Spannbäume

Definition „Spannbaum“

Es sei G ein zusammenhängender Graph und S ein zusammenhängender Teilgraph von G. S ist ein Spannbaum von G, falls *S alle Knoten von G enthält* und *S zyklenfrei* ist.

alle Knoten von G enthält

zyklenfrei

Definition „Minimaler Spannbaum“

S ist ein minimaler Spannbaum, falls S ein Spannbaum von G ist, und die Summe der *Kantengewichte* in *S kleiner oder gleich der aller anderen möglichen Spannbäume S'* von G ist[1].

Kantengewichte

kleiner oder gleich der aller anderen möglichen Spannbäume

Algorithmus von Kruskal

Durch den Algorithmus von Kruskal wird ein *minimaler Spannbaum* eines ungerichteten, zusammenhängenden und kantengewichteten Graphen bestimmt.[2]

minimaler Spannbaum

Der Algorithmus von Kruskal nutzt die Kreiseigenschaft minimaler Spannbäume. Dazu werden die Kanten in der *ersten Phase aufsteigend nach ihrem Gewicht sortiert*. In der zweiten Phase wird *über die sortierten Kanten iteriert*. Wenn eine *Kante* zwei Knoten verbindet, die *noch nicht* durch einen Pfad vorheriger Kanten *verbunden* sind, wird diese Kante zum minimalen Spannbaum *hinzugenommen*.[3]

ersten Phase

aufsteigend nach ihrem Gewicht sortiert

über die sortierten Kanten iteriert

Kante

noch nicht

verbunden

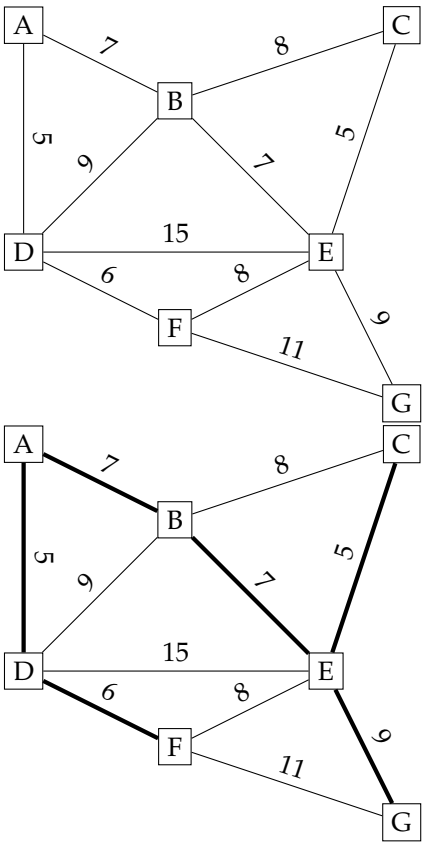
hinzugenommen

Algorithmus 1: Minimaler Spannbaum nach Kruskal[a]

Data: $G = (V, E, w)$: ein zusammenhängender, ungerichteter, kantengewichteter Graph kruskal(G)
$E' \leftarrow \emptyset$;
$L \leftarrow E$;
Sortiere die Kanten in L aufsteigend nach ihrem Kantengewicht.;
while $L \neq \emptyset$ **do**
  wähle eine Kante $e \in L$ mit kleinstem Kantengewicht;
  entferne die Kante e aus L;
  if *der Graph* $(V, E' \cup \{e\})$ *keinen Kreis enthält* **then**
    $E' \leftarrow E' \cup \{e\}$;
  end
end
Result: $M = (V, E')$ ist ein minimaler Spannbaum von G.

[a] *Wikipedia-Artikel „Algorithmus von Kruskal“.*

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 29.
[2] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 30 (PDF 24).
[3] *Wikipedia-Artikel „Algorithmus von Kruskal“.*



