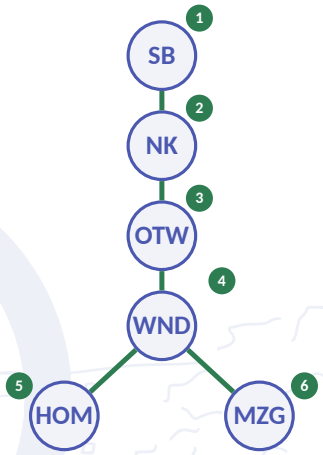
Breitensuche: erst alles Nahe



SB, NK, WND, HOM, MZG, OTW

Tiefe 2, Breite 4 — kürzeste Wege nach Kantenzahl

Tiefensuche: erst so weit wie möglich



SB, NK, OTW, WND, HOM, MZG

Tiefe 4, Breite 2 — ein langer Pfad

Beide Suchen starten in Saarbrücken und besuchen dieselben sechs Orte — in verschiedener Reihenfolge und mit verschieden geformten Suchbäumen.

	Breitensuche	**Tiefensuche**
Hilfsstruktur	Warteschlange (Queue)	Stapel (Stack) oder Rekursion
Suchbaum	breit und flach	schmal und tief
findet zuerst	Knoten mit den wenigsten Kanten Abstand	irgendeinen Knoten, möglichst weit weg
Speicherbedarf	kann riesig werden (eine ganze Schale)	nur der aktuelle Pfad
typische Aufgabe	kürzester Weg nach Kantenzahl, Nachbarschaft	Kreise finden, Labyrinthe, Zusammenhang prüfen

Im Bedienfeld lassen sich beide Suchen — und die beiden Algorithmen der nächsten Abschnitte — auf genau diesem Graphen Schritt für Schritt verfolgen. Grün heißt fertig, Blau heißt gerade in Arbeit.

> In der Webfassung steht hier ein Bedienfeld zum Ausprobieren: **Graph-Werkzeug**.

Merke

Die Breitensuche findet den Weg mit den **wenigsten Kanten** — aber nicht den kürzesten in Kilometern. Sie zählt Stationen, nicht Gewichte.

Im Saarland-Graphen sieht man das an Merzig und Neunkirchen: Die Breitensuche liefert MZG → SB → NK mit zwei Kanten, und das sind $45 + 24 = 69$ km. Der kürzeste Weg ist aber MZG → WND → OTW → NK mit drei Kanten und nur 66 km. Für gewichtete Graphen braucht es deshalb den Algorithmus aus dem nächsten Abschnitt.

Nebenbei beantworten beide Suchen eine Frage aus Abschnitt 9.2.2 ganz von selbst: Besucht die Suche von einem Knoten aus *alle* Knoten, dann ist der Graph zusammenhängend.

Arbeitsauftrag 4

17. Führen Sie Breiten- und Tiefensuche auf dem Saarland-Graphen ab Ottweiler aus. Notieren Sie beide Reihenfolgen und zeichnen Sie beide Suchbäume.
18. Beide Verfahren hängen davon ab, in welcher Reihenfolge die Nachbarn aufgelistet sind. Geben Sie für die Tiefensuche ab Saarbrücken eine Nachbarreihenfolge an, bei der Merzig als zweiter Ort besucht wird.
19. Erklären Sie, warum die Breitensuche eine Warteschlange und die Tiefensuche einen Stapel braucht — und was passiert, wenn man die beiden vertauscht.
20. Schreiben Sie die Tiefensuche ohne Rekursion, mit einer Liste als Stapel.
21. Erweitern Sie die Breitensuche so, dass sie zu jedem Knoten zusätzlich seinen Abstand in Kanten vom Startknoten ausgibt.
22. Schreiben Sie eine Funktion `zusammenhaengend(graph)`, die mit einer einzigen Suche prüft, ob der Graph zusammenhängend ist.
23. In einem Labyrinth sucht ein Roboter den Ausgang. Welches der beiden Verfahren würden Sie wählen, wenn der Speicher knapp ist, und welches, wenn der Ausgang garantiert nah liegt? Begründen Sie.

