Lehrstuhl Algorithmen
und Komplexität
Prof. Dr. Fabian Kuhn
Sebastian Daum, Oleksii Saukh

Algorithmen und Datenstrukturen
(Informatik II) SS 2014

http://ac.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Übungsblatt 4

Abgabe: Dienstag, 3. Juni, 16:00 Uhr

Laden Sie Ihre Lösungen für theoretische *Übungen im Format "loesung-xx-y.pdf" hoch, wobei "xx" die Nummer des aktuellen Übungsblattes ist (bei Bedarf mit führender Null) und "y" die Aufgabennummer, d.h., erzeugen Sie* pro (theoretische) Aufgabe *eine* .pdf*-Datei. Scans, Bilder von Scans etc. bitte zuerst ins pdf-Format umwandeln. Bei Programmieraufgaben müssen Sie – wann immer es relevant/durchführbar ist – Unit Tests sowie einen Style-check durchführen. Laden Sie außerdem eine Datei "erfahrungen.txt" hoch, in welcher Sie Ihre Erfahrungen und Meinungen zu dem Übungsblatt teilen.*

Aufgabe 1 (5 Punkte)

Betrachten Sie die Hashfunktion $h(x) = x \bmod (2^p - 1)$, wobei x als Zahl zur Basis 2^p dargestellt ist (bit-Blöcke der Länge p). D.h., $x = x_{k-1}x_{k-2}\dots x_1x_0$ und die k Ziffern x_i sind aus $\{0, 1, \dots, 2^p - 1\}$. Zeigen Sie, dass wenn y eine Zahl ist, welche man durch Permutation der Ziffern in x erhält, dass dann $h(x) = h(y)$. Geben Sie ein Beispiel an, bei welchem ein solches Verhalten einer Hashfunktion unerwünscht wäre.

Hinweis: Schlagen Sie bei Bedarf Äquivalenzregeln und Umformungen für modulo-Arithmetik nach.

Aufgabe 2 (5 Punkte) Gegeben sind zwei Integer-Arrays A und B der Länge n, und wir wissen, dass $A[0] \leq B[0]$ sowie $A[n-1] \geq B[n-1]$. Geben Sie einen Algorithmus (in Pseudocode) an, welcher möglichst schnell einen Index i findet, für welchen gilt: $A[i] \leq B[i]$ und $A[i+1] \geq B[i+1]$. Beweisen Sie mit Hilfe einer Schleifeninvariante und mit Hilfe von Vor- und Nachbedingungen, dass Ihr Algorithmus das Richtige tut.

Aufgabe 3 (5 Punkte)

Für einen Online Shop, welcher n Produkte führt, sollen Sie eine effiziente Datenstruktur zur Verwaltung des Lagerzustands entwerfen. Alle Aufträge sollen in exakt der Reihenfolge abgearbeitet werden, in welcher sie eintreffen. Ein Auftrag hierbei ist ein Tupel *(Produkt, Anzahl, Auftragsdetails)*, d.h., fuer jedes Produkt in einer Bestellung wird ein eigener Auftrag erstellt. Zudem soll Ihre Datenstruktur folgendes unterstützen:

- *Insert(Auftrag)*: Man kann einen neuen Auftrag in $O(1)$ in die Datenstruktur einfügen.
- *ExtractNext()*: Ein Auftrag wird in $O(1)$ aus der Datenstruktur entnommen.
- *Query(Product)*: Gibt in $O(1)$ die Gesamtbestellmenge des betreffenden Produkts in allen Aufträgen zurück. D.h., gibt es zum Produkt P die t Aufträge $\{A_1, \dots, A_t\} = \{(P, k_1, D_1), \dots, (P, k_t, D_t)\}$, dann soll *Query(P)* den Wert $k^P_{total} = \sum_{i=1}^{t} k_i$ zurückgeben.

Des weiteren soll die Datenstruktur speichereffizient sein, im folgenden Sinne. Man garantiert Ihnen, dass nie mehr als $m \ll n$ Aufträge gleichzeitig in der Warteschlange sind, und daher will man auch, dass obige Datenstruktur nur $O(m)$ Speicherplatz benötigt.

Beschreiben Sie, wie Sie diese Datenstruktur implementieren. Verwenden Sie *zusätzlich* Pseudocode, um Ihre Lösung zu verdeutlichen.

Aufgabe 4 (5 Punkte)

Manchmal ist es von Interesse, zu wissen, welche Wörter in einem Text am häufigsten vorkommen. Im Buch "Harry Potter and the Chamber of Secrets" zum Beispiel kommt das Wort *Harry* etwa 1500-mal vor.

Schreiben Sie ein Programm, welches einen vorgegebenen Text einliest und die k häufigsten Wörter im Text wiedergibt.[1] k ist hierbei ein Eingabeparameter. Die Laufzeit soll in Abhängigkeit von k der Anzahl Wörter des zu analysierenden Textes sein. Bei einem Text mit n Wörtern soll die Laufzeit Ihres Programmes $O(kn)$ sein. Ihr Programm soll davon ausgehen, dass der Eingabetext vorverarbeitet wurde: in jeder Zeile der Eingabe steht genau ein Wort; das sollte die Aufgabe etwas erleichtern.

Im *public* SVN Verzeichnis liegen zwei Dateien, *test.txt* und *input.txt*, beide im oben beschriebenen Format. Erstere können Sie zum Testen verwenden, die zweite Datei müssen Sie unter Verwendung von $k = 10$ analysieren und Ihr Ergebnis sollen Sie in einer Datei *output.txt* abspeichern. Zu dieser Aufgabe stellen wir auch einige vorgefertigte Strukturdateien zur Verfügung, die Ihnen eventuell helfen können.

Trivia: Mit Datenstrukturen, die Sie erst noch kennenlernen müssen, lässt sich die Laufzeit auf $O(n \log k)$ reduzieren. Es gibt allerdings auch eine etwas komplexere Methode, welche *Binary Search* und *Hashing* kombiniert, um die selbe Laufzeit zu erhalten. Wenn Sie sich herausgefordert fühlen, versuchen Sie doch, diese Lösung zu finden. ;)

[1] Einen Gleichstand können Sie beliebig auflösen, d.h., wenn z.B. $k = 5$ und die Häufigkeiten der Wörter sind (absteigend sortiert) $24, 18, 18, 9, 9, 9, 9, 8, 8, \ldots$, dann sollen die Wörter mit den Häufigkeiten $24, 18, 18$ sowie zwei beliebige Wörter mit Häufigkeit 9 zurückgegeben werden.