2. Übungsblatt zu Grundzüge von Algorithmen und Datenstrukturen, WS 12/13

Prof. Dr. Markus Bläser, Radu Curticapean, Christian Engels
http://www-cc.cs.uni-sb.de/course/38/

Abgabe: Donnerstag, 8. November 2012, 12:00 Uhr

Aufgabe 2.1 Wir bezeichnen ein Sortierverfahren S stabil, wenn auf Eingabe $a_1, \ldots, a_n$ gilt: Ist $a_i = a_j$ mit $i < j$, so steht a_i im durch S sortierten Array links von a_j. In anderen Worten: Die Reihenfolge von Elementen mit identischem Schlüssel bleibt erhalten.[1]

a) Begründen Sie, welche der Sortierverfahren aus der Vorlesung stabil sind und welche nicht. Geben Sie Gegenbeispiele für die nicht-stabilen Sortierverfahren an.

b) Gegeben sei ein beliebiges vergleichsbasiertes Sortierverfahren S. Modifizieren Sie S so, dass es ein stabiles Sortierverfahren wird, indem Sie die Eingabe für S modifizieren und die Vergleichsfunktion $\leq$ überschreiben. Die asymptotische Laufzeit von S soll sich dabei nicht verändern.

   Sie dürfen für diese Aufgabe Paare von Zahlen in den Speicherzellen der RAM speichern, etwa durch $x := (a, b)$, und mittels Funktionen π_1, π_2 auf die Elemente eines Paares zugreifen. Sie sollten auch eine geeignete Vergleichsfunktion $\leq$ auf Paaren definieren, die S anstelle des üblichen $\leq$ auf Zahlen aufruft.

Aufgabe 2.2 Wir betrachten das folgende Sortierverfahren:

```
procedure SORT(A, i, j)
    if i ≥ j then
        return
    m ← ⌊(i + j)/2⌋
    Sort(A, i, m)
    Sort(A, m + 1, j)
    if A[j] < A[m] then
        swap(A[j], A[m])
    Sort(A, i, j − 1)
```

Die Prozedur Sort erhält ein Array A und zwei Indizes i, j als Eingabe und soll A sortieren. Auf einem Array der Größe n wird initial $\text{Sort}(A, 1, n)$ aufgerufen.

a) Beweisen Sie, dass dieser Algorithmus ein Sortierverfahren ist.

b) Was können Sie über die Laufzeit sagen? Geben Sie die Rekurrenzgleichung für die Laufzeit von Sort an.

[1] Beispiel: Wird mit einem stabilen S eine alphabetisch sortierte Liste von Waren nach Preis sortiert, so ist das Ergebnis nach Preis sortiert und Waren mit identischem Preis sind alphabetisch sortiert.

Aufgabe 2.3 Wir betrachten ein Hochhaus mit K Stockwerken und N Aufzügen. In diesem Hochhaus verkehrt Aufzug i, mit $1 \leq i \leq N$, zwischen Stockwerk u_i und o_i, wobei $1 \leq u_i < o_i \leq K$. Aufzug i kann an allen Stockwerken im Intervall $[u_i, o_i]$ halten.

Es gibt keine Treppen. Der Architekt des Hochhauses möchte daher sicherstellen, dass jedes Stockwerk von jedem anderen Stockwerk aus durch das Benutzen von (ggf. mehreren) Aufzügen erreicht werden kann.

Wir suchen Algorithmen, die auf Eingabe $K, (u_1, o_1), \ldots, (u_N, o_N)$ entscheiden, ob diese Bedingung erfüllt ist. Ist die Bedingung nicht erfüllt, sollen die Algorithmen eine Liste von Zahlen $(u'_1, o'_1) \ldots (u'_t, o'_t)$ ausgeben, so dass das Hinzufügen von Aufzügen mit diesen Stockwerkgrenzen die Bedingung erfüllt und dabei $\sum_{i=1}^{t} |o'_i - u'_i|$ minimal ist.

a) Entwickeln Sie einen derartigen Algorithmus. Seine Laufzeit soll durch $O(N \log N)$ beschränkt sein. Stockwerkgrenzen dürfen hierbei in konstanter Zeit miteinander verglichen werden.

b) Entwickeln Sie einen weiteren derartigen Algorithmus. Seine Laufzeit soll jedoch durch $O(N + K)$ beschränkt sein. (Dieser Algorithmus ist also empfehlenswert für kleines K. Dies trifft für reale Gebäude sicherlich zu.)

Aufgabe 2.4 In dieser Aufgabe betrachten wir Datenstrom-Algorithmen. Ein derartiger Algorithmus erhält ein Array $A = [a_1, \ldots, a_n]$ als Eingabe, darf jedes Element von A jedoch nur einmal lesen und muss diese Elemente in der Reihenfolge $a_1, \ldots, a_n$ lesen.[2]

Wir betrachten Eingaben der Art $[a_1, \ldots, a_{n-1}]$ mit paarweise verschiedenen a_i und $\{a_1, \ldots, a_{n-1}\} \subseteq \{1, \ldots, n\}$. In anderen Worten: Wir erhalten als Eingabe alle Elemente der Menge $\{1, \ldots, n\}$, mit Ausnahme eines $j \in \mathbb{N}$, in beliebiger Reihenfolge.

Geben Sie einen Datenstrom-Algorithmus an, der auf solchen Eingaben j bestimmt. Ihr Algorithmus darf nur $O(1)$ Speicherzellen nutzen.

[2] Wurde eine Zahl gelesen, so kann sie also nicht erneut gelesen werden, sofern sie nicht in einer zusätzlichen Speicherzelle gespeichert wurde.