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]

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]

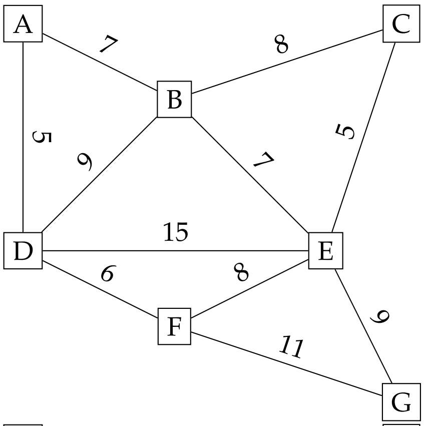
minimaler Spannbaum
ersten Phase
aufsteigend nach ihrem Gewicht sortiert
über die sortierten Kanten iteriert
Kante
noch nicht
verbunden
hinzugenommen

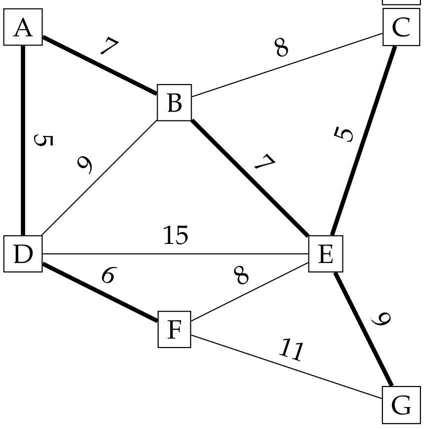
Algorithmus 1: Minimaler Spannbaum nach Kruskal[a]

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 (PDF 23).
[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

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

//https://www.geeksforgeeks.org/kruskals-algorithm-simple-implementation-for-
↪  adjacency-matrix/
// Simple Java implementation for Kruskal's
// algorithm

public class MinimalerSpannbaumKruskal extends GraphAdjazenzMatrix {

  public MinimalerSpannbaumKruskal(String graphenFormat) {
    super(graphenFormat);
  }

  int[] parent = new int[gibKnotenAnzahl()];

  // Find set of vertex i
  private int find(int i) {
    while (parent[i] != i)
      i = parent[i];
    return i;
  }

  // Does union of i and j. It returns
  // false if i and j are already in same
  // set.
  private void union1(int i, int j) {
    int a = find(i);
    int b = find(j);
    parent[a] = b;
```

```java
  }

  // Finds MST using Kruskal's algorithm
  public int führeAus() {
    int mincost = 0; // Cost of min MST.

    // Initialize sets of disjoint sets.
    for (int i = 0; i < gibKnotenAnzahl(); i++)
      parent[i] = i;

    // Include minimum weight edges one by one
    int edge_count = 0;
    while (edge_count < gibKnotenAnzahl() - 1) {
      int min = Integer.MAX_VALUE, a = -1, b = -1;
      for (int i = 0; i < gibKnotenAnzahl(); i++) {
        for (int j = 0; j < gibKnotenAnzahl(); j++) {
          if (find(i) != find(j) && matrix[i][j] < min && matrix[i][j] != 0) {
            min = matrix[i][j];
            a = i;
            b = j;
          }
        }
      }

      union1(a, b);
      System.out.printf("Edge %d:(%d, %d) cost:%d \n", edge_count++, a, b, min);
      mincost += min;
    }
    System.out.printf("\n Minimum cost= %d \n", mincost);
    return mincost;
  }

  public static void main(String[] args) {
    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]

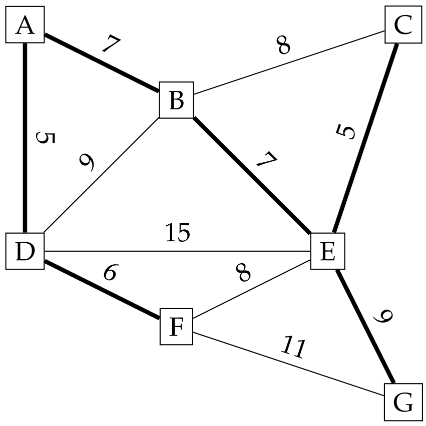
Der Algorithmus von Prim dient ebenfalls der Bestimmung eines minimalen Spannbaums, erreicht jedoch sein Ziel durch eine unterschiedliche Vorgehensweise. Er wird auch nach dem *Entdecker* Jarník genannt.[5]

Entdecker

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]

Teilgraph
schrittweise vergrößert
konkreten Startpunkt
günstigste
ausgehende Kante ausgewählt

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



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

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

/**
 * Implementation des Algorithmus von Prim / Jarník.
 *
 * Nach dem Tutorial auf <a href=
 * "https://algorithms.tutorialhorizon.com/prims-minimum-spanning-tree-mst-using-adjacency-matrix/">tutorialhorizon.com</a>.
 */
public class MinimalerSpannbaumPrim extends GraphAdjazenzMatrix {

  public MinimalerSpannbaumPrim(String graphenFormat) {
    super(graphenFormat);
  }

