

Technische Universität Darmstadt

Fachbereich Informatik

Rechnerbetriebsgruppe



Studienarbeit

Generator für Graph Algorithmen Animationen Animations-Tutorial

Autor:

Simon Kulesa
S.Kulesa@gmx.de

Betreuer:

Dr. Guido Rößling

Aktueller Stand:
GFGAA Version:
Status:

10-04-2007
0.97c
In Arbeit (**Prerelease**)

1.	Allgemeine Informationen	4
2.	Allgemeines Schema der Animationsgeneratoren	5
2.1	Voraussetzungen	6
2.2	Animations-Komponenten	6
2.2.1	Intro Beschreibung	6
2.2.2	Pseudo-Code	6
2.2.3	Tabellen Animation	7
2.2.4	Zusatz Ausgaben	7
2.3	Farbeinstellungen	7
2.4	Animation ausrichten	8
2.5	Erweiterte Einstellungen	8
3.	Algorithmus zur Tiefensuche	9
3.1	Voraussetzungen	10
3.2	Animations-Komponenten	10
3.3	Farbeinstellungen	10
3.4	Animation ausrichten	10
3.5	Erweiterte Einstellungen	11
3.6	Screenshots einer Beispielanimation	12
4.	Algorithmus zur Breitensuche	13
4.1	Voraussetzungen	14
4.2	Animations-Komponenten	14
4.3	Farbeinstellungen	14
4.4	Animation ausrichten	14
4.5	Erweiterte Einstellungen	14
4.6	Screenshots einer Beispielanimation	15
5.	Färbungsproblem – Ein NP-vollständiges Problem	16
5.1	Voraussetzungen	17
5.2	Animations-Komponenten	17
5.3	Farbeinstellungen	17
5.4	Animation ausrichten	18
5.5	Erweiterte Einstellungen	18
5.6	Screenshots einer Beispielanimation	19
6.	Kruskal – Berechnung des minimalen Spannbaumes	20
6.1	Voraussetzungen	21
6.2	Animations-Komponenten	21
6.3	Farbeinstellungen	21
6.4	Animation ausrichten	21
6.5	Erweiterte Einstellungen	22
6.6	Screenshots einer Beispielanimation	23

7.	Dijkstra – Lösen von kürzesten Wege Problemen I	24
7.1	Voraussetzungen	25
7.2	Animations-Komponenten	25
7.3	Farbeinstellungen	25
7.4	Animation ausrichten	25
7.5	Erweiterte Einstellungen	26
7.6	Screenshots einer Beispielanimation	27
8.	Bidirektionaler Dijkstra – Kürzeste Wege Probleme II	28
8.1	Voraussetzungen	29
8.2	Animations-Komponenten	29
8.3	Farbeinstellungen	29
8.4	Animation ausrichten	29
8.5	Erweiterte Einstellungen	30
8.6	Screenshots einer Beispielanimation	31
9.	Erkennen von Zyklen negativer Länge	32
9.1	Voraussetzungen	33
9.2	Animations-Komponenten	33
9.3	Farbeinstellungen	33
9.4	Animation ausrichten	33
9.5	Erweiterte Einstellungen	34
9.6	Screenshots einer Beispielanimation	35
10.	PreFlowPush – Lösen von Flussproblemen	36
10.1	Voraussetzungen	37
10.2	Animations-Komponenten	37
10.3	Farbeinstellungen	37
10.4	Animation ausrichten	37
10.5	Erweiterte Einstellungen	38
10.6	Screenshots einer Beispielanimation	39

1. Allgemeine Informationen

Dieses Tutorial soll die Erzeugung von Animation mit Hilfe des „Generator für Graphen Algorithmen Animationen (GFGAA)“ erklären. Informationen über die generelle Bedienung der GUI finden Sie im Benutzer-Tutorial. Falls Sie eigene Graphenalgorithmus-Generatoren implementieren wollen, lesen Sie bitte das Erweiterungs-Tutorial.

Bisher verfügt der Generator für Graphen Algorithmen Animationen über acht Generatoren. Die Animationen befassen sich mit folgenden Problemstellungen:

- Kapitel 3: Tiefensuche
- Kapitel 4: Breitensuche
- Kapitel 5: Färbungsproblem
- Kapitel 6: Kruskal
- Kapitel 7: Dijkstra
- Kapitel 8: Dijkstra (bidirektional)
- Kapitel 9: Erkennen negativer Zyklen
- Kapitel 10: PreFlow Push

Weitere Graphenalgorithmus-Generatoren sind geplant.

Kapitel 2 beschreibt den allgemeinen Aufbau der Graphenalgorithmus-Generatoren. Die nachfolgenden Kapitel bauen auf diesem Kapitel auf.

In diesen Kapiteln finden Sie jeweils Details zu den Voraussetzungen, die ein Graph erfüllen muss, um mit dem entsprechenden Graphenalgorithmus-Generator kompatibel zu sein. Weiterhin finden Sie Details zu den einzelnen Komponenten der Animationen und Einstellungen der Generatoren. Zu guter Letzt finden sie noch Screenshots einzelner Beispiel-Animationen, die Ihnen eine ungefähre Vorstellung vom Ablauf der Animationen geben sollen.

Im Folgenden wird davon ausgegangen, dass GFGAA bereits mittels „`java -jar gfgaa-xx-xx-xx.jar`“ (xx-xx-xx repräsentiert steht dabei für Tag-Monat-Jahr des Release) gestartet wurde. Zusätzlich muss bereits ein Graph erstellt worden sein. Sollte der Generator bestimmte Anforderungen an dem Graph stellen, werden diese jeweils im Abstand „Voraussetzungen“ beschrieben.

!!! Wichtiger Hinweis zum Einladen der Animationen in ANIMAL !!!

Falls Sie eine ältere Version von Animal (älter als 2.3.1) verwenden, kann es vorkommen, dass die erzeugte Animation die Kapazität von ANIMAL sprengt. Bei Graphen mittlerer Größe ist je nach Animation mit einer Ladezeit zwischen 5 Sekunden und 5 Minuten zu rechnen. Während der Ladezeit beträgt die Prozessor Auslastung stets 100%.

An längsten dauert das Einladen von Animationen des PreFlowPush-Algorithmus. Hier kann es auch bei relativ kleinen Graphen zu Ladezeiten bis zu 5 Minuten kommen.

Sollte ANIMAL nach 10 Minuten immer noch keine Reaktion gezeigt haben, ist es wahrscheinlich, dass ANIMAL diese Animation nicht in „endlicher Zeit“ laden kann. In diesem Fall empfiehlt es sich (nachdem man die ANIMAL Anwendung zwangsterminiert hat) die Zusatzausgabenkomponente der Animation abzuschalten. Dadurch wird in der Regel die Anzahl der Animationsschritte um mehr als die Hälfte verkürzt.

2. Allgemeines Schema der Animationsgeneratoren

Dieses Kapitel beschreibt den Aufbau der Oberfläche eines Graphenalgorithmus-Generators.

Über **(K)** können Sie unter den zur Verfügung stehenden Graphenalgorithmus-Generatoren auswählen. Der Name des ausgewählten Graphenalgorithmus-Generators und seine dazugehörige Oberfläche werden angezeigt.

(A) repräsentiert den Status der Animationskomponente an. Rot steht für nicht erzeugt, Grün für erzeugt und Blau für möglicherweise nicht mehr aktuell.

In dem Textfenster **(B)** finden Sie Informationen über den Algorithmus. In der ersten Zeile werden nochmals der Name des Algorithmus, der Autor und die Versionsnummer des dazugehörigen Algorithmengenerators aufgeführt. Darunter finden Sie Angaben zu den Voraussetzungen und den unterstützten Komponenten, sowie eine Beschreibung des Algorithmus.

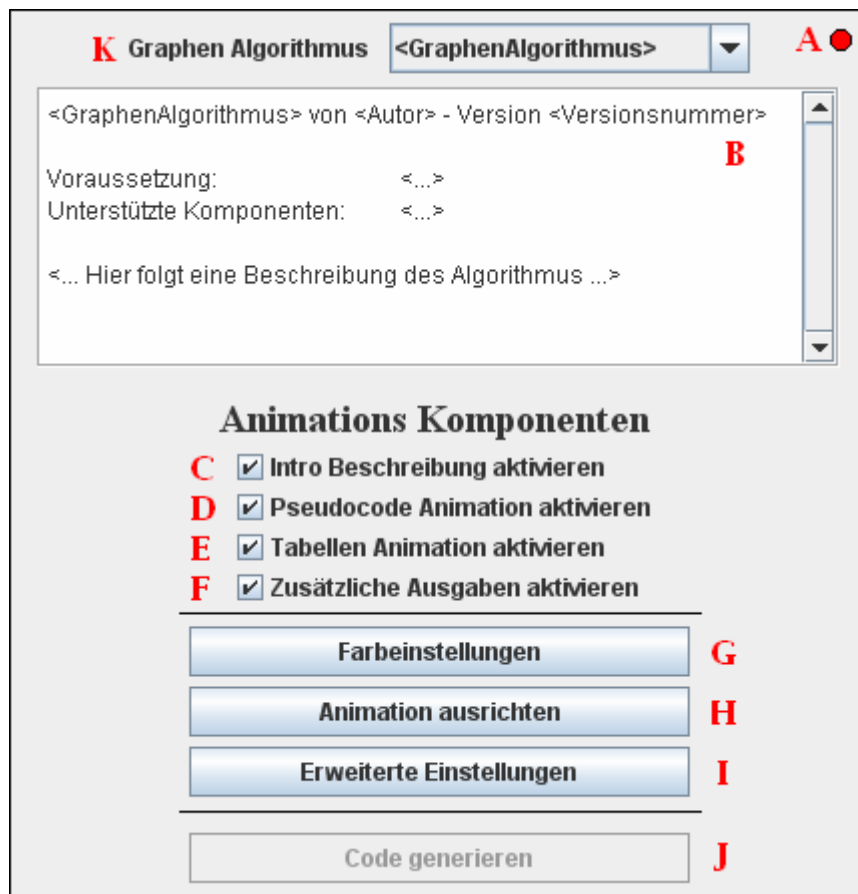


Abbildung 1: Oberfläche eines Graphenalgorithmus-Generators

2.1 Voraussetzungen

Unter diesem Punkt finden Sie eine Beschreibung der Dinge die benötigt werden, damit der Graphenalgorithmus erzeugt werden kann.

In den meisten Fällen unterstützt ein Graphenalgorithmus-Generator nur einen bestimmten Typ von Graphen. Welcher Typ für die Animation benötigt wird können Sie dann unter diesem Punkt nachlesen. Allgemeine Informationen über die zur Verfügung stehenden Graphentypen finden Sie im Benutzer-Tutorial.

Manchmal benötigt ein Graphenalgorithmus auch bestimmte Angaben (beispielsweise eine Traversierungsfolge) damit er starten kann. Diese Angaben werden hier ebenfalls aufgeführt und sind in der Regel unter dem Punkt „Erweiterte Einstellungen“ **(I)** zu machen.

Eine weitere Voraussetzung ist, dass der Graph mindestens einen Knoten enthält. Dies wird im Weiteren stillschweigend vorausgesetzt.

Sobald das Programm die benötigten Voraussetzungen vorfindet, wird der „Code generieren“ Button **(J)** aktiviert.

2.2 Animationskomponenten

Welche Komponenten zusätzlich zur Algorithmuskomponente angeboten werden, finden Sie unter dem Punkt „Unterstützte Komponenten“. Im Standardfall ist dies der Graph, gegebenenfalls wird zusätzlich die Animation der Adjazenzmatrix angeboten.

Jeder Graphenalgorithmus Generator enthält, neben der eigentlichen Algorithmus Animation, drei bis vier zusätzliche Subkomponenten. Diese voneinander unabhängigen Komponenten sollen die Animation durch erweiterte Ausgaben unterstützen. Sie können je nach Belieben aktiviert oder deaktiviert werden. In den folgenden Unterkapiteln finden Sie eine Beschreibung der einzelnen Subkomponenten.

