Hilfsunterlagen

Algorithmen Und Datenstrukturen

Kontakt sanssecours@f-m.fm
Version 8
Datum 5. Juni 2013

Inhaltsverzeichnis

Zusammenfassung

Dieser Text ist mit der Absicht entstanden zu helfen. Ich habe ihn somit natürlich nicht aus Absicht mit Fehlern gespickt. Trotzdem ist es doch sehr wahrscheinlich, dass er Fehler enthält. Ich bitte das zu entschuldigen, und möchte damit den Hinweis geben, dass für jeglichen Inhalt dieses Textes ***absolut kein Gewähr auf Richtigkeit*** gegeben wird.

Solltet ihr Fehler im Text finden wäre es sehr nett wenn ihr mir eine e-Mail schreibt, damit ich sie ausbessern kann.

1 · Laufzeiten

1.1 Beweisen oder Wiederlegen des asymptotischen Wachstums von Funktionen

1.1.1 Umformung

Der erste Schritt am Anfang der Analyse einer Funktion wie z.B.

$$f(n) = \begin{cases} 2 \cdot n^2 + \frac{n^2}{2} + 10 & \text{wenn } n < 50 \\ 0.5 \cdot n^{\frac{1}{2}} & \text{wenn } n \text{ ungerade und } n \geq 50 \\ \frac{1}{2} \cdot n + \frac{n^2}{n} \cdot \log\left(n^2\right) - \frac{1}{10} & \text{sonst} \end{cases} \quad (1)$$

besteht darin zu bestimmen welche Teile der Funktion für unendlich viele n gelten und welche Glieder wiederum ab eine bestimmten n nicht mehr vorkommen, also nur für endlich viele n gelten.

In Funktion 1 kann man den ersten Zweig vernachlässigen da er ab $n \geq 50$ nicht mehr gilt. Damit man diesen Teil auch wirklich nicht beachten muss ist n_0 auf einen Wert ≥ 50 zu setzen. Der nächste Schritt besteht darin die Funktion ein wenig zu vereinfachen. Dazu einmal ein paar einfache mathematische Umformungen die hier recht nützlich sind:

$$a^{\frac{1}{b}} = \sqrt[b]{a} \quad a^{-b} = \frac{1}{a^b} \quad \frac{a^b}{a^c} = a^{b-c} \quad a^b \cdot a^c = a^{b+c} \quad \log\left(a^b\right) = b \cdot \log\left(a\right)$$

Nach der Umformung der Funktion ohne den ersten Zweig, den wir ohnehin für den Beweis/Widerlegung vernachlässigen wollen bekommen wir:

Abbildung 1: Vereinfachte Funktion für $n \geq 50$

$$f(n) = \begin{cases} 0.5 \cdot \sqrt{n} & \text{wenn } n \text{ ungerade} \\ \frac{1}{2} \cdot n + 2n \cdot \log(n) - \frac{1}{10} & \text{sonst} \end{cases}$$

1.1.2 Vereinfachte Terme mit asymptotisch gleicher Laufzeit in Theta-Notation

Als nächsten Schritt könne wir jetzt Terme für die Zweige unserer Funktionen bestimmen die die „gleiche" Laufzeit wie die Terme unserer ursprünglichen Funktion besitzen. Das ist vor allem praktisch wenn wir angeben wollen wie die unteren und oberen Schranken unserer Funktion aussehen. Dazu bestimmen wir in jedem Zweig den „größten" und somit den für die Laufzeit bestimmenden Teil.

In der zweiten Zweig unserer Funktion ist der bestimmende Teil der Term $\sqrt{n}$. Die Konstante können wir streichen, da sie nichts am asymptotischen Wachstum der Funktion ändert.

Im dritten Zweig könne wir erkennen, dass „$2n \cdot \log n$" den für das Wachstum wichtigen Teil darstellt. Die Konstante spielt keine Rolle für das Wachstum. Somit kommen wir hier auf den Term „$n \log n$" der asymptotisch gleiches Wachstum wie unsere Ausgangsterm aufweist. Die gesamte vereinfachte Funktion, die für ein $n \geq 50$ das gleiche Wachstum wie die Ausgangsfunktion aufweist ist in Abbildung 2 zu sehen.

Abbildung 2: Funktion mit asymptotisch gleichem Wachstum wie die Ausgangsfunktion ab $n \geq 50$

$$f(n) = \begin{cases} \sqrt{n} & \text{wenn } n \text{ ungerade} \\ n \log n & \text{sonst} \end{cases}$$

Aus der vereinfachte Funktion können wir gleich erkennen, dass alle Funktionen die eine Laufzeit in $\theta\left(n \log n\right)$ besitzen oder schneller wachsen eine obere Schranke für unsere Funktion darstellen. Alle Funktionen die eine Laufzeit in $\theta\left(\sqrt{n}\right)$ aufweisen oder langsamer wachsen stellen eine untere Schranke dar. Die Funktion $g(n) = n$ hingegen würde weder eine untere noch eine obere Schranke darstellen, da sie zwar ein schnelleres Wachstum als der erste Zweig unserer Funktion aufweist, aber eben auch langsamer als der zweite Zweig wächst.

1.1.3 Beweis

Wir wollen nun beweisen, dass $\sqrt{n}$ eine untere Schranke für unsere Funktion darstellt also $f(n) = \Omega\left(\sqrt{n}\right)$ gilt. Dazu schreiben wir als erstes laut Definition an:

$$f(n) = \Omega\left(\sqrt{n}\right) \quad \Rightarrow \quad \exists c, n > 0 \quad \forall n \geq n_0 : 0 \leq c \cdot \sqrt{n} \leq f(n)$$

Wir müssen diese Bedingung nun für beide Zweige beweisen und passende Werte für c und n_0 finden, wobei wir von n_0 schon wissen, dass wir einen Wert ≥ 50 nehmen sollten, da wir ansonsten auch den ersten Zweig unserer Funktion beachten müssten, der nur für $n < 50$ gilt. Wir sehen uns nun die beiden anderen Zweige in der vereinfachten Form aus Abbildung 1 an.

$$\exists c, n > 0 \quad \forall n \geq n_0 : 0 \leq c \cdot \sqrt{n} \leq 0.5 \cdot \sqrt{n}$$

