Softwarepraktikum

Gyarmati Michael
9930772

1 Aufgabenstellung

Die Aufgabenstellung bestand darin, ein Java-Applet zu erstellen, das die Simulation von graphentheoretischen Algorithmen im Web-Browser ermöglicht. Dazu sollen Graphen live erstellt werden können aber es soll auch vorbereitete Beispielgraphen geben, die auf Knopfdruck geladen werden können.

Es sollte ferner möglich sein, weitere Algorithmen die auf Graphen basieren einfach einzufügen.

2 Programmbeschreibung

Das Applet besteht aus drei Hauptteilen, dem GUI, dem Graphen und dem Algorithmus.

2.1 GUI

Das GUI ist in der Datei „GraphenApplet.java“ enthalten, wo sich auch der Code zur Erzeugung und Verwaltung des Applets an sich befindet.

Das GUI hat folgende Funktionen zu erfüllen:

- Eingaben vom Benutzer
- Steuerung durch den Benutzer
- Ausgaben und Rückmeldungen für den Benutzer
- Darstellen des Graphen

Das GUI stellt alle verfügbaren Algorithmen zur Auswahl, ändert die Oberfläche dem aktuell ausgewählten Algorithmus entsprechend (z.B. indem nur gerichtete Graphen erlaubt werden und gleichzeitig der bestehende Graph in einen gerichteten umgewandelt wird) erlaubt die Konstruktion von Graphen (gerichtet/ungerichtet, gewichtet/ungewichtet und alle Kombinationen davon) und schließlich auch die Steuerung des Algorithmus.

2.2 Algorithmus

Jeder Algorithmus muss das Interface „Algorithm“ implementieren, das zur Kommunikation mit dem GUI benötigt wird.

Ein Algorithmus in diesem Applet ist ein Objekt, das die Funktionen aus dem „Algorithm“-Interface zur Verfügung stellt und entsprechende, spezielle Änderungen am übergebenen Graphen durchführt.

Das GUI ruft zunächst „load(g)“ auf und lädt damit den Graphen g in den Algorithmus. Danach kann der Algorithmus Schrittweise vor- („step“) und rückwärts („back“) durchgegangen werden. Weiters enthält „Algorithm“ die Methoden „reset“ und „solution“ um an den Anfang oder an das Ende der Simulation zu springen.

Es bleibt dem Programmierer überlassen, ob der eigentliche Algorithmus bereits in der „load“-Methode komplett durchgeführt wird, oder wirklich in Echtzeit und Schritt für Schritt. Es empfiehlt sich jedoch, den Algorithmus auf einmal im „load“ durchzurechnen, da dann das Rückwärtsnavigieren leichter fällt und bereits von Anfang an die notwendige Anzahl der Schritte bekannt ist und dem Benutzer angezeigt werden kann.

Neben diesen Methoden zur eigentlichen Simulation des Algorithmus muss eine Implementierung noch weitere Methoden mitbringen:

Wenn ein Algorithmus in der GUI ausgewählt wird, bringt das GUI über „getDefaultDirected“ und „getDefaultWeighted“ in Erfahrung, für welche Art von Graphen der Algorithmus gedacht ist. Der Rückgabewert ist dabei als zweistellige Binärzahl zu verstehen, wobei das LSB für false steht und das MSB für true. Gültige Werte sind 1, 2 und 3, wobei 1 für „muss falsch sein“, 2 für „muss wahr sein“ und 3 für „kann falsch oder wahr sein“ steht. Mit diesen Werten ändert das GUI den gegenwärtigen Graphen und stellt auch die Oberfläche entsprechend ein.

Ebenfalls bereits beim Auswählen eines Algorithmus wird „getExampleGraphCount“ aufgerufen, um die Anzahl der zur Verfügung stehenden Beispielgraphen in dem Algorithmus in Erfahrung zu bringen. Der Benutzer kann dann einen dieser Beispielgraphen auswählen, wodurch er vom GUI mit „getExampleGraph(x)“ herein geladen wird.

WICHTIG: Der Algorithmus verändert das originale Graph-Objekt des GUI. Es ist daher besonders darauf zu achten, dass nach dem Aufruf von „unload“ der Originalzustand wiederhergestellt wird.

Die Aufgabe des Algorithmus besteht darin, auf die Benutzereingaben zu reagieren und den Graphen entsprechend anzupassen. Wann die Berechnung geschieht und wie die Änderung des Graphen erfolgt ist dabei nahezu beliebig. Es ist sowohl möglich den Algorithmus in Echtzeit durchzuführen als auch ihn vorweg zu berechnen und eine Änderungsliste zu erstellen oder gar eine Liste von verschiedenen Graph-Objekten zu führen. Letztere Variante ist aufgrund größerem Speicherverbrauch nicht zu empfehlen, allerdings wäre sie sicher die einfachste (wenn der Graph klonbar wäre).