2.2.1 Intro Beschreibung

Wählen Sie **(C)** aus, um am Anfang Ihrer Animation eine kurze Beschreibung des Algorithmus zu erhalten. Die Beschreibung der Animation entspricht ungefähr dem Beschreibungstext, den Sie im Textfeld des Panels lesen können.

2.2.2 Pseudo-Code

Wählen Sie **(D)** aus, um den Pseudo-Code des Algorithmus eingeblendet zu bekommen. Während des Ablaufs der Animation können Sie dann erkennen, wo sich der Algorithmus im Code befindet.

2.2.3 Tabellen-Animation

Wählen Sie **(E)** aus, um die Tabellen-Animation des Algorithmus eingeblendet zu bekommen. Diese Subkomponente repräsentiert die Daten die während des Ablaufs des Algorithmus gehalten werden.

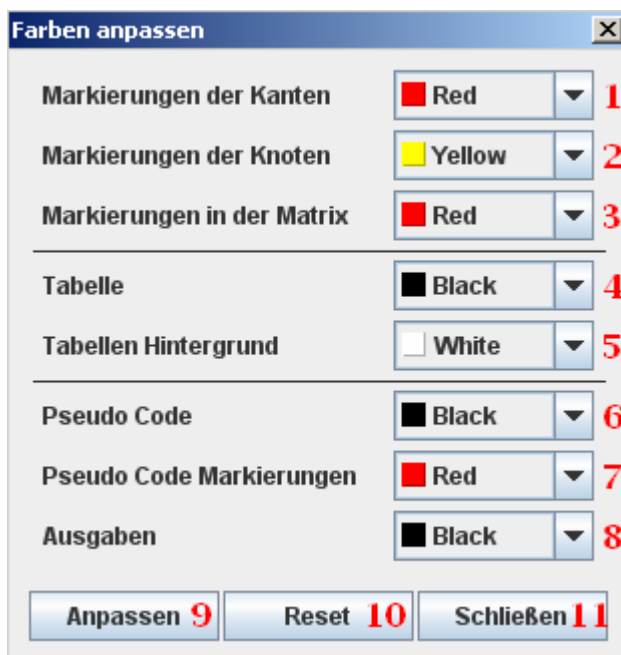
2.2.4 Zusätzliche Ausgaben

Wählen Sie **(F)** aus, um zusätzliche Kommentare zu den Algorithmenschritten zu bekommen. Dies hilft Ihnen, den Ablauf der Animation besser folgen zu können.

2.3 Farbeinstellungen

Durch Drücken von **(G)** öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können.

Abbildung 2 zeigt eine allgemeine Abbildung des Dialogs, in der einige der üblichen Farbeinstellungen gezeigt werden.



Bei **(1)** bis **(8)** können Sie die entsprechenden Farbeinstellungen ändern.

(1) - **(3)** beziehen sich auf die Farben zur Hervorhebung des aktuell vom Algorithmus benutzten Knotens oder Kante.

(4) und **(5)** beziehen sich auf die Farben der Tabellen-Animation.

(6) und **(7)** beziehen sich auf die Farben der Pseudocodeanimation.

Bei **(8)** können Sie die Farbe der „Zusätzlichen Ausgaben“ einstellen.

Abbildung 2: Farbeinstellungen

Durch Drücken von **(9)** werden die Änderungen übernommen. Durch Drücken von **(10)** werden die (in Abbildung 2 zu sehenden) Standardeinstellungen wiederhergestellt.

Um zum Hauptfenster zurück zu kehren, drücken Sie **(11)**. Beachten Sie, dass dabei nicht gespeicherte Änderungen verworfen werden.

2.4 Animation ausrichten

Ein Druck auf **(H)** öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Die Ausrichtungspunkte wurden so gewählt, dass ein maximaler Graph (sowohl von Anzahl der Knoten her, als auch von der Bounding Box her) keine Überlappungen zwischen den Animationskomponenten provoziert. Um dies zu ermöglichen wird – soweit nicht anders angegeben - davon ausgegangen, dass sich der Graph auf seiner Standardposition (50, 50) befindet.

Abbildung 3 zeigt eine allgemeine Abbildung des Dialogs, in der davon ausgegangen wird, dass der Algorithmusgenerator alle Subkomponenten unterstützt.

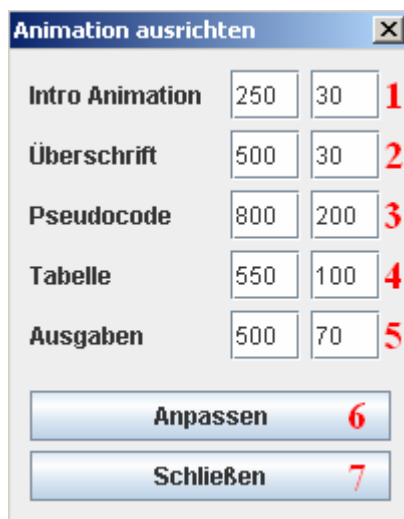


Abbildung 3: Animation ausrichten

(1) beschreibt die Mitte der obersten Zeile des Intro-Beschreibungstextes.

(2) beschreibt die Mitte des Algorithmus Titels in der Animation.

(3) beschreibt die Mitte der ersten Zeile des Pseudocodes des Algorithmus.

(4) beschreibt die obere linke Ecke der Tabellen Animation.

(5) beschreibt die Mitte der Kommentar-Ausgaben zum Algorithmus.

Wenn Sie eine der Koordinaten geändert haben, müssen Sie auf **(6)** drücken, um die Änderungen zu speichern. Durch einen Druck auf **(7)** schließt sich das Fenster, wobei nicht gespeicherte Änderungen verworfen werden.

2.5 Erweiterte Einstellungen

Durch Drücken von **(I)** öffnet sich ein Dialog, in dem Sie die erweiterten Einstellungen eines Graphenalgorithmus Generator machen können. Diese unterscheiden sich je nach Art des Algorithmus voneinander und werden in den entsprechenden Abschnitten genauer behandelt.

3. Algorithmus zur Tiefensuche

Wenn Sie „Tiefensuche (DFS)“ ausgewählt haben, sollten Sie das Panel aus Abbildung 4 vor sich sehen.

Im Textfeld (**B**) erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von (**I**) erzeugen Sie diesen Teil der Animation. (**A**) zeigt den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

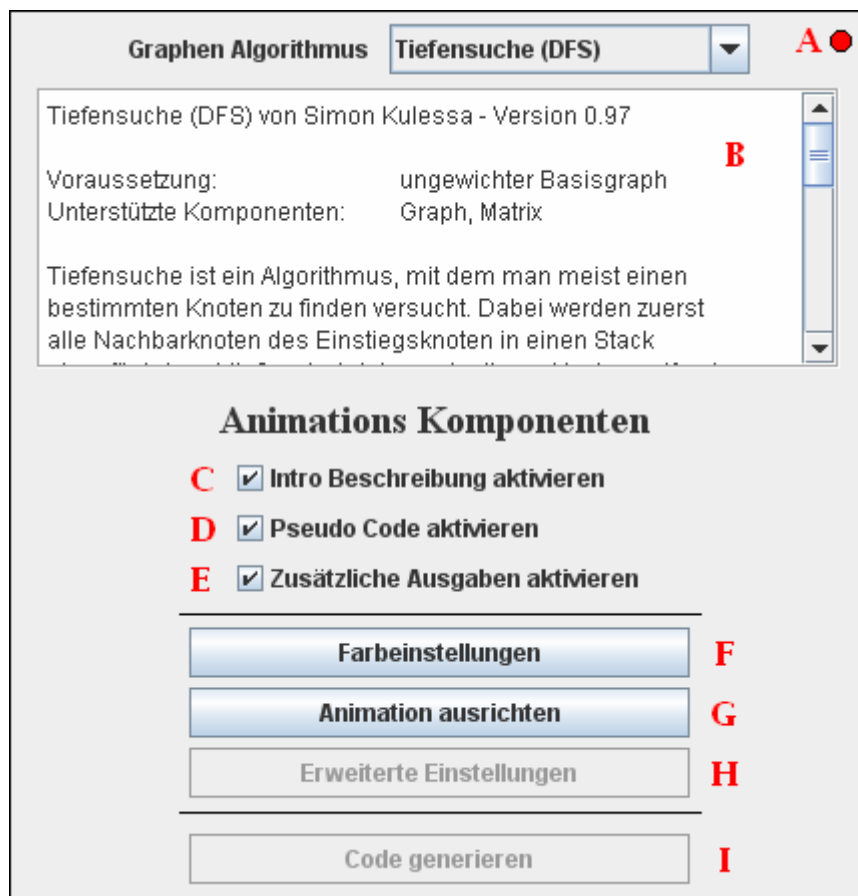


Abbildung 4: Tiefensuche

3.1 Voraussetzungen

Um eine Tiefensuche-Animation zu generieren, wird ein Graph des Typs „Basisgraph“ benötigt. Da die Gewichtung der Kanten in einem DFS Algorithmus keinerlei Rolle spielt, wird hier nur die ungewichtete Version des Basisgraphen unterstützt. Er kann sowohl gerichtet als auch ungerichtet sein und sollte mindestens zwei Knoten enthalten. Ansonsten werden keine besonderen Einstellungen benötigt.

In Abbildung 4 ist zu erkennen, dass die Buttons „Code generieren“ (**I**) und „Erweiterte Einstellungen“ (**H**) deaktiviert sind. Sobald das Programm die benötigten Voraussetzungen vorfindet, werden die Buttons aktiviert.

3.2 Animationskomponenten

Die Tiefensuche-Animation unterstützt sowohl die Animation des Graphen und als auch seiner Adjazenzmatrix. Als Subkomponenten stehen die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

3.3 Farbeinstellungen

Durch Drücken von (**F**) öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Eine genauere Beschreibung der Farbeinstellungen entnehmen Sie bitte Kapitel 2.3.

3.4 Animation ausrichten

Ein Druck auf (**G**) öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

3.5 Erweiterte Einstellungen

Durch Drücken von **(H)** öffnet sich ein Dialog, in dem Sie die erweiterten Einstellungen des Tiefensuche-Algorithmus festlegen können. Sie haben die Möglichkeit zur Änderung der äußeren Traversierungsfolge (also der Abarbeitung der Knoten von außerhalb des Graphen) und der Suche nach einem speziellen Knoten.

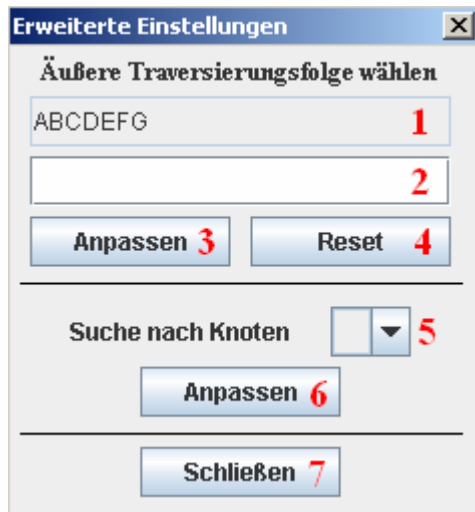


Abbildung 5: Erweiterte Einstellungen

(1) zeigt die aktuelle Traversierungsfolge. Standard ist das Abarbeiten der Knoten in alphabetischer Reihenfolge.

In das Textfeld bei **(2)** können Sie eine neue Abarbeitungsreihenfolge eingeben. Beachten Sie dabei, dass jede Kennzeichnung des Knotens genau einmal vorkommen muss.

Durch Drücken von **(3)** wird die von Ihnen eingegebene Traversierungsfolge überprüft. Sollte sie den Vorgaben entsprechen, wird sie nun bei **(1)** erscheinen, andernfalls erscheint bei **(2)** eine Fehlermeldung.

Durch Drücken des „Reset“-Buttons **(4)** wird die Standardabarbeitung (alphabetische Reihenfolge) wieder hergestellt.

Bei **(5)** können Sie den Knoten auswählen, nach dem Sie suchen wollen. Das leere Tag steht dabei für die Deaktivierung des Suchmodus. Beachten Sie, dass der Knoten, den Sie suchen wollen, im Graph vorkommen kann, aber nicht muss.

