Otto-von-Guericke-Universität Magdeburg
Institut für Mathematische Optimierung
Prof. Dr. Robert Weismantel
Dr. Matthias Köppe, Gebäude 18, Raum 311

Sommersemester 2003

3. Übungsblatt und 1. Programmieraufgabe

Computerorientierte Mathematik

`http://www.math.uni-magdeburg.de/~mkoeppe/lehre/coma-2003`

Abgabe der Übungsaufgaben: Donnerstag, 24. April, zu Beginn der Übung

Abgabe der 1. Programmieraufgabe: bis Mittwoch, 7. Mai, 18.00 Uhr

Hinweise: 1. Da die Universität über Ostern Strom sparen will, müssen wir die meisten Rechner abschalten. Deshalb wird für Sie nur `kombi10` verfügbar sein.

2. Wenn Sie die Programmieraufgaben abgeben wollen, müssen *beide Teilnehmer* einer Gruppe das Programm zeigen und erklären können.

3. Die Abgabe der Programmieraufgaben kann entweder in der Veranstaltung am Mittwochmorgen oder zu einem anderen Termin erfolgen, den Sie mit mir (Matthias Köppe) vereinbaren. Da die 90 Minuten der Mittwochsveranstaltung für die Kontrolle der Abgaben von fast 20 Arbeitsgruppen naturgemäß knapp sind, kann es passieren, daß wir es nicht schaffen, Ihre Abgabe zu kontrollieren; in diesem Fall sollten Sie sich um einen anderen Abgabetermin bemühen.

11. Aufgabe **10 Punkte**

Beweisen Sie die folgenden Aussagen bzw. ihre Entscheidungen:

(a) $\mathrm{O}(f(n)) + \mathrm{O}(f(n)) \subseteq \mathrm{O}(f(n))$.

(b) $\mathrm{O}(f_1(n)) \cdot \mathrm{O}(f_2(n)) \subseteq \mathrm{O}(f_1(n) \cdot f_2(n))$.

(c) Seien $f_1(n),\ f_2(n) \geq 0$ für $n \in \mathbf{N}$. Dann ist $\max\{f_1(n),\ f_2(n)\} \in \Theta(f_1(n) + f_2(n))$.

(d) Ist $2^{2+n} \in \mathrm{O}(2^n)$?

(e) Ist $2^{2n} \in \mathrm{O}(2^n)$?

(f) Sei $f(n) \in O(g(n)$, aber $g(n) \notin O(f(n))$. Ist dann $f(n) \in o(g(n))$?

12. Aufgabe **10 Punkte**

Wir betrachten die folgenden Algorithmen. Dabei sei $n \in \mathbf{N}$ eine natürliche Zahl mit der Eigenschaft, daß es eine natürliche Zahl $k \in \mathbf{N}$ gibt, so daß $n = 2^k$ ist.

Alg. 1: **Input:** $n \in \mathbf{N}$.
(1) **set** $P := 1$;
(2) **for** $i := 1$ **to** n **do**
set $P := 2 \cdot P$;
(3) **return** (P);

Alg. 2: **Input:** $n \in \mathbf{N}$.
(1) **set** $k := \log_2(n)$, $P := 2$;
(2) **for** $i := 1$ **to** k **do**
set $P := P \cdot P$;
(3) **return** (P);

Alg. 3: **Input:** $n \in \mathbf{N}$, $a_1 = 2$, $a_2 = 2$,..., $a_n = 2$.
(1) **set** $P := 1$;
(2) **for** $i := 1$ **to** n **do**
set $P := a_i \cdot P$;
(3) **return** (P);

Alg. 4: **Input:** $n \in \mathbf{N}$, $a_1 = 2$, $a_2 = 2$,..., $a_n = 2$.
(1) **set** $P := 0$;
(2) **for** $i_1 := 1$ **to** a_1 **do**
for $i_2 := 1$ **to** a_2 **do**
$\ddots$
for $i_{n-1} := 1$ **to** a_{n-1} **do**
for $i_n := 1$ **to** a_n **do**
set $P := P + 1$;
(3) **return** (P);

Bestimmen Sie für alle Algorithmen die Anzahl der Operationen (arithmetische Operationen, Vergleiche, Zuweisungen etc.) die das Verfahren durchführt, sowie die jeweilige Laufzeitfunktion. (**Hinweis:** Bei der Bestimmung der Anzahl der Operationen wird die Größe der Zahlen ignoriert. Zum Beispiel wird die Addition zweier Zahlen immer als eine Operation gezählt, unabhängig von den auftretenden Zahlen. Bei der Bestimmung der Laufzeitfunktion muß die Größe der Zahlen aber berücksichtigt werden. Bei der Zahl der Operationen kommt es vor allem auf die korrekte Größenordnung und nicht unbedingt auf jede einzelne Operationen an.)

Entscheiden Sie für jeden der Algorithmen, ob die jeweilige Anzahl der durchgeführten Iterationen und die jeweilige Laufzeitfunktion polynomiell oder exponentiell in der Inputgröße des Problems sind. Bestimmen Sie dazu insbesondere die Inputgröße für jeden der vier Algorithmen.

Was berechnen die jeweiligen Algorithmen?

1. Programmieraufgabe

Hier nun also die erste echte Programmieraufgabe. Das Ziel ist, kurz gesagt, den in der Vorlesung vorgestellten Depth-First-Search-Algorithmus zu implementieren.

Schreiben Sie dazu ein C-Programm, welches zuerst einen Graphen G, der in dem vorgestellten Graphen-Eingabe-Format vorliegt, einliest. Um die Funktionsweise Ihres Programmes zu testen, sind unter `/home/coma/lib/graphs/` einige Beispielgraphen zu finden. Die einfachsten und kleinsten Beispiele sind

`graph-1.gr`, `graph-2.gr`, `graph-3.gr`, `graph-4.gr` und `graph-5.gr`.

Bitte beachten Sie, daß einige Beispielgraphen recht groß sind (bis zu 70 000 Knoten, bis zu 850 000 Kanten). Ihr Programm soll auch für die großen Graphen funktionieren. Deshalb wäre es eine schlechte Idee, den Graphen als Adjazenzmatrix zu repräsentieren. (Ihre Programme dürfen auf unseren Rechnern nur höchstens 100 Megabyte Speicher verwenden.) Empfohlen wird vielmehr die Verwendung einer Adjazenzliste.

Auf der Basis dieser Adjazenzliste soll dann der DFS-Algorithmus der Vorlesung implementiert werden.

Als Ergebnis soll das Programm die folgenden Daten liefern:

(a) Die Anzahl der Komponenten, aus denen der Graph G besteht.

(b) Falls der Graph zusammenhängend ist, soll Ihr Programm dies feststellen und einen aufspannenden Baum ausgeben. Überlegen Sie sich dafür ein möglichst übersichtliches Format, in dem Sie den Baum ausgeben.

(c) Stellen Sie fest, ob der Graph einen Kreis enthält. Falls ja, dann geben Sie einen Kreis aus. Achten Sie darauf, daß der Kreis *keine* Diagonalen enthält, d. h. tatsächlich ein Kreis (und nicht nur eine geschlossene Kette) ist.