2.3 Graph

Das Graph-Objekt dient als Kommunikationsmittel zwischen Algorithmus und GUI und besteht aus Knoten (Vertex.java) und Kanten (Edge.java). Der Graph hat daneben ein paar Felder, die vom GUI benötigt werden, allerdings keine, die der Algorithmus verwenden könnte. Für den Algorithmus stellt der Graph nur ein Präsentationsobjekt, kein Arbeitsobjekt dar. Für aufwendige Algorithmen ist es zweckmäßig, im „load“ den Graph in eine eigene, spezialisierte Struktur einzulesen und auf der zu operieren.

Der Graph besitzt mehrere Methoden zum Bearbeiten und Suchen von Kanten und Knoten sowie eine „drawTo“-Methode, mit der er sich auf ein beliebiges „Graphics“ zeichnet.

2.4 Einfügen neuer Algorithmen

Ein neuer Algorithmus muss das Interface „Algorithm“ implementieren. Die Funktionen der einzelnen Methoden ist im JavaDOC ersichtlich.

Damit der Algorithmus im Applet aufscheint und verwendet werden kann, muss er an drei Stellen im Source-Code eingefügt werden und anschließend muss natürlich der Source-Code komplett neu kompiliert werden.

1. Anlegen einer neuen Konstante mit dem Namen des Algorithmus (optional)
   (siehe STEP_1 in GraphenApplet.java)
2. Einfügen des Algorithmus in die Auswahl-Listbox
   (siehe STEP_2 in GraphenApplet.java)
3. Erzeugen der Objektinstanz sofern der Algorithmus ausgewählt wird.
   (siehe STEP_3 in GraphenApplet.java)

3 Arbeitsplan / Arbeitsvorgang

Mein Plan sah vor, dass ich zunächst die wichtigen Konzepte des Applets, die ich identifiziert hatte, gleich in Java ausprobieren würde. Eine komplette Spezifikation machte in meinen Augen keinen Sinn, da es sich hierbei um mein erstes Java-Applet handelte und ich nicht wusste, was von meinen Konzepten und Ideen überhaupt umsetzbar ist.

Eine Beschreibung, wie ich vorgegangen bin, findet sich in den folgenden Abschnitten. Die Grundstruktur (Applet / GUI / Algorithmus / Graph / Knoten / Kante) hatte ich mir natürlich bereits vorher überlegt. Alles Weitere galt es aber erst auf seine Machbarkeit hin zu überprüfen.

3.1 GUI + Applet / Erste Schritte

Der erste Schritt bestand im Erstellen eines Applets und den notwendigen Panels. Die nächste Schwierigkeit bestand darin, auf die Oberfläche zu zeichnen, so dass dies erstens ohne Flackern erfolgt (Doppelbufferung) und zweitens auch sichtbar bleibt, wenn ein anderes Fenster darüber gezogen wurde. Als nächstes galt es sicherzustellen, dass auf Benutzereingaben in Form von Mausklicks, Mausbewegung und Tastatureingaben reagiert werden kann. Im vierten Schritt wurde die Funktionalität des Umschaltens von Konstruktions- in Simulationsmodus implementiert, was durch Anzeigen/Verbergen von Panels geschieht.

Nun wurde diese Minimalversion, die gerade einmal Gummibandlinien zeichnen konnte im Browser getestet und nachträglich auf Java 1.1 zurückgestutzt, da der Internet Explorer Java 1.2 nicht unterstützte.

3.2 Graph

Da als nächstes der Designer in Angriff genommen werden sollte, musste zunächst eine Repräsentationsform für das zu designende Objekt geschaffen werden. Die Wahl viel auf ein Objekt Graph das seinerseits aus Objekten Vertex und Edge besteht. Die Klassen Edge und Vertex dienen hauptsächlich dem Speichern und bringen deshalb kaum Methoden mit. Eine genaue Beschreibung und Auflistung der Felder und Methoden findet sich in der Java Dokumentation. Die Klasse Graph hat mehr Methoden, die vor allem dem Bearbeiten des Graphen dienen. Auch diese Felder und Methoden sind in der Java Dokumentation zu finden.

3.3 Designer

Als Designer bezeichne ich jenen Teil des Applets, der für das Bearbeiten und Anzeigen von Graphen zuständig ist. Zunächst musste dafür gesorgt werden, dass gegebene Graphen angezeigt werden können. Dazu baute ich einen fixen Graph ein, den ich ähnlich zu den später entstehenden Beispielgraphen im Source-Code erzeugte. Mit diesem Graphen wurden dann die günstigsten Parameter für die graphische Ausgabe ermittelt. Besonders schwierig gestaltete es sich, eine übersichtliche Repräsentation von gerichteten Kanten und ihren Pfeilen zu finden, da diese – sofern sie in beide Richtungen vorhanden sind – nicht mehr durch den Mittelpunkt der Knoten gelegt werden können.

