**LK-Q1-V**

**2. Die Datenstruktur Graph im Anwendungskontext unter Nutzung der Klassen Graph, Vertex und Edge.**

(d) Minimale Spannbäume

120502_Spbaum1.emf

**Aufgabe**

*Gegeben sind Verbraucher, die an ein Versorgungsnetz angeschlossen werden sollen. Die Kosten für die Leitung von jedem Punkt zum anderen sind bekannt.*

*Es wird ein Verbindungsnetz mit minimalen Kosten gesucht.*

Lösung:

Man kann sich leicht überlegen, dass ein minimales Versorgungsnetz ein Teilgraph des Graphen ist, der aus einer Menge von Kanten besteht, die alle Knoten miteinander verbinden. Der Teilgraph darf keine Rundwege (Zyklen) enthalten. Denn gäbe es einen geschlossenen Zyklus, könnte man ein billigeres Versorgungsnetz durch Entfernen einer Kante aus diesem Zyklus konstruieren.

**Definitionen**

Ein ungerichteter Graph heißt **zusammenhängend**, wenn es zwischen zwei Knoten mindestens einen Weg gibt.   
Ein ungerichteter Graph heißt **Baum**, falls er zusammenhängend und zyklenfrei ist.

Ein Teilgraph eines ungerichteten Graphen G heißt Spannbaum von G, wenn H ein Baum auf den Knoten von G ist.

Ein Spannbaum eines gewichteten ungerichteten Graphen G heißt minimaler Spannbaum von G, wenn S minimales Gewicht unter allen Spannbäumen von G besitzt.

**Algorithmen zur Bestimmung eines minimalen Spannbaums**

**1. Der Kruskal-Algorithmus**

Generiere nach Gewicht aufsteigend eine sortierte Liste L der Kanten des Graphen

Solange L nicht leer ist

wähle erste Kante e aus L

entferne e aus L

markiere e

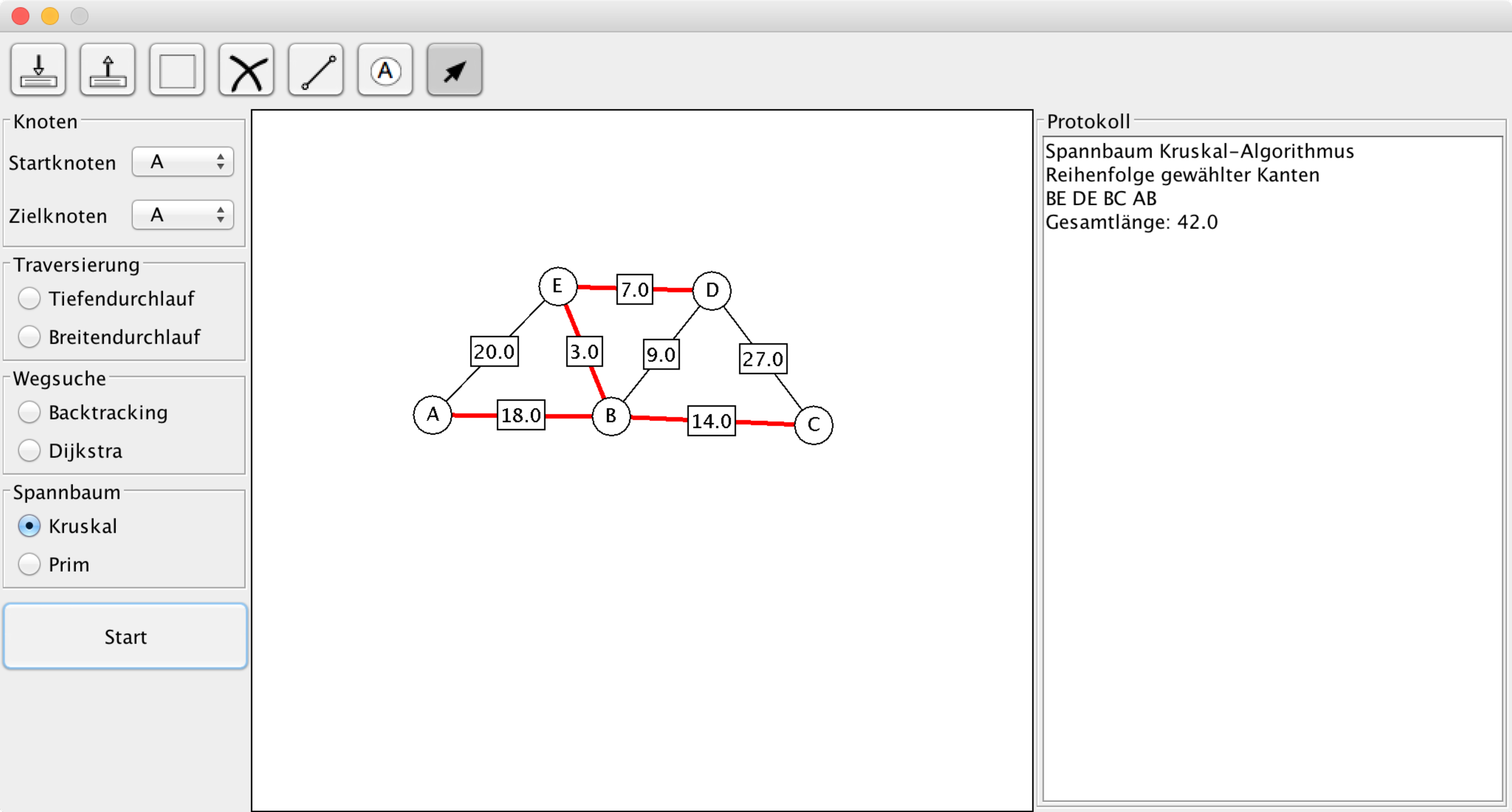
wenn der Graph aus den markierten Kanten und deren Endknoten einen Zyklus enthält

