

# Fachdidaktik Informatik

Semesterbegleitende Übung

## Breitensuche

*DRAFT*

*Ausarbeitung Leitprogrammartiger Unterrichtsunterlagen für den  
Informatikunterricht am Gymnasium*

Sabina Schellenberg & Johannes Popp  
sabinasc@student.ethz.ch & jpopp@ethz.ch

08-923-377 & 09-931-999

Lehrdiplom Informatik

HS 2017

# Inhaltsverzeichnis

<b>1</b>	<b>Konzeption</b>	<b>1</b>
1.1	Concept Map . . . . .	1
1.2	Lernziele . . . . .	1
1.2.1	Leitidee . . . . .	1
1.2.2	Dispositionsziel . . . . .	1
1.2.3	Operationalisierte Lernziele . . . . .	1
<b>2</b>	<b>Unterlagen</b>	<b>3</b>
2.1	Graphen . . . . .	3
2.1.1	Begriffe . . . . .	3
2.1.2	Defintionen . . . . .	4
2.1.3	Zusammenfassung und Kontrollaufgaben . . . . .	6
2.2	Breitensuche . . . . .	7
2.2.1	Einführung . . . . .	7
2.2.2	Algorithmus . . . . .	10
2.2.3	Anwendung und Erweiterung . . . . .	12
2.3	Lösungen . . . . .	15
<b>3</b>	<b>Anhang</b>	<b>17</b>

# Kapitel 1

## Konzeption

### 1.1 Concept Map

Das Thema der vorliegenden Arbeit ist die Ausarbeitung des Algorithmus der Breitensuche in Form Lernprogrammartiger Unterlagen für das Gymnasium. Zu diesem Zweck wurde folgende Concept-Map entworfen, die sowohl das Vorwissen als auch einen kleinen Ausblick auf weitere Themen geben soll. Dabei wurde Konzept, welches als Vorwissen schon bekannt sein sollte, aber im Rahmen dieser Arbeit nochmal wiederholt werden soll, blau gefärbt. Neue Konzepte, die in dieser Arbeit behandelt und eingeführt werden sollen, wurden grün gefärbt. Weitere Konzepte, die nicht mehr in dieser Arbeit behandelt werden, wurden orange gefärbt.

### 1.2 Lernziele

Aus dieser Concept-Map lassen sich folgende Lernziele für diese Arbeit ableiten.

#### 1.2.1 Leitidee

Graphen spielen eine wichtige Rolle in unserem Alltag (S-Bahnnetzwerk, Facebook, Websites, ...). Mit diesen Netzwerken sind viele Fragen verbunden: Über wie viel Freundschaften bin ich mit einer anderen Person verbunden? Wie lange ist die kürzeste Verbindung von A nach B? Damit man ein grundlegendes Verständnis dafür entwickelt, muss man sich überlegen, wie man sich auf Graphen bewegen kann. Eine Möglichkeit bietet die Breitensuche, welche einen naiven Einstieg bildet.

#### 1.2.2 Dispositionsziel

Die SuS wissen, dass man mit gewisse Probleme mit Hilfe von Graphen modellieren und mit Graphenalgorithmien lösen kann. Sie können Probleme analysieren und beurteilen, ob sie mit Hilfe von einer Breitensuche in einem Graphen gelöst werden können.

#### 1.2.3 Operationalisierte Lernziele

1. Die SuS kennen die Begriffe zur Darstellung von einfachen Graphen: (un)gerichtet, Knoten und Kante.

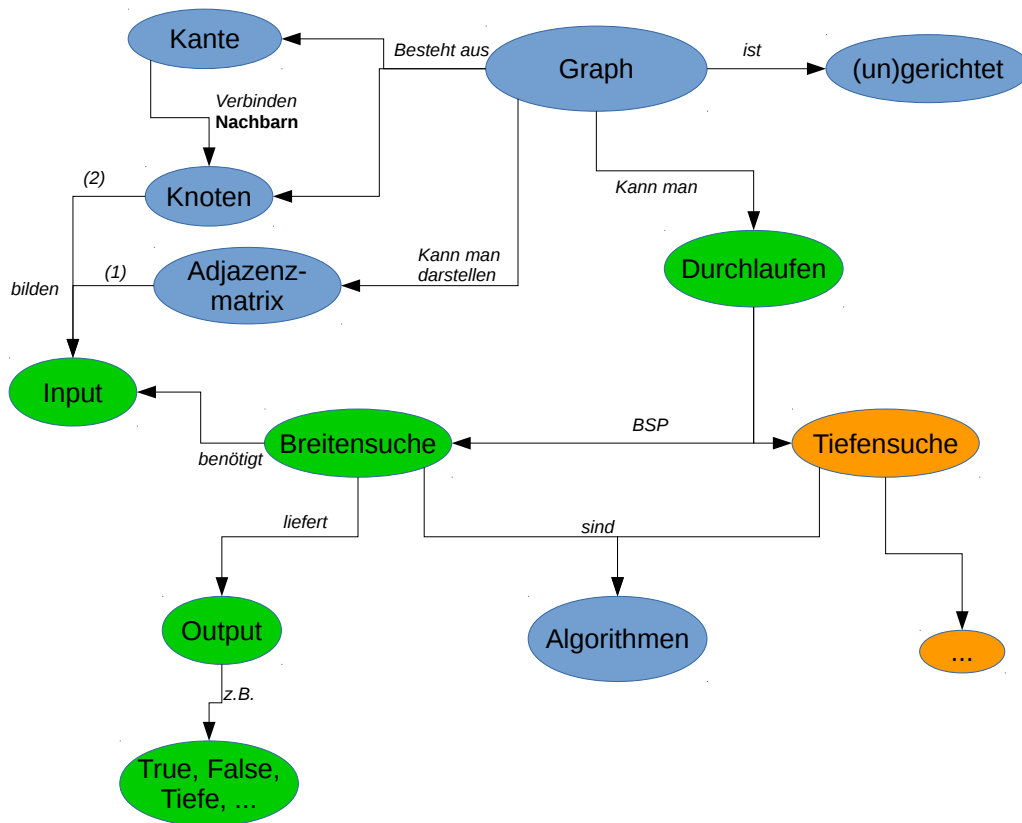


Abbildung 1.1: Concept-Map zum Thema Breitensuche. Blaue Konzepte bilden das Vorwissen ab. Grüne Konzepte werden in dieser Arbeit eingeführt und behandelt. Orange Konzepte bilden einen Ausblick auf Konzepte, welche im Rahmen dieser Arbeit nicht mehr behandelt werden können.

