



Fakultät für Mathematik

Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

## **Adaption eines didaktischen Konzepts zur Darstellung weiterführender Graphalgorithmen in einer Web-Applikation**

Algorithmus von Floyd-Warshall

Algorithmus von Hierholzer

Algorithmus von Hopcroft und Karp

Ungarische Methode

Chinese Postman Problem

**Abschlussbericht für ein interdisziplinäres Projekt von  
Mark-Johannes Becker, Aleksejs Voroncovs, Ruslan Zabrodin**

Themensteller: Prof. Dr. Peter Gritzmann

Betreuer: M.Sc. Wolfgang Ferdinand Riedl

Abgabedatum: 16. Mai 2015



Hiermit erkläre ich, dass ich diese Arbeit selbstständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 16. Mai 2015

---

Mark-Johannes Becker, Aleksejs Voroncovs, Ruslan Zabrodin



## **Abstract**

Here we give a short summary of the project or thesis of length at most a quarter of a page.

## **Zusammenfassung**

Hier schreibt man eine kurze Zusammenfassung der Arbeit im Umfang von maximal einer Viertelseite.



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Aufbau der Webanwendungen</b>	<b>3</b>
2.1 Algorithmus von Floyd-Warshall . . . . .	3
2.2 Algorithmus von Hierholzer (Mark) . . . . .	3
2.3 Algorithmus von Hopcroft und Karp . . . . .	4
2.4 Ungarische Methode . . . . .	4
2.5 Chinese Postman Problem . . . . .	5
<b>3 Implementierung</b>	<b>7</b>
3.1 Installation (Mark) . . . . .	7
3.2 MathJAX (Mark) . . . . .	7
3.3 Bipartite Graphen . . . . .	9
3.4 Multigraphen . . . . .	10
3.5 Zufällig generierte Fragen (Mark) . . . . .	10
3.6 Gemeinsam genutzte Dateien (Mark) . . . . .	12
<b>4 Zusammenfassung</b>	<b>15</b>
<b>Abbildungsverzeichnis</b>	<b>17</b>
<b>Index</b>	<b>17</b>



# **Kapitel 1**

## **Einleitung**

Einleitung Motivation Referenz auf führere IDPs (Quellen!) didaktisches Konzept, kurz Aufbau der Dokumentation, inkl. Benennung wer hat was gemacht (Benotung)



# Kapitel 2

## Aufbau der Webanwendungen

funktionale Beschreibung der Apps jedes Tab

### 2.1 Algorithmus von Floyd-Warshall

Da die Implementierung des Floyd-Warshall Algorithmus nicht kompliziert ist, war es wichtig zu zeigen, was genau während dem Ablauf schrittweise passiert. Der Algorithmus wurde in Schritte so unterteilt, dass jeder Schritt eine Verbesserung der Distanz zwischen zwei Knoten darstellt. Um die Verbesserungen anschaulich zu machen, wurde die Adjazenzmatrix implementiert. In jeder Zelle der Adjazenzmatrix wird der aktuelle Distanzwert zwischen zwei entsprechenden Knoten gespeichert. Wenn man die Maus über eine Zelle in der Matrix bewegt, dann wird der Pfad, der diesem Distanzwert entspricht, im Graphen markiert (vgl. Abbildung 2.1).

Die erste Forschungsaufgabe prüft, ob der Nutzer die Logik des Algorithmus verstanden hat. Während des Ablaufs von Floyd-Warshall auf dem bestimmten Graphen werden die Fragen über die vom Algorithmus getroffenen Entscheidungen gestellt.

In der zweiten Forschungsaufgabe soll der Nutzer die fehlende Kantengewichte bestimmen. Ein Teil der Kantenkosten sowie die Adjazenzmatrix sind am Anfang gegeben. Diese Information muss genutzt werden um die fehlende Gewichte zu bestimmen.

