DSEA WS 12-13

Dozent:
Elmar Schömer

Mitschrift von:
André Groß

Zuletzt Aktualisiert:
28. Januar 2013

`figures/uni_logo.png`

Zusammenfassung:

Im Mittelpunkt der Veranstaltung stehen Methoden zur Entwicklung (vor allem zeit-) effizienter Algorithmen. Dabei betrachten wir insbesondere solche Datenstrukturen, die eine effiziente Verwaltung von dynamischen Datenmengen ermöglichen. Ein Teil der Algorithmen und Datenstrukturen wird in den Übungen implementiert.

Mit dem Studium dynamischer Datentypen sowie weiterer Algorithmen schließt die Veranstaltung direkt an Einführung in die Programmierung bzw. Einführung in die Softwareentwicklung an. Allerdings werden nun mathematische Methoden zur Analyse von Algorithmen (Korrektheit und vor allem Aufwand) eingesetzt.

http://cg.informatik.uni-mainzde/dsea
Raum: Mi C02, Mo 03-428
Abgabe: Mittwochs 12:00
Sprache: Grundsätzlich Java, ggf auch Python

Inhaltsverzeichnis

Teil I

Skriptum

Kapitel 1

Grundlagen

1.1 Divide and Conquer

Bei einem „teile und herrsche“-Ansatz wird das eigentliche Problem so lange in kleinere und einfachere Teilprobleme zerlegt, bis man diese lösen („beherrschen“) kann. Anschließend wird aus diesen Teillösungen eine Lösung für das Gesamtproblem (re-)konstruiert.

Anwendung

„Teile und herrsche“ ist eines der wichtigsten Prinzipien für effiziente Algorithmen. Dabei wird ausgenutzt, dass bei vielen Problemen der Aufwand sinkt, wenn man das Problem in kleinere Teilprobleme zerlegt. Dies lässt sich meist durch Rekursive Programmierung umsetzen, bei der die Teilprobleme wie eigenständige Probleme gleichzeitig parallel oder sequenziell (einzeln nacheinander) behandelt werden, bis sie auf triviale Lösungen zurückgeführt sind oder der Restfehler hinreichend klein ist. Bei manchen Algorithmen steckt dabei die Kernidee im Schritt des „Teilens“, während die „Rekombination“ einfach ist (beispielsweise Quicksort). In anderen Verfahren (beispielsweise Mergesort) ist das Teilen einfach, während die Rekombination die Kernidee des Algorithmus enthält. In manchen Algorithmen sind beide Schritte komplex. [1]

1.1.1 Arithmetik großer Zahlen

In der Schule hat man bereits einen oder mehrere Algorithmen gelernt um die Grundrechenarten auf beliebige Zahlen an zu wenden. Hier sollen nun weitere vorgestellt werden.

Als Beispiel für große Arithmetische Operationen kann unter anderem die RSA angeführt werden. Speziell in Programmiersprachen wie C trifft man auf Rundungsprobleme, die abgefangen werden müssen. Hier verlässt man sich oft nicht auf die eingebaute Arithmetik und entwickelt eigene (zuweilen schnellere) Lösungsansätze.

[1] Wikipedia zu D&C

Addition

Nehmen wir nun zwei Zahlen und addieren sie wie früher in der Schule

$$\begin{array}{cccccccc} & 3 & 4 & 2 & 7 & 6 & 5 & 1 \\ + & & 2_1 & 6_1 & 4_1 & 4 & 3 & 2 \\ \hline & 3 & 6 & 9 & 2 & 0 & 8 & 3 \end{array}$$

Da Zahlen in verschiedensten Zahlensystemen benutzt und dargestellt werden können, von Binär über Dezimal bis hin zu beliebigen Basen, möchten wir eine gute allgemeingültige Darstellung wählen. Eine beliebige Zahl kann zu jeder Basis folgendermaßen dargestellt werden:

$$A = \sum_{i=0}^{n-1} a_i b^i : \qquad 0 \leq a_i < b$$

$$C = \sum_{i=0}^{n-1} c_i b^i : \qquad 0 \leq c_i < b$$

Der oben angesprochene Algorithmus hat mit der darauffolgenden Datenstruktur die Laufzeit

$$T(n) = c * n$$

Multiplikation

Nehemen wir hier nun zwei Binärzahlen:

```
 10 1011010 •1111001
          A •       C

   1010110 10. . . . . .
 +  101011 010. . . . .
 +   10101 1010. . . .
 +    1010 11010. . .
 +     000 000000 0. .
 +      00 0000000 0.
 +       1 01011010
 ----------------------
        .. 11 10001010
```

Abbildung 1.1: Binäre Multiplikation

Dieser Algorithmus kann mathematisch folgendermaßen aufgefasst werden:

$$
\begin{aligned}
&= AB \\
&= \left(\sum_{i=0}^{n-1} a_i b^i\right)\left(\sum_{j=0}^{n-1} c_j b^j\right) \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i c_j b^{i+j}
\end{aligned}
$$

Die Schulmultiplikation hat eine Laufzeit von $T(n) = c * n^2$. Dies bedeutet für große Zahlen, z.B.

$$n = 10^6 \Rightarrow T(n) = 10^{12}$$

Dies kostet uns sehr viel Zeit. Das die Addition schneller geht zeigte uns der Student Karazuba[1].

Erster Ansatz mit D&C

Wir haben Zwei Zahlen A und C von welchen wir je die vordere und die Hintere hälfte nehmen.

`figures/DundC.pdf`

Abbildung 1.2: Divide and Conquer bei zwei Feldern

Damit ergeben sich folgende Rechenschritte

$$
\begin{aligned}
A &= \sum_{i=0}^{n-1} a_i b^i = A_i b^{\frac{n}{2}} + A_0 \\
A_0 &= \sum_{i=0}^{\frac{n}{2}-1} a_i b^i \\
A_1 &= \sum_{i=0}^{\frac{n}{2}-1} a_{i+\frac{n}{2}} b^i
\end{aligned}
$$

b^n stellt einen binären Shift dar. Wenn wir nun A in zwei Hälften teilen, ist der vordere Teil (A_1) um $b^{\frac{n}{2}}$ voneinander verschoben.

$$
\begin{aligned}
A = 123456; &\quad C = 987654 \\
A \bullet C & \\
A_1 = 123; &\quad C_1 = 987 \\
A_0 = 456; &\quad C_2 = 654 \\
A_1 b^{\frac{n}{2}} + A_0 = A & \\
123000 + 456 &\quad = 123456
\end{aligned}
$$

[1]Paper von Karazuba und Offman

Hier ist der binäre Shift gut zu erkennen. Wie auch die Addition hat der Binärshift linearen Aufwand.

Das Produkt AC kann man nun auch folgendermaßen ausdrücken

$$
\begin{aligned}
AC &= (A_1 b^{\frac{n}{2}} + A_0)(C_1 b^{\frac{n}{2}} + C_0) \\
&= A_1 C_1 b^n + A_1 C_0 b^{\frac{n}{2}} \\
&\quad + A_0 C_1 b^{\frac{n}{2}} + A_0 C_0
\end{aligned}
$$

Indem das Problem auf diese Art immer weiter vereinfacht wird, überführe ich das quadratische Problem in viele Additionen mit linearem Aufwand.

Betrachten wir uns nun die Laufzeit des Algorithmus.

$$
\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + cn \\
T(1) &= c
\end{aligned}
$$

Dies nennt man eine **Rekursionsgleichung**!

Eine solche R. sagt nur bedingt etwas über die Laufzeit aus. Hierzu muss man diese Gleichung lösen und ggf. mit Induktion o.Ä. lösen. Bei Betrachtung der Glieder der rekursiven Folge erhalten wir

$$
\begin{aligned}
T(n) &= 4^k T\left(\frac{n}{2^k}\right) + cn \sum_{i=0}^{k-1} 2^i \\
&= 4^k T\left(\frac{n}{2^k}\right) + cn\left(2^k - 1\right)
\end{aligned}
$$

mit $\log_2 n$:

$$
\begin{aligned}
T(n) &= 4^{\log_2 n} T(1) + cn\left(2^{\log_2 n} - 1\right) \\
&= n^2 c + cn\,(n-1) \\
&\leq 2cn^2
\end{aligned}
$$

Wie man hier sieht, hat auch diese Methode einen quadratischen Aufwand.

1.1.2 Karazuba

Karazuba hatte die Idee, die Zahl der Teilprodukte von vier auf drei zu minimieren.