2. Die SuS kennen verschiedene Darstellungsmöglichkeiten von Graphen und können diese ineinander überführen: Zeichnung, Knoten- /Kantenmenge und Adjazenzmatrix.
3. Die SuS können die Nachbarn von Knoten auf verschiedenen Darstellungen von Graphen bestimmen.
4. Die SuS können ein Programm schreiben, welches die Nachbarn eines Knoten eines bestimmten Knoten eines beliebigen Graphen ausgibt.

... Operationalisierte Lernziele: Die SuS können die Funktionsweise einer Breitensuche in einem Graphen beschreiben Die SuS können für ein gegebenes Problem Breitensuche in einem Graphen anwenden Die SuS können ein gegebenes Problem mit einem Graphen modellieren und mittels Breitensuche eine Lösung finden (evtl. spezifischer) Die SuS können die Breitensuche in TigerJython implementieren Die SuS können die Laufzeit einer Breitensuche beurteilen (sofern O notation bekannt...) Die SuS können den Speicherbedarf der Breitensuche beurteilen



## 2.1.2 Definitionen

**Definition 2.1.** Mathematisch besteht ein **Graph**  $G$  aus einer **Menge** von **Knoten**  $V$  (engl. *vertex*) und einer **Menge** von **Kanten**  $E$  (engl. *Edge*). Die Anzahl Knoten wird mit  $|V|$  und die Anzahl Kanten mit  $|E|$  bezeichnet.

**Definition 2.2.** Jede Kante eines **ungerichteten Graphen**  $e$  besteht aus zwei Knoten  $v_1$  und  $v_2$  und wird selbst als Menge dargestellt:  $e = \{v_1; v_2\}$ . Die Reihenfolge spielt dabei keine Rolle.

**Definition 2.3.** Jede Kante eines **gerichteten Graphen**  $e$  besteht aus einem Startknoten  $v_1$  und einem Zielknoten  $v_2$  und wird selbst als 2-Tupel dargestellt:  $e = (v_1; v_2)$ . Hierbei spielt die Reihenfolge eine Rolle.

**Beispiel 2.2.** Die Abbildung 2.2 zeigt einen ungerichteten Graph mit 5 Knoten und einen gerichteten Graphen mit 6 Knoten.

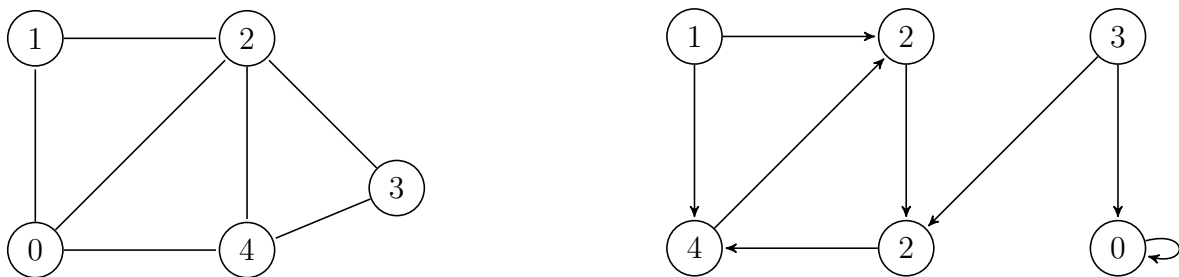


Abbildung 2.2: *Links*: Ein ungerichteter Graph mit 5 Knoten. *Rechts*: Ein gerichteter Graph mit 6 Knoten.

Die mathematische Darstellung der zwei Graphen lautet:

$$\begin{aligned} \text{Graph links: } G_l &= (V_l, E_l); \quad V_l = \{0, 1, 2, 3, 4\}; \\ E_l &= \{\{1, 2\}, \{1, 0\}, \{2, 0\}, \{2, 4\}, \{2, 3\}, \{3, 4\}, \{4, 0\}\} \end{aligned}$$

$$\begin{aligned} \text{Graph rechts: } G_r &= (V_r, E_r); \quad V_r = \{0, 1, 2, 3, 4, 5\}; \\ E_r &= \{(1, 2), (1, 4), (2, 5), (3, 5), (3, 0), (4, 2), (5, 4), (0, 0)\} \end{aligned}$$

**Aufgabe 2.1.** Bilden Sie für folgenden Graph (Abb. 2.3) die entsprechende mathematische Darstellung  $G = (V, E)$ .

**Aufgabe 2.2.** Zeichnen Sie den entsprechenden Graph, der zu folgender Mathematischen Darstellung gehört:

$$\begin{aligned} G &= (V, E); \quad V = \{0, 1, 2, 3, 4, 5, 6\}; \\ E &= \{(1, 3), (1, 6), (2, 1), (3, 3), (3, 4), (4, 2), (5, 4), (6, 3)\} \end{aligned}$$

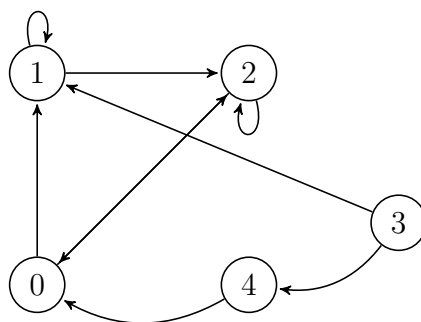


Abbildung 2.3: Ein gerichteter Graph.

