Vorlesungsprüfung

703010 VO Algorithmen und Daten Strukturen 2018

25. Juni 2018

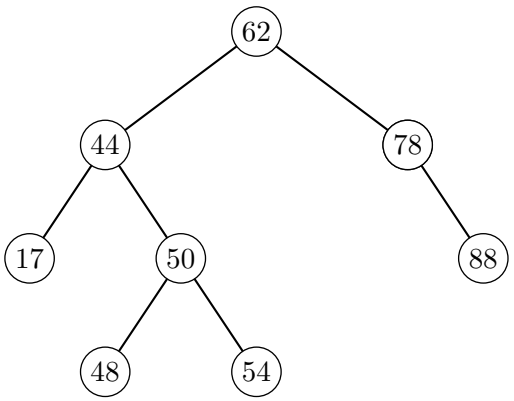
Vorname: _____ Nachname: _____

Matr. Nr.: _____ Note: _____

Diese Klausur besteht aus 5 Aufgaben und Sie können maximal 60 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 60 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 Bäume (15 Punkte)



1. Die folgenden Aufgaben beziehen sich auf den AVL-Baum T wie oben gezeigt.
   a) Zeichnen Sie den AVL-Baum, der entsteht, wenn ein Element mit Schlüssel 52 in T eingefügt wird. **[3 Punkte]**
   b) Zeichnen Sie den AVL-Baum, der entsteht, wenn das Element mit Schlüssel 62 von T entfernt wird. **[2 Punkte]**
2. Der *Balance-Faktor* einer internen Position p eines Binärbaums ist die Differenz der Höhen des rechten und linken Unterbaums von p. Zeigen Sie, wie man die Euler-Tour-Traversierung spezialisieren kann, um die Balance-Faktoren aller internen Ecken eines Binärbaums auszugeben. **[5 Punkte]**
3. Definieren Sie in Pseudocode eine nicht-rekursive Methode zur Inorder-Traversierung eines Binärbaums in linearer Zeit. **[5 Punkte]**

1. Consider the AVL tree T shown above.
   a) Draw the AVL tree resulting from the insertion of an entry with key 52 into T.
   b) Draw the AVL tree resulting from the removal of the entry with key 62 from T.
2. The *balance factor* of an internal position p of a proper binary tree is the difference between the heights of the right and left subtrees of p. Show how to specialize the Euler tour traversal to print the balance factors of all the internal nodes of a proper binary tree.
3. Describe, in pseudocode, a nonrecursive method for performing an inorder traversal of a binary tree in linear time.

2 Abhängigkeiten (10 Punkte)

Sie möchten eine Menge $\mathcal{P}$ von Softwarepaketen installieren. Jedes Paket $p \in \mathcal{P}$ hängt direkt von null, einem oder mehreren anderen Paketen p_i ab (wobei $p_i \in \mathcal{P}$ oder $p_i \notin \mathcal{P}$); sei $\mathcal{P}_p$ die Menge dieser Pakete p_i. Ebenso kann jedes Paket $q \in \mathcal{P}_p$ seinerseits von anderen Paketen abhängen, und so weiter. Verschiedene Pakete können (direkt oder indirekt) von denselben anderen Paketen abhängen. Sei $\mathcal{P}^+$ die Menge aller Pakete, die installiert werden müssen, um alle Abhängigkeiten zu erfüllen.

1. **[7 Punkte]** Geben Sie einen Algorithmus an, der alle existierenden *Abhängigkeitszyklen* findet, d.h., Fälle, wo ein Paket p (direkt oder indirekt) von einem anderen Paket abhängt, das seinerseits von p abhängt. Ihr Algorithmus sollte eine minimale Menge (direkter) Abhängigkeiten der Form „p_i hängt von p_j ab“ auflisten, so dass ein Entfernen aller in dieser Menge enthaltenen Abhängigkeiten zum Verschwinden aller Abhängigkeitszyklen führen würde.

   Um Ihren Algorithmus darzustellen, formalisieren Sie diese Aufgabenstellung in geeigneter Weise. Greifen Sie auf abstrakte Datentypen, Datenstrukturen und Algorithmen zurück, die in der Lehrveranstaltung besprochen wurden. Reproduzieren Sie keine Details von Algorithmen, die in der Lehrveranstaltung behandelt wurden.

2. **[3 Punkte]** Ermitteln und begründen Sie die asymptotische Laufzeit Ihres Algorithmus als Funktion relevanter Größen.

You want to install a set $\mathcal{P}$ of packages. Each package $p \in \mathcal{P}$ directly depends on zero or more other packages p_i (where $p_i \in \mathcal{P}$ or $p_i \notin \mathcal{P}$); let $\mathcal{P}_p$ denote the set of these packages p_i. Likewise, each package $q \in \mathcal{P}_p$ may in turn depend on further packages, and so on. Note that distinct packages may (directly or indirectly) depend on the same other packages. Let $\mathcal{P}^+$ denote the set of all packages that need to be installed to satisfy all dependencies.

1. Devise an algorithm that identifies all existing *dependency cycles*, i.e., cases where a package p (directly or indirectly) depends on some other package, which depends on p. Specifically, your algorithm should report a minimal set of (direct) dependencies of the form "p_i depends on p_j" such that, if all dependencies in this set were dropped, all dependency cycles would disappear.

