Lehrstuhl für Algorith.
und Datenstrukturen
Prof. Dr. Hannah Bast
Björn Buchhold

Algorithmen und Datenstrukturen
(ESE) WS 2011 / 2012

http://ad-wiki.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Klausur

Freitag 30. März 2012, 14:00 - 16:00 Uhr, Gebäude 101, HS 026

Es gibt 4 Aufgaben. Für jede Aufgabe gibt es maximal 20 Punkte. Sie haben insgesamt 2 Stunden Zeit. Das sind pro Aufgabe im Durchschnitt eine halbe Stunde.

Wie in der Vorlesung erklärt, werden die Punkte aus den Übungsblättern ebenfalls auf eine Punktzahl von bis zu maximal 20 Punkten umgerechnet. Falls diese Punktzahl besser ist als die kleinste Punktzahl einer der vier Aufgabe aus der Klausur, wird letztere durch erstere ersetzt.

Sie dürfen eine beliebige Menge an Papier, Büchern, etc. verwenden. Sie dürfen keinerlei elektronische Geräte wie Notebook, Mobiltelefon, etc. verwenden, insbesondere keine Geräte, mit denen Sie mit Dritten kommunizieren oder sich mit dem Internet verbinden können.

Wichtig 1: Wann immer in den folgenden Aufgaben gesagt wird „Bestimmen Sie ...“, oder nach einem bestimmten Wert oder einer bestimmten Lösung gefragt wird, sollen Sie diese selbstverständlich immer begründen!

Wichtig 2: Wann immer in den folgenden Aufgaben nach Code gefragt wird, können Sie diesen in Java oder C++ schreiben. Nicht-offensichtliche Passagen in Ihrem Code sollten Sie erklären.

Wichtig 3: Schreiben Sie bitte auf jedes Blatt Papier, das Sie abgeben, oben gut lesbar Ihren Namen (in Blockbuchstaben bitte) und Ihre Matrikelnummer, und numerieren Sie die Blätter vor der Abgabe bitte durch.

Tipp: Verbringen Sie nicht zu viel Zeit mit einer einzelnen Teilaufgabe, wenn Sie dort nicht weiterkommen. Machen Sie erstmal mit einer anderen (Teil)aufgabe weiter, und widmen Sie sich dann am Ende, wenn Sie noch Zeit haben, den Teilen, wo Sie Schwierigkeiten hatten.

Wir wünschen Ihnen viel Erfolg!

Aufgabe 1 (Hashing, 20 Punkte)

1.1 (5 Punkte) Betrachten Sie die Hashfunktion $h(x) = x^2 \bmod 4$ für eine Hashtabelle mit 4 Positionen. Zeichnen Sie den Zustand der Hashtabelle nach dem Einfügen der 8 Elemente $1, 2, 3, 4, 5, 6, 7, 8$.

1.2 (5 Punkte) Beweisen Sie, dass alle Schlüssel aus der Menge der natürlichen Zahlen von der Hashfunktion aus Aufgabe 1.1 entweder auf 0 oder auf 1 abgebildet werden, während 2 und 3 als Wert nie vorkommen.

1.3 (5 Punkte) Schreiben Sie den Code für eine Methode *remove* die für einen gegebenen Schlüssel das Element mit diesem Schlüssel aus der Hashtabelle entfernt. Wenn kein Element mit dem gegebenen Schlüssel vorhanden ist, braucht die Funktion nichts zu tun.

Sie können dabei folgende Methoden / Membervariablen als gegeben voraussetzen: (1) eine Methode *hash*, die den Hashwert für einen Schlüssel berechnet; (2) eine Membervariable *hashTable* für die Liste der Elemente, die an den einzelnen Positionen der Hashtabelle gespeichert sind; (3) eine Methode *remove* zum Löschen eines Elementes aus einem Feld / einer Liste.

