Prof. G. Zachmann
David Mainzer

TU Clausthal
Institut für Informatik
11. Juni 2010

Sommersemester 2010

Übungen zu Informatik II - Blatt 9

Abgabe in der Übung am 15. 06 / 16. 06. 2010

Bitte beachten Sie, dass die Programmieraufgaben von Ihnen in der Übung vorgeführt und erklärt werden müssen. Zusätzlich senden Sie die Lösung, unter Angabe ihres Namens, an **dm@tu-clausthal.de**.

Aufgabe 1 (Divide-and-Conquer-Algorithmus, *4+2+4+4 Punkte*)

Betrachten Sie folgende Aufgabe: gegeben ist ein unsortiertes Array A; gesucht ist das k-größte Element aus A ($0 \leq k < n$, wobei n = Anzahl Elemente in A). Wir nehmen im Folgenden an, dass alle Werte in A nur einmal vorkommen. Das größte Element ist definiert als das $(n-1)$-größte aus A, und das 0-größte Element ist definiert als das kleinste aus A.

Der naive Algorithmus würde zunächst A sortieren und dann das Element $A[k]$ liefern. Dieser hätte die Laufzeit von $O\big(n \log n\big)$.

Entwickeln Sie nun einen *randomisierten* Divide-and-Conquer-Algorithmus zur Lösung dieser Aufgabe.

Hinweise: gehen Sie ähnlich wie Quicksort vor. Dort wurde ein zufälliges Pivot-Element gewählt, mit dessen Hilfe dann das Array partitioniert wurde. Die Partitionierung kann trivialerweise auch den Index des Pivot-Elementes *nach* der Partitionierung liefern. In der vorliegenden Aufgabe muß man nach der Partitionierung, im Gegensatz zu Quicksort, nur noch eines der beiden Teil-Arrays verfolgen.

a) Beschreiben Sie Ihren Algorithmus als Pseudo-Code

b) Analysieren Sie Ihren Algorithmus. Sie dürfen zur Vereinfachung annehmen, dass die Partitionierung in jedem Fall zwei Teile liefert, von denen das größere maximal αn viele Elemente enthält, $0 < \alpha < 1$. Sie dürfen außerdem annehmen, dass n und α so sind, dass $\alpha^k n = 1$ für ein $k \in \mathbb{N}$.

c) Implementieren Sie Ihren Algorithmus in dem auf der Webpage bereitgestellten Framework `select.py`. Die Funktion `partition` ist schon vorhanden. Außerdem gibt es eine Funktion `random.randrange(a,b)`, die einen zufälligen Integer im Intervall $[a, b)$ liefert. (Wenn Sie nach dem Wort "debugging" im Framework suchen, finden Sie einige Hinweise, die selbiges erleichtern könnten.)

d) Betrachten Sie nun den deterministischen Algorithmus (von Blum, Floyd, Pratt, Rivest, und Tarjan), gegeben in folgendem Pseudo-Code:

```
1 select( A, k ):
2     n = len(A)
3     for i = 0 .. n/5-1:
4         B[i] = median5( A[5i : 5i+5] )
5
```

```
6       p = select( B, n/10 )
7       q = partition( A, B[p] )
8
9       if k < q:
10          return  select( A[0:q], k )
11      else if k > q:
12          return  select( A[q+1:n], k-q )
13      else
14          return q
```

Die Funktion `median5` liefert den Median von 5 Elementen (hier ist also der Median dasjenige Element von den Fünfen, so dass 2 davon kleiner sind und 2 größer). Die Funktion `partition` partitioniert, wie bei Quicksort, das Array so, dass das linke Teil-Array nur Elemente kleiner als das Pivot-Element enthält, das rechte nur Elemente größer als das Pivot-Element; zusätzlich liefert die Funktion den Index des Pivot-Elementes (nach der Partitionierung). (Zur Erinnerung: `A[i:j]` liefert ein Array bestehend aus den Elementen `A[i]`, ...`A[j-1]`.)

Bestimmen Sie die Rekursionsgleichung für den worst-case Aufwand $T(n) = \ldots$. Sie brauchen diese *nicht* aufzulösen. (Zu Ihrer Information: man kann zeigen, dass $T(n) \in O\big(n\big)$.)

Überlegen Sie sich dazu, wieviele Elemente in dem Array `A[0:q]` sein können im Fall $k < q$, und wieviele in `A[q+1:n]` im Fall $k > q$. Die Funktion `median5` kann man offensichtlich in $O(1)$ ausrechnen.

Aufgabe 2 (Dynamisches Programmieren, *6 Punkte*)

Gegeben ist ein Text mit n Wörtern (Strings) w_i jeweils der Länge $l_i := len(w_i)$. Der Text soll auf ein Blatt Papier so ausgedruckt werden, daß am rechten Rand möglichst wenig Leerzeichen übrig bleiben. In eine Zeile des Blattes passen M Zeichen. Zwischen 2 Wörter innerhalb einer Zeile steht *immer* ein Leerzeichen. In der m-ten Zeile, in der die Wörter w_i bis w_j stehen, ist die Anzahl der am rechten Rand verbleibenden Leerzeichen $\alpha_m := M - \Big((j-i) + \sum_{k=i}^{j} l_k\Big)$, wenn $i \leq j$. Die Anzahl der Leerzeichen in der letzten Zeile werden nicht berücksichtigt. Zu minimieren ist $\sum_{i=1}^{m-1} \alpha_m$, d.h. wir wollen einen möglichst glatten rechten Rand erreichen.

Zeigen Sie, daß die 4 Grundprinzipien der dynamischen Programmierung auf dieses Problem anwendbar sind. Geben Sie einen Dyn.-Prog.-Algorithmus an, welcher das obige Problem löst. Analysieren Sie Laufzeit und Speicherverbrauch Ihres Algorithmus.