Kante		Gewicht
AD, CE	2×5	10
DF		6
AB, BE	2×7	14
EG		9
		39

```
3   import org.bschlangaul.graph.GraphAdjazenzMatrix;
4
5   /**
6    * Implementation des Algorithmus von Kruskal.
7    *
8    * Nach dem Tutorial auf <a href=
9    * "https://www.geeksforgeeks.org/kruskals-algorithm-simple-implementation-for-
     ↪  adjacency-matrix/">geeksforgeeks.org</a>.
10   */
11  public class MinimalerSpannbaumKruskal extends GraphAdjazenzMatrix {
12
13    class Ergebnis {
14      double kosten;
15      ErgebnisKante[] kanten;
16      int aktuelleKante;
17
18      public Ergebnis(int knotenanzahl) {
19        kanten = new ErgebnisKante[knotenanzahl - 1];
20      }
21
22      public void fügeKanteHinzu(String von, String nach, double gewicht) {
23        kanten[aktuelleKante] = new ErgebnisKante(von, nach, gewicht);
24      }
25    }
26
27    class ErgebnisKante {
28      String von;
29      String nach;
30      double gewicht;
```

```
31
32      public ErgebnisKante(String von, String nach, double gewicht) {
33        this.von = von;
34        this.nach = nach;
35        this.gewicht = gewicht;
36      }
37    }
38
39    public MinimalerSpannbaumKruskal(String graphenFormat) {
40      super(graphenFormat);
41    }
42
43    Ergebnis ergebnis = new Ergebnis(gibKnotenAnzahl());
44    int[] eltern = new int[gibKnotenAnzahl()];
45
46    private int find(int i) {
47      while (eltern[i] != i) {
48        i = eltern[i];
49      }
50      return i;
51    }
52
53    private void union1(int i, int j) {
54      int a = find(i);
55      int b = find(j);
56      eltern[a] = b;
57    }
58
59    public double führeAus() {
60      double kosten = 0;
61
62      for (int i = 0; i < gibKnotenAnzahl(); i++) {
63        eltern[i] = i;
64      }
65
66      int kantenAnzahl = 0;
67      while (kantenAnzahl < gibKnotenAnzahl() - 1) {
68        double gewicht = Double.MAX_VALUE;
69        int knotenA = -1;
70        int knotenB = -1;
71        for (int i = 0; i < gibKnotenAnzahl(); i++) {
72          for (int j = 0; j < gibKnotenAnzahl(); j++) {
73            if (find(i) != find(j) && matrix[i][j] < gewicht && matrix[i][j] !=
              ↪  NICHT_ERREICHBAR) {
74              gewicht = matrix[i][j];
75              knotenA = i;
76              knotenB = j;
77            }
78          }
79        }
80
81        union1(knotenA, knotenB);
82        kantenAnzahl++;
83        ergebnis.fügeKanteHinzu(gibKnotenName(knotenA), gibKnotenName(knotenB),
          ↪  gewicht);
84        kosten += gewicht;
85      }
86      ergebnis.kosten = kosten;
87      return kosten;
88    }
89
90    public static void main(String[] args) {
```

```java
    MinimalerSpannbaumKruskal kruskal = new MinimalerSpannbaumKruskal(
        "v0--v1:2;v1--v2:3;v0--v3:6;v1--v3:8;v1--v4:5;v2--v4:7;v3--v4:9;");
    kruskal.gibMatrixAus();
    kruskal.führeAus();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/graph/algorithmen/MinimalerSpannbaumKruskal.java`

Algorithmus von Prim[4]

Entdecker

Teilgraph

schrittweise vergrößert

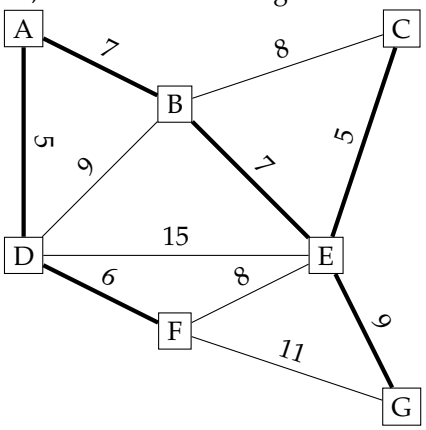
konkreten Startpunkt

günstigste

ausgehende Kante ausgewählt

Der Algorithmus von Prim dient ebenfalls der Bestimmung eines minimalen Spannbaums, erreicht jedoch sein Ziel durch eine unterschiedliche Vorgehensweise. Er wird auch nach dem *Entdecker* Jarník genannt.[5]

Der *Teilgraph T* wird *schrittweise vergrößert*, bis dieser ein Spannbaum ist. Der Algorithmus benötigt einen *konkreten Startpunkt*. Es wird die *günstigste* von einem Teilgraphen *T ausgehende Kante ausgewählt* und zu diesem Spannbaum hinzugefügt. Der Algorithmus fügt jede Kante und deren Endknoten zur Lösung hinzu, die mit allen zuvor gewählten Kanten keinen Kreis bildet.[6]



Kante	Gewicht
AD	5
DF	6
AB	7
BE	7
EC	5
EG	9
	39