Gymnasium Ottweiler

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

9.6 Spannbäume: der Algorithmus von Kruskal

LK

Eine andere Frage, dieselben Daten: Ein Anbieter will alle sechs Orte an ein Glasfasernetz anschließen und darf das Kabel nur entlang der vorhandenen Straßen verlegen. Wie viel Kabel braucht er mindestens?

Gesucht ist eine Auswahl von Kanten, die *alle* Knoten verbindet und möglichst *wenig* kostet. Kreise wären Verschwendung — jede Kante in einem Kreis ließe sich weglassen, ohne dass etwas zerfällt. Was übrig bleibt, ist nach Abschnitt 9.2.3 ein Baum.

Definition: Minimaler Spannbaum

Ein **Spannbaum** eines zusammenhängenden Graphen $G = (V, E)$ ist ein Teilgraph, der alle Knoten aus V enthält, zusammenhängend ist und keinen Kreis besitzt. Er hat damit stets genau $|V| - 1$ Kanten.

Ein **minimaler Spannbaum** ist ein Spannbaum, bei dem die Summe der Kantengewichte so klein wie möglich ist.

Der Algorithmus von **Joseph Kruskal** (1956) ist erfreulich einfach — und er ist *gierig*: Er trifft in jedem Schritt die lokal beste Entscheidung und nimmt sie nie zurück.

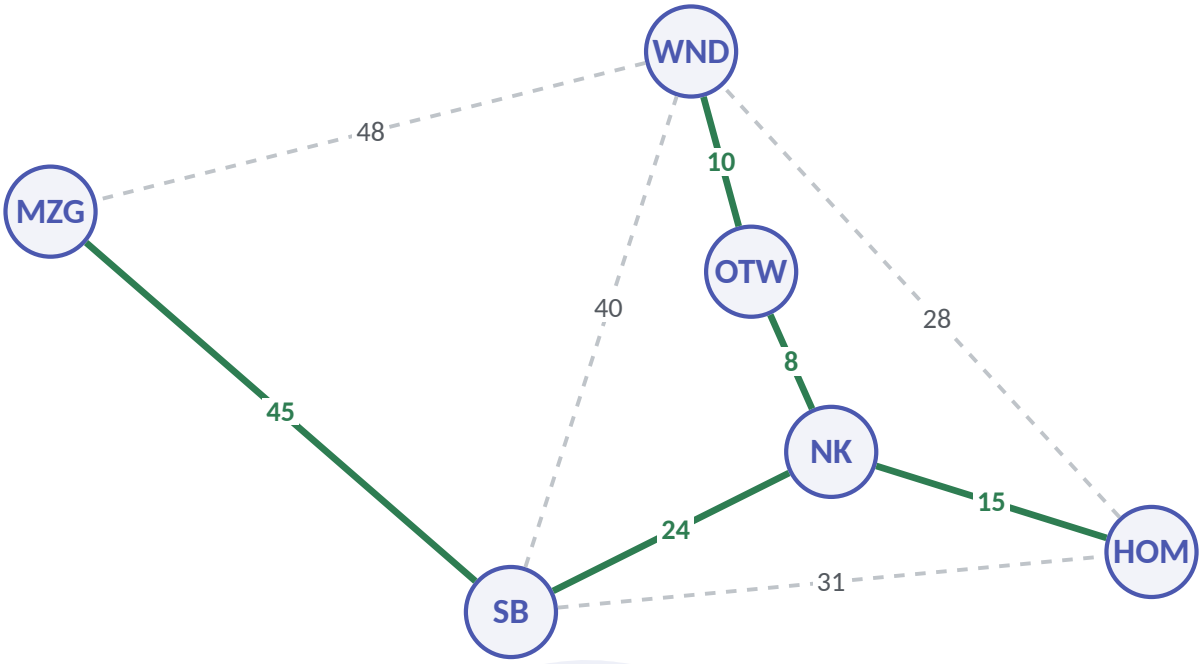
Definition: Algorithmus von Kruskal

1. Sortiere alle Kanten aufsteigend nach ihrem Gewicht.
2. Gehe die Liste von vorn durch. Nimm eine Kante in den Baum auf, **wenn sie keinen Kreis schließt** — wenn ihre beiden Endknoten also noch nicht im selben Teilstück hängen.
3. Fertig, sobald $|V| - 1$ Kanten gewählt sind.

