
Vorlesung 9

Vorlesung9

- Funktionen, die andere Funktionen benutzen!
- Rekursive Funktion
- Mergesort und Quicksort

Problemstellung1

Aufgabe:

Es soll ein Programm geschrieben werden, das die Standardabweichung der eingegeben Zahlen berechnet.

Der Mittelwert von N Werten

$x_0, x_1, x_2 \dots x_{N-1}$ ist:

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x_i = \frac{x_0 + x_1 + x_2 + \dots + x_{N-1}}{N}$$

Die Standardabweichung als Maß für die mittlere Abweichung der Einzelwerte vom Mittelwert ist so definiert:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \bar{x})^2}$$

Verfahren

Lösungsweg:

- Mittelwert berechnen.
- Mittelwert benutzen um die Standardabweichung zu berechnen.

Funktion, die andere Funktion aufruft.

Lösungsweg:

- Eine Funktion für die Mittelwertbildung implementieren.
- Eine zweite Funktion für die Standardabweichung schreiben, in der man die Funktion der Mittelwertbildung benutzt.

Funktion, die andere Funktion aufruft.

Lösungsweg:

- Für die Mittelwertbildung wird eine Funktion verwendet, die das Array mit den Zahlenwerten übergeben wird.

double mittelwert(double x[],int anz);

Funktion, die andere Funktion aufruft.

Lösungsweg: Definition der Funktion mittelwert.

```
double mittelwert(double x[ ],int anz)  
{  
    int i;  
    double m;  
    m=0.0;  
    for(i=0;i<anz;i++)  
        m=m+x[i];  
    m=m/(double)anz;  
  
    return m;  
}
```

Funktion, die andere Funktion aufruft.

Lösungsweg: Berechnung der
Standardabweichung die Funktion streuung.

double streuung(double x[], int anz);

Funktion, die andere Funktion aufruft.

Lösungsweg: Berechnung der Standardabweichung die Funktion streuung.

```
double streuung(double x[],int anz)
{
    double m = mittelwert(x,anz);
    double s = 0.0;
    int i;
    for(i=0;i<anz;i++)
        s+=(x[i]-m)*(x[i]-m);
    return sqrt(s);
}
```

vorlesung9.cpp

Funktion, die sich selber aufruft: Die Rekursion

Problemstellung2

Aufgabe: Schreiben Sie ein Programm, das die
Berechnung: 3^y durch rekursiven Aufruf
realisiert.

Die Rekursion: Definition

- Rekursiver Schritt $3^y = 3 * 3^{y-1}$
- Ende $3^0 = 1$

Funktion, die sich selber aufruft: Die Rekursion

Lösungsweg:

Für $y=2$ ergibt sich folgendes Aufrufschema und als Ergebnis 9:

1. `DreiPotenz(2)` ---> `return 9;`

2. `DreiPotenz(1)` --> `return 3;`

3. `DreiPotenz(0)` ---> `return 1;`

Funktion, die sich selber aufruft: Die Rekursion

Lösungsweg:

Für die Berechnung von 3^y wird eine Funktion verwendet, der y mit dem Zahlenwert übergeben wird.

int DreiPotenz (int y);

Funktion, die sich selber aufruft: Die Rekursion

Lösungsweg:

```
int DreiPotenz (int y)  
{  
    if (y==0)  
        return 1;  
    else return 3* DreiPotenz(y-1);  
}
```

Funktion, die sich selber aufruft: Die Rekursion

Allgemein gilt, dass rekursive Lösungsansätze Speicherplatz- und Laufzeit intensiver sind, der Programmcode dafür aber kompakter und eleganter. So dass die Rekursion mitunter sehr effektive Algorithmen ermöglicht: Ein schönes Beispiel dafür ist das Sortieralgorithmus Quicksort.

Problestellung3 (Quicksort)

Aufgabe:

Es soll ein Programm geschrieben werden, das die folgende Liste [27 59 3 17 41 45 12 13 86 33 67 1] unter der Benutzung von Quick sortiert .

Quicksort

Lösungsweg

[27 59 3 17 41 45 12 13 86 33 67 1]

Quicksort

Lösungsweg

Schranke wählen hier die 27

[27 59 3 17 41 45 12 13 86 33 67 1]

Quicksort

Lösungsweg

Schranke in der Mitte setzen.

[12 59 3 17 41 45 27 13 86 33 67 1]

Quicksort

Lösungsweg

Neu Schranke bzw neu Schranken suchen.

[12 59 3 17 41 45 27 13 86 33 67 1]

Schranken 12 und 45.

Quicksort

- Schranken in der Mitte.

