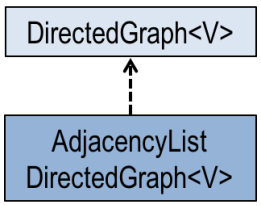
Hochschule Konstanz
Fakultät Informatik

Aufgabenblatt 2

Auf der Web-Seite finden Sie verschiedene Klassen und ein Interface, die für die Lösung folgender Teilaufgaben verwendet werden sollen. In den Klassen ist jeweils eine main-Funktion zum Testen vorhanden.

1. Realisieren Sie eine Java-Klasse **AdjacencyListDirectedGraph** für gerichtete und gewichtete Graphen. Die Klasse benutzt für die Speicherung der Nachfolgerknoten und der Vorgängerknoten jeweils eine doppelte TreeMap. Die doppelte TreeMap ordnet jedem Paar von Knoten ein Double-Wert als Gewicht zu (siehe auch Skript Seite 7-38 und 7-39).



2. Schreiben Sie eine Java-Klasse **DepthFirstOrder**, mit der eine **rekursive Tiefensuche** in einem gerichteten Graphen g durchgeführt wird (siehe auch Skript Seite 8-2, 8-3, 8-12 und 8-13). Abb. 1 zeigt einen Graphen mit zugehörigem Tiefensuchwald.

   Bei der rekursiven Tiefensuche soll eine **PreOrder-** und eine **PostOrder-Reihenfolge** der Knoten erzeugt werden. Die PreOrder-Reihenfolge ergibt sich, indem jeder Knoten, sobald er besucht wird, in eine Liste angehängt wird. Bei der Post-Order-Reihenfolge wird der Knoten erst dann in eine Liste angehängt, sobald die rekursive Besuchsmethode für den Knoten verlassen wird.

   Für den Graphen in Abb. 1 ergibt sich:

   PreOrder: 1, 2, 5, 6, 3, 7, 4
   PostOrder: 5, 6, 2, 1, 4, 7, 3

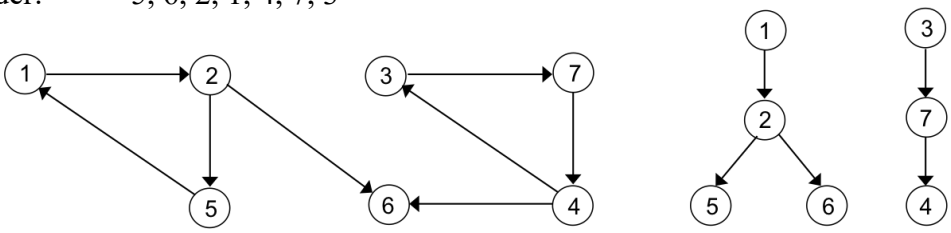


Abb. 1: Gerichteter Graph mit Tiefensuchwald

3. Schreiben Sie eine Java-Klasse **DirectedCycle,** mit der Zyklen in einem gerichteten Graphen mittels Tiefensuche erkannt werden. Setzen Sie dazu den in der Vorlesung besprochenen Algorithmus auf Seite 8-22 bis 8-24 um. Beachte: Ihr Algorithmen wird für den Graph in Abb. 2 nicht alle Zyklen finden.

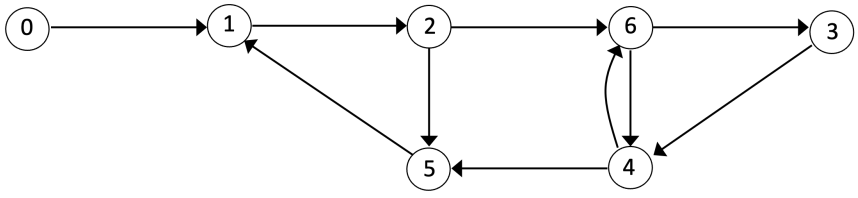


Abb. 2: Gerichteter Graph mit insgesamt 5 einfachen Zyklen

4. Schreiben Sie eine Java-Klasse **TopologicalSort** mit der ein gerichteter Graph topologisch mittels Tiefensuche sortiert werden kann. Setzen Sie dazu den in der Vorlesung besprochenen Algorithmus auf Seite 8-34 bis 8-37 um.