Hier erkennt man direkt aus dem angeschriebenen Werten, dass diese Gleichung für $c \leq 0.5$ gilt. Kommen wir nun zum interessanteren zweiten Zweig:

$$\exists c, n > 0 \quad \forall n \geq n_0 : 0 \leq c \cdot \sqrt{n} \leq \frac{1}{2} \cdot n + 2n \cdot \log(n) - \frac{1}{10}$$

Wir dividieren durch $\sqrt{n}$ und kommen damit auf:

$$0 \leq c \leq \frac{1}{2}\sqrt{n} + 2\sqrt{n} \cdot \log(n) - \frac{1}{10\sqrt{n}}$$

Der rechte Teil wächst somit mit steigendem n. Wir können also ein c finden dass ab einem bestimmten Wert n_0 immer kleiner bleibt als unsere Funktion. Setzten wir nun z.B. für $n = 50$ ein ergibt sich ein Wert von ca. 28. Somit können wir als Konstanten die für beide Zweige der Funktion gelten z.B. $n_0 = 50$ und $c = 0.5$ wählen.

1.1.4 Widerlegung

Wir wollen nun widerlegen, dass unsere Funktion ein Wachstum von $\theta\left(n \log n\right)$ besitzt. Aus Abbildung 2 können wir erkennen, dass das Wachstum für einen der Zweige $\theta\left(n \log n\right)$ ist. Wir nehmen also den anderen Teil um die Behauptung zu widerlegen und schreiben laut Definition:

$$\exists c_1, c_2, n > 0 \quad \forall n \geq n_0 \ : \ 0 \leq c_1 \cdot n \log n \leq 0.5 \cdot \sqrt{n} \leq c_2 \cdot n \log n$$

Wir dividieren ähnlich wie zum Beweis von vorher durch $n \log n$.

$$0 \leq c_1 \leq \frac{0.5}{\sqrt{n} \log n} \leq c_2$$

Man kann erkennen, dass die Funktion gegen 0 konvergiert. Der Wert der Funktion wird also immer kleiner je größer n wird. Mit dieser Erkenntnis können wir ein c_2 finden, dass die Funktion nach oben begrenzt und sehen somit dass $f(n) = O\left(n \log n\right)$ für diesen Zweig gilt. Wir können aber kein $c_1 > 0$ angeben das ab einem bestimmten positiven Wert n immer kleiner als der Wert der Funktion ist, da wir immer ein größeres n finden können ab dem ein bestimmter Wert größer 0 unterschritten wird.

Somit hat die Funktion keine untere Schranke $n \log n$ $\left(f(n \log n) \neq \Omega(n)\right)$ und damit auch nicht die „gleiche Laufzeit" wie $n \log n \Rightarrow f(n) \neq \theta(n \log n)$.

1.2 Kreuzerlbeispiele Laufzeiten

1.2.1 Test vom 26.11.2010

Tabelle 1: Beispiel 1b) der Gruppe A

Annahme	Folgerung: $f(n)$ ist			
	$\Omega\left(g(n)\right)$	$O\left(g(n)\right)$	$\Omega(h(n))$	$O(h(n))$
$f(n) = O\left(g(n) + f(n)\right) \wedge g(n) = \Omega(h(n))$				

In Tabelle 1 darf kein Feld angekreuzt werden, da $g(n)$ sowohl eine untere als auch eine obere Schranke von $f(n)$ sein kann.

1.3 Erfassung von Algorithmen-Laufzeiten

1.3.1 Allgemeine Hinweise

Verschachtelte Schleifen

Bei verschachtelten Schleifen werden die Laufzeiten multipliziert sofern diese unabhängig sind. Das heißt es werden in der inneren Schleife keine Werte verändert die die Laufvariable der äußeren Schleife ändern und umgekehrt. Folgender Code zeigt zwei unabhängige Schleifen:

```
j = n
for i = 1...log(n) {
    for j > 1 {
        c = 10 + 1
        j = j - 1
    }
    j = n
}
```

Die Laufzeiten der äußeren Schleife $\theta\left(\log\left(n\right)\right)$ und der inneren Schleife $\theta\left(n\right)$ zusammen ergibt eine Gesamtlaufzeit von $\theta\left(n \cdot \log\left(n\right)\right)$.

Zwei voneinander abhängige Schleifen werden z.B. in Abschnitt 1.3.2 gezeigt.

Zurücksetzten von Variablen

Der folgende Code zeigt zwei verschachtelte Schleifen.

```
i = n
j = n
for i > 0 {
    i = i - 1
    for j > 0 {
        c = c + 1
        j = j - 1
    }
}
```

Auf den ersten Blick könnte man vermuten, dass sich hier eine Laufzeit von $\theta\left(n^2\right)$ ergibt. Da `j` aber nur einmal am Anfang auf `n` gesetzt wird, wird die innere Schleife zwar `n` mal durchgeführt, das geschieht allerdings nur im ersten Durchlauf der äußeren Schleife. Somit kann man die Laufzeit der inneren Schleife vernachlässigen und kommt somit auf eine Laufzeit von $\theta\left(n\right)$. Der folgende Code zeigt die Änderung die vorgenommen werden müsste damit wir eine Laufzeit von $\theta\left(n^2\right)$ erhalten:

```
i = n
j = n
for i > 0 {
    i = i - 1
```

```
    for j > 0 {
        c = c + 1
        j = j - 1
    }
    j=n
}
```

Änderungen von Werten

Bei vielen Aufgaben stellt sich die Frage wie oft denn z.B. ein bestimmter Term von einer Variable abgezogen oder addiert werden kann bis die Variable einen anderen Wert (z.B 0,n,...) erreicht. Grundsätzlich gibt es ein paar verschiedene Varianten die üblicherweise auftreten:

- eine Konstante wird abgezogen/addiert z.B. $a = a - 1, a = a + 1$
- Multiplikation mit Konstante oder Division durch eine Konstante
- ein Term abhängig von n wird abgezogen/addiert z.B. $a = a + \sqrt{n}, a = a - log(n)$

Ein Beispiel für den erste Punkt wäre:

```
i = 0
für i < log n {
    i=i+1
}
```

Die äußere Schleife läuft hier $\log(n)$ mal bis wir den Wert von $\log(n)$ erreichen und kommt somit auch auf eine Laufzeit von $\theta\left(log(n)\right)$. Nach dem gleichen Schema ergibt sich bei nachfolgendem Beispiel eine Laufzeit in $\theta\left(n\sqrt{n}\right)$:

```
a = n * sqrt (n)
solange a > 0 {
    a = a - 2
}
```

Die wiederholte Multiplikation mit einem Wert bedeutet üblicherweise, dass sich eine exponentielle Laufzeit ergibt. Die Umkehrung der Operation, also die Division durch einen bestimmten Wert, weist folglich meist auf eine logarithmische Laufzeit hin.