### 2.2 Algorithmus von Hierholzer (Mark)

Beim Algorithmus von Hierholzer war es uns wichtig, die resultierende Eulertour verständlich zu visualisieren. Dazu werden zunächst alle Subtouren in unterschiedlichen Farben dargestellt. Einzelne Subtouren können auf der Ergebnisseite einzeln hervorgehoben werden. Der Benutzer erkennt mittels der Farben außerdem, welcher Teil der Tour aus welcher Subtour stammt. Die Tour kann durch eine Animation (vgl. Abbildung 2.2) in der richtigen Reihenfolge abgelaufen werden, sodass der Nutzer verifizieren kann, dass es sich wirklich um eine Eulertour handelt.

Die erste Forschungsaufgabe soll sicherstellen, dass der Nutzer die wichtigsten Abläufe des Algorithmus versteht. Sie behandelt daher im Wesentlichen den Ablauf des Algorithmus, das Konzept des Knotengrads und das Verbinden von Touren.

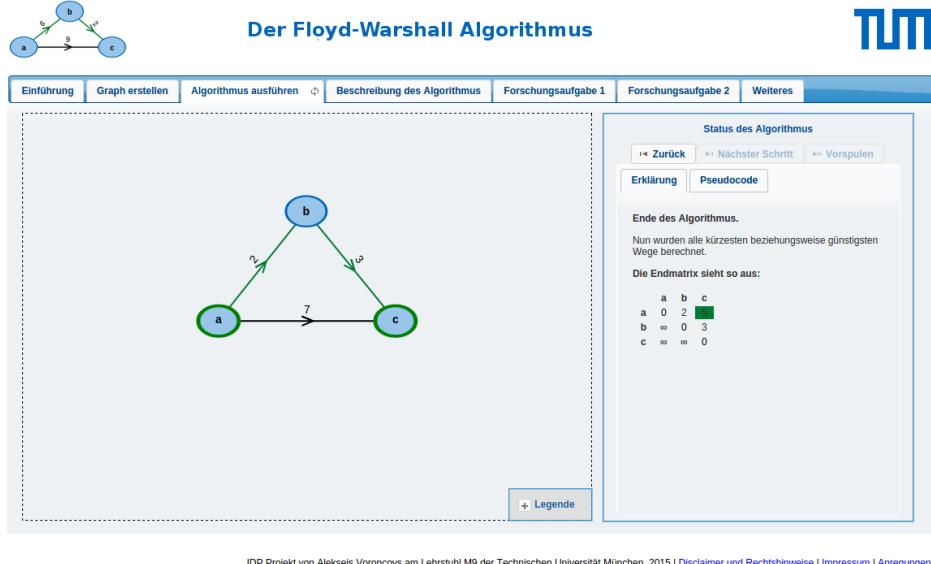


Abbildung 2.1: Visualisierung des verbesserten Pfades

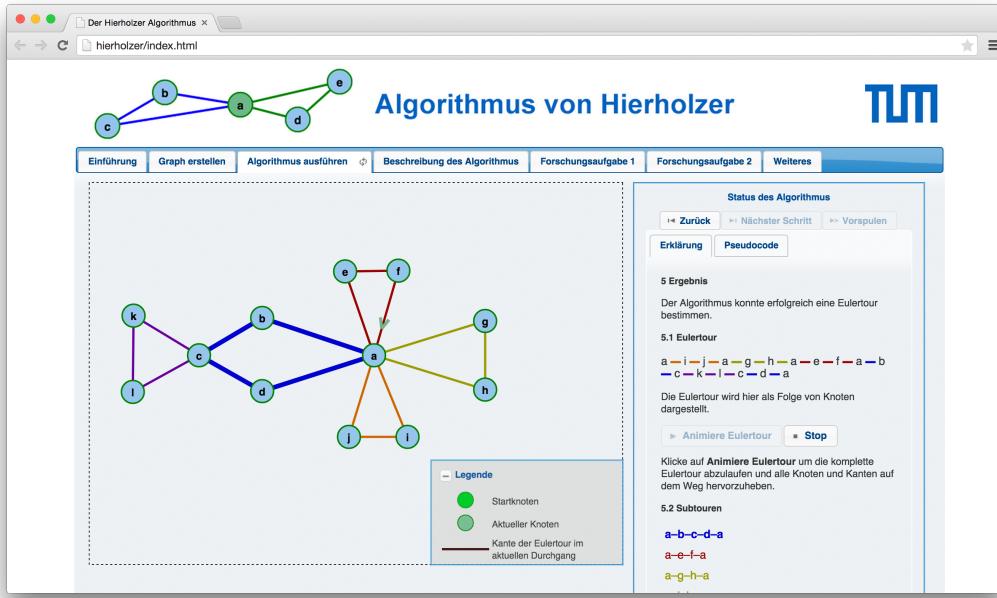
In einer weiteren Forschungsaufgabe werden die Besonderheiten des Algorithmus bei Anwendung auf einen gerichteten Graphen betrachtet. Der Nutzer betrachtet die geänderten Voraussetzungen, die der Graph erfüllen muss und die veränderte Auswahl der Kanten zur Konstruktion von Subtouren.

