Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Algorithmen & Datenstrukturen (AD)

Übung: Graphen (A4)

Themen: Begriffe, Eigenschaften, Adjazenzmatrix, "Adjazenzlisten", Breitensuche, Tiefensuche, Algorithmus von Floyd, Algorithmus von Dijkstra.
Zeitbedarf: ca. 300min.

Hansjörg Diethelm, Version 1.00 (FS 2017)

1 Begriffe und Eigenschaften (ca. 30')

1.1 Lernziele

- Begriffe und Eigenschaften von Graphen kennen.
- Entsprechende Graphen beispielhaft aufzeichnen können.

1.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A51_IP_Graphen". Die Aufgabe beinhaltet keine Programmierung.

1.3 Aufgaben

Zeichnen Sie nachfolgend jeweils beispielhaft einen entsprechenden Graphen auf. Jeder Graph soll mindestens 7 Knoten beinhalten:

a) Einen gerichteten Graphen ohne Zyklen.

b) Einen ungerichteten, nicht zusammenhängenden Graphen.

c) Einen Baum mit Gewichten.

d) Verifizieren Sie an Ihrem Baum gemäss c) die Formel mit dem Zusammenhang Knoten/Kanten bei Bäumen.

e) Einen gerichteten Graphen mit zwei Zyklen.

f) Zeichnen Sie in einen ungerichteten, zusammenhängenden Graphen mit relativ vielen Kanten auf. Markieren Sie nun einen Baum, welcher den Graphen aufspannt.

g) Markieren Sie im Graph von f) zudem einen Pfad der Länge 4.

2 Besetzung eines Graphen (ca. 30')

2.1 Lernziele

- Dicht und dünn besetzte Graphen unterscheiden können.
- Zweckmässige Überlegungen betreffend deren Implementierung machen können.

2.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A51_IP_Graphen". Die Aufgabe beinhaltet keine Programmierung.

2.3 Aufgaben

a) Zeichnen Sie beispielhaft zwei ungerichtete Graphen auf, und zwar je mit mindestens 7 Knoten. Der eine soll dicht, der andere dünn besetzt sein. Machen Sie diese Aufgabe vorerst rein gefühlsmässig.

b) Mit Hilfe welcher beiden Ungleichungen differenzieren wir dicht vs. dünn besetzt? Verifizieren Sie die beiden Beispiele im Input.

c) Überprüfen Sie nun entsprechend Ihre Graphen von a). Lagen Sie richtig? Sie sollen schlussendlich zwei repräsentative Beispiele vorliegen haben.

d) Wie viele Kanten |E| kann im Maximum ein Graph mit |V| Knoten besitzen? Wie viele im Minimum?

e) Mit was für Datenstrukturen bildet man typisch dicht bzw. dünn besetzte Graphen ab? Anmerkung: Natürlich ist diese Differenzierung bei kleinen Graphen noch nicht so entscheidend.

3 Eisenbahnnetz (ca. 15')

3.1 Lernziel

- Begriffe und Eigenschaften von Graphen kennen.

3.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A51_IP_Graphen". Die Aufgabe beinhaltet keine Programmierung.

3.3 Aufgaben

Unten abgebildeter Graph repräsentiert das Eisenbahnnetz um Rotkreuz. Die Zahlenangaben entsprechen in etwa der Fahrzeit in Minuten zwischen zwei Orten/Stationen. Bestimmen Sie für untenstehenden Graphen folgende Eigenschaften:

a) gerichtet/ungerichtet?

