7. NOVEMBER 2013

DOKUMENTATION

ZU AUFGABENBLATT 03 AUS DER VORLESUNGSREIHE „ALGORITHMEN UND DATENSTRUKTUREN“

HAW HAMBURG

DOKUMENTATION

ZU AUFGABENBLATT 03 AUS DER VORLESUNGSREIHE „ALGORITHMEN UND DATENSTRUKTUREN“

ÜBUNGSAUFGABE 3.1

TEILAUFGABE 1

Bestimmen Sie Anzahl der Operationen, die der folgende Algorithmus ausführt:

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

```
1: x = 0
2: for i = 1 to n do
3:     x = x + A[i]
4: end for
5: return x
```

In Zeile 1 wird eine Zuweisung gemacht. (1)
In Zeile 2 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)
In Zeile 3 wird eine Addition und eine Zuweisung durchgeführt. Allerdings passiert das in einem Assemblertakt, sodass wir dies nur als eine Operation werten. Trotzdem wird diese Operation n mal durchgeführt. (n)
In Zeile 5 wird eine Rückgabe gemacht. (1)

Zusammen beträgt die Laufzeit T(n)=(2n + 2)

TEILAUFGABE 2

Bestimmen Sie Anzahl der Operationen, die der folgende Algorithmus ausführt:

Algorithm 2 Alg. 1

```
1: for i = 1 to n do
2:     A[i] = i
3: end for
4: for i = 1 to n do
5:     C[i] = 0
6:     for j = n downto 1 do
7:         if A[j] > C[i] then
8:             C[i] = A[j]
9:         end if
10:    end for
11: end for
12: return C
```

In Zeile 1 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)
In Zeile 2 wird eine Zuweisung gemacht. Diese findet n mal statt. (n)
In Zeile 4 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)
In Zeile 5 wird eine Zuweisung gemacht. Diese findet n mal statt. (n)
In Zeile 6 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. Dies wird n mal gemacht (n^2)
In Zeile 7 wird ein Vergleich durchgeführt. Dies findet n^2 mal statt. (n^2)
In Zeile 8 findet eine Zuweisung statt. Diese kann maximal n^2 mal statt finden. (n^2)
In Zeile12 findet eine Rückgabe statt. (1)

Zusammen beträgt die Laufzeit $T(n)=(3n^2 + 4n + 1)$

TEILAUFGABE 3

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

Algorithm 3 Matrixmultiplikation

```
1: for i = 1 to n do
2:     for j = 1 to n do
3:         C[i][j] = 0
4:         for k = 1 to n do
5:             C[i][j] = A[i][k] * B[k][j]
6:         end for
7:     end for
8: end for
9: return C
```

In Zeile 1 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)
In Zeile 2 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. Dies wird n mal gemacht (n^2)
In Zeile 3 findet eine Zuweisung statt. Dieses passiert n^2 mal. (n^2)
In Zeile 4 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. Dies wird n^2 mal gemacht (n^3)
In Zeile 5 findet eine Multiplikation und eine Zuweisung statt. Da dies in einem Assemblertakt passiert, werten wir dies als eine Operation. Das findet n^3 mal statt. (n^3)
In Zeile 9 findet eine Rückgabe statt. (1)

Zusammen beträgt die Laufzeit $T(n)=(2n^3 + 2n^2 + n + 1)$

TEILAUFGABE 4

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

Algorithm 4 Alg. Beispiel

```
1: for i = 1 to n do
2:     for j = 1 downto i do
3:         x = x + A[i][j]
4:     end for
5: end for
6: return x
```

In Zeile 1 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)
In Zeile 2 wird erst eine zuweisung gemacht und in jedem darauf folgenden Durchlauf I inkrementiert. Dies findet n mal statt, allerdings veringert sich die Menge der Inkrementationen bei jedem Durchlauf um 1. Daher lässt sich der Aufwand mit der Gaußschen Summenformel beschreiben. $((n^2 + n) / 2)$
In Zeile 3 findet eine Addition und eine Zuweisung statt. Da dies in einem Prozessortakt statt findet werten wir dies als eine Operation. Diese findet $(n^2 + n) / 2$ mal statt. $((n^2 + n) / 2)$
In Zeile 6 findet eine Rückgabe statt. (1)

Zusammen beträgt die Laufzeit $T(n)=(n^2 + 2n + 1)$

ÜBUNGSAUFGABE 2

TEILAUFGABE 1

Implementieren Sie folgendes Verfahren zum Potenzieren von x :

Algorithm 5 $exp(x, k)$; berechnet x^k, $k \geq 0$

```
1: r = 1
2: for i = 1 to k do
3:     r = r * x
4: end for
5: return r
```

Das Verfahren wurde implementiert. Die Implementierung befindet sich im Anhang zu dieser Dokumentation.