Dazu stellen wir das D&C Produkt AC um.

$$
\begin{aligned}
AC &= A_0 C_1 b^{\frac{n}{2}} + A_1 C_0 b^{\frac{n}{2}} + A_1 C_1 b^n + A_0 C_0 \\
&= (A_0 C_1 + A_1 C_0)\, b^{\frac{n}{2}} + A_1 C_1 b^n + A_0 C_0 \\
&= P_3 b^{\frac{n}{2}} + P_2 b^n + P_1
\end{aligned}
$$

Wenn wir uns nun die drei Produkte einzeln an-sehen ist die Vereinfachung schnell zu erkennen

$$
\begin{aligned}
P_1 &= A_0C_0 \\
P_2 &= A_1C_1 \\
P_3 &= A_0C_1 + A_1C_0 \\
&= (A_1C_1 + A_1C_0 + A_0B1 + A_0B_0) \\
&\quad - A_1C_1 - A_0C_0 \\
&= ((A_1 + A_0)(C_1 + C_0)) - P_2 - P_1
\end{aligned}
$$

denn man sieht, dass man bei P_3 nur noch ein Produkt rechnen muss und die anderen beiden Produkte zur Korrektur mittels Multiplikation und damit konstantem Aufwand in die Rechnung einfließen.

Bei der Frage an das Auditorium, wie hoch nun die Laufzeit sei, gab es die folgende

Behauptung

$$T_K(n) = cn^{\log_2 3}$$

Hier ist schön zu sehen, um wie viel niedriger

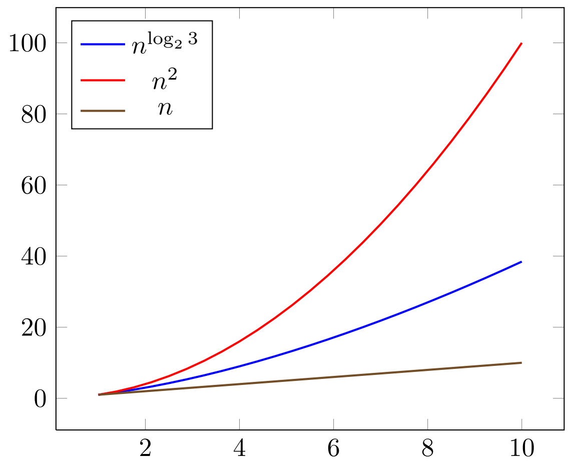


Abbildung 1.3: Laufzeit Multiplikation

der Aufwand im Gegenzug zu n^2 ausgefallen ist. Nachfolgend ein kleines Beispiel mit einer Größenordnung von 6 für n.

$$
\begin{aligned}
n &= 10^6 \\
n^2 &= 10^{12} \\
n^{\log_2 3} = n^{1{,}58\ldots} &\approx 10^9
\end{aligned}
$$

Normalerweise wird erwartet dass der Verlauf eines effizienteren Algorithmuses folgendermaßen verläuft, wobei man gut erkennen kann, dass der effizientere Algorithmus erst ab einem bestimmten n effizienter ist, als der naive Ansatz.

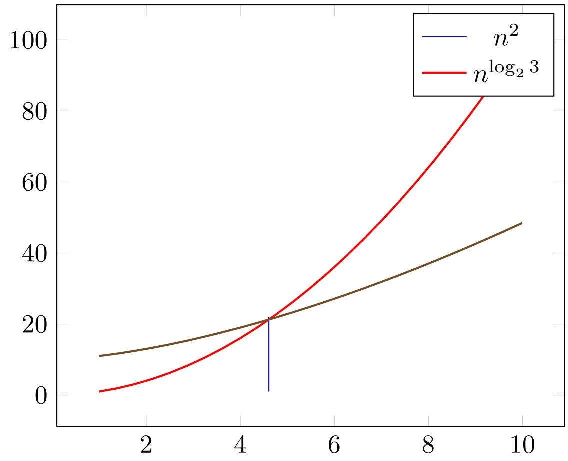


Abbildung 1.4: Laufzeit est. Break-even Point

Wie verhällt sich nun die Laufzeit für die doppelte Menge an Eingabedaten?

$$
\begin{aligned}
\frac{T_S(2n)}{T_S(n)} &= \frac{c(2n)^2}{c(n)^2} && = 4 \\
\frac{T_K(2n)}{T_K(n)} &= fracc(2n)^{\log_2 3}c(n)^{\log_2 3} && = 3
\end{aligned}
$$

Gut zu sehen ist, dass sich der Aufwand bei Verdopplung des Naiven vervierfacht und er sich bei K. nur verdreifacht.

1.1.3 Schnelle Matrizenmultiplikation nach Strassen

wir nehmen zwei Matrizen $A, B \in \mathbb{R}^{n \times n}$ und führen auf diesen eine Multiplikation aus.

$$
\begin{aligned}
C &= AB \\
c_{ij} &= \sum_{k=1}^{n} a_{ik} b_{kj} \qquad 1 \leq i, j \leq n
\end{aligned}
$$

deren Aufwand

$$T(n) = cn^3$$

entspricht.

$$
\left(\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{1,1} & A_{2,2} \end{array}\right)
\left(\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{1,1} & B_{2,2} \end{array}\right)
=
\left(\begin{array}{c|c} C_{1,1} & C_{1,2} \\ \hline C_{1,1} & C_{2,2} \end{array}\right)
$$

$$A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2} \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2}}$$

$$
\begin{aligned}
C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
C_{1,2} &= \ldots \\
&\vdots \\
C_{2,2} &=
\end{aligned}
$$

Der Aufwand ist hier nun

$$T(n) = 8T\left(\frac{n}{2}\right) + cn^2$$
$$T(1) = c$$

Wenn man diese Rekursionsgleichung löst erhält man für den Aufwand

$$T(n) = c^3$$

Hier kommt nun wieder Karazubas Idee ins Spiel den Aufwand von acht zu sieben zu minimieren.

Dies geschieht folgendermaßen:
Hab ich keine Lust zu...
Daraus folgt

$$\Rightarrow T(n) = 7T\left(\frac{n}{2}\right) + cn^2$$

Die Lösung hiervon, werden wir nun mit Hilfe des Mastertheorems ausrechnen, welches wir uns nach der Zusammenfassung herleiten wollen.

1.2 Master Theorem

Wir wollen nun einen Ansatz liefern um im allgemeinen Rekursionsgleichungen lösen. Dazu betrachten wir uns Probleme der Größe $\frac{n}{b}$, welche man als Teilprobleme eines Problems $T(n)$ auffassen kann. Die Laufzeit solcher Probleme ist $T\left(\frac{n}{b}\right)$. Mit der Anzahl an Teilproblemen und der Betrachtung der konstanten Anteile erhalten wir allgemein für Rekursionsgleichungen die Form

$$T(n) = aT\left(\frac{n}{b}\right) + cn^\alpha$$
$$T(1) = c$$

für $T\left(\frac{n}{b}\right)$ folgt

$$T\left(\frac{n}{b}\right) = aT\left(\frac{n}{b^2}\right) + c\left(\frac{n}{b}\right)^\alpha$$
$$T\left(\frac{n}{b^2}\right) = aT\left(\frac{n}{b^3}\right) + c\left(\frac{n}{b^2}\right)^\alpha$$

damit ergibt sich für $T(n)$

$$\begin{aligned} T(n) &= a(aT\left(\frac{n}{b^2}\right) + c\left(\frac{n}{b}\right)^\alpha) + cn^\alpha \\ &= a^2T\left(\frac{n}{b^2}\right) + ac\left(\frac{n}{b}\right)^\alpha + cn^\alpha \\ &\vdots \\ &= a^kT\left(\frac{n}{b^k}\right) + cn^\alpha \sum_{i=0}^{k-1}\left(\frac{a}{b^\alpha}\right)^i \end{aligned}$$

Die Rekursion bricht ab, wenn

$$\frac{n}{b^k} = 1 \Rightarrow k = \log_b n$$

$$n = b^k$$

Um die Lösung zu ermitteln müssen wir uns die möglichen Lösungen Betrachten.

Exkurs Geometrische Reihe:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}\text{ , für}|x| < 1$$
$$\sum_{i=0}^{k-1} x^i = \frac{x^k - 1}{x-1}\text{ , für}x \neq 1$$

Umschreiben der Potenzen:

$$\begin{aligned} a &= b^{\log_b a} \\ a^{\log_b n} &= \left(b^{\log_b a}\right)^{\log_b n} \\ &= b^{\log_b a \log_b n} \\ &= \left(b^{\log_b n}\right)^{\log_b a} \\ &= n^{\log_b a} \end{aligned}$$

1. Fall $\frac{a}{b^\alpha} < 1 \Leftrightarrow a < b^\alpha \Leftrightarrow \alpha < \log_b a$

$$\begin{aligned} T(n) &= a^{\log_b n}T(1) + cn^\alpha \frac{1}{1-\left(\frac{a}{b^\alpha}\right)} \\ &= n^{\log_b a}c + c'n^\alpha \\ &\leq c''n^\alpha \text{ , für hinreichend große n} \end{aligned}$$

2. Fall $\frac{a}{b^\alpha} > 1$

$$\begin{aligned} T(n) &= cn^{\log_b a}T(1) + cn^\alpha \frac{\left(\frac{a}{b^\alpha}\right)^{\log_b n} - 1}{\frac{a}{b^\alpha} - 1} \\ &= cn^{\log_b a} + c'n^\alpha \frac{n^{log_b a}}{n^\alpha} \\ &\leq c''n^{\log_b a} \end{aligned}$$

