Programmieren für Physiker

Interfakultatives Institut für Anwendungen der Informatik
Institut für Theoretische Teilchenphysik

Prof. Dr. M. Steinhauser, Dr. A. Mildenberger
http://comp.physik.kit.edu

SS 2011 – Blatt 09
Bearbeitungszeitraum: bis 15. Juni 2011

Aufgabe 23: Quicksort **Pflichtaufgabe**

Quicksort (C.A.R. Hoare, 1961/62) ist ein schneller, rekursiver Sortieralgorithmus. Hier eine Skizze der Funktionsweise: Es wird aus der zu sortierenden Liste ein „Pivotelement“ m gewählt (im einfachsten Fall kann das erste Element der Liste benutzt werden) und die Liste wird in einem Durchgang in zwei Teillisten geordnet. In der linken Teilliste sind alle Elemente $\leq m$, alle Elemente der rechten Teilliste sind $\geq m$. Dieser Schritt heißt auch Partitionierung. Nun können die beiden noch unsortierten Teillisten jeweils für sich mit dem gleichen Verfahren bearbeitet werden, dies geschieht rekursiv. Teillisten mit weniger als zwei Elementen brauchen natürlich nicht mehr sortiert zu werden.
Schreiben Sie eine Funktion `QuickSort (double A[], int s, int t)`, die im Feld `A` zwischen den Indizes `s` und `t` sortiert. Bei Bedarf soll diese Funktion eine Partitionierung durchführen und sich selbst aufrufen. Entwerfen Sie eine eigene Partitionierung oder verwenden Sie folgendes Schema:

```
Partitionierung (A, s, t)
  pivot := A[s]
  l := s
  Schleife i von s+1 bis t
    Falls A[i] < pivot
    dann:  l := l+1
           Tausche (A[i],A[l])
  Tausche (A[s],A[l])
```

Wie funktioniert diese Partitionierung? Wo steht nach Ausführung dieser Routine das Pivotelement, welche zwei Teillisten sind nun also zu sortieren?

Um Ihre Sortierfunktion zu testen, generieren Sie ein Feld mit 1000 zufälligen `double`-Werten im Bereich $[0, 1]$. Jeden von diesen erzeugen Sie am Einfachsten durch `rand()/(double)RAND_MAX`, dabei benötigen Sie im Kopf `#include <cstdlib>`. Diesen eingebauten Generator initialisieren Sie z.B. durch einmaliges `srand((unsigned int)time(0))`. Verwenden Sie zum Tausch von Feldelementen und für die Partitionierung jeweils Funktionen.

Überlegen Sie sich bitte, wie viele und welche Modifikationen in Ihrem Programm nötig sind, um die Zahlenliste absteigend statt aufsteigend zu sortieren.

Zusatzfragen (freiwillig): In welchen Ausgangssituationen ist die Laufzeit von Quicksort in der obigen Version am ungünstigsten, d.h. sind am meisten Vergleiche durchzuführen? Messen Sie die Laufzeit des Programms mit geeignet langen Zahlenfeldern für verschiedene Ausgangssituationen.

Aufgabe 24: Auflösung einer Rekursion **freiwillig**

Gegeben sei eine rekursive Funktionen, die folgende, relativ einfache Struktur habe:

```
Type2 f (Type1 x)
{
  if( test(x)) return q(x) ;
  return f(p(x)) ;
}
```

Dabei sind `Type1 p (Type1)` und `Type2 q (Type1)` sowie `bool test (Type1)` beliebige Funktionen, die hier für Rechenoperationen mit dem Argument `x` stehen. Geben Sie mit Hilfe einer Schleife eine nicht-rekursive Formulierung der obigen Funktion `f` an.

Zum Knobeln: Selbstbeschreibende Folge

Es gibt genau eine monoton wachsende Folge (a_n), $n = 1, 2, \ldots$ von positiven ganzen Zahlen, die folgende Eigenschaft hat: Jede Zahl n steht genau a_n mal in der Folge.
Verdeutlichen Sie sich, dass der Folgenanfang so aussieht:

n	1	2	3	4	5	6	7	8	9	10	11	12
a_n	1	2	2	3	3	4	4	4	5	5	5	6

Schreiben Sie eine Funktion, die für ein gegebenes n den Folgenwert a_n berechnet.

Einige Folgenwerte zu Testzwecken:

n	100 000	1 000 000	10 000 000	100 000 000	1 000 000 000
a_n	1479	6137	25474	105729	438744

Kann Ihr Programm auch den letzten Wert der Tabelle in weniger als einer Minute berechnen?