1.3.2 Beispiele

Test vom 16.11.2007

Die innerste solange-Schleife des in Abbildung 3 zu sehenden Codestücks wird auf Grund dessen, dass die Variable l nicht mehr rückgesetzt wird, nur ein einziges mal, im allerersten Durchlauf der beiden äußeren Schleifen durchgeführt. Damit kann man die Laufzeit der inneren Schleife vernachlässigen, da sie für alle anderen Durchläufe einen konstanten zeitlichen Aufwand besitzt.

Abbildung 3: Beispiel 1.c) der Gruppe A

$m = 3 \cdot n^2$; $l = \sqrt{n}$;
für $j = 1, \ldots, m$ **{**
  $k = \lfloor \frac{j}{2} \rfloor$;
  für $i = k, \ldots, 1$ **{**
    solange $l \geq 1$ **{**
      $c = c + b$; $l = \lfloor \frac{l}{2} \rfloor$;
    }
  }
}

Abbildung 4: Beispiel 1.c) der Gruppe B

$b = \sqrt{n}$; $a = 3 \cdot n^3$;
für $j = 1, \ldots, a$ **{**
  $c = \lfloor \frac{j}{2} \rfloor$;
  für $i = c, \ldots, 1$ **{**
    solange $b \geq 1$ **{**
      $l = l + m$; $b = \lfloor \frac{b}{2} \rfloor$;
    }
  }
}

Die äußerste Schleife läuft von 1 bis $3n^2$ und besitzt somit ein quadratischen Laufzeit. Die zweite für-Schleif läuft jeweils bis zu $\frac{j}{2}$. Das heißt die Laufzeit beträgt $\frac{1}{2}, \frac{2}{2}, \frac{3}{2}, \frac{4}{2} \ldots \frac{3n^2}{2}$. Lassen wir die Konstante weg ($3n^2 \Rightarrow n^2$) und ziehen den Term $\frac{1}{2}$ nach vorne erhalten wir $\frac{1}{2}\left(1, 2, 3, 4 \ldots n^2\right)$. Mit der Summenformel von Gauß ergibt sich also $\frac{1}{2} \cdot (n^2 + 1) \cdot \frac{n^2}{2}$ und somit insgesamt eine Laufzeit von $\theta\left(n^4\right)$ für das gesamte Codestück.

Die Analyse des Codes von Abbildung 4 führt analog zu dem vorhergehenden Beispiel auf eine Laufzeit von $\theta\left(n^6\right)$.

Test vom 25.4.2008

Wir analysieren den in Abbildung 5 dargestellten Code. Die beiden inneren Schleifen sind abhängig voneinander, da in der äußeren Schleife j auf ein Drittel verringert wird. Die innere Schleife läuft also $n, \frac{n}{3}, \frac{n}{9}, \frac{n}{27} \ldots$ solange $j \geq 1$. Die Summe über diese Terme liegt zwischen n und $2n$. Die inneren Schleifen ergeben also einen Aufwand von $\theta(n)$.

Abbildung 5: Beispiel 1.c) der Gruppe A

```
i = 1; c = 0;
solange i < n^2 {
  j = n;
  solange j ≥ 1 {
    für k = 1, ..., j {
      c = c + k;
    }
    j = ⌊j/3⌋;
  }
  i = i + 1;
}
```

Abbildung 6: Beispiel 1.c) der Gruppe B

```
i = 1; c = 0;
für i = 1, ..., n {
  j = n;
  solange j ≥ 1 {
    für k = 1, ..., j {
      c = c + k;
    }
    j = ⌊j/4⌋;
  }
  i = i + 1;
}
```

Die äußere Schleife läuft n^2 mal. Somit ergibt sich ein Gesamt-Aufwand von $\theta\left(n^3\right)$.

Nach analoger Analyse zum vorhergehenden Beispiel ergibt sich für den Code von Abbildung 6 eine Laufzeit von $\theta\left(n^2\right)$.

Test vom 7.11.2008

Abbildung 7: Beispiel 1.c) A der Gruppe A

```
a = n^2;
j = log n;
b = 2^n;
solange a > 0 {
  b = ∛b;
  für i = j, j−1, ..., 1 {
    b = 2b + i;
  }
  a = a − n;
}
```

Der zu analysierende Code ist in Abbildung 7 dargestellt. In der äußeren Schleife wird von a immer wieder n abgezogen. Das können wir n mal machen bis $a \leq 0$ ist. Die äußere Schleife hat somit eine Laufzeit in $\theta(n)$. In der inneren Schleife wird von der Variable j, die am Anfang den

Wert $log(n)$ besitzt solange 1 abgezogen bis wir auf einen Wert von 1 kommen. Dieser Schritt kann $log(n)$ mal durchgeführt werden. Insgesamt ergibt sich also eine Laufzeit von $\theta\left(log(n) \cdot n\right)$

Test vom 15.04.2011

Abbildung 8: Beispiel 1.a) B der Gruppe A

$a = 1;$
$b = 1;$
für $i = 1, \ldots, n$ **{**
  solange $a < n$ **{**
    $a = a + 1;$
    falls $a < \log n$ **dann {**
      $b = 2 \cdot b;$
    }
  }
}
$a = b^2;$
für $k = 1, \ldots, a$ **{**
  $b = 2 \cdot k + b;$
}

Der in Abbildung 8 gezeigte Code soll analysiert werden. Die äußerste „für"-Schleife wird n mal durchgeführt. Da die innere „so-lange"-Schleife nur beim ersten dieser n Durchgänge ausgeführt wird – a wird nachdem es den Wert n erreicht hat nicht mehr auf einen kleineren Wert gesetzt – kann die Laufzeit der inneren Schleife von $\log(n)$ vernachlässigt werden. Insgesamt ergibt sich für die erste „für"-Schleife also eine Laufzeit von $\theta(n)$. Diese Tatsache kann man auch erkennen wenn man sich die einzelnen Laufzeiten der äußersten Schleife ansieht:

