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

186.866 Algorithmen und Datenstrukturen VU

Übungsblatt 3

für die Übung am Montag den 7. bzw. Dienstag den 8. Mai 2018.

Geben Sie bis **spätestens Sonntag, 6.5.2017, 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 3*
    Button *Abgabe bearbeiten*
    Bearbeitete Beispiele anhaken und *Änderungen speichern.*
  - Link *Hochladen Lösungen Übungsblatt 3*
    Button *Abgabe hinzufügen*
    PDF-Datei mit Lösungen hochladen und *Änderungen sichern.*
  - Wenn Sie das Programmierbeispiel gelöst haben:
    Link *Hochladen Source-Code Übungsblatt 3*
    Button *Abgabe hinzufügen*
    Java-Datei (`E3.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

1. Sortieren Sie das folgende Array jeweils mit Merge- und Quicksort. Geben Sie die Zwischenschritte wie auf den Vorlesungfolien grafisch an.

   Wählen Sie für den Quicksort stets das letzte Element als Pivot-Element und implementieren Sie den Schritt des Aufteilens so, dass die Elemente einer Subfolge immer in der gleichen Reihenfolge angeordnet werden wie in der Originalfolge.

32	5	45	10	1	12	56	23	89	22	8

2. Im Folgenden ist die Beschreibung eines verallgemeinerten Bucket-Sort-Algorithmus für beliebige Arrays von natürlichen Zahlen gegeben, sowie der Beweis, dass dieser Algorithmus in linearer Zeit läuft. Dies widerspricht der unteren Schranke für Sortieralgorithmen aus der Vorlesung. Wo ist der Fehler im Algorithmus, bzw. der Argumentation?

   Als erstes machen wir einen Durchlauf durch das Array und kopieren jeden Eintrag k in ein Bucket-Array A_k für Einträge der Größe k, bzw erstellen das Array, falls es der erste Eintrag im Array gleich k ist. Dann kopieren die Bucket-Arrays der Größe nach in ein neues Array B. B ist dann die sortierte Form von A. Der erste Schritt ist ein Durchlauf durch das Array, kann also in linearer Zeit ausgeführt werden. Der zweite Schritt ist ein Durchlauf durch die Buckets, von denen es höchstens n gibt. Er hat also ebenfalls eine lineare Laufzeit in n.

Aufgabe 2 Gegeben ist ein Array in dem jedem Element eine der Farben Rot, Gelb oder Grün zugeordnet ist. Die Element des Arrays sollen so sortiert werden, dass erst alle roten, dann alle gelben und dann alle grünen Elemente kommen. Die einzigen Operationen, die auf dem Array ausgeführt werden können sind:

- Färbung(A, j) - gibt die Farbe des j-ten Elements von A an (Output $0, 1, 2$ für rot, gelb, grün).
- Tauschen(A, i, j) - Tauscht in dem Array A das i-te mit dem j-ten Element.

(a) Geben Sie einen Algorithmus zum lösen dieses Problems in detailierten Pseudo-Code an, der in linearer Zeit läuft. Begründen Sie warum der Algorithmus korrekt ist und analysieren Sie seine Laufzeit.

(b) Argumentieren Sie, dass es keinen Algorithmus gibt, der das Problem schneller als linear lösen kann.

Aufgabe 3

(a) Übertragen Sie den Dichtestes-Punkte-Paar Algorithmus aus der Vorlesung auf den ein dimensionalen Fall, dass heißt auf den Fall von Punkten auf einer Linie. Nehmen Sie an es, gibt den Datentyp Punkt und den Datentyp Linie. Punkte haben einen eindeutigen Schlüssel, der sie identifiziert. Außerdem gibt es die Funktion $\text{Dist}(p_1, p_2)$, die die Distanz zwischen zwei Positionen p_1 und p_2 ausgibt (als reele Zahl). Die Positionen können, ein Punkt, der Anfang oder das Ende eine Linie sein. $\text{Dist}(p_1, p_2)$ läuft in konstanter Zeit.

Eine Linie hat einen Anfang und einen Ende sowie eine Liste mit beliebig vielen Punkten, die dazwischen liegen. Ein Punkt p liegt zwischen dem Anfang a und Ende e, falls $\text{Dist}(a, p)+\text{Dist}(p, e) = \text{Dist}(a, e)$ gilt. Wir nehmen an, dass die Größe einer Linie der Anzahl der enthaltenen Punkte entspricht. Die folgenden Operationen werden unterstützt:

- Die Operationen $\text{Split}(L)$ gibt zwei Objekte des Datentyps Linie L_1, L_2 aus, so dass L_1 alle Punkte p enthält, die in der ersten Hälfte der Linie L liegen, dass heißt $\text{Dist}(a, p) \leq \text{Dist}(p, e)$ und L_2 alle Punkte enthält, die in der zweiten Hälfte der Linie L liegen, dass heißt $\text{Dist}(a, p) > \text{Dist}(p, e)$. Die Operation braucht lineare Zeit in der Größe von L.
- $\text{Erster}(L)$ gibt den Punkt der Linie L aus, der am nächsten am Anfangspunkt liegt. Die Operation braucht lineare Zeit in der Größe von L.
- $\text{Letzter}(L)$ gibt den Punkt der Linie L aus, der am nächsten am Endpunkt liegt. Die Operation braucht lineare Zeit in der Größe von L.

