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

09-931-999

Lehrdiplom Informatik

HS 2017

Inhaltsverzeichnis

Kapitel 1

Konzeption

1.1 Concept Map

Das Thema der vorliegenden Arbeite 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 Graphenalgorithmen 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.

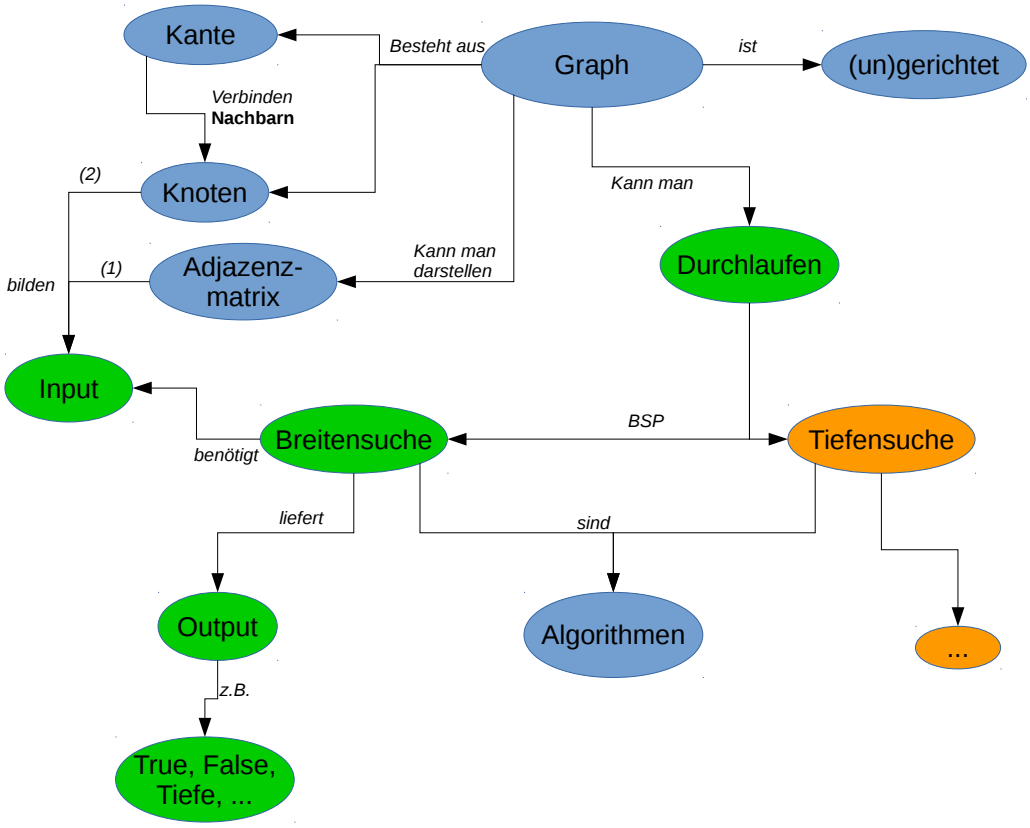


Abbildung 1.1: Concept-Map zum Thema Breitensuche. Blaue Konzepte bilden das Vorwissen ab. Grüne Konzepte werden in dieser Arbeit eingeführt und behandelt. Orange Konzepte bilden einen Ausblick auf Konzepte, welche im Rahmen dieser Arbeit nicht mehr behandelt werden können.

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 verschieden 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

Kapitel 2

Unterlagen

2.1 Graphen

2.1.1 Begriffe

Ein Graph ist eine abstrakte Struktur, die eine Menge von Objekten zusammen mit den zwischen diesen Objekten bestehenden Verbindungen repräsentiert.
Die Objekte werden dabei **Knoten** des Graphen genannt. Die paarweisen Verbindungen zwischen Knoten heissen **Kanten**. Die Kanten können **gerichtet** oder **ungerichtet** sein, wenn die Verbindungen zwischen den Knoten, eine Richtung beinhalten (oder nicht).
Am einfachsten kann man Graphen zeichen, indem Knoten durch Punkte und die Kanten durch Linien (oder Pfeile) darstellt.