Als nächstes galt es, die Konstruktion und Bearbeitung von Graphen zu implementieren. Als problematisch erwies sich das Anlegen von gewichteten Kanten. Ursprünglich hatte ich geplant, dass nach dem Angeben der beiden Eckpunkte ein Fenster kommt und nach dem Gewicht verlangt. Da ich dies im Java-Applet nicht bewerkstelligen konnte, änderte ich die Vorgangsweise. Das Gewicht sollte jetzt während dem Erstellen der Kante eingegeben werden und in einer Eingabebox am oberen Rand erscheinen. Das Gewicht wird dann mit dem Klick auf den Zielknoten automatisch mitbestätigt.

Zu diesem Zeitpunkt erschien es mir zweckmäßig, dass die Ausgabe auch vergrößert werden können sollte, weshalb ich die anfangs verwendeten Konstanten durch Variablen ersetzte und dem Benutzer eine Anpassung derselben ermöglichte.

3.4 Algorithmus-Interface

Da das Applet erweiterbar sein sollte, gestaltete ich den Designer so, dass er auch für gerichtete und ungewichtete Graphen geeignet ist. Weiters sieht die Erweiterbarkeit vor, dass sich alle Algorithmen an einen Standard halten. Diesen Standard habe ich in ein Interface verpackt.

3.5 Algorithmen

Schließlich galt es noch die eigentliche Funktionalität zu implementieren. Dazu wurde das GUI noch um die Möglichkeit der Algorithmenauswahl erweitert und die Steuerung derselben ausimplementiert. Dann habe ich den Kruskal-Algorithmus implementiert und sogleich neue Ideen erhalten. Zum Beispiel schien es sehr zweckmäßig, Beispielgraphen im Designer zu erzeugen und auf die Konsole als Java-Source-Code ausgeben zu lassen anstatt sie mühevoll selbst im Sourcecode zu implementieren. Weiters fand ich es praktisch zu wissen, wie viele Schritte der Algorithmus bis zur Beendigung benötigen würde. Auch die Hintergrundfarbe sollte nicht im Source-Code fix programmiert sein, sondern wenigstens im HTML-Code festlegbar sein. Damit man sich besser zurechtfindet wurden die Debug-Meldungen auch noch durch sinnvolle Hilfstexte ersetzt, die angeben, welche Optionen man zur Verfügung hat.

4 Algorithmen

4.1 Minimaler Spannbaum – Kruskal

Der Kruskal-Algorithmus benötigt in der vorliegenden Implementierung einen ungerichteten, gewichteten Graphen, dessen Kantengewichte jeweils größer als 0 sein müssen. Bereits bei der Initialisierung des Algorithmus (beim Hereinladen des Graphen) wird Kruskal bis zum Ende durchgerechnet und alle Änderungen am Graphen in einem Vektor gespeichert. Dadurch (bei Kruskal theoretisch auch generell) ist die Anzahl der notwendigen Schritte (= Anzahl der Kanten) bekannt. Navigiert der Benutzer nach vorne oder zurück so wird die vorher gespeicherte Änderung am Graphen vorgenommen, wobei hier jeweils nur eine Kantenfarbe betroffen ist.

Die Berechnung des Minimalen Spannbaumes nach Kruskal funktioniert folgendermaßen:

1. Sortiere die Kanten nach ihrem Gewicht
2. Initialisiere den Vector der im MSB enthaltenen Kanten (selectedEdges)
3. Gehe alle Kanten der Reihe nach in Richtung steigendem Gewicht durch:
   Prüfe ob ein Kreis entstehen würde, wenn die neue Kante in den MSB aufgenommen würde, indem ein alternativer Weg mit den Kanten in selectedEdges gesucht wird. Existiert dieser, so bildet die neue Kante einen Kreis und gehört nicht zum MSB. Existiert kein anderer Weg so gehört die Kante zum MSB dazu.

Damit das Verständnis des Algorithmus besser überprüft werden kann, habe ich eine zweite Version implementiert, die vor der eigentlichen Aufnahme oder Ablehnung einer Kante diese neutral einfärbt, damit das lästige Suchen nach der Kante mit dem nächst höheren Gewicht entfällt.

Farben:

Schwarze Kanten wurden noch nicht behandelt.
Rote Kanten sind im MSB enthalten.
Graue Kanten sind nicht im MSB enthalten.
Lila Kanten werden als nächstes bearbeitet (nur Kruskal 2).

Beschreibung der Ausgabe:

Der Algorithmus geht behandelt jede Kante und markiert sie, sofern „Kruskal 2“ geladen ist, zunächst lila und erst im nächsten Schritt grau oder rot, je nachdem, ob die Kante in den MSB gehört oder nicht.

4.2 Minimaler Spannbaum – Prim

Auch der Prim-Algorithmus benötigt in der vorliegenden Implementierung einen ungerichteten, gewichteten Graphen, dessen Kantengewichte jeweils größer als 0 sein müssen. Auch dieser Algorithmus wird bei der Initialisierung durchgerechnet und ein Vektor mit den Änderungen am Graphen gefüllt. Dadurch ist auch hier die Anzahl der notwendigen Schritte bekannt.

