Mieke Narjes - Luka Hartwig - David Hoeck

Abgabe 03

Initialer Entwurf (De)Kompressionsalgorithmus

Wie funktioniert unser Verfahren im Prinzip?

Idee 1:

(evtl) Bits in Hex oder ASCII ‘umrechnen’ und dadurch um faktor 8 komprimieren

Das Verfahren:

1.
00010001 // 8 Zeichen einlesen

2.
[00000000, // In Byte-Darstellung umwandeln (48 abziehen)
00000000,
00000000,
00000001,
00000000,
00000000,
00000000,
00000001]

3.
[00000000, // Um 7 - Index nach links shiften
00000000,
00000000,
00010000,
00000000,
00000000,
00000000,
00000001]

4.
00010001 // Summe bilden

5.
In OutputStream schreiben und nächsten Block lesen

Idee 2:

1. Pattern suchen mit LZW
   nützliche Visualisierung - http://www.data-compression.com/lempelziv.shtml
2. evtl Patterns nach Häufigkeit sortieren und je nach Vorkommen mit kürzeren keys codieren (Huffman-code)
3. Dekomprimieren analog reverse

Wie greifen Wir die Erzeugungsstruktur der Daten auf?

Die Erzeugungsstruktur der Daten greifen wir nur in der zweiten Idee auf:
Hier suchen wir nach häufig auftretenden Sequenzen, die wir durch kürzere 'Codes' ersetzen.

Welche Datenstrukturen sind nötig?

Wir benötigen einen Buffer zum Einlesen,
Hashmap für das dictionary,
Binary tree für Huffmancode

Welches Datenformat besitzt die komprimierte Datei?

Für Idee 1 benötigen wir nur die Länge im Header.
Für Idee 2 benötigen zusätzlich ein dictionary.

Wie spezifizieren Wir die Korrektheit?

Festlegung von Pre und Postconditions und Prüfung nach jedem Schritt ob diese verletzt wurden.
Invariante?

Welche Testfälle sind sinnvoll?

Zieldatei mit Quelldatei vergleichen.
unerwartetes Format der Quelldatei erkennen.
Kompressionsrate auf Erwartungshaltung prüfen.

Mit welcher Laufzeit rechnen wir?

Die Laufzeit der Komprimierung ist linear, da wir nur einmalig jedes Zeichen der Datei lesen. Auch bei der Dekomprimierung erwarten wir eine lineare Laufzeit.

Aufgabe 3.1 Bei Laufzeitabschätzung von Algorithmen geht man davon aus, dass alle "eingebauten" Konstrukte (wie Addition, Zuweisung usw.) eine konstante Zeit benötigen. Die Konstante selbst kennen wir nicht, sie ist typischerweise auch stark von der Hardware abhängig. Die Laufzeit von Prozeduren hängt dagegen i.a. von den Argumenten ab.

Op

1. Bestimmen Sie Anzahl der Operationen, die der folgenden Algorithmus ausführt:

Algorithm 1 Quersumme von $A[1..n]$

```
x = 0
for i = 1 to n do
    x = x + A[i]
end for
return x
```

Op $= n$

da for-Schleife von $1 \to n$

2. Bestimmen Sie Anzahl der Operationen, die der folgenden Algorithmus ausführt:

Algorithm 2 Alg. 1

```
for i = 1 to n do
    A[i] = i
end for
for i = 1 to n do
    C[i] = 0
    for j = n downto 1 do
        if A[j] > C[i] then
            C[i] = A[j]
        end if
    end for
end for
return C
```

Op $= n + n^2$

da 1x einfache for-Schleife

+ verschachtelte for-Schleife

3. Bestimmen Sie Anzahl der Operationen, die der folgenden Algorithmus ausführt:

Algorithm 3 Matrixmultiplikation

```
for i = 1 to n do
    for j = 1 to n do
        C[i][j] = 0
        for k = 1 to n do
            C[i][j] = A[i][k] * B[k][j]
        end for
    end for
end for
return C
```

Op $= n^3$

s.o.