Beispiel 2.1. Anschauliche Beispiele für Graphen sind ein Stammbaum oder das S-Bahn-Netz (s. Abb. 2.1) einer Stadt. Bei einem Stammbaum stellt jeder Knoten ein Familienmitglied dar und jede Kante ist eine Verbindung zwischen einem Elternteil und einem Kind. In einem S-Bahn-Netz stellt jeder Knoten eine S-Bahn-Station dar und jede Kante eine direkte Zugverbindung zwischen zwei Stationen.

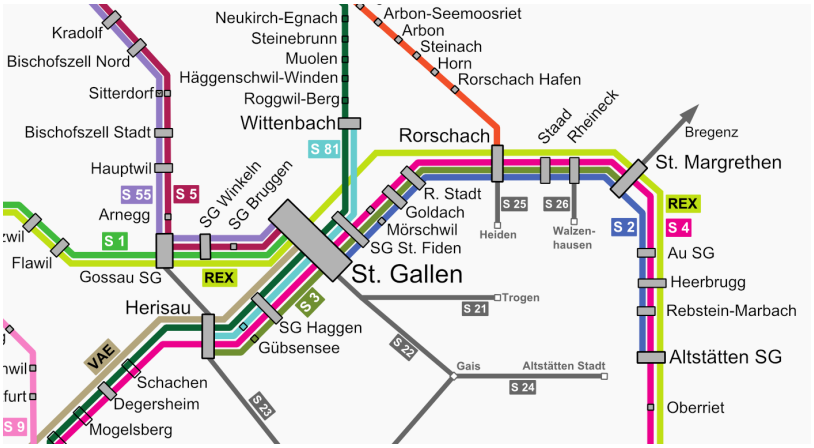


Abbildung 2.1: Darstellung eines Ausschnitts des Sankt Galler S-Bahn-Netzes als (ungerichteter) Graph. Jede Station bildet dabei einen Knoten und die Verbindung zwischen den Knoten ist eine Kante.

2.1.2 Defintionen

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 ein Rolle.

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

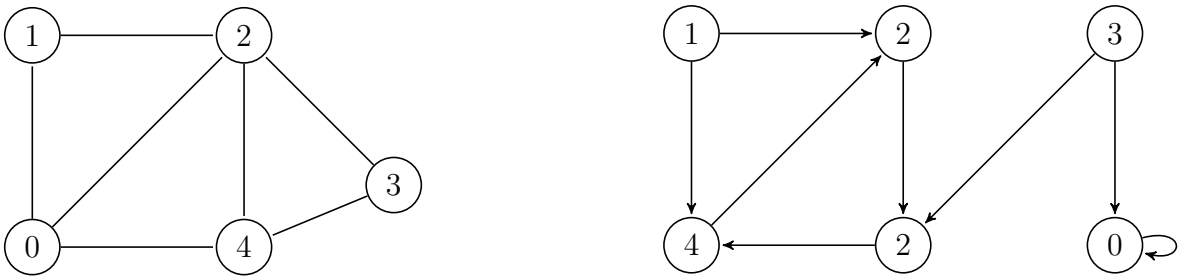


Abbildung 2.2: *Links:* Ein ungerichteter Graph mit 5 Knoten. *Rechts:* Ein gerichteter Graph mit 6 Knoten.

Die mathematische Darstellung der zwei Graphen lautet:

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

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

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:

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

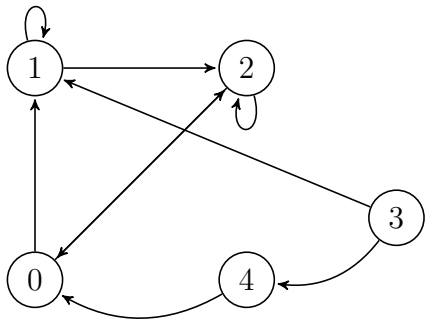


Abbildung 2.3: Ein gerichteter Graph.

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 spielt 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.