Durch Drücken von **(6)** werden die Einstellungen zur Suche nach einem Knoten gespeichert. Ein Druck auf **(7)** schließt das Dialogfenster und verwirft alle nicht gespeicherten Änderungen.

3.6 Screenshots einer Beispielanimation

Tiefensuche (DFS)

Tiefensuche ist ein Algorithmus, mit dem man meist einen bestimmten Knoten zu finden versucht. Dabei werden zuerst alle Nachbarknoten des Einstiegsknoten in einen Stack eingefügt. Anschließend wird der erste dieser Knoten entfernt und das selbe Schema wird auf seine Nachbarknoten angewandt. Beim Einfügen in den Stack ist zu beachten das immer vorne eingefügt wird.

In Graphen kann man diesen Algorithmus auch dazu benutzen, um festzustellen, welche Komponenten eines Graphens zusammenhängend sind.

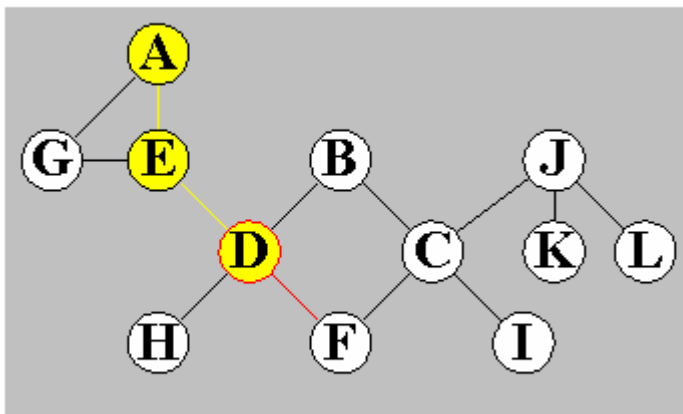
Tiefensuche ist eigentlich effizienter als Breitensuche, kann im Einzelfall allerdings ungünstig verlaufen. Die Komplexitätsklasse des Algorithmus ist $O(n)$.

Anmerkung:

Die hier vorgestellte Variante verwendet einen spezielle LIFO Stack der anfangs alle Knoten enthält. Sobald ein Knoten in den Stack eingefügt werden soll, wird überprüft ob er noch im Stack vorhanden ist. Ist dies der Fall, wird er auf die Spitze des Stacks verschoben. Andernfalls wird er nicht eingefügt.

Abbildung 6: Intro Beschreibung

Tiefensuche (DFS)

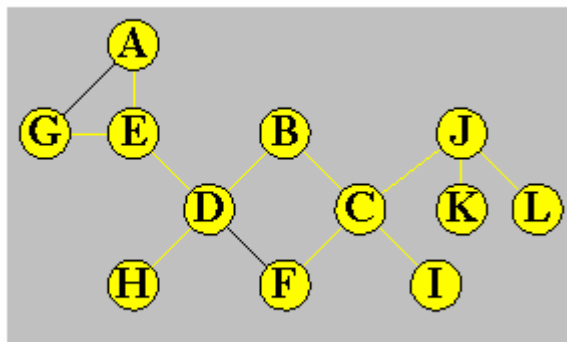


```
public void dfs () {
    Stack stack = initStack(graph);
    do {
        Node current = stack.pop();
        for (Edge e : current) {
            stack.push(e.getEnd());
        }
    } while (!stack.isEmpty());
}
```

Lade Nachbarknoten F und versuche ihn in den Stack einzufügen.

Abbildung 7: Zwischenschritt der Animation

Tiefensuche (DFS)



```
public void dfs () {
    Stack stack = initStack(graph);
    do {
        Node current = stack.pop();
        for (Edge e : current) {
            stack.push(e.getEnd());
        }
    } while (!stack.isEmpty());
}
```

Alle Knoten wurden aus dem Stack entfernt. Die Traversierungsfolge ist A E D B C J L K I F H G .

Abbildung 8: Ende der Animation

4. Algorithmus zur Breitensuche

Wenn Sie „Breitensuche (BFS)“ ausgewählt haben, sollten Sie das Panel aus Abbildung 9 vor sich sehen.

Im Textfeld **(B)** erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von **(I)** erzeugen Sie diesen Teil der Animation. **(A)** zeigt, wie gewohnt, den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

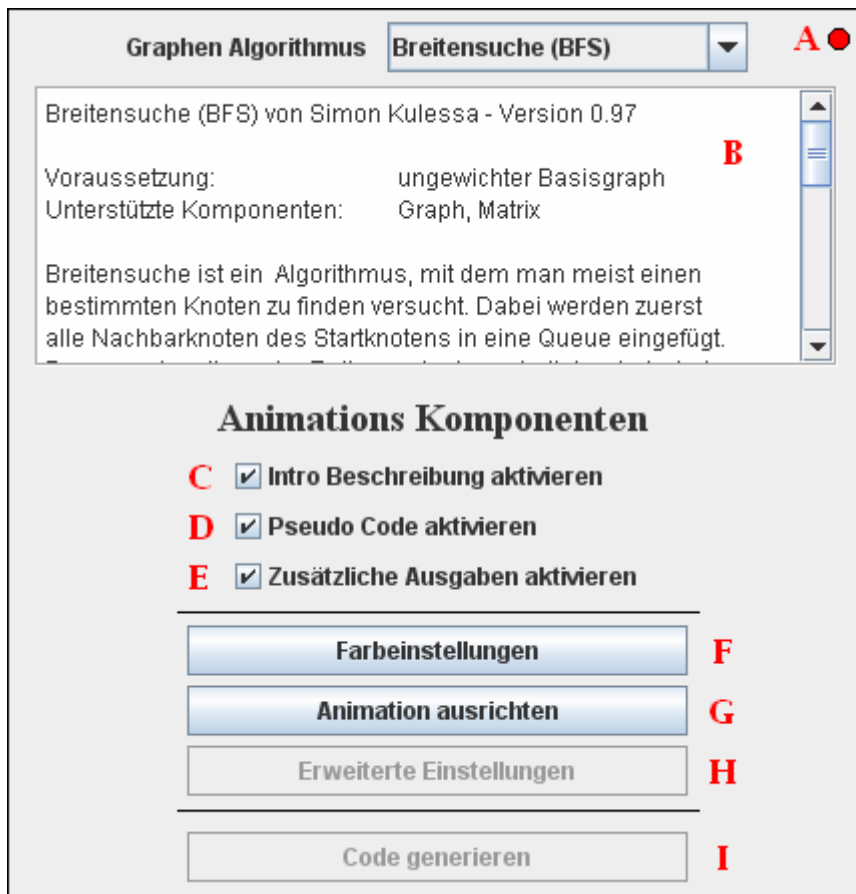


Abbildung 9: Breitensuche

4.1 Voraussetzungen

Um eine Breitensuche-Animation zu generieren, wird ein Graph des Typs „Basisgraph“ benötigt. Da die Gewichtung der Kanten in einem BFS Algorithmus keinerlei Rolle spielt, wird hier nur die ungewichtete Version des Basisgraphen unterstützt. Er kann sowohl gerichtet als auch ungerichtet sein und sollte mindestens zwei Knoten enthalten. Ansonsten werden keine besonderen Einstellungen benötigt.

In Abbildung 9 ist zu erkennen, dass die Buttons „Code generieren“ und „Erweiterte Einstellungen“ deaktiviert sind. Sobald das Programm die benötigten Voraussetzungen vorfindet, werden die Buttons aktiviert.

4.2 Animationskomponenten

Die Breitensuche-Animation unterstützt sowohl die Animation des Graphen und als auch seiner Adjazenzmatrix. Als Subkomponenten stehen die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

4.3 Farbeinstellungen

Durch Drücken von **(F)** öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Eine genauere Beschreibung der Farbeinstellungen entnehmen Sie bitte Kapitel 2.3.

4.4 Animation ausrichten

Ein Druck auf **(G)** öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

4.5 Erweiterte Einstellungen

Die erweiterten Einstellungen entsprechen denen des Tiefensuche Algorithmus. Eine genaue Beschreibung entnehmen Sie bitte Kapitel 3.5.

4.6 Screenshots einer Beispielanimation

Breitensuche (BFS)

Breitensuche ist ein Algorithmus, mit dem man meist einen bestimmten Knoten zu finden versucht. Dabei werden zuerst alle Nachbarknoten des Startknotens in eine Queue eingefügt. Dann werden diese der Reihe nach abgearbeitet, wobei wiederum ihre noch nicht besuchten Nachbarknoten in die Queue eingefügt werden. Das Ausbreitungsschema des BFS Algorithmus wird auch als Wellenfront bezeichnet.

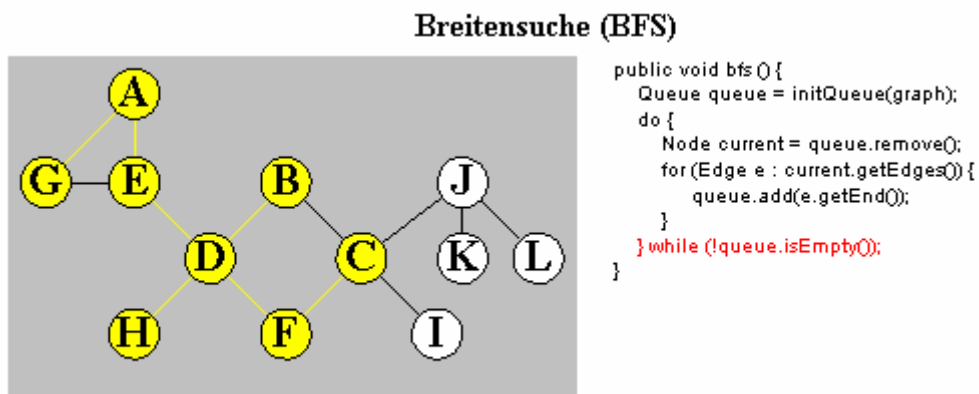
In Graphen kann man diesen Algorithmus auch dazu benutzen, um festzustellen welche Komponenten eines Graphen zusammenhängend sind.

Die Komplexitätsklasse des Algorithmus ist $O(n)$.

Anmerkung:

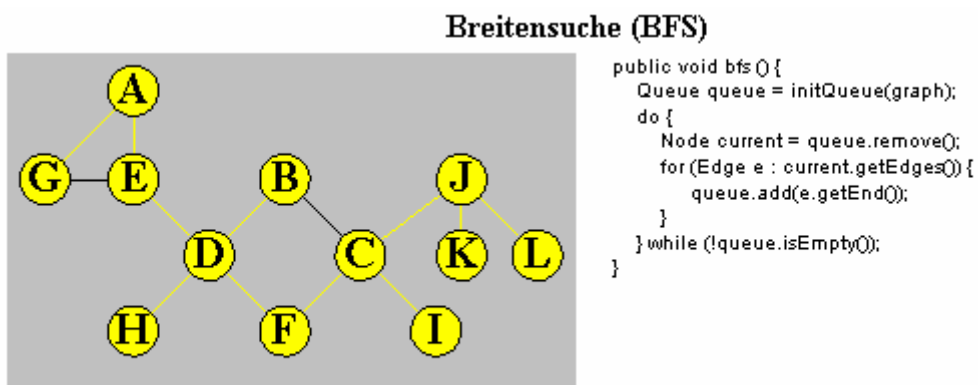
Die hier vorgestellte Variante verwendet eine spezielle FIFO Queue, die anfangs in einer zweiten Lage alle Knoten enthält. Sobald ein Knoten in die Queue eingefügt werden soll, wird überprüft ob er in dieser Lage vorhanden ist. Ist dies der Fall wird er aus dieser zweiten Lage entfernt und in die eigentliche Queue eingefügt. Andernfalls wird er nicht eingefügt.

Abbildung 10: Intro Beschreibung



Prüfe ob die Queue leer ist, andernfalls entferne nächsten Knoten von der Queue.

Abbildung 11: Zwischenschritt der Animation



Alle Knoten wurden aus der Queue entfernt. Die Traversierungsfolge ist A G E D F H B C I J K L .