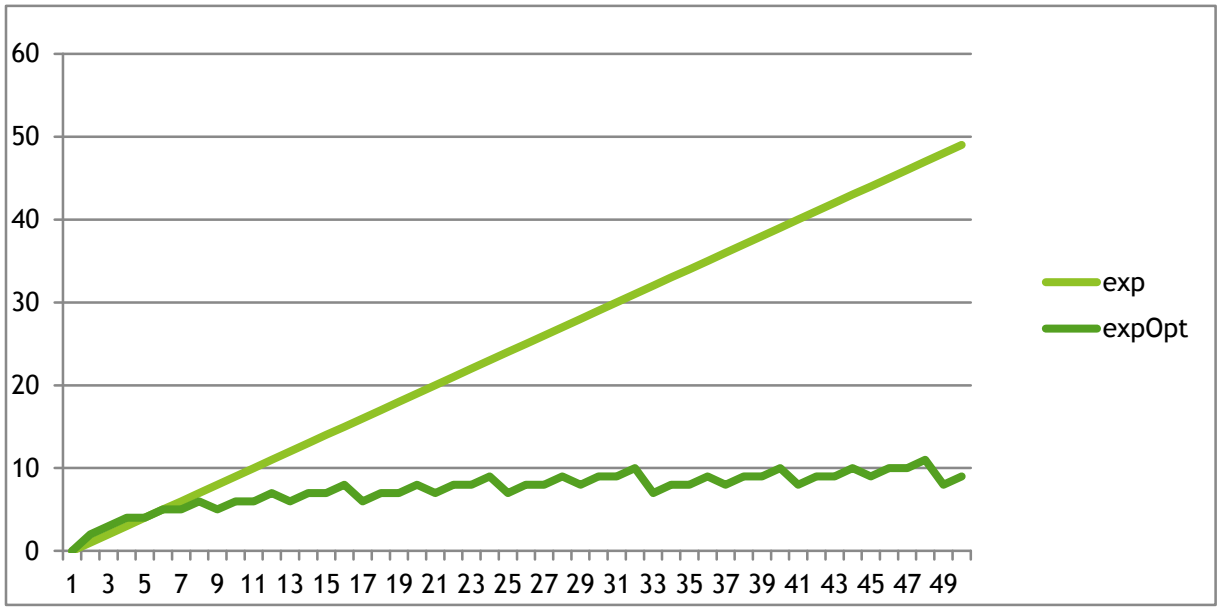
TEILAUFGABE 2

Nutzen Sie folgende Gleichungen aus, um Potenzen schneller zu berechnen:

$$x^0 = 0$$
$$x^{2n} = (x^{2n})^2$$
$$x^{2n+1} = x * (x^{2n})^2$$

TEILAUFGABE 3

Vergleichen Sie die Laufzeit der Algorithmen in Abhängigkeit von k! Machen Sie geeignete Experimente und stellen Sie die beiden Laufzeiten graphisch dar.