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

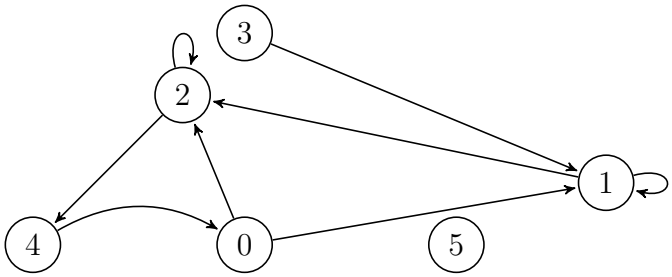


Abbildung 2.4: Ein weiterer Graph.

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 Graphenalgorithmen 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.

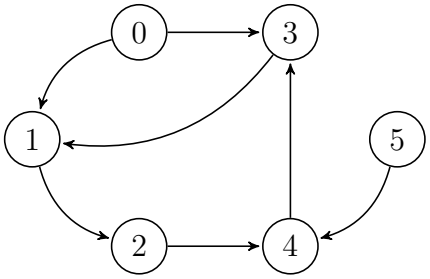


Abbildung 2.5: Ein Beispiels-Graph

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.

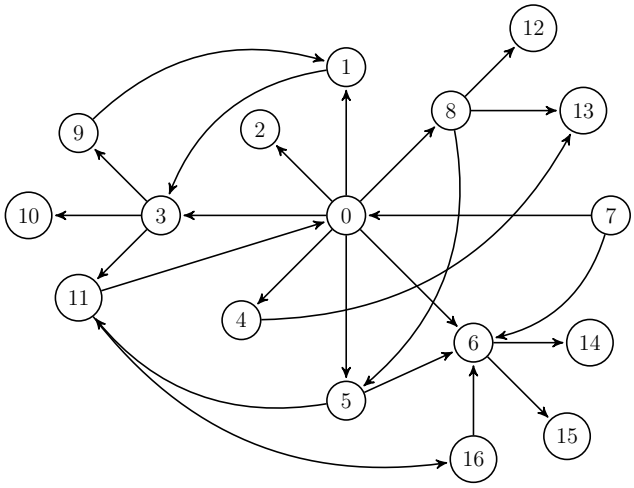


Abbildung 2.6: Ein etwas komplexerer Graph

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, strukturierteres 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.

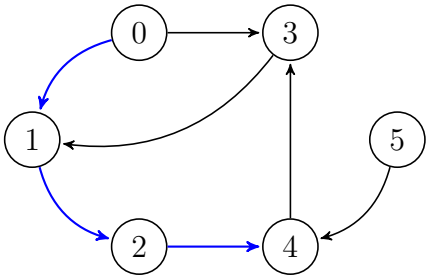


Abbildung 2.7: Weg von Knoten 0 zu Knoten 4 über Knoten 1 und 2

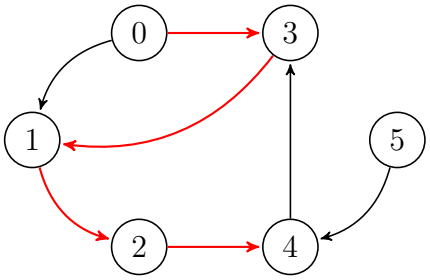


Abbildung 2.8: Weg von Knoten 0 zu Knoten 4 über Knoten 3, 1 und 2

Aufgabe: 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 nähmlich, 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 fï 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 eingefïbt.

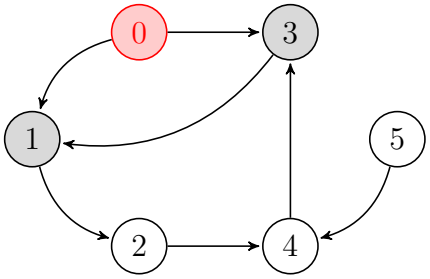


Abbildung 2.9: Startknoten und seine Nachbarn

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 Distanz 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. In Abb. 2.10 sehen wir, 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 schwarz einfärben. Gleichzeitig färben wir Knoten, die wir als Nachbarn eines Knoten gesehen 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.

