

Datenstrukturen & Algorithmen

Peppo Brambilla
Universität Bern
Frühling 2018

Übersicht

Greedy Algorithmen

- Einführung (Aktivitäten-Auswahl-Problem)
- Greedy Entwurfsstrategie
- Huffman Codierung

Greedy Algorithmen

- Entwurfsstrategie ähnlich wie dynamische Programmierung
- Optimierungsprobleme
- Grundidee
 - Wenn Entscheidung getroffen werden muss, wähle diejenige, die im Moment am besten aussieht
 - Hoffe (besser: zeige), dass lokal beste Entscheidung zu global optimaler Lösung führt
- Nicht alle Probleme erlauben optimale Lösung durch Greedy Strategie

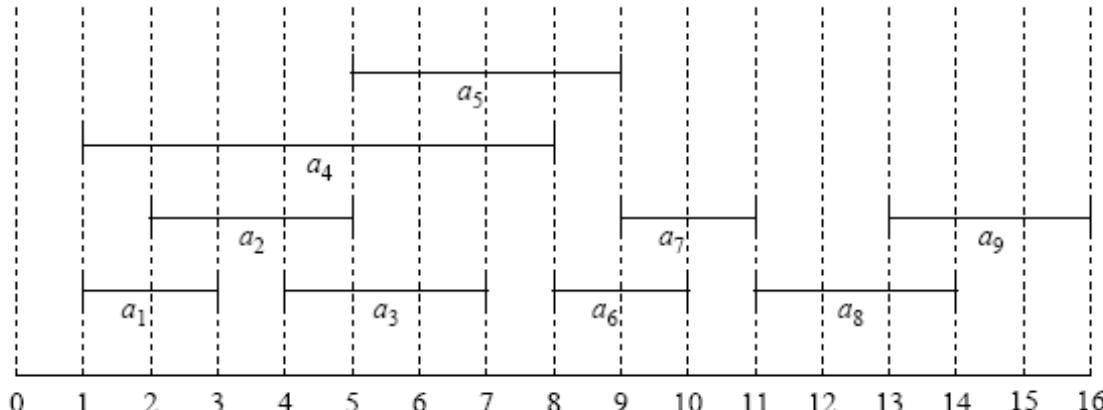
Aktivitäten-Auswahl-Problem

- n Aktivitäten brauchen Resource, die je nur von einer Aktivität gleichzeitig benutzt werden kann
 - Vorlesungen und Hörsaal
- **Ziel:** finde maximale Teilmenge kompatibler (nicht überlappender) Aktivitäten
- Anmerkung: andere Ziele wären auch sinnvoll
 - Maximiere Zeit, über die der Raum belegt ist
 - Maximiere Vermietungsgebühren

Aktivitäten-Auswahl-Problem

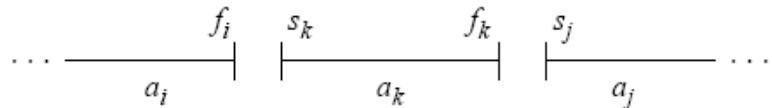
- Menge der Aktivitäten $S = \{a_1, \dots, a_n\}$
- Aktivität a_i beginnt zum Zeitpunkt s_i und endet zum Zeitpunkt f_i . Es gilt $0 \leq s_i < f_i < \infty$.
- Beispiel: S sortiert nach Endzeit

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

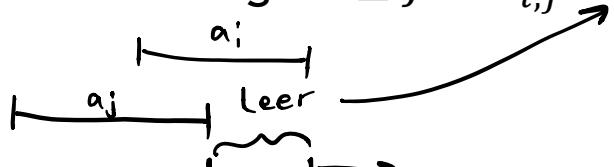


Aktivitäten-Auswahl-Problem

- Menge der Aktivitäten $S = \{a_1, \dots, a_n\}$
- Menge $S_{i,j} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$
 - Aktivitäten, die nach Beendigung von a_i beginnen und vor Beginn von a_j enden



- Aktivitäten in $S_{i,j}$ sind kompatibel mit allen Aktivitäten, die
 - vor f_i enden
 - nach s_j beginnen
- Fiktive Aktivitäten $a_0 = [-\infty, 0)$, $a_{n+1} = [\infty, \infty+1)$
 - Daraus folgt: $S = S_{0,n+1}$
- Annahme: Aktivitäten sortiert $f_0 \leq f_1 \leq \dots \leq f_n \leq f_{n+1}$
 - Daraus folgt: $i \geq j \Rightarrow S_{i,j} = \emptyset$