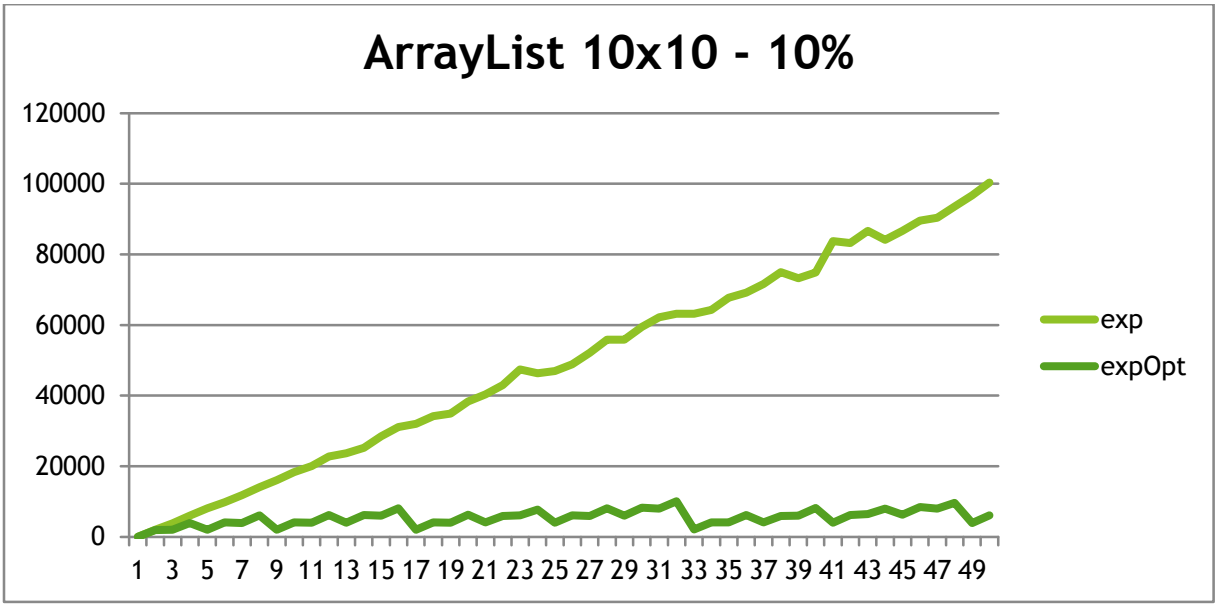
Für welche Werte von x und k kommen die Implementation an ihre Grenzen?

Die Obere Grenze von expOpt liegt bei k = 30 für n = 2.
Die Obere Grenze von exp liegt ebenso bei k = 30 für n = 2.
Die Obere Grenze von expOpt liegt bei n = 46340 für k = 2.
Die Obere Grenze von exp liegt bei n = 46340 für k = 2.

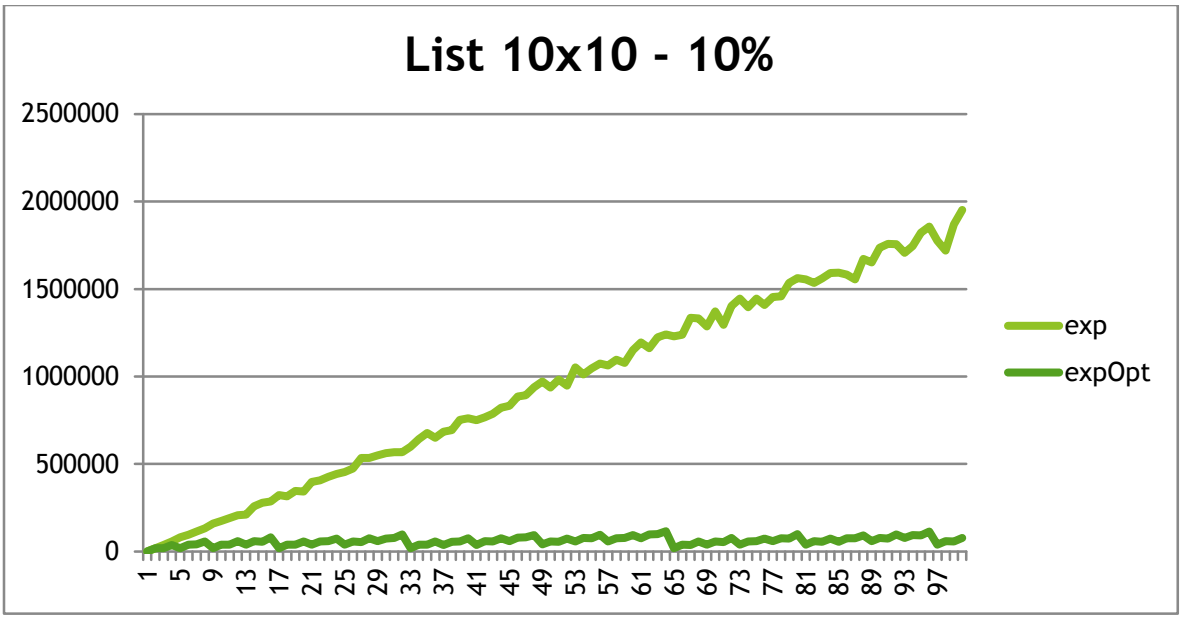
Daraus ist zu entnehmen, dass die optimierte Version von expOpt keine höheren Zahlen zur Berechnung benötigt als exp und so dieselben Grenzen hat wie exp, dafür aber wesentlich schneller ist.

