Technische Universität
Kaiserslautern

Fachbereich Informatik
AG Algorithmen und Komplexität

Ausgabedatum: 10.01.2013
Version: 2013-01-14 16:19

12. Übungsblatt zur Vorlesung Entwurf und Analyse von Algorithmen, WS 12/13

Abgabe: *Bis Donnerstag, 17.01.2013, 12:00 Uhr, Abgabenkasten im Treppenhaus 48-6.*

Wir wünschen ein frohes und erfolgreiches Jahr 2013!

74. Aufgabe

Track ε [2]
Track AI 2

Wir nehmen an, ein Tennis-Turnier wird unter 2^n Spielern derart ausgetragen, dass in jeder Runde der unterlegende Spieler das Turnier verlässt, sich die Anzahl der verbliebenen Spieler also pro Runde jeweils halbiert. Dabei werden die Gegner der ersten Spiele zufällig ausgelost, anschließend gibt es aber eine von vorne herein festgelegte Strategie, wie die Sieger der einzelnen Spiele aufeinandertreffen.

Der Spieler, der das Spiel zwischen den letzten beiden verbliebenen Teilnehmern gewinnt, ist Sieger des Turniers, der unterlegene dieser beiden Spieler erhält den zweiten Preis. Wir nehmen an, es existierte eine transitive Rangordnung unter den Teilnehmern, was bedeutet, dass wenn Spieler A gegen Spieler B und Spieler B gegen Spieler C gewinnt, wir sicher sein können, dass auch A gegen C gewänne, also der bessere Spieler ist.

Mit welcher Wahrscheinlichkeit erhält in diesem Fall tatsächlich der zweitbeste Spieler unter den Turnierteilnehmern den zweiten Preis? Beweisen Sie Ihre Behauptung.

Hinweis: Für die Beantwortung dieser Frage ist es hilfreich, sich das Turnier als einen perfekten Binärbaum vorzustellen.

75. Aufgabe

Track ε [2]
Track AI [2]

Betrachten Sie ein zweidimensionales Feld $A = (a_{i,j})_{i,j=1,\ldots,n}$ mit n Spalten und n Zeilen. Nehmen Sie an, dass jedes Element $a_{i,j}$ mit einer ganzen Zahl besetzt wurde. Wir gehen nun folgendermaßen vor:

- Wir sortieren jede Zeile aufsteigend.
- Danach sortieren wir jede Spalte aufsteigend.

Zeigen oder widerlegen Sie, dass nach Abschluss des zweiten Sortierens (also nach dem Sortieren der Spalten) die Zeilen immer noch sortiert sind.

76. Aufgabe

Track ε	[2]
Track AI	[2]

Ein k-Kellerautomat ist ein Rechner, der auf seinen Speicher nur mittels der Stack-Operationen zugreifen kann. Er besitzt somit keinen wahlfreien Zugriff auf den Speicher, sondern nur die Stack-Operationen `PUSH`, `TOP`, `POP` und `EMPTY`. (Auf die k Stacks kann aber unabhängig voneinander zugegriffen werden.)

Ferner besitzt ein k-Kellerautomat noch eine konstante Anzahl von Arbeitsregistern, mit deren Hilfe er beispielsweise Werte vergleichen oder einen Zähler einer `FOR`-Schleife implementieren kann; Sie dürfen annehmen, dass genügend Register für alle in ihrem Programm verwendeten Zähler, Flags etc. sowie für drei Datenelemente vorhanden sind.

Die Eingabe und die Ausgabe eines k-Kellerautomaten wird im ersten Keller des Kellerautomaten bereitgestellt.

Geben Sie ein Sortierverfahren für einen 3-Kellerautomaten an, das eine beliebige Eingabe mit $\mathcal{O}(n \log_2 n)$ Stackoperationen sortiert. Begründen Sie Ihre Behauptungen.

77. Aufgabe

Track ε	2
Track AI	2

Erweitern Sie Mergesort so, dass der Algorithmus während des Sortierens auch die Anzahl der beseitigten Inversionen berechnet und zurückgibt. Dabei soll die asymptotische Laufzeit nicht verschlechtert werden.

Begründen Sie die Korrektheit und Effizienz Ihrer Erweiterung.

78. Aufgabe

	a)	**b)**	**c)**	**d)**	**e)**
Track ε	1	1	1	[1]	[2]
Track AI	1	1	1	1	2

In der Praxis kommt es oft vor, dass mehrere unterschiedliche Einträge einer Wörterbuchdatenstruktur den gleichen Schlüssel haben. Ein Sortierverfahren, das die Reihenfolge der Einträge mit gleichem Schlüssel untereinander nicht ändert, heißt *stabil*. Dies ist zum Beispiel wünschenswert, wenn man sukzessive nach mehreren Schlüsseln sortieren möchte.

Untersuchen Sie die Sortieralgorithmen aus der Vorlesung auf Stabilität! Geben Sie jeweils eine präzise Begründung für Ihre Behauptung an.