Abbildung 12: Ende der Animation

5. Färbungsproblem – Ein NP-vollständiges Problem

Wenn Sie „Färbungsproblem“ ausgewählt haben, sollten Sie das Panel aus Abbildung 13 vor sich sehen.

Im Textfeld **(B)** erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von **(I)** erzeugen Sie diesen Teil der Animation. **(A)** zeigt, wie gewohnt, den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

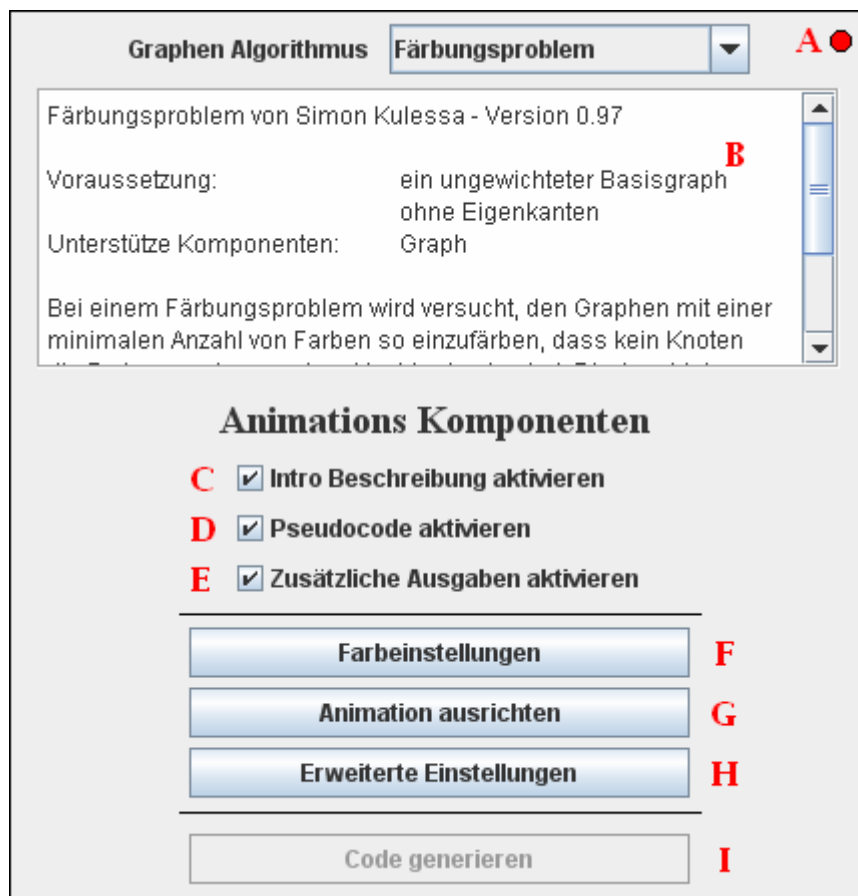


Abbildung 13: Färbungsproblem

5.1 Voraussetzungen

Um einen Färbungsproblem-Animation zu generieren, wird ein Graph des Typs „Basisgraph“ benötigt. Da die Gewichtung der Kanten im Färbungsproblem-Algorithmus keine Rolle spielt, wird hier nur die ungewichtete Version des Basisgraphen unterstützt. Er kann sowohl gerichtet als auch ungerichtet sein und sollte mindestens zwei Knoten enthalten. Weiterhin darf er keine Eigenkanten enthalten, Kanten der Form $A \rightarrow A$ sind also nicht erlaubt. Ansonsten werden keine besonderen Einstellungen benötigt.

In Abbildung 13 ist zu erkennen, dass der „Code generieren“ Button deaktiviert ist. Sobald das Programm die benötigten Voraussetzungen vorfindet, wird der Button aktiviert.

5.2 Animationskomponenten

Die Färbungsproblem-Animation unterstützt eine Animation des Graphen, die auch unbedingt ausgewählt werden sollte, da nur so die „Färbung“ als solche zu erkennen ist. Die Animation der Matrix hingegen wird nicht unterstützt. Als Subkomponenten stehen die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

5.3 Farbeinstellungen

Durch Drücken von **(F)** öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Neben den in Kapitel 2.3.beschriebenen Farben, finden Sie hier eine Farbhierarchie.

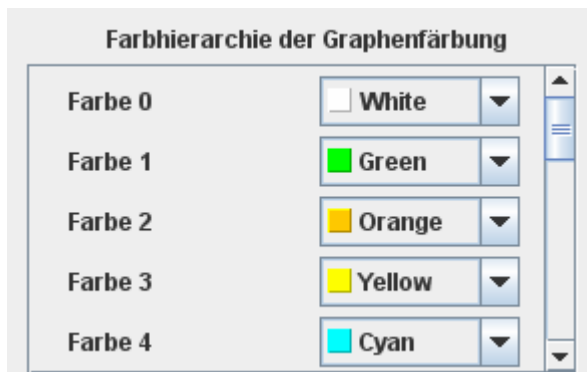


Abbildung 14:
Färbungsproblem Farbhierarchie

Bei dieser können Sie die Farben einstellen, die bei der Färbung des Graphen benutzt werden sollen.

Farbe 0 entspricht der „Nicht-Färbung“ eines Knotens, alle anderen Farben entsprechen der jeweiligen Farbnummer.

Bitte beachten Sie, dass keine Farbe zweimal in der Farbhierarchie vorkommen sollte. Dies würde die visuelle Unterscheidung in der Animation unmöglich machen.

5.4 Animation ausrichten

Ein Druck auf **(G)** öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

5.5 Erweiterte Einstellungen

Durch Drücken von **(H)** öffnet sich ein Dialog, in dem sie die erweiterten Einstellungen des Färbungsproblems-Algorithmus festlegen können. Sie können hier die Länge der Animation festzulegen.

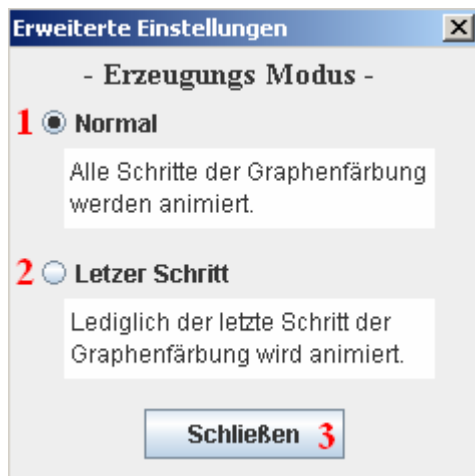


Abbildung 15: Erweitere Einstellungen

Falls Sie **(1)** auswählen, wird jeder Schritt des Färbungsproblems animiert. Dies können unter Umständen sehr viele Schritte sein.

Bei der Auswahl von **(2)** wird nur der letzte Schritt der Animation erzeugt.

Bitte beachten Sie, dass die Erstellung einer Animation im zweiten Modus etwas langsamer als der erste Modus ist, da hier der letzte Schritt („Färbe den Graphen mit x Farben“) zweimal ausgeführt werden muss.

Ein Druck auf **(3)** schließt das Dialogfenster und speichert die aktuelle Auswahl.

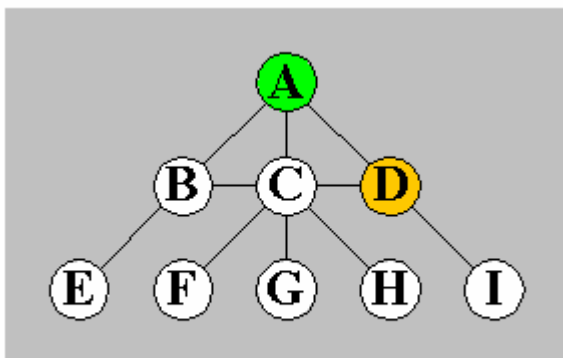
5.6 Screenshots einer Beispielanimation

Färbungsproblem

Bei einem Färbungsproblem wird versucht, den Graphen mit einer minimalen Anzahl von Farben so einzufärben, dass kein Knoten die gleiche Farbe wie einer seiner Nachbarknoten hat. Die Anzahl der benötigten Farben wird chromatische Zahl genannt.

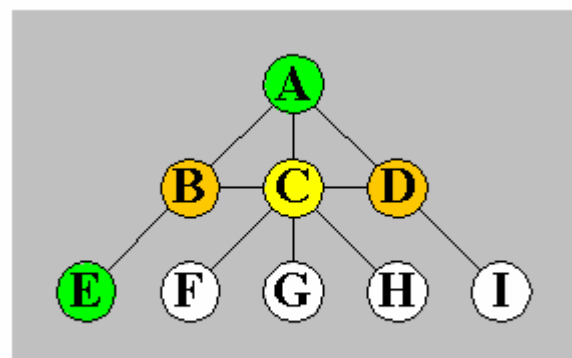
Dieses Problem ist NP-vollständig. Das bedeutet, dass es im Allgemeinen nur durch Ausprobieren aller Möglichkeiten gelöst werden kann.

Abbildung 16: Intro Beschreibung



Gültige Färbung für Knoten D gefunden (2).

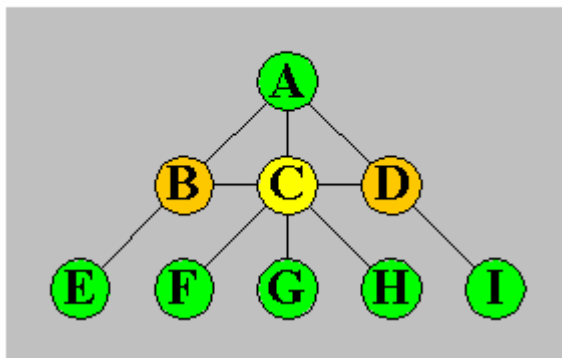
Abbildung 17: Zwischenschritt



Versuche nun den Knoten (G) zu färben.

Abbildung 18: Zwischenschritt

Färbungsproblem



Gültige Färbung gefunden -
Die Chromatische Zahl ist 3.

```
public boolean color(Node[] dfs, int max) {
    initColors(dfs, 0);
    int i = 0;
    do {
        boolean ok = false;
        while (!ok && ++dfs[i].color <= max) {
            ok = true;
            for (Edge e : dfs[i] && ok)
                ok = dfs[i].color != e.getEnd().color;
        }
        if (ok) i++;
        else dfs[i--].color = 0;
    } while (i > 0 && i < dfs.length);
    return (i == max);
}
```

```
public int getChromaticNumber() {
    Node[] dfs = precomputeDFS(graph);
    int i = 0;
    do {
        found = color(dfs, ++i);
    } while (!found);
    return i;
}
```

Abbildung 19: Ende der Animation

6. Kruskal – Berechnung des minimalen Spannbaumes

Wenn Sie „Kruskal“ ausgewählt haben, sollten Sie das Panel aus Abbildung 20 vor sich sehen.