- 1. Durchgang: $\theta\left(\log(n)\right)$
- 2. Durchgang: $\theta(1)$
- 3. Durchgang: $\theta(1)$
- …
- n. Durchgang: $\theta(1)$

Zusammengezählt kommen wir damit auf eine Laufzeit in Theta-Notation von

$$\theta\left(\log(n) + (n-1)\cdot 1\right) \equiv \theta\left(n\right)$$

Der Wert von b nach den ersten beiden Schleifen beträgt $2^{\log(n)}$. Dieser Wert ergibt sich dadurch, dass in der inneren Schleife b $\log(n)$ mal verdoppelt wurde. Dabei gehen wir hierbei davon aus, das es sich bei $\log(n)$ um den Zweierlogarithmus handelt. Dadurch heben sich der Logarithmus und die Zweierpotenz (Umkehrfunktion des Zweierlogarithmus) auf und der Wert von b beträgt somit n.

Der neue Wert der in a gespeichert wird ist also n^2 ($a = b^2$). Damit kommen wir für die letzte „für"-Schleife auf eine Laufzeit in Theta-Notation von $\theta(n^2)$. Die Gesamtlaufzeit beträgt somit:

$$\theta\left(n + n^2\right) \equiv \theta\left(n^2\right)$$

2 · Sortieralgorithmen

2.1 Test vom 25.4.2008

Abbildung 9: Beispiel 2.c) von Gruppe A

```
Sortiere(var A, n)
  wiederhole {
    vertauscht = falsch;
    für i = 1, 2, ..., n − 1 {
      falls A[i + 1] ≥ A[i] dann {
        vertausche A[i + 1] und A[i];
        vertauscht = wahr;
      }
    }
  } bis vertauscht == falsch;
```

Der in Abbildung 9 gezeigte Sortieralgorithmus wird auf ein Feld $A = (A[1], \ldots, A[n])$ angewendet. Er sortiert nicht alle Eingabefolgen korrekt. Wo liegt das Problem und wie kann man es beheben?

Es ist möglich, dass der Algorithmus nicht terminiert. Eine Folge die z.B. zu einer Endlosschleife führt ist [1,1]. Hier werden als erstes die beiden

Elemente vertauscht da $1 \geq 1$ ist und dann die Variable `vertauscht` auf wahr gesetzt. Somit wird die wiederhole-Schleife wieder ausgeführt. In dieser werden wieder die beiden Elemente vertauscht und `vertauscht` auf wahr gesetzt…

Damit der Algorithmus terminiert muss „$A[i+1] \geq A[i]$" durch „$A[i+1] > A[i]$" ersetzt werden.

3 · Hashverfahren

3.1 Test vom 25.4.2008

Abbildung 10: Beispiel 1.a) von Gruppe B

Gegeben seien zwei **unabhängige** Hashtabellen mit Tabellengröße $m = 7$ in denen bereits Schlüssel eingefügt wurden. Als Hashfunktionen sollen

$$h_1(k) = (k \ mod \ 5) + 1$$
$$h_2(k) = (k \ mod \ 7)$$

und zur Kollisionsbehandlung Double-Hashing mit der Verbesserung nach Brent verwendet werden.

- Fügen Sie in die folgende Hashtabelle den Schlüssel 15 ein.

Schlüssel \ Index	0	1	2	3	4	5	6
k		10	1	12	8		

- Fügen Sie in die folgende Hashtabelle den Schlüssel 8 ein.

Schlüssel \ Index	0	1	2	3	4	5	6
k		5		12	10	14	

- Sind die oben angeführten Hashfunktionen $h_1(k)$ bzw. $h_2(k)$ für eine beliebige Hashtabelle der Größe 7 eine gute Wahl? Begründen Sie ihre Antwort.

Angabe Die Angabe zum Beispiel ist in Abbildung 10 zu sehen.

Lösung

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$
$$h(k,i) = ((k \bmod 5 + 1) + i \cdot (k \bmod 7)) \bmod 7$$

$$h(15,0) = (15 \bmod 5 + 1) \bmod 7 = 1$$

$$h(15,1) = (1 + 15 \bmod 7) \bmod 7 = 2$$
$$h(10,x) = (1 + 10 \bmod 7) \bmod 7 = 4$$

$$h(15,2) = (2 + 15 \bmod 7) \bmod 7 = 3$$
$$h(1,x) = (2 + 1 \bmod 7) \bmod 7 = 3$$

$$h(15,3) = (3 + 15 \bmod 7) \bmod 7 = 4$$
$$h(12,x) = (3 + 12 \bmod 7) \bmod 7 = 1$$

$$h(15,4) = (4 + 15 \bmod 7) \bmod 7 = 5$$

15 wird an die Stelle 5 gegeben.

$$h(8,0) = (8 \bmod 5 + 1) \bmod 7 = 4$$
$$h(8,1) = (4 + 8 \bmod 7) \bmod 7 = 5$$
$$h(10,x) = (4 + 10 \bmod 7) \bmod 7 = 0$$

10 wird an Stelle 0 gegeben während 8 an die ursprüngliche Stelle von 10 wandert, also an Stelle 4 eingesetzt wird.

Die Hashfunktionen sind keine gute Wahl, da bei $h_2(k)$ eine Schrittweite von 0 möglich ist, weiters belegt $h_1(k)$ niemals Platz 0 und 6.

4 · Graphen

4.1 Test vom 14.1.2011

a) Wir wenden schrittweise den in Abbildung 11 dargestellten Algorithmus auf den Graphen der ebenfalls in Abbildung 11 zu sehen ist an und geben jeweils den Zustand des Graphen und des „previous"-Arrays nach einem Durchlauf von $FUNKTION1(v_4)$ an.