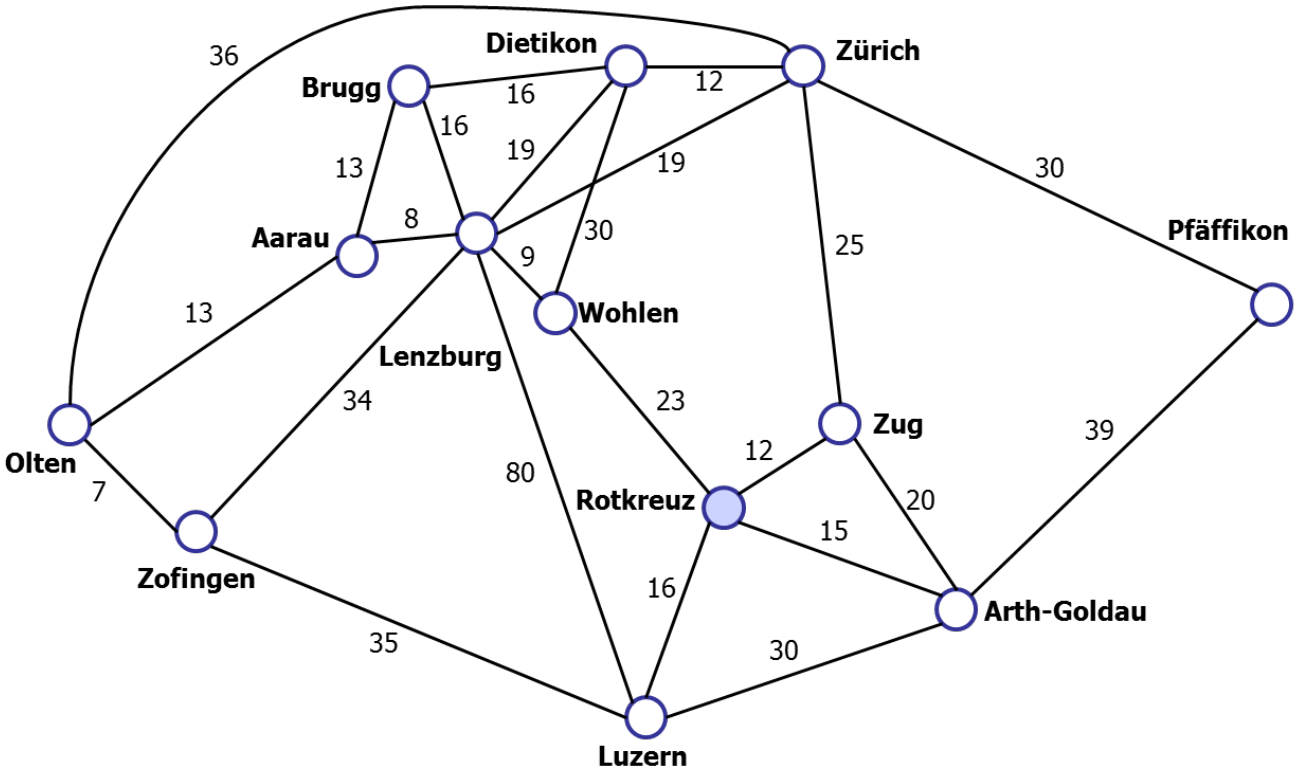
b) gewichtet/ungewichtet?

c) Zyklen ja/nein?

d) zusammenhängend ja/nein?

e) |V|=? und |E|=?

f) stark oder dünn besetzt?



4 Adjazenzmatrix (ca. 60')

4.1 Lernziel

- Graph mittels Adjazenzmatrix implementieren.

4.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A51_IP_Graphen". Der Code im AD-Input zeigt in Kürze primär das Implementationsprinzip auf. Programmieren Sie nachfolgend möglichst etwas eleganter!

4.3 Aufgaben

Es geht darum, obiges Eisenbahnnetz (Klasse `RailwayNet1`), oder zumindest einen Ausschnitt daraus, als Graph mit Hilfe einer Adjazenzmatrix zu implementieren.

a) Die Klasse soll im Wesentlichen folgende Funktionalität bieten:
   - Station hinzufügen [löschen].
   - Strecke hinzufügen [löschen].
   - Anzahl Stationen abfragen.
   - Anzahl Strecken abfragen.
   - Abfragen, ob direkte Strecke zwischen zwei Orten.
   - Für einen Ort abfragen, nach welchen allen anderen Orten direkte Strecken führen.
   - Fahrzeit für eine bestimmte direkte Strecke abfragen.

b) Testen Sie Ihre Klasse vernünftig.

5 "Adjazenzlisten" (ca. 60')

5.1 Lernziel

- Graph mittels "Adjazenzlisten" implementieren.

5.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A51_IP_Graphen". Der Code im AD-Input zeigt in Kürze primär das Implementationsprinzip auf. Programmieren Sie nachfolgend möglichst etwas eleganter!

5.3 Aufgaben

Es geht darum, das Eisenbahnnetz (Klasse RailwayNet2), oder zumindest ein Ausschnitt daraus, als Graph mit Hilfe von "Adjazenzlisten" zu implementieren. Im Input haben Sie gesehen, dass es ganz verschiedene Möglichkeiten gibt.

a) Die Klasse soll im Wesentlichen folgende Funktionalität bieten:
   - Station hinzufügen [löschen].
   - Strecke hinzufügen [löschen].
   - Anzahl Stationen abfragen.
   - Anzahl Strecken abfragen.
   - Abfragen, ob direkte Strecke zwischen zwei Orten.
   - Für einen Ort abfragen, nach welchen allen anderen Orten direkte Strecken führen.
   - Fahrzeit für eine bestimmte Strecke abfragen.

b) Testen Sie Ihre Klasse vernünftig.