lösche Markierung von e

Der Graph aus den markierten Kanten und deren Endknoten ist ein minimaler Spannbaum

**Der Kruskal-Algorithmus schrittweise**

|  |  |
| --- | --- |
| kruskal01 | kruskal02 |
| kruskal03 | kruskal04  Zyklus |
| kruskal05 | kruskal06 |
| kruskal07 | kruskal08 |
| kruskal09 |  |

Mit dem Java-Programm GraphTool lassen sich interaktiv Graphen erzeugen und verschiedene Graphenalgorithmen anwenden und veranschaulichen. 

**Quelltext Kruskal-Algorithmus**

public String spannbaumKruskal(){

bspGraph.setAllVertexMarks(false);

bspGraph.setAllEdgeMarks(false);

List<Edge> kantenListe =this.gibAufsteigendSortiert(bspGraph.getEdges());

String kantenString = "";

kantenListe.toFirst();

while (!kantenListe.isEmpty()){

Edge aktuelleKante = kantenListe.getContent();

Vertex[] knoten = (aktuelleKante.getVertices());

aktuelleKante.setMark(true);

resetStatus(bspGraph.getVertices());

if (hatZyklus((Vertex1)knoten[0], null)){

aktuelleKante.setMark(false);

} else {

kantenString = kantenString + knoten[0].getID()+" "+

knoten[1].getID()+":"+aktuelleKante.getWeight()+"\n";

}

kantenListe.remove();

}

return kantenString;

}

Die Implementation entspricht dem umgangssprachlich formulierten Algorithmus. Die Methode liefert einen String mit den Kanten und ihren Längen zurück, die zum minimalen Spannbaum gehören. bspGraph ist eine Referenz auf den Graphen, von dem ein minimaler Spannbaum erzeugt werden soll. Die Variable ist als Attribut der Klasse Spannbaum, die spannbaumKruskal als Methode enthält, deklariert. Die Methode verwendet drei Hilfsmethoden:

1. gibAufsteigendSortiert erzeugt die sortierte Kantenliste
2. hatZyklus prüft, ob die markierten Kanten einen Kreis enthalten
3. resetStatus setzt den Status aller Knoten auf 0.

Vertex1 ist eine Unterklasse der Klasse Vertex, die das Attribut status vom Typ int enthält, das für die Prüfung, ob ein Zyklus vorliegt, benötigt wird.

**2. Der Prim-Algorithmus**

Ein zweiter Algorithmus zur Bestimmung minimaler Spannbäume ist der Algorithmus von Prim.

**Umgangssprachliche Formulierung**

Lösche alle Knotenmarkierungen

Wähle einen Knoten aus und markiere ihn

Führe den folgenden Schritt so oft aus, bis alle Knoten markiert sind:

Suche die Kante mit dem geringsten Gewicht zwischen einem Knoten aus der Menge der markierten Knoten und einem Knoten aus der Menge der nicht markierten Knoten. Markiere den nicht markierten Knoten.

Der Prim-Algorithmus schrittweise

|  |  |
| --- | --- |
| prim01 | prim02 |
| prim03 | prim04 |
| prim05 |  |

**Quelltext Prim-Algorithmus**

public String spannbaumPrim(Vertex startKnoten){

bspGraph.setAllVertexMarks(false);

startKnoten.setMark(true);

String kantenString = "";

List<Edge> kanten = bspGraph.getEdges(startKnoten);

while (!bspGraph.allVerticesMarked()){

Edge minKante = null;

kanten.toFirst();

Vertex[] endKnoten;

while (kanten.hasAccess()){

Edge kante = kanten.getContent();

endKnoten = kante.getVertices();

if (!endKnoten[0].isMarked() || !endKnoten[1].isMarked()){

if (minKante == null || kante.getWeight() < minKante.getWeight()){

minKante = kante;

}

}

kanten.next();

}

minKante.setMark(true);

endKnoten = minKante.getVertices();

kantenString = kantenString+endKnoten[0].getID()+" "

+endKnoten[1].getID() +" \n";

Vertex neuerKnoten = null;

if (endKnoten[0].isMarked()){

neuerKnoten = endKnoten[1];

} else {

neuerKnoten = endKnoten[0];

}

//Neu verbundenen Knoten markieren.

neuerKnoten.setMark(true);

//Mit dem neuen Knoten inzidente Kanten der Kantenliste hinzufuegen.