Geben Sie für die nicht stabilen Verfahren außerdem an, ob und wie man sie mit kleinen Änderungen stabil machen kann.

a) Bubble-Sort

b) Shell-Sort

c) Quicksort

d) Mergesort

e) Angenommen, Sie müssen einen nicht stabilen Sortieralgorithmus für ganze Zahlen verwenden, brauchen aber Stabilität. Können Sie die Eingabe so transformieren, dass das sortierte Ergebnis stabil ist? Sie können dabei annehmen, dass Sie dem vorliegenden Algorithmus eine Vergleichsfunktion übergeben können.

   Entwerfen Sie einen entsprechenden Algorithmus, begründen Sie seine Korrektheit und analysieren Sie die Laufzeit.

Hinweis: Untersuchen Sie genau die im Buch gegebenen Implementierungen!

79. Aufgabe

	a)	b)	c)
Track ε	[2]	[2]	[1]
Track AI	[2]	[2]	[1]

In der Vorlesung haben wir Quicksort unter der Annahme analysiert, die Eingabe sei eine uniform zufällig gewählte Permutation. In der Praxis wird diese Annahme in den wenigsten Fällen erfüllt sein; wie sieht es mit anderen Eingabeverteilungen aus?

Wie verhält sich die erwartete Laufzeit von Quicksort in der Implementierung der Vorlesung auf den folgenden Eingaben? Analysieren Sie die jeweiligen Extremfälle exakt und diskutieren Sie, inwiefern etwaige negative Effekte auch für weniger extreme Fälle auftreten.

Schlagen Sie – sofern nötig – Verbesserungen an unserer Implementierung vor, die den beobachteten Effekten entgegenwirken.

a) Duplikate – Schlüssel können mehrfach vorkommen.

b) Vorsortierung – die Eingaben sind schon teilweise sortiert, haben also sehr wenig Inversionen.

c) Inverse Vorsortierung – die Eingaben sind teilweise vorsortiert, aber *falsch herum*, haben also viele Inversionen.

	a)	b)	c)	d)
Track ε	2	1	1	3
Track AI	[2]	1	1	[3]

80. Aufgabe

Betrachten Sie den folgenden Algorithmus:

```
procedure sort(A) {
  n = A.length
  B = new Array[0..n-1]

  for ( i = 0 to n-1 ) {
    B[i] = new List
  }

  for ( i = 0 to n-1 ) {
    B[floor(n * A[i])].append(A[i])
  }

  for ( i = 0 to n-1 ) {
    B[i] = insertionSort(B[i])
  }

  j = 0
  for ( i = 0 to n-1 ) {
    if ( B[i].length > 0 ) {
      cur = B[i].first
      while ( cur != NIL ) {
        A[j] = cur.value
        cur = cur.next
        j = j + 1
      }
    }
  }
}
```

Die Prozedur `floor` gibt hier die größte ganze Zahl zurück, die kleiner oder gleich dem Parameter ist. `insertionSort` arbeitet analog zu „Sortieren durch direktes Einfügen“ aus der Vorlesung, nur auf Listen statt in Arrays.

a) Zeigen Sie, dass `sort(A)` das Array `A` sortiert, wenn dieses Zahlen aus $[0, 1)$ enthält.

b) Geben Sie eine Best-Case-Eingabe für `sort` (allgemein in Länge des Arrays n) an und leiten Sie die asymptotische Best-Case-Laufzeit in n ab.

c) Geben Sie eine Worst-Case-Eingabe für `sort` (allgemein in n) an und leiten Sie die asymptotische Worst-Case-Laufzeit in n ab.

d) Nehmen Sie an, dass die Einträge von `A` uniform zufällig aus $[0, 1)$ gezogen wurden. Bestimmen Sie die Θ-Klasse (in n) der erwarteten Laufzeit von `sort(A)` und beweisen Sie Ihre Behauptung.

81. Aufgabe

Track ε 1
Track AI 1

Beweisen oder widerlegen Sie: Gibt es für mindestens ein Teilproblem mehr als eine optimale Lösung, so kann die dynamische Programmierung auch dann nicht mehr verwendet werden, wenn das Bellmansche Optimalitätskriterium erfüllt ist.

82. Aufgabe

Track ε 2
Track AI 2

In einem Turnier sollen n Teams gegeneinander antreten, wobei jedes Team gegen jedes andere spielen soll. n sei hierbei eine Zweierpotenz.

Geben Sie einen Divide and Conquer Algorithmus an, der einen entsprechenden Spielplan in Form einer $n \times (n-1)$ Tabelle aufstellt, wobei jede Zeile für ein Team und jede Spalte für eine Runde steht. Der Eintrag an der Position (i, j) bestimmt das Gegnerteam von Team i in Runde j.

Begründen Sie die Korrektheit Ihres Algorithmus.