```java
import org.bschlangaul.graph.GraphAdjazenzMatrix;
import org.bschlangaul.graph.report.GraphReporter;

class PrimReporter extends GraphReporter {

  GraphAdjazenzMatrix matrix;

  public PrimReporter(GraphAdjazenzMatrix matrix) {
    this.matrix = matrix;
  }

  public String knoten(int knotenNr) {
    return konsolenAusgabe.grün(matrix.gibKnotenName(knotenNr));
  }

  public String zahl(Double zahl) {
    return matrix.formatiereZahl(zahl);
  }
```

[4] *Wikipedia-Artikel „Algorithmus von Prim“.*
[5] *Wikipedia-Artikel „Algorithmus von Prim“.*
[6] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 32, (PDF 26).

```
21
22    public String kante(int eltern, int kind, double gewicht) {
23      return "v(" + knoten(eltern) + ", " + knoten(kind) + ", " +
        ↪  konsolenAusgabe.gelb(zahl(gewicht)) + ")";
24    }
25
26    void knotenBesuch(int knotenNr) {
27      System.out.println("Besuche Knoten „" + knoten(knotenNr) + """);
28    }
29
30    void kantenHinzufügen(int eltern, int kind, double gewicht) {
31      System.out.println("Füge Kante " + kante(eltern, kind, gewicht) + " hinzu");
32    }
33
34    void kantenAktualisierung(int eltern, int kind, double gewicht) {
35      System.out.println("Aktualisiere Kante " + kante(eltern, kind, gewicht));
36    }
37  }
38
39  /**
40   * Implementation des Algorithmus von Prim / Jarník. Momentan beginnt der
41   * Algorithmus immer vom ersten Knoten in der Adjazenzmatrix, d.h. mit dem
42   * Knoten mit der Knotennummer 0.
43   *
44   * Nach dem Tutorial auf <a href=
45   * "https://algorithms.tutorialhorizon.com/prims-minimum-spanning-tree-mst-using-
     ↪  adjacency-matrix/">tutorialhorizon.com</a>.
46   */
47  public class MinimalerSpannbaumPrim extends GraphAdjazenzMatrix {
48
49    PrimReporter berichte;
50
51    /**
52     * Ein Feld, in dem gespeichert wird, ob der Elternknoten bereits besucht
     ↪  wurde.
53     * Wurde der Elternknoten mit der Nummer 3 besucht wird besucht[3] auf true
54     * gesetzt.
55     */
56    boolean[] besucht;
57
58    /**
59     * Ein Feld, in dem alle Ergebnisse gespeichert werden. Ein Ergebnis wird unter
60     * der Kindknoten-Nummer gespeichert. Manche Ergebniseinträge werden
61     * überschrieben.
62     */
63    Ergebnis[] ergebnisse = new Ergebnis[gibKnotenAnzahl()];
64
65    /**
66     * Das Gewicht wird unter der Kindknoten-Nummer gespeichert.
67     */
68    double[] gewichte;
69
70    public MinimalerSpannbaumPrim(String graphenFormat) {
71      super(graphenFormat);
72      besucht = new boolean[gibKnotenAnzahl()];
73      ergebnisse = new Ergebnis[gibKnotenAnzahl()];
74      gewichte = new double[gibKnotenAnzahl()];
75      berichte = new PrimReporter(this);
76    }
77
78    /**
79     * Gib den Knoten mit dem minimalen Gewicht, der sich noch nicht im minimalen
```