   Der Vorranggraph (gerichteter Graphen) in Abb. 3 beschreibt das morgendliche Anziehen im Winter. Generieren Sie mit Ihrer Klasse eine korrekte Anziehreihenfolge durch topologische Sortierung. Was liefert Ihr Algorithmus, wenn noch die (etwas abwegige) Bedingung eingehalten werden muss, dass die Hose nur mit einem Schal angezogen werden darf?

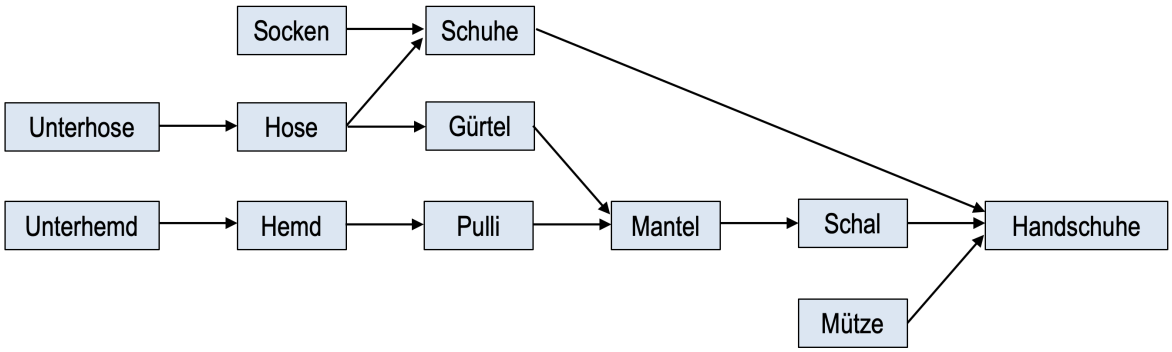


Abb. 3: Vorrangraph für morgendliches Anziehen im Winter

5. Eine **starke Zusammenhangskomponente** ist ein maximaler Teilgraph, in dem es von jedem Knoten v zu jedem anderen Knoten w einen Weg gibt. Der in Abb. 4 gezeigte Graph hat genau 4 starke Zusammenhangskomponenten. Werden die starken Zusammenhangskomponenten zu einem Knoten zusammengefasst, erhält man den **reduzierten Graph** (Abb. 5). Der reduzierte Graph muss azyklisch sein (warum?). Reduzierte Graphen helfen, die Struktur eines gerichteten Graphen wesentlich zu vereinfachen.

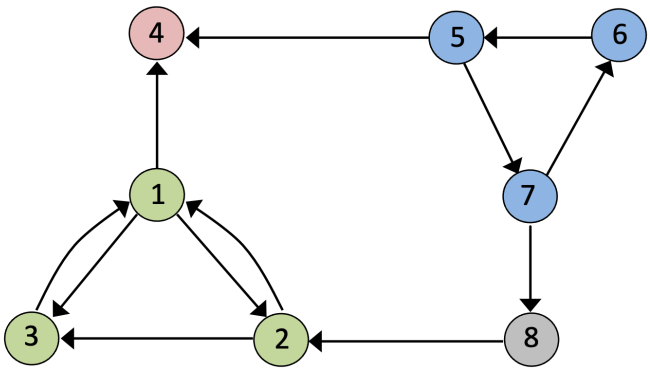


Abb. 4: Gerichteter Graph g mit farblich gekennzeichneten starken Zusammenhangskomponenten.

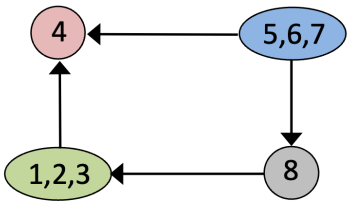


Abb. 5: Reduzierter Graph der azyklisch sein muss.

Schreiben Sie eine Java-Klasse **StrongComponents**, mit der alle starken Zusammenhangskomponenten ermittelt werden können.