List<Edge> neueKanten = bspGraph.getEdges(neuerKnoten);

neueKanten.toFirst();

while (neueKanten.hasAccess()){

kanten.append(neueKanten.getContent());

neueKanten.next();

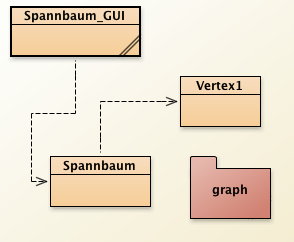
}

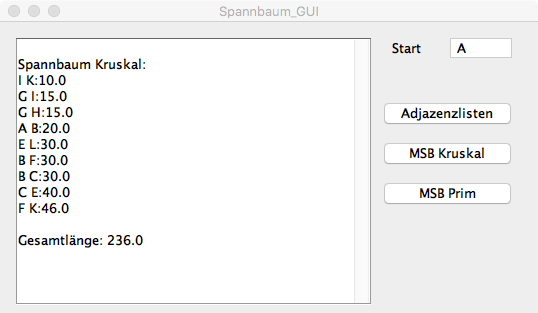
}

return kantenString;

}

Die Implementation entspricht dem umgangssprachlich formulierten Algorithmus. Die Methode liefert einen String mit den Kanten und ihren Längen zurück, die zum minimalen Spannbaum gehören. Die markierten Kanten bilden mit ihren Endknoten den minimalen Spannbaum.

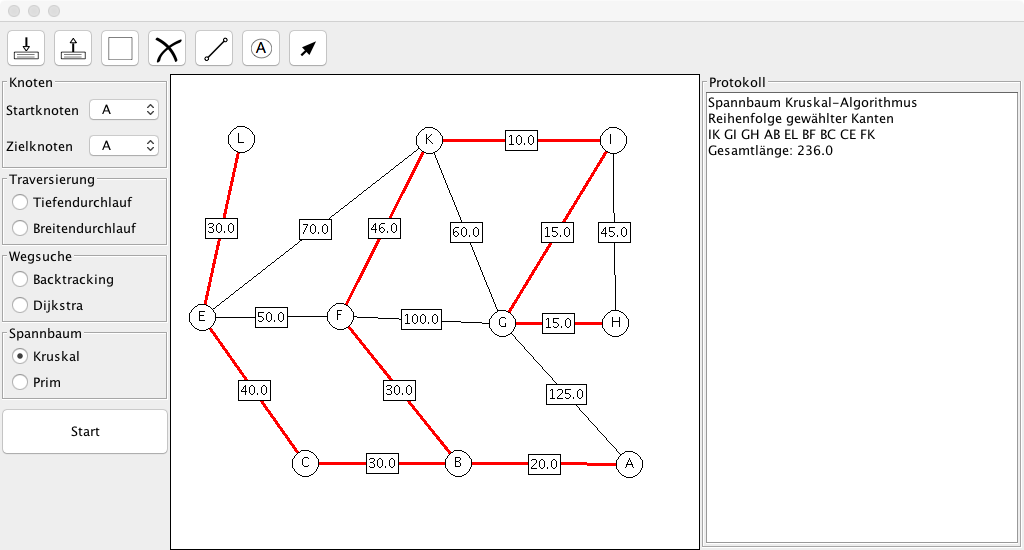
Die vollständige Implementation ist im BlueJ –Projekt Spannbaum enthalten. Um die Quelltexte übersichtlich zu halten, wurden nicht alle möglichen Fehler abgefangen. Ohne Titel



**Vergleich Kruskal-Prim**

Ein Graph kann mehrere minimale Spannbäume haben. Diese haben aber immer dieselbe minimale Länge. Der Algorithmus von Prim vermeidet das Sortieren der Kanten, das beim Algorithmus von Kruskal erforderlich ist. Allerdings ist die Auswahl der Kanten beim Algorithmus von Prim komplizierter als beim Algorithmus von Kruskal.

**Aufgabe**

1. *Ermitteln Sie mithilfe des Kruskal-Algorithmus’ einen minimalen Spannbaum für den nebenstehenden Graphen*
2. *Ermitteln Sie mithilfe des Prim-Algorithmus’ einen minimalen Spannbaum mit dem Startknoten A für den nebenstehenden Graphen.*