Karazuba:

$$a = 3, b = 2, \alpha = 1$$
$$\Rightarrow T_K(n) = cn^{\log_2 3}$$

Strassen:

$$a = 7, b = 2, \alpha = 2$$
$$\Rightarrow T_S(n) = cn^{\log_2 7}$$

3. Fall $\frac{a}{b^\alpha} = 1 \Rightarrow \alpha = \log_b a$

$$T(n) = cn^{\log_b a} + cn^\alpha \log_b n$$
$$T(n) \leq n''n^{\log_b a} \log_b n$$

$$a = 2, b = 2, \alpha = 1$$
$$\Rightarrow T(n) = cn \log n$$

Mergesort

$$T(n) = \begin{cases} c & \text{für } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{für } n \neq 1 \end{cases}$$

mit

$$\begin{aligned} T(n) &= aT(\frac{n}{b} + cn^{\alpha}) \\ &\Rightarrow a = 1, b = 2, \alpha = 1 \\ &\Rightarrow 3. \text{ Fall} \\ &\log_b a = \alpha \\ T(n) &= c' n \log_2 n \end{aligned}$$

z.B. binäre Suche

$$T(n) = \begin{cases} c & \text{für } n = 1 \\ T(\frac{n}{2}) + c & \text{für } n \neq 1 \end{cases}$$

$$\begin{aligned} &\Rightarrow a = 1, b = 2, \alpha = 0 \\ &\Rightarrow 3. \text{ Fall} \\ &\log_2 1 = 0 \\ T(n) &= c' \log_2 n \end{aligned}$$

1.2.1 Erweiterung des Theorems

$$f(n) = \begin{cases} c & \text{für } n = 1 \\ af(\frac{n}{b}) + cn^{\alpha} & \text{sonst} \end{cases}$$

1. Fall $\alpha < \log_b a \Rightarrow f(n) \in \mathcal{O}(n^{\log_b a})$
2. Fall $\alpha > \log_b a \Rightarrow f(n) \in \mathcal{O}(n^{\alpha})$
3. Fall $\alpha = \log_b a \Rightarrow f(n) \in \mathcal{O}(n^{\alpha} \log_2 n)$

Umschreiben der Potenzen:

$$\log_b n \in \mathcal{O}(\log_e n)$$

$$\log_b n = \frac{\log_e n}{\log_e b}$$

1.3 Akra Bazzi- Theorem (1998)

$$f(x) = \begin{cases} h(x) & \text{für } 1 \leq x \leq x_0 \\ af(\frac{x}{b}) + g(x) & \text{für } x > x_0 \end{cases}$$

Verallgemeinerte Form:

$$\sum_{i=1}^{m} a_i f\left(\frac{x}{b_i}\right) + g(nx) \text{ für } x > x_0$$

$$a > 0, b > 1, g : \mathbb{R}^+ \to \mathbb{R}^+$$

$\exists d_1, d_2 > 0 : d_1 \leq h(x) \leq d_2$ für $1 \leq x \leq x_0$

$\exists c_1, c_2 > 0 : c_1 g(x) \leq g(u) \leq c_2 g(x)$ für $x - \frac{x}{b} \leq u \leq x$

$$f(x) \in \theta(x^p(1 + \int_1^x \frac{g(u)}{u^{p+1}} du))$$

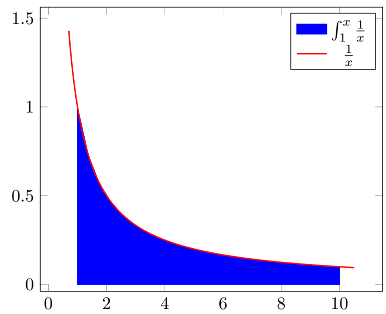
mit $p = \log_b a, b^p = a$
Beweis (Idee):

$$\begin{aligned} f(x) &= af(\frac{x}{b}) + g(x) \\ f(\frac{x}{b} &= af(\frac{x}{b^2}) + g(\frac{x}{b}) \\ &\vdots \\ f(x) &= a^k f(\frac{x}{b^k} + \sum_{i=0}^{k-1} a^i g(\frac{x}{b^i}) \end{aligned}$$

bis $\frac{x}{b^k} = 1 \Rightarrow k = \log_b x$

Beispiel:

$$H_n = \sum_{i=1}^{n} \frac{1}{i} \approx \int_1^n \frac{1}{x} dx \approx \ln n$$



$$\sum_{i=0}^{k-1} a^i g(\frac{x}{b^i}) \approx \int_0^{k-1} a^i g(\frac{x}{b^i}) di$$

Substitution:

$$\begin{aligned} u &= \frac{x}{b^i} = cb^{-i} \\ \frac{du}{di} &= -\ln bxb^{-i} \\ &= -\ln bu \\ di &= -\frac{1}{\ln bu} du \end{aligned}$$

$$\begin{aligned} \Rightarrow &= -\frac{1}{\ln b} \int a^i g(u) \frac{1}{u} du \\ &= \frac{1}{\ln b} \int_b^x (\frac{x}{u})^p g(u) \frac{1}{u} du \\ &= \frac{1 x^p}{\ln b} \int_b^x \frac{g(u)}{u^{p+1}} du \end{aligned}$$

mit

$$
\begin{aligned}
i &= \log_b \frac{x}{u} \\
a^i &= b^{\log_b a \log_b \frac{x}{u}} = (\frac{x}{u})^{\log_b a} = (\frac{x}{u})^p \\
i &= k-1 \Rightarrow u = xb^{-k+1} = (\frac{x}{b^k})b
\end{aligned}
$$

1.3.1 Anwendung von Akra Bazzi

$$
\begin{gathered}
T(n) = 2T(\frac{n}{2}) + n \log n \\
g(n) = n \log n \\
T(n) = \theta(n^1(1 + \int_1^n \frac{u \log u}{u^2} du)) \\
p = \log_b a = 1 \\
\int_1^n \frac{\ln u}{u} du = [\ln^2 u]_1^n = \frac{1}{2} \ln^2 n \\
T(n) = \theta(n(1 + \log^2 n)) = \theta(n \log^2 n)
\end{gathered}
$$

1.4 Landau-Symbole

$$
\begin{gathered}
f, g : \mathbb{N} \to \mathbb{N} \\
f(n) \in \mathcal{O}(g(n)) :\Leftrightarrow \exists c > 0 \exists n_0 \forall n > n_0 : f(n) \leq cg(n) \\
\Leftrightarrow \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty
\end{gathered}
$$

In Worten "f wächst nicht schneller als g"
z.B.

$$
\begin{gathered}
f(n) = 4n^3 + 6n^2 - 7n + 9 \in \mathcal{O}(n^3) \\
\in \Omega(n^3) \\
\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{4n^3 + 6n^2 - 7n + 9}{n^3} = 4 < \infty \\
> 0 \\
f(n) \in \Omega(g(n)) :\Leftrightarrow \exists c > 0 \exists n_0 \forall n > n_0 : f(n) \geq cg(n) \\
\Leftrightarrow \liminf_{n\to\infty} \frac{f(n)}{g(n)} > 0
\end{gathered}
$$

Bsp:

Martin

Kapitel 2

Sortieren und Ordnen

2.1 Quicksort

Als Beispiel für einen randomisierten D&C Algorithmus soll hier Quicksort dienen.

Quicksort ist ein Replace Verfahren, welches den Vorteil hat, dass man keinen zusätzlichen Speicher zum Sortieren benötigt.

Das Verfahren von Quicksort lässt sich gut an einem Beispiel erklären. Wir nehmen hierzu ein Feld von Zahlen. Man versucht beim partionieren mehr Intelligenz in die Sache zu stecken. Wir suchen uns dazu ein beliebiges Element (Pivot-Element -P-) heraus und bilden zwei Mengen von Elementen, eine Menge $X \leq P$ und eine Menge $Y \geq P$. Wir bewegen nun das Pivot-Element ans Ende (tauschen) und durchlaufen das Feld mit dem Index i und schreiben die Zahlen kleiner P an den Anfang und die Zahlen Größer P an das Ende. dann Tauschen wir das Pivot-Element wieder zurück und haben nun zwei Mengen größer, kleiner P im Feld.

```
partition(int a[],int l,int r){
   int x=a[r];
   int i=l-1;
   for(int j=l;j<r;j++){
      if(a[j]<=x){
         i=i+1;
         swap(a,i,j);
      }
   }
   swap(a,i+1,r);
   return i+1;
}
quicksort(int a[], int l, int r){
   if(l>=r) return;
   int p=partition(a,l,r);
   quicksort(a,l,p-1);
   quicksort(a,p+1,r);
}
```

Abbildung 2.1: Quicksort

2.1.1 Laufzeitanalyse

Worst-case
Pivotelement X ist immer das kleinste / größte Element in der betrachteten Teilfolge.