Setzen Sie dazu den **Kosaraju-Sharir-Algorithmus** um:

a. Durchlaufen Sie den Graphen g in einer Tiefensuche und ermitteln Sie dabei die PostOrder-Reihenfolge p. Daraus wird die invertierte PostOrder-Reihenfolge p_i bestimmt, indem die Reihenfolge in p umgekehrt wird. (siehe Abb. 6).

b. Erzeugen Sie aus dem Graphen g den invertierten Graph g_i, indem jede Kante in umgekehrter Richtung abgespeichert wird (siehe Abb. 7).

c. Starten Sie nun eine Tiefensuche in g_i, wobei die Knoten in der obersten Tiefensuchebene (d.h. äußerste Schleife im Skript auf Seite 8-12) in der invertierten Post-Order-Reihenfolge p_i besucht werden. Jeder Baum im Tiefensuchwald ergibt dann genau eine starke Zusammenhangskomponente (siehe Abb. 8).

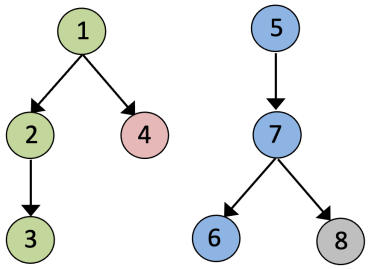


Abb. 6: Tiefensuchwald für den Graph g aus Abb. 3.
Die Post-Order-Reihenfolge p ist: 3, 2, 4, 1, 6, 8, 7, 5.
Die invertierte Post-Order-Reihenfolge p_i ist: 5, 7, 8, 6, 1, 4, 2, 3.

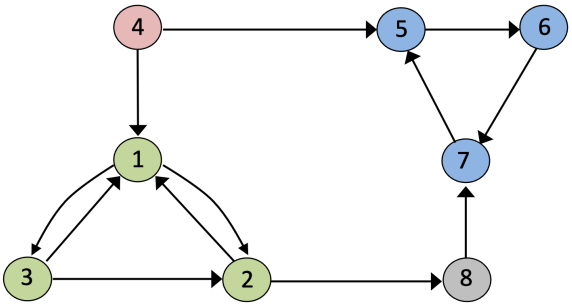


Abb. 7: Der invertierte Graph g_i.

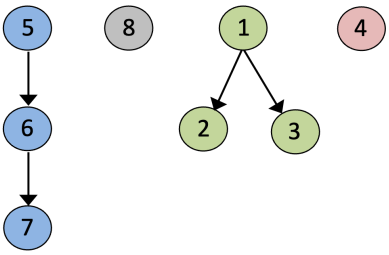


Abb. 8: Tiefensuchwald für g_i.
Die Knoten werden in der invertierten Post-Order-Reihenfolge 5, 7, 8, 6, 1, 4, 2, 3. besucht.
Die einzelnen Tiefensuchbäume ergeben die starken Zusammenhangskomponenten.

6. In der Klasse **AnalyzeWebSite** finden Sie eine Methode buildGraphFromWebSite, die aus einem Verzeichnis mit Web-Seiten alle Links ermittelt und damit einen gerichteten Graphen aufbaut. Berechnen Sie mit dem Kosaraju-Sharir-Algorithmus die Anzahl der strengen Zusammenhangskomponenten vom Verzeichnis data/WebSiteKlein (4 Web-Seiten) bzw. WebSiteGross (4000 Web-Seiten).

7. Der **PageRank**-Algorithmus ist ein Verfahren, um Gewichte (Ranks) von Web-Seiten aufgrund ihrer Link-Struktur zu ermitteln. Der Algorithmus wurde von den Google-Gründern Larry Page und Sergey Brin entwickelt und dient der Google-Suchmaschine als Grundlage für die Bewertung von Seiten. Das Verfahren ist in doc/PageRankVerfahren.pdf beschrieben.

   Implementieren Sie das Verfahren als Methode **pageRank** der Klasse **AnalyzeWebSite.**

   Welche Page-Ranks ergeben sich für WebSiteKlein und wie lautet die Top-Gerankte Seite von WebSiteGross? Besuchen Sie die Web-Seite und überzeugen Sie sich.