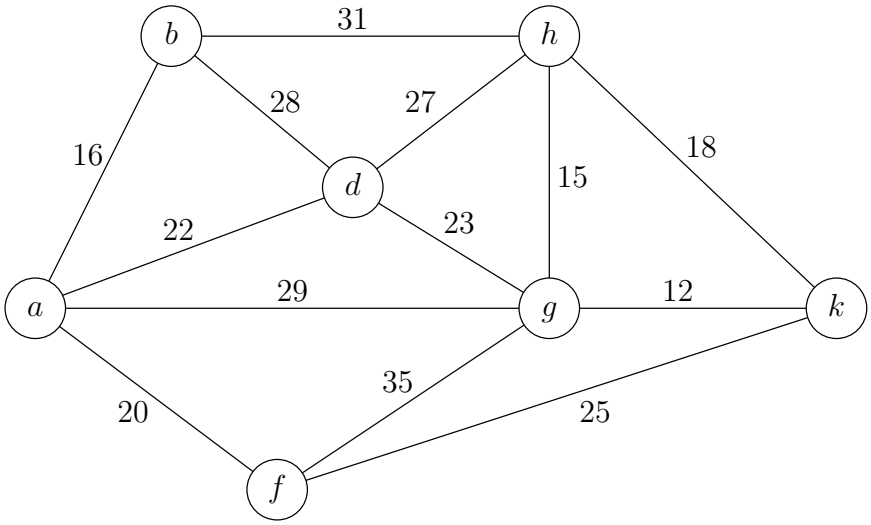
Dieses Praktikum widmet sich dem Thema Graphalgorithmen. Ziel ist die Implementierung der in der Vorlesung behandelten Algorithmen und Datenstrukturen.

Auf dem Vorlesungsserver lagert wie üblich ein ZIP Archiv mit dem Java Quellcode, der im folgenden vervollständigt bzw. erweitert wird. Das Archiv enthält folgende Klassen:

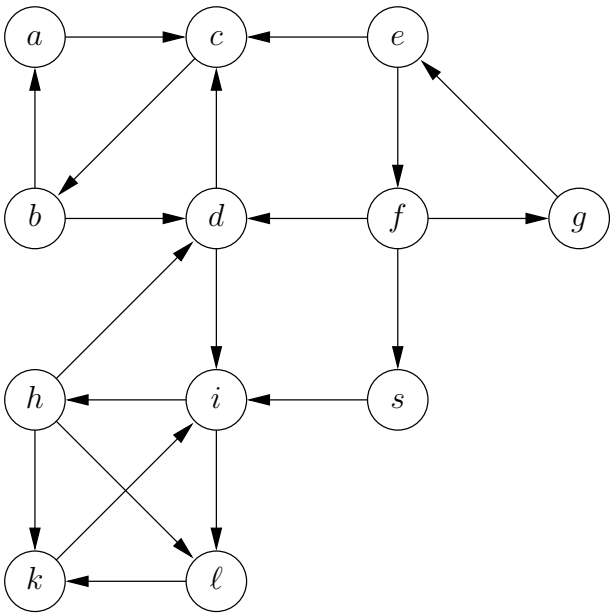
Name	Paket	Beschreibung
`Graph`	`graph`	Interface zum Zugriff auf Graphdatenstrukturen
`AdjListGraph`	`graph`	Implementierung des `Graph` Interfaces
`BreadthFirstSearch`	`graph`	Klasse zur Durchführung einer Breitensuche
`DepthFirstSearch`	`graph`	Klasse zur Durchführung einer Tiefensuche
`StronglyConnectedComponents`	`graph`	Klasse zur Berechnung der Starken Zusammenhangskomponenten in einem Graphen
`MinimumSpanningTree`	`graph`	Klasse zur Berechnung eines minimal aufspannenden Baums
`SingleSourceShortestPath`	`graph`	Klasse zur Berechnung von kürzesten Pfaden ausgehend von einem Knoten

Hinweis: Mehrere Graphalgorithmen verwenden intern eine Priority Queue. Ein Binomial Heap (siehe Praktikum 1) ist für diesen Zweck hervorragend geeignet.

Im Rahmen dieses Praktikums werden gerichtete Graphen mit gewichteten Kanten untersucht. Die Knoten selbst sind Strings. Ein Beispiel für einen solchen Graphen ist der Graph G_1:



Der Einfachheit halber werden bei Graphen das Kantengewicht nicht angegeben, wenn *alle* Kanten das Gewicht 1 haben. Ein solcher Graph ist der Graph G_2:



Aufgabe 1. Der Zugriff auf einen Graphen wird mit dem Interface `Graph` festgelegt. Studieren Sie den entsprechenden Quellkode und beantworten Sie die folgenden Fragen:

a) Wie wird die Knotenmenge des Graphen festgelegt?

b) Wie fügt man eine gerichtete Kante in den Graphen ein?

c) Gibt es eine Methode, um eine ungerichtete Kante in den Graphen einzufügen? Was passiert in einem solchen Fall?

d) Kann man eine Kante aus dem Graphen löschen?

e) Kann man einen Knoten aus dem Graphen löschen?

Aufgabe 2. Nach Festlegung des Interfaces wird nun eine Datenstruktur zum Speichern eines Graphen implementiert. Es handelt sich hierbei um eine (leicht) modifizierte Form der Adjazenzliste, bei der die von Java bereitgestellte `TreeMap` Datenstruktur eingesetzt wird. Konkret werden die Knoten und Kanten in dem Attribut

```
TreeMap<String,TreeMap<String,Float>> adj_list
```

gespeichert. Die äußere Tabelle enthält für jeden Knoten des Graphen eine Tabelle vom Typ `TreeMap<String,Float>`. Diese Tabelle speichert die Nachbarn eines Knotens (d.h. die vom Knoten ausgehenden Kanten) mit zugehörigem Gewicht.

a) Implementieren Sie in der Klasse `AdjListGraph` alle Methoden des `Graph` Interfaces.

b) Speichern Sie die Graphen G_1 und G_2 in Instanzen der Klasse `AdjListGraph`.