1.4 (5 Punkte) Sei das Schlüsseluniversum wieder die Menge aller natürlichen Zahlen. Betrachten Sie jetzt die Hashtabellengröße $m = 2$ und die folgenden beiden Hashfunktionen: $h_1(x) = x \bmod 2$ und $h_2(x) = x + 1 \bmod 2$. Ist die Klasse $H = \{h_1, h_2\}$ 1-universell? Beweisen Sie die Aussage, oder wiederlegen Sie sie durch ein Gegenbeispiel.

Aufgabe 2 (a,b-Bäume, 20 Punkte)

2.1 (5 Punkte) Malen sie den Zustand eines $(2, 4)$-Baumes nach dem Einfügen der 10 Elemente $1, 2, 3, \ldots, 10$ in dieser Reihenfolge. Vor dem Einfügen dieser Elemente ist der Baum leer.

2.2 (5 Punkte) Nehmen wir an, wir fügen weiter Elemente in aufsteigender Reihenfolge ein (ohne Beschränkung der Allgemeinheit seien das $11, 12, 13, \ldots$). Bei dem Einfügen welchen Elementes bekommt der Baum zum ersten Mal Tiefe 3 (das heißt, die Pfade von der Wurzel zu einem Blatt haben 3 Kanten)?

Wenn man die Zahlen in einer anderen Reihenfolge einfügt, bekommt der Baum erst später Tiefe 3. Geben Sie eine Reihenfolge von Zahlen an, so dass der Baum (wenn man beginnend mit dem leeren Baum diese Zahlen in dieser Reihenfolge einfügt) so spät wie möglich Tiefe 3 bekommt.

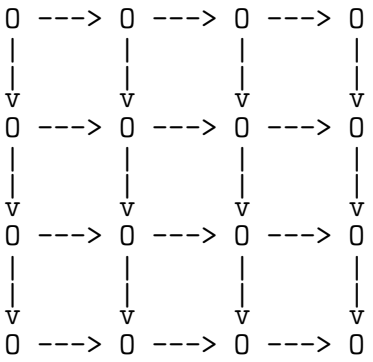
2.3 (5 Punkte) Wieviele Verschmelzungen sind notwendig wenn man nach Aufgabe 2.1 die 10 Elemente in umgekehrter Reihenfolge $(10, 9, 8, \ldots, 1)$ wieder entfernt?

Hängt die Anzahl der notwendigen Verschmelzungen von der Reihenfolge ab, in der man die Elemente entfernt? Vergessen Sie nicht, Ihre Antwort zu begründen!

2.4 (5 Punkte) Wofür braucht man bei unserer Definition des allgemeinen (a, b)-Baumes die Bedingung $b \geq 2a-1$? Und warum brauchen wir bei unserer Analyse die etwas stärkere Bedingung $b \geq 2a$?

Aufgabe 3 (Breitensuche, 20 Punkte)

3.1 (6 Punkte) Betrachten Sie folgenden Gittergraphen, bei dem die 16 Knoten durch Ihre Gitterkoordinaten benannt sind. So heißt der Knoten oben links $(1, 1)$, der Knoten unten links $(4, 1)$, der Knoten oben rechts $(1, 4)$ und der Knoten unten rechts $(4, 4)$. Führen Sie Breitensuche auf diesem Graphen aus, mit Startknoten $(1, 1)$. Malen Sie dazu den Graphen und markieren Sie die einzelnen Level der Breitensuche.



3.2 (7 Punkte) Betrachten Sie jetzt einen allgemeinen Gittergraphen von der Art wie in Aufgabe 3.1, der die n^2 Knoten (i, j) enthält, für alle $i = 1, \ldots, n$ und $j = 1, \ldots, n$. Bestimmen Sie die Anzahl der Level bei einer Breitensuche auf diesem Graphen mit Startknoten $(1, 1)$ sowie welche Knoten in welchem Level enthalten sind. Vergessen Sie nicht, dass der Startknoten einen eigenen Level (Level 0) bildet.