**Definition 2.4.** Für einen Graphen  $G = (V, E)$  nehmen wir an, dass die Knoten in beliebiger Weise von 0 bis  $|V| - 1$  nummeriert sind. So bildet die **Adjazenzmatrix-Darstellung** des Graphen  $G$  eine  $|V| \times |V|$ -Matrix  $A = a_{ij}$  mit den Elementen

$$a_{ij} = \begin{cases} 1 & \text{falls } (i, j) \in E, \\ 0 & \text{sonst.} \end{cases}$$

**Beispiel 2.3.** Die Adjazenz-Matrizen ( $A_l$  und  $A_r$ ) zu den zwei Graphen in Abb. 2.2 lauten:

$$A_l = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \quad A_r = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

**Aufgabe 2.3.** Bilden Sie für den Graphen aus Abbildung 2.3 die entsprechende Adjazenzmatrix  $G = (V, E)$ .

**Aufgabe 2.4.** Zeichnen Sie zu folgender Adjazenzmatrix den entsprechenden Graphen.

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

---

**Programm 2.1.** Damit man einen Graphen in einer Textdatei speichern und lesen kann, benutzt man häufig ';' (Semicolon) um die Zahlen einer Zeile zu trennen. Mit einem Absatz wird eine neue Zeile der Matrix bestimmt. Schreiben Sie ein Programm, dass die Adjazenzliste eines Graphen auf den Bildschirm ausgibt (*print*). Benutzen Sie dafür das vorgegebene Programm, welches Graphen aus einer Textdatei einlesen kann und überprüfen Sie die Ausgabe mit den Textdateien.

**Definition 2.5.** Zwei Vertices sind zu **benachbart**, wenn eine direkte Verbindung durch eine Kante besteht. In einem ungerichteten Graphen sind immer beide Vertices benachbart. Hingegen spielt in einem gerichteten Graphen die Richtung der Kante eine Rolle, so dass die Nachbarschaft nur in eine Richtung gelten kann.

**Beispiel 2.4.** Im Graphen der Abbildung 2.2 links hat der Knoten 2 die Nachbarn 1,5,4 und 3. Dies kann man auch aus der Adjazenzmatrix  $A_l$  ablesen: Die zweite Reihe hat dort bei den Spalten 1,3,4 und 5 eine eins stehen.

Im Gegensatz dazu hat der Knoten 2 im Graphen der Abbildung 2.2 rechts nur den Nachbarknoten 5. Auch dies kann man wieder aus der zweiten Reihe der Adjazenzmatrix  $A_r$  ablesen.

**Aufgabe 2.5.** Welche Nachbarn hat der Knoten 3, der Knoten 1 und der Knoten 0 des Graphen in Abbildung 2.3.

**Programm 2.2.** Implementieren Sie eine Funktion `NACHBARKNOTEN(graph, knoten)`: Sie hat als Input einen Graphen und einen Knoten und gibt als Output eine Liste von Nachbarknoten des Knotens im Graphen aus. Testen Sie die Funktion, indem Sie sie mit verschiedenen Knoten aus einem Graphen aufrufen.

### 2.1.3 Zusammenfassung und Kontrollaufgaben

In diesem Kapitel haben Sie die Darstellung von Graphen wiederholt. Insbesondere haben Sie verschiedene Darstellung der Graphen kennen gelernt: Zeichnung, Menge von Kanten und Knoten und Adjazenzmatrix. Zusätzlich kennen Sie den Unterschied zwischen gerichteten und ungerichteten Graphen und können die Nachbarn eines Knoten in einem Graphen bestimmen.

**Kontrollaufgabe 1.** Beschreiben Sie Unterschiede und Gemeinsamkeiten von gerichteten und ungerichteten Graphen.

**Kontrollaufgabe 2.** Nennen Sie drei weitere Beispiele aus dem Alltag für Graphen. Bestimmen Sie dabei immer was die Knoten und was die Kanten darstellen. Handelt es sich dabei um gerichtete oder ungerichtete Graphen?

**Kontrollaufgabe 3.** Betrachten Sie folgenden Graphen (s. Abb. 2.4) und bestimmen Sie seine Knoten und Kanten Menge und bestimmen Sie zusätzlich die Adjazenzmatrix.

**Kontrollaufgabe 4.** Betrachten Sie folgende Knoten- und Kantenmenge. Zeichnen Sie den dazugehörigen Graphen und Bestimmen Sie die dazugehörige Adjazenzmatrix.

$$G = (V, E); \quad V = \{2, 3, 4, 5, 6, 7, 9\};$$
$$E = \{\{7, 3\}, \{9, 6\}, \{2, 6\}, \{3, 2\}, \{3, 4\}, \{4, 2\}, \{5, 4\}, \{7, 9\}\}$$



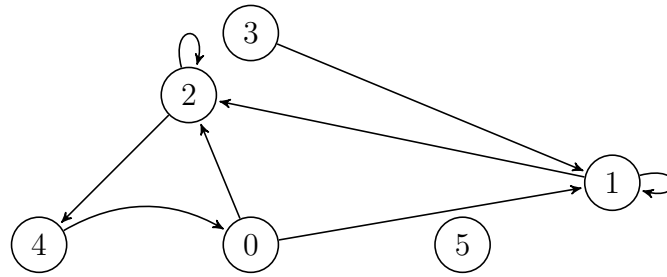


Abbildung 2.4: Ein weiterer Graph.

**Kontrollaufgabe 5.** Betrachten Sie folgende Adjazenzmatrix, zeichnen Sie den dazugehörigen Graphen und bestimmen Sie die Knoten- und Kantenmenge.

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

## 2.2 Breitensuche

### 2.2.1 Einführung

**Modellierung mit Graphen:** Viele Probleme heutzutage können mit einem Graphen modelliert und mit Hilfe entsprechender Graphenalgorithmien gelöst werden.