TEILAUFGABE 4

Beide Implementation können auch dann eingesetzt werden, wenn x keine Zahl, sondern eine Matrix ist. Vergleichen Sie die Laufzeiten, indem Sie wieder quadratische (geeignet große) Zufallsmatrizen erzeugen und diese Potenzieren.



Es ist zu erkennen, dass die optimierte Exponentialfunktion für große Exponenten wesentlich effizienter ist als die gewöhnliche. Da die Dereferenzierungen der expOpt nicht über 15.000 steigen während die exp nahezu linear ansteigt.



Hier ist zu erkennen, dass auch bei der List Implementation die optimierte Exponentialfunktion wesentlich effizienter ist.

ÜBUNGSAUFGABE 3

TEILAUFGABE 1

Beweisen sie:

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

Zeigen durch:

$$f(n) \in O(g(n)) \Leftrightarrow \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) < \infty$$

Einsetzen:

$$\lim_{n\to\infty}\frac{15n^2}{n^3} = \lim_{n\to\infty}\frac{15}{n} = 0$$

Da 0 offensichtlich kleiner als Unendlich ist, ist die Aussage wahr.

TEILAUFGABE 2

Beweisen sie:

$$\frac{1}{2n^3} \notin O(n^2)$$

Zeigen durch:

$$f(n) \in O(g(n)) \Leftrightarrow \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) < \infty$$

Einsetzen:

$$\lim_{n\to\infty}\frac{1/2n^3}{n^2} = \lim_{n\to\infty} 1/2n = \infty$$

Da das Ergebnis nicht < unendlich ist, ist die Aussage wahr.

TEILAUFGABE 3

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

A)

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

Wir wählen $f(n) = n$

Da der höchste Exponent von f kleiner als der von g ist, ist es offensichtlich, dass

$$f(n) \in O(g)$$

Ab einem n_0 von 0 gilt, dass

$$f(n) < g(n)$$

Wie durch die Konstante +3 in $g(n)$ zu sehen ist.

B)

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

Wir wählen $f(n) = 2n^2 + 4$

Da f und g bis auf die Konstanten identisch sind gilt

$$f(n) \in O(g)$$

Ab einem n_0 von 0 gilt, dass

$$f(n) > g(n)$$

Wie durch die größere Konstante +4 in $f(n)$ und sonstige Gleichheit von f und g zu sehen ist.

TEILAUFGABE 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* $a_i \in N$ *und wenn* $a \neq 0$. *Das Polynom hat dann den Grad k.*
Zeigen Sie: Für zwei Polynome f und g mit gleichem Grad gilt $f \in \Theta(g)$.

Bei der Komplexität sind Konstanten per Definition in der Landau Notation von vernachlässigbarer Relevanz, und somit sind hier einzig die Polynome von Relevanz. Sobald der höchste Koeffizient zweier Polynome identisch ist sind die Funktionen aufgrund der hiesigen Form in derselben Komplexitätsklasse.

Weil f und g denselben Grad haben, gilt

$$k_g = k_f$$

Somit ist offensichtlich

$$f \in O(g) \text{ und } g \in O(f)$$

TEILAUFGABE 5

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

Da bei einer Summation von Polynomen mit natürlichzahligen Koeffizienten nur derjenige für die Komplexität der Funktion f relevant ist, der den größten Exponenten hat, können wir sagen, dass

$$K(\sum_{i=0}^{n} 2^i) = K(2^n)$$

Also reicht es zu zeigen, dass

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

Was offensichtlich wahr ist.