Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

04.05.2009

3. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Montag, 11.05.2009, 10:00 Uhr

1 Theorie

1.1 Komplexität.

Die folgende Java-Methode `sum42` ermittelt alle Möglichkeiten, den Wert `42` als Summe dreier Elemente des Arrays `a` darzustellen. Bestimmen Sie die worst-case Laufzeit der Methode in Abhangigkeit von `n:= a.length` in O-Notation und begründen Sie Ihre Antwort (`println` hat konstanten Zeitaufwand).

```
static void sum42(int[] a) {
  // Es wird vorausgesetzt, dass a nur nichtnegative Zahlen enthaelt.
  for (int i = 0; i < a.length; ++i) {                          // Schleife 1
    if (a[i] <= 42) {                                           // Bedingte Anw. 1
      for (int j = i + 1; j < a.length; ++j) {                  // Schleife 2
        if (a[i] + a[j] <= 42) {                                // Bedingte Anw. 2
          for (int k = 0; k < a.length; k++) {                  // Schleife 3
            if (a[i] + a[j] + a[k] == 42) {                     // Bedingte Anw. 3
              System.out.println(42 + " = " + a[i] + " + "      // Ausgabe
                  + a[j] + " + " + a[k]);
            }
          }
        }
      }
    }
  }
}
```

Variante 1:

- Der Rumpf der innersten Schleife hat einen Aufwand von O(1).
- Diese Schleife wird O(n)-mal wiederholt. $\rightsquigarrow$ O(n)
- Im schlimmsten Fall gilt $a[i] + a[j] \leq 42$ für alle $0 \leq i, j < n$ und somit wird die innerste Schleife für jede der O(n) Iterationen der zweiten Schleife wiederholt. $\rightsquigarrow$ O(n^2)
- Ebenso gilt $a[i] \leq 42$ im schlimmsten Fall für alle O(n)-Iterationen der äußeren Schleife, so dass die zweite Schleife in jeder Runde ausgeführt wird. $\rightsquigarrow$ O(n^3)

Variante 2:

- Nach Voraussetzung hat die *Ausgabe* einen Aufwand von O(1).
- Entsprechend den „Rechenregeln“ für bedingte Anweisungen hat die *dritte* **`if`**-*Anweisung* somit einen Aufwand von O(1) + max(O(1)) = O(1).
- Damit hat die *dritte Schleife* nach den „Rechenregeln“ für **`for`**-Schleifen einen Aufwand von O(1) $\cdot$ n = O(n).
- Also hat die *zweite* **`if`**-*Anweisung* einen Aufwand von O(1) + max(O(n)) = O(n).
- Es folgt ein maximaler Aufwand von O(n) $\cdot$ $(n-1)$ = O(n^2) für die *zweite Schleife*.
- Somit hat die *erste* **`if`**-*Anweisung* einen Aufwand von O(1) + O(n^2) = O(n^2).
- Es ergibt sich insgesamt also ein maximaler Aufwand von O(n^2) $\cdot$ n = O(n^3) für die *erste Schleife*.

Variante 3: Bezeichne C_1 den Rumpf der äußeren Schleife, C_2 den Rumpf der ersten **`if`**-Anweisung, C_3 den Rumpf der zweiten Schleife, C_4 den Rumpf der zweiten **`if`**-Anweisung, C_5 den Rumpf der inneren Schleife und C_6 den Rumpf der inneren **`if`**-Anweisung. Schließlich bezeichne T_i die maximale Laufzeit von C_i für $i = 1, \ldots, 6$.