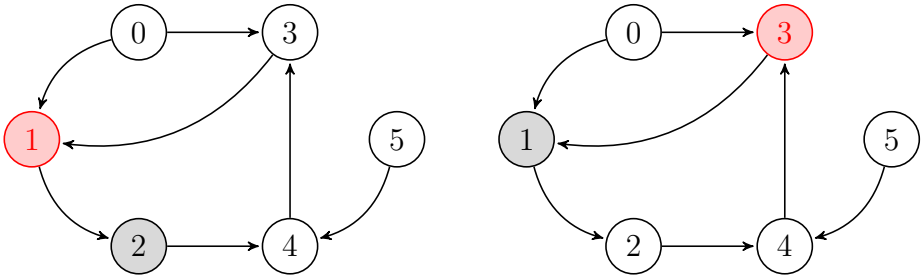


Abbildung 2.10: Aktuelle Knoten und ihre Nachbarn

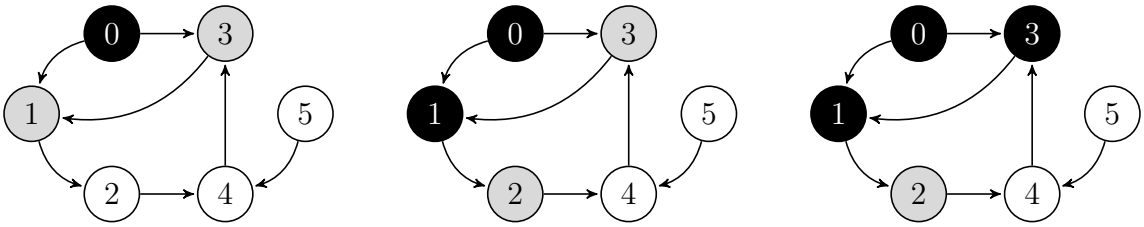


Abbildung 2.11: Verlauf der Einfärbungen

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!

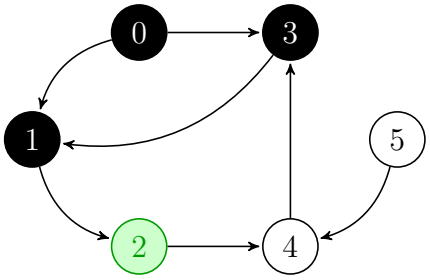


Abbildung 2.12: Zielknoten wurde gefunden

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

säter 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 Warteschlage 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 narchbar in NACHBARKNOTEN(graph, current) do
8:              if nachbar nicht schwarz then   ▷ Falls nachbar noch nicht bearbeitet
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: Auf Zeile 7 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: 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 umsetzten, 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

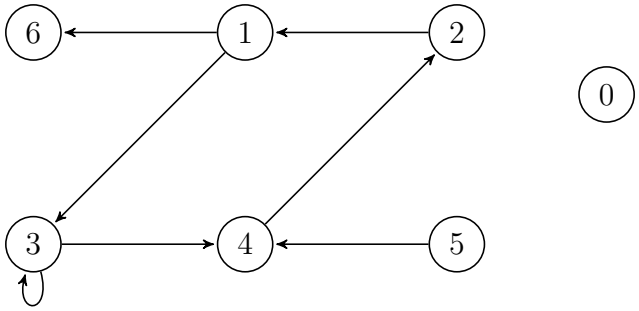
2.3 Lösungen

Lösung zu Aufgabe 2.1.

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

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

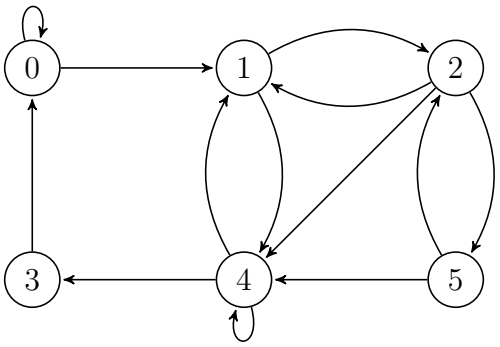
Lösung zu Aufgabe 2.2.



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.

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

Kapitel 3

Anhang