Die Berechnung des Minimalen Spannbaumes nach Prim funktioniert folgendermaßen:

1. Initialisiere den Komponentenvektor mit dem Startknoten
2. Solange es einen Knoten gibt, der noch nicht besucht wurde (visited == false) führe den folgenden Teil aus: (dies ist notwendig, damit nicht zusammenhängende Graphen komplett abgearbeitet werden.)

Wähle den Startknoten wenn dieser noch nicht besucht ist, sonst wähle einen anderen.
Füge alle adjazenten Knoten in die Queue hinzu
Solange die Queue nicht leer ist:
Nimm die Kante mit dem geringsten Gewicht heraus
Füge sie in den MSB hinzu
Überprüfe alle adjazenten Knoten:
Noch nie besucht: füge die Kante in die Queue ein
Bereits im MSB: tu nichts
sonst: wenn die neue Kante günstiger zum Knoten führt, als diejenige die in der Queue gespeichert ist, dann ersetze diese.

Farben:

Schwarze Kanten wurden noch nicht behandelt.
Rote Kanten sind im MSB enthalten.
Graue Kanten sind nicht im MSB enthalten.
Lila Kanten sind bereits in der Queue enthalten.

Der Knoten mit dem Blauen Rahmen ist der Startknoten. Immer der zuletzt bearbeitete Knoten erhält die blaue Umrahmung. Um einen anderen Knoten auszuwählen klicken Sie doppelt an.

Beschreibung der Ausgabe:

Der Algorithmus startet beim blau umrahmten Knoten und markiert zunächst alle Kanten, die von dort wegführen. Dies entspricht der Aufnahme der Zielknoten in die Queue. Ein künstlicher Determinismus wurde geschaffen, indem die Kanten nach ansteigendem Gewicht aufgenommen werden. Dies ist für die Korrektheit nicht nötig, da sowieso die Queue selbst sortiert ist.

Daraufhin wird die Kante mit dem kleinsten Gewicht ausgewählt und in den MSB aufgenommen, somit rot markiert. Im nächsten Schritt werden alle Kanten verarbeitet, die vom roten „Teilbaum“ aus jetzt neu erreichbar sind. Bildet sich dadurch ein Kreis so wird diejenige Kante aus der Queue entfernt, die das größere Gewicht hat und somit grau dargestellt, da diese Kante garantiert nicht Teil des MSB ist.

Die Ausgabe zeigt also folgende Schritte:

1. Queue füllen (lila)
2. Kante mit kleinstem Gewicht aus Queue nehmen und in MSB einfügen (rot)
3. Queue füllen / bearbeiten (lila / grau)
4. Gehe zu Schritt 2 solange noch ein Element in der Queue enthalten ist.

4.3 Kürzester Wege Spannbaum – Dijkstra

Dieser Algorithmus ähnelt sehr stark dem Prim-Algorithmus. Die hier verwendete Implementierung kommt mit ungerichteten und gerichteten, gewichteten Graphen zurecht, wobei das Kantengewicht jeweils größer 0 sein muss.

Der Hauptunterschied zum Prim-Algorithmus besteht nur darin, dass statt der kürzesten Entfernung zum wachsenden MSB hier immer die Entfernung zum Startknoten gemessen wird.

Ein weiterer Unterschied zum Prim-Algorithmus ist, dass bei Dijkstra das Neustarten bei nicht zusammen hängende Graphen keinen Sinn macht. Die Distanz von Knoten zu denen es keine Verbindung gibt ist und bleibt Unendlich. Beim Prim-Algorithmus kann man durch Wahl eines nicht behandelten Knotens Spannwälder wie mit Kruskal erzeugen.

Farben:

Schwarze Kanten wurden noch nicht behandelt (oder ausgespart).
Rote Kanten sind Teil des Kürzeste-Wege-Spannbaumes.
Graue Kanten sind nicht in demselben enthalten.
Lila Kanten sind bereits in der Queue enthalten.

Knoten:

Zusätzlich wird in den Knoten statt der Nummer des Knotens die Entfernung zum Startknoten angegeben. Diese ist anfangs „inf“ für Unendlich außer beim Startknoten, wo sie natürlich 0 ist.

Beschreibung der Ausgabe:

Die Ausgabe erfolgt wie beim Prim-Algorithmus, nur wird zusätzlich die kleinste bekannte Entfernung zum Startknoten in jedem Knoten angezeigt.

Anmerkung zu Prim / Dijkstra:

Obwohl der Unterschied zwischen den beiden Algorithmen nur gering ist, gibt es natürlich (manchmal) unterschiedliche Ergebnisse. Wobei sich das Konstruieren eines Graphen mit unterschiedlichem Ergebnis anfangs gar nicht so leicht darstellt.