$$\begin{aligned} T(1) &= 0 \\ T(n) &= T(n-1) + cn \\ &= T(n-2) + c(n-1+n) \\ &= T(n-3) + c(n-2+n-1+n) \\ &\vdots \\ &= c\sum_{i=1}^{n} = c\frac{n(n+1)}{2} \in \theta(n^2) \end{aligned}$$

Erinnerung:

$$\mathcal{X} = \text{Zufallsvariable}$$
$$E(\mathcal{X}) = \sum pr(\mathcal{X} = x_i)x_i$$

z.B. Erwartete Augenzahl bei fairem Würfel

$$E(\mathcal{X}) = \sum_{i=1}^{6} \frac{1}{6} i = \frac{1}{6}\frac{67}{2} = 3.5$$

z.B. Wieviele Münzwürfe benötigt man bis zum ersten Mal ”Kopfërscheint?

$$\begin{aligned} E(\mathcal{X}) &= \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i i \\ E(\mathcal{X}) &= \frac{1}{2}\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i i = \frac{1}{2}\frac{1}{(1-\frac{1}{2})^2} = \frac{1}{2}\frac{1}{\frac{1}{4}} \\ &= 2 \end{aligned}$$

Nebenrechnung:

$$\begin{aligned} \sum_{i=1}^{\infty} ip^{i-1} &= \frac{d}{dp}\sum_{i=0}^{\infty} p^i \\ &= \frac{d}{dp}\frac{1}{1-p} = \frac{1}{(1-p)^2} \\ &|p| < 1 \end{aligned}$$

Weiter mit QS:

$$
\begin{aligned}
T(n) &= \text{Erwartungswert} \\
T(n) &= \sum_{i=1}^{n} \frac{1}{n} \left(T(i-1) + T(n-i)\right) + cn \\
&\in \theta((n+1)ln(n+1) = \theta(n \log n) \\
nT(n) &= \sum_{i=1}^{n} T(i-1) + \sum_{i=1}^{n} T(n-i) + cn^2 \\
&= 2\sum_{i=0}^{n-1} T(i) + cn^2 \\
&(n-1)T(n-1) \\
&= 2\sum_{i=0}^{n-2} T(i) + c(n-1)^2
\end{aligned}
$$

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + c(n^2 - (n^2 - 2n + 1))$$

$$
\begin{aligned}
nT(n) &= (n+1)T(n-1) + c'n \\
T(n) &= \frac{n+1}{n} T(n-1) + c' \\
&= \frac{n+1}{n}\left(\frac{n}{n-1} T(n-2) + c'\right) + c' \\
&= \frac{n+1}{n-1} T(n-2) + \frac{n+1}{n} c' + \frac{n+1}{n+1} c' \\
&= \frac{n+1}{n-k} T(n-k-1) + c'(n+1) \sum_{i=n-k+1}^{n+1} \frac{1}{i} \\
&6k := n-1 \\
&= (n+1)T(0) + c'(n+1)\sum_{i=1}^{n+1}\frac{1}{i}
\end{aligned}
$$

2.2 Selektionsproblem

Gegeben: Eine Folge von n Elementen (unsortiert) Gesucht: das $k-t$ kleinste Element

```
int select(int a[],int l,int r, int k){
   int p= partition(a,l,r);
   if(l+k<p)
      return select(a,l,p-1,k);
   if(l+k>p)
      return select(a,p+1,r,k+l-p-1);
   return a[p];
}
```

Abbildung 2.2: Selection Sort

Wir erhalten damit die Rekursionsgleichung

$$T(n) = cn^1 + T(\frac{n}{2})$$

Womit wir die Werte$a = 1$; $b = 2$; $\alpha = 1$; für die Lösung mit dem Mastertheorem erhalten.

$$
\begin{aligned}
&\Rightarrow T(n) \in \mathcal{O}(n) \\
T(n) &= \text{Erwartungswert der Laufzeit von select} \\
T(n) &\le cn + \sum_{i=1}^{n} \frac{1}{n} \max\left(T(i-1), T(n-i)\right) \\
&\le cn + \frac{a}{n} \sum_{1=1}^{n} \max\left((i-1), (n-i)\right) \\
&\le cn + 2\frac{a}{n} \sum_{j=\frac{n}{2}}^{n-1} j \\
&\le cn + 2\frac{a}{n}\left(\frac{(n+1)n}{2} - \frac{(\frac{n}{2}-1)\frac{n}{2}}{2}\right) \\
&\le cn + a\left(\frac{3}{4}n - \frac{1}{2}\right) \\
&\le cn + \frac{3}{4}an
\end{aligned}
$$

hier fehlt noch was....
Mit Gaussformel:

$$\sum_{j=a}^{b} j = \frac{b(b+1)}{2} - \frac{(a-1)a}{2}$$

2.3 Deterministischer Algorithmus für Selektionsproblem

Das Selektionsproblem sucht den Median von einem Feld indem es das Feld in Blöcke unterteilt

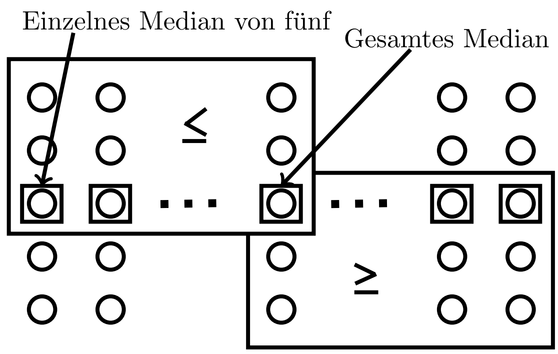


Abbildung 2.3: Selektionsproblem

$\frac{3n}{10}$ der Elemente sind$\le M$

$$T(n) = cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

Akra Bazzi Theorem:

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{k} a_i T\left(\frac{n}{b_i} + g(n)\right) \\
\Rightarrow T(n) &= \mathcal{O}\left(n^p\left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right) \\
T(n) &= \mathcal{O}\left(n^p\left(1 + \int_1^n \frac{cu}{u^{p+1}} du\right)\right) \\
&= \mathcal{O}\left(n^p\left(1 + \frac{c}{1-p} n^{-p+1}\right)\right) \\
&= \mathcal{O}(n^p + n^1) = \mathcal{O}(n)
\end{aligned}
$$

Nebenrechnung:

$$
\begin{aligned}
&\sum_{i=1}^{k} \frac{a_i}{b_i^p} = 1 \\
&g(n) = cn \\
&a_1 = a_2 = 1;\, b_1 = 5;\, b_2 = \frac{10}{7} \\
&\left(\frac{1}{5}\right)^p + \left(\frac{7}{10}\right)^p = 1 \\
&p \approx 0,84 \leq 1
\end{aligned}
$$

Kapitel 3

Einfache Datenstrukturen

3.1 Binäre Bäume

Wir haben die Zufallsvariablen $\mathcal{X}_1, \mathcal{X}_2$ für die Augenzahl zweier fairer Würfel mit

$$E(\mathcal{X}_1) = E(\mathcal{X}_2) = 3,5$$

$$E(\max(\mathcal{X}_1, \mathcal{X}_2)) \approx 4,47$$

$$= \sum pr(Z = z)z = \frac{1}{36}1 + \frac{3}{36}2 + \frac{5}{36}3 + \dots$$

Frage:
Was ist die erwartete maximale Tiefe eines naturwüchsigen binären Baumes?

t_n zugehörige Zufallsvariable

$$E(T_n) = \sum_{i=1}^{n} \frac{1}{n} E(\max(t_{i-1}, T_{n-i}) + 1$$

Nebenrechnung:

$$\max(\mathcal{X}, \mathcal{Y}) \leq \log_2(2^x + 2^y)$$

$$E(\max(2^{\mathcal{X}}, 2^{\mathcal{Y}})) \leq E(2^{\mathcal{X}} + 2^{\mathcal{Y}}) = E(2^{\mathcal{X}}) + E(2^{\mathcal{Y}})$$

$$\text{z.B.} \mathcal{X} = 10,\ \mathcal{Y} = 5$$

$$\log_2(2^{10} + 2^5)$$

neue Zufallsvariable

$$\mathcal{X}_n = 2^{T_n} \text{ expotentielle Tiefe}$$

$$\begin{aligned}
E(\mathcal{X}_n) &= 2 \sum_{i=1}^{n} \frac{1}{n} E(\max(\mathcal{X}_{i-1}, \mathcal{X}_{n-i})) \\
&\leq 2 \sum_{i=1}^{n} \frac{1}{n} (E(\mathcal{X}_{i-1}) + E(\mathcal{X}_n - i)) \\
&= \frac{4}{n} \sum_{j=0}^{n-1} E(\mathcal{X}_j)
\end{aligned}$$

Merke:

$$E(\mathcal{X}_n) = \frac{4}{n} \sum_{i=0}^{n-1} E(\mathcal{X}_j)$$

Teil II

Die letzte(n) Vorlesung(en)

Kapitel 4

VL 12.11.