  /**
   * Gib den Knoten mit dem minimalen Gewicht, der sich noch nicht im minimalen
   * Spannbaum befindet.
   *
   * @param minimalerSpannbaum Ein Feld mit der gleichen Länge, wie es Knoten im
   *                           Graphen gibt. Befindet sich beispielsweise ein
   *                           Knoten mit der Index-Nummer 3 im minimalen
   *                           Spannbaum, so wird das Feld an der Index-Position 3
   *                           auf wahr gesetzt.
   * @param kantenGewichte Eine Feld mit den Kantengewichten
   *
   * @return Die ID des Knoten mit dem minimalen Gewicht. Es können Zahlen
   *         beginnend mit 0 vorkommen.
   */
  private int gibMinimumKnoten(boolean[] minimalerSpannbaum, int[] kantenGewichte) {
    int minGewicht = Integer.MAX_VALUE;
    int knoten = -1;
    for (int i = 0; i < gibKnotenAnzahl(); i++) {
      if (minimalerSpannbaum[i] == false && minGewicht > kantenGewichte[i]) {
        minGewicht = kantenGewichte[i];
        knoten = i;
      }
    }
    return knoten;
  }

  /**
   * Die Instanzen der Klasse Ergebnis wird in das Feld {@code ergebnisse}
   * gespeichert. Die Klasse speichert das Kantengewicht von einem bestimmten
```

```java
   * Knoten zu seinem Elternknoten.
   */
  class Ergebnis {
    /**
     * Die Index-Nummer des Elternknoten über den der aktuelle Knoten erreicht
↪  wird.
     */
    int eltern;

    /**
     * Das Gewicht der Kante vom Elternknoten zum aktuellen Knoten.
     */
    int gewicht;
  }

  /**
   * Führe den Algorithmus von Prim / Jarník aus.
   *
   * @return Die Summer aller Kantengewichte.
   */
  public int führeAus() {
    boolean[] minimalerSpannbaum = new boolean[gibKnotenAnzahl()];
    Ergebnis[] ergebnisse = new Ergebnis[gibKnotenAnzahl()];
    int[] gewichte = new int[gibKnotenAnzahl()];

    for (int i = 0; i < gibKnotenAnzahl(); i++) {
      // Initialisiere alle Gewichte mit Unendlich
      gewichte[i] = Integer.MAX_VALUE;
      // Erzeuge leere Ergebnis-Instanzen.
      ergebnisse[i] = new Ergebnis();
    }

    // start from the vertex 0
    gewichte[0] = 0;
    ergebnisse[0] = new Ergebnis();
    ergebnisse[0].eltern = -1;

    for (int i = 0; i < gibKnotenAnzahl(); i++) {
      int knoten = gibMinimumKnoten(minimalerSpannbaum, gewichte);
      System.out.println("Besuche Knoten „" + gibKnotenName(knoten) + """);
      minimalerSpannbaum[knoten] = true;
      for (int j = 0; j < gibKnotenAnzahl(); j++) {
        if (matrix[knoten][j] > 0) {
          if (minimalerSpannbaum[j] == false && matrix[knoten][j] < gewichte[j])
           ↪  {
            gewichte[j] = matrix[knoten][j];
            ergebnisse[j].eltern = knoten;
            ergebnisse[j].gewicht = gewichte[j];
            System.out
                .println("Aktualisiere Kante " + gibKnotenName(knoten) + "--" +
                ↪  gibKnotenName(j) + ": " + gewichte[j]);
          }
        }
      }
    }

    return gibErgebnisAus(ergebnisse);
  }

  /**
   * Gib die Ergebnisse aus.
   *
```

```java
   * @param ergebnisse Ein Feld mit allen Ergebnissen.
   *
   * @return Die Summer aller Kantengewichte.
   */
  private int gibErgebnisAus(Ergebnis[] ergebnisse) {
    int summeGewichte = 0;
    System.out.println("Minimaler Spannbaum: ");
    for (int i = 1; i < gibKnotenAnzahl(); i++) {
      System.out.println("Kante: " + gibKnotenName(ergebnisse[i].eltern) + "--" + gibKnotenName(i) + " Gewicht: "
          + ergebnisse[i].gewicht);
      summeGewichte += ergebnisse[i].gewicht;
    }
    System.out.println("Summer aller Kantengewichte: " + summeGewichte);
    return summeGewichte;
  }

  public static void main(String[] args) {
    MinimalerSpannbaumPrim prim = new MinimalerSpannbaumPrim(
        "v0--v1:2;v1--v2:3;v0--v3:6;v1--v3:8;v1--v4:5;v2--v4:7;v3--v4:9;");
    prim.gibMatrixAus();
    prim.führeAus();
  }

}
```

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 $\rightarrow$ 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`.