

## Aufgabenblatt 8

letzte Aktualisierung: 20. June, 14:04 Uhr

Ausgabe: 19.06.2016  
 Abgabe: 29.06.2016 23:59

**Thema:** Minimale Spannbäume

## Abgabe

Die folgenden Dateien müssen für eine erfolgreiche Abgabe im svn Ordner  
 Tutorien/txx/Studierende/deinname@TU-BERLIN.DE/Abgaben/  
 eingecheckt sein:

### Geforderte Dateien:

Blatt08/src/UnionFindSet.java	Aufgabe 2.1
Blatt08/src/DiGraph.java	Aufgabe 2.2

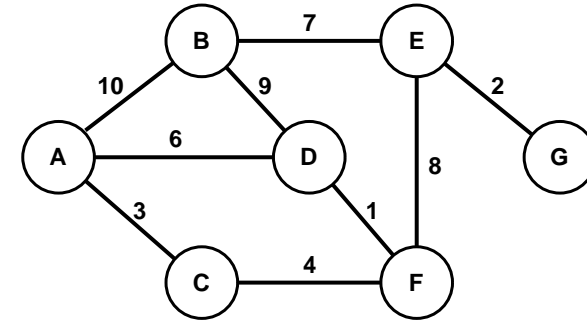
## Wichtige Ankündigungen

- Unit tests dürfen geteilt werden. Lösungen dürfen auf keinen Fall geteilt werden! Wir testen auf Plagiate.
- Verändert die Methodenköpfe vorgegebener Klassen nicht. Verändert keine Klassen die nicht Teil der Abgabe sind und fügt keine neuen hinzu.

### 1. Aufgabe: Minimale Spannbäume

Der Spannbaum  $T$  eines Graphen  $G$  ist ein Baum (d.h.  $T$  ist ein zusammenhängender, ungerichteter und zyklensfreier Graph) der alle Knoten des Graphen  $G$  verbindet, und dabei nur Kanten verwendet die auch in  $G$  vorhanden sind. Ein Spannbaum ist dann minimal, wenn die Summe seiner Kantengewichte gleich oder kleiner ist als die aller anderen möglichen Spannbäume.

Beispiel: Betrachtet den folgenden ungerichteten Graphen als Ausgangssituation:



Wie sieht der minimale Spannbaum für diesen Graph aus? Wie kann dieser berechnet werden?

**1.1. (Tut) Minimalen Spannbaum (MST) mit Prim's Algorithmus berechnen** Diskutiert kurz die Funktionsweise des in der Vorlesung vorgestellten Algorithmus von Prim und simuliert diesen auf dem gegebenen Graphen.

**1.2. (Tut) Minimalen Spannbaum (MST) mit Kruskal's Algorithmus berechnen** Diskutiert kurz die Funktionsweise des in der Vorlesung vorgestellten Algorithmus von Kruskal und simuliert diesen auf dem gegebenen Graphen.

**1.3. (Tut) Die Union-Find-Datenstruktur** Eine Union-Find bzw. Disjoint-Set Datenstruktur verwaltet disjunkte Mengen (Partitionen).

Besprecht, wie eine Union-Find Struktur funktioniert und wie sie sinnvoll für Kruskals Algorithmus verwendet werden kann.

Besprecht auch die Bedeutung der einzelnen Methoden einer Union-Find Datenstruktur:

- `add` fügt neue Partitionen, die jeweils genau ein Element enthalten, hinzu.
- `getRepresentative` findet die Partition, zu der ein bestimmtes Element gehört, und gibt dessen Repräsentanten zurück. ("find")
- `union` vereint zwei Partitionen miteinander.

### 2. Aufgabe: Minimale Spannbäume - Implementierung des Kruskal Algorithmus

Der Algorithmus von Kruskal überprüft bei jeder neu betrachteten Kante, ob deren inzidente Knoten bereits verbunden sind. Ein naiver Ansatz wäre, dies mit Hilfe einer Breiten- oder Tiefensuche ausgehend von einem der beiden beteiligten Knoten zu testen.

Mit Hilfe einer *Union-Find-Datenstruktur* kann der Algorithmus von Kruskal jedoch wesentlich effizienter implementiert werden.

**2.1. Union-Find Datenstruktur - Implementierung (40 Punkte)** Die vorgegebene Klasse `UnionFindSet.java` stellt die folgenden Methoden zur Verwaltung einer Union-Find Struktur bereit.

- `void add(T element)` Erstellt eine einzelne neue Partition mit dem übergebenem Element.
- `void add(List<T> elements)` Erstellt  $n$  Partitionen aus den  $n$  Elementen in `elements`.

- 
- `T getRepresentative(T x)` Sucht die Partition, die `x` enthält, und gibt deren Repräsentanten zurück.
  - `union(T x, T y)` Die Methode sucht die Partitionen, die `x` und `y` enthalten. Sollten die Partitionen verschieden sein, werden sie vereinigt - d.h. sie bekommen den gleichen Repräsentanten zugeordnet.

Implementiert diese Methoden entsprechend der Beschreibung!

Hinweise:

Partitionen werden in unserer Aufgabe stellvertretend durch einen in der Partition enthaltenen Knoten repräsentiert. Für alle Knoten einer Partition gibt `getRepresentative(T x)` den selben "Repräsentanten" zurück.

Es gibt mehrere Methoden zur Implementierung der Union-Find Datenstruktur. Benutzt in dieser Aufgabe die vorgegebene `HashMap element2representant` die jedem Knoten seinen Repräsentanten zuordnet!

**2.2. Kruskal's Algorithmus - Implementierung (60 Punkte)** Implementiert den Algorithmus von *Kruskal* in der Methode `toMinSpanTree()` der Klasse `DiGraph.java`. Die Methode `DiGraph toMinSpanTree()` soll den minimalen Spannbaum zurückgeben.

Verwendet in eurer Implementierung eine `PriorityQueue` für die sortierte Bearbeitung der Kanten, und ein `UnionFindSet` um hinzufügbare Kanten zu erkennen. Die benötigte Methode `compareTo` in `Edge.java` haben wir bereits implementiert.

Behandelt fehlerhafte Eingabegraphen (z.B. nicht zusammenhängende Graphen) mit dem Werfen einer `RuntimeException()`, oder indem ihr `NULL` zurückgebt.