## 2.3 Algorithmus von Hopcroft und Karp

Besonderheiten Visualisierung Forschungsfragen

## 2.4 Ungarische Methode

Obwohl es mehrere Implementierungen der Ungarischen Methode gibt, die Idee bleibt bei allen gleich. Die Applikation geht nicht tief in die Einzelheiten der konkreten Implementierung, sondern erklärt das generelle Prinzip der Ungarischen Methode. Dafür war der Einsatz des bipartiten gewichteten Graphs notwendig. Als die Ungarische Methode perfektes Matching im bipartiten Graphen sucht, wird eingegebener Graph in der Anwendung bis zum kompletten Graphen vervollständigt falls der Nutzer das nicht gemacht hat. Um die genutzte Begriffe (wie z. B. Matching oder Augmentationsweg) verständlich zu machen, werden unterschiedliche Farben für Knoten und Kanten während des Ablaufs verwendet (vgl. Abbildung 2.3).



**Abbildung 2.2:** Visualisierung der Eulertour mittels Animation

Die erste Forschungsaufgabe prüft, wie gut der Nutzer den Aufbau des Algorithmus verstanden hat. Dazu werden die Fragen während des Ablaufs auf dem beliebigen Graphen gestellt. Der Nutzer muss beantworten, wie der Algorithmus die Markierungen zuweist und welche Schritte macht der Algorithmus als weitere.

Bei der zweiten Forschungsaufgabe werden auch die Fragen während des Ablaufs der Ungarischen Methode gestellt. Der Graph ist in diesem Fall bestimmt und die Fragen fassen den Aufbau des Augmentationsweges und Gleichheitsgraphs um.