3.3 (7 Punkte) Nehmen Sie an, Sie haben eine Klasse *Graph* für einen ungerichteten Graphen. Nehmen Sie weiterhin an, diese Klasse hat eine Methode *doBreadthFirstSearch*, die Ihnen für einen gegebenen Startknoten ein Feld / eine Liste mit allen Knoten zurückgibt, die von diesem Knoten aus erreicht werden können (eingeschlossen der Startknoten selber). Schreiben Sie dann eine Methode *computeNumberOfConnectedComponents*, die die Anzahl der Zusammenhangskomponenenten des Graphen berechnet.

Aufgabe 4 (O-Notation / IO-Effizienz / Editierdistanz, 20 Punkte)

4.1 (6 Punkte) Betrachten Sie die folgenden fünf Funktionen von n:

$$n \cdot \log n,$$
$$n^2,$$
$$n \cdot \log(n^2),$$
$$n \cdot (\log n)^2,$$
$$n.$$

Bringen Sie die Funktionen so in eine Reihenfolge f_1, f_2, f_3, f_4, f_5, dass $f_i = O(f_{i+1})$, für $i = 1, 2, 3, 4$. Bestimmen Sie außerdem für welche i gilt dass $f_i = \Theta(f_{i+1})$ und für welche nicht.

Selbstverständlich sollen Sie alle Ihre Entscheidungen begründen, insbesondere auch die i wo *nicht* $f_i = \Theta(f_{i+1})$. Sie können für alle Ihre Begründungen die Grenzwert-Definition von O und Θ benutzen.

4.2 (7 Punkte) Betrachten Sie den Algorithmus aus der Vorlesung zur Berechnung der Editierdistanz zwischen zwei gegebenen Strings mit dynamischem Programmieren. Bestimmen Sie die Größenordnung, also $\Theta(\ldots)$, der Anzahl der *Block*operationen in Abhängigkeit von den Längen n und m der beiden Strings und der Blockgröße B.

Sie können dabei annehmen, dass die Blockgröße B ein Vielfaches von sowohl n als auch m ist, und dass $M \geq 4B$.

4.3 (7 Punkte) Nehmen wir jetzt an, wir haben für gegebene Strings x und y bereits eine Tabelle P berechnet mit folgenden Werten für $i = 0, \ldots, |x|$ und $j = 0, \ldots, |y|$. Dabei bezeichnet $\mathrm{ED}(x[1..i], y[1..j])$ die Editierdistanz zwischen dem Präfix der Länge i von x und dem Präfix der Länge j von y.

$P[i,j] = 1 \;\Rightarrow\; \mathrm{ED}(x[1..i], y[1..j]) = \mathrm{ED}(x[1..i], y[1..j-1]) + 1$

$P[i,j] = 2 \;\Rightarrow\; \mathrm{ED}(x[1..i], y[1..j]) = \mathrm{ED}(x[1..i-1], y[1..j]) + 1$

$P[i,j] = 3 \;\Rightarrow\; \mathrm{ED}(x[1..i], y[1..j]) = \mathrm{ED}(x[1..i-1], y[1..j-1]) + 1$

$P[i,j] = 4 \;\Rightarrow\; \mathrm{ED}(x[1..i], y[1..j]) = \mathrm{ED}(x[1..i-1], y[1..j-1])$

(Falls es mehrere Möglichkeiten gibt, $\mathrm{ED}(x[1..i], y[1..j])$ aus einem der Werte auf der rechten Seite zu berechnen, wird in P nur eine der Möglichkeiten abgespeichert. Der Wert von $P[0,0]$ ist beliebig und spielt keine Rolle.)

Schreiben Sie eine Methode *outputSequenceOfOperations*, die, gegeben ein solches P, eine Folge von $\mathrm{ED}(x, y)$ Operationen (zum Beispiel: *insert(2, 'A')*, *replace(4, 'D')*, *delete(5)*, ...) ausgibt, um x in y zu überführen. Es ist ok, wenn Sie die Folge in umgekehrter Reihenfolge ausgeben.