6 Breitensuche (ca. 30')

6.1 Lernziele

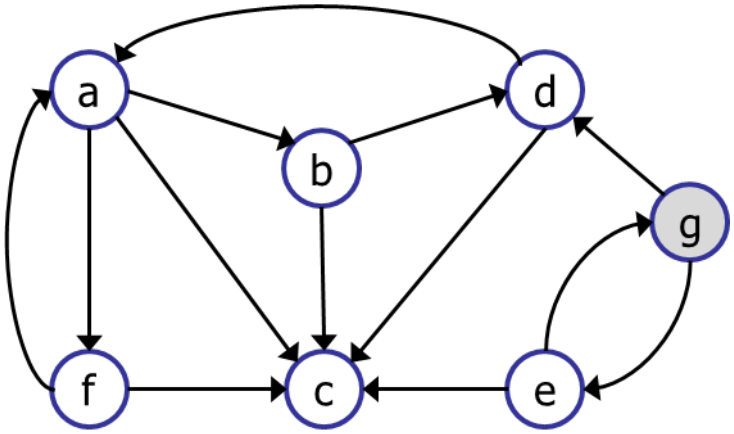
- Arbeitsweise der Breitensuche genau verstehen.
- Entsprechende iterative Umsetzungen mit einer Queue (FIFO) verstehen.

6.2 Grundlagen

Diese Aufgabe basiert auf den AD-Inputs "A52_IP_GraphenAlgorithmen" und "A51_IP_Graphen". Die Aufgabe beinhaltet keine Programmierung.

6.3 Aufgaben

a) Durchsuchen Sie nachfolgenden Graphen "auf Papier" gemäss Breitensuche, und zwar ausgehend vom Knoten g. In welcher Reihenfolge g, … werden die Knoten traversiert?



b) Ist die Reihenfolge von a) die einzige, oder gibt es noch andere korrekte "Lösungen"? Falls ja, welche?

c) Korrespondieren die Ergebnisse von a) und b) mit Spanning Trees? Muss dies in jedem Fall so sein (vgl. Input)?

d) Gehen Sie die `bfs()`-Methode bzw. den entsprechenden Code Anweisung für Anweisung gedanklich genau durch (Walkthrough). Sie können dazu das obige Beispiel heranziehen. Machen Sie sich dazu ein paar Illustrationen. Die Abarbeitung mit der FIFO-Queue `wait` müssen Sie genau verstehen.

7 Tiefensuche (ca. 30')

7.1 Lernziele

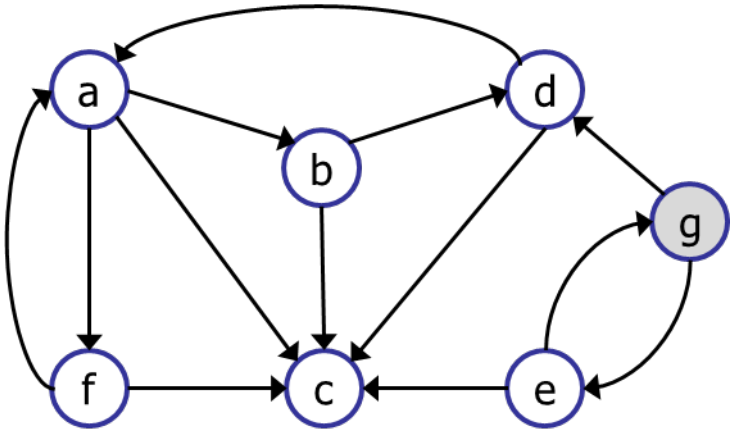
- Arbeitsweise der Tiefensuche genau verstehen.
- Entsprechende iterative Umsetzungen mit einem Stack (LIFO) verstehen.
- Rekursive Umsetzung verstehen.

7.2 Grundlagen

Diese Aufgabe basiert auf den AD-Inputs "A52_IP_GraphenAlgorithmen" und "A51_IP_Graphen". Die Aufgabe beinhaltet keine Programmierung.