Beispiel: Das Glasfaserkabel im Saarland

Kante	km	Entscheidung
NK – OTW	8	genommen
OTW – WND	10	genommen
NK – HOM	15	genommen
SB – NK	24	genommen
HOM – WND	28	**verworfen** — HOM und WND hängen schon über NK und OTW zusammen
SB – HOM	31	**verworfen** — SB und HOM hängen schon über NK zusammen
SB – WND	40	**verworfen**
SB – MZG	45	genommen — jetzt sind alle sechs Orte verbunden
WND – MZG	48	nicht mehr nötig

Fünf Kanten, zusammen $8 + 10 + 15 + 24 + 45 = 102$ km. Weniger geht nicht.



$8 + 10 + 15 + 24 + 45 \quad = \quad 102\,\text{km}$

Der minimale Spannbaum, durchgezogen. Die vier gestrichelten Kanten hätten jeweils einen Kreis geschlossen.

Die ganze Kunst steckt in der Frage „schließt diese Kante einen Kreis?“. Einfach, aber langsam ist es, jedem Knoten eine Gruppennummer zu geben und beim Verschmelzen zweier Gruppen alle Nummern umzuschreiben:

```
def kruskal(graph):
    kanten = sorted((laenge, a, b)
                    for a in graph for b, laenge in graph[a].items() if a < b)
    gruppe = {knoten: knoten for knoten in graph}
    baum = []
    for laenge, a, b in kanten:
        if gruppe[a] != gruppe[b]:              # verschiedene Gruppen: kein Kreis
            alt, neu = gruppe[b], gruppe[a]
            for knoten in gruppe:               # Gruppen verschmelzen
                if gruppe[knoten] == alt:
                    gruppe[knoten] = neu
            baum.append((a, b, laenge))
    return baum
```

```
>>> baum = kruskal(GRAPH)
>>> sum(laenge for _, _, laenge in baum)
102
```

Stolperstein

„Gierig" funktioniert selten. Dass es hier funktioniert, ist eine Besonderheit des Spannbaumproblems und muss bewiesen werden — bei den Sortierverfahren aus Kapitel 4 hätte eine solche Strategie kläglich versagt, und beim Rundreiseproblem im nächsten Abschnitt tut sie es auch. Verwechseln Sie „einfach" nicht mit „offensichtlich richtig".

Arbeitsauftrag 6

31. Führen Sie Kruskal auf dem Saarland-Graphen durch, ohne in das Beispiel zu schauen, und vergleichen Sie danach.
32. Begründen Sie, warum ein Spannbaum bei sechs Knoten immer genau fünf Kanten hat.
33. Verdoppeln Sie die Kante SB – MZG auf 90 km und bestimmen Sie den neuen minimalen Spannbaum. Welche Kante rückt nach, und um wie viel wird das Netz teurer?
34. Ein minimaler Spannbaum ist nicht immer eindeutig. Geben Sie einen kleinen Graphen an, der zwei verschiedene minimale Spannbäume besitzt.
35. Ist der kürzeste Weg von Merzig nach Ottweiler im minimalen Spannbaum enthalten? Prüfen Sie nach und erklären Sie das Ergebnis.
36. Das Verschmelzen der Gruppen im Programm läuft über alle Knoten. Schätzen Sie den Gesamtaufwand bei n Knoten und m Kanten ab.

Gymnasium Ottweiler

9.7 Hamiltonkreise und das Problem des Handlungsreisenden LK

Euler fragte nach einem Weg über jede **Kante**. Die zweite berühmte Frage dreht das um: Gibt es einen Rundweg über jeden **Knoten** genau einmal?

Definition: Hamiltonkreis

Ein **Hamiltonkreis** ist ein geschlossener Weg, der jeden Knoten des Graphen genau einmal besucht. Benannt ist er nach dem irischen Mathematiker William Rowan Hamilton.

Die beiden Fragen sehen sich zum Verwechseln ähnlich — und könnten in ihrer Schwierigkeit kaum weiter auseinanderliegen. Für den Eulerweg genügt ein Blick auf die Knotengrade. Für den Hamiltonkreis ist bis heute **kein** vergleichbar einfaches Kriterium bekannt, und es wird vermutet, dass es keines gibt.

Beispiel: Die Rundreise durchs Saarland

Im Saarland-Graphen kommt man trotzdem mit reinem Nachdenken ans Ziel — über die Knotengrade.