Geben Sie den Dichtestes-Punkte-Paar Algorithmus für den eindimensionalen Fall in detailliertem Pseudocode an. Ihr Algorithmus soll eine Linie als Input erhalten und die Distanz zwischen den zwei Punkten, mit dem geringsten Abstand zueinander, ausgeben.

(b) Nehmen Sie nun an, dass Punkte durch ganze Zahlen repräsentiert werden. Linien haben jeweils eine ganze Zahl als Anfang und Ende, sowie ein (unsortiertes) Array aus Punkten die zwischen Anfang und Ende liegen, gespeichert als ihre entsprechende Zahl. Beschreiben Sie wie die vier oben angegeben Operationen (Split, Dist, Erster, Letzter) implementiert werden können, sodass die oben gegebenen Laufzeitschranken erfüllt werden.

Bei Linien dürfen Sie davon ausgehen, dass der Anfang echt kleiner als der Endpunkt ist.

(c) Nehmen Sie an, Linie und Punkt sind wie in (b) beschrieben implementiert. Beschreiben Sie einen alternativen Algorithmus für das Dichtestes-Punkt-Paar Problem, der die selbe Laufzeit wie der Divide and Conquer Algorithmus aus (a) hat. Sie dürfen dabei das Sortieren von ganzen Zahlen (mit Laufzeit $O(n \log(n))$) als gegebene Operation verwenden.

Aufgabe 4

(a) Gegeben sei ein leerer Binärbaum B über die natürlichen Zahlen und zwei Arrays *In* und *Post*. Letztere sind die entsprechenden Inorder- und Postorder-Darstellungen eines unbekannten Binärbaums B'. B' enthält keine Duplikate. Geben Sie einen Algorithmus in detailliertem Pseudo-Code an, der B, *In* und *Post* entgegennimmt und mit $B = B'$ terminiert, d.h., welcher den unbekannten Baum aus den beiden Array-Darstellungen rekonstruiert.

Ihnen stehen die Methoden $B.insertRoot(x)$, $B.insertLeftChild(parent, x)$, $B.insertRightChild(parent, x)$ auf Binärbäumen und *length* und $find(x)$ auf Arrays zur Verfügung. $find(x)$ gibt den Index eines Elements x im jeweiligen Array zurück. Die Operationen auf dem Binärbaum geben jeweils den eingefügten Knoten zurück. Weiters können Sie Subarrays über $subArray \leftarrow Array[startIndex..endIndexInklusiv]$ erzeugen. Wenn $endIndexInklusiv < startIndex$ bzw. $startIndex >= Array.length$, dann wird dabei ein leeres Array zurückgegeben.

(b) Wenden Sie diesen Algorithmus auf folgende Eingabe an und zeichnen Sie den rekonstruierten Binärbaum auf.

$$In = [5, 4, 1, 6, 12, 7, 3, 8, 9, 13]$$
$$Post = [4, 5, 6, 12, 1, 3, 9, 8, 13, 7]$$

Aufgabe 5 Geben Sie eine Zahlenfolge an, welche Sie in der gegebenen Ordnung in einen zu Beginn leeren AVL-Baum einfügen, sodass zumindest eine Einfach- und eine Doppelrotation passiert. Zeichnen Sie den Baum inkl. Balancen nach jedem Einfügen eines Elements und nach jeder Rotationsoperation auf. Beschreiben Sie genau die Zwischenschritte der Rotationsoperationen.

Aufgabe 6 (Programmieraufgaben) Ihr Aufgabe ist es, den aus der Vorlesung bekannten Quicksort-Algorithmus zu adaptieren, sodass dieser anstatt zu sortieren das k-te Element zurück gibt. Dies vergleichen Sie dann mit dem naiven Ansatz einer vollständigen Sortierung.

Die **detaillierte Angabe** zu dieser Programmieraufgabe finden Sie **im TUWEL** (PDF-Dokument „Programmieraufgabe 3“). Dieses Dokument enthält auch eine Beschreibung der zu implementierenden Funktionen und die Theoriefragen dieser Aufgabe.

Sie bekommen das benötigte Framework („AlgoDat_E3.zip“) ebenfalls im TUWEL zur Verfügung gestellt. Implementieren Sie darin in der Datei „exercise/E3.java“ die geforderten Methoden.

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