Datenstrukturen und Algorithmen, SS 2024
Prof. Dr. P. Rossmanith
D. Cloerkes, M. Gehnen, D. Mock, N. Speetzen

Theoretische Informatik | RWTH AACHEN UNIVERSITY

Abgabe: Abgabe: 19.04.2024

Übungsblatt mit Lösungen 01

Aufgabe T1

Ordnen Sie die folgenden Funktionen in der Reihenfolge Ihres asymptotischen Wachstums (ohne lange Nachzudenken). Mit $\log n$ bezeichnen wir den Logarithmus zur Basis 2.

- $\log(n)$
- $\log(n^5)/\log\log(n)$
- $n^{\log\log n}$
- $\log(n)^{\log n}$
- n^2
- $n \log n$
- 2^n
- $\sqrt{n}$

Lösungsvorschlag

Mit $f(n) \prec g(n)$ drücken wir aus, dass $f(n) = O(g(n))$ gilt.

$$\log(n^5)/\log\log(n) \prec \log(n) \prec \sqrt{n} \prec n\log n \prec n^2 \prec n^{\log\log n} = \log(n)^{\log n} \prec 2^n$$

Aufgabe T2

Beweisen oder widerlegen Sie, dass $\frac{1}{2}n^2 + 7n + 14 = O(n^2)$ gilt.

Lösungsvorschlag

Wir müssen zeigen, dass es ein $c \in \mathbf{R}^+$ und ein $N \in \mathbf{N}$ gibt, sodass für alle $n \geq N$ gilt, dass $|\frac{1}{2}n^2 + 7n + 14| \leq c|n^2|$. Dazu wählen wir $c = 1,5$ und $n = 100$. Es ist mit hilfe der Standardmethoden der Analysis leicht auszurechnen, dass die Differenz der Funktionen $n^2 - 7n - 14$ sein lokales Extrema am Wert $3,5$ hat und dies ein minimum ist. Daher ist die Funktion ab $n = 3,5$ monoton steigend. Durch einsetzen können wir leicht verifizieren, dass bei $n = 100$ die Funktion positiv ist, somit ist auch die obige Ungleichung spätestens ab $n = 100$ erfüllt.

Aufgabe T3

Sie haben zwei *einfach* verkettete Listen gleicher Länge. Wie können den *zip* dieser Listen berechenen? Der *zip* zweier Folgen $a_1, \ldots, a_n$ und $b_1, \ldots, b_n$ ist einfach

$$a_1, b_1, a_2, b_2, \ldots, a_n, b_n.$$

Natürlich kann man einfach beide Listen durchgehen und eine neue List erzeugen. Falls man die alten Listen aber nicht mehr braucht, gibt es eine bessere Methode.

Lösungsvorschlag

```
class node :
  def __init__(self, x) :
    self.data = x
    self.next = self

def create_list(a) :
  head = node(None)
  q = head
  for x in a :
    qq = node(x)
    qq.next = head
    q.next = qq
    q = qq
  return head

def print_list(head) :
  q = head.next
  while q ≠ head :
    print(q.data)
    q = q.next

def zip(head1, head2) :
  q1, q2 = head1.next, head2.next
  while q1 ≠ head1 :
    qq1, qq2 = q1.next, q2.next
    q1.next, q2.next = q2, qq1
    q1, q2 = qq1, qq2

l1 = create_list([1, 2, 3, 4, 5])
l2 = create_list([11, 12, 13, 14, 15])

zip(l1, l2)
print_list(l1)
```

Aufgabe H1 (10 Punkte)
Beweisen Sie formal mithilfe der Definition der O-Notation, dass

$$\sqrt{2n(n+2)(n-1)} = \Theta(n^{3/2}).$$

Lösungsvorschlag
Wir schätzen die gegebene Funktion zuerst durch eine obere Schranke ab. Für $n \geq 2$ gilt

$$\sqrt{2n(n+2)(n-1)} \leq \sqrt{2n \cdot 2n \cdot n} = \sqrt{4n^3} = 2n^{3/2}.$$

Für $n \geq 2$ finden wir außerdem folgende untere Schranke.

$$\sqrt{2n(n+2)(n-1)} \geq \sqrt{2n \cdot n \cdot n/2} = \sqrt{n^3} = n^{3/2}.$$

Damit existieren Konstanten $c_1 = 1$, $c_2 = 2$ und $N = 2$, sodass

$$c_1 \cdot n^{3/2} \leq \sqrt{2n(n+2)(n-1)} \leq c_2 \cdot n^{3/2}$$

für alle $n \geq N$.