1. Schritt

	v1	v2	v3	v4	v5
previous[v]	NULL	NULL	NULL	NULL	NULL

Wir setzen als erstes alle Vorgänger auf den Wert „NULL". Dann beginnen wir mit der Ausführung von $FUNKTION1(v4)$ die als erstes den Vorgänger von $v4$ auf sich selbst setzt und dann $v4$ in den Stack gibt. Danach

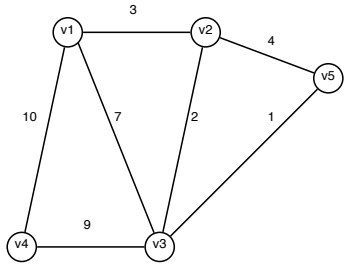
Abbildung 11: Beispiel 3.A vom Test am 14.1.2011

Aufgabe 3.A: Graphen **(18 Punkte)**

Gegeben sei folgender Algorithmus *WasBinIch*, der auf einen ungerichteten, zusammenhängenden, gewichteten Graphen $G(V, E)$ angewendet wird. Der Parameter v ist ein Knoten aus dem Graphen G.

Algorithmus WasBinIch($G(V, E), v$)

```
1:  Globale Variable: G(V, E);
2:  Globale Variable: Feld previous;
3:  found = true;
4:  solange found == true {
5:    für alle w ∈ V {
6:      previous[w] = NULL;
7:    }
8:    found = FUNKTION1(v);
9:  }
10: retourniere G(V, E);
```



Graph G_1

Algorithmus FUNKTION1(v)

```
1:  S = neuer Stack;
2:  previous[v] = v;
3:  S.push(v); // fügt Element vorne hinzu
4:  solange nicht S.isEmpty() {
5:    k = S.pop();     // entfernt vorderstes
                       // Element und liefert es zurück
6:    für alle Knoten w ∈ N(k) {
7:      falls previous[w] == NULL dann
        {
8:        S.push(w);
9:        previous[w] = k;
10:     } sonst falls previous[k] ≠ w dann
        {
11:       E = E\{(k, w)};
12:       return true;
13:     }
14:   }
15: }
16: return false;
```

a) (8 Punkte) Wenden Sie den Algorithmus *WasBinIch* durch den Aufruf $G_2 = \text{WasBinIch}(G_1(V, E), v4)$ auf den gegebenen Graphen G_1 an und zeichnen Sie den Graphen G_2. Das Auslesen der Nachbarn $w \in N(k)$ eines Knoten k erfolgt, bezogen auf die Knotenbezeichnung, in lexikographischer Reihenfolge.

b) (4 Punkte)

- Auf welchem aus der Vorlesung bekannten Verfahren beruht *WasBinIch*?
- Was berechnet der Algorithmus *WasBinIch*?

c) (6 Punkte) Kreuzen Sie zutreffende Aussagen an. Jede Zeile wird nur dann gewertet wenn Sie vollständig richtig ist.

- Die Laufzeit (Worst-Case) beträgt bei einem vollständigen Graph:
  $\Theta(|V|^2)$ ☐ $\Theta(|V|^2 log|V|)$ ☐ $\Theta(|V|^3)$ ☐ keine der angeführten ☐
- Die Laufzeit (Worst-Case) beträgt bei einem dünnen Graphen ($|E| = \Theta(|V|)$):
  $\Theta(|V|^2)$ ☐ $\Theta(|V|^2 log|V|)$ ☐ $\Theta(|V|^3)$ ☐ keine der angeführten ☐
- Bei dem oben angeführten Algorithmus handelt es sich um ein...
  rekursives Programm ☐ iteratives Programm ☐ keines von beiden ☐

wird $v4$ aus dem Stack entnommen und all Nachbarknoten von $v4$ der Reihenfolge nach betrachtet.

Der Vorgänger vom ersten Nachbar $v1$ und vom zweiten Nachbar $v3$ werden auf $v4$ gesetzt. Weiters werden $v1$ und $v3$ der Reihe nach in den Stack S gegeben.

	v1	v2	v3	v4	v5
previous[v]	v4	NULL	v4	NULL	NULL

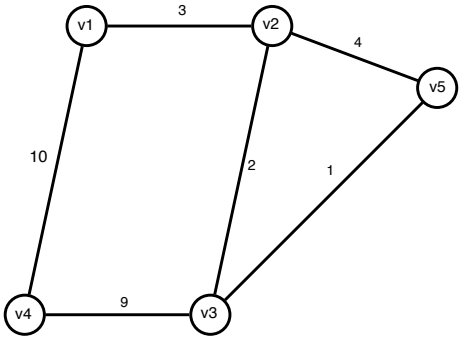
S: $\begin{array}{|c|}\hline v3 \\ \hline v1 \\ \hline \end{array}$

Danach wird $v3$ (=k) aus dem Stack entnommen und seine Nachbarn betrachtet. Da der Vorgänger von $v3$ schon vorher auf $v4$ (=$previous[k]$) gesetzt wurde und somit nicht $v1$ (=w) entspricht wird die Kante $(v3, v1)$ aus dem Graphen entfernt und die Funktion retourniert den Wert „true". Damit ergibt sich nach dem ersten Aufruf von $FUNKTION1(v4)$ folgender Zustand von „previous":

	v1	v2	v3	v4	v5
previous[v]	v4	NULL	v4	v4	NULL

Wir erhalten den in Abbildung 12 dargestellten Graphen.

Abbildung 12: Graph nach der ersten Anwendung von $FUNKTION1(v4)$



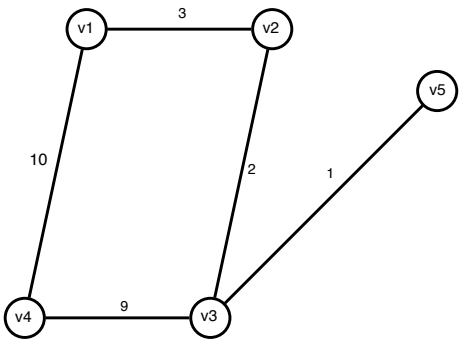
2. Schritt

Nach einer weiteren Anwendung von $FUNKTION1(v4)$ kommen wir auf folgenden Zustand von „previous":

	v1	v2	v3	v4	v5
previous[v]	v4	v3	v4	v4	v3

Da es sich beim Vorgänger von $v5$ nicht um $v2$ handelt wird die Kante $(v5, v2)$ gelöscht und es ergibt sich der in Abbildung 13 dargestellte Graph.