Im Textfeld **(B)** erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von **(J)** erzeugen Sie diesen Teil der Animation. **(A)** zeigt, wie gewohnt, den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

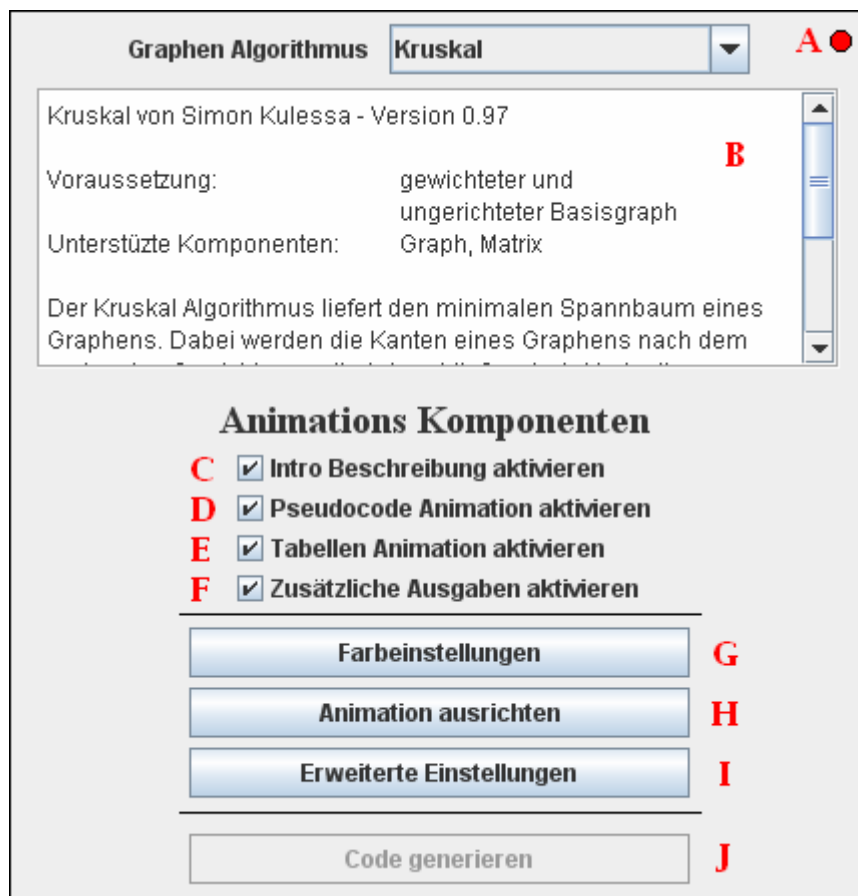


Abbildung 20: Kruskal

6.1 Voraussetzungen

Um eine Kruskal-Animation zu generieren, wird ein Graph des Typs „Basisgraph“ benötigt. Die Gewichtung der Kanten ist ebenso zwingend erforderlich wie die Ungerichtetheit der Kanten. Ansonsten werden keine besonderen Einstellungen benötigt.

In Abbildung 20 ist zu erkennen, dass der „Code generieren“ Button deaktiviert ist. Sobald das Programm die benötigten Voraussetzungen vorfindet, wird der Button aktiviert.

6.2 Animationskomponenten

Die Kruskal-Animation unterstützt sowohl die Animation des Graphen und als auch seiner Adjazenzmatrix. Als Subkomponenten stehen wie gewohnt die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Weiterhin ist eine Tabellen-Animation verfügbar, wobei die Tabelle neben die Kanten des minimalen Spannbaumes, sowie die Teilkomponenten des Graphen, in die er während des Algorithmus zerfällt, enthält.

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

6.3 Farbeinstellungen

Durch Drücken von **(G)** öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Neben den üblichen Farbeinstellungen können Sie hier den Kanten im Spannbaum eine eigene Farbe zuzuweisen.

Eine genauere Beschreibung der Farbeinstellungen entnehmen Sie bitte Kapitel 2.3.

6.4 Animation ausrichten

Ein Druck auf **(H)** öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

6.5 Erweiterte Einstellungen

Durch Drücken von **(I)** öffnet sich ein Dialog, in dem sie die erweiterten Einstellungen des Kruskal Algorithmus festlegen können. Sie haben hier die Möglichkeit die Animation des Graphen, sowie den Ablauf der Animation zu beeinflussen.

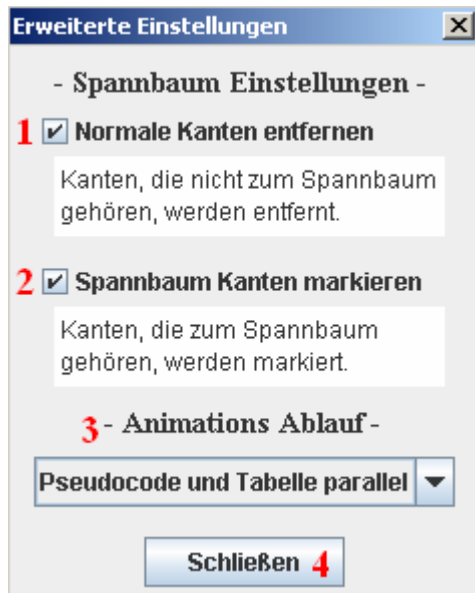


Abbildung 21: Erweitere Einstellungen

Falls Sie **(1)** auswählen, werden Kanten die nicht zum Spannbaum gehören entfernt.

Falls Sie **(2)** auswählen, werden Kanten, die zum Spannbaum gehören, markiert.

Bei **(3)** können Sie den Ablauf der Animation einstellen. Die Auswahl „Pseudocode und Tabelle parallel“ entspricht dem normalen Animationsablauf.

Die beiden anderen Einstellungen („Erst Pseudocode, dann Tabelle“ und „Erst Tabelle, dann Pseudocode“) führen dazu dass der Algorithmus zweimal animiert wird. Einmal wird nur der Pseudocode, und beim anderen Mal nur die Tabelle animiert.

Beachten Sie, dass sich die Ausrichtung der Komponenten nur am Standardmodus orientiert.

Ein Druck auf **(4)** schließt das Dialogfenster und speichert die aktuelle Auswahl.

6.6 Screenshots einer Beispielanimation

Kruskal

Der Kruskal Algorithmus liefert den minimalen Spannbaum eines Graphens. Dabei werden die Kanten eines Graphens nach dem geringsten Gewicht vorsortiert. Anschließend wird jede dieser Kanten überprüft und aus den einzelnen Knoten bilden sich dabei zusammenhängende Komponenten.

Die Effizienz des Algorithmus liegt in der Verwendung des Greedy Prinzips. Dieses lautet: Treffe Entscheidungen aufgrund vorliegender Informationen und revidiere diese Entscheidung nie.

Abbildung 22: Intro Beschreibung

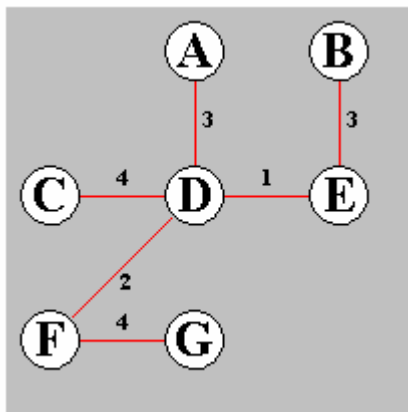


Abbildung 23:
Minimaler Spannbaum des Graphen

Kante	T	zusammenhängende Komponenten
	{	{A}, {B}, {C}, {D}, {E}, {F}, {G}
(D,E)	(D,E)	{A}, {B}, {C}, {D, E}, {F}, {G}
(D,F)	(D,F)	{A}, {B}, {C}, {D, E, F}, {G}
(B,E)	(B,E)	{A}, {B, D, E, F}, {C}, {G}
(A,D)	(A,D)	{A, B, D, E, F}, {C}, {G}
(F,G)	(F,G)	{A, B, D, E, F, G}, {C}
(C,D)	(C,D)	{A, B, C, D, E, F, G}
(E,G)		
(B,D)		
(D,G)		
(C,F)		
(A,B)		
(A,C)		
	}	{A, B, C, D, E, F, G}

Abbildung 24: Tabellen Animation

	A	B	C	D	E	F	G
A	0	7	9	3	0	0	0
B	0	0	0	5	3	0	0
C	0	0	0	4	0	6	0
D	0	0	0	0	1	2	6
E	0	0	0	0	0	0	5
F	0	0	0	0	0	0	4
G	0	0	0	0	0	0	0

Abbildung 25:
Matrix Animation

```

public Vector<Edge> kruskal(Graph g) {
    Edge[] list = getSortedEdgeList(g);
    Cluster[] c = createClusters(g);
    Vector<Edge> spanningTree = new Vector<Edge>();
    for (Edge e: list) {
        Cluster a = getCluster(e.getStart(), c);
        Cluster b = getCluster(e.getTarget(), c);
        if (a != b) {
            spanningTree.add(e);
            combine(a, b);
        }
    }
    return spanningTree;
}
    
```

Abbildung 26: Pseudocode

7. Dijkstra – Lösen von kürzesten Wege-Problemen I

Wenn Sie „Dijkstra“ ausgewählt haben, sollten Sie das Panel aus Abbildung 27 vor sich sehen.

Im Textfeld (**B**) erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von (**J**) erzeugen Sie diesen Teil der Animation. (**A**) zeigt, wie gewohnt, den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

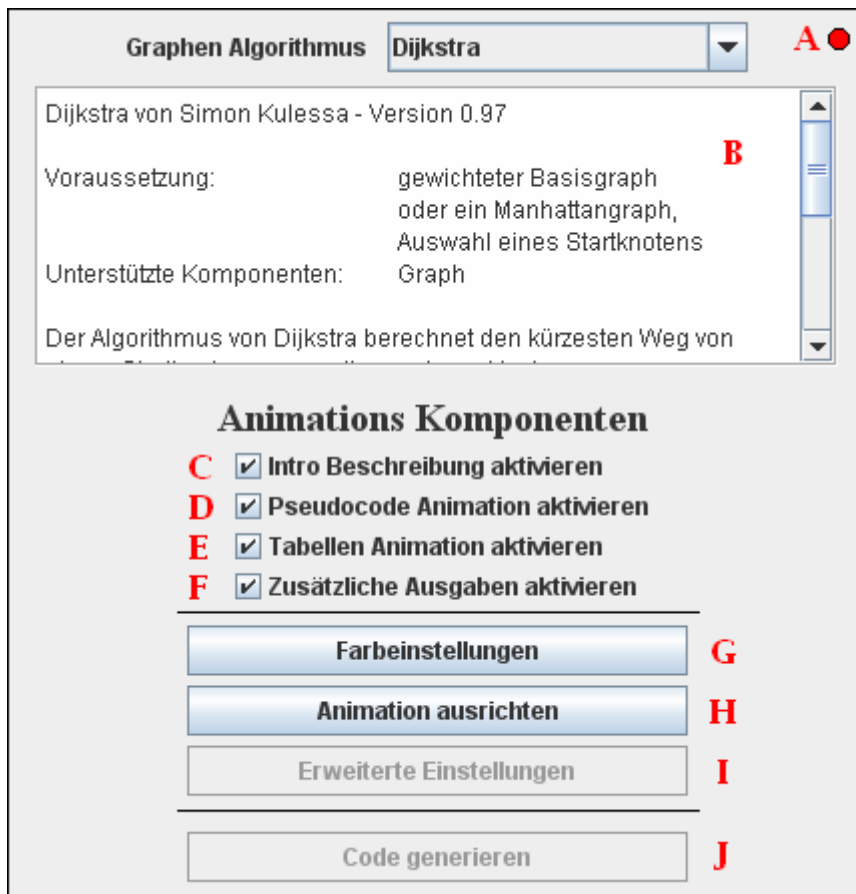


Abbildung 27: Dijkstra

7.1 Voraussetzungen

Um einen Dijkstra-Animation zu generieren, wird ein Graph des Typus „Basisgraph“ oder „Manhattangraph“ benötigt. Die genaue Definition eines Manhattangraphen finden Sie im Benutzer-Tutorial. Der Graph muss gewichtet sein, darf aber sowohl gerichtet als auch ungerichtet sein.

Damit der Algorithmus generiert werden kann, muss in den „Erweiterten Einstellungen“ ein Startknoten ausgewählt werden. Der Button „Erweiterte Einstellungen“ (**I**) wird aktiviert, sobald ein Graph geladen wurde.

Der „Code generieren“ Button (**J**) wird aktiviert, sobald diese Einstellungen bestätigt wurden.

7.2 Animationskomponenten

Die Dijkstra-Animation unterstützt nur die Animation des Graphen. Als Subkomponenten stehen wie gewohnt die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Weiterhin ist eine Tabellen-Animation verfügbar, wobei die Tabelle neben der Abarbeitungsschritte, auch die Entfernung des Startknotens zu allen anderen Knoten enthält. Das Zeichen * steht dabei für unendlich.

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

7.3 Farbeinstellungen

Durch Drücken von (**G**) öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Neben den üblichen Farbeinstellungen können Sie hier die Farbe für Kanten im kürzesten Weg, für temporäre und endgültige Label festlegen.

Eine genauere Beschreibung der Farbeinstellungen entnehmen Sie bitte Kapitel 2.3.

