Technische Universität Wien
Institut für Computergraphik und Algorithmen
Arbeitsbereich für Algorithmen und Komplexität

186.866 Algorithmen und Datenstrukturen VU

Übungsblatt 2

für die Übung am Montag den 16. bzw. Dienstag den 17. April 2018.

Geben Sie bis **spätestens Sonntag, 15.04.2018, 13:00 Uhr** über TUWEL an, welche Beispiele Sie bearbeitet und gelöst haben. Gehen Sie dabei folgendermaßen vor:

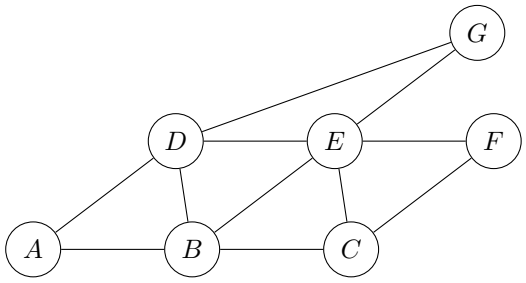
- TUWEL (`https://tuwel.tuwien.ac.at`)
  Kurs *186.866 Algorithmen und Datenstrukturen (VU 5.5)* im Abschnitt *Übungsblätter*
- Bearbeitete Beispiele ankreuzen **und** abgeben
  - Link *Ankreuzen Übungsblatt 2*
    Button *Abgabe bearbeiten*
    Bearbeitete Beispiele anhaken und *Änderungen speichern.*
  - Link *Hochladen Lösungen Übungsblatt 2*
    Button *Abgabe hinzufügen*
    PDF-Datei mit Lösungen hochladen und *Änderungen sichern.*
  - Wenn Sie das Programmierbeispiel gelöst haben:
    Link *Hochladen Source-Code Übungsblatt 2*
    Button *Abgabe hinzufügen*
    Java-Datei (`E2.java`) hochladen und *Änderungen sichern.*

Bitte beachten Sie:

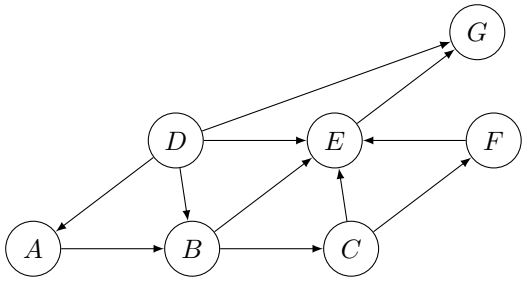
- Sie können **vor** der Deadline beliebig oft ihre Auswahl an Beispielen und das zugehörige Lösungs-PDF verändern, aber **nach** der Deadline gibt es **keine** Veränderung ihrer angekreuzten Beispiele **und** der abgegebenen Dateien!
- Sie können Ihre Lösungen entweder direkt in einem Textverarbeitungsprogramm erstellen, oder aber auch gut leserliche Scans bzw. Fotos von handschriftlichen Ausarbeitungen einreichen.
- Bitte geben Sie Ihren Namen, Matrikelnummer und E-Mail-Adresse in den Ausarbeitungen an.
- Wenn Sie das Programmierbeispiel kreuzen, muss sowohl die theoretische Ausarbeitung im PDF, als auch der Source-Code im Tuwel abgegeben werden.
- Beachten Sie die Richtlinien für das An- und Aberkennen von Beispielen in den Vorbesprechungsfolien. Neben der Überprüfung in der Übungseinheit werden danach stichprobenartig weitere Abgaben auf Spekulation und Plagiate überprüft.

Aufgabe 1 Führen Sie auf die folgenden Graphen die gegebenen Algorithmen aus. Wenn Ihnen der Algorithmus die Auswahl zwischen mehreren Knoten gibt, gehen Sie in alphabetischer Reihenfolge vor.

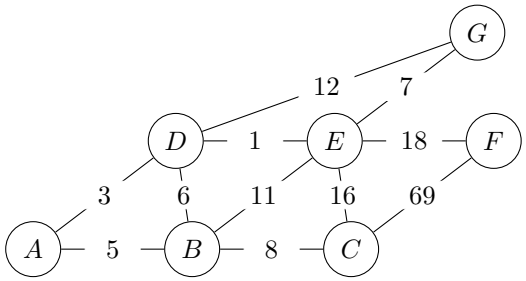
(a) Führen Sie auf dem nachfolgendem Graphen Breiten- und Tiefensuche entsprechend den Algorithmen aus den Vorlesungsfolien durch. Geben Sie dabei jeweils eine gültige Reihenfolge an, in der die Knoten besucht werden. Verwenden Sie jeweils A als Startknoten.



(b) Bestimmen Sie für den nachfolgenden Graphen eine topologische Sortierung. (Für die Abgabe genügt es, eine entsprechend gereihte Liste der Knoten anzugeben.)



(c) Finden Sie einen Minimalen Spannbaum in dem nachfolgenden Graph mittels des Algorithmen von Kruskal oder Prim. (Für die Abgabe genügt es, eine entsprechend gereihte Liste der Kanten anzugeben.)