## 2.5 Chinese Postman Problem

Besonderheiten Visualisierung Forschungsfragen

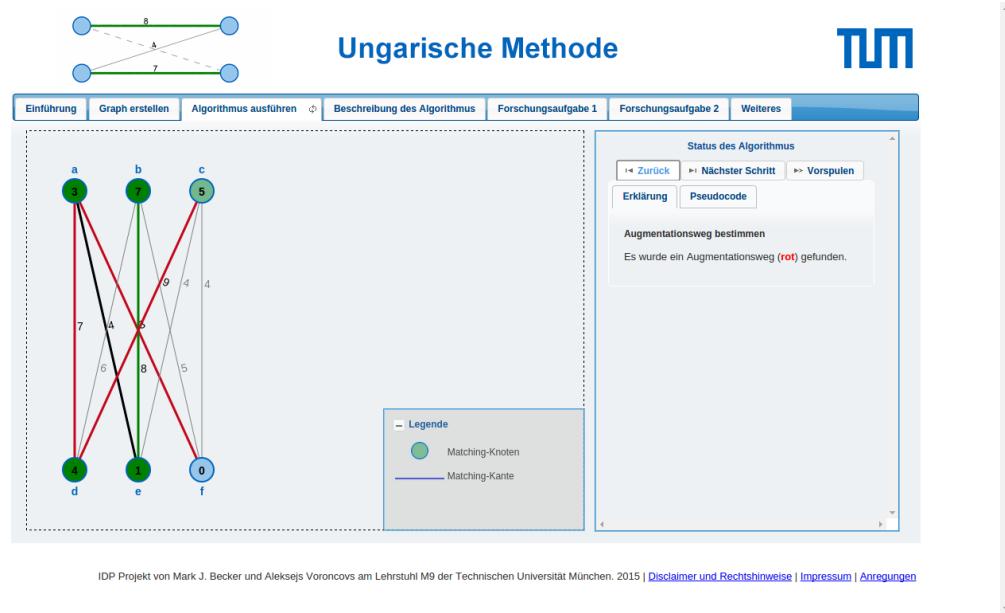


Abbildung 2.3: Augmentationsweg wird rot markiert

# Kapitel 3

## Implementierung

### 3.1 Installation (Mark)

Zur Installation sind keine speziellen Anforderungen zu erfüllen. Die Webapplikationen wurden vollständig mittels HTML5, JavaScript und CSS implementiert, sodass keine zusätzliche serverseitige Software benötigt wird. Zum Aufrufen der Anwendungen wird ein moderner Webbrowser benötigt.

Die Bereitstellung erfolgt über den Online Dienst GitHub, der das verteilte Versionskontrollsystem Git benutzt. Alle Anwendungen liegen in einem gemeinsamen Repository, welches unter <https://github.com/herzog31/adv-graph-algorithms> erreichbar ist. Zur Installation kann man entweder auf der Repository Seite die letzte freigegebene Version (Release) als Zip oder Tar Archiv herunterladen oder die aktuellste Version des Repositories mittels folgendem Befehl in das aktuelle Verzeichnis kopieren.

```
1 git clone -b master https://github.com/herzog31/adv-graph-algorithms.git
```

**Abbildung 3.1:** Befehl zum Kopieren des GitHub Repositories

Um die Anwendungen als Webanwendungen online bereitzustellen wird ein Webserver benötigt. Hier empfiehlt sich die Installation des Apache oder nginx HTTP Servers.

Die lokale Ausführung ist ohne einen Webserver möglich. Verschiedene Webbrowers besitzen allerdings eine Sicherheitsrichtlinie, die das Öffnen von lokalen Dateien über JavaScript verbieten. Diese Sicherheitsrichtlinie kann jedoch durch spezielle Einstellungen umgangen werden. Für Google Chrome ist dies über den Start Parameter `--allow-file-access-from-files` möglich.

### 3.2 MathJAX (Mark)

Unter den Tabs „Beschreibung des Algorithmus“ werden die komplizierten Algorithmen, wie bspw. die Ungarische Methode, in möglichst einfachen Worten als Fließtext erklärt. Wir gehen davon aus, dass der Nutzer während der Bearbeitung der Forschungsaufgaben,

verschiedene Sachverhalte in der Algorithmenbeschreibung nachschlagen muss. Das wiederholte Lesen von vollständigen Absätzen wollten wir allerdings vermeiden.

Dazu haben wir zusätzlich zur textuellen Beschreibung, wichtige Punkte der Algorithmen auch als mathematische Formeln dargestellt. Der Nutzer erlangt so durch das erste Lesen der Beschreibung ein Grundverständnis der Algorithmen und kann während der Bearbeitung aufkommende Fragen durch die dargestellten Formeln schneller erschließen.

Zur Darstellung der Formeln in den Webapplikationen verwenden wir JavaScript Bibliothek MathJAX<sup>1</sup>. Diese ist frei unter der Apache-Lizenz erhältlich und wird u.a. auch von Wikipedia zur Darstellung jeglicher mathematischer Ausdrücke genutzt. Mit der Bibliothek ist es möglich mit LaTeX beschriebene Ausdrücke als SVG oder PNG im Webbrower zu rendern.

Damit die Bibliothek in den Webanwendungen genutzt werden kann, muss sie wie folgt eingebunden werden (vgl. Abbildung 3.2).

```
1 <script type="text/javascript" src="../library/js/mathjax/~/  
    ↪ MathJax.js?config=TeX-AMS-MML_SVG.js&locale=de"></script>  
2 <script type="text/x-mathjax-config">  
3   MathJax.Hub.Config({  
4     showMathMenu: false,  
5     showMathMenuMSIE: false  
6   });  
7 </script>
```

**Abbildung 3.2:** Einbinden der MathJAX Bibliothek

Nach der Einbindung werden sämtliche LaTeX Ausdrücke welche sich zwischen den Begrenzungszeichen \(\) und \(\) befinden, automatisch übersetzt. In einem Beispiel wird hier der HTML Code in Abbildung 3.3 von MathJAX im Browser gerendert (vgl. Abbildung 3.4).

```
1 <p style="text-align:center;">  
2   \(\Delta = \min\limits_{s \in S} \wedge_{y \in Y} \setminus T \cap  
    ↪ \{l(s) + l(y) - w(s,y)\}\)  
3 </p>
```

**Abbildung 3.3:** MathJAX Beispiel: HTML Code

Werden Ausdrücke zur Laufzeit in das DOM mittels JavaScript eingefügt, so müssen diese separat übersetzt werden (vgl. Abbildung 3.5).

