

Implementierung von Graphen

- Ziel ist ein Programm zur Arbeit mit Graphen
 - Speicherung der Struktur des Graphen
 - Anwendung von Algorithmen auf dem Graphen
- Optionale Ziele:
 - Einlesen des Aufbaus aus einer Datei
 - Grafische Darstellung

- Identifiziert geeignete Klassen und deren Attribute/Methoden, um einen Graphen zu repräsentieren
 - Zeichnet für jede Klasse ein UML-Diagramm
- Zu Bedenken:
 - Manche Algorithmen (z. B. Hierholzer) arbeiten auf Teilgraphen eines Graphen. Ein einzelner Knoten kann also gleichzeitig in mehreren Graphen enthalten sein

Möglicher Implementierungsansatz

Klasse Knoten Repräsentiert einen einzelnen Knoten

- Name/Label
- ggf. weitere Daten

Klasse Kante Repräsentiert eine Kante zwischen zwei Knoten

- Referenzen auf Start- und Zielknoten
- Kantengewicht
- Gerichtet/ungerichtet?

Klasse Graph Repräsentiert einen Graphen

- Liste von Knoten
- Liste von Kanten
- Methoden zur Hinzufügen/Verändern/Auslesen
- Erzeugen von Adjazenzlisten/-matrix

- Die Implementierung orientiert sich stark an der mathematischen Definition eines Graphen ($G = (V, E)$)
- Teilgraphen sind problemlos möglich
 - Jeder Teilgraph ist ein neues Objekt der Klasse **Graph**
 - Objekte vom Typ **Knoten** oder **Kante** können zu mehreren Graphen hinzugefügt werden
- Problem:
 - Wie findet man alle Kanten, die von einem bestimmten Knoten ausgehen (und somit dessen Nachbarn)?

- Ein Knoten “weiß” nichts über die Kanten, die von ihm ausgehen
- Jede Kante “kennt” ihren Anfangs- und Endknoten
- Lediglich das jeweilige **Graph**-Objekt kennt alle Kanten (und Knoten)
- Mögliche Lösung:
 - Durchlaufe die Liste aller Kanten, prüfe für jede Anfangs- und Endknoten
 - Gib alle Kanten zurück, die beim “gewünschten” Knoten anfangen
 - Aufwand bei n Kanten: $O(n)$

Die Klasse `HashMap`

- Erlaubt die Speicherung von Paaren der Form `<Key, Value>`
 - Mit der Methode `put` lässt sich ein `<Key, Value>`-Paar der `HashMap` hinzufügen
 - Die Methode `get` liefert `Value` vom angegebenen `Key` zurück
 - Beide Methoden haben konstante Laufzeit ($O(1)$)
- Beispiel:

```
HashMap<Integer, String> map = new HashMap<>();  
map.put(1, "Eins");  
map.put(528, "Fünfhundertachtundzwanzig");  
  
System.out.println(map.get(1));  
System.out.println(map.get(528));
```

HashMap für Knoten und Kanten

- Zum schnellen Auffinden von Kanten ausgehend von einem bestimmten Knoten können wir eine `HashMap<Knoten, Kante>` verwenden
 - Problem dabei: In einer `HashMap` darf jeder `Key` maximal ein Mal vorkommen, von einem Knoten können aber beliebig viele Kanten ausgehen
- Lösung:
 - Verwende stattdessen `HashMap<Knoten, LinkedList<Kante>>`
 - Für einen Knoten als `Key` liefert die `HashMap` dann eine `LinkedList` von Kanten, die von diesem Knoten ausgehen

Aufgabe 2 - Implementierung

- Schreibe ein Programm zum Arbeiten mit Graphen
 - Repräsentation von Kanten, Knoten, Graphen gemäß unserer Vorüberlegungen
 - Ausgabe von Adjazenzlisten und Adjazenzmatrix
 - Möglichkeit, Algorithmen auf den Graphen anzuwenden
- Erstelle einen Graphen aus mindestens 4 Knoten und 5 Kanten

Aufgabe 3 - Algorithmen auf Graphen

- Implementiere den Greedy-Algorithmus zur Knotenfärbung
 - Erweitere die Klasse `Knoten` um ein Attribut für die zugewiesene Farbe (bspw. als `int`)