4.1 AVL-Bäume

Dies sind Bbinäre Suchbäume, welche sich selbst balancieren.

`figures/avlbase.pdf`

Abbildung 4.1: AVL-Baum

`figures/avlbase.pdf`

Abbildung 4.2: AVL-Bäume mit Tiefe $t = 2$

`figures/avlbase.pdf`

Abbildung 4.3: AVL-Bäume mit Tiefe $t = 3$

`figures/avlbase.pdf`

Abbildung 4.4: Ein AVL-Baum mit Tiefe $t = 3$ sieht niemals so aus, da er balanciert ist.

`figures/avlins.pdf`

Abbildung 4.5: Ein AVL-Baum

`figures/avlins.pdf`

Abbildung 4.6: Einfügen in A

figures/avlins.pdf

Abbildung 4.7: Einfügen in C

Kapitel 5

VL 14.11.

Kapitel 6

VL 19.11.

6.1 AVL Bäume Wiederholung

Die Logaritmische Tiefe ist wichtig für die Suchzeit usw.
Linker und Rechter Teilbaum dürfen sich für Suche usw. in ihrer Tiefe nur um max. Eins unterscheiden.

$$
\begin{aligned}
f(z) &= (z+c)^p \\
&= \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(z0)(z-z_0) \\
\text{wähle } z0 &= 0 \\
f'(z) &= p(z+c)^{p-1} \\
f''(z) &= p(p-1)(z+c)^{p-2} \\
f^{(n)}(z) &= p(p-1)(p-2)\dots(p-n+1)(z+c)^{p-n} \text{ (Taylor)}
\end{aligned}
$$

hier fehlt was!

$$
\begin{aligned}
b_0 = 1, b_n &= \sum_{i=1}^{n} b_{i-1} b_{n-i} \\
B(z) &= \sum_{n=0}^{\infty} b_n z^n \Rightarrow B(z) \frac{1-\sqrt{1-4z}}{2z} \\
\sqrt{1-4z} &= (1-4z)^{\frac{1}{2}} \\
&= \sum_{n=0}^{\infty} \left(\begin{matrix} \frac{1}{2} \\ n \end{matrix} \right) 1^{\frac{1}{2}-n} (-4z)^n \\
&= \sum_{n=0}^{\infty} \begin{matrix} \frac{1}{2} \\ n \end{matrix} (-4)^n z^n \\
&= 1 + \sum_{n=0}^{\infty} \begin{matrix} \frac{1}{2} \\ n \end{matrix} (-4)^n z^n \\
&= 1 + \sum_{n=0}^{\infty} \begin{matrix} \frac{1}{2} \\ n+1 \end{matrix} (-4)^{n+1} z^{n+1}
\end{aligned}
$$

hier fehlt auch noch ne Menge...

$$
b_n = -\frac{1}{2} \begin{matrix} \frac{1}{2} \\ n+1 \end{matrix} (-4)^{n+1}
$$

den rest kann ich nicht lesen....

6.2 Bijektion zwischen Binärbäumen

$$
n = 3; c_c = \frac{1}{n+1} \begin{matrix} 2n \\ n \end{matrix} \Rightarrow c_3 = 5
$$

Graphik

6.3 Amortisierte Analyse am Beispiel von 2-5-Bäumen

a-b-Bäume: Jeder Knoten Des Baumes hat mind a Kinder und Höchstens b Kinder. Blattorientierte Speicherung der zu verwaltenden Elemente

figures/aa25baum.pdf

Abbildung 6.1: 2-5-Baum

Zahl der Blätter n :

$$
2^t \le n \le 5^t
$$

$$
\log_5 n \le t \le \log_2 n
$$

$$
\Rightarrow t \in \theta(\log n)
$$

Strategie zum Einfügen und Löschen von Elementen `figures/25bauminsert.pdf`

Abbildung 6.2: Einfügen in 2-5-Baum

Beim Löschen von Elementen kann eine Kaskade Von Fusionsoperatoren auf dem Suchpfad notwendig werden. Ggf. Wurzel löschen und Kinder Zusammenlegen.

Laufzeit für Suchen, Einfügen, Löschen $\in \theta(\log n)$

Kapitel 7

VL 21.11

7.1 Amortisierte Analyse am Beispiel des Binärzählers

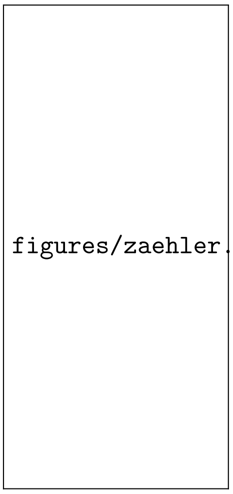


Abbildung 7.1: Binärzähler

Worst case Kosten einer Inkrement- Operation

$$\mathcal{O}(\log_2 n)$$

Gesamtkosten für eine Folge von n Inkrement-Operationen (beginnend beim Zählerstand 0)

$\frac{n}{2}$	verursachen Kosten 1	Endung 0
$\frac{n}{4}$	verursachen Kosten 2	Endung 01
$\frac{n}{8}$	verursachen Kosten 3	Endung 011
$\frac{n}{16}$	verursachen Kosten 4	Endung 0111

Gesamtkosten:

$$\leq \sum_{i=1}^{\infty} i\frac{n}{2^i} = n\sum_{i=1}^{\infty} i\left(\frac{1}{2}\right)^i = 2n$$

Nebenrechnung:

$$\begin{aligned} x\sum_{i=1}^{\infty} ix^{i-1} &= x\left(\sum_{i=0}^{\infty} x^i\right)' \\ &= x\left(\frac{1}{1-x}\right)' \\ &= \frac{x}{(1-x)^2} \end{aligned}$$

7.1.1 Kontomethode

$$\begin{aligned} Konto(i) &= \text{Kontostand vor der i-ten Operation} \\ cost(i) &= \text{tatsächliche Kosten der i-ten Operation} \\ \sum_{i=1}^{n} cost(i) &= \sum_{i=1}^{n} Konto(i) - Konto(i+1) + a(i) \\ a(i) &= \text{ammotisierte Kostender i-ten Operation} \\ \Rightarrow \sum_{i=1}^{n} cost(i) &= Konto(i) - Konto(n+1) + \sum_{i=1}^{n} a(i) \end{aligned}$$

7.1.2 amortisierte Analyse der Rekonstruierungskosten für eine Folge von m Einfüge- oder Löschoperation in einem 2-5-Baum

Ausgangspunkt: leerer Baum
Nicht betrachtet werden die Suchkosten. Wir konzentrieren uns auf die Split- und Fusions-Operationen.

Kontoführung:

Knotengrad	1	2	3	4	5	6
Sparbetrag	2	1	0	0	1	2

Sparplan:
$2RE$ pro Einfüge- bzw.Löschoperation
$a_i = 2$
Einfügen:

figures/zaehler.pdf

Abbildung 7.2: Kosten 2-5-Baum einfügen

Löschen:

figures/zaehler.pdf

Abbildung 7.3: Kosten 2-5-Baum Löschen

Die m Einfüge- und Löschoperationen auf einem anfangs leeren Baum haben amotisierte Kosten=2. $\Rightarrow$ Rekonstruierungskosten insgesamt belaufen sich auf $2m$

Kapitel 8

VL 26.11.

8.0.3 Skiplisten

```
class ListNode{
   int kex;
   ListNode next,down;

   boolean search(int x){
      ListNode node = head;
      do{
          while(x>node.next.key)
            node=node.next;
         if (x == node.next.key)
            return true;
         node = node.down;
      }while(node!=null)
      return false;
      }
}
```

Abbildung 8.1: Skiplisten Pseudocode

figures/skiplist.pdf

Abbildung 8.2: Skipliste, ein Beispiel

figures/skiplist.pdf

Abbildung 8.3: In Skipliste einfügen

Erwartete Tiefe:
mit Wahrscheinlichkeit p soll ein Element von Level i auf Level $i+1$ angehoben werden
Erwartete Zahl von Elementen auf Level i: $p^i n$

$$p^i n < 1 \Rightarrow n < \left(\frac{1}{p}\right)^i \Rightarrow i > \log_{\frac{1}{p}} n$$

Suchzeit:
Wir müssen die Ebenen von oben nach unten durchlaufen.
Die Frage ist, wie viele Elemente bei der verfeinerten Liste hinzukommen.

figures/skiplist.pdf

Abbildung 8.4: In Skipliste einfügen

$\mathcal{X}$ =Zahl der Elemente auf Level i zwischen zwei benachbarten Elementen auf Level i+1

$$E(\mathcal{X}) = \sum_{j=1}^{\infty} j(1-p)^{j-1}p = p\frac{1}{(1-(1-p))^2} = \frac{1}{p}$$