---

<sup>1</sup><https://www.mathjax.org/>

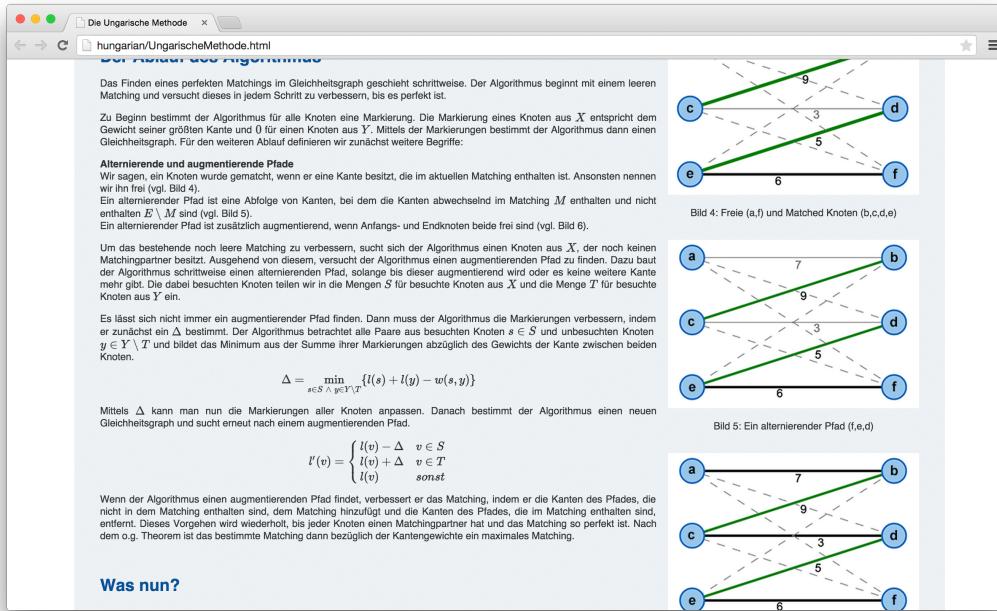


Abbildung 3.4: MathJAX Beispiel: Darstellung im Browser

```
1 MathJax.Hub.Queue([ "Typeset" , MathJax.Hub ]);
```

Abbildung 3.5: Befehl zum erneuten Rendern von Ausdrücken im HTML Dokument

### 3.3 Bipartite Graphen

Zwei Algorithmen, die im Rahmen des Projekts implementiert wurden, nämlich Hopcroft & Karp Algorithmus und die Ungarische Methode, müssen auf bipartiten Graphen ausgeführt werden. Dafür haben wir das Konzept des bipartiten Graphen im existierenden Framework entwickelt. Um die Darstellung des Matching-Problems und dessen Lösung möglichst anschaulich zu machen, haben wir die Knoten im Graphen in zwei Gruppen geteilt (vgl. Abbildung 3.6). Die Kanten können nur zwischen den Knotengruppen erstellt werden. Bei der Ungarischen Methode war es wichtig den Graph mit extra Knoten und Kanten zu vervollständigen, wenn der ursprüngliche Graph nicht komplett war. Das ist notwendig, weil die Ungarische Methode perfektes Matching sucht.