Abbildung 13: Graph nach der zweiten Anwendung von $FUNKTION1(v4)$



3. Schritt

Nach dem dritten Schritt ergibt sich folgendes „previous"-Array:

	v1	v2	v3	v4	v5
previous[v]	v4	v3	v4	v4	v3

Nachdem der Vorgänger von $v2$ nicht $v1$ ist wird die Kante $(v2, v1)$ gelöscht und es ergibt sich der Graph von Abbildung 14.

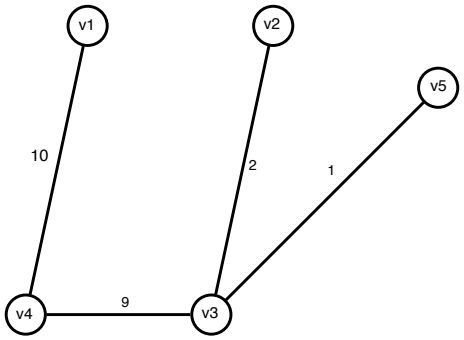
4. Schritt

Nach dem 4. Schritt erhalten wir sowohl das gleiche „previous"-Array als auch den gleichen Graphen wie im vorigen Schritt. Diesmal wird von der Funktion allerdings „false" zurückgegeben und der Algorithmus terminiert. Bei G_2 handelt es sich also um den in Abbildung 14 dargestellten Graphen.

b)

- Der Algorithmus beruht auf dem Tiefensuche-Prinzip (Aus dem Stack wird immer der Knoten der als letztes reingegeben wurde als erstes wieder entnommen und betrachtet).

Abbildung 14: Graph nach der dritten Anwendung von $FUNKTION1(v4)$



- „WasBinIch" entfernt die Kreise eines Graphen und bildet somit – bei einem zusammenhängenden Graphen – einen (Spann)-Baum.

c)

- Die Worst-Case Laufzeit beträgt bei einem vollständigen Graphen $\theta\left(|V^3|\right)$, da der Graph $\theta\left(|V|^2\right)$ Kanten besitzt die Theta $\theta(|V|)$ mal durchgegangen werden [Jug11].
- Die Worst-Case Laufzeit beträgt bei einem dünnen Graphen $\theta\left(|V^2|\right)$, da der Graph $\theta(|V|)$ Kanten besitzt die Theta $\theta(|V|)$ mal durchgegangen werden [Jug11].
- Bei dem Algorithmus handelt es sich um ein iteratives Programm.

5 · Beispiel für dynamische Programmierung

5.1 Angabe

Mit 40℃ im Schatten hat sich der Sommer gerade eingestellt. Da dir heiß ist, möchtest du die hohe Temperatur mit dem Verzehr eines großen Eises für dich etwas erträglicher gestalten. Als Student verfügst du nur über ein begrenztes Einkommen und beschließt daher maximal sieben Euro auszugeben. Die Eisreserven sind gerade knapp und deswegen darfst von jeder Sorte nur eine Kugel nehmen. Um zu bestimmen welche Sorten die bestmögliche Kombination unter den gegebenen Bedingungen darstellen, beschließt du dynamische Programmierung zu verwenden.

Eissorte	Genussfaktor	Kosten
Creme de la Creme	10	3
Brokkoli	3	1
Vanille	7	2
Málaga	8	2
Schokolade	7	3
Kaviar	6	8

Tabelle 2: Eissorten

5.2 Erstellen der Matrix

Als erstes erstellen wir eine Tabelle mit den nötigen Einträgen. Da wir die Wahl zwischen sechs verschiedenen Sorten haben, sollte unsere Matrix eigentlich sieben Zeilen besitzen (eine zusätzliche Zeile für die Wahl keiner einzigen Sorte). Die Sorte Kaviar überschreitet allerdings unsere bescheidenen Mittel von sieben Euro. Deshalb brauchen wir diese nicht zu beachten und somit können wir auf die Zeile für diese Sorte komplett verzichten. Somit erstellen wir eine Tabelle mit insgesamt 6 Zeilen (Tabelle 3).

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte								
Creme de la Creme								
Brokkoli								
Vanille								
Málaga								
Schokolade								

Tabelle 3: Matrix für unser Beispiel

Wie wir sehen stehen die Spalten für die Kosten (Gewicht) des Eis und die Zeilen für die verschiedenen Eissorten (Gegenstände). In den Zellen der Tabelle tragen wir den Genussfaktor (Wert) ein.

Anmerkung:

Üblicherweise werden in der LVA „Algorithmen und Datenstrukturen 1", Koffer oder Rucksäcke so gepackt, dass ein bestimmtes Gewicht nicht überschritten wird und sich gleichzeitig ein möglichst hoher

Spaßfaktor ergibt.

In diesem Zusammenhang wird der Wert der maximiert werden soll auch manchmal als Kosten oder „c" bezeichnet. In diesem Musterbeispiel stellen die Kosten des Eises allerdings das eigentliche Gewicht „w", also die Randbedingung die nicht überschritten werden soll dar. Dahingegen wird die Variable die maximiert werden soll hier als Genussfaktor bezeichnet, sie stellt also laut LVA-Regelung das eigentliche „c" dar.

Ich hoffe mal diese Erläuterung verwirrt nicht mehr als sie hilft.

Kurzfassung:
w…Kosten des Eises
c…Genussfaktor

Es ist leicht feststellbar, dass wenn wir keine einzige Sorte Eis auswählen wir davon auch keinen Nutzen haben. Somit können wir die erste Spalte mit lauter Nullen füllen. Da es leider kein „Gratis-Eis" gibt können wir auch keine Sorte wählen ohne zumindest etwas zu zahlen. Daraus resultiert, dass auch die Spalte mit Kosten Null mit lauter Nullen gefüllt werden muss. Insgesamt ergibt sich also das in Tabelle 4 dargestellte Bild.

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte	0	0	0	0	0	0	0	0
Creme de la Creme	0							
Brokkoli	0							
Vanille	0							
Málaga	0							
Schokolade	0							

Tabelle 4: Initialisierung der Matrix

Nach diesem Schritt könne wir endlich anfangen die erste Eissorte zu betrachten. „Creme de la Creme" besitzt einen Genussfaktor von 10 und Kosten von 3. Wir tragen also in der ersten Zeile bei Kosten 3 einen Genussfaktor von 10 ein (Tabelle 5).

