TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Übungsblatt 2

PDF erstellt am: 19. März 2024

Deadline für dieses Übungsblatt ist **Montag, 15.4.2024, 20:00 Uhr**. Damit Sie für diese Übung Aufgaben anerkannt bekommen können, 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* bzw. *Abgabe bearbeiten*
     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*
     Aufgaben entsprechend anhaken und *Änderungen speichern*.

Bitte beachten Sie:

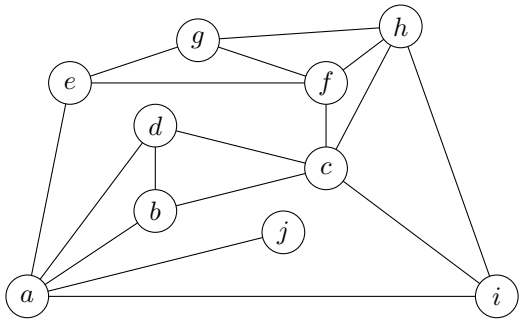
- Bis zur Deadline können Sie sowohl Ihr hochgeladenes PDF, als auch Ihre angekreuzten Aufgaben beliebig oft überschreiben. Sollte kurz vor der Deadline etwas schief gehen (Ausfall TUWEL, Internet, Scanner, etc.) und Sie die Endversion mit allen gelösten Aufgaben nicht mehr hochladen können, haben Sie zumindest Ihre Lösungen teilweise schon hochgeladen und angekreuzt. Nach der Deadline ist keine Veränderung mehr möglich. Die Deadline ist strikt – es werden ausnahmslos keine Nachabgabeversuche (z.B. per E-Mail) akzeptiert.
- Sie können Ihre Lösungen entweder direkt in einem Textverarbeitungsprogramm erstellen und hochladen, 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 dazu finden Sie in den Folien der Vorbesprechung.

Aufgabe 1.

(a) 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.

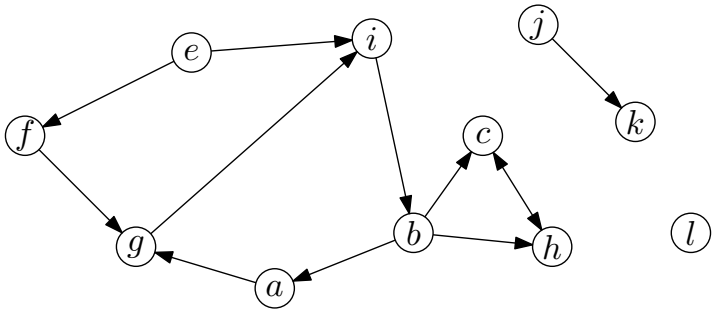
Verwenden Sie jeweils a als Startknoten. Haben Sie die Wahl zwischen mehreren Knoten, gehen Sie alphabetisch vor.



(b) Geben Sie einen zusammenhängenden Graphen mit Knoten a, b, c, d, e an, in welchem eine Breiten- und eine Tiefensuche mit Startpunkt a die Knoten zwingend in der gleichen Reihenfolge besucht.

Aufgabe 2. Lösen Sie folgende Unteraufgaben zum Thema Zusammenhangskomponenten.

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



(b) Wir wollen zeigen, dass die schwachen Zusammenhangskomponenten die Knoten des Graphen partitionieren. Begründen Sie dafür die folgenden beiden Aussagen.

- Jeder Knoten ist in mindestens einer schwachen Zusammenhangskomponente.
- Jeder Knoten ist in höchstens einer schwachen Zusammenhangskomponente.

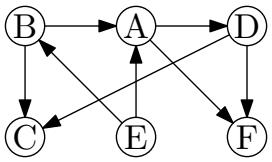
Für Interessierte: Die gleichen Aussagen gelten ebenfalls für die starken Zusammenhangskomponenten.

(c) Entwerfen Sie einen Algorithmus, der die schwachen Zusammenhangskomponenten eines gerichteten Graphen berechnet.

Hinweis: Eine Laufzeit von $O(|V| + |E|)$ ist möglich.

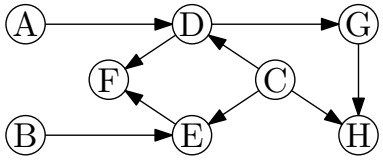
Aufgabe 3.

(a) Bestimmen Sie für den nachfolgenden DAG alle topologischen Sortierungen. Es genügt entsprechend gereihte Listen von Knoten anzugeben.



