Graph Visualisierung 2: vistra

Didaktische Desktop-Applikation zur Visualisierung von Algorithmen und Datenstrukturen

Projekt-Dokumentation, Revision 1.9

Studiengang:	Informatik, Modul BTI-7301 Projekt 1, HS 2013/14
Autor:	Roland Bruggmann (brugr9@bfh.ch)
Betreuer:	Dr. Jürgen Eckerle, juergen.eckerle@bfh.ch
Datum:	17.1.2014

Versionen

Version	Datum	Status	Bemerkungen
1.0	04.10.2013	Inception	Requirements: Vision
1.1	11.10.2013	Inception	Project Management
1.2	18.10.2013	Inception	Requirements: Vision; Project Management
1.3	25.10.2013	Inception	Requirements, Use Cases Model: Actors, Diagram, Brief Format
1.4	01.11.2013	Inception	Requirements, Use Cases Model: Fully Dressed Format
1.5	08.11.2013	Elaboration	Design: Domain Model, Design Model
1.6	15.11.2013	Implementation	
1.7	22.11.2013	Implementation	
1.8	20.12.2013	Implementation	
1.9	17.01.2014	Presentation	

Inhaltsverzeichnis

1 Requirements **1**
1.1 Vision . . . 1
1.2 Supplementary Specification . . . 3

2 Project Management **5**
2.1 Time Management . . . 5
2.2 Object Oriented Analysis and Design . . . 5
2.3 Development Environment Description . . . 5

3 Design **7**
3.1 Domain Model . . . 7
3.2 Design Model . . . 10

4 Solution **15**

1 Requirements

1.1 Vision

Es soll eine Software erstellt werden, welche das Traversieren von Graphen mit verschiedenen Algorithmen darstellen kann.

Ein beliebiger Algorithmus, wie etwa derjenige von Dijkstra soll mit diesem Werkzeug auf einfache Weise visualisiert werden. Das Werkzeug soll sich als didaktisches Hilfsmittel eignen. Neue Algorithmen sollen ohne grossen Aufwand hinzugefügt werden können. Zudem soll die Software Graphen aus einer Datei importieren können.

1.1.1 Problem Statement

Graphen

Das System soll ungerichtete, ungewichtete und (positiv) gewichtete einfache Graphen mit einfachen Kanten (simple connected graphs, neither self-loops nor parallel edges) verarbeiten können.

Ein Graph wird durch Kreise (Knoten), Geraden (ungerichtete Kanten), Pfeile (gerichtete Kanten) und Beschriftungen (Knotenbezeichnungen, Kantenbezeichnungen und Kantengewichte) dargestellt. Das System kann folgende Dateioperationen ausführen:

- Es können neue ungerichtete und gerichtete Graphen erstellt werden.
- Es können als Datei vorliegende Graphen geöffnet werden.
- Graphen können bearbeitet werden.
  - Die Anordnung der Knoten kann verändert werden: Diese können mit der Maus verschoben werden.
- Graphen können gespeichert werden.

Algorithmen

Das System soll mindestens folgende implementierte Algorithmen zur Auswahl bereit halten:

- Rekursive Tiefensuche (Depth-First Search, DFS)
- Breitensuche (Breadth-First Search, BFS)
- Dijkstra: Suchen der Distanzen zwischen einem als Start festgelegten Knoten und der weiteren Knoten in einem ungerichteten, gewichteten Graphen
- Kruskal: minimaler Spannbaum berechnen (Spanning Tree)

Es sollen nur diejenigen Algorithmen auswählbar sein, die auf den aktuellen Graph-Typ angewendet werden können.

Für manche Algorithmen soll ein Startknoten angegeben werden können.

Ein Entwickler von Algorithmen soll auf möglichst einfache Weise weitere Algorithmen implementieren und dem System hinzufügen können.

Traversierung

- Die Visualisierung der Traversierung soll Schrittweise erfolgen ('Step-by-step'): Dabei soll der Benutzer vor, zurück, zum Anfang oder zum Ende der Visualisierung gelangen können.
- Die Schrittlänge soll verändert werden können.
- Die Visualisierung der Traversierung soll auch abgespielt werden können (Animation). Dabei soll der Benutzer die Animation starten, pausieren oder stoppen können.
- Das Tempo der Animation soll verändert werden können.

1.1.2 Other Requirements and Constraints

- Zu öffnende Graphen werden validiert.
- Ein Entwickler von Algorithmen muss ein gegebenes Interface implementieren.
- Für ungewichtete Graphen wird ein Kanten-Gewicht der Grösse 1 angenommen, es wird also nicht unterschieden zwischen ungewichtet und gewichtet.
- Die Visualisierung der Traversierung zeigt Schrittweise Farbänderungen von Knoten und Kanten, evtl. auch errechnete Zwischenergebnisse.
- Mit jedem Step der Visualisierung wird auf einer Protokollpanele eine Statusmeldung ausgegeben, die die begangenen Traversierungsschritte erläutert.

1.2 Supplementary Specification

Platform: Das System soll auf verschiedenen Betriebssystemen lauffähig sein, im minimum auf GNU/Linux, Mac OS und Microsoft Windows.

I18n: Das System soll in den drei Sprachen Deutsch, Französisch und Englisch bedienbar sein.

2 Project Management

2.1 Time Management

Das Modul BTI-7301 Projekt 1 startete mit Beginn des Herbstsemester 2013/14. In der Woche 38 wurden die Projekte durch die Dozenten vorgestellt. Es wurden Teams gebildet und den den Projekten bzw. den Dozenten zugeteilt. Ein erstes Treffen des zuständigen Dozenten mit dem Team fand statt und erste Vereinbarungen wurden getroffen. Der Zeitplan gliedert sich nun wie folgt in vier Phasen:

Phase 1: Projektplanung und Systemarchitektur Wochen 39/40/41(/42) 2013 (3-4 Wochen)

- Einarbeiten in die Thematik (Graphen, Algorithmen, ...)
- Erstellen der Requirements, Spezifikation
- Design der Systemarchitektur

Phase 2: Wöchentliche Sprintzyklen Wochen (42/)43 bis 51 2013 (9-10 Wochen)

- Use Case wählen für nächsten Sprint
- (Re-)Design der Interfaces und Klassenhierarchie, Test und Implementation
- Dokumentation

Phase 3: Projektabschluss Wochen 52 2013 und 01 2014 (2 Wochen)

- Dokumentation abschliessen
- Erstellen der Präsentation

Phase 4: Präsentation des Projektes Wochen 02 und 03 2014 (2 Wochen)

- Besprechen des Ablaufes der Präsentation
- Checklisten Medien und Geräte
- Präsentation

2.2 Object Oriented Analysis and Design

Die Entwicklung des Projektes erfolgt objektorientiert und wird laufend dokumentiert. Die Elaboration der Komponenten und der Aufbau der Dokumentation richten sich nach C. Larman [**?**].

2.3 Development Environment Description

2.3.1 Programming Language and Libraries

Das System wird in der Programmiersprache Java implementiert. Es werden das Java Universal Network/Graph Framework JUNG [**?**], das XML-basierte Format GraphML [**?**], die Apache Commons IO Library und das Java Swing Framework verwendet. Als integrierte Entwicklungsumgebung kommt die Software Eclipse IDE zum Einsatz.

2.3.2 Sourcecode Management

Für das Sourcecode Management (SCM) wird die Software git verwendet, das Projekt wird auf der webbasierten Plattform *github* gehalten. In der Woche 45 wurde entschieden, das Projekt als Maven-Projekt zu halten. In der Woche 51 wurde entschieden, dass die Teammitglieder je ein Projekt alleine finalisieren.

- Ab Woche 39: `https://github.com/brugr9/ch.bfh.bti7301.hs2013.GraphVisualisierung2`
- Ab Woche 45: `https://github.com/brugr9/gravis`
- Ab Woche 51: `https://github.com/brugr9/vistra`

Die Vorgänger-Projekte sind weiterhin bis mindestens nach Erhalt der Modul-Bewertung erreichbar.