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte	0	0	0	0	0	0	0	0
Creme de la Creme	0			10				
Brokkoli	0							
Vanille	0							
Málaga	0							
Schokolade	0							

Tabelle 5: Matrix nach der Betrachtung der ersten Eissorte

Nun sehen wir uns die Eissorte „Brokkoli" an, die nur einen geringen Genussfaktor von 3 besitzt, dafür aber auch nur einen Euro kostet. Da in der Spalte mit Kosten 1 in der vorhergehenden Zeile noch kein Wert steht, können wir in diese Spalte den Genussfaktor dieses Eises eintragen.

Von der vorhergehenden Zeile sehen wir, dass wir schon die Möglichkeit besitzen ein Eis zu wählen das die Kosten 3 und Genussfaktor 10 besitzt. Dieses können wir natürlich weiterhin auswählen. Deshalb übertragen wir den Wert von der oberen Zeile in Spalte 3.

Weiters können wir jetzt die Eissorte von vorher nehmen und mit der aktuellen Sorte, Brokkoli kombinieren. Insgesamt ergeben sich dadurch Kosten von 4 und ein Genusswert von 13 den wir in die aktuelle Zeile eintragen. Nach diesen Überlegungen erhalten wir die in Tabelle 6 dargestellte Matrix.

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte	0	0	0	0	0	0	0	0
Creme de la Creme	0			10				
Brokkoli	0	3		10	13			
Vanille	0							
Málaga	0							
Schokolade	0							

Tabelle 6: Matrix nach der Betrachtung der zweiten Eissorte

Die Geschmackssorte Vanille besitzt einen Genussfaktor von 7 bei Kosten von 2 Euro. Da bei der Spalte mit Kosten 2 noch kein Wert eingetragen ist schreiben wir dort den Genussfaktor von der aktuellen Eissorte

Vanille an.

Jetzt betrachten wir wieder die Zeile davor um zu sehen welche weiteren Kombinationen möglich sind. Wenn wir Vanille-Eis und den ersten Wert in der oberen Zeile wählen kommen wir auf Kosten von 3 und einen Genussfaktor von 10. Dieser Wert entspricht dem von der vorhergehenden Zeile. Wir haben also eine Kombination gefunden die gleich gut wie die vorhergehende war. Im Endeffekt bleibt der Wert in der Spalte 3 also der Selbe.

In Spalte 3 der vorhergehenden Zeile befindet sich der Genussfaktor 10. Zusammen mit der aktuellen Sorte ergibt sich ein Genussfaktor von 17 bei Kosten von 5 Euro. Da sich in der Spalte 5 noch kein Wert befindet tragen wir diesen Genussfaktor also dort ein.

Zum Schluss betrachten wir noch den Genussfaktor von 13 bei Kosten von 4. Zusammen mit der aktuellen Eissorte ergibt sich ein Genussfaktor von 20 bei Kosten von 6. Wir tragen also diesen Wert ein. Die anderen Werte werden von oben übernommen, da wir noch keine besseren Kombinationen für die Kosten von 1 und 4 gefunden haben (Tabelle 7).

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte	0	0	0	0	0	0	0	0
Creme de la Creme	0			10				
Brokkoli	0	3		10	13			
Vanille	0	3	7	10	13	17	20	
Málaga	0							
Schokolade	0							

Tabelle 7: Matrix nach der Betrachtung der dritten Eissorte

„Málaga" besitzt einen Genussfaktor von 8 bei Kosten von 2 Euro. Da der Genusswert höher als der bisher gefundene von 7 ist schreiben wir diesen in die dafür vorgesehene Spalte.

Bei den Kosten von 1 übernehmen wir wieder den Wert von oben, da mit der aktuellen Eissorte natürlich keine Kombination möglich ist die Kosten von 1 aufweist.

Weiters tragen wir in Spalte 3 einen Genusswert von 11 ein, der sich aus

dem Genusswert 3 aus der ersten Spalte der letzten Zeile und dem Genusswert von Málaga ergibt.

Wir betrachten den nächsten Wert von 7 bei Kosten von 2 aus der vorhergehenden Zeile. Eine Kombination mit Málaga gibt einen Genussfaktor von 15 bei Kosten von 4. Dieser Wert ist höher als der Vorhergehende in Spalte 4 und wird deshalb dort eingetragen.

Der nächste Genussfaktor aus der vorhergehenden Zeile ist 10. Gemeinsam mit der aktuellen Sorte kommt man also auf einen Genussfaktor von 18 bei einem Gewicht von fünf. Dieser wird da er größer als 17 ist, in die fünfte Spalte eingetragen.

13, der nächste Genusswert, ergibt zusammen mit dem aktuellen Eis einen Wert von 21 und wird deshalb in Spalte 6 (=4+2) eingetragen.

Genussfaktor 17 kombiniert mit dem Genussfaktor von Málaga-Eis ergibt den Wert von 25 der auf Grund der Kosten von 7 Euro in die korrespondierende Spalte geschrieben wird.

Da der letzte Wert aus der vorhergehenden Zeile schon Kosten von 6 besitzt und diese zusammen mit den Kosten des aktuellen Eises über die festgesetzten Maximalkosten von 7 Euro hinausgeht brauchen wir diesen nicht mehr zu betrachten und kommen schließlich auf Tabelle 8.

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte	0	0	0	0	0	0	0	0
Creme de la Creme	0			10				
Brokkoli	0	3		10	13			
Vanille	0	3	7	10	13	17	20	
Málaga	0	3	8	11	15	18	21	25
Schokolade	0							

Tabelle 8: Matrix nach der Betrachtung der vierten Eissorte

Nach der gleichen Vorgehensweise wie in den Schritten zuvor kommen wir, nachdem wir auch die letzte Eissorte betrachtet haben auf die in Tabelle 9 dargestellte Matrix.

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte	0	0	0	0	0	0	0	0
Creme de la Creme	0			10				
Brokkoli	0	3		10	13			
Vanille	0	3	7	10	13	17	20	
Málaga	0	3	8	11	15	18	21	25
Schokolade	0	3	8	11	15	18	21	***25***

Tabelle 9: Matrix nach der Betrachtung der fünften Eissorte

5.3 Rückverfolgung

Wenn wir die letzte Zeile von Tabelle 9 betrachten, sehen wir das wir einen maximalen Genussfaktor von 25 erhalten. Um nun zurückzurechnen welche Auswahl uns zu diesem Wert geführt haben gehen wir die Tabelle rückwärts durch.