Erwartete Suchzeit:

$$T_P(n) = \frac{1}{p}\log_{\frac{1}{P}} n = \mathcal{O}(\log n)$$

plot $\frac{x}{lnx}$ mit hervorheben des Sattelpunktes

$$\begin{aligned}
f(x) &= x \log_x n \\
&= \frac{x \ln n}{\ln x} \\
f'(x) &= \ln n \frac{1 \ln x - x\frac{1}{x}}{(\ln x)^2} = 0 \\
&\Rightarrow \ln x = 1 \\
&\Rightarrow x = e \\
&\Rightarrow p = \frac{1}{e}
\end{aligned}$$

8.0.4 Dynamisches Programmieren

es ist eine Technik um sich optimale Lösungen für Teilprobleme zu suchen.
Beispiel:
Sie sollen eine reihe von Matrizen multiplizieren. Die Dimensionen der Matrizen können durchaus unterschiedlich sein, was ist nun eine geschickte Reihenfolge um möglichst wenig Operationen zu benötigen?
geg: n-Matrizen A_i für $i = 1, \ldots, n$, $A_i \in \mathbb{R}^{d_i \times d_{i+1}}$
ges: Optimale Auswerte-Reihenfolge für das Produkt $A_1\dot{A}_2\dot{A}_3.\ddots.\dot{A}_n$
Beispiel: $n = 3$

$$\begin{aligned} (A_1A_2)A_3 &= 2105 + 258 && = 180 \\ A_1(A_2A_3) &= 1058 + 2108 && = 560 \end{aligned}$$

$$cost(A_iA_{i+1}) = \theta(d_id_{i+1}d_{i+2})$$

Bild 2

$$(A_1 \ldots A_k)(A_{k+1} \ldots A_n)$$

Idee: Suche nach optimaler Teillösung und Konstruiere daraus die optimale Gesamtlösung.

$$cos(A_iA_{i+1} \ldots A_j) = c_{ij}$$

$$c_{ij} = \min_{i<k<j} c_{ik} + c_{k+1,j} + d_id_{k+i}d_{j+1}$$

Kapitel 9

VL 28.11

Andere Schreibweise:

$$c_{ij} = cost(A_i A_{i+1} \dots A_j)$$

$$c_{ij} = \min_{k=1,\dots j-1} (c_{ik} +_{i+k,j-k} + d_i d_{i+k} d_{i+j})$$

`figures/skyscraper.pdf`

Abbildung 9.1: ka was das sein sollte :)

9.0.5 Egg Dropping

$$n = 15[\text{Stockwerke}] \qquad k = 2[\text{Eier}]$$

`figures/skyscraper.pdf`

Abbildung 9.2: Egg dropping von einem 15 stöckigen Hochhaus

Die Zahl der verbleibenden Versuche bei n verbleibenden Stockwerken und k verbleibenden Eier bezeichnen wir als $t_{n,k}$. Betrachten wir nun den Wurf eines der k Eier aus Stockwerk x und die beiden möglichen Ergebnisse.

$$t_{n,k} = 1 + \min_{2 \leq x \leq n} \max(t_{x-1,k-1}, t_{n-x,k})$$

1. Fall: Ei zerbricht

$$t_{x-1,k-1}$$

2. Fall: Ei überlebt

$$t_{n-x,k}$$

$$t_{15,2} = 1 + \min(\max(t_{1,1}, t_{13,2}), \max(t_{2,1}, t_{12,2}, \dots)$$

Kapitel 10

VL 3.12.

Kapitel 11

VL 5.12.

Informationstheoretische untere Schranke für vergleichsbasierte Sortierverfahren

$$a[0], a[1], a[2]$$

Entscheidungsbaum:

figures/vergleichsbaum.pdf

Abbildung 11.1: Vergleichsbaum

$$a_i \neq a_j \text{ für } 0 \leq i \neq j < n$$

#Blätter im entscheidungsbaum

$$\geq n!$$

Sei $t_{\max}$ die maximale Tiefe im Entscheidungsbaum

$$\Rightarrow 2^{t_{\max}} \geq n! \Rightarrow t_{\max} \geq \log_2 n!$$

$$\ln n! = \ln(1$$

figures/m1.jpg

Sei t' die mittlere Tiefe der Blätter im Entscheidungsbaum

figures/vergleichsbaum.pdf

Abbildung 11.2: Entscheidungsbaum

$$mt'(m) = l(t'(l) + 1) + r(t'(r) + 1)$$

$$t'(m) = \frac{l}{m} t'(l) + \frac{r}{m} + 1 \text{ wird minimal für } l = r = \frac{m}{2}$$

$$t'(m) \geq t'(\frac{m}{2}) + 1 \Rightarrow t'(m) \geq \log_2 m$$

11.1 Hashing mit Verkettung

BILD
Verwaltung einer unsortierten Menge.

- search
- insert
- delete

$$|U| >> m \; n = \#\text{verwaltete Elemente}$$

$$h : U \to \{0, 1, \ldots, m-1\} \text{ Hashfunktion}$$

Belegungsfaktor der Hashtabelle

$$\alpha = \frac{n}{m}$$

Idealisierte Annahme:
Hashfunktion h verteilt die n Elemente zufällig gleichverteilt auf die m Tabelleneinträge.

n_i sei die erwartete Anzahl von Elementen, die auf Tabelleneintrag i gehasht werden.

$$n_i = \alpha = \frac{n}{m} \Rightarrow \text{ Suchzeit } = \mathcal{O}(1 + \alpha)$$

unter der Vorraussetzung, dass sich die Hashfunktion in Konstanter Zeit auswerten lässt.
Beispiel der Wahl einer Hashfunktion:

$$h(k) = (ak + b) \mod m$$

Kapitel 12

VL 10.12.

Idealisierte Annahme: Die n Elemente werden unabhängig und gleichverteilt auf die m Tabellenplätze abgebildet.

$$h : U \to \{0, \dots, m-1\}$$

Frage: Wie hoch ist die mittlere Suchzeit für ein zufälliges Element aus der Tabelle? Sei $k_1, k_2, \dots, k_n$ die Einfügereihenfolge der n Schlüssel in die Tabelle?

$$\mathcal{X}$$

12.1 Universelles Hashing

$\mathcal{H}$ Klasse von Hashfunktionen z.B. $h(k) = (ak + b) \mod p \mod m$ $1 \leq a < p, 0 \leq b < p m < p$Primzahl

Def. $\mathcal{H}$ heißt universell

$$\Leftrightarrow \forall k \neq l \in U : |\{h \in \mathcal{H} | h(k) = k(l)\}| \leq \frac{|\mathcal{H}|}{m}$$

Sei n_i die erwartete Länge der Kollisionsliste bei Feldeintrag i.

y_k = Länge der Kollisionsliste für den Schlüssel k

Kapitel 13

VL 12.12.

$$E(\sum_{i=0}^{n-1} n_i^2) = E(\sum_{i=0}^{n-1}(n_i + 2\binom{n_i}{2})) = E(\sum_{i=0}^{n-1} n_i) + 2E(\sum_{i=0}^{n-1}\binom{n_i}{2})$$

Dies entspricht der Zahl der Kollidierendden Paare in dar Primärtabelle.

$$= n + 2E(\sum_{k\neq l\in T} \mathcal{X}_{kl}) = n + 2\sum_{k\neq l\in T} E(\mathcal{X}_{kl})$$

$$= n + 2\binom{n}{2}\frac{1}{n} = n + \frac{2n(n-1)}{2n} \leq 2n$$

Weitere Klasse von univversellen Hashfunktionen

$$E(a\mathcal{X} + b\mathcal{Y}) = aE(\mathcal{X}) + bE(\mathcal{Y})$$

$$k = (k_1, k_2, \ldots, k_d), 0 \leq k_i < p$$

$$a = (a_1, \ldots, a_d)$$

$$h_a(k) = \sum_{i=1}^{d} a_i k_i \mod p$$

Tabellengröße $m = p$ Primzahl.

$$\mathcal{H} = \{h_a | a = (a_1, \ldots, a_d), 1 \leq a_i < p\}$$

$$\mathcal{H} = (p-1)^d$$

Wähle $k \neq l$

$$h_a(k) = \sum_{i=1}^{d} a_i k_i \mod p = \sum_{i=1}^{d} a_i l_i \mod p = h_a(l)$$

FEHLEND

Für jede der $(p-1)^{d-1}$ Möglichkeiten die a_i's auf der rechten Seite zu wählen gibt es genau ein a_j, das zu einer Kollission zwischen den Schlüsseln k und l führt.

13.1 Graphen

Ein Graph G besteht in der Regel aus Vertices V und Kanten (Edges) E.

$$G = (V, E)$$

$$E \subseteq V \times V$$

$(u, v) \in E$ gerichtet
$\{u, v\}$ ungerichtet
DAG : directed acyclicgraph
planare Graphen
Eulersche Polyederformel

$$|V| + |F| = |E| + 2$$

Würfel: $8 + 6 = 12 + 2$

13.1.1 Adjazenzmatrix A

$$A_{u,v} = \begin{cases} 1 & \text{falls } (u, v) \in E \\ 0 & \text{sonst} \end{cases}$$

Speicherbedarf: $\mathcal{O}(|V|^2)$

Adjazenzliste
FIGURE

13.2 Tiefensuche

engl. Depth-First-Search

```
forall v in V
   visited [v] =false;
stack S;
S.push(s); // s entspricht Startknoten
while(!S.empty()) {
   u=S.pop();
   forall(u,v) in E
      if(!visited[v]){
         S.push(v);
      }
}
visited[v]=true;
```

Abbildung 13.1: Pseudocode Tiefensuche

Kapitel 14

VL 17.12.