2.3.3 Team-split Woche 51

1. Mein Projekt heisst nun vistra.
2. Ich habe den von Patrick Kofmel geschriebene Code nun selber neu implementiert sowie Teile umgeschrieben (gravis gui.view.component.viewer), damits auf meine Implementation passt.
3. Ich habe auf github ein eigenes git-repository aufgesetzt.
    - Gitweb: `https://github.com/brugr9/vistra`
    - Klonen: git clone https://github.com/brugr9/vistra
4. Dokumentation: Der Code ist sowohl mit Javadoc (GraphVisualisierung2/doc/vistra/javadoc) als auch mit Doxygen (GraphVisualisierung2/doc/vistra/doxygen) dokumentiert.

3 Design

3.1 Domain Model

3.1.1 Domain Model Diagram

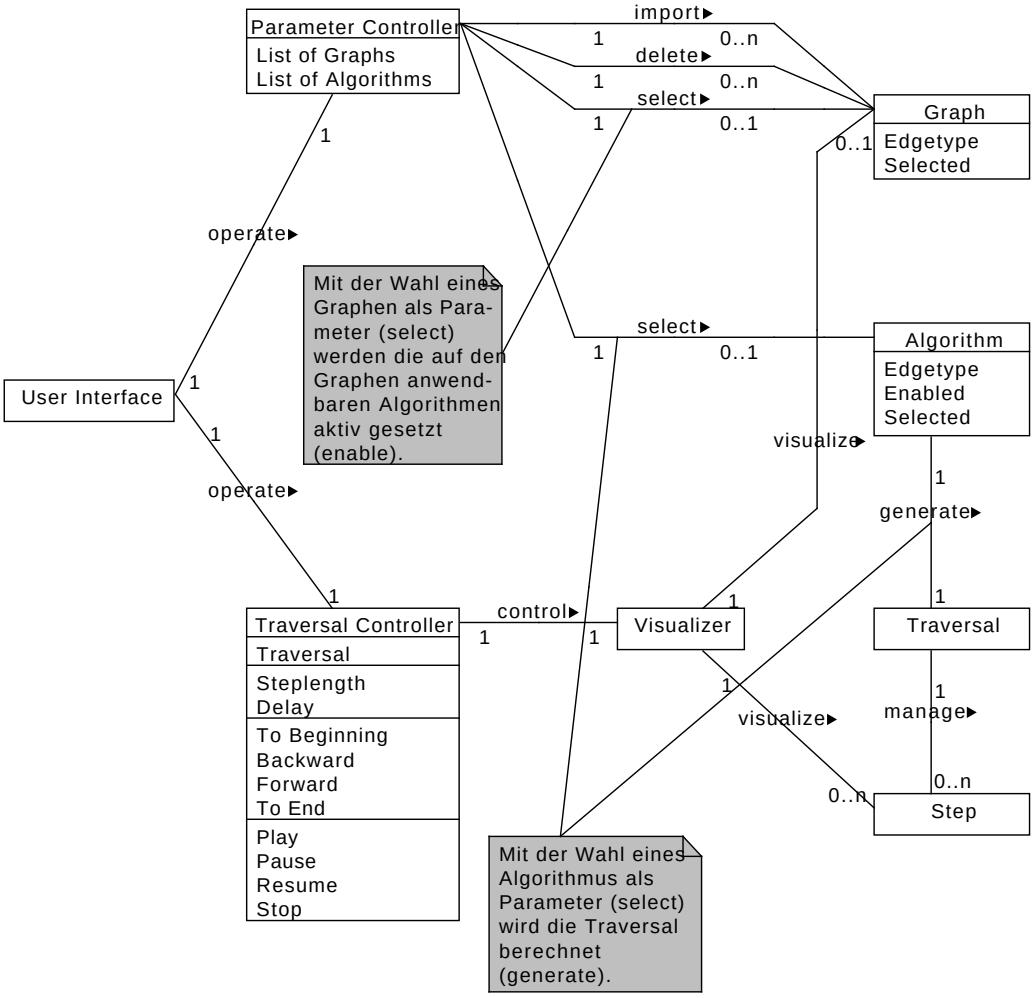

Abbildung 3.1: Domain Model Diagram

3.1.2 Domain Model Description

Es folgt eine Beschreibung der Konzeptklassen (conceptual classes) mit Assoziationen, wie im Domain Model Diagram gezeigt. Multiplizitäten und Attribute werden in Klammern angegeben.

User Interface

- Über ein User Interface (1) kann ein User einen Parameter Controller (1) bedienen (*operate*).
- Über ein User Interface (1) kann ein User einen Traversal Controller (1) bedienen (*operate*).

Parameter Controller

- Ein Parameter Controller verwaltet die zur Traversal benötigten Parameter Graph und Algorithm (*List of Algorithms*).
- Per Parameter Controller (1) kann ein User einen Graphen (0..n) des Formates *.vistra (GraphML) Ã¶ffnen (*import*).
- Mit der Wahl eines Graphen (1) als Parameter werden die auf den Graphen anwendbaren Algorithmen (0..n) aktiv gesetzt (*enable*).
- Ein Parameter Controller hält eine gegebene Anzahl Algorithmen als Vorlagen bereit.
- Per Parameter Controller (1) kann ein User einen Algorithmus (0..1) als Parameter auswählen (*select*).

Graph

- Ein Graph hat ungerichtete oder gerichtete Kanten (*Edgetype*).
- Ein Graph kann ausgewählt werden (*Selected*).

Algorithm

- Ein Algorithm kann Graphen mit ungerichteten Kanten (*Edgetype*) verarbeiten.
- Ein Algorithm (1) generiert eine Traversal (1) (*generate*).
- Ein Algorithm (*Enabled*) kann ausgewählt werden (*Selected*).
- Mit der Wahl eines Algorithm (1) als Parameter (*Selected*) wird eine Traversal (1) erstellt (*generate*).

Traversal

- Die Traversierung ist ein visualisierbares Objekt.
- Eine Traversal (1) kann einen oder mehrere Steps (0..n) verwalten (*manage*).

Step

- Ein Step ist ein Schritt der Traversierung.

Traversal Controller

- Ein Traversal Controller (1) steuert einen Visualizer (1) (*control*).
- Der Traversal Controller hält eine Traversierung (*Traversal*).
- Per Traversal Controller kann ein User die Zeit zwischen zwei Bildern (*Delay*) einstellen.
- Per Traversal Controller kann ein User Step-by-Step-Elemente (*Forward, Backward, To Beginning, To End*) bedienen.
- Per Traversal Controller kann ein User Animations-Elemente (*Play, Pause, Resume, Stop*) bedienen.

Visualizer

- Ein Visualizer (1) kann einen Graphen (0..1) visualisieren (*visualize*).
- Ein Visualizer (1) kann einen oder mehrere Steps (0..n) visualisieren (*visualize*).

3.2 Design Model

Die Systemkomponenten sind in zwei Gruppen unterteilt: *Framework* und *App*.

3.2.1 Framework

Die Komponente Framework ist zuständig für:

- Handling des Graphen mit Knoten und Kanten (I/O: Öffnen, Editieren, Speichern)
- Handling der Algorithmen
- Traversierung mit Steps als visualisierbare Lösung
- ParameterManager als Schnittstelle, welche der Komponente App zur Verfügung steht
- Die Komponente hält die für die Implementation eines Algorithmus zu verwendende Schnittstelle bereit.

vistra.framework.adt

Scope: Graph, Algorithm, Traversal.

Folgendes Konzept wurde implementiert:

- Ein Element (Item) eines Graphen befindet sich stets in einem Zustand (ItemState): unexplored, visited etc.
- Der Zustand wird durch das Item selber verwaltet: Ein Item ist ein Element-Zustands-Handhaber (ItemState-Handler), implementiert als State-Pattern.
- Die Zustandsänderung wird mit dem Aufruf eines Kommandos (ItemStateCommand) erreicht.
- Ein einziges oder mehrere Kommandos auf eines oder mehrere Graph-Elemente werden als Schritt (Step) zusammengefasst.
- Ein einziger oder mehrere Schritte zusammen ergeben eine Traversierung (Traversal).

