Dr. Jürg M. Stettbacher
Neugutstrasse 54
CH-8600 Dübendorf

Telefon: +41 43 299 57 23
Fax: +41 43 299 57 25
E-Mail: dsp@stettbacher.ch

Algorithmen

Lösungen zu den Übungsaufgaben

Version 2.11
2014-08-19

Zusammenfassung: Dieses Dokument enthält die vollständigen Lösungen zu den Übungsaufgaben im Skript Algorithmen - Konzept und Anwendungen in der Informatik.

1 Lösungen zu den Übungsaufgaben

Die Übungsaufgaben aus dem Skript *Algorithmen - Konzept und Anwendungen in der Informatik* werden hier beantwortet. Einige Lösungen gehen möglicherweise über die geforderten Antworten hinaus.

1.1 Sieb des Eratosthenes

Algorithmus in Worten:

1. Notiere die Zahlen von 2 bis N in einer Liste.
2. Wähle die Zahl 2 in der Liste.
3. Falls die gewählte Zahl gestrichen ist, gehe zu Punkt 5.
4. Die Zahl ist eine Primzahl. Streiche alle Vielfachen dieser Zahl in der Liste.
5. Falls die Liste noch Zahlen enthält, wähle die nächste Zahl in der Liste. Sonst gehe zu Punkt 7.
6. Gehe zu Punkt 3.
7. Ende. Die nicht gestrichenen Zahlen in der Liste sind Primzahlen.

Algorithmus in C Syntax:
Der folgende Code ist lauffähig.

```
#include <stdio.h>
#define N 100

int main(void) {

        // Sieb als Array definieren.
        // Der Index des Arrays laeuft von 0 bis N, also durch N+1 Stellen.
        int array[N+1];
        int k, n;

        // Sieb (Array) initialisieren:
        for (k = 0; k <= N; k++) {
                array[k] = 0;
        }

        // Alle Vielfachen von Primzahlen im Array markieren.
```

```
        // Alle Kandidaten k von 2 bis N durchlaufen:
        for (k = 2; k <= N; k++) {

                // Zahl k ist eine Primzahl, falls im Array nicht markiert:
                if (array[k] == 0)
                        // Zahl k ist eine Primzahl.
                        // Nun werden alle Vielfachen im Array markiert:
                        n = 2;
                        while (n*k <= N) {
                                array[n*k] = 1;
                                n++;
                        }
        }

        // Primzahlen anzeigen.
        // Alle Kandidaten k von 2 bis N durchlaufen:
        for (k = 2; k <= N; k++) {
                // Ist k eine Primzahl?
                if (array[k] == 0) {
                        // Ja, k ist eine Primzahl.
                        printf("   %d\n", k);
                }
        }
}
```

Eigenschaften des Algorithmus

- Deterministisch:
  Sowohl der Algorithmus, als auch das Resultat sind deterministisch. Bei gleicher Ausgangslage (Parameter N) durchläuft der Algorithmus immer genau die gleiche Abfolge von Elementarschritten und erzielt stets das selbe Resultat, nämlich die Liste aller Primzahlen zwischen 2 und N.

- Stochastisch:
  Der Algorithmus ist nicht stochastisch, da es keine Zufallskomponenten gibt. Statt dessen ist er, wie oben erläutert, deterministisch.

- Terminierend:
  Der Algorithmus ist terminierend, da er für jedes N eine endliche und wohl definierte Anzahl[1] von Primzahlen findet. Entsprechend der Idee des Siebs von Eratosthenes ist auch die Suche nach diesen Primzahlen mit endlichem Aufwand verbunden und daher terminierend.

1.2 Fakultät

Die Fakultät $n!$ können wir auch so schreiben:

$$n! = n \cdot (n-1)!$$

[1] Falls N gegen unendlich strebt, so wird die Anzahl Primzahlen ebenfalls unendlich, aber abzählbar unendlich.

Zudem gilt $1! = 1$ und $0! = 1$. Damit können wir die rekursive Vorschrift in Worten formulieren:

1. Start der Fakultät-Berechnung.
   Es wird ein Parameter übergeben, wie nennen ihn n.
2. Falls $n \leq 1$: Gib das Resultat 1 zurück.
   Sonst: Gib das Resultat $n \cdot (n-1)!$ zurück, wobei $(n-1)!$ ein erneuter Aufruf der Fakultät-Berechung mit dem Parameter $n-1$ ist.

Implementation des rekursiven Algorithmus in C:

```
#include <stdio.h>

// ------------------------------------------------
// Funktion:
int fakultaet(int n) {
        if (n <= 1) {
                return(1);
        } else {
                return(n*fakultaet(n-1));
        }
}

// ------------------------------------------------
// Hauptprogramm:
int main(void) {
        // Variablen:
        int n;
        int resultat;

        // Eingabe der Zahl n:
        printf("   Zahl n = ");
        scanf("%d", &n);

        // Fakultaet berechnen:
        resultat = fakultaet(n);

        // Resultat anzeigen:
        printf("   Fakultaet n! = %d\n", resultat);
}
```

Eigenschaften des Algorithmus

- Deterministisch:
  Der Algorithmus ist deterministisch, genauso wie das Resultat. Für jeden Wert des Parameters n wird eine definierte Anzahl von Rekursionen durchgeführt.
- Stochastisch:
  Der Algorithmus ist nicht stochastisch, da es keine Zufallskomponenten gibt. Statt dessen ist er, wie oben erläutert, deterministisch.

- Terminierend:
  Der Algorithmus ist terminierend, denn zu jedem (endlichen) n gibt es eine wohl definierte Anzahl von Rechenschritten, die zum Resultat führen.

Wenn mit dem oben gezeigten Code 50! berechnet werden soll, so entstehen auf einem üblichen Rechner Überläufe. In der Folge liefert das Programm ein falsches Resultat. Die grösste darstellbare Zahl hängt vom verwendeten Integer-Format ab. Die folgenden Formate sind typisch:

Integer-Format	grösster darstellbarer Wert
16 Bit	32'767
32 Bit	2'147'483'647
64 Bit	$9.22 \cdot 10^{18}$

Das korrekte Resultat wäre aber $50! = 3.04 \cdot 10^{64}$, was in keinem der typischen Integer-Formate dargestellt werden kann.

1.3 Nicht terminierender Algorithmus

Wenn die Fehlergrenze $\varepsilon = 0$ gesetzt wird, terminiert das Verfahren im Allgemeinen nicht mehr. Als Beispiel betrachten wir $X = 2$. Die Zahl $\sqrt{2}$ ist - wie man leicht zeigen kann - irrational[2] und hat demnach unendlich viele, nicht-periodische Nachkommastellen. Das Heron-Verfahren muss bei $\varepsilon = 0$ alle diese Stellen nachbilden, was unendlich lange dauert.

[2] Irrational heisst, dass die Zahl nicht als Bruch dargestellt werden kann. Der Beweis, dass $\sqrt{2}$ irrational ist, stammt von Euklid (3. Jh. v. Chr.).