Optimale Teilstruktur

- Annahme: Lösung von $S_{i,j}$ beinhaltet a_k
- Teilprobleme
 - $S_{i,k}$ (startet nach Ende von a_i , beendet vor Start von a_k)
 - $S_{k,j}$ (startet nach Ende von a_k , beendet vor Start von a_j)
- Falls optimale Lösung von $S_{i,j}$ wirklich a_k beinhaltet, dann beinhaltet es auch optimale Lösungen von $S_{i,k}$ und $S_{k,j}$
 - Beweis durch Widerspruch

Rekursive Lösung

- Sei $c[i, j] =$ Grösse der grössten Teilmenge kompatibler Aktivitäten in $S_{i,j}$
- Rekursionsformel

$$c[i, j] = \begin{cases} 0 & \text{if } S_{i,j} = \emptyset \\ \max_{a_k \in S_{i,j}} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{i,j} \neq \emptyset \end{cases}$$

- Könnten nun Tabelle $c[i, j]$ mit dynamischer Programmierung bottom-up berechnen
- Es geht aber **einfacher!**

Umwandlung in Greedy Lösung

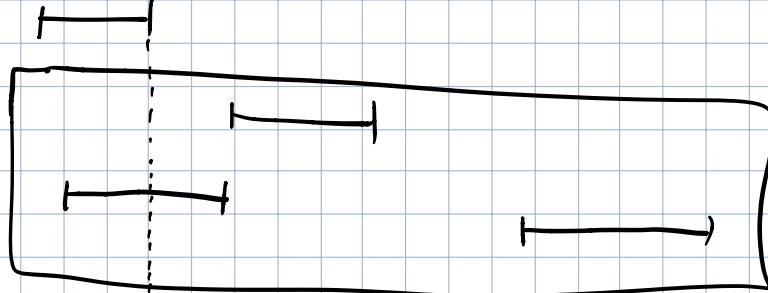
Theorem (**gierige Auswahl Eigenschaft**)

- Gegeben: Nichtleeres Teilproblem $S_{i,j}$, Aktivität a_m mit frühester Endzeit in $S_{i,j}$
- Dann gilt
 1. Aktivität a_m kommt in einer maximalen Teilmenge kompatibler Aktivitäten von $S_{i,j}$ vor
 2. Teilproblem $S_{i,m}$ ist leer, d.h. nach Wahl von a_m muss nur noch das Teilproblem $S_{m,j}$ gelöst werden

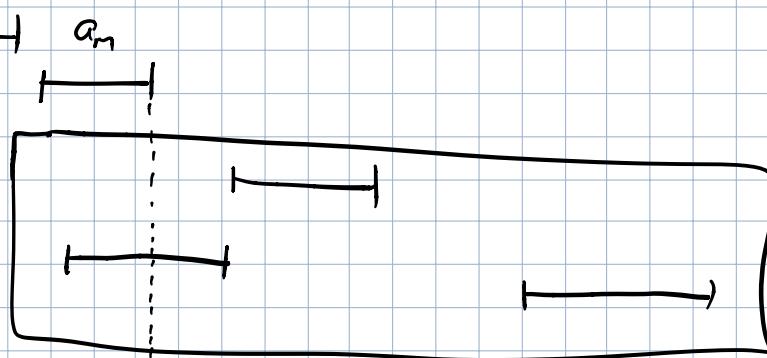
Beweis

1. Gegeben sei eine optimale Lösung von $S_{i,j}$, die a_m nicht enthält. Zeige, dass die erste Aktivität a_k in der optimalen Lösung durch a_m ersetzt werden kann. Dies geht garantiert ohne Konflikt, weil $a_k \in S_{i,j}$, und auch $f_m \leq f_k$.
2. Wenn $S_{i,m}$ nicht leer wäre, gäbe es eine Aktivität a_k in $S_{i,j}$ die vor a_m aufhört. Das widerspricht aber der Annahme, dass a_m die Aktivität mit frühester Endzeit in $S_{i,j}$ ist.