Der Dijkstra-Algorithmus liefert in der Tat auch einen Spannbaum. Dieser ist jedoch nicht zwingend minimal, so wie der MSB den der Prim-Algorithmus liefert. Unterschiede kommen dann zustande, wenn der direkte Weg zu einem Knoten vom Startknoten aus betrachtet kürzer ist, als der indirekte. Dann wird Dijkstra den direkten Weg wählen, während Prim und Kruskal den indirekten bevorzugen.

Beispielgraph 4 des Dijkstra-Algorithmus demonstriert dies anhand eines einfachen Graphen. Es handelt sich um ein Quadrat mit den Kantengewichten 1,2,3 und 4, wobei an den Startknoten die Kanten mit den Gewichten 4 und 3 angrenzen. Ein Spannbaum besteht hier logischerweise aus genau drei Kanten, wobei der minimale die teuerste Kante (4) nicht enthält. Aufgrund der Anforderung den kürzesten Weg vom Startknoten zu jedem anderen

Knoten zu finden, muss Dijkstra über die Kante mit dem Gewicht 4 gehen, da das Gewicht entlang des Weges „außen herum“ 6 betragen würde.

Zur Verdeutlichung, dass Dijkstra nicht unbedingt einen MSB zurückliefert, gibt das Applet in der Statuszeile das Gesamtgewicht des Spannbaumes aus (allerdings nur, wenn der Benutzer auf „Solution“ klickt). Dieses Gesamtgewicht wird in entsprechender Weise auch bei Prim und Kruskal ausgegeben.

4.4 Kürzester Wege Spannbaum – Bellman-Ford

Graphen mit negativen Kantengewichten können mit Dijkstra nach einer Art Vorbearbeitung behandelt werden. Dabei wird eine konstante Zahl addiert, so dass alle Kantengewichte wieder größer Null werden.

Ist jedoch die direkte Verwendung von negativen Kantengewichten erwünscht, so kann der Bellman-Ford Algorithmus verwendet werden. In der einfachsten Implementierung benötigt er (|V|-1) * |E| Schritte. Die Anzahl der Kanten im längsten, kreisfreien Weg ist |V|-1, so dass nach |V|-1-maligem Durchprobieren aller Kanten der gesuchte Spannbaum gefunden ist. Allerdings nur dann, wenn der Graph keine negativen Kreise enthält.

Farben:

Schwarze Kanten wurden noch nicht behandelt.
Rote Kanten sind (soweit bisher bekannt) Teil des Kürzeste-Wege-Spannbaumes.
Graue Kanten sind nicht in demselben enthalten.
Lila Kanten werden gerade bearbeitet.

Ausgabe:

Der Algorithmus geht alle Kanten der Reihe nach durch, wobei die Reihenfolge pseudo-zufällig ist aber in den Durchgängen gleich bleibt. Die aktuell betrachtete Kante wird lila hervorgehoben und dann entsprechend rot oder grau dargestellt. Anders als bei den vorhergehenden Algorithmen kann sich die Einfärbung später noch ändern, da der Spannbaum erst am Ende gefunden ist, und die Zwischenschritte nur den aktuellen Stand darstellen.

Anmerkung:

Dieser Algorithmus ist zwar sehr einfach zu implementieren, aber nicht sehr effizient. Wie man beim Simulieren sieht, werden viele Kanten unnötigerweise mehrmals behandelt, obwohl sie aus der Liste längst herausgestrichen werden könnten, oder darin anfangs gar nichts verloren hätten. So macht die Behandlung aller Kanten, die nicht vom Startknoten wegführen anfangs keinen Sinn. Es ist auch zweifelhaft, ob es klug ist, Kanten immer und immer wieder zu überprüfen, die schon einmal als „nicht zum Spannbaum gehörig“ identifiziert wurden – oder ist dies doch nötig? – Eine Optimierung des Bellman-Ford-Algorithmus stellt der Bellman-Moore-Algorithmus dar.

Achtung: Ein ungerichteter Graph, der ein negatives Kantengewicht enthält, enthält auch einen negativen Kreis, so dass das Ergebnis nicht korrekt sein wird.

4.5 Kürzester Wege Spannbaum – Bellman-Moore

Das Grundprinzip bleibt dasselbe wie bei Bellman-Ford. Allerdings werden hier nicht generell alle Kanten betrachtet, sondern nur solche, die in einer „Kandidatenliste“ geführt werden. Am Anfang werden nur alle Kanten betrachtet, die vom Startknoten ausgehen. Alle anderen Kanten führen von Knoten aus, die mit Unendlicher Distanz initialisiert wurden, weshalb eine Betrachtung von denen noch keinen Sinn macht. In Folge wird jeweils ein Knoten aus der Kandidatenliste genommen und alle seine Nachbarknoten bearbeitet und in die Kandidatenliste eingefügt. Nach maximal |V|-1 Durchgängen wird der Algorithmus abgebrochen. Er kann aber auch früher terminieren, wenn die Kandidatenliste leer wird, man sozusagen in eine Sackgasse geraten ist.