In der Abbildung 3.7 kann man sehen, was passiert, wenn man bei der Ungarischen Methode einen nicht kompletten Graphen eingibt. Falls eine Kante zwischen beliebigen

The screenshot shows a software interface titled "Ungarische Methode" (Hungarian Method) with the TUM logo in the top right. The main window has tabs: Einführung, Graph erstellen, Algorithmus ausführen, Beschreibung des Algorithmus, Forschungsaufgabe 1, Forschungsaufgabe 2, and Weiteres. The "Graph erstellen" tab is active. On the left, there's a bipartite graph with two sets of nodes. The left set has nodes labeled 1, 2, 3, 4, 5, 6, 7, 8, 9. The right set has nodes labeled 1, 2, 3, 4, 5, 6, 7. Edges connect nodes from the left to the right: (1,1), (1,2), (1,3), (2,2), (2,3), (2,4), (3,3), (3,4), (3,5), (4,4), (4,5), (4,6), (5,5), (5,6), (5,7), (6,6), (6,7), (7,7). Some edges have weights: (1,1)=7, (1,2)=6, (1,3)=4, (2,2)=8, (2,3)=7, (2,4)=9, (3,3)=4, (3,4)=5, (3,5)=6, (4,4)=5, (4,5)=4, (4,6)=3, (5,5)=3, (5,6)=2, (5,7)=1, (6,6)=1, (6,7)=0. A legend indicates that blue circles represent "Knoten" (nodes) and black lines represent "Kante" (edges). On the right, a sidebar provides instructions for creating partitions and edges. It also includes a "Selberstellter Graph" dropdown and a "Fertig – weiter zum Algorithmus!" button.

Abbildung 3.6: Bipartiter Graph

zwei Knoten fehlt, wird eine zusätzliche graue Kante mit dem Gewicht 0 hinzugefügt. Wenn die Knotengruppen ungleiche Anzahl der Knoten enthalten, dann werden zusätzliche Knoten dem Graphen hinzugefügt. Alle Nachbarkanten bei diesen zusätzlichen Knoten haben das Gewicht 0.

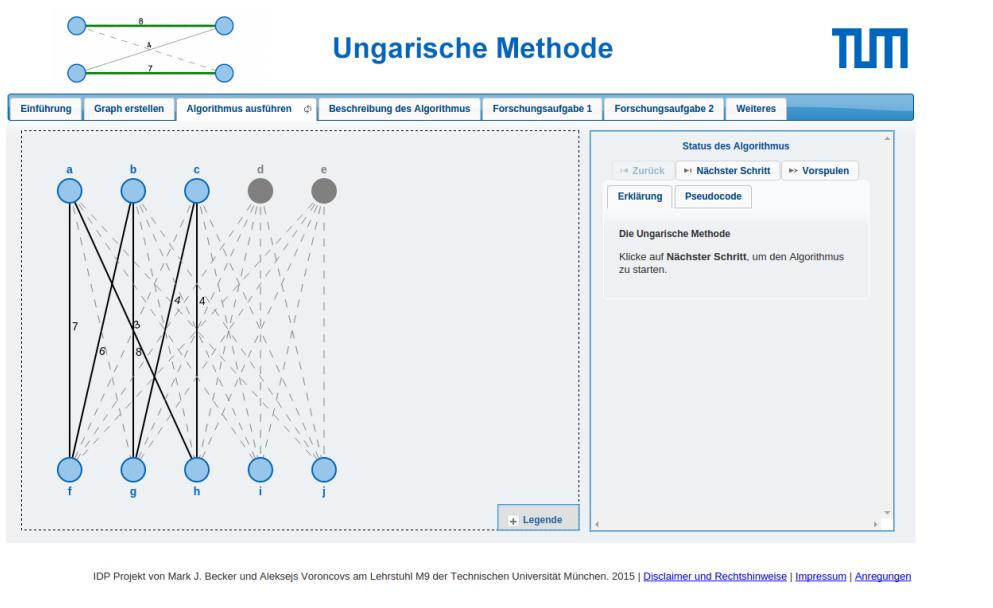
### 3.4 Multigraphen

Motivation Beispiel

### 3.5 Zufällig generierte Fragen (Mark)

In bisherigen auf diesem Framework basierenden Projekten war die Varianz der Fragen in den Forschungsaufgaben problematisch. Die Forschungsaufgaben behandelten meistens statische Fragen an vorher festgelegten Stellen zu vorgegebenen Graphen. Daraus resultierend bietet sich dem Nutzer ein wenig abwechslungsreiches Erlebnis und die Motivation, eine Forschungsaufgabe mehrfach zu bearbeiten, ist nicht gegeben.