7.3 Aufgaben

a) Durchsuchen Sie nachfolgenden Graphen "auf Papier" gemäss Tiefensuche, und zwar ausgehend vom Knoten g. In welcher Reihenfolge g, … werden die Knoten traversiert?



b) Ist die Reihenfolge von a) die einzige, oder gibt es noch andere korrekte "Lösungen"? Falls ja, welche?

c) Korrespondieren die Ergebnisse von a) und b) mit Spanning Trees?

d) Gehen Sie die `dfs()`-Methode bzw. den entsprechenden Code Anweisung für Anweisung gedanklich genau durch (Walkthrough). Sie können dazu das obige Beispiel heranziehen. Machen Sie sich dazu ein paar Illustrationen. Die Abarbeitung mit dem Stack bzw. mit der LIFO-Queue `wait` müssen Sie genau verstehen.

e) Kann/will man von der Rekursion Gebrauch machen, so lässt sich die Tiefensuche relativ einfach umsetzen, siehe `dfs2()`. Spielen Sie auch mit dieser Methode obiges Beispiel mit ein paar Illustrationen durch.

8 Algorithmus von Dijkstra (ca. 30')

8.1 Lernziele

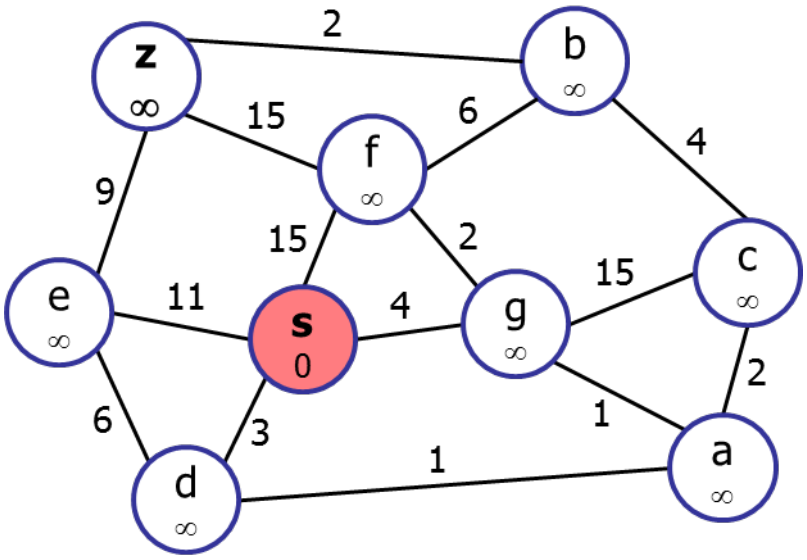
- Arbeitsweise des Algorithmus von Dijkstra genau verstehen.
- Algorithmen von Dijkstra und Floyd einander gegenüber stellen.
- Zwischen Entfernungen und Längen unterscheiden.

8.2 Grundlagen

Diese Aufgabe basiert auf den AD-Inputs "A52_IP_GraphenAlgorithmen" und "A51_IP_Graphen". Die Aufgabe beinhaltet keine Programmierung.

8.3 Aufgaben

a) Suchen Sie in folgendem Graphen mit dem Algorithmus von Dijkstra den Pfad mit der kürzesten Entfernung nach z, und zwar ausgehend vom Knoten s. Wie gross ist diese Entfernung und über welche Knoten s, …, z führt der Pfad?



b) Auch der Algorithmus von Floyd berechnet Entfernungen. Inwiefern unterscheiden sich die beiden Algorithmen?

c) Angenommen, Sie möchten nicht die kürzeste Entfernung, sondern die kürzesten Pfad-Länge bestimmen, d.h. die Anzahl Kanten. Oben ergäbe dies offenkundig die Pfad-Länge 2 von s nach z. Wie kann man diese berechnen?

9 Optional: Eisenbahnnetz II (ca. 60')

9.1 Lernziele

- Graph implementieren.
- Algorithmus von Floyd (oder Dijkstra) anwenden.

9.2 Grundlagen

Diese Aufgabe basiert auf den AD-Inputs "A52_IP_GraphenAlgorithmen" und "A51_IP_Graphen" sowie auf obigen Aufgaben.

9.3 Aufgaben

Es geht darum, eine Klasse `RailwayNet3` zu implementieren (basierend auf `RailwayNet1`), welche zusätzlich folgende Funktionalität besitzt:

a) Kürzeste Fahrzeit für eine beliebige Strecke von A nach B abfragen (Idealfall: sofort Anschluss, null Zeit fürs Umsteigen).

b) Für eine bestimmte Station alle kürzesten Fahrzeiten nach allen anderen Orten ausgeben lassen, und zwar alphabetisch sortiert nach Ort.

c) Können Sie auch eine kürzeste Rundfahrt anbieten, welche alle Orte besucht? Damit wären wir beim sogenannten Travelling Salesman Problem (TSP), eines der bekanntesten Probleme in der Informatik!

   siehe z.B. https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden