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

27.4.2009

2. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Montag, 04.05.2009, 10:00 Uhr

1 Theorie

1.1 Berechenbarkeit.

Entscheiden Sie, ob sich die im folgenden beschriebenen Aufgabenstellungen algorithmisch lösen lassen.

Funktionale Sortierung: Für die Programmierumgebung Eclipse soll ein Plugin geschrieben werden, welches alle lokal verfügbaren Java-Quelldateien durchsucht und nach Funktionalität sortiert in einer Baumstruktur auflistet. So sollen z.B. unter dem Knoten „*Sortieren*“ alle Quelldateien aufgelistet werden, in denen ein Sortierverfahren implementiert ist.

Programmverifikation: Ein weiteres Plugin für Eclipse soll in großen Projekten dabei helfen, fehlerhafte Programmteile zu identifizieren. Hierzu sollen alle Programmschleifen (`for`,`while`, etc.) untersucht werden, ob sie bei beliebigen Eingabeparametern gesichert terminieren.

Nicht terminierende Schleifen sollen farbig (rot) hervorgehoben werden.

Programmverifikation mit Restriktionen: Für einige sicherheitskritische Anwendungen mit fest definiertem Einsatzgebiet lassen sich in der Phase der Programmspezifikation alle mögliche Eingabeparameter genau festlegen und auf einen endlichen Bereich einschränken. Für solche Anwendungen sollen alle innerhalb eines Projekts programmierten Funktionen überprüft werden, ob sie auf den spezifizierten Parametern nach einer vorgegebenen Anzahl von Schritten die gewünschte Ausgabe liefern.

(6 Punkte)

1. Nein, denn mit diesem Plugin könnte entschieden werden, ob zwei gegebene Programme dieselbe Funktion berechnen (im Widerspruch zur Vorlesung, Folie 2-9).
2. Nein, denn hiermit wäre das Halteproblem gelöst (im Widerspruch zur Vorlesung, Folie 2-3 ff).
3. Ja, indem systematisch jede Funktion mit jedem möglichen Eingabeparameter (endlich viele!) ausgeführt und nach Überschreitung der vorgegebenen Anzahl von Schritten abgebrochen wird.

1.2 Komplexität.

Fingerübungen zur O-Notation: Gegeben sind folgende Funktionen $f_i\colon \mathbb{N} \to \mathbb{N}$, $i = 1, \ldots, 5$:

$$f_1(n) := \tfrac{2}{3}n \qquad f_2(n) := \tfrac{1}{3}n^3 + 10n^2 - n \qquad f_3(n) := n^3 \qquad f_4(n) := 10^{42} \cdot n^2 \qquad f_5(n) := a^n$$

Untersuchen Sie, ob $f_i \in \mathrm{O}(f_{i+1})$ für $i = 1, \ldots, 4$ gilt. Begründen Sie Ihre Behauptung. Dabei können Sie folgende Eigenschaften verwenden:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \implies f \in \mathrm{O}(g)$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \implies f \notin \mathrm{O}(g)$$

(8 Punkte)

Ternäre Suche: Überlegen Sie sich, wie der Algorithmus **BinarySearch** aus der Vorlesung abgewandelt werden muss, so dass in jeder Runde das Eingabefeld in drei gleichgroße Abschnitte aufgeteilt und die Suche im passenden Drittel fortgesetzt wird (falls der Schlüssel noch nicht gefunden ist und der Abschnitt mehr als ein Element enthält).

Diskutieren Sie die Laufzeit analog zur Vorlesung und entscheiden Sie, ob und gegebenenfalls wann dieser Algorithmus der ursprünglichen Variante **BinarySearch** vorzuziehen ist und wann nicht. (6 Punkte)

Matrix-Rechnung: Betrachten Sie folgenden Ausschnitt eines Java-Programms und beantworten Sie die anschließenden Fragen:

algorithm Matrix (U, e) $\to M$
Eingabe: $n \times n$-Matrix U über $\mathbb{Z}$, natürliche Zahl e

```
long V[][] = new long[n][n];
long W[][] = new long[n][n];
copyMatrix(V, U);

for( int h=0; h<e; h++ ) {
   for( int i=0; i<n; i++ ) {
      for( int j=0; j<n; j++) {
         W[i][j] = 0;
         for( int k=0; k<n; k++ )
            W[i][j] += V[i][k]*U[k][j];
      }
   };
   copyMatrix(V,W);
}
return V;
```

Dabei erzeugt **copyMatrix(Target, Source)** eine Kopie des Arrays **Source** in das Array **Target**.

1. Was berechnet der angegebene Algorithmus bei Eingabe einer Matrix U und natürlichen Zahl e?

2. Wieviele arithmetische Operationen (Additionen und Multiplikationen) braucht der Algorithmus zur Berechnung bei Eingabe einer $n \times n$-Matrix U und natürlichen Zahl e? Geben Sie Ihr Ergebnis als Funktion von n und e an.

(6 Punkte)

Fingerübungen: *i*) Es gilt $f_1 \in \mathrm{O}(f_2)$, da $\lim_{n\to\infty} \frac{f_1}{f_2} = 0$.

ii) Es gilt $f_2 \in \mathrm{O}(f_3)$, da $f_1(n) = \frac{1}{3}n^3 + 10n^2 - n \leq 11n^3$ für alle $n \geq 0$ gilt.

iii) Es gilt $f_3 \notin \mathrm{O}(f_4)$, da $\lim_{n\to\infty} \frac{f_3}{f_4} = \infty$.

iv) Hier ist leider ein Tippfehler passiert: Die Konstante a ist nicht weiter spezifizier wurden, deshalb hier eine entsprechende Fallunterscheidung (die Wahl einer konkreten Konstanten, z.B. 2 wäre auch ausreichend gewesen) wobei wir uns auf nicht negative Größen a beschränken, da wir hier nur an oberen Schranken für positive Werte (Speicherverbrauch, Laufzeit, etc.) interessiert sind.

Fall 1: $a \leq 1$. Dann gilt $f_4 \notin \mathrm{O}(f_5)$, da $\lim_{n\to\infty} \frac{f_4}{f_5} = \infty$.

Fall 2: $a > 1$. Dann gilt $f_4 \in \mathrm{O}(f_5)$, da:

$$\begin{aligned}
\lim_{n\to\infty} \frac{f_4}{f_5} &= \ln(a) \cdot 2 \cdot 10^{42} \cdot \lim_{n\to\infty} \frac{n}{e^{n\ln(a)}} && \text{L'Hospital} \\
&= \ln(a)^2 \cdot 2 \cdot 10^{42} \cdot \lim_{n\to\infty} \frac{1}{e^{n\ln(a)}} && \text{L'Hospital} \\
&= 0
\end{aligned}$$

Ternäre Suche:

- In jeder „Runde“ verkleinert sich die Größe des Suchfeldes auf ein Drittel der Ursprünglichen Größe. D.h. falls das gesuchte Element noch nicht gefunden ist, bleiben in der i-ten Runde $\sim n/3^i$. Damit durchläuft der Algorithmus maximal $\sim \log_3 n$ Runden.
- Der Aufwand in jeder Runde ist höher als bei der binären Suche (c-mal so „aufwendig“ für eine Konstante c).
- *Asymptotisch* verschwinden beide Unterschiede in der O-Notation: Nehmen wir die Laufzeit von der Binären Suche als $k \cdot \log_2 n$ an (k Operationen in jeder Runde). Dann erhalten wir (vereinfacht) für die Ternäre Suche $c \cdot k \cdot \log_3 n$. Wegen $\log_3 n = \frac{\log_2 n}{\log_2 3}$ erhalten wir mit $\ell := \frac{ck}{\log_2 3}$ die Laufzeit $d \cdot \log_2 n$. Da k und ℓ Konstanten sind, haben sowohl binäre, als auch ternäre Suche eine asymptotische Laufzeit von $\mathrm{O}(\log n)$.
- Für konkrete Werte können trotzdem der eine oder der andere Algorithmus effizienter sein (großes n vs. großes c).
- Für formale Korrektheitsbeweise sind möglichst einfache Algorithmen vorzuziehen (in diesem Fall die Binäre Suche).

Matrix-Rechnung: 1. Der Algorithmus berechnet U^{e+1}, wobei U^n für $n \in \mathbb{N}$ induktiv wie folgt definiert ist:

$$U^0 := I \qquad \text{die } \textit{Einheitsmatrix}$$
$$U^{n+1} := U^n \otimes U \qquad \text{das } \textit{Matrixprodukt} \text{ von } U^n \text{ mit } U$$

2.
- in jeder Iteration der innersten **for**-Schleife (**k**) wird genau eine Addition und genau eine Multiplikation ausgeführt.
  $\rightsquigarrow 2n$ Operationen
- mit jeder Iteration der sie umschließenden **for**-Schleife (**j**) wird die innerste **for**-Schleife (**k**) ausgeführt.
  $\rightsquigarrow n \cdot 2n$ Operationen
- mit jeder Iteration weiter umschließenden **for**-Schleife (**i**) wird die **for**-Schleife (**k**) ausgeführt.
  $\rightsquigarrow n \cdot n \cdot 2n$ Operationen
- in jeder Iteration der äußersten **for**-Schleife (**e**) wird **for**-Schleife (**i**) ausgeführt.
  $\rightsquigarrow e \cdot n \cdot n \cdot 2n$ Operationen

Insgesamt werden also $t(n, e) := 2en^3$ arithmetische Operationen ausgeführt.

2 Programmieren

Schreiben Sie ein Java-Programm zur Primfaktorzerlegung einer natürlichen Zahl. Das heißt, zu gegebener Zahl n sollen alle Primzahlen p bestimmt werden, welche n ohne Rest teilen.
Die Ausgabe Ihres Programms bei Eingabe der Zahl 2600 soll wie folgt aussehen (es gilt $2600 = 2^3 * 5^2 * 13$):

```
Bitte geben Sie eine natürliche Zahl n ein: 2600
Primteiler von 2600 sind:
   2
   5
   13
```

Begründen Sie kurz die Korrektheit ihres Programms und geben Sie *obere Schranken* für folgende Größen in Abhängigkeit von n an:

a) die Anzahl von Divisionen und Modulo-Operationen

b) die Anzahl von Multiplikationen

Zum Beispiel ist $t(n) \coloneqq 2n^2$ eine obere Schranke für $\frac{2}{3}n^2+\frac{4}{2}n$, da für alle natürlichen Zahlen n gilt:

$$\begin{aligned} 2n^2 &= n^2 + n^2 \\ &\geq \frac{2}{3}n^2 + \frac{4}{2}n^2 \\ &\geq \frac{2}{3}n^2 + \frac{4}{2}n \end{aligned}$$

(10 Punkte)

Viel Spaß und viel Erfolg!