7.4 Animation ausrichten

Ein Druck auf (**H**) öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

7.5 Erweiterte Einstellungen

Durch Drücken von **(I)** öffnet sich ein Dialog, in dem Sie die erweiterten Einstellungen des Dijkstra-Algorithmus festlegen können. Hier wird der Start- und Zielknoten des kürzesten Wege-Problems festgelegt. Außerdem haben Sie hier die Möglichkeit, den Ablauf der Animation zu beeinflussen.

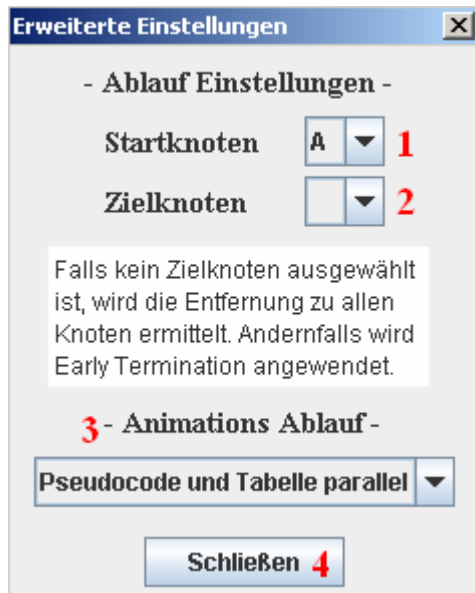


Abbildung 28: Erweiterte Einstellungen

Bei **(1)** müssen Sie den Startknoten der Suche festlegen. Diese Aktion ist eine der Voraussetzung dafür, dass die Animation generiert werden kann.

Bei **(2)** können Sie optional den Zielknoten des kürzesten Wege-Problems festlegen. Falls Sie keinen Knoten auswählen, werden die kürzesten Wege zu allen Knoten bestimmt.

Bei **(3)** können Sie den Ablauf der Animation einstellen. Die Auswahl „Pseudocode und Tabelle parallel“ entspricht dem normalen Animationsablauf.

Die beiden anderen Einstellungen („Erst Pseudocode, dann Tabelle“ und „Erst Tabelle, dann Pseudocode“) führen dazu dass der Algorithmus zweimal animiert wird. Einmal wird nur der Pseudocode, und beim anderen Mal nur die Tabelle animiert.

Beachten Sie, dass sich die Ausrichtung der Komponenten nur am Standardmodus orientiert.

Ein Druck auf **(4)** schließt das Dialogfenster und speichert die aktuelle Auswahl.

7.6 Screenshots einer Beispielanimation

Dijkstra

Der Algorithmus von Dijkstra berechnet den kürzesten Weg von einem Startknoten aus zu allen anderen Knoten. Anfangs haben alle Knoten einen Abstand von +UNENDLICH. Während des Algorithmus werden sukzessive alle Nachbarknoten überprüft, wobei jeweils vom nächstgelegenen Knoten ausgegangen wird.

Der Algorithmus gehört zur Klasse der Greedy-Algorithmen und besitzt eine Komplexität von $O(n \log n)$, wobei n die Anzahl der Knoten im Graphen repräsentiert.

Abbildung 29: Intro Beschreibung

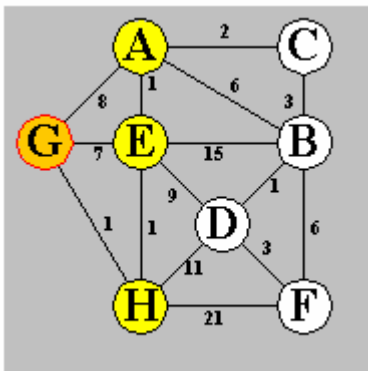
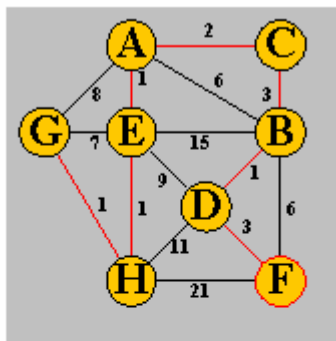


Abbildung 30:
Graph während der Animation

	A	B	C	D	E	F	G	H
G	*	*	*	*	*	*	0	*
	8				7			1

Abbildung 31:
Tabelle zum rechts abgebildeten Graphen

Dijkstra



Der kürzeste Weg von G nach F hat die Länge 12.

	A	B	C	D	E	F	G	H
G	*	*	*	*	*	*	0	*
H	8	*	*	12	2	22	0	1
E	3	17	*	11	2	22	0	1
A	3	9	5	11	2	22	0	1
C	3	8	5	11	2	22	0	1
B	3	8	5	9	2	14	0	1
D	3	8	5	9	2	12	0	1
F								

Abbildung 32: Screenshot vom Ende der Animation

```

public int dijkstra(Node start, Node target) {
    start.distance = 0;
    set.add(start);
    while (!set.isEmpty()) {
        Node n = set.getNextNeighbour();
        if (n == target) {
            return n.distance;
        }
        for (Edge e : n.getEdges()) {
            v = e.getDestination();
            if (!set.contains(v)) {
                v.distance = n.distance + e.weight;
                set.add(v);
            } else if (v.distance > n.distance + e.weight) {
                v.distance = n.distance + e.weight;
            }
        }
    }
    return -1;
}

```

8. Bidirektionaler Dijkstra - Kürzesten Wege-Problemen II

Wenn Sie „Dijkstra (Bidirektional)“ ausgewählt haben, sollten Sie das Panel aus Abbildung 33 vor sich sehen.

Im Textfeld (**B**) erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von (**J**) erzeugen Sie diesen Teil der Animation. (**A**) zeigt, wie gewohnt, den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

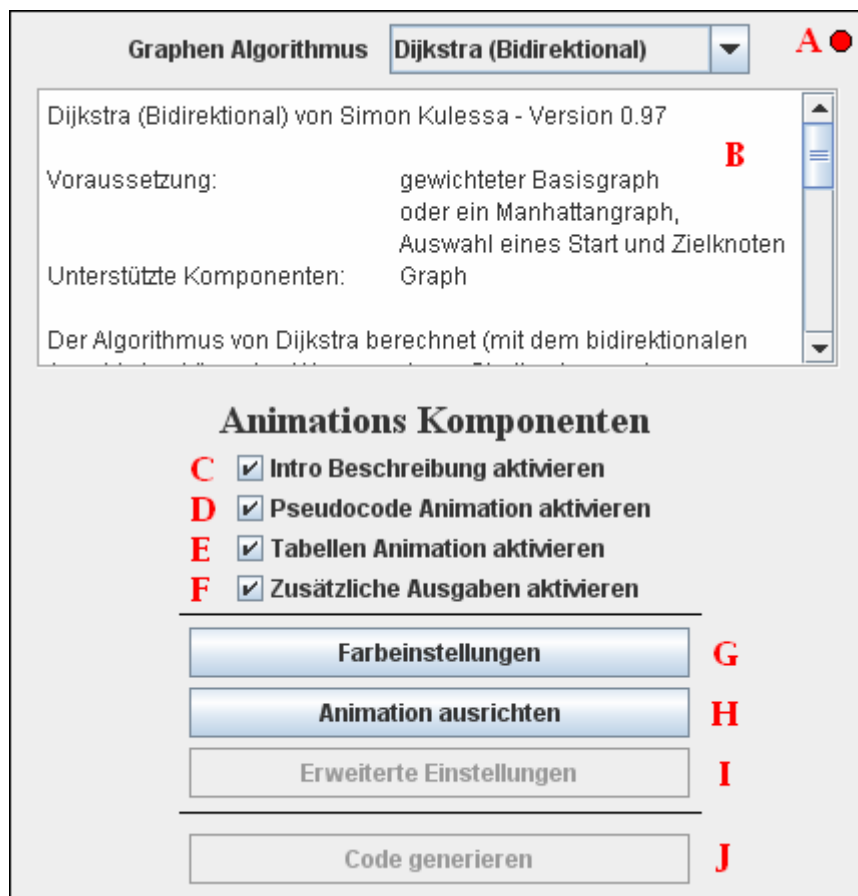


Abbildung 33 : Bidirektionaler Dijkstra

8.1 Voraussetzungen

Um eine bidirektionale Dijkstra-Animation zu generieren, wird ein Graph des Typus „Basisgraph“ oder „Manhattangraph“ benötigt. Die genaue Definition eines Manhattangraphen finden Sie im Benutzer-Tutorial. Der Graph muss gewichtet sein, darf aber sowohl gerichtet als auch ungerichtet sein.

Damit der Algorithmus generiert werden kann, muss in den „Erweiterten Einstellungen“ sowohl ein Startknoten als auch ein Zielknoten ausgewählt werden. Der „Erweiterte Einstellungen“ Button (**I**) wird aktiviert, sobald ein Graph geladen wurde.

Der „Code generieren“ Button (**J**) wird aktiviert, sobald alle Voraussetzungen erfüllt wurden.

8.2 Animationskomponenten

Die bidirektionale Dijkstra-Animation unterstützt nur die Animation des Graphen. Als Subkomponenten stehen wie gewohnt die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Weiterhin ist eine Tabellen-Animation verfügbar, wobei die Tabelle neben der Abarbeitungsschritte auch die Entfernung des Start- und des Zielknotens zu allen anderen Knoten enthält (das Zeichen * steht dabei für unendlich).

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

8.3 Farbeinstellungen

Durch Drücken von (**G**) öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Neben den üblichen Farbeinstellungen können Sie hier die Farbe für Kanten im kürzesten Weg, für temporäre und endgültige Label beider Abarbeitungsrichtungen festlegen.

Eine genauere Beschreibung der Farbeinstellungen entnehmen Sie bitte Kapitel 2.3.

8.4 Animation ausrichten

Ein Druck auf (**H**) öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

8.5 Erweiterte Einstellungen

Durch Drücken von **(I)** öffnet sich ein Dialog, in dem sie die erweiterten Einstellungen des Bidirektionalen Dijkstra Algorithmus festlegen können. Hier wird der Start- und Zielknoten des kürzesten Wege-Problems festgelegt. Außerdem haben Sie hier die Möglichkeit, den Ablauf der Animation zu beeinflussen.

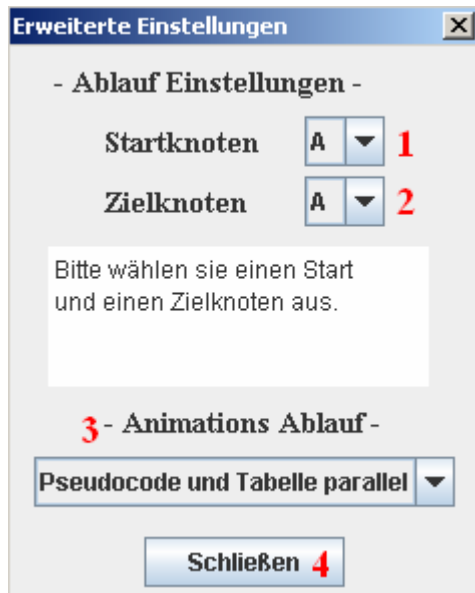


Abbildung 34: Erweiterte Einstellungen

Bei **(1)** und **(2)** müssen Sie die Start und Zielknoten der Suche festlegen. Diese Aktion ist eine der Voraussetzung dafür, dass die Animation generiert werden kann.

Bei **(3)** können Sie den Ablauf der Animation einstellen. Die Auswahl „Pseudocode und Tabelle parallel“ entspricht dem normalen Animationsablauf.

Die beiden anderen Einstellungen („Erst Pseudocode, dann Tabelle“ und „Erst Tabelle, dann Pseudocode“) führen dazu dass der Algorithmus zweimal animiert wird. Einmal wird nur der Pseudocode, und beim anderen Mal nur die Tabelle animiert.

Beachten Sie, dass sich die Ausrichtung der Komponenten nur am Standardmodus orientiert.

Ein Druck auf **(4)** schließt das Dialogfenster und speichert die aktuelle Auswahl.

8.6 Screenshots einer Beispielanimation

Dijkstra (Bidirektional)

Der Algorithmus von Dijkstra berechnet (mit dem bidirektionalen Ansatz) den kürzesten Weg von einem Startknoten zu einem Zielknoten.