Farben:

Schwarze Kanten wurden noch nicht behandelt.
Rote Kanten sind (soweit bisher bekannt) Teil des Kürzeste-Wege-Spannbaumes.
Graue Kanten sind nicht in demselben enthalten.
Lila Kanten werden gerade bearbeitet

Ausgabe:

Die Ausgabe erfolgt wie bei Bellman-Ford, es fällt hier jedoch auf, dass der Spannbaum wie bei Prim von der Wurzel zu wachsen beginnt, anstatt diffus wie bei Kruskal oder Bellman-Ford.

4.6 Suchbaum - Tiefensuche

Dieser Algorithmus kann auf gerichtete und ungerichtete Graphen angewendet werden, die jedoch nicht gewichtet sein dürfen.

Der Algorithmus beginnt beim Startknoten, geht dann sofort in die Tiefe, d.h. von einem Nachfolger zum nächsten, so lange, bis er in der untersten Ebene angelangt ist. Erst wenn alle Nachfolger eines Knotens abgearbeitet sind, geht der Algorithmus zum nächsten Knoten auf derselben Ebene (d.h. dem Knoten mit derselben Entfernung vom Startknoten) weiter.

Ausgabe:

Die gerade behandelte Kante wird lila dargestellt und anschließend rot oder grau eingefärbt, je nachdem, ob sie Teil des Spannbaumes/Suchbaumes ist oder nicht.

4.7 Suchbaum - Breitensuche

Dieser Algorithmus kann auf gerichtete und ungerichtete Graphen angewendet werden, die jedoch nicht gewichtet sein dürfen.

Der Algorithmus beginnt beim Startknoten, geht alle direkten Nachfolger durch und versucht dabei immer Ebene für Ebene weiter hinunter zu steigen. D.h. erst wenn alle Knoten, die x Schritte vom Startknoten entfernt sind abgearbeitet sind, geht der Algorithmus eine Ebene

tiefer zu den Knoten mit der Entfernung x+1. Der Baum wächst also gleichmäßig von der Wurzel bis zu den Blättern.

Ausgabe:

Der Algorithmus gibt die als nächstes betrachtete Kante lila zunächst aus, und färbt sie dann rot oder grau, je nachdem ob sie zum Spannbaum/Suchbaum gehört oder nicht.

Anmerkung: Der Algorithmus in dieser Implementierung betrachtet immer alle Nachbarn, also auch den, von dem er auf den aktuellen Knoten gekommen ist (Vater).

4.8 Designer Demo – Dummy Algorithmus

Dieser „Algorithmus“ dient lediglich dazu, dass der Designer (also das GUI) zur Gänze ausgetestet werden kann. Alle anderen Algorithmen beschränken die Funktionalität des Designers auf bestimmte Typen von Graphen (z.B. lässt Kruskal nur ungerichtete, gewichtete Graphen zu). Natürlich kann mit diesem „Algorithmus“ nichts simuliert werden.

5 Offene Probleme

5.1 Graph-Objekt ist nicht klonbar

In der gegenwärtigen Implementierung kann der Graph nicht geklont werden. Dies wäre für ein Backup vor der Simulation sinnvoll. Dann müsste der Algorithmus nicht darauf achten, dass wieder derselbe Zustand hergestellt wird wenn die Simulation beendet wird.

Bedienungsanleitung

1 Betriebsmodi

Es gibt zwei Betriebsmodi, die zu Anfang erklärt werden sollen:

1.1 Konstruktionsmodus

Dieser Modus ist aktiviert, wenn das Applet geladen wird. Hierin können (den Anforderungen des ausgewählten Algorithmus entsprechende) Graphen konstruiert werden.

Alternativ können auch Beispielgraphen (die vom Algorithmus abhängig sind) geladen werden, um das aufwendige Konstruieren zu umgehen.

Ist der gewünschte Graph geladen oder konstruiert kann in den Simulationsmodus gewechselt werden.

1.2 Simulationsmodus

Damit dieser Modus eingeschaltet werden kann, muss ein, für den ausgewählten Algorithmus, gültiger Graph vorhanden sein. Erst dann bewirkt ein Klick auf „Simulation“ den Wechsel in den Simulationsmodus. Durch Klick auf „Construction“ kann jederzeit wieder in den Konstruktionsmodus zurückgeschaltet werden.

Im Simulationsmodus kann der ausgewählte Algorithmus schrittweise ausgeführt bzw. visualisiert werden. Eine Play-Funktion mit einstellbarer Geschwindigkeit ermöglicht auch einen automatisierten Ablauf. Weiters kann der Ablauf des Algorithmus beliebig „vor- und zurückgespult“ werden. „Reset“ springt zurück auf Anfang, „Solution“ zeigt sofort die endgültige Lösung an.