Wir haben gesehen dass ein Graph aus Knoten und Kanten besteht, wobei eine Kante jeweils zwei Knoten verbindet. Viele Probleme aus unserem Alltag können mit diesen Grundelementen repräsentiert werden. Zum Beispiel kann man das Netz des öffentlichen Verkehrs als Graphen darstellen, indem man alle Stationen als Knoten abbildet, und die Linien, die die Stationen verbinden als gerichtete Kanten im Graphen, je nachdem in welche Richtung die Verkehrsmittel von der einen Station zur nächsten fahren können.

**Graphenalgorithmus:** Ein konkretes Problem kann also mit Hilfe von einem Graphen abstrahiert werden. Ein Graphenalgorithmus kann auf einem Graphen angewendet werden, und das Resultat kann wieder auf das ursprüngliche, konkrete Problem übertragen werden. Durch diese Abstraktion kann also der gleiche Graphenalgorithmus eine Menge von verschiedenen konkreten Problemen lösen.

**Traversieren:** Es gibt viele verschiedene Algorithmen für Graphen, und eine wichtige Klasse von Algorithmen ist das Durchsuchen bzw. Traversieren von Graphen. Beim Traversieren von einem Graphen werden die Knoten des Graphen besucht, und man bewegt sich dabei entlang den Kanten. Der Graph kann auch traversiert werden, um einen bestimmten Knoten im Graphen zu suchen.

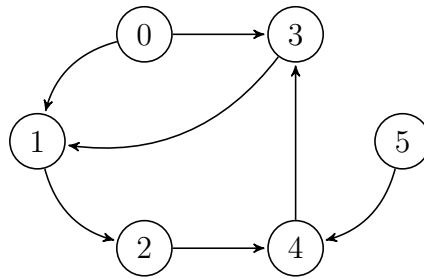


Abbildung 2.5: Ein Beispiels-Graph

**Erreichbarkeit:** Betrachten Sie den Graphen in Abb. 2.5. Ausgehend von einem Startknoten möchten wir untersuchen, ob ein Zielknoten erreichbar ist oder nicht. Ist der Knoten 2 von Knoten 0 aus erreichbar? Ist Knoten 5 auch erreichbar?

Um diese Fragen zu beantworten haben Sie vermutlich automatisch den Graphen betrachtet und visuell beurteilt, ob Sie vom Knoten 0 aus gewissen Kanten folgen können um den Zielknoten zu erreichen.

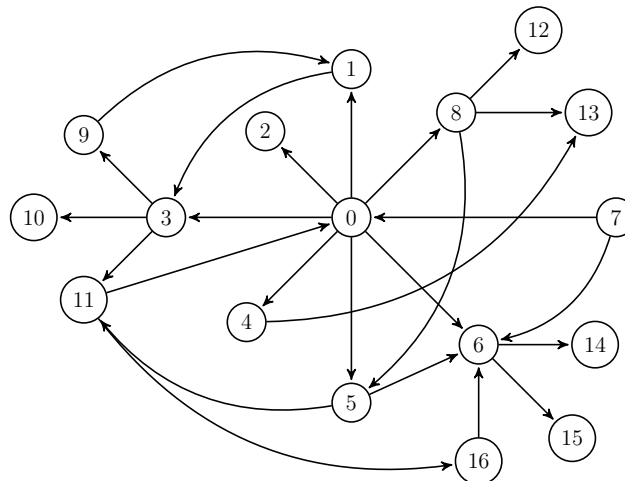


Abbildung 2.6: Ein etwas komplexerer Graph

Betrachten Sie nun den Graphen in Abb.2.6 und beurteilen Sie, ob man von Knoten 7 den Knoten 16 erreichen kann. Kann man von Knoten 5 den Knoten 1 erreichen?

Wie Sie merken, ist das Lösen der Aufgabe in einem etwas komplizierteren Graphen schon viel schwieriger. Stellen Sie sich jetzt aber vor, Sie haben einen Graphen mit tausenden von Knoten vor sich. Eine visuelle Beurteilung wird nun fast unmöglich und das Problem ist von Hand nicht mehr lösbar. Wir möchten also einen Algorithmus entwickeln, damit ein Computer das Problem für uns lösen kann. Wie wir gesehen haben, können Graphen mit Hilfe von Matrizen dargestellt werden. Ein Computer kann aber nicht auf eine visuelle Beurteilung zurückgreifen und braucht zusätzlich noch Instruktionen, die ihm mitteilen, wie das Problem überhaupt gelöst werden kann.

**Algorithmus entwickeln:** Wir möchten also einen Algorithmus entwickeln, der für einen Startknoten in einem Graphen beurteilen kann, ob ein anderer Knoten von diesem

Startknoten erreichbar ist oder nicht. Bei der Beantwortung der Frage zu den Graphen in Abb. 2.5 und 2.6 haben Sie wahrscheinlich intuitiv begonnen, ein paar Wege auszuprobieren und je nachdem wieder zu verwerfen. Damit das Problem von einem Computer gelöst werden kann braucht er aber ein klares, strukturiertes Vorgehen, um die Frage nach der Erreichbarkeit zu beantworten.

Es gibt verschiedene Ansätze für ein solches Vorgehen. Bevor wir nun ein solches entwickeln überlegen wir uns noch zusätzlich, ob wir ausser der Erreichbarkeit sonst noch Informationen haben wollen. Wenn wir herausgefunden haben, dass ein Knoten von einem Startknoten aus erreichbar ist, möchten wir oftmals auch gerne wissen, wie weit entfernt er ist (in Anzahl Kanten, die traversiert werden müssen, um den Knoten zu erreichen). Falls es mehrere Wege zum Knoten gibt sind wir häufig an der kürzesten Distanz interessiert. Nicht zuletzt interessiert uns dann auch der konkrete Weg, der vom Startknoten zum Zielknoten führt.

**Kürzester Weg:** Im Graphen aus Abb. 2.5 ist Knoten 4 von Knoten 0 über Knoten 1 und 2 erreichbar, aber auch von Knoten 0 über Knoten 3, 1, und 2. Der erste Weg ist jedoch der Kürzere.