Aufgabe H2 (10 Punkte)
Tragen Sie für jede Kombination in der Tabelle ein, wie sie sich im Sinne der O-Notation zueinander verhalten.
Wenn f die Funktion einer Zeile und g die Funktion einer Spalte ist, dann finden Sie heraus ob $f = O(g)$, $f = \Omega(g)$ oder gar $f = \Theta(g)$ git. Tragen Sie dann O, Ω oder Θ in das entsprechende Feld ein. (Wenn mehrere Symbole richtig sind, wählen Sie das aussagekräftigste.)

	$n!$	$\log(n!)$	$\log(n^n)$	n^n
$n!$				
$\log(n!)$				
$\log(n^n)$				
n^n				

Lösungsvorschlag

	$n!$	$\log(n!)$	$\log(n^n)$	n^n
$n!$	Θ	Ω	Ω	O
$\log(n!)$	O	Θ	Θ	O
$\log(n^n)$	O	Θ	Θ	O
n^n	Ω	Ω	Ω	Θ

Die Botschaft dieser Aufgabe: Wenn $f = O(g)$ gilt, so bedeutet das noch lange nicht, dass auch $h(f) = O(h(g))$ gilt.

Aufgabe H3 (10 Punkte)
Neulich hat Matthias bemerkt, dass Autoren auch faule Menschen sind und Wörter mehrfach benutzen. Um sich ein Bild davon zu machen, wie verbreitet dieses Phänomen ist, beschließt er, die Häufigkeit jedes Wortes zählen zu lassen, und stellt euch daher folgende Aufgabe:
In einer doppelt verketteten Liste soll jedes Wort als Schlüssel gespeichert werden, das dazugehörige Datum die Anzahl des Wortes sein. Folgende Strategien sind denkbar:

- Naiv: Wenn ein Wort in der Liste gefunden wird, wird dort der Zähler um eins erhöht. Wird das Wort nicht gefunden, wird es an die hinten Liste angehangen.
- Move to Front: Wenn ein Wort in der Liste gefunden wird, wird der Zähler um eins erhöht und das Wort an den Anfang der Liste verschoben. Wird das Wort nicht gefunden, wird es vorne in der Liste eingefügt.

Schreiben Sie dazu je ein Programm, dass die benötigten Listenoperationen zählt. Der Einfachheit nehmen wir an, dass jedes Löschen, Einfügen, sowie ein Schritt auf das nächste Element je eine Operation ist. Was beobachten Sie bei der Anwendung auf die drei beigefügten Texte?
Konkret sind die Aufgaben:

- Schreiben Sie ein gut lesbares, kommentiertes Programm für beide Verfahren in einer Standardsprache (Java, C++, Python, ...), und hängen Sie es ihrer Lösung an.
- Wenden Sie beide Programme es auf die drei Texte an: Wieviele Operationen benötigen sie jeweils?
- Begründen Sie das erwartete Verhalten.

Bemerkung: Jeder der drei Texte enthält in der ersten Zeile die Anzahl der Wörter n, gefolgt von n Wörtern in je einer eigenen Zeile. Die Texte sind um Sonderzeichen und Groß/Kleinschreibung bereinigt. Ein Lösungsvorschlag wird in Java veröffentlicht.

Lösungsvorschlag

Die ungefähre Anzahl an Operationen sind:
Winnetou: naiv ca. 46 Mio, MTF 31 Mio.
Schatz im Silbersee: naiv ca. 308 Mio, MTF 213 Mio.
Beide: naiv ca. 982 Mio, MTF 310 Mio.

Die Ergebnisse sind nicht überraschend: Bei normalen Texten sind beide ungefähr gleich gut, (mit leichten Vorteilen für MTF), da häufige Wörter vermutlich auch schon relativ früh am Anfang eines Buches auftauchen. Taucht beispielsweise ein Name erst in der Mitte des Buches auf, dann aber häufig, ist MTF im Vorteil, da der Name sich vorne einordnet (und ein vielleicht verlassener Schauplatz der ersten Kapitel später nach hinten durchgereicht wird.)

