Klausurvorbereitung
WiSe-2024
Algorithmen & Datenstrukturen - I
Aufgabensammlung

25. November 2024

Inhaltsverzeichnis

1 Aufgabenbereich: Probleme & Algorithmen

1.1 Beschreibung von Algorithmen

1. Ist ein Rezept in einem Kochbuch ein Algorithmus? Begründen Sie Ihre Antwort.

2. Welche Vor- und Nachteile hat es, Algorithmen direkt in einer Programmiersprache wie Java zu beschreiben (Statt in einer Umgangssprache oder Pseudocode)?

3. Beschreiben Sie umgangssprachlich den Algorithmus zur stellenweisen Addition zweier Dezimalzahlen. Wovon hängt die Laufzeit des Verfahrens wesentlich ab?

4. Wie gehen Sie vor, wenn Sie die Summe von n Zahlen berechnen wollen, aber in einem Schritt immer nur zwei Zahlen addieren können? Wieviele Additionen braucht man dafür?

1.2 Verifikation von Algorithmen: Addition (1/2)

Gegeben sei folgender Algorithmus zur Addition zweier ganzer Zahlen x und y.

$$x, y \in \mathbb{Z}$$

Folgender Code (Python) ist gegeben:

```
def ADD(x,y):
    s = x
    i = y
    while i is not 0:
        s = s + 1
        i = i - 1
    return s
```

beachten Sie, dass $x, y \in \mathbb{Z}$ auch Negativ sein können.

1.3 Verifikation von Algorithmen: Addition (2/2)

a) Für welche Inputs terminiert der Algorithmus, für welche nicht?
Begründen Sie Ihre Antwort.

b) Zeigen Sie, dass der Algorithmus partiell korrekt ist, also das richtige Ergebnis liefert, falls er terminiert. Finden Sie dazu eine Beziehung (Gleichung) zwischen den verwendeten Variablen, die vor Beginn und nach jedem Durchlauf der while-Schleife gilt - eine sogenannte **Schleifeninvariante**.

c) Durch welche Modifikation der while-Anweisung kann man erreichen, dass der Algorithmus immer terminiert?
Ist er dann noch korrekt?

d) Wie verhält sich der Algorithmus, wenn x oder y reelle Zahlen sind?

2 Aufgabenbereich: Analyse von Algorithmen

2.1 Binäre Suche

Suchen Sie in der Folge $[2, 12, 25, 32, 40, 43, 47, 48, 50, 99]$ nach

a) x = 32

b) x = 49

Geben Sie jeweils die Reihenfolge der Elemente an, mit denen x verglichen wird.

2.2 Maximum von n Zahlen

Die Position des Maximums einer eingegebenen Folge von n Zahlen soll bestimmt werden.

a) Geben Sie einen einfachen Algorithmus dazu in Pseudocode an.

b) Was liefert Ihr Algorithmus, wenn das Maximum mehrfach in der Folge vorkommt?

c) Wie viele Vergleiche benötigt Ihr Algorithmus im Best / Worst Case?

d) Können Sie einen besseren Algorithmus finden, wenn Sie wissen, dass die eingegebene Folge schon aufsteigend Sortiert ist?

e) Lohnt es sich also, eine Folge erst zu sortieren, nur um das Maximum zu bestimmen?

f) Fomulieren Sie eine **Schleifeninvariante** für die zentrale Schleife Ihres Algorithmus aus ***a)***.

2.3 Average-Case-Aufwand der Linearen Suche

Bestimmen Sie die durchschnittliche Anzahl von Vergleichen fpr die lineare Suche in einer Folge von $n \in \mathbb{N}$ verschiedenen Zahlen.

a) Für eine erfolgreiche Suche unter der Annahme, dass nach jedem der $n \in \mathbb{N}$ vorhandenen Elemente gleich wahrscheinlich gesucht wird.

b) Unter der Annahme, dass die Suchanfrage eine (gleichverteilte) 64-Bit-Ganzzahl ist, und die Folge $n \in \mathbb{N}$ verschiedene solche Zahlen enthält.

Was bedeutet das Ergebnis für praktische Anwendungen?

3 Aufgabenbereich: Einfache Sortierverfahren

3.1 Sortieren durch Auswahl

Die Folge $[21, 22, 54, 17, 64, 30, 94, 74, 84, 90]$ wird mit Selection Sort sortiert. Geben Sie jeweils den Zustand der Folge nach den ersten 4 vertauschungen an.