4. Bestimmen Sie Anzahl der Operationen, die der folgenden Algorithmus ausführt:

Algorithm 4 Alg. Beispiel

```
for i = 1 to n do
    for j = i downto 1 do
        x = x + A[i][j]
    end for
end for
return x
```

$Op = \frac{n(n+1)}{2}$ "kleiner Gauß"

addiere alle Zahlen von 1 - n

Aufgabe 3.2 Die O-Notation.

1. Zeigen Sie: $15n^2 \in O(n^3)$.

Es gilt $15n^2 \leq c \cdot n^3$ für $c = 15$ und $n \in /$

$\Rightarrow 15n^2 \leq 15n^3 \Rightarrow 15n^2 \in O(n^3)$

2. Zeigen Sie: $\frac{1}{2}n^3 \notin O(n^2)$.

IA: für $n = 2$

$1/2 \cdot 2^3 = 4{,}5 > 4 = 2^2$

IB: für ein beliebiges aber festes $n \in \mathbb{N}$ mit $n \geq 2$ gilt $\frac{1}{2}n^3 \notin O(n^2)$

IS: zz $n \rightarrow n+1$:

$\frac{1}{2}(n+1)^3 > (n+1)^2$

$= \frac{1}{2}n^3 + \frac{3}{2}n^2 + \frac{3}{2}n + \frac{1}{2} > n^2 + 2n + 1$ | Abschätzung

$= \frac{1}{2}n^3 > n^2$, somit gilt die Behauptung

3. Betrachte $g(n) = 2n^2 + 3$.

a) Geben Sie eine Funktion $f_1 : \mathbb{N} \to \mathbb{N}$ an, für die $f_1 \in O(g)$ und $f_1(n) < g(n)$ für alle n ab einem n_0 gilt.

b) Geben Sie eine Funktion $f_2 : \mathbb{N} \to \mathbb{N}$ an, für die $f_2 \in O(g)$ und $f_2(n) > g(n)$ für alle n ab einem n_0 gilt.

(Beide Funktionen f_1 und f_2 liegen also in $O(g)$, aber $f_1(n)$ ist stets kleiner und $f_1(n)$ ist stets größer als $g(n)$.)

a) $f_1(n) = n$

b) nach Def. gilt: $O(g(n)) = \{f \mid \exists c \in \mathbb{R}^+ : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$

↳ ab diesem Wert gilt es für alle folgenden n

somit gibt es keine Funktion die gilt $f_2 \in O(g) \wedge f_2(n) > g(n)$ für $\forall n \geq n_0$

4. Wir betrachten Polynome mit natürlichzahligen Koeffizienten, d.h. Funktionen der Form $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ mit $a_i \in \mathbb{N}$ und wenn $a_k \neq 0$. Das Polynom hat dann den Grad k.

Zeigen Sie: Für zwei Polynome f und g mit gleichem Grad gilt $f \in \Theta(g)$.

$f(n) \in c_k O(n^k) + c_{k-1} O(n^{k-1}) + \ldots + c_1 O(n) + c_0 \overset{*}{\Longleftrightarrow} f(n) \in O(n^k) + O(n^{k-1}) + \ldots + O(n)$

*siehe PDB VL 3 Speicherung v. Graphen, Wege & Kreise

Sequenzenregel

$\Longleftrightarrow f(n) \in O(\max(n^k, n^{k-1}, \ldots, n) \in O(n^k)$ das gilt für alle Polinome, somit auch f und g

$O(f(n)) = O(g(n)) \Rightarrow f(n) \in \Theta(g(n))$

5. Zeigen Sie: Sei $f(n) := \sum_{i=0}^{n} 2^i$. Es gilt $f \in O(2^n)$

vgl. geometrische Reihe

$$
\begin{aligned}
f(n) := \ & \textstyle\sum_{i=0}^{n} 2^i \\
= \ & \frac{2^{(n+1)} - 1}{2 - 1} \\
= \ & 2^{n+1} - 1 \\
= \ & 2^n \cdot \underbrace{2^1 - 1}_{c}
\end{aligned}
$$

c ist Konstante

$$
\begin{aligned}
= \ & 2^n \cdot c \\
= \ & O(2^n)
\end{aligned}
$$

somit gilt $f \in O(2^n)$