- Es gilt nach Folie 2-56: $T(n) = T_1(n) \cdot n$
- Es gilt nach Folie 2-59: $T_1(n) = \mathrm{O}(1) + \max(T_2(n))$
- Es gilt nach Folie 2-56: $T_2(n) = T_3(n) \cdot (n-1)$
- Es gilt nach Folie 2-59: $T_3(n) = \mathrm{O}(1) + \max(T_4(n))$
- Es gilt nach Folie 2-56: $T_4(n) = T_5(n) \cdot n$
- Es gilt nach Folie 2-59: $T_5(n) = \mathrm{O}(1) + \max(T_6(n))$
- Es gilt nach Voraussetzung: $T_6(n) = \mathrm{O}(1)$

Insgesamt ergibt sich:

$$\begin{aligned} T(n) &= (\mathrm{O}(1) + (\mathrm{O}(1) + (\mathrm{O}(1) + \mathrm{O}(1)) \cdot n) \cdot (n-1)) \cdot n \\ &= \mathrm{O}(n^3) \end{aligned}$$

(10 Punkte)

2 Programmierung

Sequenzielle Suche: Schreiben Sie eine Java-Klasse **LinearSearch** zur sequenziellen Suche, welche das Interface **DataSearch** von der Webseite zur Vorlesung implementiert. Das heißt, eine Methode „**doSearch**“ implementiert den in der Vorlesung vorgestellten Algorithmus zur sequenziellen Suche und liefert bei Eingabe eines Arrays **data** von ganzen Zahlen (**int**) und eines Schlüsselwerts **key** (**int**) den Wahrheitswert **true** zurück, falls **key** in **data** enthalten und **false** sonst. (5 Punkte)

Binäre Suche: Implementieren Sie den Algorithmus **BinarySearch** aus der Vorlesung in Java. Schreiben Sie hierzu eine Klasse **BinarySearch**, welche das Interface **DataSearch** von der Webseite zur Vorlesung implementiert. (5 Punkte)

Laufzeitvergleich: Schreiben Sie eine Java-Klasse **StopWatch** zum Laufzeitvergleich der binären und sequenziellen Suche. Dabei sollen die Klassen **LinearSearch** und **BinarySearch** zur Suche der Zahlen 3, 2147483145, 1074047991 und 42 in Feldern der Größen 10^4, 10^7 und 10^8 verwendet und die jeweiligen Laufzeiten in Millisekunden ausgegeben werden. Verwenden Sie als Datenquelle die Klasse **DataSource** von der Webseite zur Vorlesung. Mittels des Aufrufs `DataSource.getSortedData(100)` erhalten Sie beispielsweise ein Array mit 100 sortierten Zufallszahlen. Beachten Sie, dass Ihr Java-Programm viel Speicher benötigt und stellen Sie Ihrer JVM ausreichend zur Verfügung (Aufruf z.B. mittels „**java -Xmx512m StopWatch**“).

Die Ausgabe der **main**-Methode von **StopWatch** soll wie folgt aussehen (wobei $< X_i >$ jeweils für die Laufzeit der Suche nach dem Schlüssel *i* steht): (10 Punkte)

```
Laufzeitvergleich der linearen und binaeren Suche in Daten der Groessen n:
Sequenzielle Suche (n=10000):
    3: <X1> ms
    42: <X2> ms
    1074047991: <X3> ms
    2147483145: <X4> ms
Sequenzielle Suche (n=10000000):
    3: <X1> ms
    42: <X2> ms
    1074047991: <X3> ms
    2147483145: <X4> ms
Sequenzielle Suche (n=100000000):
    3: <X1> ms
    42: <X2> ms
    1074047991: <X3> ms
    2147483145: <X4> ms
Binaere Suche (n=10000):
    3: <X1> ms
    42: <X2> ms
    1074047991: <X3> ms
```

```
    2147483145: <X4> ms
Binaere Suche (n=10000000):
    3: <X1> ms
    42: <X2> ms
    1074047991: <X3> ms
    2147483145: <X4> ms
Binaere Suche (n=100000000):
    3: <X1> ms
    42: <X2> ms
    1074047991: <X3> ms
    2147483145: <X4> ms
```

Viel Spaß und viel Erfolg!