Term	State			Command		
	ItemState	ItemStateHandler		ItemStateCommand	Step	
Edge	EdgeState	EdgeStateHandler		Edge	Both Item	Edge
Unexplored	UnexploredEdgeState	IItemStateHandler	handleUnexplored	–	–	–
Visited	VisitedEdgeState		handleVisited	VisitedEdgeCommand	VisitedStep	–
Solution	SolutionEdgeState		handleSolution	SolutionEdgeCommand	SolutionStep	–
Back	BackEdgeState	IEdgeStateHandler	handleBack	BackEdgeCommand	–	BackEdgeStep
Forward	ForwardEdgeState		handleForward	ForwardEdgeCommand	–	ForwardEdgeStep
Cross	CrossEdgeState		handleCross	CrossEdgeCommand	–	CrossEdgeStep
Discarded	DiscardedEdgeState		handleDiscarded	DiscardedEdgeCommand	–	DiscardedEdgeStep
Vertex	VertexState	VertexStateHandler		Vertex		Vertex
Unexplored	UnexploredVertexState	IItemStateHandler	handleUnexplored	–	–	–
Visited	VisitedVertexState		handleVisited	VisitedVertexCommand	VisitedStep	–
Solution	SolutionVertexState		handleSolution	SolutionVertexCommand	SolutionStep	–
Initialised	InitialisedVertexState	IVertexStateHandler	handleInitialised	InitialisedVertexCommand	–	InitialisedVertexStep
Updated	UpdatedVertexState		handleUpdated	UpdatedVertexCommand	–	UpdatedVertexStep

Abbildung 3.2: From state to step: Zustände von Kanten und Knoten (ItemState) und Zustandsänderungen (Step)

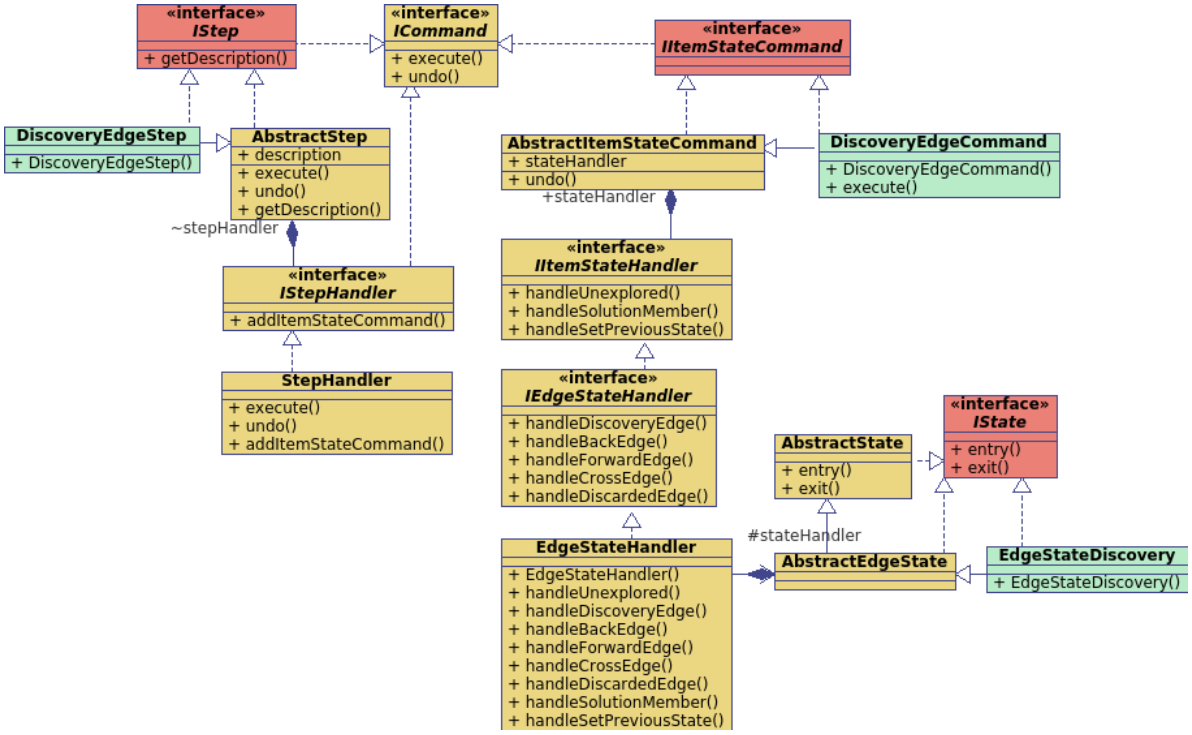

Abbildung 3.3: From step to state: Implementation einer Kanten-Zustandsänderung als Step

Ein Algorithmus erhält über seine execute-Methode als Parameter einen traversierbaren Graphen (ITraversableGraph). Das Interface eines traversierbaren Graphen sieht mehrere (überladene) Step-Methoden vor. Die Implementation einer solchen Step-Methode erstellt einen Step und fügt diesen einem bidirektionalen Iterator hinzu, welcher in einem Traversal-Objekt gehalten wird.

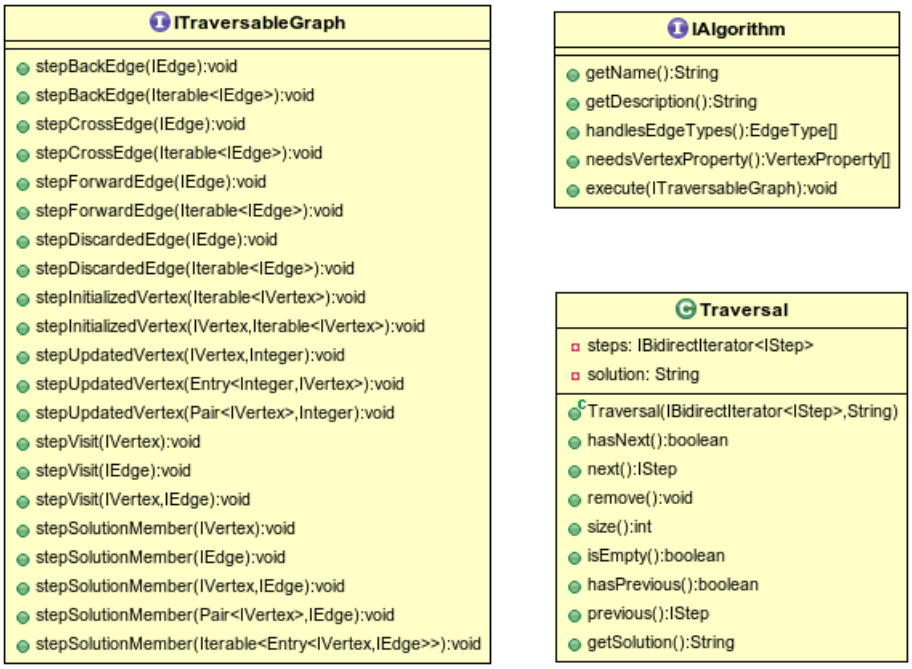

Abbildung 3.4: Die Schnittstellen IAlgorithm und ITraversableGraph und die Klasse Traversal

vistra.framework.io

Scope: read/write GraphML.