```
80      * Spannbaum befindet.
81      *
82      * @return Die ID des Knoten mit dem minimalen Gewicht. Es können Zahlen
83      *         beginnend mit 0 vorkommen.
84      */
85    private int gibMinimumKnoten() {
86      double minGewicht = Double.MAX_VALUE;
87      int knoten = -1;
88      for (int i = 0; i < gibKnotenAnzahl(); i++) {
89        if (besucht[i] == false && minGewicht > gewichte[i]) {
90          minGewicht = gewichte[i];
91          knoten = i;
92        }
93      }
94      return knoten;
95    }
96
97    /**
98     * Die Instanzen der Klasse Ergebnis wird in das Feld {@code ergebnisse}
99     * gespeichert. Die Klasse speichert das Kantengewicht von einem bestimmten
100    * Knoten zu seinem Elternknoten.
101    */
102   class Ergebnis {
103     /**
104      * Die Index-Nummer des Elternknoten über den der aktuelle Knoten erreicht
      ↪ wird.
105      * -1 bedeuted, dass das Ergebnis keinen Elternknoten hat
106      */
107     int eltern = -1;
108
109     /**
110      * Das Gewicht der Kante vom Elternknoten zum aktuellen Knoten. Das Feld wird
111      * benötigt um den minimale Knoten zu finden.
112      */
113     double gewicht = NICHT_ERREICHBAR;
114   }
115
116   /**
117    * Führe den Algorithmus von Prim / Jarník aus.
118    *
119    * @return Die Summer aller Kantengewichte.
120    */
121   public double führeAus() {
122     for (int i = 0; i < gibKnotenAnzahl(); i++) {
123       // Initialisiere alle Gewichte mit Unendlich
124       gewichte[i] = Double.MAX_VALUE;
125       // Erzeuge leere Ergebnis-Instanzen.
126       ergebnisse[i] = new Ergebnis();
127     }
128
129     gewichte[0] = 0;
130     ergebnisse[0] = new Ergebnis();
131     ergebnisse[0].eltern = -1;
132
133     for (int i = 0; i < gibKnotenAnzahl(); i++) {
134       int eltern = gibMinimumKnoten();
135       berichte.knotenBesuch(eltern);
136       besucht[eltern] = true;
137       for (int kind = 0; kind < gibKnotenAnzahl(); kind++) {
138         if (matrix[eltern][kind] > 0) {
139           if (besucht[kind] == false && matrix[eltern][kind] < gewichte[kind]) {
140             gewichte[kind] = matrix[eltern][kind];
```

```
141
142             if (ergebnisse[kind].eltern != -1) {
143               berichte.kantenAktualisierung(eltern, kind, gewichte[kind]);
144             } else {
145               berichte.kantenHinzufügen(eltern, kind, gewichte[kind]);
146             }
147             ergebnisse[kind].eltern = eltern;
148             ergebnisse[kind].gewicht = gewichte[kind];
149           }
150         }
151       }
152     }
153
154     return gibErgebnisAus();
155   }
156
157   /**
158    * Gib die Ergebnisse aus.
159    *
160    * @return Die Summer aller Kantengewichte.
161    */
162   private int gibErgebnisAus() {
163     int summeGewichte = 0;
164     System.out.println("Minimaler Spannbaum: ");
165     for (int i = 1; i < gibKnotenAnzahl(); i++) {
166       System.out.println("Kante (Eltern->Kind): " +
        ↪  gibKnotenName(ergebnisse[i].eltern) + "--" + gibKnotenName(i)
167           + " Gewicht: " + ergebnisse[i].gewicht);
168       summeGewichte += ergebnisse[i].gewicht;
169     }
170     System.out.println("Summer aller Kantengewichte: " + summeGewichte);
171     return summeGewichte;
172   }
173
174   public static void main(String[] args) {
175     MinimalerSpannbaumPrim prim = new MinimalerSpannbaumPrim(
176         "v0--v1:2;v1--v2:3;v0--v3:6;v1--v3:8;v1--v4:5;v2--v4:7;v3--v4:9;");
177     prim.gibMatrixAus();
178     prim.führeAus();
179   }
180
181 }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/graph/algorithmen/MinimalerSpannbaumPrim.java`

Vergleich Kruskal und Prim

	Kruskal	**Prim**
Arbeitsweise	sortiert Kanten nach Gewichten	Startknoten → Nachbarknoten
Kanten-Sichtweise	globale Sicht	lokale Sicht
Zyklen-Vermeidung	aktiv	passiv
Gierigkeit	greedy	greedy
Teilgraph	mehrere Teilgraphen möglich	nur ein Teilgraph möglich
Datenstrukturen	Union-Find	Min-Heap
Laufzeit	$\mathcal{O}(\lvert E\rvert \cdot \log(\lvert E\rvert))$	$\mathcal{O}(\lvert E\rvert + \lvert V\rvert \cdot \log(\lvert V\rvert))$

Literatur

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6. Graphen*. https://www.studon.fau.de/file2635324_download.html.

[2] *Wikipedia-Artikel „Algorithmus von Kruskal“*. https://de.wikipedia.org/wiki/Algorithmus_von_Kruskal.

[3] *Wikipedia-Artikel „Algorithmus von Prim“*. https://de.wikipedia.org/wiki/Algorithmus_von_Prim.