14.1 Breitensuche

Fehlend

14.2 Kürzeste Wege Algorithmen.

Eigenschaften von Algorithmen , die mittels Kantenrelaxierungen kürzeste Wege bestimmen.

Obere Schranke: $\forall v \in V : d[v] \geq S(s,v)$ Beweis durch Induktion nach der Zahl der Kantenrelaxierungen.

IA Nach initialisierung gilt die Induktions-Annahme.

IS $relax(u,v)$

- **1.Fall** $d[v]$ wird nicht verändert, woraus die Richtigkeit der IA folgt.
- **2.Fall** ...

Konvergenzeigenschaft:
Sei $s \rightsquigarrow u \to v$ein Kürzester Weg von s nach V mit $d[u] = S(s,u)$, denn gilt nach dem Relaxieren der Kante $(u,v) : d[v] = S(s,l)$

$$d[v] \leq d[u]+w(u,v) = S(s,u-+w(u,v) = S(s,v)$$

Korrektheitsbeweis Bellman-Ford
Betrachte einen kürzesten Weg $s = v_0 \to v_1 \to \ldots \to v_k = v$ Folgende Invariante gilt: Nach dem i-ten Schleifendurchlauf: $d[v_i = S(s,v_i)$ Ind. Bew. $s \rightsquigarrow v_i \to v_{i+1}$ Im $(i+1)$-ten Schleifendurchlauf wird irgendwann auch die Kante (v_i, v_{i+1}) relaxiert. Konvergenzeigenschaft: $d[v_{i+1}] = S(s, v_{i+1})$

Erkennung negativer Zyklen

```
forall (u,v) in E
   if (d[v] > d[u] +w(u,v))
      return negativer Zyklus erkannt
return kein negativer Zyklus vorhanden
```

Sei $c = (s = v_0 \to v_1 \to \ldots \to v_k = v)$ ein negativer Zyklus $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$

Kapitel 15

VL 19.12.

15.1 Dijkstra-Algorithmus

$$G = (V, E)\, w : E \to \mathbb{R}^+$$

```
init();
PriorityQueue Q = V;
S = \emptyset
while(!Q.empty()){
   u=Q.deletemin();
   S = Sv{u};
   relax(u,v){
      if(d[v]>d[u]+w(u,v)){
         d[v]=d[u]+w(u,v);
         pred[v]=u;
         Q.decreasekey(v);
      }
      forall (u,v) \in E
         relax(u,v);
   }
```

Laufzeit:

$$\mathcal{O}(|v| cost(deletemie) + |E| cost(decreasekey))$$

$$\mathcal{O}((|v| + |E|) \log |v|)$$

wenn PriorityQueue durch balancierten Suchbaum verwaltet wird.

alternativ $\mathcal{O}(|v| \log |v| + |E|)$ PQ mit Hilfe von Fibonacci-Heaps

15.1.1 Korrektheitsbeweis

BILD

Beh. $\forall v \in V$ gilt nach Ablauf des Algorithmus $d[v] = S(s, v)$

Bew. durch Widerspruch
Annahme $\exists v : d[v] > S(s, v)$ (obere Schrankeneigenschaft)
Sei v der erste Knoten, für den dies gilt, zum Zeitpunkt, wenn er aus Q entnommen wird.

Sei $s \rightsquigarrow x \to y \rightsquigarrow v$ ein kürzester Weg von s nach v.

$$d[v] \le d[y] = S(s, y) \le S(s, v) < d[v]$$

$\le$ gilt aufgrund der positiven Kantengewichte.

$d[x] = S(s, x)$ weil x vor dem "falschen" Knoten v betrachtet wird.

Konvergenzeigenschaft $d[y] = S(s, y)$ weil $x \to y$ relaxiert und $s \rightsquigarrow x \to y$ kürzester Pfad.

$d[v] \le d[y]$, v wird vor y aus der PQ entnommen.

Kapitel 16

VL 07.01.

Kapitel 17

VL 09.01.

Kapitel 18

VL 14.01.

```
FibNode{
   Fibnode left, right, down, up;
   int deg;
   bool mark;
   float key;
}

FibHeap{
   Fibnode min;
   ... decreaseked, deletemin,
         consolidate, insert;
}

void decreasekey (Fibnode x, float newkey){
   if(newhey>x.key error;
   x.key=newkey;
   if(x.up ==null){
      if(x.key<minkey) min=x;
      return;
   }
   FibNode y=x.up;
   if(x.key >= y.key) return;
      //löse x ab
   do{ y=x.up;
      remove x from y childlist;
      y.degree--;
      add x to rootlist;
      x.up F null;
      x.mark = false;
      if (x.key <min.key) min = x;
      x=y;
   } while (x.mark== true);
}

void consolidate(){
   int dmax =floor(log(phi,n));
   Fibnode[] A=new FibNode(dmax+1);
   for(int i=0;i<=dmax;i++){
      FibNode x=v;
      int d =x.deg;
      while (A[d]!=Null){
         FibNode y=A[d];
         if(x.key>y.key)
            swap(x,y);
            remove y from rootlist;
            add y to childlist of x; x.deg++;
            d++;
      }
      A[d]=x;
   }
   for(int i=0;i<=dmax;i++)
      if (A[i]!= Null){
         add A[i] to new rootlist;
         if (A[i].key<min.key) min =A[i];
      }
}
```

18.1 Minimal aufspannende Bäume

$G = (V, E)$ zusammenhängend, ungerichtet.

Mit einer Kostenfunktion $w : E \to \mathbb{R}$

Gesucht ist ein **Spannbaum** $T_{min} \subseteq E$ mit

$$w(T_{min}) = \sum_{(u,v)\in T_{min}} w(u, v)\text{minimal}$$

```
Greedy-Algo
T= emptyset;
while (T noch kein Spannbaum){
   wähle eine sichere Kante (u,v) in E
   T=T cup {(u,v)};
}
```

18.1.1 Lemma

Sei T_{min} ein minimaler Spannbaum für $G = (V, E)$ bzgl. $w : E \to \mathbb{R}$ und $T \subseteq T_{min} \subseteq E$ eine Teillösung.

Sei (S, VS) ein Schnitt von G und die Kanten von T respektieren diesen Schnitt $\nexists(u, v) \in T, u \in S, v \in VS)$ dann ist die Kante (u, v) mit minimalem Gewicht, die den Schnitt kreuzt, eine sichere Kante für T.

Somit $T' = T \cup \{(u, v)\}$ und T' zu T_{min} ergänzt werden kann.

18.1.2 Beweis

Sei T_{min} ein minimal aufspannender Baum, der aus der Teillösung T entstanden ist.

Kapitel 19

VL 16.01.

Sei $(u, v) \in E$ die Kante mit kleinstem Gewicht, die über den Schnitt führt.

$$T_{min} \tag{19.1}$$
$$T \tag{19.2}$$
$$T \cup T_{min} \tag{19.3}$$

Der gewählte Schnitt (S, VS) **respektiert** die Teillösung T, d.h. keine Kante von T läuft über diesen Schnit

$$T' = T \cup \{(u, v)\}$$

T' kann zu einem minimalen Spannbaum T'_{min} ausgebaut werden.

$$w(T'_{min} = w(T_{min}$$

Durch Einfügen von (u, v) in T_{min} entsteht ein Zyklus C. Da $u \in S$ und $v \in VS$ gibt es miedestens eine Kante von $C\{(u, v)\}$ die über den Schnitt führt: $w(x, y) \geq w(u, v)$

$$T'_{min} = T_{min}\{(x, y)\} \cup \{(u, v)\}$$

T'_{min} ist ebenfalls Spannbaum.

$$w(T_{min}) \leq w(T'_{min}) F w(T_{min}) - w(x, y) + w(u, v) \leq W(T_{min})$$

BILD

v	1	2	3	...	7	...	12	13
rep	1	2	3	...	1	...	12	13
next	7	-1	-1	...	-1	...	12	13

rep ist der Repräsentanten-Knoten zur Identifikation einer Komponente.

Die Korrektheit folgt unmittelbar aus dem Cut-Lemma.

19.0.3 Laufzeit

Sortierphase

$$\mathcal{O}(|E| \log |V|)$$

mit

$$|E| \leq |V|^2 \ , \ \log |E| = \mathcal{O}(\log |V|)$$

Baumphase

$$|E| * find - Op$$

$$|V| * union - Op$$

19.1 Bemerkungen zu den Kosten der Union-find-Operationen

Beim Kruskal-Algorithmus:

$$cost(find) = \mathcal{O}(1), \text{ da nur 2 Feldzugriffe nötig}$$

$$\sum_{i=1}^{|V|-1} cost(union_i) = \mathcal{O}(|V| \log |V|)$$

Voraussetzung: Repräsentanten der jeweilskürzeren Liste werden umbenannt.