Aufgabe 2 Ein Händler möchte seinen Laden in ein anderes Land übersiedeln. Leider kann er nicht seine ganze Ware mitnehmen, sondern maximal W Kilo. Von jeder seiner Waren weiß er, wie viele Kilo er im Bestand hat und was dieser Bestand wert ist.

Gegeben ist ein Array A so dass $A[i] = (g_i, w_i)$, wobei g_i das Gesamtgewicht des Bestands und w_i der Wert des gesamten Bestands der Ware i ist. Er kann von jeder Ware einen beliebigen Teil des Bestandes mitnehmen.

Wie viel sollte er von jeder Ware mitnehmen, um den Wert der mitgenommenen Waren zu maximieren?

(a) Finden Sie einen Greedy Algorithmus, der das beschrieben Problem löst. Geben Sie den Algorithmus in detailliertem Pseudo Code an.

Sie dürfen eine Funktion $Sort(A)$ verwenden, die ein Array A mit $A[i] = (a_1, \ldots, a_k)$ nach der Größe des ersten Eintrags sortiert und dessen Laufzeit in $O(n \log n)$ liegt.

(b) Analysieren Sie die Laufzeit Ihres Algorithmus in O-Notation und begründen Sie Ihr Ergebnis.

(c) Nehmen Sie an, dass eine eindeutige optimale Lösung existiert. Begründen Sie, dass Ihr Algorithmus diese findet. Gehen Sie dabei am besten indirekt vor.

(d) Nehmen Sie an, es muss immer entweder der gesamte Bestand einer Ware oder nichts von dieser Ware mitgenommen werden. Kann das Problem dann immer noch mit Ihrem Greedy Algorithmus optimal gelöst werden? Wenn nein, geben Sie ein Gegenbeispiel an.

Aufgabe 3 Sei $G = (V, E)$ ein ungerichteten Graph und sei F eine Menge mit k Elementen. Dann nennen wir eine Zuordnung $f : V \to F$ von Farben (F) zu Knoten (V) eine k-Färbung von G, falls Knoten, zwischen denen es eine Kante gibt unterschiedliche Farben haben. In anderen Worten: es folgt aus $(u, v) \in E$ immer $f(u) \neq f(v)$. Wir bezeichnen einen ungerichteten Graphen $G = (V, E)$ als *k-färbbar*, falls eine k-Färbung von G existiert.

Finden Sie einen Algorithmus, der in linearer Laufzeit bestimmt, ob ein gegebener Graph 2-färbbar ist. Entwickeln Sie eine Lösung in detailliertem Pseudocode und argumentieren Sie, dass der Algorithmus korrekt ist und die geforderte Laufzeitschranke gilt.

Aufgabe 4 Die Leitung eines Skigebiets behauptet, das ein durchschnittlicher Skifahrer die Berg und Talstation jedes Lifts von der Berg- oder Talstation jedes anderen Lifts in maximal einer halben Stunde erreichen kann. Für jeden Lift ist bekannt, wie lange er von der Tal- zur Bergstation braucht und für jede Piste ist bekannt, wie lange ein durchschnittlicher Skifahrer für die Abfahrt braucht. Wir nehmen zur Vereinfachung an, dass sich keine zwei Lifte eine Station teilen, dass jede Piste von einer Berg- oder Talstation zu einer anderen Berg- oder Talstation führt und dass es keine Wartezeiten bei den Liften gibt.

(a) Geben Sie an, wie man diese Situation mithilfe eines Graphen modellieren kann.

(b) Geben Sie einen Algorithmus an, der in $O(|V|^3)$ überprüft, ob ein durchschnittlicher Skifahrer tatsächlich die Berg und Talstation jedes Lifts von der Berg- oder Talstation jedes anderen Lifts in maximal einer halben Stunde erreichen kann.

Eine verbale Beschreibung das Algorithmus mit Begründung der Korrektheit und der Laufzeitschranke ist ausreichend.

Aufgabe 5 (Programmieraufgaben, 2 Punkte) Ihre Aufgabe ist die effizientere Variante des Dijkstra Algorithmus aus der Vorlesung, sowie die dazu gehörige Datenstruktur, eine Priority Queue in Form eines Min-Heaps, zu implementieren.

Die **detaillierte Angabe** zu dieser Programmieraufgabe finden Sie **im TUWEL** (PDF-Dokument „Programmieraufgabe 2“). Dieses Dokument enthält auch eine Beschreibung der zu implementierenden Funktionen und die Theoriefragen dieser Aufgabe.

Sie bekommen das benötigte Framework („AlgoDat_E2.zip“) ebenfalls im TUWEL zu Verfügung gestellt. Implementieren Sie darin in der Datei „exercise/E2.java“ die notwendigen Methoden für den Dijkstra Algorithmus, sowie für den Min-Heap.

Wichtig: Damit das Beispiel als positiv absolviert gilt, müssen **alle** geforderten Methoden korrekt implementiert und die Theoriefragen in der PDF Abgabe ausführlich beantwortet werden!