2 Oberfläche

Die Oberfläche besteht aus vier wesentlichen Teilen (von oben nach unten):

1 Globale Einstellungen / Algorithmenwahl

2 Vom Modus und Algorithmus abhängige Kontrollelemente

3 Zeichenfläche

4 Statusleiste (des Applets)

2.1 Globale Einstellungen / Algorithmenwahl

Zuallererst sollte man den gewünschten Algorithmus in diesem Teil der Oberfläche auswählen. Nachdem die Auswahl getätigt ist, passt sich das Applet an die Anforderungen des ausgewählten Algorithmus an, deaktiviert beispielsweise einige Kontrollelemente.

Beachten Sie, dass der Algorithmus während der Simulation nicht geändert werden kann.

Hier befindet sich auch der Button, mit dem zwischen Simulations- und Konstruktionsmodus umgeschaltet werden kann. Ganz rechts befindet sich ein Kontrollelement, mit dem global die Größe angepasst werden kann. Dies dürfte vor allem für Präsentationen hilfreich sein, damit auch in der letzten Reihe noch alles erkennbar ist.

2.2 Vom Modus und Algorithmus abhängige Kontrollelemente

1.2.2.1 Konstruktionsmodus

Im Konstruktionsmodus finden sich hier die zum erzeugen eines Graphen notwendigen Kontrollelemente. Sie können vom Algorithmus abhängige Beispielgraphen laden, die Zeichenfläche löschen und Parameter für den Graph angeben. Ob ein Graph gerichtet bzw. gewichtet sein muss oder darf hängt vom Algorithmus ab, weshalb diese beiden Möglichkeiten nicht immer zur Verfügung stehen. Sofern der konstruierte Graph gewichtet ist, erscheint rechts neben „Weighted“ eine Eingabebox für das gegenwärtig verwendete Kantengewicht. Siehe Abschnitt *Konstrukion*.

1.2.2.2 Simulationsmodus

Im Simulationsmodus kann der Algorithmus gesteuert werden. Neben der Geschwindigkeit, mit der das automatische Ausführen („Play“) ablaufen soll kann auch manuell an den Anfang, ans Ende und dazwischen Schritt für Schritt vor und zurück geblättert werden. Ganz rechts werden in einer Box der aktuelle Ausführungsschritt und die Anzahl der notwendigen Schritte bis zum Ende angezeigt.

2.3 Zeichenfläche

Die Zeichenfläche entspricht der gesamten hellen Fläche des Applet und wird hauptsächlich mit der Maus bedient. Wie dies genau geschieht steht im folgenden Abschnitt *Konstruktion*.

Im Simulationsmodus kann der Graph nicht verändert werden, mit der Ausnahme, dass weiterhin die Knoten verschoben werden können.

2.4 Statusleiste des Applets

In der Statusleiste werden Hinweise, Anleitungen und Fehlermeldungen, weshalb Sie einen Blick darauf werfen sollten, wenn sich das Applet nicht wie erwartet verhält.

3 Konstruktion

Die Konstruktion eines Graphen ist nur im Konstruktionsmodus möglich. Dieser ist beim Starten des Applets automatisch aktiviert. Vom Simulationsmodus gelangen Sie durch Klick auf „Construction“ wieder in den Konstruktionsmodus.

3.1 Auswahl des gewünschten Graphentyps

Wählen Sie zuerst den Typ des gewünschten Graphen aus, sofern dies der ausgewählte Algorithmus gestattet. Wenn Sie dies nachträglich ändern, gehen eventuell einige Eigenschaften Ihrer Konstruktion

verloren. Beispielsweise werden beim Umschalten von „gerichtet“ auf „ungerichtet“ alle doppelten Kanten entfernt, so dass beim erneuten Umschalten auf „gerichtet“ einige Kanten fehlen werden.

Ungewichtete Kanten werden beim Umschalten auf „gewichtet“ mit dem Wert 0 belegt. Umgekehrt bleiben die Gewichte beim Umschalten auf einen ungewichteten Graph intern erhalten.

3.2 Anlegen eines Knotens

Klicken Sie mit der **linken** Maustaste auf eine beliebige, aber freie Stelle auf der Zeichenfläche. Das Applet wird einen neuen Knoten einfügen und ihn mit einer fortlaufenden Nummer kennzeichnen. Wenn Sie Knoten löschen werden Nummern nicht wieder verwendet. Allerdings fängt das Applet wenn Sie „Clear“ betätigen wieder bei 0 an.

3.3 Verschieben eines Knotens

Klicken Sie mit der **rechten** Maustaste auf einen beliebigen Knoten und lassen Sie die Maustaste wieder los. Der Knoten ändert seine Farbe und symbolisiert damit, dass er sich im Verschiebemodus befindet. Wenn Sie jetzt die Maus bewegen, wird er dieser folgen. Zeigen Sie mit der Maus auf den gewünschten Zielpunkt und klicken Sie mit einer beliebigen Maustaste. Das Verschieben des Knotens ist damit abgeschlossen und er hat wieder die normale Farbe. Alle Kanten die vom Knoten ausgehen oder zu ihm hingehen werden natürlich „angepasst“.