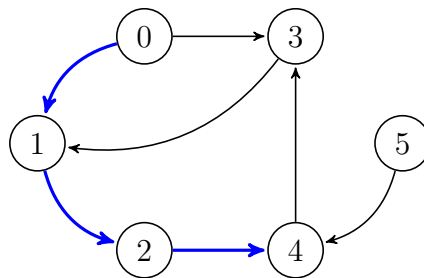


Abbildung 2.7: Weg von Knoten 0 zu Knoten 4 über Knoten 1 und 2

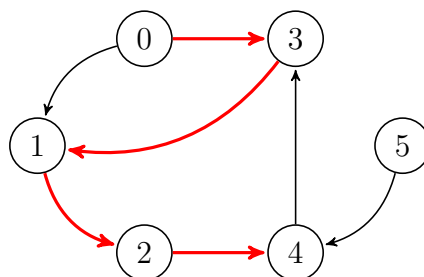


Abbildung 2.8: Weg von Knoten 0 zu Knoten 4 über Knoten 3, 1 und 2

**Aufgabe 2.6.** Wir möchten also ein Vorgehen entwickeln, welches in einem Graphen von einem Startknoten aus einen Knoten auf dem kürzesten Weg sucht und uns darüber informiert, ob er vom Startknoten aus erreichbar ist. Überlegen Sie sich, wie so ein Vorgehen aussehen könnte.

Beachten Sie dabei Folgendes: wenn wir beim Startknoten beginnen, möchten wir zuerst alle Knoten absuchen, welche mit minimaler Distanz vom Startknoten aus erreichbar sind. Welche Knoten sind das? Wenn wir in kürzester Distanz den gesuchten Knoten

nicht gefunden haben müssen wir die nächste grössere Distanz in Kauf nehmen, in der Hoffnung, den Knoten dort zu finden. Wenn wir so vorgehen und den Knoten finden wissen wir nämlich, dass wir ihn auf dem kürzesten möglichen Weg gefunden haben.

## 2.2.2 Algorithmus

Wie wir schon erwähnt haben gibt es mehrere Möglichkeiten für ein Vorgehen, welches einen Graphen traversiert und nach einem bestimmten Knoten sucht. In diesem Kapitel werden wir einen solchen Algorithmus, nämlich die Breitensuche, Schritt für Schritt erarbeiten. Die Breitensuche ermöglicht es uns nämlich einen Knoten zu suchen, und, falls er gefunden wurde, seine kürzeste Distanz beziehungsweise auch den konkreten Weg zum Knoten ganz einfach herauszufinden.

Der Algorithmus für die Breitensuche benötigt drei Inputs: einen Graphen, einen Startknoten und einen Zielknoten. In einem ersten Schritt wird der Startknoten betrachtet. Wir haben gesehen, dass vom Startknoten aus direkt all seine Nachbarn erreicht werden können. In Abb. 2.9 sehen Sie den Graphen aus Abb. ???. Wir betrachten Knoten 0 als Startknoten (rot eingefärbt). Alle Nachbarn, die von diesem Startknoten aus erreichbar sind, sind grau eingefärbt.

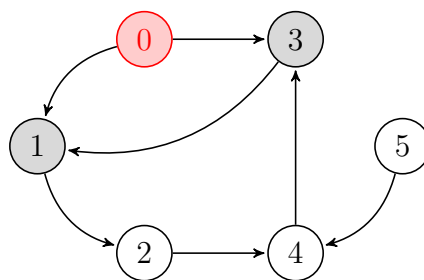


Abbildung 2.9: Startknoten und seine Nachbarn

Wir möchten nun wissen, ob Knoten 2 von diesem Startknoten aus erreichbar ist. Als Erstes prüfen wir also, ob sich der Knoten unter den Nachbarn des Startknotens befindet. Da dies nicht der Fall ist, müssen wir weitersuchen. Da wir mit Distanz 1 den Knoten nicht gefunden haben, müssen wir bei einer grösseren Distanz weitersuchen, also betrachten wir alle Knoten, die mit einer maximalen Distanz von 2 vom Startknoten aus erreichbar sind. Dies sind alle Knoten, die von den Nachbarn des Startknotens aus erreichbar sind. Wir wählen also den ersten Nachbarn, Knoten 1, und betrachten dessen Nachbarn.

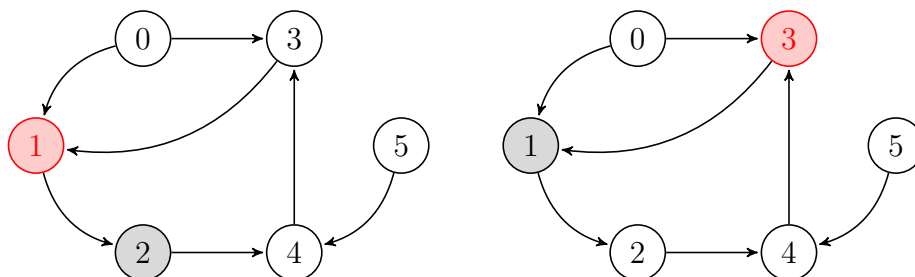


Abbildung 2.10: Aktuelle Knoten und ihre Nachbarn

In Abb. 2.10 sehen wir, dass der Knoten 2 ein Nachbarsknoten von Knoten 1 ist. Nun wählen wir den 2. Nachbarn unseres Startknotens, Knoten 3, und betrachten dessen Nachbarn. Der einzige Nachbarsknoten von Knoten 3 ist Knoten 1, welchen wir aber gerade eben schon betrachtet haben und deshalb nicht mehr betrachten müssen. Wir müssen uns also irgendwie merken, welche Knoten wir schon betrachtet haben, damit wir sie nicht nochmals bearbeiten wenn wir später auf einem anderen Weg nochmals darauf stossen.

Wir erreichen dies, indem wir einen betrachteten Knoten speziell markieren, beispielsweise indem wir ihn schwarz einfärben. Gleichzeitig färben wir Knoten, die wir als Nachbarn eines Knoten schon entdeckt haben, aber noch nicht bearbeiten haben, grau ein. Abb. 2.11 zeigt den Graphen, nachdem der Startknoten (Knoten 0) betrachtet wurde (links), nachdem sein erster Nachbar (Knoten 1) betrachtet wurde (rechts) und nachdem sein zweiter Nachbar (Knoten 3) betrachtet wurde.

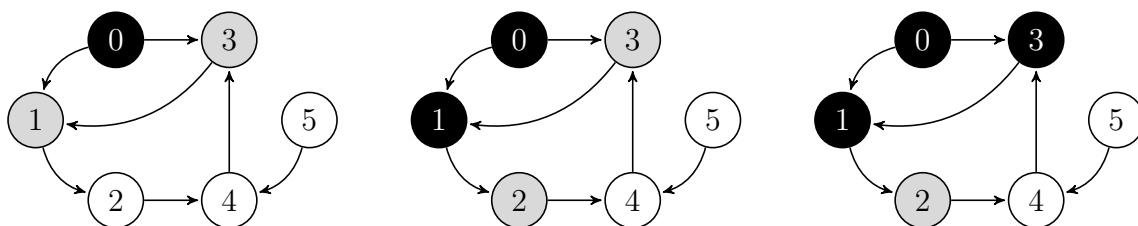


Abbildung 2.11: Verlauf der Einfärbungen

Im nächsten Durchlauf wird der nächste graue Knoten, also Knoten 2 betrachtet. Knoten 2 war der gesuchte Knoten - das heisst wir haben den Knoten gefunden!

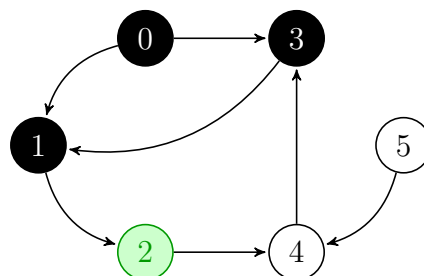


Abbildung 2.12: Zielknoten wurde gefunden

**Zusammenfassung:** Wir möchten nun das Vorgehen zusammenfassen und in einem Algorithmus formulieren.

Wir beginnen mit einem Startknoten und färben ihn sogleich grau ein. Der nächste zu betrachtende Knoten ist jeweils der nächste graue Knoten. Da anfangs nur den Startknoten grau ist, beginnen wir gleich mit diesem Knoten. Falls der Knoten der gesuchte Knoten ist, haben wir ihn gefunden und können das Programm beenden. Andernfalls färben wir alle Nachbarn des Knotens grau ein. Am Schluss färben wir den betrachteten Knoten selbst schwarz ein, um zu markieren, dass wir diesen nun bearbeitet haben. Wir fahren fort mit dem grauen Knoten, der als erstes grau gefärbt wurde und wiederholen das beschriebene Vorgehen.

Es ist wichtig, dass die grauen Knoten in der Reihenfolge betrachtet werden, in der sie eingefärbt werden. Damit stellen wir sicher, dass zuerst alle Knoten von einer kleineren Distanz zum Startknoten bearbeitet werden als die weiter entfernten, da jene auch erst später eingefärbt werden. Man kann sich dies also wie eine Warteschlange vorstellen, in der die grauen Knoten eingereiht werden: die Knoten die zuerst eingereiht wurden, werden auch zuerst wieder von der Warteschlange entfernt.

---

**Algorithm 1** Breitensuche
 

---

```

1: procedure BREITENSUCHE(graph, start, ziel)
2:   start grau einfärben
3:   graueKnoten = [start]                                ▷ Warteschlange für graue Knoten
4:   while graueKnoten ≠ ∅ do
5:     current ← vorderster Knoten aus graueKnoten
6:     if current = ziel then return true                ▷ Zielknoten gefunden
7:     for all nachbar in NACHBARKNOTEN(graph, current) do
8:       if nachbar nicht schwarz oder grau then          ▷ Neu entdeckter Knoten
9:         nachbar grau einfärben
10:      nachbar zuhinterst in graueKnoten einreihen
11:     current schwarz einfärben                            ▷ Knoten fertig bearbeitet
12:     current aus graueKnoten entfernen
13:   return false                                           ▷ Zielknoten nicht gefunden
  
```

---

Wir werden nun Algorithmus 1 Schritt für Schritt implementieren.

**Aufgabe 2.7.** Auf Zeile 8 wird eine Funktion NACHBARKNOTEN für den aktuellen Knoten aufgerufen. Sie haben gelernt, wie Graphen mit Hilfe von Adjazenzmatrizen dargestellt werden können und wie daraus Nachbarn eines Knotens bestimmt werden können. Implementieren Sie nun die Funktion NACHBARKNOTEN(*graph*, *knoten*): sie hat als Input einen Graphen und einen Knoten und gibt als Output eine Liste von Nachbarknoten des Knotens im Graphen aus. Testen Sie die Funktion, indem Sie sie mit verschiedenen Knoten aus einem Graphen aufrufen.

**Aufgabe 2.8.** Benutzen Sie ihre Funktion NACHBARKNOTEN um Algorithmus 1 zu implementieren. Testen Sie Ihr Programm indem sie die Breitensuche für verschiedene Start- und Endknoten aufrufen.

**Hinweis:** Sie können das Einfärben der Knoten so umsetzen, dass sie in einem Array für jeden Knoten seine Farbe speichern. Farben können Sie einfach als Zahlen darstellen (z.B. weiss=0, grau=1, schwarz=2). Für die Umsetzung der Warteschlange der grauen Knoten können Sie auch ein Array verwenden. Mit dem Befehl *pop(x)* kann ein Element an Stelle x im Array ausgelesen und entfernt werden.

## 2.2.3 Anwendung und Erweiterung

Wir werden nun den entwickelten Algorithmus anhand eines konkreten Beispiels betrachten und weiterentwickeln.

**Soziale Netzwerke:** In der heutigen Gesellschaft sind soziale Netzwerke zu einem wichtigen Werkzeug geworden, um Beziehungen zu pflegen, z.B. über Berufsnetzwerke oder

Freundesnetzwerke. Solche Netzwerke können wir als Graphen modellieren. Mit Hilfe von Graphenalgorithmien können gewisse Eigenschaften und Strukturen der Netzwerke erforscht werden.

In Abb. 2.13 sehen Sie einige Personen und ihre Vernetzungen zu anderen Personen. Wir möchten dieses kleine Personennetzwerk etwas genauer untersuchen, z.B. um herauszufinden, ob bestimmte Personen (auch über andere Personen) miteinander vernetzt sind.

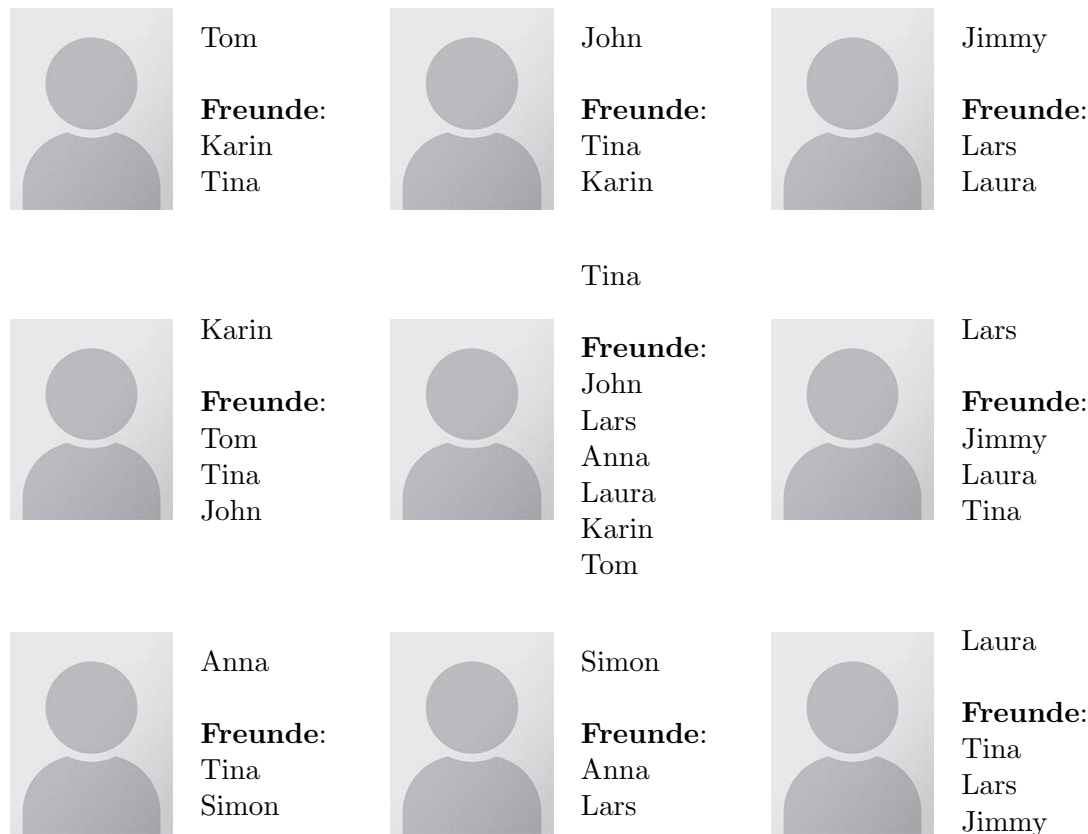


Abbildung 2.13: Ein kleines Personennetzwerk

Dieses Netzwerk kann mit einem Graphen modelliert werden, bei dem die Knoten die Personen repräsentieren und die Kanten die Freundschaftsbeziehungen zwischen den Personen. Den daraus resultierenden Graphen sehen Sie in Abb. 2.14

**Aufgabe 2.9.** Modellieren Sie diesen Graphen als Adjazenzmatrix und lesen Sie ihn in ihr bisheriges Programm ein.

**Aufgabe 2.10.** Testen Sie mit ihrem Programm, ob es irgendeine Verbindung zwischen Anna und John gibt.

**Aufgabe 2.11.** Tina überlegt sich schon seit einiger Zeit, das Netzwerk wieder zu verlassen. Falls sie tatsächlich das Netzwerk verlassen würde, gäbe es dann trotzdem noch eine Verbindung von Anna zu Jimmy? Und von Simon zu Tom?

Wir können mit unserem Programm jetzt also einfach beurteilen, ob es eine Verbindung zwischen zwei Personen gibt. Nun möchten wir aber gerne wissen, über wieviele Personen zwei Personen sozusagen miteinander verbunden sind. Wir müssen also unseren Algorithmus so anpassen, dass wir am Ende sagen können, wieviele Verbindungen eine Person von einer anderen Person mindestens entfernt ist. Beispiel: Wenn Anna mit Tina

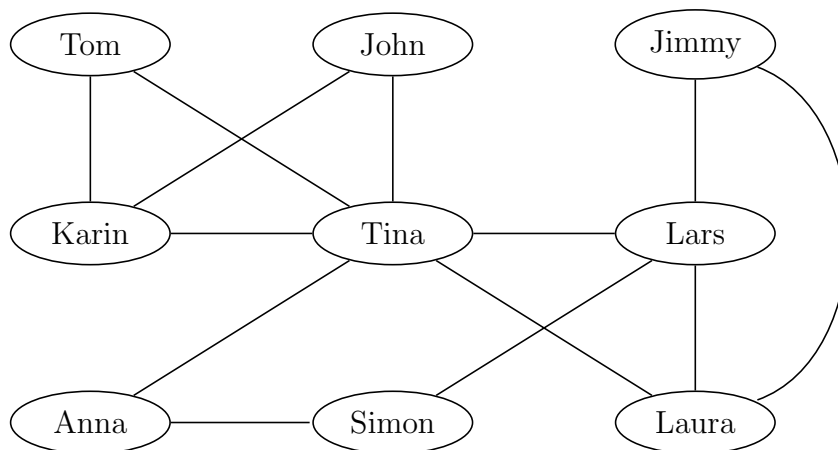


Abbildung 2.14: Modellierung des Personennetzwerks als Graph

befreundet ist und Tina mit Lars, dann ist Lars über zwei Freundschaftsbeziehungen mit Anna verbunden.