Aufgabe 3. Die Breitensuche ist ein elementarer Suchalgorithmus für Graphen. Dieser wird in der Klasse `BreadthFirstSearch` implementiert.

a) Untersuchen Sie die in der Klasse enthaltenen Datenstrukturen. Wie wird die Knotenfärbung, die Abstandsinformation und die Vorgängerinformation gespeichert?

b) Implementieren Sie in der Methode `search(Graph g, String s)` die Breitensuche in einem Graphen G ausgehend vom Startknoten s.

c) Führen Sie im Graphen G_2 ausgehend vom Knoten s eine Breitensuche durch.

d) Implementieren Sie die Methode `isReachable(String u)`, mit der man abfragen kann, ob der Knoten u vom Startknoten s aus erreichbar ist?

e) Sind im Graphen G_2 die Knoten a und g von s aus erreichbar?

Aufgabe 4. Ein weiterer wichtiger Suchalgorithmus ist die Tiefensuche. Für dessen Implementierung ist die Klasse `DepthFirstSearch`

a) Untersuchen Sie die in der Klasse enthaltenen Datenstrukturen. Wie wird die Färbung, die Entdeckungs- und Anschlusszeit und die Vorgängerinformation eines Knotens gespeichert?

b) Implementieren Sie die Methode `visit(Graph g, String u)`, die ausgehend vom Knoten u eine Tiefensuche durchführt.

c) Implementieren Sie die Methode `search(Graph g)` die im Graphen G eine Tiefensuche durchführt.

d) Führen Sie im Graphen G_2 eine Tiefensuche durch. Geben Sie für jeden Knoten die Entdeckungs- und Beendigungszeit sowie die Vorgängerinformation aus. Wieviele Bäume enthält der Depth-First Wald und wie sind diese aufgebaut?

Aufgabe 5. Eine interessante Anwendung der Tiefensuche ist die Berechnung der starken Zusammenhangskomponenten in einem gerichteten Graphen. Dieser Algorithmus wird durch die Methoden der Klasse `StronglyConnectedComponents` implementiert.

a) Untersuchen Sie die in der Klasse enthaltenen Datenstrukturen. Wie wird die Färbung, die Entdeckungs- und Anschlusszeit und die Vorgängerinformation eines Knotens gespeichert?

b) Implementieren Sie in der Methode `computeTranspose(Graph g)` einen Algorithmus, der den transponierten Graphen G^T von G berechnet.

c) Implementieren Sie die Methode `visit(Graph g, String u)`, die ausgehend vom Knoten u eine Tiefensuche in G durchführt.

Hinweis: Diese Methode ist identisch mit der Methode in `DepthFirstSearch`.

d) Implementieren Sie die Methode `search(Graph g)` zur Durchführung einer Tiefensuche in G, wobei die Knoten in absteigender Reihenfolge der Beendigungszeiten durchlaufen werden.

Hinweis: Um die Knoten anhand ihrer Endzeiten zu durchlaufen, wird eine `TreeMap<Integer,String> node_order` eingesetzt, in der alle Zeit-Knotenpaare gespeichert werden. Da die Map die Schlüssel in aufsteigender Reihenfolge speichert, durch läuft der Iterator die Elemente in der aufsteigenden und somit verkehrten Reihenfolge. Dies kann man verhindern, indem man vor dem Einfügen der Elemente die Zeiten negiert.

e) Implementieren Sie die Berechnung der Starken Zusammenhangskomponenten in der Methode `compute(Graph g)`.

f) Berechnen Sie die Starken Zusammenhangskomponenten des Graphen G_2.

Aufgabe 6. Der minimale Spannbaum eines Graphen wird mittels der Methoden der Klasse `MinimumSpanningTree` berechnet.

a) Untersuchen Sie die in der Klasse enthaltenen Datenstrukturen. Wie wird die Vorgängerinformation eines Knotens gespeichert?

b) Implementieren Sie in der Methode `compute(Graph g, String s)` den Algorithmus von Prim, der ausgehend von s einen minimalen Spannbaum von G berechnet.

c) Berechnen Sie mit Ihrem Algorithmus ausgehend vom Knoten a einen minimal aufspannenden Baum des Graphen G_1.

Aufgabe 7. Die letzte Aufgabe des Praktikums besteht in der Implementierung von Algorithmen zur Berechnung von kürzesten Pfaden. Hierzu dient die Klasse `SingleSourceShortestPath`.

a) Untersuchen Sie die in der Klasse enthaltenen Datenstrukturen. Wie wird die Vorgängerinformation eines Knotens gespeichert?

b) Implementieren Sie die Methode `initSingleSource(Graph g, String s)` zur Initialisierung der für die Suche benötigten Datenstrukturen.

c) Implementieren Sie die Methode `relax(Graph g, String u, String v)` zur Durchführung eines Relaxationsschritts.

d) Implementieren Sie den Algorithmus von Bellman und Ford in der Methode `bellmanFord(Graph g, String s)`.

e) Implementieren Sie den Algorithmus von Dijkstra in der Methode `dijkstra(Graph g, String s)`.

f) Berechnen Sie im Graphen G_1 alle kürzesten Pfade die vom Knoten a ausgehen. Verwenden Sie hierzu sowohl den Algorithmus von Bellman und Ford als auch den Dijkstra Algorithmus.