Frage aus der Sicht eines Knotens: "Wie oft werde ich umbenannt?"

Antwort: Nach jeder Umbenennung verdoppelt sich die Komponentengröße mindestens.

$$\text{nach } k\text{-Umbenennungen} \leq 2^k \Rightarrow k \leq \log_2 |V|$$

Gesamtlaufzeit

$$\mathcal{O}(|E| \log |V| + |E| + |V| \log |V|) = \mathcal{O}(|E| \log |V|)$$

Geht es schneller?

Kapitel 20

VL 21.01.

20.1 Prim-Algorithmus

BILD

Distanzfeld d verwaltet die minimale Entfernung des Knotens zu den Knoten S der Teillösung T.

```
if(w(v,v_1)<d[v_1]){
   d[v_1]=w(v,v_1);
   pi[v_1]=v;
}

forall v\in V{
   d[v]= \infty ;
   pi[v]=-1;
}
d[s]=0;
T=\emptyset ;
PriorityQueue Q(V,d);
while (!Q.empty(){
   v=Q.deletemin();
   if(v \neq s)
      T=T \cup {(pi(v),v)};
   forall (v' \in Adj(v) && v' \in Q)
      if(w(v,v')<d[v]){
         d[v']=w(v,v');
         pi[v']=v;
   }
}
```

Nach Ablauf des Prim-Algo. gilt:

$$T_{min} = \{(\pi(v), v) | v \in V\{s\}\}$$

Laufzeit:

$$\mathcal{O}(|V| cost(deletemin) + |E| cost(decreasekey))$$

$$= \mathcal{O}(|V| \log |V| + |E|)$$

20.2 Flussalgorithmen

$$G = (V, E) \text{ gerichtet.}$$

mit zwei ausgezeichneten Knoten s und t.

Gesucht ist ein maximaler Fluss

$$f : E \to \mathbb{R}^{>0}$$

sodass

$$0 \leq f(u, v) \leq c(u, v)$$

Kapazitätsfunktion:

$$c : E \to \mathbb{R}^{>0}$$

Flusserhaltung:

$$\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u')$$

für alle $v \in V\{s, t\}$
$|f|$ = Wert des Flusses, der von s nach t transportiert werden kann.

Nettofluss:

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

Ziel: Finde eine Belegung der Kanten mit Flusswerten f, so dass $|f|$ maximal und f erfüllt Kapazitätsbedingung und Flusserhaltung. Es gilt:

$$|f| = \sum_{v \in V} f(v, t) - \sum_{v \in V} f(t, v)$$

20.2.1 Restgraph

BILD!!!

Idee: Finde einen Fluss f' im Restnetzwerk $G_f = (V, E_f)$ und verbessere den Ausgangsfluss f im Originalgraph durch Addition von f und $f' : f \oplus f'$

Kapitel 21

VL 20.01.

Restnetzwerk

$$G_f = (V, E_f)$$

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{für } (u,v) \in E \\ f(v,u) & \text{für } (v,u) \in E \\ 0 & \text{sonst} \end{cases}$$

1. Idee Suche einen Fluss verbessernden Pfad p im Restnetzwerk G_f
Bottleneck-Kante:

$$c_f(p) = \min\{c_f(u,v) \text{für} (u,v) \in p\}$$

Sei f ein Fluss im Originalgraphen G und f' im Restnetzwerk G_f, dann ist auch

$$(f \oplus f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{für } (u,v) \in E \\ 0 & \text{sonst} \end{cases}$$

(1)

$$\begin{aligned}(f \oplus f')(u,v) &= f(u,v) + f'(u,v) - f'(v,u) \\ &\leq f(u,v) + f'(u,v) - f(u,v) \\ &= f'(u,v) \geq 0\end{aligned}$$

(2)

$$\begin{aligned}(f \oplus f')(u,v) &= f(u,v) + f'(u,v) - f'(v,u) \\ &\leq f(u,v) + f'(u,v) \\ &\leq f(u,v) + c(u,v) - f(u,v) \\ &= c(u,v)\end{aligned}$$

(3)

$$\begin{aligned}\sum_{v \in V} (f \oplus f')(u,v) &= \sum_{v \in V} \{f(u,v) + f'(u,v) - f'(v,u)\} \\ &= \sum f(u,v) + \sum f'(u,v) - \sum f'(v,u) \\ &= \sum f(v,u) + \sum f'(v,u) - \sum f'(\overline{u,v}) \ldots \\ &= \sum_{v \in V} (f \oplus f')(v,u)\end{aligned}$$

$$\begin{aligned}|(f \oplus f')| &= |f| + |f'| \\ &= \sum (f \oplus f')(s,v) - \sum (f \oplus f')(v,s) \\ &= \sum f(s,v) + f'(s,v) - f'(v,s) - \sum f(v,s) + f'(v,s) \\ &= \sum f(s,v) - \sum f(v,s) + \ldots ? \\ &= |f| + \ldots\end{aligned}$$

21.1 Schnitte

Sei $V = S \sqcup T$ mit $s \in S$ und $T \in T$ ein Schnitt durch G.

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

Definition:

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in T} \sum_{v \in S} f(u,v)$$

Es gilt:

$$f(S,T) \leq \sum_{u \in S} \sum_{v \in T} f(u,v) \leq \sum_{u \in S} \sum_{v \in T} c(u,v) \leq c(S,T)$$

Lemma:
Sei f ein Fluss und (S,T) ein beliebiger Schnitt, dann gilt:

$$|f| \leq c(S,T)$$

$$\begin{aligned}|f| &= \sum_v f(s,v) - \sum_v f(v,s) =^! f(S,T) \\ &= \sum_v f(s,v) - \sum_v f(v,s) + \sum_{v \in V} \sum_{u \in S\{s\}} f(u,v) - \sum_{u \in S\{s\}} f(v,u) \\ &= \sum_{v \in V} \sum_{u \in S} f(u,v) - \sum_{v \in V} \sum_{u \in S} f(v,u)\end{aligned}$$

Kapitel 22

VL 28.01.

22.1 Max-Flow-Min-Cut Theorem

Gegeben sei ein Fluss f im Netzwerk $G = (V, E)$ von s nach t.

Folgende Aussagen sind äquivalent:

1. f ist ein maximaler Fluss

2. Es gibt keinen flussverbessernden Pfad im Restnetzwerk $G_f = (V, E_f)$

3. Es gibt einen Schnitt (S, T) mit $|f| = c(S, T)$

Beweis: (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1)

Mit Kontraposition:
$\neg(2) \Rightarrow \neg(1)$

Sei f' ein Fluss im G_f entlang des existierenden flussverbessernden Pfades

$$|(f \oplus f')| = |f| + |f'| > |f| \Rightarrow f \text{ nicht maximal}$$

(2) $\Rightarrow$ (3) Definiere $S = \{v \in V | \exists s \rightsquigarrow v \text{ in } G_f\}, T = VS, t \notin S$ da kein flussverb. von s nach t in G_f nach Vor.

Sei (u, v) mit $u \in S$ und $v \in T$, dann gilt: $f(u, v) = c(u, v)$

Annahme: $f(u, v) < c(u, v) \Rightarrow (u, v) \in E_f$ mit $c_f(u, v) > 0 \Rightarrow s \rightsquigarrow u \rightarrow v$ in G_f Blitz $v \in T$

Sei (u, v) mit $u \in T$ und $v \in S$, dann gilt: $f(u, v) > 0$ $f(u, v) > 0 \Rightarrow (v, u) \in E_f$ mit $c_f(v, u) = f(u, v) > 0 \Rightarrow S \rightsquigarrow v \rightarrow u$ in G_f Blitz $u \in T$

22.1.1 Laufzeit FF-Algo

Ann. $c(u, v) \in \mathbb{N} \forall (u, v) \in E$

$$C = \sum_{v \in V} c(s, v) \in \mathbb{N}$$

Zahl der Iterationen $\leq C$

Laufzeit $\mathcal{O}(C|E|)$

22.2 Edmund-Karp Algo

Idee: Verwende als flussverbessernde Pfade nur solche mit minimaler Kantenzahl. Sei $G_f = (V, E_f)$ das Restnetzwerk mit $c_f(u, v) > 0 \forall (u, v) \in E_f$ $S_f(s, v) =$ Zahl der Kanten auf einem Kürzesten Weg von s nach v im Restnetzwerk G_f. Sei f' der Fluss, der aus f durch eine Flussverbesserung entlang eines Kürzesten Weges in G_f entstanden ist dann gilt: $\delta_{f'}(s, v) \geq \delta_f(s, v)$

Layernetzwerk Bild

Sei (u, v) eine Bottleneck-Kante, die durch die Flussverbesserung saturiert wird.