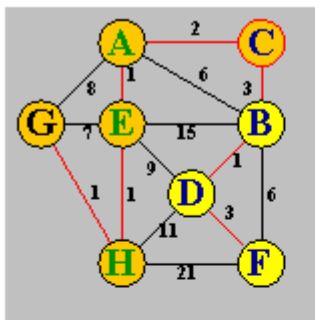
Dabei wird abwechselnd ein Dijkstra Schritt vom Start und ein Dijkstra Schritt vom Zielknoten ausgeführt. Das heißt, es werden immer die Nachbarknoten des jeweiligen ausgewählten Knotens betrachtet und die aktuelle minimale Entfernung berechnet.

Sobald einer der ausgewählten Knoten von der anderen Richtung als abgearbeitet gekennzeichnet wurde, kann mit den Dijkstra-Schritten aufgehört werden. Es muss nun sichergestellt werden, ob der gefundene Weg auch der kürzeste Weg ist. Dazu betrachtet man alle Kanten zwischen den Mengen der abgearbeiteten Knoten und prüft ob der erhaltenene Weg der kürzeste ist.

Der Algorithmus gehört zur Klasse der Greedy-Algorithmen und besitzt eine Komplexität von $O(n \log n)$, wobei n die Anzahl der Knoten im Graphen repräsentiert.

Abbildung 35: Intro Beschreibung

Dijkstra (Bidirektional)



Der kürzeste Weg von G nach F hat die Länge 12.

->	A	B	C	D	E	F	G	H
G	*	*	*	*	*	*	0	*
H	8	*	*	*	7	*	0	1
E	3	17	*	11	2	22	0	1
A	3	9	5	11	2	22	0	1
C								

<-	A	B	C	D	E	F	G	H
F	*	6	*	3	*	0	*	21
D	*	4	*	3	12	0	*	14
B	10	4	7	3	12	0	*	14
C	9	4	7	3	12	0	*	14

```
private Node n;

private boolean search(Queue queue, boolean dir) {
    n = queue.removeMin();
    n.setFinished(dir);
    if (n.isFinished(!dir)) {
        return false;
    }
    for(Edge e : n.getEdges(dir)) {
        Node v = e.getOther(n);
        if (v.notLabelled(dir)) {
            v.setDistance(dir, v.getDistance(dir) + e.distance);
            queue.add(v);
        } else if (v.getDistance(dir) > v.getDistance(!dir) + e.distance) {
            v.setDistance(dir, v.getDistance(!dir) + e.distance);
        }
    }
    if (queue.isEmpty()) {
        n = null;
        return false;
    }
    return true;
}

public int bidijkstra(Node start, Node target) {
    start.setDistance(true, 0);
    queueA.add(start);
    target.setDistance(false, 0);
    queueB.add(target);

    while (search(queueA, true) && search(queueB, false));
    if (n == null) {
        return -1;
    }
    return secureShortestPath(n.getDistance(true) + n.getDistance(false),
        queueA.getFinishedNodes(),
        queueB.getFinishedNodes());
}
```

Abbildung 36: Ende der Animation

9. Erkennen von Zyklen negativer Länge

Wenn Sie „Erkennen Negativer Zyklen“ ausgewählt haben, sollten Sie das Panel aus Abbildung 37 vor sich sehen.

Im Textfeld (**B**) erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von (**J**) erzeugen Sie diesen Teil der Animation. (**A**) zeigt, wie gewohnt, den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

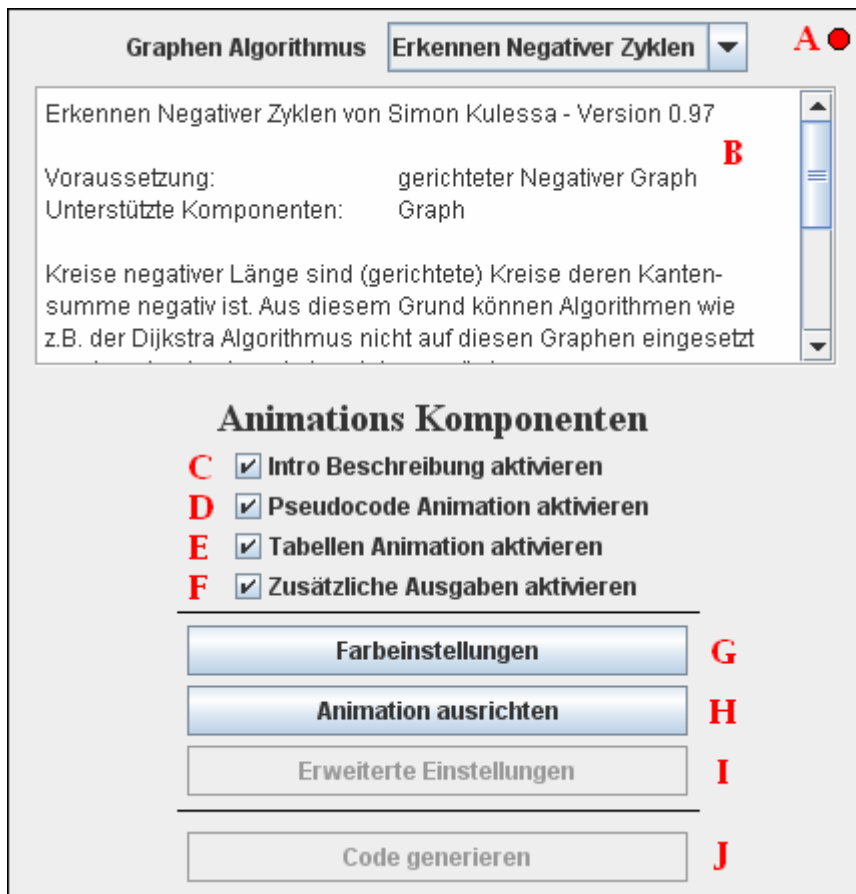


Abbildung 37: Erkennen von Zyklen negativer Länge

9.1 Voraussetzungen

Um einer „Erkennen von Zyklen negativer Länge“-Animation zu generieren, wird ein Graph des Typus „Negativer Graph“ benötigt. Die genaue Definition eines „Negativen Graphen“ finden Sie im Benutzer-Tutorial. Der Graph muss gerichtet sein.

Damit der Algorithmus generiert werden kann, muss in den „Erweiterten Einstellungen“ sowohl ein Startknoten als auch ein Zielknoten ausgewählt werden. Der „Erweiterte Einstellungen“ Button (**I**) wird aktiviert, sobald ein Graph geladen wurde.

Der „Code generieren“ Button (**J**) wird aktiviert, sobald alle Voraussetzungen erfüllt wurden.

9.2 Animationskomponenten

Die „Erkennen von Zyklen negativer Länge“-Animation unterstützt nur die Animation des Graphen. Als Subkomponenten stehen wie gewohnt die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Weiterhin ist eine Tabellen-Animation verfügbar, wobei die Tabelle die Vorgänger jedes Knoten sowie die aktuellen Labels enthält.

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

9.3 Farbeinstellungen

Durch Drücken von (**G**) öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Neben den üblichen Farbeinstellungen können Sie hier die Farben für Kanten im negativen Kreis und für Knoten in der Queue festlegen.

Eine genauere Beschreibung der Farbeinstellungen entnehmen Sie bitte Kapitel 2.3.

9.4 Animation ausrichten

Ein Druck auf (**H**) öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

9.5 Erweiterte Einstellungen

Durch Drücken von **(I)** öffnet sich ein Dialog, in dem Sie die erweiterten Einstellungen des „Erkennen von Zyklen negativer Länge“ - Algorithmus festlegen können. Sie haben die Möglichkeit zur Änderung der äußeren Traversierungsfolge (d.h. der Abarbeitung der Knoten von außerhalb des Graphen).

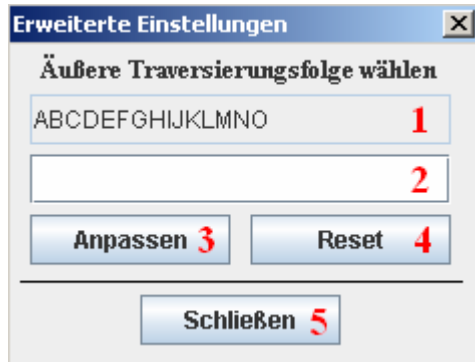


Abbildung 38: Erweiterte Einstellungen

(1) zeigt die aktuelle Traversierungsfolge. Standard ist das Abarbeiten der Knoten in alphabetischer Reihenfolge.

In das Textfeld bei **(2)** können Sie eine neue Abarbeitungsreihenfolge eingeben. Beachten Sie dabei, dass jede Kennzeichnung des Knotens genau einmal vorkommen muss.

Durch Drücken von **(3)** wird die von Ihnen eingegebene Traversierungsfolge überprüft. Sollte sie den Vorgaben entsprechen, wird sie nun bei **(1)** erscheinen, andernfalls erscheint bei **(2)** eine Fehlermeldung.

Durch Drücken des „Reset“-Buttons **(4)** wird die Standardabarbeitung (alphabetische Reihenfolge) wieder hergestellt.

9.6 Screenshots einer Beispielanimation

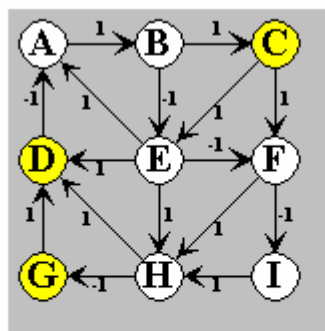
Erkennen Negativer Zyklen

Kreise negativer Länge sind (gerichtete) Kreise deren Kantensumme negativ ist. Aus diesem Grund können Algorithmen wie z.B. der Dijkstra Algorithmus nicht auf diesen Graphen eingesetzt werden, da sie niemals terminieren würden.

Zum Finden von Kreisen negativer Länge kann man einen Label-korrigierenden Algorithmus verwenden. Man merkt sich neben der Entfernung zum Ausgangsknoten auch den aktuellen Vorgängerknoten, wodurch ein Vorgängergraph (Predecessorgraph) aufgebaut wird. Sobald sich in diesem Vorgängergraph ein gerichteter Kreis befindet, muss es sich um einen Kreis negativer Länge handeln.

Abbildung 39: Intro Beschreibung

Erkennen Negativer Zyklen

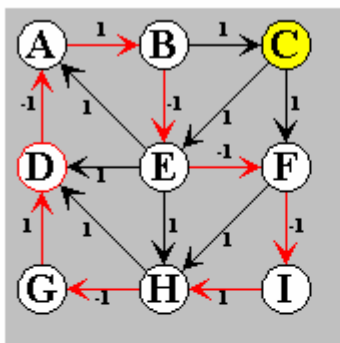


Node	A	B	C	D	E	F	G	H	I
Label	0	1	2	0	0	-1	-2	-1	-2
Predecessor	-	A	B	H	B	E	H	I	F

Knoten D wird in die Queue eingefügt.

Abbildung 40: Zwischenschritt der Animation

Erkennen Negativer Zyklen



Node	A	B	C	D	E	F	G	H	I
Label	0	1	2	-1	0	-1	-2	-1	-2
Predecessor	-	A	B	G	B	E	H	I	F

Ein Kreis negativer Länge wurde gefunden.

```

public boolean hasNegativeCycles() {
    for (Node n = graph.getNodes())
        n.set(INFINITY, null);

    boolean hasNegativeCycles = false;
    for (Node n = graph.getNodes() && !hasNegativeCycles) {
        if (n.label == INFINITY) {
            hasNegativeCycles = search(n);
        }
    }
    return hasNegativeCycles;
}

public boolean negativeCycles(Node start) {
    start.set(0, null);
    queue.add(start);
    while (!queue.isEmpty()) {
        Node n = queue.next();
        for (Edge e : n.getEdges()) {
            v = e.getDestination();
            if (v.label > n.label + e.weight) {
                if (isPredecessor(v, n)) {
                    return true;
                }
                v.set(n.label + e.weight, n);
                queue.add(v);
            }
        }
    }
    return false;
}
    
```

Abbildung 41: Ende der Animation

10. Preflow Push – Lösen von Flussproblemen

Wenn Sie „Preflow Push“ ausgewählt haben, sollten Sie das Panel aus Abbildung 42 vor sich sehen.

