Übungen zu Algorithmen
WSI für Informatik
Kaufmann/Kuckuk/Förster

Wintersemester 2021/22
Universität Tübingen
17. Dezember 2021

Aufgabensammlung Teil 10

Thema: Sortieren

Aufgabe 1: Bucketsort

Sortieren Sie die folgenden Wörter über dem Alphabet $\Sigma = \{e, i, m, n, s\}$ mit Bucketsort.

miene, einem, miese, seine, meine, nenne, messe, essen,
niese, seien, meins, innen, eisen, mimen, meise, eines

Aufgabe 2: Quicksort

Dies ist eine alte Klausuraufgabe!

Quicksort hat eine erwartete Laufzeit von $\mathcal{O}(n \log n)$. Die Laufzeit ist im Worst-Case jedoch schlechter, da man nicht zwingend ein gutes Pivot-Element wählt. Um dies zu verdeutlichen, nehmen wir in dieser Aufgabe an, dass stets der Median als Pivot-Element in konstanter Zeit gewählt werden kann. Dadurch wird das zu sortierende Array A beim Vergleich mit dem Pivot stets in zwei gleich große Teilarrays A_1 und A_2, die dann rekursiv sortiert werden, zerlegt.

a) Beweisen Sie, dass Quicksort unter der Annahme, dass der Median in konstanter Zeit bestimmt werden kann, ein Array der Größe n in Zeit $\mathcal{O}(n \log n)$ sortiert.

Wenn man das Pivot stets optimal bestimmt, lässt sich Quicksort auch sehr gut parallelisieren. Grafikkarten haben heute sehr viele Cores. Vereinfacht nehmen wir an, dass es so viele Cores gibt, dass alle Teilprobleme derselben Größe parallel berechnet werden können, d.h. alle diese Berechnungen laufen in derselben Zeit ab. Dies ermöglicht eine starke Verbesserung der Gesamtlaufzeit.

b) Beweisen Sie, dass parallelisiertes Quicksort unter der Annahme, dass der Median in konstanter Zeit bestimmt werden kann, ein Array der Größe n in Zeit $\mathcal{O}(n)$ sortiert.

Hinweis: Sie dürfen jeweils annehmen, dass die Größe n des Arrays A eine Zweierpotenz ist.

Aufgabe 3: Sortieren von Intervallen

Betrachten Sie das folgende Sortierproblem: Wir kennen die Zahlen nicht genau, sondern haben stattdessen n Intervalle $I_j = [a_j, b_j]$ mit $a_j \leq b_j$ (wobei $a_j, b_j \in \mathbb{R}$) gegeben. Gesucht ist eine Sortierung $(I_{j_1}, I_{j_2}, \ldots, I_{j_n})$ der Intervalle mit der Eigenschaft: Für jedes $k = 1, \ldots, n$ gibt es ein $c_k \in I_{j_k}$, so dass $c_1 \leq c_2 \leq \cdots \leq c_n$.

a) Entwerfen Sie einen Algorithmus in Pseudocode, der das Problem in einer erwarteten Laufzeit von $\mathcal{O}(n \log n)$ löst. Ihr Algorithmus soll so aufgebaut sein, dass eine Laufzeit von $\mathcal{O}(n)$ erreicht wird, wenn sich alle Intervalle schneiden (das heißt, wenn der Schnitt $\bigcap_{j=1}^{n} I_j$ nicht leer ist). Verwenden Sie in Ihrem Pseudocode sinnvolle Variablennamen und erklären Sie die Funktionsweise Ihres Pseudocodes. Begründen Sie die Korrektheit Ihres Algorithmus.

   Tipp: Beachten Sie die folgenden Hinweise:

   - Für die linken Endpunkte der Intervalle (die a_i's) sollte Ihr Algorithmus eine ähnliche Struktur wie Quicksort haben.
   - Nutzen Sie zusätzlich „überlappende Intervalle", um Ihre Laufzeit zu verbessern: Merken Sie sich als Pivotelement eine Untergrenze u und eine Obergrenze o; diese passen Sie an, falls $[u, o] \cap [a_j, b_j] \neq \varnothing$ (wie und warum?).

b) Argumentieren Sie, dass Ihr Algorithmus die geforderte erwartete Laufzeit von $\mathcal{O}(n \log n)$ hat. *Hinweis:* Laufzeiten, die Sie aus der Vorlesung kennen, müssen nicht bewiesen werden.

c) Argumentieren Sie, dass Ihr Algorithmus eine Laufzeit von $\mathcal{O}(n)$ hat, wenn sich alle Intervalle schneiden.

Lösungen:

1. *Iteration* 1: miene, miese, seine, meine, nenne, messe, niese, meise, einem, essen, seien, innen, eisen, mimen, meins, eines

   Iteration 2: einem, essen, seien, innen, eisen, mimen, eines, miene, seine, meine, nenne, meins, miese, messe, niese, meise

   Iteration 3: miene, miese, niese, seien, seine, meine, meins, meise, mimen, einem, innen, eines, nenne, essen, eisen, messe

   Iteration 4: seien, seine, meine, meins, meise, nenne, messe, miene, miese, niese, mimen, einem, eines, eisen, innen, essen

   Iteration 5: einem, eines, eisen, essen, innen, meine, meins, meise, messe, miene, miese, mimen, nenne, niese, seien, seine

2. a) Kann der Median immer in konstanter Zeit bestimmt werden, zerteilt man das Problem in zwei Teilprobleme halber Größe (in Zeit $O(n)$). Dann gilt, dass $T(n) = 2T(n/2) + \mathcal{O}(n)$.

   b) Da man alle Teilprobleme derselben Größe parallel berechnen kann, erhält man $T(n) = T(n/2) + \mathcal{O}(n)$.

3. a) Verfahre wie bei Quicksort. Dabei wird ein Pivot-Intervall mit dem ersten Intervall initialisiert und die anderen Arrays entsprechend dieses Pivots sortiert:

      (i) Arrays, die komplett kleiner als das Pivot sind

      (ii) Arrays, die komplett größer als das Pivot sind

      (iii) Arrays, die einen Schnitt mit dem Pivot haben

      Im letzten Fall (iii) wird zudem das Pivot-Intervall geupdatet (Schnitt des aktuell betrachteten Arrays + vorheriges Pivot-Intervall). Die Arrays, die unter Fall (i) bzw. (ii) fallen, werden rekursiv sortiert.

   b) Nutze dieselbe Argumentation wie bei Quicksort.

   c) Hier fallen alle Arrays zu Beginn unter Fall (iii) und es wird kein rekursiver Aufruf nötig.