Lehrstuhl für Algorith.
und Datenstrukturen
Prof. Dr. Hannah Bast
Claudius Korzen

Algorithmen und Datenstrukturen
(ESE + IEMS) WS 2012 / 2013

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

UNI FREIBURG

Klausur

Samstag 23. März 2013, 11:00 - 13:00 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 30 Minuten.

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.

Schreiben Sie Ihre Lösungen bitte wenn möglich auf die Klausur!

Benutzen Sie dabei für jede Aufgabe zuerst die Vorderseite und dann die Rückseite. Wenn Sie zusätzliches Papier benötigen, schreiben Sie auf jedes dieser Blätter bitte oben rechts gut lesbar Ihren Namen (in Blockbuchstaben bitte) und Ihre Matrikelnummer.

Tipp: Verbringen Sie nicht zu viel Zeit mit einer einzelnen Teilaufgabe. Wenn Sie nicht weiterkommen, machen Sie erstmal mit einer anderen (Teil)aufgabe weiter. Widmen Sie sich dann am Ende, wenn Sie noch Zeit haben, den Teilen, bei denen Sie Schwierigkeiten hatten.

Wir wünschen Ihnen viel Erfolg!

Aufgabe 1 (Hashing, 20 Punkte)

Sei h eine Hashfunktion, die eine beliebige Zeichenkette x über dem Alphabet $\{a, b\}$ auf den Wert $(3 \cdot n_a + n_b) \bmod 4$ abbildet, wobei n_a die Anzahl der Vorkommen von a und n_b die Anzahl der Vorkommen von b in x ist. Zum Beispiel ist $h(\text{"}ababa\text{"}) = (3 \cdot 3 + 2) \bmod 4 = 3$.

1.1 (5 Punkte) Zeichnen Sie den Zustand einer Hashtabelle mit 4 Positionen nach dem Einfügen der folgenden Zeichenketten unter Verwendung der obigen Hashfunktion: *ab*, *aab*, *aabb*, *aabbb*, *aaabbb*.

1.2 (5 Punkte) Zeigen Sie, dass alle Zeichenketten, die gleich viele *a*s wie *b*s enthalten, auf den selben Wert abgebildet werden.

1.3 (5 Punkte) Schreiben Sie eine Funktion *hash*, die für eine gegebene Zeichenkette x den Hashwert $h(x)$ gemäß der obigen Hashfunktion berechnet. Wenn x andere Zeichen außer a und b enthält, soll -1 zurückgegeben werden. Sie können Ihre Funktion in Java oder in C++ schreiben.

1.4 (5 Punkte) Betrachten Sie jetzt die beiden Hashfunktionen $h_1(x) = (3 \cdot n_a + n_b) \bmod 4$ und $h_2(x) = (n_a + 3 \cdot n_b) \bmod 4$. Zeigen Sie, dass die Klasse $H = \{h_1, h_2\}$ nicht 1-universell ist.

Aufgabe 2 (Prioritätswarteschlangen und Sortieren, 20 Punkte)

2.1 (5 Punkte) Fügen Sie nacheinander die sieben Elemente $7, 6, 5, 4, 3, 2, 1$ in eine zu Beginn leere Prioritätswarteschlange (PW) ein, die als binärer Heap implementiert ist. Es soll dabei die gewöhnliche Ordnung $<$ auf den natürlichen Zahlen verwendet werden (d.h. das kleinste Element steht oben). Zeichnen Sie den Zustand der PW (als Baum) nach jeder der sieben Einfügungen.

2.2 (5 Punkte) Zeichen Sie den Zustand einer solchen zu Beginn leeren Prioritätswarteschlange nach dem Einfügen von sieben Zahlen $x_1, \ldots, x_7$ für die gilt, dass $x_1 < x_2 < \ldots < x_7$. Es reicht, wenn Sie den Zustand am Ende der sieben Einfügungen zeichnen.

2.3 (5 Punkte) Schreiben Sie eine Funktion *pqSort*, die ein gegebenes Feld von ganzen Zahlen mit Hilfe einer Prioritätswarteschlange, die als binärer Heap implementiert ist, in $O(n \log n)$ Zeit sortiert und das sortierte Feld zurückgibt. Sie können dabei annehmen, dass es einen Datentyp *PriorityQueue* mit den üblichen Methoden gibt. Wenn Sie eine solche Methode benutzen, beschreiben Sie kurz, was diese Methode macht! Sie können Ihre Funktion in Java oder in C++ schreiben.

