Vorlesungsprüfung

703010 VO Algorithmen und Daten Strukturen 2019

24. Juni 2019

Vorname: _____ Nachname: _____

Matr. Nr.: _____ Note: _____

Diese Klausur besteht aus 5 Aufgaben und Sie können maximal 40 Punkte erreichen. Es sind keine elektrotechnischen Geräte oder andere Hilfsmittel zulässig. Sie haben 90 Minuten Zeit und müssen *alle* Zettel nach Abschluss der Klausur abgeben. Wenn nötig, benennen Sie Ihre Annahmen. Geben Sie präzise und knappe Antworten.

This exam consists of 5 questions with a total of 40 points. The use of electronic devices is not allowed. The exam is closed book and closed notes. The total duration of this exam is 90 minutes. Participants have to return this copy and *all* additional sheets at the end of the exam. Specify your own assumptions if needed, and provide precise and concise answers.

1 Groß-O (5 Punkte)

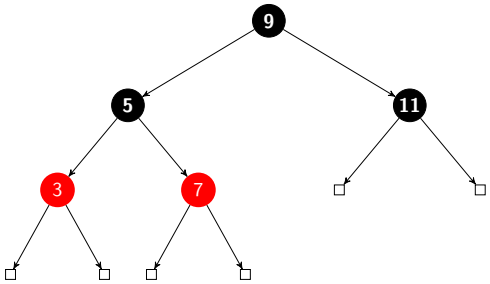
Beweisen Sie durch vollständige Induktion:
Prove by induction:

$$\sum_{i=1}^{n} i^2 \in \mathcal{O}(n^3)$$

2 Suchbäume (10 Punkte)

1. **[4 Punkte]** Was sind die 4 Eigenschaften, die ein Rot-Schwarz-Baum erfüllen muss? What are the 4 properties of a red-black tree?

2. **[6 Punkte]** Führen Sie folgende Operationen auf dem gegebenen Rot-Schwarz-Baum aus: Consider the following red-black tree and perform the operations listed below.



   a) `put(6, null)` (Schlüssel (key) 6; Daten irrelevant)

   b) `put(12, null)`

   c) `put(14, null)`

   d) `remove(5)`

3 Mehrheit (5 Punkte)

1. **[4 Punkte]** Entwerfen Sie einen Algorithmus, der das Mehrheitselement einer Liste zurückgibt, sofern es existiert. Eine Liste mit n Elementen hat ein Mehrheitselement, falls *mehr als* $n/2$ Elemente wertgleich sind. Sollte es kein Mehrheitselement geben, geben Sie *None* zurück. Es genügt für diese Aufgabe, Pseudocode zu schreiben.

   Beispiel:

   Die Liste [2, 3, 2, 3, 1, 2, 2] hat 7 Elemente. Das Mehrheitselement dieser Liste ist 2, welches viermal vorkommt.

   Die Liste [3, 1, 2, 2] hat kein Mehrheitselement, da kein Element mehr als zweimal vorkommt.

   Design an algorithm to find the majority element of a list, if any. A list of n elements is said to have a majority element, if *more than* $n/2$ elements are equal. If there is no majority element, return *None*. It is sufficient to write pseudo-code for this exercise.

   Example:

   The list [2, 3, 2, 3, 1, 2, 2] has 7 elements. The majority element of this list is 2, which occurs four times.

   The list [3, 1, 2, 2] has no majority element, as no element occurs more than two times.

2. **[1 Punkt]** Welche asymptotische Laufzeitkomplexität hat Ihr Algorithmus? Begründen Sie Ihre Antwort!

   What is the asymptotic running-time complexity of your algorithm? Justify your answer!

4 Graphen (10 Punkte)

1. Gegeben ist der unten dargestellte ungerichtete Graph. Beantworten Sie die folgenden Fragen, wobei die Adjazenzen eines Knotens immer in alphabetischer Reihenfolge berücksichtigt werden:

