Übung 11 – Topologisches Sortieren & Depth First Search

1. Topologisches Sortieren

Implementieren Sie in Ihrer Graphenklasse von Übung 10 das Interface `graph.GraphAlgorithms.TopSort`. Die in diesem Interface definierte Methode `public void sort()` hat weder Parameter noch einen Rückgabewert. Sie wirft (im Gegensatz zum Pseudocode aus der Lektion) auch *keine* Exception. Vielmehr soll das Resultat auf der Konsole ausgegeben werden. Geben Sie also entweder eine topologische Reihenfolge des Graphen aus, oder eine Fehlermeldung (z.B. ob Zyklen vorkommen oder ob es sich nicht um einen gerichteten Graphen handelt). Programmieren Sie diese Methode mit Aufwand in O(|V| + |E|) so, wie wir es in der Lektion besprochen haben.

Welche zusätzlichen Attribute in welcher/welchen Klasse/n sind notwendig? Erweitern Sie die Datenstrukturen möglichst sparsam. Wenn Sie neue Methoden einführen, deklarieren Sie diese als `private`.

Zur Berechnung des Eingangsgrades können Sie entweder die `addEdge`- und `removeEdge`-Methoden ergänzen, oder zu Beginn von `sort` über alle Knoten den Eingangsgrad der Nachbarn nachführen. Für welche Variante haben Sie sich entschieden? Warum? Gibt es Unterschiede in der Effizienz?

Aufgaben:

1. Beantworten Sie auf maximal einer Seite:
   - Welche(s) zusätzlichen Attribut(e) haben Sie eingeführt?
   - Waren zusätzliche Methoden erforderlich?
   - Wie führen Sie den Eingangsgrad der Knoten nach? Beantworten Sie obige Fragen dazu.

2. Programmieren Sie `sort` in der O(|V| + |E|) - Variante.

2. Depth-First-Search

Implementieren Sie einen depth-first-search Algorithmus, indem Sie in Ihrer Graphenklasse aus Übung 10 das Interface `graph.GraphAlgorithms.DFS` implementieren. Die in diesem Interface definierte Methode `public Graph traverse(Object startVertex)` erhält als Argument einen Startknoten bei dem die Traversierung beginnen soll.

Welche zusätzlichen Attribute werden benötigt? Seien Sie auch hier möglichst zurückhaltend mit neuen Attributen, neue Methoden sollten nur `private` Zugriff benötigen.

Aufgaben:

1. Schreiben Sie jeden erstmals besuchten Knoten auf die Konsole (mit `System.out.print(o.data)`). Sie haben somit eine DFS-Ordnung erstellt. Geben Sie als Resultat den unveränderten Graphen (`return this`) zurück.

2. Erstellen Sie beim Traversieren einen Spannbaum und geben Sie diesen Graphen als Resultat zurück. Er wird dann im Editor dargestellt.
   Vorschlag zum Vorgehen:
   i. Erzeugen Sie einen neuen leeren Graphen – der Spannbaum.
   ii. Zusätzlich zur Ausgabe auf der Konsole, erzeugen Sie jeweils einen neuen Knoten im Spannbaum (Übergeben sie `addVertex` das `data`-Feld des besuchten Knotens!)
   iii. Erzeugen Sie eine Kante vom letzten besuchten Knoten zum aktuellen Knoten. Bei `addEdge` können Sie wiederum die `data`-Felder übergeben.