Das Dateiformat *.vistra ist eine Erweiterung des XML-Dialektes GraphML.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns/graphml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml">
<key id="coord.x" for="node">
<desc>Vertex x coordinate</desc>
<default>0.0</default>
</key>
<key id="start" for="node">
<desc>Vertex is start</desc>
<default>false</default>
</key>
<key id="value" for="node">
<desc>Vertex value</desc>
<default></default>
</key>
<key id="coord.y" for="node">
<desc>Vertex y coordinate</desc>
<default>0.0</default>
</key>
<key id="end" for="node">
<desc>Vertex is end</desc>
<default>false</default>
</key>
<key id="weight" for="edge">
<desc>Edge weight</desc>
<default>1</default>
</key>
<key id="name" for="edge">
<desc>Edge name</desc>
<default></default>
</key>
<graph edgedefault="undirected">
<node id="A">
  <data key="coord.x">204.0</data>
  <data key="start">true</data>
  <data key="value">
  </data>
  <data key="coord.y">175.0</data>
  <data key="end">false</data>
</node>
<node id="L">
```

Abbildung 3.5: Das Dateiformat *.vistra als GraphML

3.2.2 App

Die Komponente App implementiert ein Model-View-Control (MVC) unter Verwendung des Java-Observer-Pattern:

- Model: Observable mit Attributen und deren getter- und setter-Methoden
- View: Observer mit grafischen Elementen wie z.B. Menubar, Knöpfe, Regler und Text-Panelen
- Control: Implementiert Listeners mit deren Methoden und evtl. private Methoden.

Das Control für die drei GUI-Hauptkomponenten Parameter, Step-by-Step und Animation wird als Zustandsautomat (state machine) verstanden. Demzufolge wird ein State-Pattern implementiert. Die State-Handler (Parameter-, StepByStep- und Animation-State-Handler) sind auch EventListeners, um auf Benutzerinteraktionen mittels Menu- und Tool-Bar sowie Buttons und Dropdowns reagieren zu können.

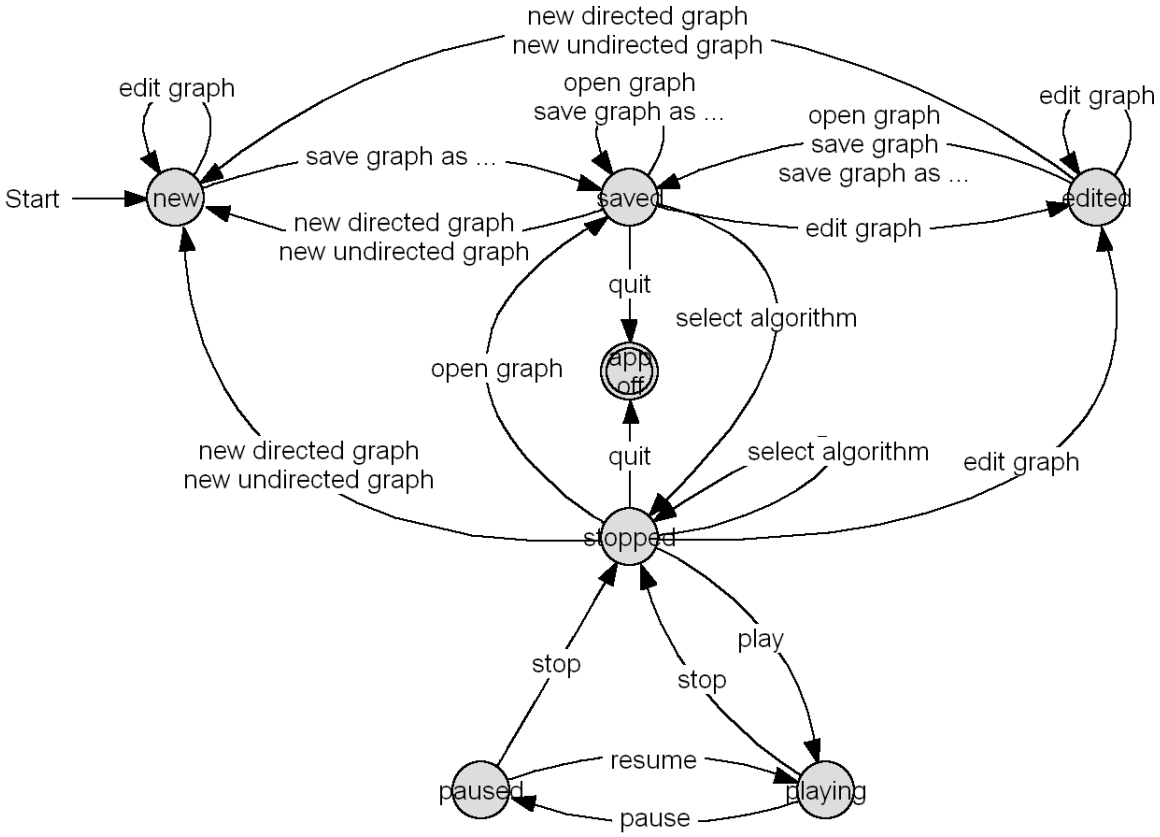

Abbildung 3.6: Control als state machine (Entwurf für Parameter und Animation)

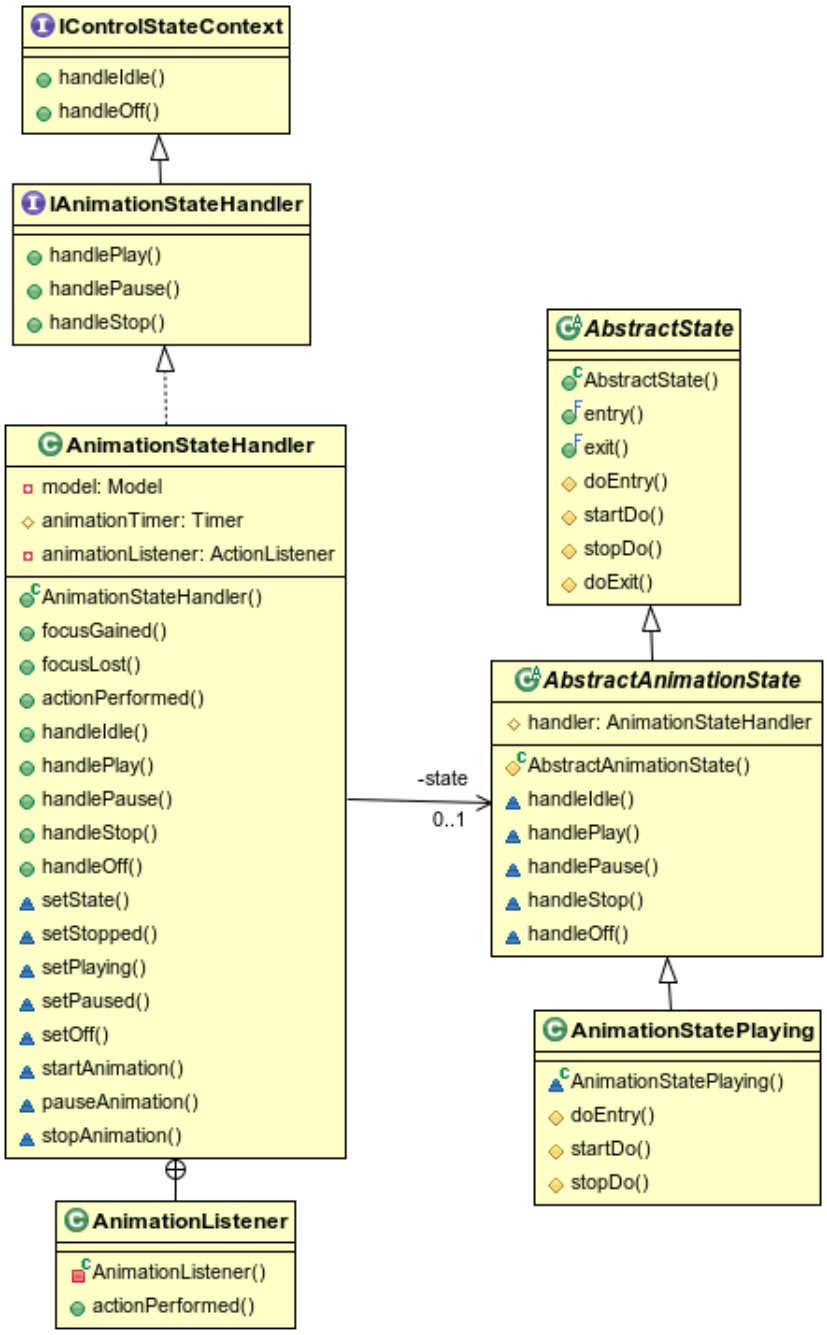

Abbildung 3.7: Control für die GUI-Hauptkomponente Animation

4 Solution

Abbildung 4.1: Graphisches User Interface

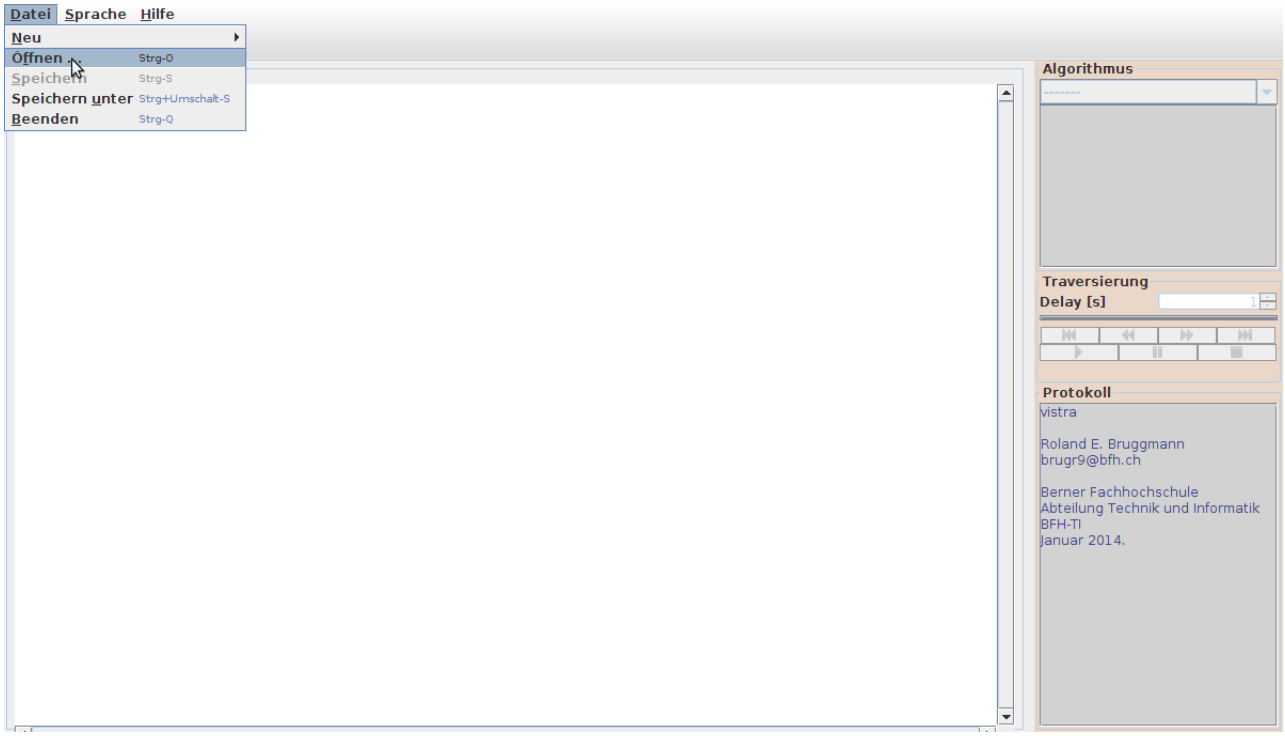

Abbildung 4.2: Menu: Datei > Öffnen

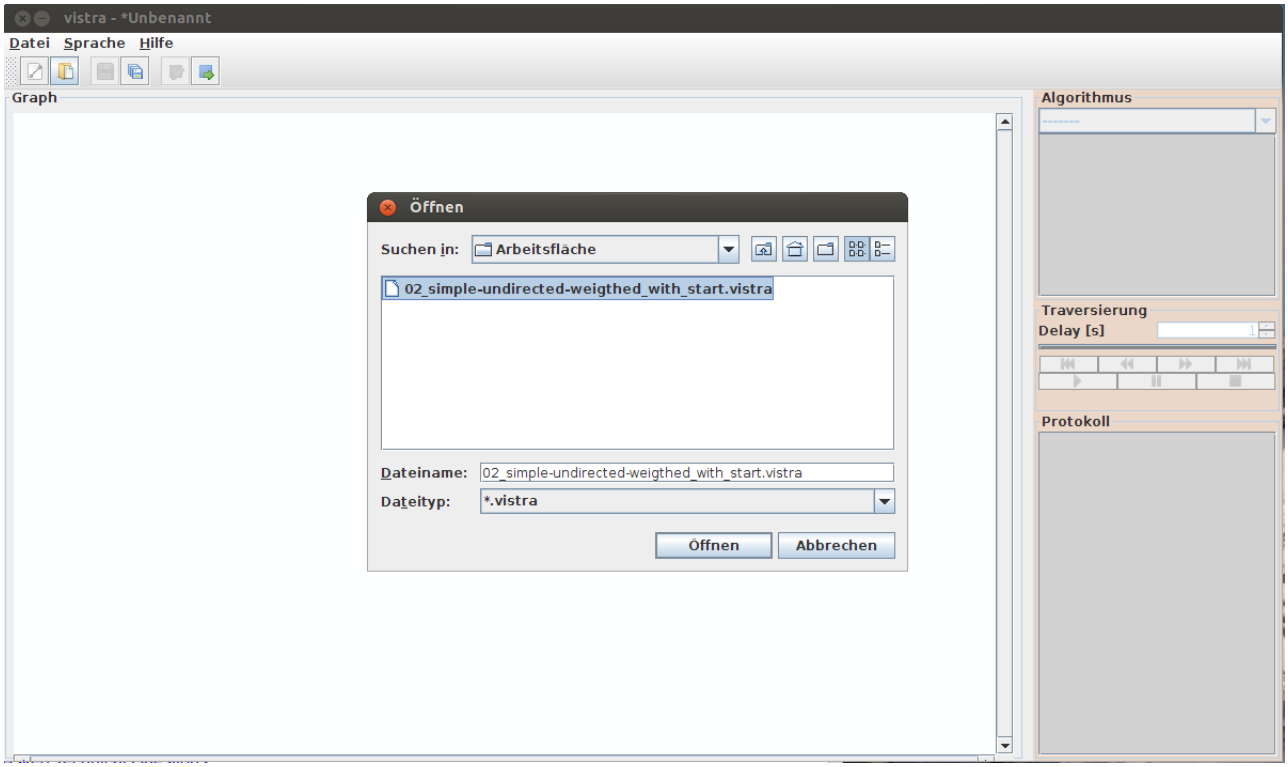

Abbildung 4.3: Dialog: Öffnen

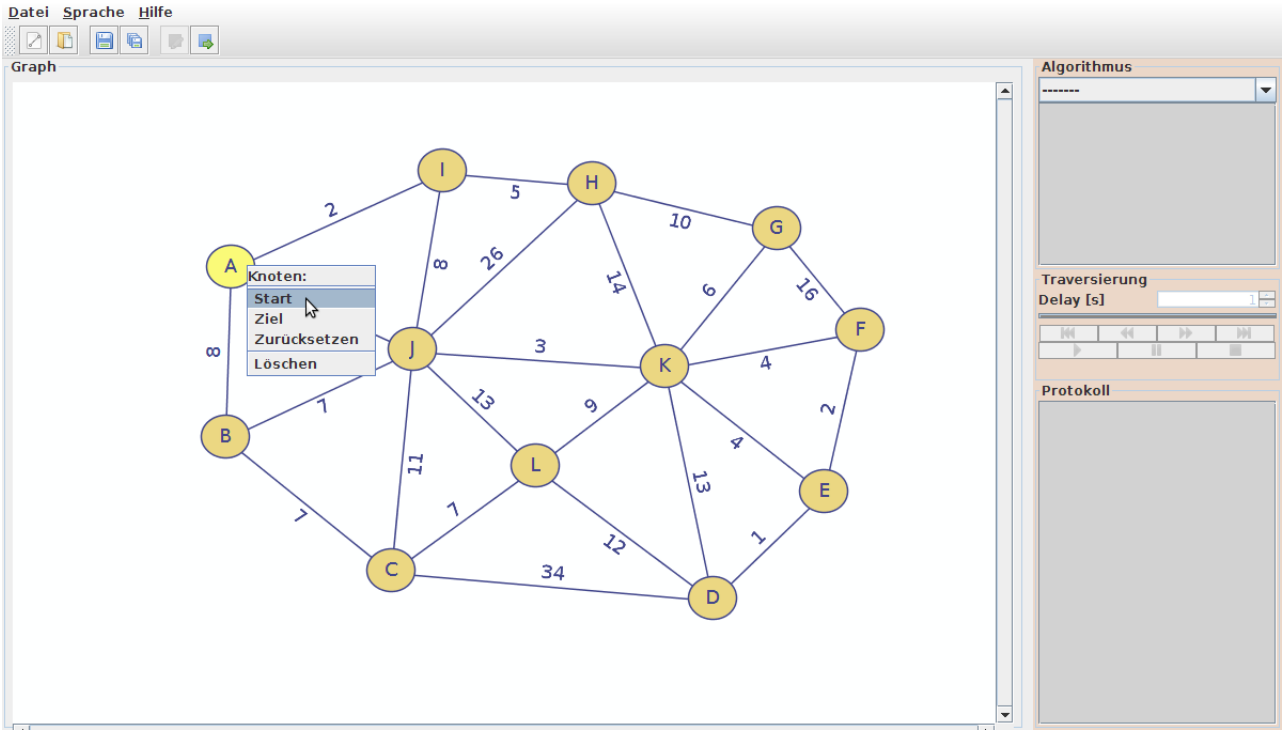

Abbildung 4.4: Popup: Knoten editieren, Start festlegen

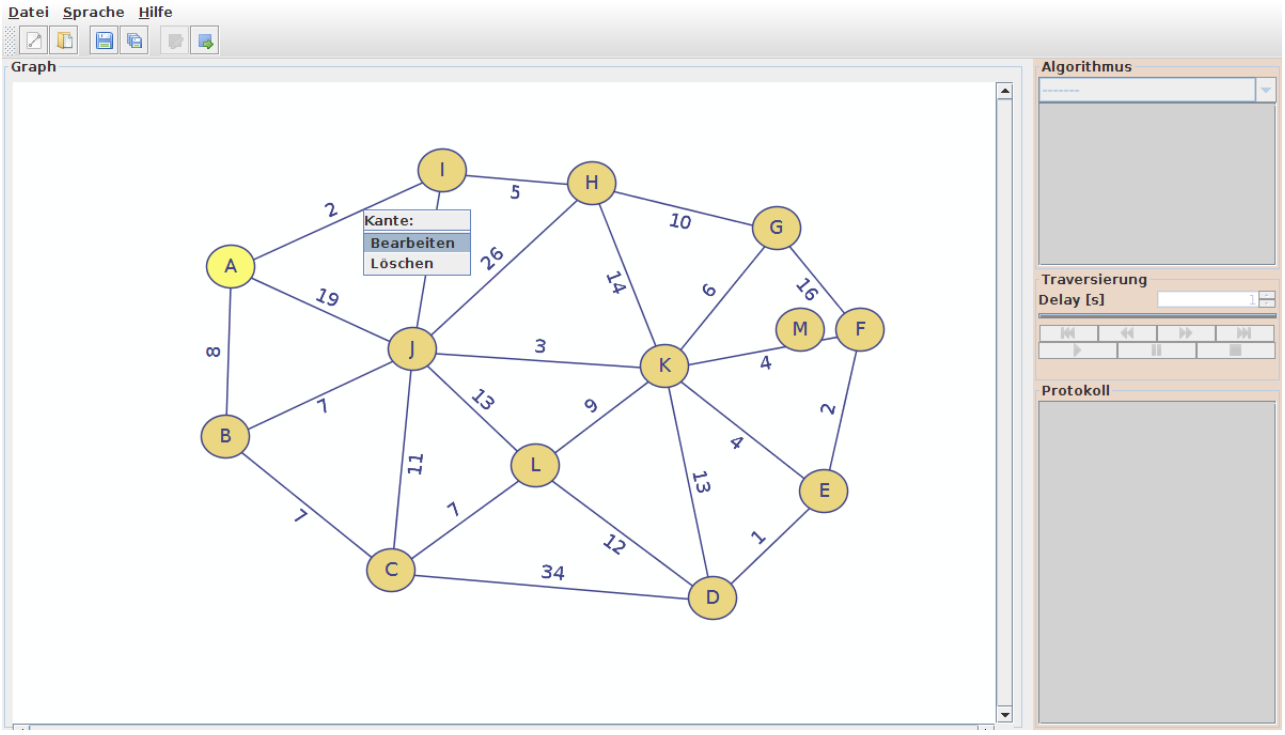

Abbildung 4.5: Popup: Kante editieren