Beispiel Code: (Python)

```python
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i+1, len(arr)):
            if arr[min_idx] > arr[j]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

3.2 Sortieren durch Einfügen

Die Folge $[20, 13, 32, 11, 88, 98, 46, 24, 39, 61]$ wird mit Insertion Sort sortiert. Geben Sie für die ersten 11 Bewegungen (Zuweisungen) jeweils den Zustand der Folge und des Zwischenspeichers t nach der Bewegung in einer Tabelle an.

Wenn man das folgende verfahren ohne weitere Optimierung verwendet, werden auch Elemente, die "richtig" stehen, in den Zwischenspeicher t und zurück bewegt. Zählen Sie solche, eigentlich unnötigen, Bewegungen mit.

Beschreibung: In Iteration i fügt man a_i an der richtigen Stelle in die bereits sortierte Teilfolge a_0,..., a_{i-1} ein:

1) t = a_i zwischenspeichern
2) jeden größeren Wert links davon um 1 Position nach rechts verschieben, um Platz zu schaffen.
3) In die entstandene Lücke den Wert t einfügen.

Beispiel Code: (Python)

```python
def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        t = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > t:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = t
    return arr
```

3.3 Laufzeitprognose für Insertion Sort

Angenommen, Ihre Implementierung von Insertion Sort braucht zum Sortieren von $n = 100.000$ Elementen 1 Sekunde. Wie lange würde voraussichtlich das Sortieren von $2n = 200.000$ bzw. von $10n = 1.000.000$ Elementen dauern? Begründen Sie Ihre Antwort.

4 Aufgabenbereich: Mathematische Analyse

4.1 Verdopplung der Eingabegröße

a) Wie entwickelt sich vorausssichtlich die Rechenzeit eines Algorithmus mit asymptotischer Laufzeit

- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(n \log(n))$
- $\Theta(\log(n))$

Wenn die Eingabegröße verdoppelt wird?

b) Wie zuverlässig sind solche Prognosen für reale Anwendungen?

4.2 Induktionsbeweis: Summe der ungeraden Zahlen

Beweisen Sie durch Induktion:

$$\sum_{j=1}^{n}(2i-1) = n^2 \text{ für alle } n \in \mathbb{N} : n \geq 1$$

4.3 Asymptotischer Vergleich von Funktionen

Ordnen Sie folgende Funktionen aufsteigend bezüglich der $\mathcal{O}$-Notation:

$f1 = 4n \log(n)$

$f2 = n^n$

$f3 = 1000n^2$

$f4 = (\sqrt{n})^3$

$f5 = 5 \log(n)$

$f6 = n^2$

$f7 = \log(\log(n))$

$f8 = 2^n$

Warum ist es dafür egal, zu welcher Basis die Logarithmen gebildet werden?

4.4 Laufzeitprognosen

a) Ein Algorithmus mit quadratischer Laufzeit $\Theta(n^2)$ benötige für die Lösung eines Problems mit $n = 2.000.000$ Elementen eine Rechenzeit von 2 minuten. Untersuchen Sie, wie sich die Rechenzeit voraussichtlich (nährerungsweise) ändern wird, wenn sich die Problemgröße verfünffacht.

Hinweise :

1: $n \rightarrow 5n$

2: $n = 2.000.000 \rightarrow 5n = 10.000.000$

3: $T(n) = 2min \rightarrow T(?)$

4: $\Theta(n^2) \rightarrow \Theta(?)$

4.5 Maximale Problemgröße pro Zeit

Wenden Sie die Laufzeitprognose umgekehrt an und untersuchen, welche maximale Problemgröße n ein Algorithmus in einer vorgegebenen Zeit bewältigen kann. Angenommen, die Funktionen $f(n)$ in der Tabelle geben für eine Eingabemenge der Größe n die Laufzeit eines Algorithmus in Mikrosekunden an. Berechnen Sie jeweils die größte Problemgröße n, die in der angegebenen Zeit gelöst werden kann. *Hinweis:* Runde Sie große Zahlen, wenn nötig.

$f(n)$	**1 Minute**	**1 Tag**	**1 Jahr**
$\sqrt{n}$			
n			
$n \log_2(n)$			
n^2			
2^n			
$n!$			

Tabelle 1: Maximale Problemgrößen für verschiedene Laufzeitfunktionen

