Ole Elija Dziewas

AuD-Blatt2

Aufgabe 1) Geordnete Funktionen nach asymptotischer Komplexität

1) $\log_2 n^{49} = \log_2 n^7 \in O(\log_2 n)$

2) $c \cdot n^2 \in O(n^2)$

3) $10 \cdot n^2 + \frac{1}{2} \cdot n^3 \in O(n^3)$

4) $3 \cdot 2^n \in O(2^n)$

5) $2 \cdot 3^n \in O(3^n)$

6) $\frac{n!}{n} \in O(n!)$

Aufgabe 2) Beweise oder Widerlege

a) $5 \cdot n^2 + n \cdot \log_2 n^n \in O(n^2)$

$$= 5 \cdot n^2 + n^2 \cdot \log_2 n$$

Beweis mit lim: $\lim_{n\to\infty} \frac{5 \cdot n^2 + n^2 \cdot \log_2 n}{n^2} = 5 + \log_2 n$

Da $\log_2 n$ keine konstante handelt, ist die oben stehende Aussage ein Widerspruch und somit falsch.

b) Nach dem Theorem der polynomialen Kostenfunktionen werden $P(n) = \sum_{i=0}^{g} a_i n^i$ durch den höchsten Grad g dominiert.

$$(n + a)^b = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

Auch bei $(n + a)^b$ handelt es sich um eine solche polynomiale Kostenfunktion, die also den höchsten Grad n hat und damit zur Komplexitätsklasse O(n^b) gehört. Da n^b natürlich zur selben Komplexitätsklasse gehört gilt: $f \in \theta(g) \Leftrightarrow f \in O(g) \wedge g \in O(f)$. Damit ist Aussage b korrekt.

Aufgabe 3) Suche in Zeichenketten-Algorithmus

a) Idee: Die Zeichenkette wird nach und nach durchgegangen, bis eine Übereinstimmung des ersten Buchstabens gefunden wird. Danach wird überprüft ob die folgenden Buchstaben auch stimmen. Wenn dies der Fall ist wird, die Stelle, an der der Teil String begonnen hat, zurückgegeben. Ansonsten wird die Suche am ersten falschen Buchstaben der Überprüfung fortgesetzt, bis entweder die Zeichenkette gefunden wird oder auch nicht.

b) Spezielle Eingabefälle: Wenn B.length() > A.length() ist, kann B kein Teilstring von A sein und der Algorithmus kann sofort -1 returnen. Wenn A oder B ein leerer String ist, wird das als fehlerhaft definiert und soll auch -1 zurückgeben.

c) Pseudo-Code:

Input: Zwei Character-Arrays oder Strings (String A, String B)

Result: Integer (Stelle, an der der Teilstring vorkommt oder -1, falls nicht vorhanden)

```
if (B.length() > A.length()) then
    return -1
if (B.length() == 0 || A.length() == 0) then
    return -1

boolean itsIn
for (i = 0; i<= A.length()-B.length(); i++) do
    itsIn = true
    if (A.charAt(i) == B.charAt(0)) then
        for (e = 1; e<B.length(); e++) do
            if (B.charAt(e) != A.charAt(i+e)) then
                itsIn = false
                i = i+e-1
                break
        if (itsIn == true) then
            return i+1

return -1
```

//Getestet in Intellij, oben ist Pseudo-Java-Code. Es wird nicht der Index, sondern die von eins an gezählte Stelle des Buchstaben zurückgegeben, wie im Aufgabenblatt vorgegeben

d) Die Elementaroperationen sind O(1). Die erste for-Schleife ist O(n). Da in der zweiten for-Schleife durch i=i+e-1 weitergezählt wird, wird die Komplexität nicht weiter erhöht. Da O(n), die höchste Komplexität ist, ist es auch die Gesamtkomplexität des Algorithmus.
Best Case (Teilstring wird direkt gefunden): n = Länge des Teilstrings B; $10 + n \cdot 2 + 1$
Worst Case (Teilstring wird gar nicht gefunden): a=Länge des Gesamtstrings; n = Länge des Teilstrings; $7 + (a - n) \cdot 2$

Aufgabe 4) Taytonym-Identifizierung-Algorithmus

a) Idee: Der Algorithmus überprüft, ob der Buchstabe an erster Stelle mit dem, nach der Hälfte der Länge des Teilstrings, übereinstimmt. Wenn dies der Fall ist ruft der Algorithmus die Funktion für den nächsten Buchstaben auf. Wenn der Algorithmus jedes Buchstabenpaar kontrolliert und bestätigt hat gibt er *true* zurück. In jedem anderen Fall wird *false* zurückgegeben.

b) Spezielle Eingabefälle: Wenn A eine ungerade Länge hat, kann A keine zwei identischen Teile haben. Ein leerer String wird auch hier als *false* definiert.

c) Pseudo-Code:

Input: String A (Zeichenkette die auf Taytonym überprüft wird), Integer i (Startindex = 0)

Result: boolean false oder true, je nachdem ob String Taytonym oder nicht

```
boolean identical(String A, int i)
    boolean res = false

    if (A.length()%2==1 || A.length() == 0) then
        return false

    if (i>= A.length()/2) then
        return true

    if (A.charAt(i) == A.charAt(i+A.length()/2)) then
        res = identical(A, i+1) //Rekursion

    return res
```

d) Innerhalb der Rekursion werden nur Elementaroperationen ausgeführt mit einer Komplexität von O(1). Die Rekursion selbst wird maximal $\frac{1}{2} \cdot n$ mal ausgeführt. Daher ist die Komplexität des gesamten Algorithmus O(n).
Best Case und Worst Case sind hier identisch, unter der Annahme, dass der Best Case das Herausfinden eines Taytonyms ist. Wenn der Best Case aber auch der Abbruch durch Sonderbedingungen oder ein Teilstring der kein Taytonym ist, sein kann ist der Best Case schon mit 2 Elementarschritten zu erreichen.
Der Worst Case entsteht wenn das Wort ein Taytonym ist, also jedes Buchstabenpaar überprüft werden muss. Bei einer Länge n des Wortes wäre der Worst Case: $6 \cdot \frac{n}{2} + 4$