Wir starten bei Wert 25 in der letzten Zeile. Da es sich beim Genussfaktor in der darüberliegenden Zeile in derselben Spalte um den gleichen Wert handelt schließen wir daraus, dass wir die Sorte Schokolade nicht ausgewählt haben.

Wenn wir nach dem ersten Vorkommen des Wertes 25 in der Spalte mit Kosten 7 suchen landen wir in der vorletzten Zeile und sehen, dass die Sorte ***Málaga*** ausgewählt wurde. Wir ziehen also vom gesamten Genussfaktor den Genussfaktor dieser Sorte ab. Somit kommen wir auf einen Wert von 17. Diesen Wert müssen wir in der Spalte 5 suchen, da von den Gesamtkosten von 7 die Kosten des Málaga-Eises von 2 abgezogen werden.

Wir suchen die Tabellenzeilen in Spalte 5 nach dem ersten Auftreten des Genussfaktor 17 ab und werden in der Zeile mit der Sorte ***Vanille*** fündig. Wir haben also auch diese Eissorte ausgewählt. Wiederum ziehen wir den Genussfaktor dieses Eises vom dem übrig gebliebenen Genussfaktor 17 ab und bekommen den Wert 10. Als neue Kosten erhalten wir den Wert 3 (=5-2).

Der Genussfaktor von 10 tritt erstmals in der Zeile der Eissorte ***„Creme de la Creme"*** auf, die also auch gewählt wurde. Da diese Sorte einen Genussfaktor von 10 besitzt ergibt sich bei weiterer Subtraktion vom vorher

übrig gebliebenen Genussfaktor ein Wert von Null. Es wurde also keine weitere Sorte gewählt.

Eine Plausibilitätsüberprüfung der gesamten Werte ist in Gleichung 2 und 3 zu sehen. Sie ergibt die erwarteten Werte und lässt so zumindest darauf hoffen, dass wir die optimale Lösung, unter der gegebenen Einschränkung nicht mehr als sieben Euro auszugeben, gefunden haben.

$$w_{gesamt} = w_{Malaga} + w_{Vanille} + w_{Creme\_de\_la\_Creme} = 2 + 2 + 3 = 7 \quad (2)$$

$$c_{gesamt} = c_{Malaga} + c_{Vanille} + c_{Creme\_de\_la\_Creme} = 8 + 7 + 10 = 25 \quad (3)$$

5.4 Vollständiges Ausfüllen der Tabelle

Tabelle 9 stellt alle möglichen Lösungen eigentlich schon dar und sollte auch beim Test durchaus als richtig gewertet werden. Um eine meiner Meinung nach nur verwirrendere „vollständige" Darstellung nach der gegebenen Formel zu erhalten muss man die vorhergehende Tabelle noch leicht verändern.

So sind die fehlenden Werte jeweils durch den größtmöglichen Wert den sich durch die vorhergehende Zeile oder Spalte ergeben zu ersetzen. Auch schon ausgefüllte Werte müssen durch eventuell auftretende größere Werte, die sich in vorhergehenden Zeilen oder Spalten ergeben haben können, ersetzt werden.

Die „vollständige" Matrix für das vorhergehende Beispiel wird in Tabelle 10 gezeigt. Zu beachten ist, dass diese Matrix wie leicht nachzuvollziehen ist auch „nicht sinnvolle" Werte enthält.

⇓Eissorte ⇒ Kosten	0	1	2	3	4	5	6	7
Keine Sorte	0	0	0	0	0	0	0	0
Creme de la Creme	0	0	0	10	10	10	10	10
Brokkoli	0	3	3	10	13	13	13	13
Vanille	0	3	7	10	13	17	20	20
Málaga	0	3	8	11	15	18	21	25
Schokolade	0	3	8	11	15	18	21	25

Tabelle 10: „Vollständige" Tabelle

6 · Uni Wien

6.1 Test vom 3.10.2011

Aufgabe 1

a) Geben Sie ein Programm in C++ artigem Pseudocode an, dessen Aufwand von der Ordnung $\Omega(n)$ und $O(n^2)$, aber nicht von der Ordnung $\Theta(n)$ oder $\Theta(n^2)$ ist.

```
for (long count = 0; count < n * log(n); count++);
```

Aufwand: $\Theta\left(n \cdot \log(n)\right)$

b) Geben Sie in C++ artigem Pseudocode eine rekursive Funktion an, deren Laufzeitordnung $\Theta\left(n^4 \log(n)\right)$ ist. Zeigen Sie mit Hilfe des Mastertheorems, dass Ihre Funktion wirklich die geforderte Laufzeitordnung hat.

```
void function(int n)
{
    if (n < 1) return;
    /* Loop with Theta (n^4) - f(n) = n^4, c = 4 */
    for (int count = 0; count < pow(n,4); count++);
    /*  a = 16 (calls), b = 2 (half the run time for recursive
        function) */
    for (int count = 0; count < 16; count++) function(n/2);
}
```

Da $\log_b(a) = c$ ergibt sich eine Laufzeit $T(n) = n^c \log n = n^4 \log n$.

Aufgabe 2 Addieren Sie zu Ihrer Matrikelnummer die Zahl 573296148. Die Ziffern der Summe seien in der Reihenfolge von links nach rechts in einem Array gespeichert. Sortieren Sie die Ziffern aufsteigend mit

a. Counting Sort.

b. Quicksort.

c. Selection Sort.

Geben Sie jeweils alle benötigten Zwischenschritte so genau an, dass der Ablauf des Algorithmus klar ersichtlich wird.

a) *Matrikelnummer* + 573296148 = 1234567 + 573296148 = 574530715

Zu sortierendes Array (A):

5	7	4	5	3	0	7	1	5

Zählerarray (C):

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

Sortiertes Array (B):

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

Literatur

[Jug11] Juggl3r. ***Laufzeit des Algorithmus von Beispiel 3 – Test vom 14.01.2011***. Juni 2011. URL: http://www.informatik-forum.at/showthread.php?85164-After-Test-14.01.2011&p=693282&viewfull=1#post693282.