Ottweiler hat nur zwei Nachbarn, Neunkirchen und St. Wendel. Ein Rundweg muss Ottweiler einmal betreten und einmal verlassen, also müssen **beide** Kanten an Ottweiler zur Tour gehören. Dasselbe gilt für Merzig mit seinen Nachbarn Saarbrücken und St. Wendel.

Vier Kanten liegen damit fest: NK – OTW, OTW – WND, WND – MZG und MZG – SB. Und jetzt kommt der entscheidende Schluss: St. Wendel hat damit **schon zwei** Kanten in der Tour. Mehr darf kein Knoten haben, also scheiden WND – SB und WND – HOM aus. Übrig bleibt der Streckenzug

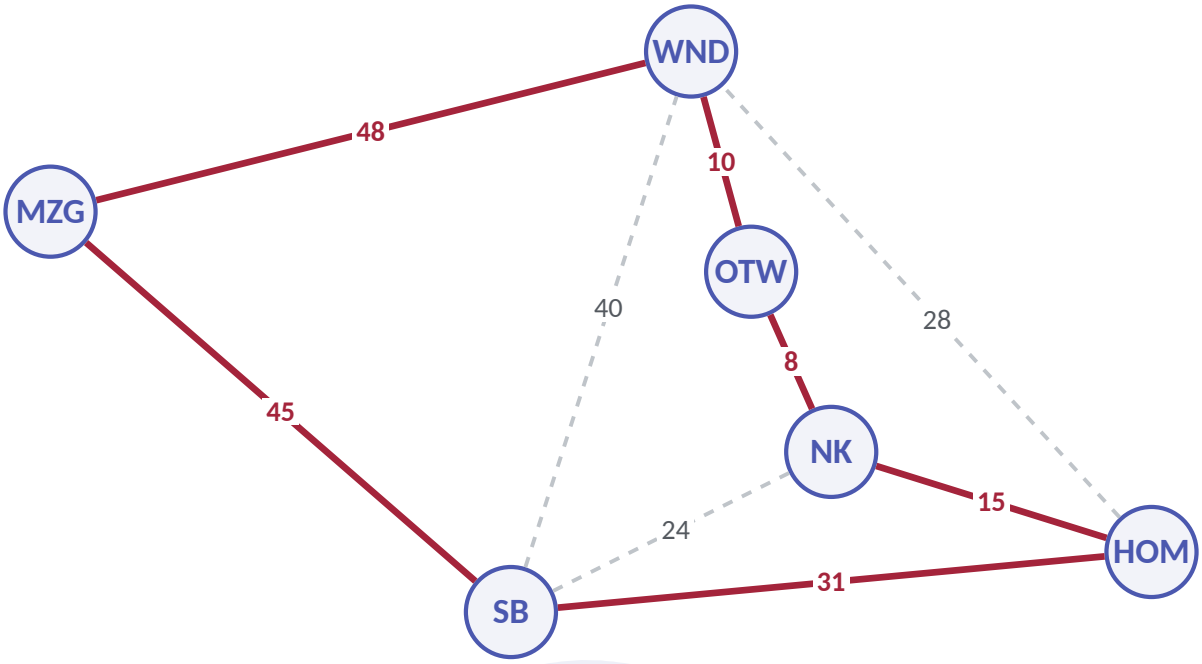
$$\text{NK} - \text{OTW} - \text{WND} - \text{MZG} - \text{SB}$$

mit den losen Enden NK und SB. Es fehlt nur noch Homburg, und der muss genau diese beiden Enden verbinden. Die Kanten NK – HOM und HOM – SB gibt es, also schließt sich der Kreis:

$$\text{SB} \to \text{MZG} \to \text{WND} \to \text{OTW} \to \text{NK} \to \text{HOM} \to \text{SB}$$

Die Länge: $45+48+10+8+15+31 = 157$ km. Und weil jeder Schritt der Überlegung erzwungen war, ist dies der **einzige** Hamiltonkreis des Graphen — gefunden ohne eine einzige durchprobierte Tour.

Gymnasium Ottweiler



SB → MZG → WND → OTW → NK → HOM → SB = 157 km

Der einzige Hamiltonkreis. Die drei gestrichelten Kanten bleiben ungenutzt.