Hängt man beide Werke aneinander, wie in ¡¡beide¿¿, so hat MTF einen deutlichen Vorteil. Dies liegt vermutlich daran, dass viele Wörter des ersten Werkes nicht im zweiten auftauchen, und umgekehrt. Im konkreten Beispiel ist das durchaus wahrscheinlich, immerhin ist ¡¡Winnetou¿¿ ein englisches Buch, und ¡¡De Schat in het Zilvermeer¿¿ in der niederländischen Sprache mit teils disjunktem Wortschatz ist.

(Gewählt wurden diese Sprachen, um mögliche Probleme mit Umlauten wie im Deutschen zu umgehen)

Eine Beispiellösung, die auf den Implementationen der Vorlesung aufbaut, ist die folgende:

```
package DSAL;

import DSAL.*;
import java.util.*;

public class KarlMay extends ADList<String, Integer> {
int numOperations = 0; // Operationen zählen

  void insertWord(String w) {
    Listnode<String, Integer> node = findnode(w);
    if(node == null) {
      append(w, 1);
      numOperations++;
    }else {
      node.data = node.data + 1;
    }
  }

  void insertWordMTF(String w) {
    Listnode<String, Integer> node = findnode(w);
    if(node == null) {
      prepend(w, 1);
      numOperations++;
    }else {
      int old_count = node.data;
      node.delete();
      prepend(w, old_count + 1);
      numOperations += 2;
    }
  }

  @Override
  protected Listnode<String, Integer> findnode(String k) {
    Listnode<String, Integer> n;
    head.key = k;
    for(n = head.succ; !n.key.equals(k); n = n.succ) { numOperations++; }
    head.key = null;
    if(n == head) return null;
    return n;
  }

  public static void main(String args[]) {
    KarlMay wordCountNormal = new KarlMay();
    KarlMay wordCountMTF = new KarlMay();
    Scanner in = new Scanner(System.in);
    int n;
    n = in.nextInt();
    for(int i = 0; i < n; i++) {
      String w;
      w = in.next();
      wordCountNormal.insertWord(w);
      wordCountMTF.insertWordMTF(w);
    }
    System.out.println("Ops normal: " + wordCountNormal.numOperations);
    System.out.println("Ops mtf:    " + wordCountMTF.numOperations);
  }
}
```

Aufgabe H4 (Zusatzaufgabe)
Finden Sie möglichst einfache Funktionen f.

a) $\sum_{k=1}^{n} k^{7/2} = f(n) + O(n^4)$

b) $(n+1)^n = en^n\big(1 + f(n) + O(n^{-2})\big)$

c) $\prod_{k=n}^{2n}\Big(1 + \frac{1}{k^2}\Big) = f(n) + O(1/n^2)$

Lösungsvorschlag

a) Hier reicht der Standardtrick, eine Summe als Integral zu approximieren: $\sum_{k=1}^{n} k^{7/2} \leq \int_{k=1}^{n} k^{7/2} \leq \sum_{k=1}^{n+1} k^{7/2}$. Da der Fehler von der Summe zum Integral also maximal $\sum_{k=1}^{n+1} k^{7/2} - \sum_{k=1}^{n} k^{7/2} = (n+1)^{7/2} = O(n^4)$ sein kann, kann man $f(n) = \int_{k=1}^{n} k^{7/2} = \frac{2}{9}n^{9/2} - 1$ wählen (wobei das -1 auch im Fehler von $O(n^4)$ verschwindet).

b) In dieser Aufgaben nehmen wir am Anfang den Logarithmus von $(n+1)^n = n^n(1+\frac{1}{n})^n$ (es erinnert ja an die Folgendefinition der Exponentialfunktion). Damit erhalten wir

$$\begin{aligned}\ln\big(1+\frac{1}{n}\big)^n &= n\ln\big(1+\frac{1}{n}\big)\\ &= n\big(\frac{1}{n} - \frac{1}{n^2} + O\big(\frac{1}{n^3}\big)\big),\end{aligned}$$

wobei wir im letzten Schritt die Taylorentwicklung des Logarithmus genutzt haben.

Wenden wir hierauf jetzt die Umkehrfunktion an, approximieren die Exponentialfunktion bei eins durch e und taylorn später die Exponentialfunktion, erhalten wir

$$\big(\frac{1}{n}\big)^n = e^{1-\frac{1}{n}+O(\frac{1}{n^2})} = ee^{-\frac{1}{n}+O(\frac{1}{n^2})} = e\big(1 - \frac{1}{n} + O\big(\frac{1}{n^2}\big)\big)$$

.

