TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Übungsblatt 2

PDF erstellt am: 23. März 2023

Deadline für dieses Übungsblatt ist **Montag, 17.04.2023, 20:00 Uhr**. Um Aufgaben für diese Übung anerkannt zu bekommen, gehen Sie folgendermaßen vor:

1. Öffnen Sie den TUWEL-Kurs der Lehrveranstaltung *186.866 Algorithmen und Datenstrukturen (VU 5.5)* und navigieren Sie zum Abschnitt *Übungsblätter*.
2. Teilen Sie uns mit, welche Aufgaben Sie gelöst haben **und** welche gelösten Aufgaben Sie gegebenenfalls in der Übungseinheit präsentieren können. Gehen Sie dabei folgendermaßen vor:
   - Laden Sie Ihre Lösungen in einem einzigen PDF-Dokument in TUWEL hoch.
     Link *Hochladen Lösungen Übungsblatt 2*
     Button *Abgabe hinzufügen*
     PDF-Datei mit Lösungen hochladen und *Änderungen sichern*.
   - Kreuzen Sie an, welche Aufgaben Sie gegebenenfalls in der Übung präsentieren können. Die Lösungen der angekreuzten Aufgaben müssen im hochgeladenen PDF enthalten sein.
     Link *Ankreuzen Übungsblatt 2*
     Button *Abgabe bearbeiten*
     Bearbeitete Aufgaben anhaken und *Änderungen speichern*.

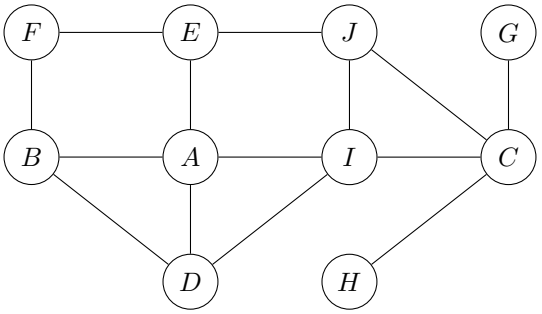
Bitte beachten Sie:

- Bis zur Deadline können Sie sowohl Ihr hochgeladenes PDF, als auch Ihre angekreuzten Aufgaben beliebig oft verändern. Nach der Deadline ist keine Veränderung mehr möglich. Es werden ausnahmslos keine Nachabgabeversuche (z.B. per E-Mail) akzeptiert.
- Sie können Ihre Lösungen entweder direkt in einem Textverarbeitungsprogramm erstellen, oder aber auch gut leserliche Scans bzw. Fotos von handschriftlichen Ausarbeitungen hochladen (beachten Sie die maximale Dateigröße).
- Beachten Sie die Richtlinien für das An- und Aberkennen von Aufgaben (Details finden Sie in den Folien der Vorbesprechung).

Aufgabe 1. Führen Sie auf dem nachfolgenden Graphen die Breiten- und Tiefensuche entsprechend den Algorithmen in den Vorlesungsfolien durch. Geben Sie dabei jeweils eine gültige Reihenfolge an, in der die Knoten besucht werden.

Zeichnen Sie zusätzlich den Breiten- und Tiefensuchbaum. Eine Kante $v \to w$ in diesen Bäumen drückt aus, dass Knoten w von Knoten v aus entdeckt wurde.

Verwenden Sie jeweils A als Startknoten. Haben Sie die Wahl zwischen mehreren Knoten, gehen Sie alphabetisch vor.



Aufgabe 2. Eine starke/schwache Zusammenhangskomponente eines gerichteten Graphen ist ein maximaler Teilgraph der stark/schwach zusammenhängend ist. Lösen Sie folgende Unteraufgaben zum Thema Zusammenhangskomponenten.

(a) Vervollständigen Sie die beiden Sätze.

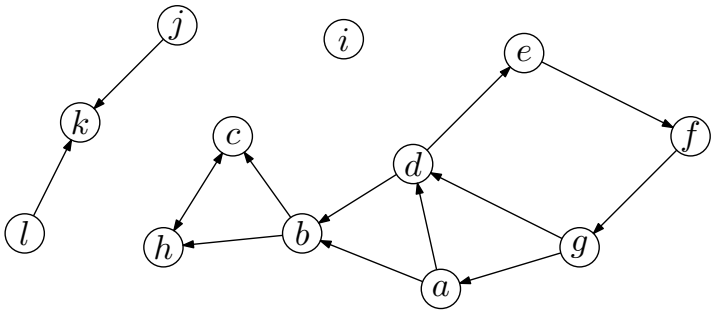
- Zwei Knoten u, v sind gemeinsam in einer starken Zusammenhangskomponente genau dann, wenn ...

- Zwei Knoten u, v sind gemeinsam in einer schwachen Zusammenhangskomponente genau dann, wenn ...

(b) Begründen Sie, dass jeder Knoten in mindestens einer schwachen Zusammenhangskomponente ist.

(c) Begründen Sie, dass jeder Knoten in höchstens einer schwachen Zusammenhangskomponente ist.