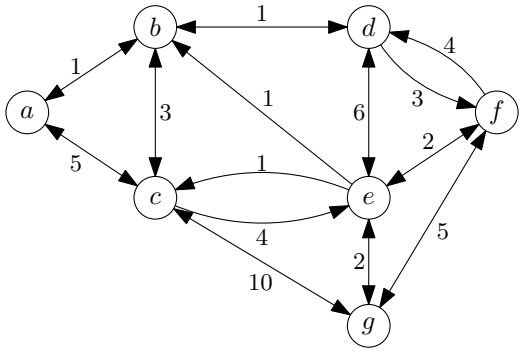
(b) Angenommen es wird beim DAG aus Unteraufgabe (a) eine Kante von C zu E hinzugefügt. Lässt sich für den resultierenden Graphen noch eine topologische Sortierung angeben? Falls ja, geben Sie diese an. Falls nein, begründen Sie kurz warum nicht.

(c) A, B, C, D, E, F, G, H ist eine gültige topologische Sortierung in folgendem DAG. Fügen Sie acht weitere Kanten ein, sodass dies weiterhin eine gültige topologische Sortierung ist.



Aufgabe 4. Lösen Sie folgende Unteraufgaben zum Thema *„Kürzeste Pfade“*.

(a) Führen Sie den Algorithmus von Dijkstra auf dem nachfolgenden Graphen mit a als Startknoten aus. Geben Sie dabei am Ende jedes Durchlaufes der äußeren Schleife den Inhalt des Distanzarrays `d` an, und welcher Knoten neu als `Discovered` markiert wurde. Geben Sie außerdem den kürzesten Pfad von a nach g an.

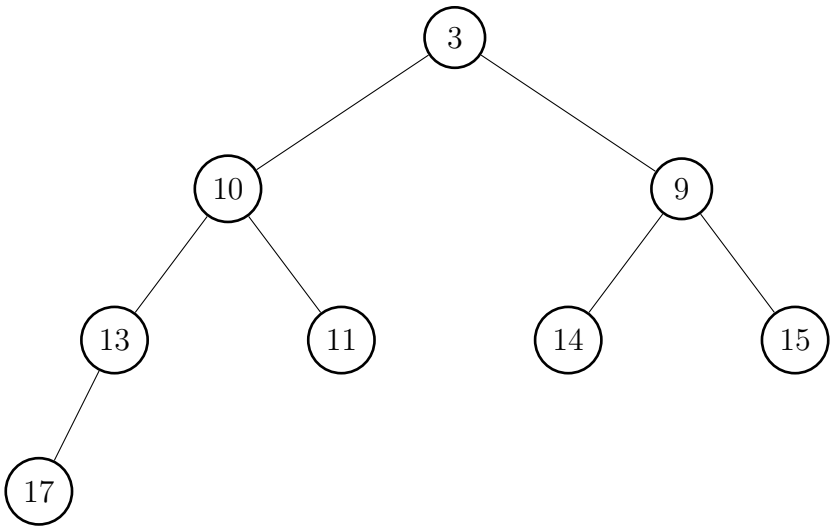


`Discovered`	`d`(a)	`d`(b)	`d`(c)	`d`(d)	`d`(e)	`d`(f)	`d`(g)
a	0	1	5	∞	∞	∞	∞

(b) Angenommen, Sie sind nicht am kürzesten Pfad zwischen a und g interessiert, sondern an dem Pfad, welcher die wenigsten Knoten beinhaltet. Im Graph von Unteraufgabe (a) wäre dies der Pfad a, c, g mit 3 Knoten. Beschreiben Sie einen Algorithmus, welcher in einem gegebenen gerichteten Graphen G die minimale Anzahl an Knoten auf einem Pfad zwischen zwei gegebenen Knoten s und t berechnet. Ihr Algorithmus soll eine strikt bessere Laufzeit als $O(|E| + |V| \log |V|)$ aufweisen.

Eine ausreichend detaillierte Beschreibung des Algorithmus mit Begründung der Korrektheit und Laufzeit genügt. Es ist kein Pseudocode notwendig.

Aufgabe 5. Gegeben ist der folgende **Min-Heap**:



(a) Entnehmen Sie zwei Mal das kleinste Element. Stellen Sie nach jedem Löschvorgang den resultierenden Heap als Baum dar. Erklären Sie dabei, wie Sie **Heapify-down** korrekt anwenden.

(b) Betrachten Sie den ursprünglichen Heap, und fügen Sie das Element 1 ein. Stellen Sie nach dem Einfügevorgang den resultierenden Heap als Baum dar. Erklären Sie dabei, wie Sie **Heapify-up** korrekt anwenden.

(c) Ein bekannter Sortieralgorithmus, der einen Heap als zentrale Datenstruktur verwendet, ist *Heapsort*. Der Algorithmus besteht aus folgenden zwei Phasen.

- Erst erstelle einen Min-Heap, der die Elemente des Eingabearrays enthält.
- Danach, solange der Heap nicht leer ist, entferne wiederholt das kleinste Element. Platziere die entfernten Elemente der Reihe nach im Ausgabearray.

Analysieren Sie die asymptotische Laufzeit dieses Sortieralgorithmus, und geben Sie eine möglichst geringe obere Schranke in O-Notation an.