**Aufgabe 2.12.** Überlegen Sie sich, wie der Algorithmus 1 angepasst werden könnte, sodass die Distanz zum Zielknoten gespeichert wird. **Hinweis:** Es reicht nicht, die Distanz in einer einzigen Variablen zu speichern. Wieso nicht?

TODO Algorithmus mit Distanz erarbeiten

**Aufgabe 2.13.** Ergänzen Sie Ihr Programm so, dass Sie am Schluss die kürzeste Distanz von einem Startknoten zu einem Zielknoten ausgelesen werden kann, falls der Zielknoten vom Startknoten aus erreichbar ist.

Wir haben nun also ein Programm, das uns sagen kann, wie weit entfernt ein Startknoten von einem Zielknoten mindestens liegt (falls der Zielknoten überhaupt vom Startknoten aus erreichbar ist). Wir möchten unseren Algorithmus nun noch etwas weiter ausbauen, so dass wir am Schluss, falls wir den Zielknoten gefunden haben, herausfinden können, über welche Knoten der kürzeste Weg vom Startknoten zum Zielknoten führt. Wir müssen uns also während dem Traversieren irgendwie merken, wie wir zu einem aktuellen Knoten gelangt sind. Es ist möglich, dass es mehrere Wege von einem Startknoten zu einem Zielknoten gibt, aber wir sind stets am kürzesten Weg interessiert. Das heisst, wir können während der Suche, beim Abarbeiten eines Knotens, einfach speichern, von welchem Knoten aus wir zu diesem Knoten gelangt sind. Dann können wir vom Schlussknoten aus Knoten für Knoten zurückverfolgen, auf welchem Weg wir zum Ziel gelangt sind.

**Beispiel 2.5.** In unserem Netzwerk aus Abb. 2.13 ist Tom mit Karin befreundet und Karin mit John. Tom ist also über Karin mit John verbunden. Ein möglicher Pfad wäre also Tom - Karin - John. Einen kürzeren Pfad gibt es nicht, da Tom nicht direkt mit John befreundet ist. Den Pfad können wir konstruieren, indem wir uns während der Breitensuche merken, dass wir von Tom aus auf Karin gestossen sind und von Karin auf John.

Für unseren Algorithmus bedeutet das, dass wir für jeden Knoten, den wir entdecken, mitspeichern, von welchem Knoten aus wir ihn entdeckt haben. In Algorithmus 2 sehen Sie diese Erweiterung in den rot markierten Zeilen.

TODO: als Argument übergeben?



---

**Algorithm 2** Breitensuche

---

```
1: procedure BREITENSUCHE(graph, start, ziel)
2:   start grau einfärben
3:   graueKnoten  $\leftarrow$  [start] ▷ Warteschlange für graue Knoten
4:   vorgaenger = array[Anzahl Knoten]
5:   while graueKnoten  $\neq$   $\emptyset$  do
6:     current  $\leftarrow$  vorderster Knoten aus graueKnoten
7:     if current = ziel then return true ▷ Zielknoten gefunden
8:     for all nachbar in NACHBARKNOTEN(graph, current) do
9:       if nachbar nicht schwarz oder grau then ▷ Neu entdeckter Knoten
10:        nachbar grau einfärben
11:        nachbar zuhinterst in graueKnoten einreihen
12:        vorgaenger[nachbar] = current ▷ Von wo der Knoten entdeckt wurde
13:        current schwarz einfärben ▷ Knoten fertig bearbeitet
14:        current aus graueKnoten entfernen
15:   return false ▷ Zielknoten nicht gefunden
```

---

**Aufgabe 2.14.** Ergänzen Sie Ihr Programm nun auch so, dass am Schluss der Weg ausgelesen werden kann, der vom Startknoten zum Zielknoten führt, falls dieser vom Startknoten aus erreichbar ist.

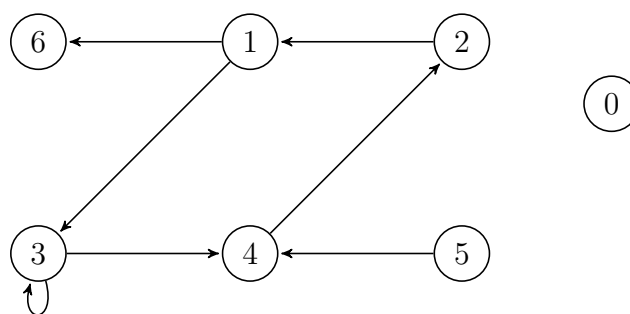
## 2.3 Lösungen

**Lösung zu Aufgabe 2.1.**

$$G = (V, E) \text{ mit: } V = \{0, 1, 2, 3, 4\};$$

$$E = \{(0, 1), (0, 2), (1, 1), (1, 2), (2, 2), (2, 0), (3, 1), (3, 4), (4, 0)\}.$$

**Lösung zu Aufgabe 2.2.**

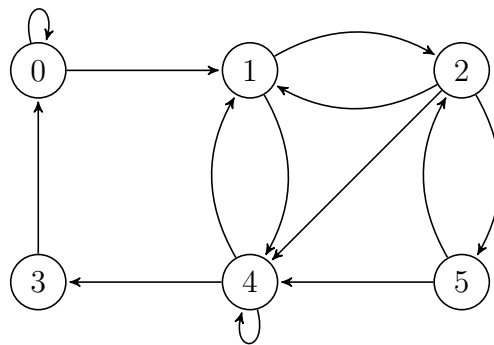


**Lösung zu Aufgabe 2.3.**

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

---

**Lösung zu Aufgabe 2.4.**



**Lösung zu Aufgabe 2.5.**

3 : {1};    1 : {2} und sich selber;    0 : {1, 2}

# Kapitel 3

## Anhang