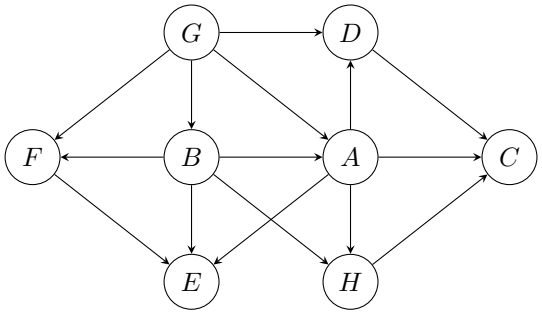
(d) Bestimmen Sie die starken und schwachen Zusammenhangskomponenten des folgenden Graphen.



(e) Gibt es einen Graphen mit mehr schwachen als starken Zusammenhangskomponenten?

Aufgabe 3. Lösen Sie folgende Unteraufgaben zum Thema topologische Sortierung.

(a) Bestimmen Sie für den nachfolgenden DAG eine topologische Sortierung. Für die Abgabe genügt es, eine entsprechend gereihte Liste der Knoten anzugeben. Haben Sie die Wahl zwischen mehreren Knoten, gehen Sie alphabetisch vor.



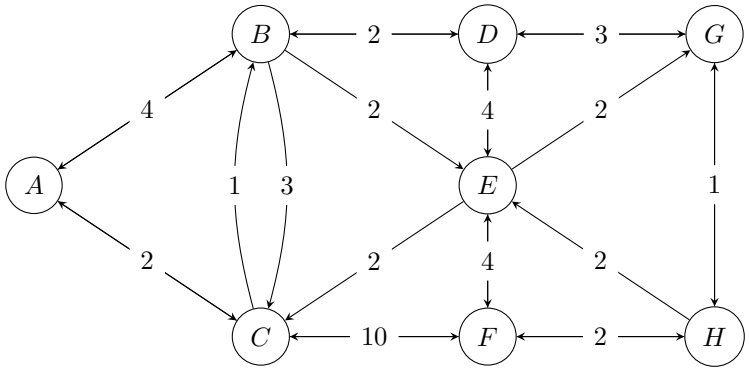
(b) Angenommen es wird beim DAG aus Unteraufgabe (a) eine Kante von H zu G hinzugefügt. Lässt sich für den resultierenden Graphen noch eine topologische Sortierung angeben? Falls ja, geben Sie diese an. Falls nein, begründen Sie kurz warum nicht.

(c) Behauptung: Ein DAG mit n Knoten hat maximal $\frac{n(n-1)}{2}$ Kanten.

Beweisen oder widerlegen Sie diese Behauptung.

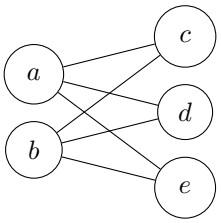
Aufgabe 4. Lösen Sie folgende Unteraufgaben zum Thema Dijkstra-Algorithmus.

(a) Führen Sie den Algorithmus von Dijkstra auf dem nachfolgenden Graphen mit A als Startknoten aus. Geben Sie die Reihenfolge an, in der die Knoten als untersucht/discovered markiert werden. Bestimmen Sie für jeden Knoten die Länge des kürzesten Pfades ausgehend von A. Geben Sie außerdem den kürzesten Pfad von A nach H an.



(b) Für den Korrektheitsbeweis des Dijkstra-Algorithmus wurde in der Vorlesung angenommen, dass alle Kanten nicht-negative Gewichte haben. Tatsächlich kann ohne dieser Annahme nicht garantiert werden, dass der Algorithmus von Dijkstra den kürzesten Pfad zwischen zwei Knoten findet. Geben Sie einen gerichteten Graphen an, sodass der Algorithmus von Dijkstra den korrekten kürzesten Pfad zwischen zwei von Ihnen gewählten Knoten in diesem Graphen nicht findet. Dazu muss zumindest eine Kante des Graphen ein negatives Gewicht haben.

Aufgabe 5. Wir bezeichnen einen ungerichteten Graphen $G = (V, E)$ als *bipartit*, wenn sich dessen Knoten in zwei Mengen V_1, V_2 aufteilen lassen, für die gilt: $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ und $E \subseteq V_1 \times V_2$. Der Graph lässt sich also in zwei Hälften teilen, sodass es nur Kanten zwischen den Knoten unterschiedlicher Hälften gibt, nicht aber zwischen den Knoten derselben Hälfte. Der nachfolgende Graph ist beispielsweise bipartit, da er sich in die Hälften $V_1 = \{a, b\}$ und $V_2 = \{c, d, e\}$ aufteilen lässt und es keine Kante zwischen a und b, bzw. c, d, und e gibt.



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

Aufgabe 6. 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 Bestandes und w_i der Wert des gesamten Bestandes der Ware i ist. Der Händler 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 beschriebene Problem löst. Geben Sie den Algorithmus in detailliertem Pseudocode an.

Sie dürfen eine Funktion $Sort(B)$ verwenden, die ein Array B, in welchem jeder Eintrag $B[i]$ ein Tupel $(b_1^i, \ldots, b_k^i)$ ist, absteigend nach der Größe des ersten Tupelwertes sortiert. Nach Ausführen von $Sort(B)$ gilt für alle $B[i] = (b_1^i, \ldots, b_k^i)$ und $B[j] = (b_1^j, \ldots, b_k^j)$ mit $i < j$, dass $b_1^i \geq b_1^j$. Die Laufzeit von $Sort(B)$ liegt in $O(n \log n)$.

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

(c) 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 ja, begründen Sie warum. Wenn nein, geben Sie ein Gegenbeispiel an.