[3 1 12 17 13 27 33 41 45 86 67 59]

Quicksort

- Neu Schranken finden

[3 1 12 17 13 27 33 41 45 86 67 59]

Schranken: 3,17,33,86

[**3** 1 12 **17** 13 27 **33** 41 45 **86** 67 59]

Quicksort

- Wichtig für die Effizienz von Quick-Sort ist eine gute Wahl der Schranke. Quick-Sort ist am effizientesten, wenn das Array bei jeder Zerlegung in gleich große Teile partitioniert wird.**
- Man muss also dafür sorgen, dass alle Elemente, die kleiner oder gleich einer bestimmten Schranke sind, in den linken Teil und alle Elemente größer gleich der Schranke in den rechten Teil des Arrays wandern. Dies wird durch die Funktion 'partition' erreicht.**
- Nach dem Aufteilen des Arrays wird auf beide Teile das gleiche Verfahren wieder angewendet. Dies geschieht durch den rekursiven Aufruf der Quick-Sort-Routine.**

Quicksort

- Die Abbruchbedingung der Rekursion ist dann gegeben, wenn das zu sortierende Teil-Array leer ist oder nur noch aus einem einzigen Element besteht (was sollte dann noch sortiert werden?).

Algorithmen: *Problemstellung3*

- Aufgabe:

Es soll ein Programm geschrieben werden, das die folgende Liste [9,6,2,8,4,7,3,5,1] unter der Benutzung von Mergesort sortiert .

Algorithmen: Sortierung

Lösungsweg

[9 6 2 8 4 7 3 5 1]

-Die Liste wird in der Mitte geteilt.

([9 6 2 8] [4 7 3 5 1])

Algorithmen: Sortierung

Lösungsweg

- beide Hälften werden rekursiv sortiert
- und die Ergebnislisten geordnet
zusammengemischt.

Algorithmen: Sortierung

Lösungsweg Mergesort

$([9\ 6\ 2\ 8] \quad [4\ 7\ 3\ 5\ 1])$

$(([9\ 6] \ [2\ 8]) \ ([4\ 7] \ [3\ 5\ 1]))$

Algorithmen: Sortierung

Lösungsweg II

- Sortieren durch Mergesort:

$([9\ 6\ 2\ 8] \quad [4\ 7\ 3\ 5\ 1])$

$(([9\ 6] \ [2\ 8]) \ ([4\ 7] \ [3\ 5\ 1]))$

$((([9] \ [6]) \ ([2] \ [8])) \ (([4] \ [7]) \ ([3] \ [5\ 1])))$

Algorithmen: Sortierung

Lösungsweg II

- Sortieren durch Mergesort:

$([9\ 6\ 2\ 8]\quad [4\ 7\ 3\ 5\ 1])$

$(([9\ 6]\ [2\ 8])\ ([4\ 7]\ [3\ 5\ 1]))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ [5,1])))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ ([5]\ [1]))))$

Algorithmen: Sortierung

- Sortieren durch Mergesort:

$([9\ 6\ 2\ 8]\quad [4\ 7\ 3\ 5\ 1])$

$(([9\ 6]\ [2\ 8])\ ([4\ 7]\ [3\ 5\ 1]))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ [5,1])))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ ([5]\ [1]))))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ [1\ 5])))$

Algorithmen: Sortierung

$([9\ 6\ 2\ 8]\quad [4\ 7\ 3\ 5\ 1])$

$(([9\ 6]\ [2\ 8])\ ([4\ 7]\ [3\ 5\ 1]))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ [5,1])))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ ([5]\ [1]))))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ [1\ 5])))$

$(([6\ 9]\ [2\ 8])\ ([4\ 7]\ [1\ 3\ 5]))$

Algorithmen: Sortierung

$([9\ 6\ 2\ 8]\quad [4\ 7\ 3\ 5\ 1])$

$(([9\ 6]\ [2\ 8])\ ([4\ 7]\ [3\ 5\ 1]))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ [5,1])))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ ([5]\ [1]))))$

$((([9]\ [6])\ ([2]\ [8]))\ (([4]\ [7])\ ([3]\ [1\ 5])))$

$(([6\ 9]\ [2\ 8])\ ([4\ 7]\ [1\ 3\ 5]))$

$([2\ 6\ 8\ 9]\ [1\ 3\ 4\ 5\ 7])$

Algorithmen: Sortierung

Loesungsweg:

([9 6 2 8] [4 7 3 5 1])

(([9 6] [2 8]) ([4 7] [3 5 1]))

((([9] [6]) ([2] [8])) ([4] [7]) ([3] [5, 1])))

((([9] [6]) ([2] [8])) ([4] [7]) ([3] ([5] [1]))))

((([9] [6]) ([2] [8])) ([4] [7]) ([3] [1 5])))

(([6 9] [2 8]) ([4 7] [1 3 5]))

[2 6 8 9] [1 3 4 5 7]

[1 2 3 4 5 6 7 8 9]

Quicksort

Aufgabe