Wenn der Graph gewichtet ist, interessiert nicht nur, *ob* es eine Rundreise gibt, sondern welche die **kürzeste** ist. Das ist das **Problem des Handlungsreisenden** (englisch *traveling salesman problem*, TSP) — eine Vertreterin soll alle Kunden besuchen und mit möglichst wenig Fahrerei wieder zu Hause ankommen.

Gymnasium Ottweiler

Beispiel: Vier Orte, drei Rundreisen

Für eine Rundreise zählt nicht, ob zwischen zwei Orten eine direkte Straße liegt, sondern nur, wie weit es ist. Wir nehmen deshalb nicht die Kantengewichte, sondern die **kürzesten Entfernungen** — und die hat Dijkstra im vorigen Abschnitt schon geliefert. Für Ottweiler, Neunkirchen, St. Wendel und Homburg sind das:

	NK	WND	HOM
OTW	8	10	23
NK	—	18	15
WND		—	28

Damit ist jeder Ort mit jedem verbunden, und es gibt genau drei verschiedene Rundreisen:

Rundreise	Rechnung	Länge
OTW → NK → HOM → WND → OTW	$8 + 15 + 28 + 10$	**61 km**
OTW → HOM → NK → WND → OTW	$23 + 15 + 18 + 10$	66 km
OTW → NK → WND → HOM → OTW	$8 + 18 + 28 + 23$	77 km

Bei vier Orten reicht Hinschreiben und Nachrechnen. Das ist kein Algorithmus, sondern erschöpfendes Durchprobieren — und genau da liegt das Problem.

Drei Möglichkeiten kann man durchprobieren. Und bei mehr Orten? In einem vollständigen Graphen mit n Knoten gibt es $\frac{(n-1)!}{2}$ verschiedene Rundreisen: $(n-1)!$ Reihenfolgen für die übrigen Knoten, geteilt durch 2, weil jede Tour vorwärts und rückwärts dieselbe ist.

Orte	Rundreisen
4	3
5	12
6	60
8	2 520
10	181 440
12	19 958 400
15	43 589 145 600
20	60 822 550 204 416 000

Gymnasium Ottweiler

Merke

Das Rundreiseproblem gehört zu den Aufgaben, für die kein Verfahren bekannt ist, das wesentlich besser wäre als Ausprobieren. Es ist damit ein Verwandter der Fragen aus Abschnitt 4.4: Nicht unlösbar — jede einzelne Instanz lässt sich im Prinzip ausrechnen —, aber für wachsendes n praktisch unbezahlbar.

Die Praxis lebt deshalb mit **Näherungen**. Ein einfaches Verfahren ist *Nächster Nachbar*: Fahre immer zum nächsten noch nicht besuchten Ort und am Ende zurück zum Start. Im Vier-Orte-Beispiel findet es von jedem Startort aus die 61 km — das Optimum. Garantiert ist das aber nicht, im Gegenteil: Mit mehr Orten wird das Verfahren regelmäßig schlechter, weil die letzte Rückfahrt beliebig teuer werden kann. Genau wie bei Kruskal im vorigen Abschnitt gilt: Gierig ist einfach, nicht richtig — nur hat man dort einen Beweis, hier nicht.

In der Logistik sind solche Näherungen trotzdem Alltag: Eine Tour, die 5 % länger ist als das Optimum, aber in einer Sekunde berechnet wird, ist einem perfekten Ergebnis nach drei Tagen Rechenzeit haushoch überlegen.

