Universität Potsdam
Institut für Computational Science und Informatik
Sommersemester 2025

Praxis der Programmierung

Hausaufgabenprojekt zur Pogrammierung in Java

Aufgabe ist die Programmierung eines ungerichteten, zusammenhängenden Graphen. Jeder Knoten im Graphen hat einen Namen und kann eine Information eines bestimmten Datentyps, der durch eine Java-Klasse definiert ist, speichern. Die Knoten in einem zusammenhängenden Graphen haben die gleichen Datentypen.

1. Definieren Sie das generische Interface `Taggable` wie in der Vorlesung vom 27. Juni.

2. Schreiben Sie eine generische Klasse `Node`, die das Interface `Taggable` implementiert. Die Klasse soll einen einzelnen Knoten in einem ungerichteten Graphen repräsentieren. Der Name des Knotens wird durch den Tag realisiert und hat einen beliebigen, aber festen Datentypen. Der Datentyp der Information, die im Knoten gespeichert ist, wird durch ein weiteres Typargument bestimmt. Zusätzlich speichert jeder Knoten eine Liste seiner direkten Nachbarknoten (Adjazenzliste). Knoten werden mit einem Initialisierungskonstruktor erzeugt, der den Tag und die Information als Parameter übergeben bekommt. Die Liste der Nachbarknoten ist bei der Initialisierung leer. Die Datenelemente sind soweit wie möglich gekapselt, Konstruktoren und Methoden sind öffentlich.

   Nutzen Sie für die Liste eine Datenstruktur aus dem Collections Framework!

3. Es gibt Getter und Setter für die im Knoten gespeicherte Information.

4. Definieren Sie zwei Methoden `connect` und `disconnect`, um einen Knoten mit einem anderen zu verknüpfen (d.h. zwei Knoten mit einer Kante zu verbinden) bzw. um eine bestehende Verknüpfung zu lösen. Beachten Sie, das Knoten maximal durch eine Kante verbunden sein können.

5. Geben Sie der Klasse `Node` eine Methode `traverse`, die alle Knoten des Graphen nach Level-Order aufsucht (beginnend am Knoten, für den die Methode aufgerufen wird) und alle gefundenen Knoten in Form einer Liste zurückgibt. Gehen Sie also nach der Strategie des Algorithmus *Breitensuche* vor.[1]
   Hinweis: Der Algorithmus Breitensuche verwendet intern eine Warteschlange (Queue). Verwenden Sie als Queue den Datentyp `LinkedList` für Knoten Ihres Graphen. Suchen Sie geeignete Methoden aus der Java-API, die die Operationen enqueue und dequeue im Datentyp `LinkedList` realisieren.

6. Definieren Sie folgende Methoden zur Ausgabe auf der Konsole:

   (a) `printNode` gibt den Tag des Knotens und die gespeicherte Information aus.

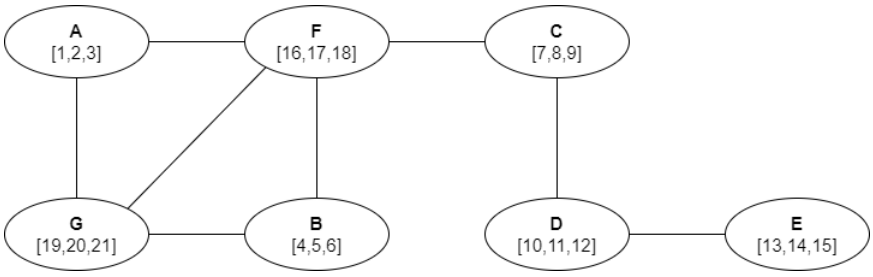
   (b) `printNeighbours` gibt die Tags der Nachbarknoten des Knotens aus.

   (c) `printNetwork` gibt den gesamten zusammenhängenden Graphen aus, startend bei dem Knoten, für den Sie aufgerufen wurde. Um den Graphen vor der Ausgabe als Liste zu erzeugen, wird die Methode `traverse` genutzt.

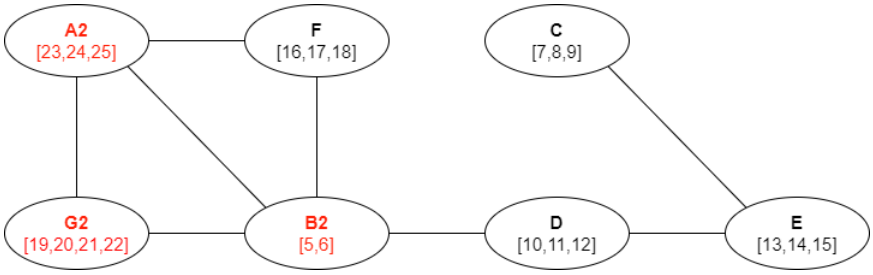
[1] Informieren Sie sich gegebenenfalls selbstständig über diesen Algorithmus, mit dem alle Knoten eines Graphen aufgesucht werden.

7. Definieren Sie eine Klasse, in der Sie Ihre Implementierungen testen. Dazu wird zuerst Graph 1 für Listen von ganzen Zahlen aufbaut und dann zu Graph 2 verändert ohne neue Knoten zu erzeugen. Geben Sie beide Graphen startend von Knoten A auf der Konsole aus.

Graph 1:



Graph 2:



Allgemeine Anforderungen an Abgaben:

- Die Lösungen müssen bis Sonntag, den 13. Juli 2025, um 23:59 Uhr auf Moodle hochgeladen werden.
- Der Kommentarblock am Anfang jedes Quellcodes muss Ihren Namen und Ihre Matrikelnummer enthalten.
- Das Java-Programm muss auf dem Linux-Server `ts-linux.cs.uni-potsdam.de` ausführbar sein. Wir empfehlen dies vor der Abgabe mindestens einmal zu testen (z.B. im PC-Pool). Jede Klasse muss sich in einer eigenen Datei befinden. Alle Bezeichner müssen den Vorgaben entsprechen. Abgaben, die dies nicht erfüllen, werden mit 'nicht bestanden' bewertet!
- Sie können zusätzliche Funktionen o.ä. implementieren, um die Implementierung des Geforderten zu vereinfachen.
- Kommentieren Sie Ihren Code angemessen.

Freiwillige Zusatzaufgabe:
Überlegen Sie sich eine sinnvolle Sortierung für die Adjazenzlisten und setzen Sie diese um.