In unserer Implementierung verfolgen wir daher einen anderen Ansatz. Die Forschungsaufgaben sind grundsätzlich so aufgebaut, dass sie die gleiche Struktur und den gleichen Quellcode wie die reguläre Ausführung des Algorithmus benutzen. Das schließt auch den vom Nutzer selbst gewählten Graph aus dem „Graph erstellen“ Tab mit ein.



**Abbildung 3.7:** Der Graph wird mit fehlenden Knoten und Kanten erweitert

Statt statischen Fragen definiert der Entwickler eine Menge von Fragetypen. Vor jedem Schritt des Algorithmus wird dann anhand einer Wahrscheinlichkeitsmatrix bestimmt, ob zu dem folgenden Schritt eine Frage gestellt wird und wenn ja, welcher Fragetyp gewählt werden soll. Außerdem sollen wenn möglich die Inhalte der generierten Fragen eines Fragetyps variieren. So werden beispielsweise die Knoten, zu denen ein Wert bestimmt werden soll, zufällig gewählt (vgl. Abbildung 3.8).

Mit dieser Implementierung gelingt es, dass in jeder Bearbeitung einer Forschungsaufgabe unterschiedliche Kombinationen von Fragen gestellt werden. Zusätzlich kann der Nutzer durch selbst erstelle Graphen auch Spezialfälle testen.

## Implementierung

In der Funktion `this.nextStepChoice` wird vor der Auswahl des nächsten Schritts des Algorithmus mittels der Funktion `this.askQuestion` bestimmt, ob und welche Frage gestellt werden soll. Für jeden Fragetyp exisiert eine separate Funktion (bspw. `this.generateNextStepQuestion`), die auf den aktuellen Graph zugreift und eine Frage mit zufälligen Elementen generiert und in das DOM Element `#tf1_div_questionModal` schreibt. Die gegebene Antwort wird durch Aufruf der Funktion `this.saveAnswer` gespeichert und mit der richtigen Lösung verglichen. Am Ende der Forschungsaufgabe kann mittels `this.showQuestionResults` eine Übersicht angezeigt werden, die zeigt, wieviele Fragen richtig beantwortet wurden. Weitere Informationen sind der

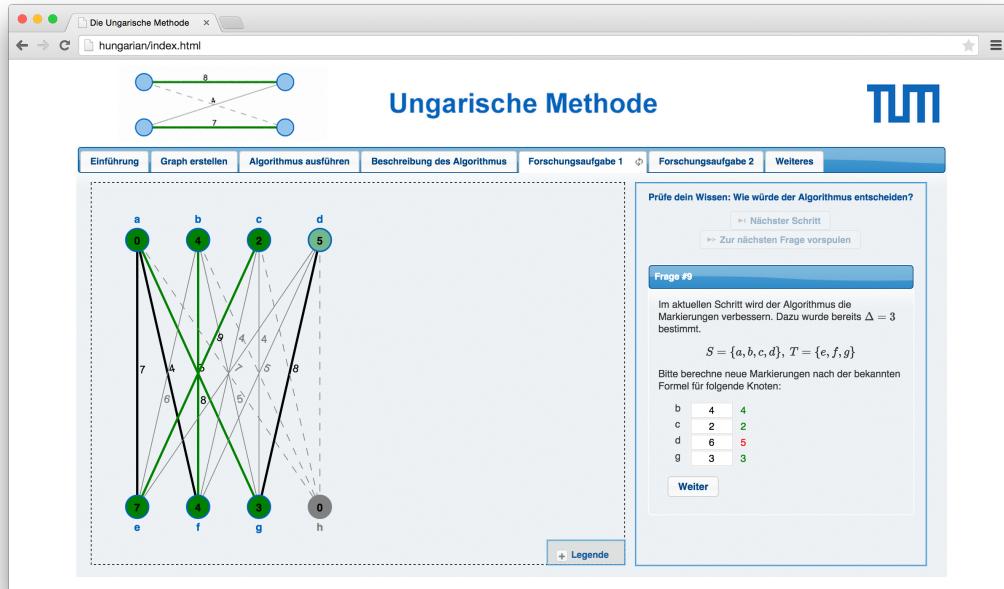


Abbildung 3.8: Beispiel für eine zufällig generierte Frage

Inline-Dokumentation zu entnehmen.