Arbeitsauftrag 7

37. Prüfen Sie die Überlegung aus dem Beispiel nach: Warum müssen beide Kanten an einem Knoten vom Grad 2 zu jedem Hamiltonkreis gehören?
38. Besitzt der Königsberger Graph einen Hamiltonkreis? Begründen Sie.
39. Geben Sie einen zusammenhängenden Graphen an, der einen Eulerkreis, aber keinen Hamiltonkreis besitzt — und einen mit Hamiltonkreis, aber ohne Eulerkreis.
40. Konstruieren Sie einen vollständigen Graphen mit vier Knoten, bei dem *Nächster Nachbar* vom Startknoten aus **nicht** die kürzeste Rundreise findet. Tipp: Die letzte Rückfahrt ist die Stelle, an der das Verfahren blind ist.
41. Ein Rechner prüft eine Million Rundreisen pro Sekunde. Wie lange braucht er bei 15, bei 20 Orten? Vergleichen Sie mit der Brute-Force-Tabelle aus Abschnitt 8.3.3.
42. Schreiben Sie ein Programm, das zu einem vollständigen gewichteten Graphen alle Rundreisen durchprobiert und die kürzeste ausgibt. Testen Sie es mit dem Vier-Orte-Beispiel. Ab wie vielen Orten wird Ihr Rechner spürbar langsam?
43. Erklären Sie den Unterschied zwischen Eulerweg und Hamiltonkreis in zwei Sätzen, ohne die Wörter „Kante” und „Knoten” zu verwechseln.

Gymnasium Ottweiler

9.8 Zusammenfassung

LK

- Ein Graph $G = (V, E)$ besteht aus einer Knotenmenge und einer Kantenmenge; die Kantenmenge ist eine Relation auf den Knoten. Ist sie symmetrisch, heißt der Graph ungerichtet. Tragen die Kanten zusätzlich Zahlen, heißt er gewichtet.
- Der Knotengrad zählt die Kanten an einem Knoten. Die Summe aller Grade ist doppelt so groß wie die Kantenzahl, also gerade.
- Schlicht heißt: keine Schlingen, keine Mehrfachkanten. Vollständig heißt: jeder mit jedem. Zusammenhängend heißt: von überall nach überall.
- Bäume sind zusammenhängende, kreisfreie Graphen mit genau $n - 1$ Kanten.
- Im Rechner steht ein Graph als Adjazenzmatrix (n^2 Plätze, schneller Kantentest) oder als Adjazenzliste (sparsam bei dünnen Graphen, schnelle Nachbarschaft). Reale Netze sind dünn, deshalb gewinnt fast immer die Liste.
- Die Breitensuche arbeitet mit einer Warteschlange und findet die Wege mit den wenigsten Kanten; die Tiefensuche arbeitet mit einem Stapel oder per Rekursion und läuft erst einmal in die Tiefe. Ohne Markieren der besuchten Knoten laufen beide im Kreis.
- Der Algorithmus von Dijkstra löst das Kürzeste-Wege-Problem in gewichteten Graphen, solange kein Gewicht negativ ist: immer den kleinsten offenen Knoten abhaken und die Nachbarn relaxieren.
- Der Algorithmus von Kruskal konstruiert einen minimalen Spannbaum: Kanten nach Gewicht sortieren und jede nehmen, die keinen Kreis schließt.
- Ein Eulerweg benutzt jede Kante einmal — ob es ihn gibt, verrät der Knotengrad. Ein Hamiltonkreis besucht jeden Knoten einmal — dafür ist kein einfaches Kriterium bekannt. Die gewichtete Fassung ist das Problem des Handlungsreisenden, dessen $\frac{(n-1)!}{2}$ Rundreisen jede Brute-Force-Suche sprengen.

Wiederholungsaufgaben

44. Definieren Sie einen gerichteten Graphen und erklären Sie, was es bedeutet, dass die Kantenmenge eine Relation ist.
45. Geben Sie zu einem selbst gezeichneten Graphen mit fünf Knoten die Adjazenzmatrix, die Adjazenzlisten und die mengentheoretische Darstellung an.
46. Führen Sie auf diesem Graphen Breiten- und Tiefensuche durch und zeichnen Sie beide Suchbäume.
47. Erklären Sie den Unterschied zwischen dem Ergebnis der Breitensuche und dem Ergebnis von Dijkstra. Wann liefern beide denselben Weg?
48. Vergleichen Sie Dijkstra und Kruskal: Was ist das Ziel, was die Strategie, und warum reicht bei Kruskal eine gierige Vorgehensweise?
49. Begründen Sie, warum sich das Rundreiseproblem nicht einfach dadurch lösen lässt, dass man zwischen allen Ortspaaren erst Dijkstra laufen lässt.
50. Modellieren Sie den Stundenplan Ihres Kurses als Graphen: Knoten sind Fächer, eine Kante verbindet zwei Fächer, wenn dieselbe Lehrkraft sie unterrichtet. Welche Frage über den Stundenplan beantwortet der Knotengrad?