   To present your algorithm, formalize this problem in a suitable fashion. Use abstract data types, data structures, and algorithms as discussed in class. Do not re-explain algorithms we discussed in class.

2. Give and justify the asymptotic running time of your algorithm as a function of appropriate variables of interest.

3 Teile und herrsche (10 Punkte)

Betrachten sie folgende rekursive Implementierung von Mergesort:

Consider the following recursive implementation of mergesort:

public class Merge public static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) if (hi ¡= lo) return; int mid = lo + (hi - lo) / 2; sort(a, aux, lo, mid); sort(a, aux, mid+1, hi); merge(a, aux, lo, mid, hi); // merges 2 sorted subarrays into a[lo..hi]. System.out.print(lo + + hi +); for (int i = lo; i ¡= hi; i++) System.out.print(a[i] +); System.out.println();
public static void sort(Comparable[] a) int N = a.length; Comparable[] aux = new Comparable[N]; sort(a, aux, 0, N-1);

Beachten Sie bitte, dass die letzten drei Zeilen der rekursiven Methode dahingehend angepasst wurden, sodass die Indizes und der Inhalt des Arrays ausgegeben werden. Die Ausgabe, welche beim Aufruf dieser Methoden mit folgendem Code, produziert wird, ist unten in nicht sortierter Reihenfolge, gegeben:

Note that the last three lines of the recursive method have been instrumented to print the values of the indices and the contents of the array. The output produced by these methods when invoked by the following code appears below in scrambled order:

```
Character[] a = { 'z', 'y', 'x', 'w', 'v', 'u', 't' , 's' , 'r' };
Merge.sort(a);
```

A. `0 2 x y z`

B. `3 4 v w`

C. `0 4 v w x y z`

D. `5 8 r s t u`

E. `0 8 r s t u v w x y z`

F. `0 1 y z`

G. `7 8 r s`

H. `5 6 t u`

1. **[7 Punkte]** Geben Sie die Reihenfolge wieder, in welcher die Ausgabe erscheint (**1 Punkt pro richtiger Ausgabe**), indem Sie jeweils einen Buchstaben auf einen der freien Plätze eintragen (der letzte ist schon befüllt).

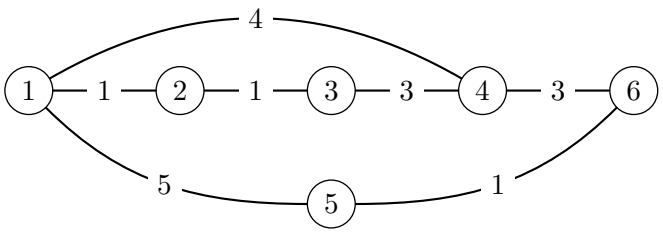
   Give the order in which these lines actually appear in the output by writing one letter in each of the blanks below (the last one is filled in for you).

   _____ _____ _____ _____ _____ _____ _____ __ E __

2. **[3 Punkte]** Betrachten Sie eine modifizierte Version von Mergesort, welche das Eingabearray anstelle von zwei in *drei* gleich große Teilarrays aufsplittet. Was ist die asymptotische Laufzeit dieses Algorithmus? Begründen Sie Ihre Antwort!

   Consider a modified version of mergesort which splits the input array into *three* equally sized sub arrays instead of just two. What is the estimated asymptotical runtime of this algorithm? Justify your answer!

4 Graphen (15 Punkte)



Betrachten Sie den gewichteten, ungerichteten Graphen G wie oben gezeichnet.

1. **[5 Punkte]** Repräsentieren Sie den Graphen G in zwei verschiedenen, geeigneten Datenstrukturen. Erklären Sie, wo und wie die Eigenschaften des Graphen G innerhalb dieser Datenstrukturen repräsentiert werden.

2. **[4 Punkte]** Durchlaufen Sie den Graphen G mit der Tiefensuche, beginnend mit Knoten 6. Wählen Sie zuerst Knoten mit größeren Zahlen. Notieren Sie in welcher Reihenfolge die Knoten besucht wurden.

3. **[6 Punkte]** Finden Sie den *längsten* Weg von Knoten 1 zu Knoten 6 im Graphen G. Verwenden Sie einen effizienten Algorithmus. Schreiben Sie die Zwischenschritte nieder und beschreiben Sie diese.

Consider the weighted undirected graph G shown above.

1. Represent the graph G with two different suitable data representations. Explain where and how the properties of the graph G are defined within these representations.

2. Traverse the graph G with depth-first search starting from vertex 6. Choose vertices with bigger numbers first. Write down in which order the vertices were visited.

3. Find the *longest* path from vertex 1 to vertex 6 in graph G. Apply an efficient algorithm. Write down and describe your intermediate steps.

5 Greedy (10 Punkte)

1. [**3 Punkte**] Beschreiben Sie ein Beispiel, in welchem ein Greedy-Algorithmus kein optimales Ergebnis liefert und erklären Sie warum.

2. [**7 Punkte**] Komprimieren Sie den String „institution“ unter Verwendung der Huffman Codierung. Zeichnen Sie den resultierenden Baum sowie zumindest zwei Zwischenschritte. Geben Sie zusätzlich den Code in tabellarischer Form an, und schreiben Sie den codierten String.

1. Come up with an example where a greedy algorithm does not return an optimal result. Explain why.

2. Compress the string “institution” using Huffman encoding. Draw the final tree and state at least two intermediate steps. Write down the resulting coding table as well as the coded string.