### 3.6 Gemeinsam genutzte Dateien (Mark)

Im Rahmen dieses interdisziplinären Projekts wurden Webapplikationen für insgesamt fünf Algorithmen entwickelt. Jede Applikation liegt in einem von den anderen Algorithmen unabhängigen Projektordner, der jeweils auch das Graph Framework aus früheren Projekten enthält. Daher lagen im gesamten Repository viele Dateien mehrfach vor. Dies war insofern problematisch, da Änderungen, die alle Anwendungen betreffen in mehreren Dateien vorgenommen werden mussten.

Unser Ziel war es daher, gemeinsam genutzte Dateien auszulagern. Wir beschränken uns hier auf Dateien, welche das Layout und Design der Anwendungen definieren sowie universelle Programmbibliotheken (bspw. MathJAX). Diese Beschränkung ist notwendig, um die Flexibilität im Entwicklungsprozess zu gewährleisten. Es existieren weitere Dateien, die sich zum aktuellen Zeitpunkt anwendungsübergreifend nicht signifikant unterscheiden. Diese sind allerdings algorithmenspezifisch und könnten in zukünftigen Entwicklungsphasen weiter verändert werden.

Zur Auslagerung legten wir zunächst einen `library` Ordner an. Das Finden von Dateiduplikaten übernimmt ein Python Skript (vgl. `deduplicate.py`), welches mittels

### *3.6 Gemeinsam genutzte Dateien (Mark)*

---

der **difflib** Bibliothek alle Dateien in den Projektordnern paarweise miteinander vergleicht und einen Ähnlichkeitsquotienten bestimmt. Das Skript zeigt dann alle Dateien, die einen festgelegten Ähnlichkeitsgrenzwert überschreiten an (vgl. Abbildungen 3.9 und 3.10). Anhand des Quotienten lässt sich ablesen, ob Dateien im Gesamten oder in Teilen ausgelagert werden können. Das Verschieben in den **library** Ordner und die Anpassung der Pfade in den Anwendungen erfolgt dann manuell.

```
chinese-postman/img/TUMLogo.png
-> 100.00% identical to floyd-warshall/img/TUMLogo.png
-> 100.00% identical to hierholzer/img/TUMLogo.png
-> 100.00% identical to hopcroft-karp/img/TUMLogo.png
-> 100.00% identical to hungarian/img/TUMLogo.png
```

**Abbildung 3.9:** Vollständige Übereinstimmung bei der Datei `TUMLogo.png`

```
chinese-postman/js/siteAnimation.js
-> 67.01% identical to hierholzer/js/siteAnimation.js
-> 91.49% identical to hopcroft-karp/js/siteAnimation.js
-> 71.13% identical to hungarian/js/siteAnimation.js
```

**Abbildung 3.10:** Mäßige Übereinstimmungen bei der Datei `siteAnimation.js`

Für zukünftige Projekte können die Parameter des Skripts angepasst werden. Weitere Informationen hierzu sind der Inline-Dokumentation zu entnehmen.



# **Kapitel 4**

## **Zusammenfassung**

Zusammenfassung der wesentlichen Punkte



# Abbildungsverzeichnis

2.1	Floyd-Warshall Matrix . . . . .	4
2.2	Eulertour Animation . . . . .	5
2.3	Ungarische Methode . . . . .	6
3.1	Repository Kopieren . . . . .	7
3.2	MathJAX Einbindung . . . . .	8
3.3	MathJAX Beispiel Code . . . . .	8
3.4	MathJAX Beispiel Browser . . . . .	9
3.5	MathJAX Render Befehl . . . . .	9
3.6	Ungarische Methode . . . . .	10
3.7	Ungarische Methode . . . . .	11
3.8	Zufällig generierte Frage . . . . .	12
3.9	Gemeinsame Dateien, Beispiel 1 . . . . .	13
3.10	Gemeinsame Dateien, Beispiel 2 . . . . .	13