Schließlich fügen wir das fallengelassende n^n vom Beginn dieses Lösungsweges wieder hinzu, und erhalten $n^n e(1 - \frac{1}{n} + O(\frac{1}{n^2}))$.

(Die genannte Taylorentwicklung ist - falls das Konzept nicht bekannt sein sollte - in dieser Aufgabe nichts anderes als die Reihendarstellung, über die die Funktionen üblicherweise definiert werden).

c) Diese Aufgabe kombiniert die Tricks der vorherigen Teile (Summe integrieren, Taylorentwicklung sowie den Logarithmus zum vereinfachen nutzen).
Wieder fangen wir also mit dem Logarithmus von $\prod_{k=n}^{2n}(1+\frac{1}{k^2})$ an:

Betrachten wir also $\ln(\prod_{k=n}^{2n}(1+\frac{1}{k^2})) = \sum_{k=n}^{2n}\ln(1+\frac{1}{k^2}) = \sum_{k=n}^{2n}(\frac{1}{k^2} - \frac{1}{2k^4}) + O(\frac{1}{n^3})$ (hier wurde die Taylorentwicklung des ln genutzt, das $O(\frac{1}{n^3})$ versteckt das Restglied).

Nun formen wir $\frac{1}{k^2} = \frac{1}{k(k-1)} - \frac{1}{k} + \frac{1}{k^3-k^4}$ um, um später die eine Teleskopsumme nutzen

zu können:

$$\begin{aligned}
&\sum_{k=n}^{2n}\Big(\frac{1}{k(k-1)}-\frac{1}{k^3}+\frac{1}{k^3-k^4}-\frac{1}{2k^4}\Big)+O\Big(\frac{1}{n^3}\Big)\\
&=\sum_{k=n}^{2n}\Big(\frac{1}{k(k-1)}-\frac{1}{k^3}+O\Big(\frac{1}{k^4}\Big)\Big)+O\Big(\frac{1}{n^3}\Big)\\
&=\sum_{k=n}^{2n}\Big(\frac{1}{k(k-1)}-\frac{1}{k^3}\Big)+O\Big(\frac{1}{n^3}\Big)\\
&=\sum_{k=n}^{2n}\Big(\frac{1}{k-1}-\frac{1}{k}-\frac{1}{k^3}\Big)+O\Big(\frac{1}{n^3}\Big)\\
&=\frac{1}{n-1}-\frac{1}{2n}+\sum_{k=n}^{2n}\frac{1}{k^3}+O\Big(\frac{1}{n^3}\Big)\\
&=\frac{1}{n-1}-\frac{1}{2n}+\int_{k=n}^{2n}\frac{1}{k^3}+O\Big(\frac{1}{n^3}\Big)\\
&=\frac{1}{n-1}-\frac{1}{2n}-\frac{1}{2n^2}+\frac{1}{8n^2}+O\Big(\frac{1}{n^3}\Big)\\
&=\frac{1}{n}+\frac{1}{n^2}-\frac{1}{2n}-\frac{1}{2n^2}+\frac{1}{8n^2}+O\Big(\frac{1}{n^3}\Big)\\
&=\frac{1}{2n}+\frac{5}{8n^2}+O\Big(\frac{1}{n^3}\Big)
\end{aligned}$$

erhalten. Hierbei haben wir auch (wie in Aufgabenteil a) eine Summe durch ein Integral abgeschätzt.

Nun sind wir fast fertig, wir müssen uns lediglich noch mit dem Logarithmus aus dem ersten Schritt kümmern, also auf das ganze die Umkehrfunktion anwenden und erhalten $e^{\frac{1}{2n}+\frac{5}{8n^2}+O\left(\frac{1}{n^3}\right)}$. Wieder nutzen wir die Taylor-Approximation, diesmal der Exponentialfunkion, und erhalten

$$1+\frac{1}{2n}+\frac{5}{8n^2}+\frac{1}{8n^2}+O\Big(\frac{1}{n^3}\Big)=1+\frac{1}{2n}+\frac{3}{2n^2}+O\Big(\frac{1}{n^3}\Big),$$

womit wir sogar eine noch genauere Approximation erhalten als die Aufgabenstellung verlangt.