3.4 Löschen eines Knotens

Klicken Sie mit der **linken** Maustaste auf einen beliebigen Knoten und lassen Sie die Maustaste wieder los. Der Knoten ändert seine Farbe. Jetzt drücken Sie die **Entfernen**-Taste auf Ihrer Tastatur um den Knoten mit allen einmündenden und ausgehenden Kanten zu löschen. Wenn Sie den Knoten versehentlich angeklickt haben, klicken Sie ihn erneut an, um diesen Modus zu verlassen. Wenn Sie die Maus vom Knoten wegbewegen, werden Sie eine Linie feststellen. Diese wird für das Erzeugen von Kanten zwischen Knoten benötigt.

3.5 Erzeugen einer Kante

Eine Kante verbindet zwei Knoten. Damit Sie folglich eine Kante erzeugen können, müssen Sie zuerst mindestens zwei Knoten erzeugt haben. Eine Kante kann ein Gewicht und eine Richtung haben. Dies hängt vom ausgewählten Graphentyp ab (gerichtet, gewichtet, ungewichtet-gerichtet, etc.). Den aktuellen Typ ersehen Sie an „Directed“ und „Weighted“ oberhalb der Zeichenfläche. Beachten Sie, dass diese beiden Eigenschaften in erster Linie vom ausgewählten Algorithmus bestimmt werden und nur dann von Ihnen gesetzt werden können, wenn der Algorithmus mit beiden Typen zurechtkommt.

Um eine ungerichtete Kante zu zeichnen, klicken Sie mit der **linken** Maustaste auf einen beliebigen der beiden Knoten, die sie miteinander verbinden möchten und lassen die Maustaste wieder los. Der Knoten hat seine Farbe geändert, so wie weiter oben auch beim Löschen beschrieben. Natürlich könnten Sie ihn jetzt auch löschen.

Sofern es sich um eine gewichtete Kante handelt, müssen Sie jetzt das Kantengewicht eingeben. Dazu stehen Ihnen die Zahlen 0 bis 9 und sowie die Vorzeichen „+“ und „-„ und die Backspace-Taste zur Verfügung. Das eingegebene Kantengewicht wird im Feld neben „Weighted“ angezeigt. Beachten Sie, dass die Entfernen-Taste ohne Rückfrage den Knoten löscht! Das Vorzeichen des Gewichtes können Sie jederzeit mit den Tasten „+“ und „-„ ändern.

Wenn Sie die Maus bewegen, werden Sie feststellen, dass eine Gummibandlinie nachgezogen wird. Diese entspricht der Kante, die Sie gerade anlegen. Sobald Sie mit der Maus auf den Zielknoten Ihrer

Kante zeigen, wird sich dieser ebenfalls verfärben. Klicken Sie ihn mit einer beliebigen Maustaste an, um die Kante skizzierte anzulegen.

Sollten Sie doch keine Kante anlegen wollen, klicken Sie irgendwohin auf die Zeichenfläche oder auf den Startknoten. Wenn die Kante wider Erwarten nicht angelegt wird, beachten Sie bitte die Statuszeile, um den Grund dafür zu erfahren.

Eine gerichtete Kante zeichnen Sie vom Schaft zur Spitze, also vom Startknoten zum Zielknoten. Ansonsten verfahren Sie genau gleich wie bei ungerichteten Kanten.

3.6 Bearbeiten eines Kantengewichtes

Sie können das Kantengewicht einer bestehenden Kante überschreiben, indem Sie diese Kante erneut, jedoch mit einem anderen Gewicht, anlegen. Dadurch wird das Gewicht der alten Kante durch das neue Gewicht ersetzt.

3.7 Löschen einer Kante

Zum Löschen einer Kante verfahren Sie gleich wie beim Anlegen, geben aber kein Gewicht an, sofern es sich um eine gewichtete Kante handelt. Eine ungewichtete Kante wird gelöscht, wenn Sie sie doppelt anlegen.

Achten Sie bei gerichteten Kanten darauf, dass Sie dir Knoten in der richtigen Reihenfolge von der Spitze zum Schaft anklicken.

4 Simulation

Im Simulationsmodus können Sie den ausgewählten Algorithmus für den konstruierten Graph schrittweise ausführen und auch zurückblättern. Die Schaltfläche „Play“ führt den Algorithmus automatisch vorwärts aus und verwendet dabei die daneben eingestellte Geschwindigkeit von standardmäßig einem Schritt pro Sekunde. Wenn Sie den Regler nach links bewegen wird der Vorgang verlangsamt, wenn Sie ihn nach rechts bewegen beschleunigt. Ist der Algorithmus am Ende angelangt, stoppt er automatisch. Klicken Sie erneut auf „Play“ beginnt der Algorithmus wieder am Anfang. Mit Klick auf „Stop“ können Sie den Algorithmus jederzeit anhalten.

Wenn Sie den Simulationsmodus durch Klick auf „Construction“ verlassen, wird der Algorithmus angehalten und der Graph wieder in seinen Ursprungszustand zurückgesetzt.