Im Textfeld **(B)** erhalten Sie einige Informationen zu dem gewählten Algorithmus und zu der Teilkomponente selber, sowie einige Hinweise zum Graph oder zu anderen Komponenten.

Durch Drücken von **(J)** erzeugen Sie diesen Teil der Animation. **(A)** zeigt, wie gewohnt, den Status der Komponente an (grün für erzeugt, rot für nicht erzeugt und blau für möglicherweise nicht mehr aktuell).

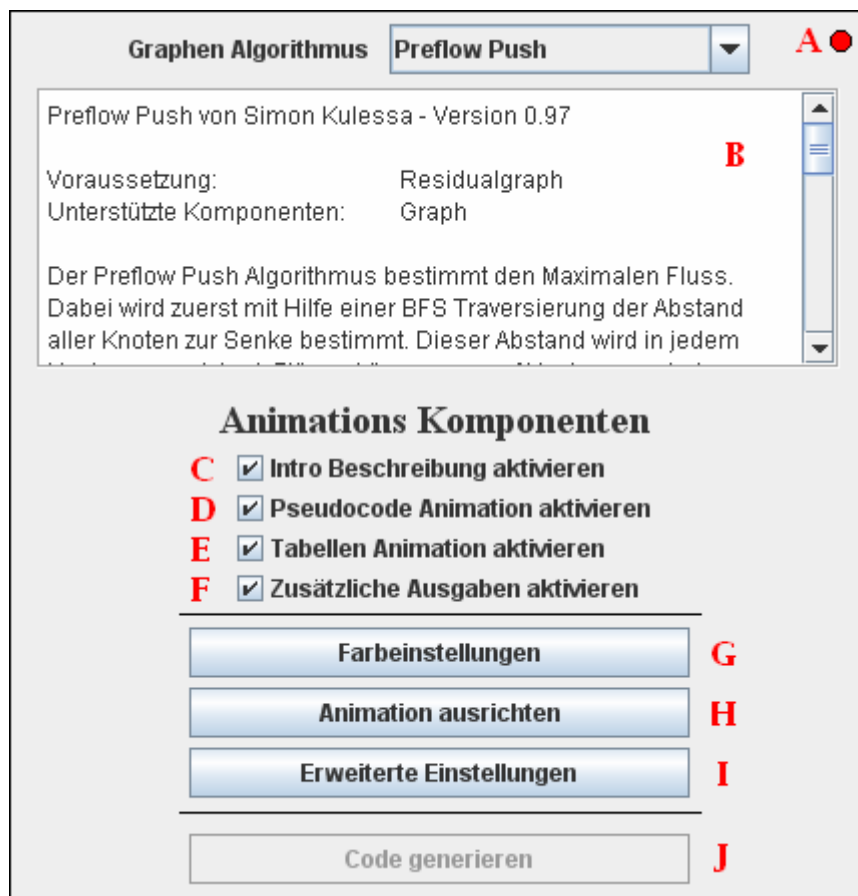


Abbildung 42: Preflow Push-Algorithmus

10.1 Voraussetzungen

Um einen „Preflow Push“-Animation zu generieren, wird ein Graph des Typus „Residual Graph“ benötigt. Die genaue Definition eines „Residual Graphen“ können Sie dem Benutzer-Tutorial entnehmen. Er sollte über wenigstens eine Quelle und eine Senke verfügen.

In Abbildung 42 ist zu erkennen, dass der „Code generieren“ Button (**J**) deaktiviert ist. Sobald das Programm die benötigten Voraussetzungen vorfindet, wird der Button aktiviert.

10.2 Animationskomponenten

Die „Preflow Push“-Animation unterstützt eine Animation des Graphen, die auch unbedingt ausgewählt werden sollte, da so der Fluss am besten zu erkennen ist. Die Animation der Matrix hingegen wird nicht unterstützt. Als Subkomponenten stehen die Intro Beschreibung, der Pseudocode und die „Zusätzlichen Ausgaben“ zur Verfügung.

Weiterhin ist eine Tabellen-Animation verfügbar. Dabei handelt es sich um zwei Tabellen, von denen die eine die Distanz und Excess Label der Knoten und die andere alle Kanten und ihren momentanen Fluss, sowie die Kapazität der Kante, enthält.

Eine genauere Beschreibung der Animationskomponenten entnehmen Sie bitte Kapitel 2.2.

10.3 Farbeinstellungen

Durch Drücken von (**G**) öffnet sich ein Dialog, mit dem Sie die Farbeinstellungen an Ihre persönlichen Wünsche anpassen können. Eine genauere Beschreibung der Farbeinstellungen entnehmen Sie bitte Kapitel 2.3.

10.4 Animation ausrichten

Ein Druck auf (**H**) öffnet einen Dialog zum Ändern der Ausrichtungs-Punkte der einzelnen Komponenten des Algorithmus. Eine genauere Beschreibung der Animationsausrichtung entnehmen Sie bitte Kapitel 2.4.

10.5 Erweiterte Einstellungen

Durch Drücken von **(I)** öffnet sich ein Dialog, in dem Sie die erweiterten Einstellungen des Preflow Push - Algorithmus festlegen können. Sie haben hier die Möglichkeit die Animation zu erweitern.

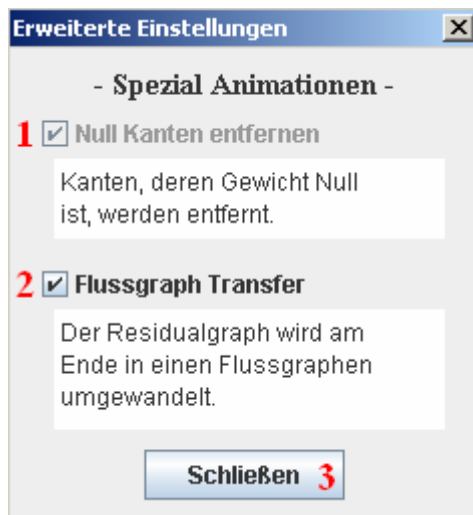


Abbildung 43: Erweiterte Einstellungen

Durch Drücken des „Reset“-Buttons **(3)** wird die Standardabarbeitung (alphabetische Reihenfolge) wieder hergestellt.

(1) ist derzeit noch deaktiviert. Kanten, deren Gewicht 0 ist, werden automatisch entfernt.

(2) aktiviert den Flussgraph Transfer. Dabei wird der Residualgraph am Ende der Animation in einen normalen Graphen transformiert. Dabei stellen die Gewichte an den Kanten den anliegenden Fluss dar.

10.6 Screenshots einer Beispielanimation

Preflow Push

Der Preflow Push Algorithmus bestimmt den maximalen Fluss. Dabei wird zuerst mit Hilfe einer BFS Traversierung der Abstand aller Knoten zur Senke bestimmt. Dieser Abstand wird in jedem Knoten gespeichert. Flüsse können nur auf Kanten verschoben werden, deren Startknoten eine höhere Entfernung zur Senke als der Zielknoten hat.

Begonnen wird der Algorithmus damit, dass soviel Fluss wie möglich in den Graphen hineingepumpt wird. Danach wird jeweils ein aktiver Knoten (d.h. ein Knoten in den mehr oder weniger hineinfließt als herausfließt - mit Ausnahme der Senke oder Quelle) ausgewählt.

Der Algorithmus benutzt nun zwei Operationen: Die Push Operation, wenn der Knoten eine geeignete Kante enthält oder die Relabel Operation, falls dem nicht so ist. Beim Relabeln wird das Label auf die minimalen Label der Nachbarknoten + 1 gesetzt.

Der Algorithmus terminiert wenn kein aktiver Knoten mehr übrig ist. Der dann festgestellte Fluss muss maximal sein. Die Komplexitätsklasse des Algorithmus ist $O(n^2 * m)$, wobei n die Anzahl der Knoten und m die Anzahl der Kanten beschreibt.

Anmerkung:
Die hier vorgestellte Variante verwendet eine FIFO Queue. Es werden keine Heuristiken zur Verbesserung angewandt.

Abbildung 44: Intro Beschreibung

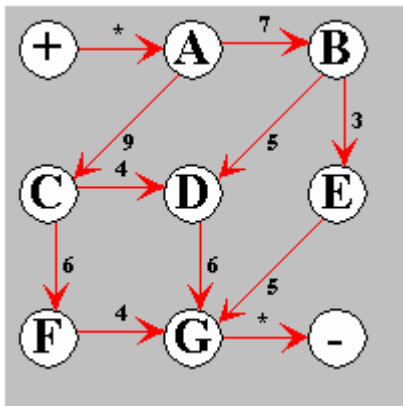


Abbildung 45: Anfangsgraph

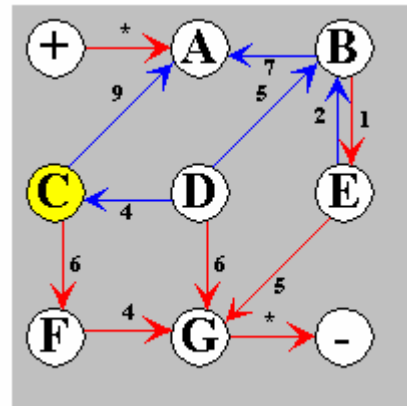


Abbildung 46: Zwischenschritt

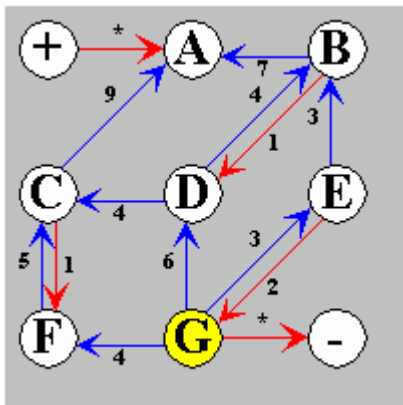


Abbildung 47: Zwischenschritt

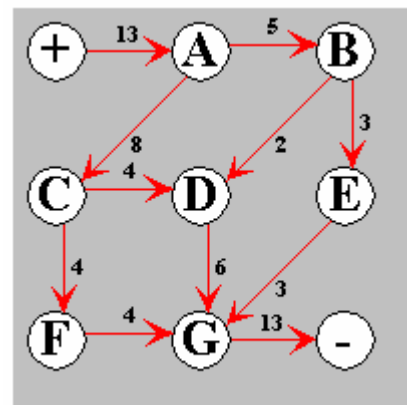


Abbildung 48: Flussgraph

Preflow Push

Ein gültiger maximaler Fluss von 13 Einheiten wurde bestimmt.

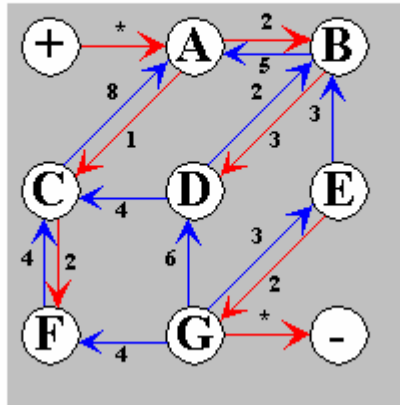


Abbildung 49: Maximaler Fluss

Node	+	-	A	B	C	D	E	F	G
Distance	0	0	10	11	11	12	2	12	1
Excess	0	0	0	0	0	0	0	0	0

Edge	Flow	Capacity
(+,A)	13	10000
(A,B)	5	7
(A,C)	8	9
(B,D)	2	5
(B,E)	3	3
(C,D)	4	4
(C,F)	4	6
(D,G)	6	6
(E,G)	3	5
(F,G)	4	4
(G,-)	13	10000

```

public int preflowPush(Node source, Node sink) {
    initDistanceLabels();
    pushInitialFlow();
    while (graph contains an active node) {
        select an active node v;
        if (v has an admissible arc(v, w) in D(x)) {
            push(arc(v, w), min(e(v), r_w));
        } else {
            relabel(v);
        }
    }
    return sink.getExcess();
}
    
```

Abbildung 50: Tabellen Animation und Pseudocode am Ende der Animation