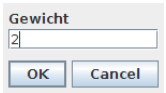

Abbildung 4.6: Dialog: Kantengewicht editieren

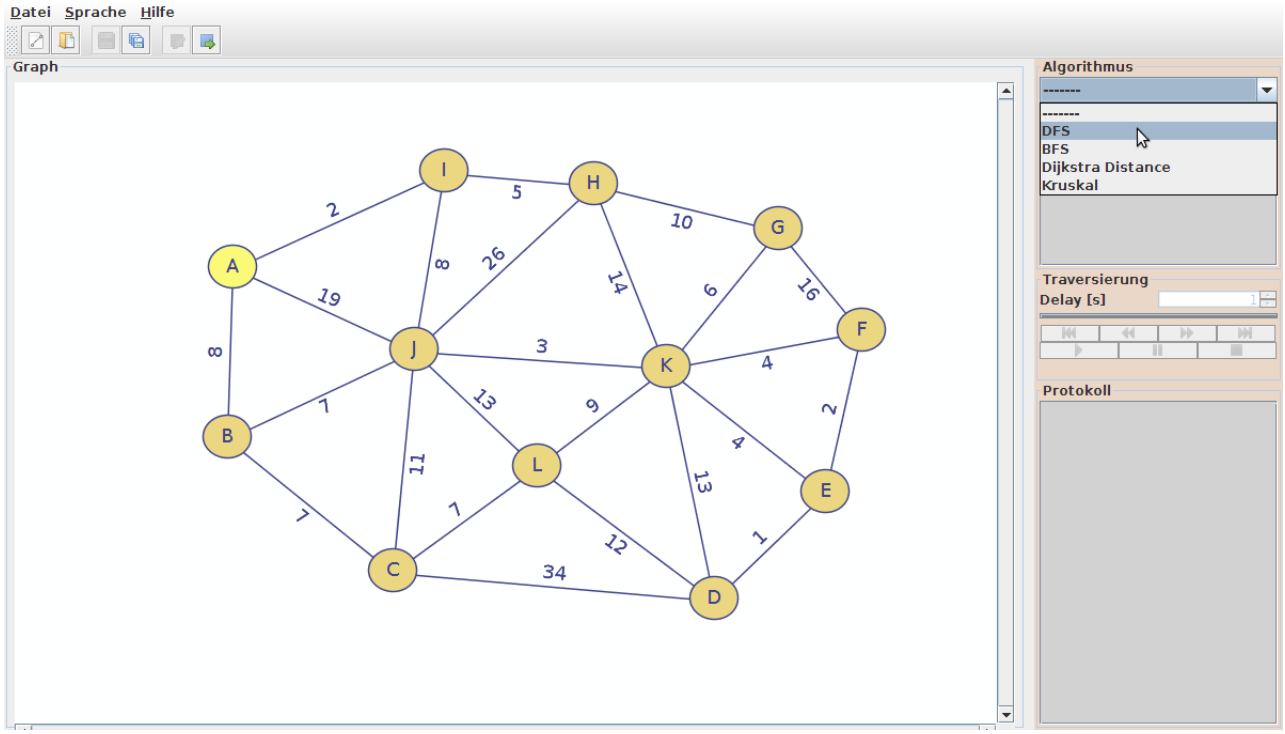

Abbildung 4.7: Dropdown: Algorithmus wählen

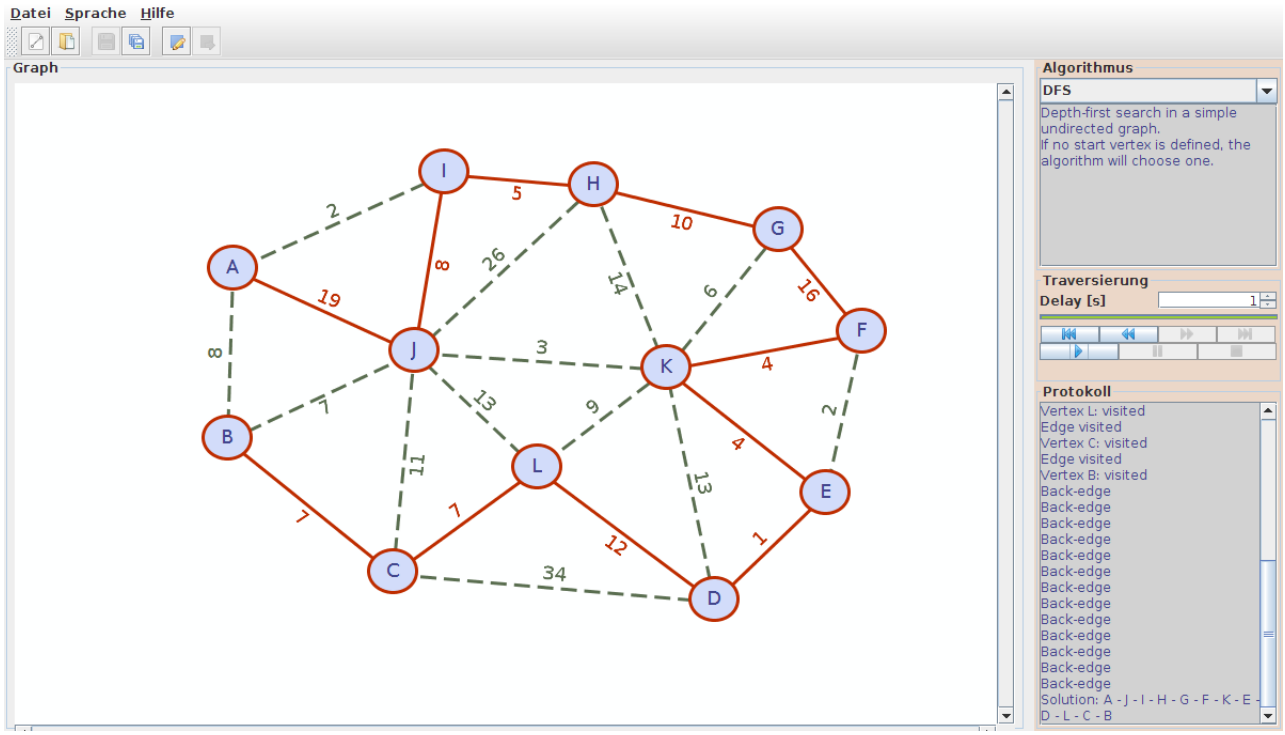

Abbildung 4.8: Resultat: Algorithmus DFS

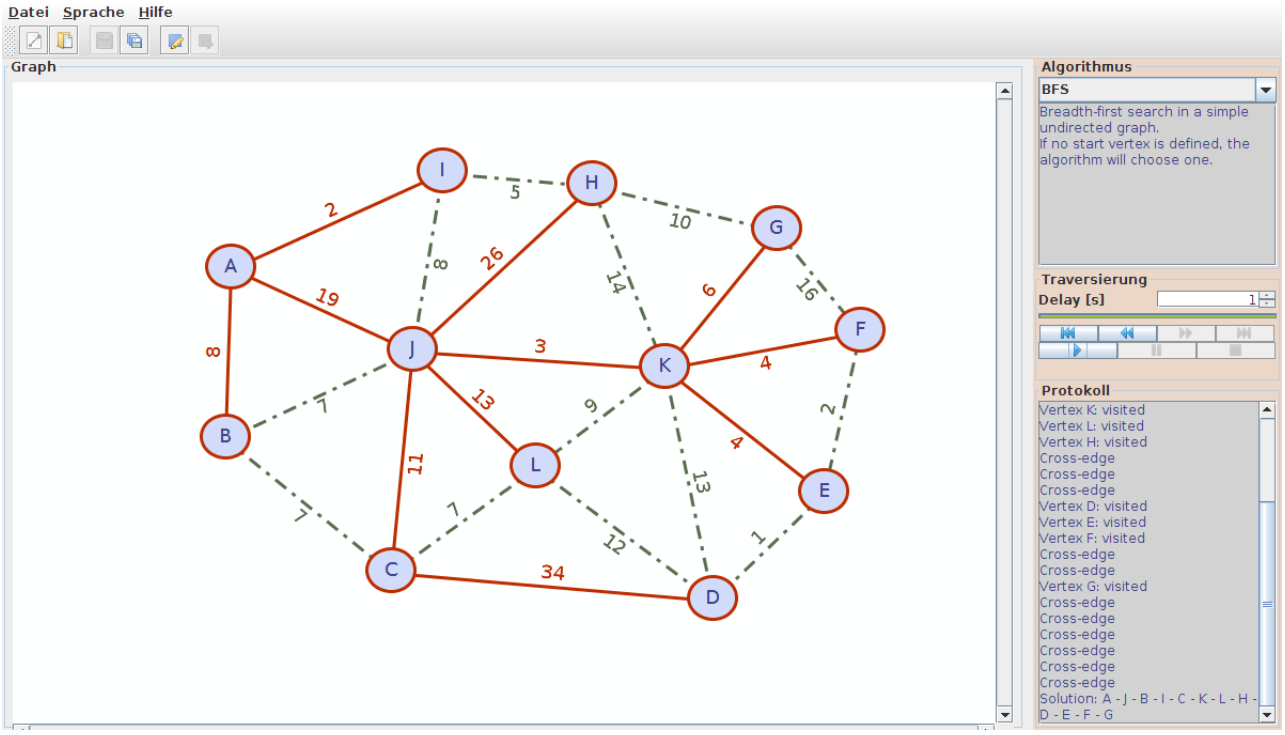

Abbildung 4.9: Resultat: Algorithmus BFS

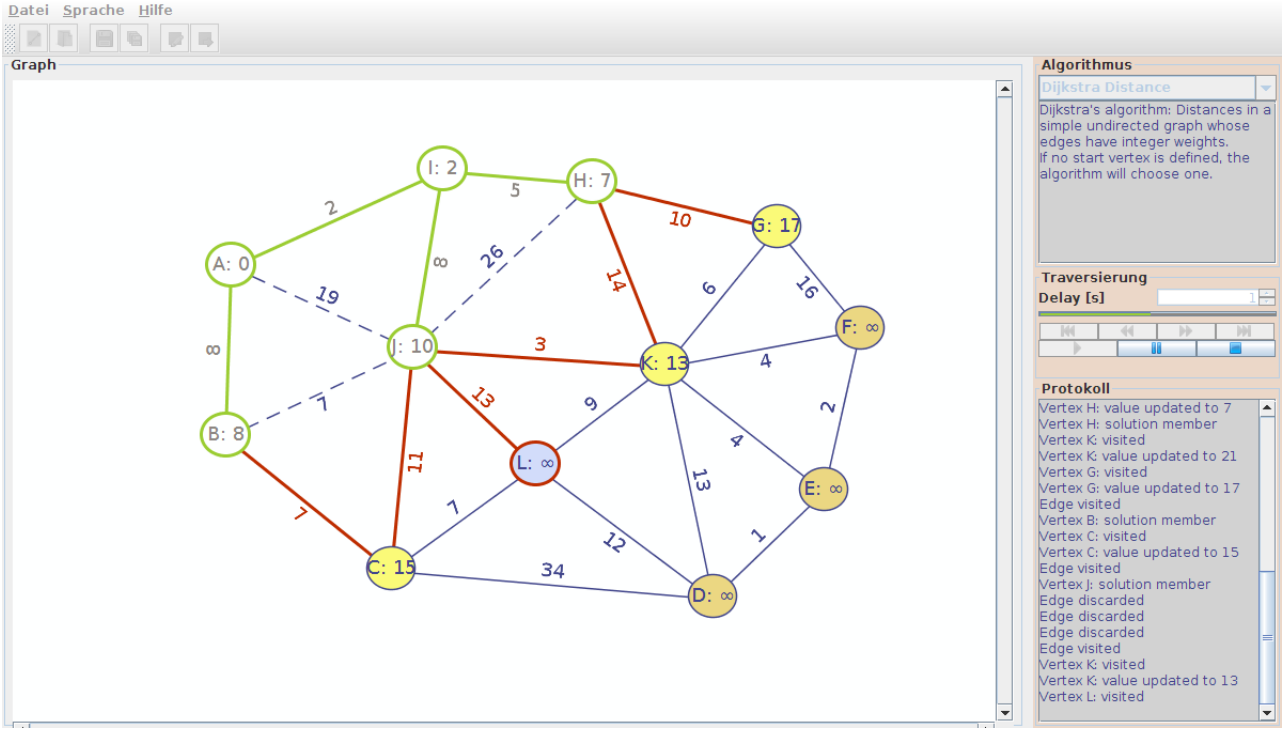

Abbildung 4.10: Animation: Algorithmus Dijkstra Distance

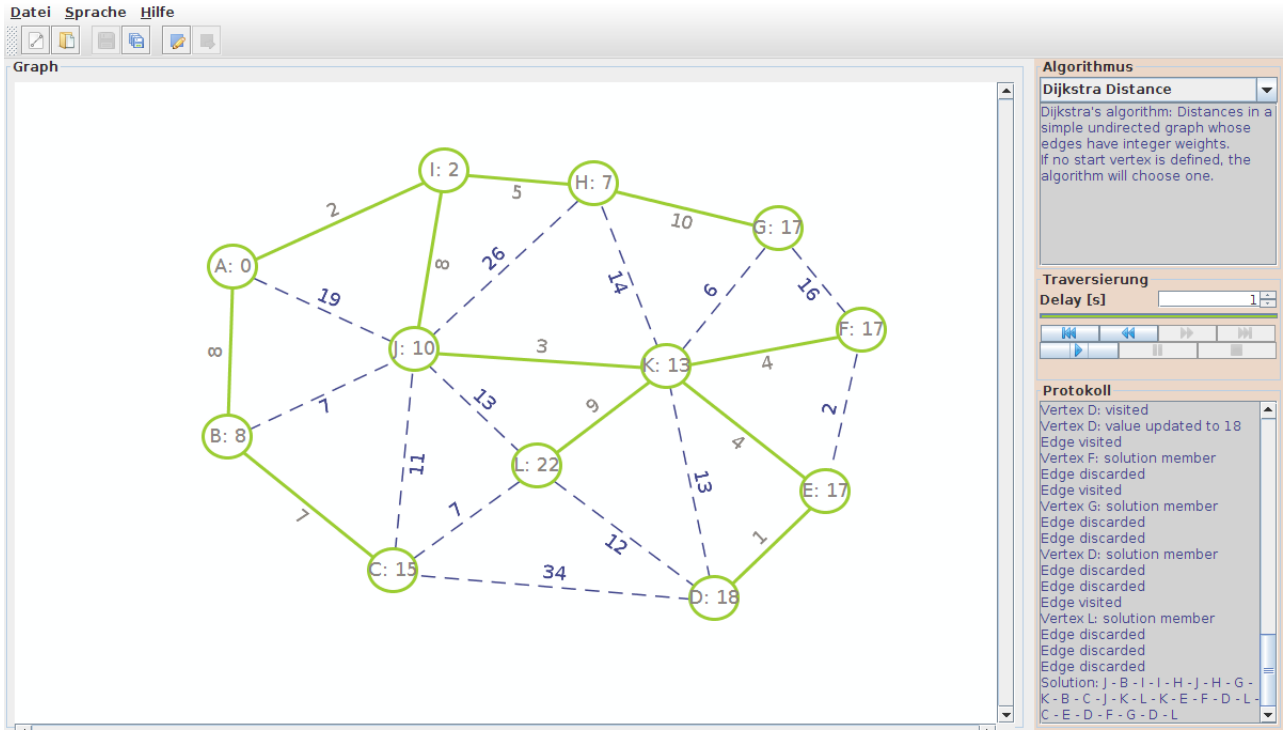

Abbildung 4.11: Resultat: Algorithmus Dijkstra Distance

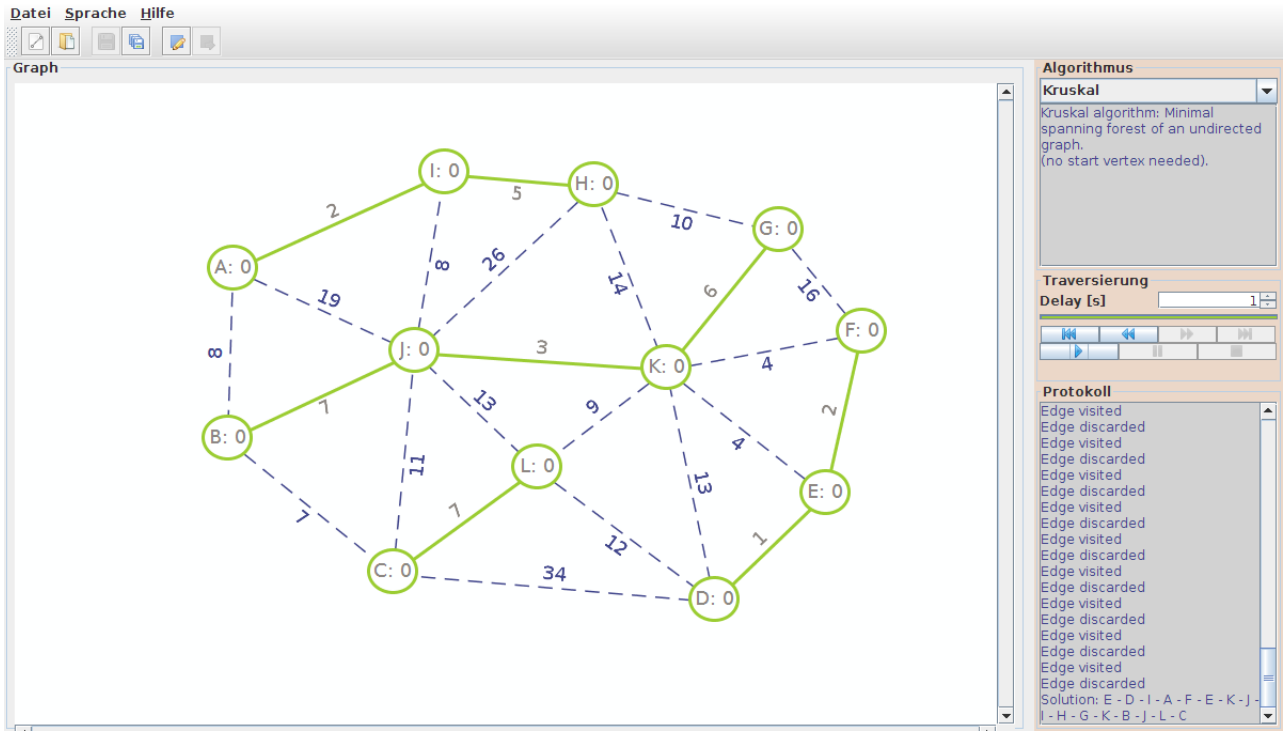

Abbildung 4.12: Resultat: Algorithmus Kruskal