2.4 (5 Punkte) Wie ist die Laufzeit Ihrer Methode (als Θ von einer Funktion von n) wenn das Eingabefeld bereits sortiert ist? Mit Begründung!

Aufgabe 3 (Dijkstra und Durchmesser, 20 Punkte)

Der Wagenrad-Graph mit $n+1$ Knoten ist wie folgt als *ungerichteter* Graph definiert. Die „Felge“ besteht aus n Knoten mit den Nummern $1, \ldots, n$ sowie den n Kanten $\{1,2\}, \{2,3\}, \ldots, \{n-1, n\}, \{n, 1\}$. Die „Nabe“ besteht aus einem Knoten $n+1$. Die „Speichen“ bestehen aus n Kanten $\{1, n+1\}, \{2, n+1\}, ..., \{n, n+1\}$. Die n Kanten auf der Felge sollen alle Kosten 3 haben, die n Kanten der Speichen alle Kosten 1.

3.1 (5 Punkte) Zeichnen Sie den Wagenrad-Graphen für $n = 5$ (mit Kantenkosten) und führen Sie Dijkstras Algorithmus ausgehend vom Knoten 1 aus. Es soll so lange iteriert werden, bis alle Knoten gelöst (settled) sind. Aus Ihrer Zeichnung sollte ersichtlich sein, welcher Knoten in welcher Iteration von Dijkstras Algorithmus gelöst wird, sowie alle vom Algorithmus berechneten Zwischendistanzen (*dist*-Werte) auf dem Weg zur optimalen Lösung. Verwenden Sie der Lesbarkeit halber bitte mindestens zwei Farben!

3.2 (5 Punkte) Betrachten Sie jetzt einen allgemeinen Wagenrad-Graphen für beliebiges n. Wie groß sind die Kosten des kürzesten Weges von 1 nach i, für alle $i = 1, \ldots, n+1$? Sie müssen für diese Aufgabe nichts mehr zeichnen. Die Begründung soll in der folgenden Teilaufgabe 3.3 geliefert werden.

3.3 (5 Punkte) Beweisen Sie Ihre Behauptung aus 3.2, indem Sie angeben, in welcher Iteration von Dijkstras Algorithmus welcher Knoten welchen *dist*-Wert hat und warum.

3.4 (5 Punkte) Wie groß ist der Durchmesser des Wagenrad-Graphen für ein gegebenes n? Mit Begründung!

Aufgabe 4 (O-Notation + Induktionsbeweis + Laufzeitbestimmung, 20 Punkte)

4.1 (5 Punkte) Zeigen Sie, dass für alle $k \in \mathbb{N}$ gilt, dass $(\log n)^k = O((\log n)^{k+1})$. Benutzen Sie dabei die explizite Definition von O mittels C und n_0 (d.h. ohne Grenzwertbetrachtungen).

4.2 (5 Punkte) Zeigen Sie, dass die Aussage aus 4.1 nicht gilt, wenn man O durch Θ ersetzt. Für diese Aufgabe ist Ihnen freigestellt, ob Sie sie über die explizite Definition mittels C und n_0 oder über geeignete Grenzwertbetrachtungen lösen.

4.3 (5 Punkte) Beweisen Sie über vollständige Induktion, dass für alle natürlichen Zahlen n gilt, dass $\sum_{i=1}^{n}(n-i) = 1/2 \cdot n \cdot (n-1)$.

4.4 (5 Punkte) Erklären Sie kurz, was folgende Funktion mit ihrer Eingabe macht. Geben Sie insbesondere ein Beispiel, bei dem das Eingabefeld verändert wird (es reicht der Zustand vor und nach dem Aufruf der Funktion). Bestimmen Sie außerdem die Laufzeit der Funktion als Θ von einer Funktion von n, wobei n die Größe des gegebenen Feldes *array* ist.

```
void doSomething(int[] array) {
  for (int i = 0; i < array.length; i++) {
    for (int j = 0; j < array.length - i - 1; j++) {
      if (array[j] > array[j+1]) {
        int tmp = array[j];
        array[j] = array[j+1];
        array[j+1] = tmp;
      }
    }
  }
}
```