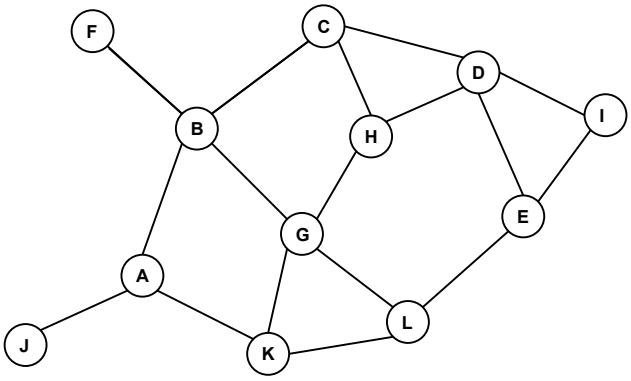
   a) **[2 Punkte]** Führen Sie eine Tiefensuche beginnend bei Knoten G durch. Notieren Sie die Knoten in der Reihenfolge, in der sie besucht werden. Zeichnen Sie auch den Graphen neu, indem Sie die Entdeckungskanten mit durchgezogenen Pfeilen und die anderen Kanten mit gestrichelten Pfeilen darstellen.

   b) **[2 Punkte]** Führen Sie eine Breitensuche beginnend bei Knoten D durch. Notieren Sie die Knoten und zeichnen Sie den Graphen wie bei der vorigen Aufgabe.

   Consider the undirected graph shown in the figure below. Answer the following questions, considering a vertex' adjacencies always in alphabetical order:

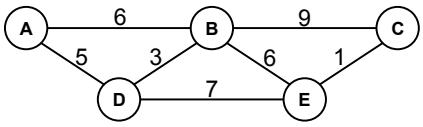
   a) Perform depth-first traversal on the graph starting at vertex G. Write down the vertices in the order they are visited. Also redraw the graph using solid arrows for the discovery edges and dashed arrows for the other edges.

   b) Perform breadth-first traversal on the graph starting at vertex D. Write down the vertices and redraw the graph as for the preceding question.



2. **[6 Punkte]** Unten sehen Sie einen gewichteten, ungerichteten Graphen. Finden Sie mittels Dijkstras Algorithmus den kürzesten Weg von Knoten A zu allen anderen Knoten. Zeigen Sie die Schritte übersichtlich, und schreiben Sie für jeden Schritt die Werte auf, die in Ihren Datenstrukturen gespeichert sind.

Consider the weighted, undirected graph shown below. Using Dijkstra's algorithm, find the shortest path from vertex A to all other vertices. Show the steps clearly, and for each step write down the values stored in your data structures.



5 Terminkalender (10 Punkte)

Die Arbeitswoche einer Beraterin ist in eine fixe Sequenz von Zeitabschnitten eingeteilt. Mittels der u.g. Methoden können sich Kunden (identifizert durch `name`) Termine (identifizert durch `datetime`) geben lassen bzw. stornieren, und die Beraterin kann ihren Terminkalender abfragen.

A consultant's work week is divided into a fixed sequence of N time slots. Clients (identified by `name`) can make or cancel appointments (identified by `datetime`) with her, and the consultant can check her calendar using the methods below.

```
datetime = makeAppointment(name)
cancelAppointment(datetime)
name = getClient(datetime)
```

Beschreiben Sie Implementierungen dieser Methoden, die jeweils in **konstanter Zeit** ablaufen. Insbesondere:

- `makeAppointment()` muss einen freien Zeitabschnitt in konstanter Zeit finden.
- `cancelAppointment()` muss den gegebenen Zeitabschnitt in konstanter Zeit finden, und muss ihn in konstanter Zeit dem Pool der verfügbaren Zeitabschnitte hinzufügen.
- `getClient()` muss ebenfalls den gegebenen Zeitabschnitt in konstanter Zeit finden.

Eine etwaige Initialisierung Ihrer Datenstrukturen sollte der asymptotischen Laufzeit der Verwaltung eines relativ vollen Terminkalenders nichts hinzufügen, d.h. sie sollte in $O(N)$ Zeit erfolgen. Beschreiben Sie Ihre Lösung auf der Basis abstrakter Datentypen, Datenstrukturen und Algorithmen, die in der Vorlesung besprochen wurden. Beschreiben Sie Ihre Lösung in präziser Sprache und/oder Pseudocode. Schreiben sie keinen Java-Code o.ä.

Describe implementations of these methods such that all run in **constant time**. In particular:

- `makeAppointment()` needs to find an available slot in constant time.
- `cancelAppointment()` needs to access the given slot in constant time, and to return it to the pool of available slots in constant time.
- `getClient()` also needs to access the given slot in constant time.

Any initialization of your data structures should not increase the overall asymptotic complexity of managing a densely-populated calendar, i.e., initialization time should be $O(N)$. Describe your solution in terms of abstract data types, data structures and algorithms covered in class. Use precise wording and/or pseudo-code. Do *not* give Java code or similar.