5 Aufgabenbereich: Grundlegende Datenstrukturen

5.1 Stack Simulation

Auf einem Stack von Buchstaben führen sie die folgende Sequenz von Operationen aus:

$$NSH - A - QDR - - - EU - - - OT - I - --$$

Ein Buchstabe bedeutet dabei eine **push-**, ***ein Strich*** bedeutet eine **pop-Operation**.

a) Geben Sie die von den pop-Operationen gelieferte Ausgabefolge an.

b) Wie ändert sich das Ergebnis, wenn man statt einem Stack eine Queue verwendet?

5.2 Stacks: Welche Ausgabe ist möglich?

- Auf einem Stack wird eine gemischte Folge von 10 push- und pop- Operationen ausgeführt.
- Die push- Operationen legen nacheinander die Ziffern 0 bis 9 auf den Stack.
- Die pop- Operationen drucken jeweils die entfernte Ziffer aus.
- Wie sich die push- und pop-Operationen abwechseln, ist nicht bekannt.

Welche der folgenden Ausgabefolgen können so prodziert werden? Geben Sie jeweils eine Folge von push- und pop- Operationen an, die zu dieser Ausgabe führt, oder begründen Sie, warum die Ausgabe nicht möglich ist.

1) 2 1 3 0 4 5 6 7 9 8
2) 1 3 0 2 7 9 8 6 5 4
3) 0 2 3 4 1 5 6 7 9 8
4) 6 5 4 3 2 1 0 7 8 9
5) 1 3 0 4 6 5 8 9 7 2

5.3 Queues: Welche Ausgabe ist möglich?

- Auf einer Queue wird eine gemischte Folge von 10 enqueue- und dequeue-Operationen ausgeführt.
- Die enqueue- Operationen fügen nacheinander die Ziffern 0 bis 9 in die Queue ein.
- Die dequeue- Operationen drucken jewils die entfernte Ziffer aus.
- Wie sich die enqueue- und dequeue- Operationen abwechseln, ist nicht bekannt.

Welche der folgenden Ausgabefolgen können so prodziert werden? Geben Sie jeweils eine Folge von enqueue- und dequeue- Operationen an, die zu dieser Ausgabe führt, oder begründen Sie, warum die Ausgabe nicht möglich ist.

1) 2 1 3 0 4 5 6 7 9 8

2) 1 3 0 2 7 9 8 6 5 4

3) 0 2 3 4 1 5 6 7 9 8

4) 6 5 4 3 2 1 0 7 8 9

5) 1 3 0 4 6 5 8 9 7 2

6 Aufgabenbereich: Rekursion

6.1 Rekursive Implementierung der binären Suche

Implementieren Sie die binäre Suche rekursiv in Pseudocode:

- binaere_suche(a, *unten*, *oben*, x) sucht in folge a zwischen Index *unten* und Index *oben* (beide inklusive) nach dem wert x.
- Wie sieht der *oberste* Aufruf für die Suche in der gnazen Folge aus?
- Was ist die Abbruchbedingung?
  (Tipp: betrachten Sie den Fall, dass das gesuchte Element nicht gefunden wird)
- Stellen Sie sicher, dass das Ergebnis für "nicht gefunden" unmissverständlich ist.

6.2 Analyse einer rekursiven Funktion

Gegeben sei folgende rekursive Funktion für $n \in \mathbb{N}$:

$$F : \mathbb{N} \to \mathbb{N} \qquad F(n) = \begin{cases} 0 & (n \leq 1) \\ F(n-2) + 1 & (n > 1) \end{cases}$$

a) Berechnen Sie $F(n)$ für einige kleine $n \in \mathbb{N}$.

b) Welche Mathematische Funktion wird von F berechnet?

c) Für die Laufzeitanalyse der Funktion $F(n)$ zählen wir jetzt die Anzahl der Additionen und Subtraktionen, $T(n)$. Stellen Sie eine Rekusionsgleichung für $T(n)$ auf.

d) Geben Sie eine geschlossene Formel für $T(n)$ an.

6.3 Auf und Ab

Schreiben Sie eine rekursive Funktion $aufundab(unten, oben)$, die von der ganzen Zahl *unten* zur ganzen Zahl *oben* hoch- und wieder herunter- zählt. Sie dürfen nur Rekursion verwenden, keine Schleifen.

- Beispiele:

  a) $aufundab(2, 5)$ gibt 2 3 4 5 4 3 2 aus.

  b) $aufundab(3, 3)$ gibt 3 2 1 2 3 aus.

  Tipp: Suchen Sie im Ergebnis von $aufundab(2, 5)$ nach einer "etwas kleineren" Teilfolge, die sich durch einen rekursiven Aufruf von $aufundab$ erzeugen lässt. Was muss man tun, um aus dieser Teilfolge die gesamte Folge zu erzeugen? Wann bricht die Rekursion ab?