1.)

 $S_{i,j} :$ 

2.)

 $S_{i,n}$ $S_{i,j} :$ 

Greedy Lösung

- Konsequenzen des Theorems

	Dynamisches Programmieren	Greedy gestützt auf Theorem
#Teilprobleme in optimaler Lösung	2	1
#Möglichkeiten zu untersuchen	$j - i - 1$	1

- Problem kann **top-down** gelöst werden
 - **Zuerst**: wähle Aktivität a_m die zuerst endet („gierige“ Wahl)
 - **Dann**: löse Teilproblem $S_{m,j}$

Greedy Lösung

- Iterativer Algorithmus

- Annahme: Aktivitäten a_i nach aufsteigenden Endzeiten sortiert a_1 hört an ersten auf

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1   $n = s.length$ 
2   $A = \{a_1\}$ 
3   $k = 1$           //  $a_k$  ist letzte zu  $A$  hinzugefügte Aktivität
4  for  $m = 2$  to  $n$ 
5    if  $s[m] \geq f[k]$       Beispiel oben:  $a_1$  hinzu,  $a_2$  nicht (Startzeit vor  
Endzeit),
6       $A = A \cup \{a_m\}$            $a_3$  hinzu,  $a_6, a_8$ 
7       $k = m$ 
8  return  $A$ 
```

- Linearer Aufwand $\Theta(n)$

Übersicht

Greedy Algorithmen

- Einführung (Aktivitäten-Auswahl-Problem)
- Greedy Entwurfsstrategie
- Huffman Codierung

Greedy Entwurfsstrategie

Abgeleitet von dynamischer Programmierung

- Zeige optimale Teilstruktur
- Entwickle rekursive Lösung
 - Zeige, dass in jedem Schritt die **Greedy Auswahl** eine **optimale Wahl** ist
 - Zeige, dass die Greedy Auswahl zu einem **einzigem übrigbleibenden Teilproblem** führt
- Leite einen rekursiven oder iterativen Greedy Algorithmus ab

Greedy Entwurfsstrategie

Direkte Strategie

1. Beschreibe Lösung, wo in jedem Schritt **eine greedy Wahl** getroffen wird und **ein Teilproblem** übrigbleibt
2. Zeige, dass die greedy Wahl in einer optimalen Lösung vorkommt
3. Zeige, dass die greedy Wahl **und** eine optimale Lösung des Teilproblems zu einer **optimalen Lösung des Problems** führt

Welche Probleme können mit der greedy Strategie optimal gelöst werden?

- Probleme brauchen
 - **Gierige-Auswahl-Eigenschaft**
 - **Optimale Teilstruktur**

Gierige-Auswahl-Eigenschaft

- „Wir können eine optimale Lösung eines Problems finden, indem wir
 - **lokal** optimale Entscheidungen treffen“
 - eine gierige Auswahl treffen, **ohne** vorher Teilprobleme zu lösen“

Vergleich

- Dynamische Programmierung
 - Optimale Entscheidung in jedem Schritt hängt von optimalen Lösungen von Teilproblemen ab
 - Bottom-up Lösung
- Greedy Algorithmen
 - Treffen gierige Auswahl in jedem Schritt **bevor** das Teilproblem gelöst wird
 - Top-down Lösung

Optimale Teilstruktur

- „Eine optimale Lösung des Problems beinhaltet optimale Lösungen von Teilproblemen“
- Bei Greedy Algorithmen
 - Zeige, dass gierige Auswahl zusammen mit optimaler Lösung des übrigbleibenden Teilproblems zu optimaler Lösung des Problems führt

Vergleich: Rucksackproblem

0-1-Rucksackproblem

- n Gegenstände
- Gegenstand i hat Wert v_i , Gewicht w_i
- Finde wertvollste Teilmenge von Gegenständen, so dass Totalgewicht $\leq W$
- Können keine Bruchteile von Gegenständen einpacken

Fraktionales Rucksackproblem

- Wie oben, aber Bruchteile erlaubt

Eigenschaften

- Beide haben optimale Teilstruktur
- Fraktionales Problem hat gierige Auswahl Eigenschaft, das 0-1-Problem hat sie nicht

Vergleich: Rucksackproblem

Greedy Algorithmus für fraktionales Problem

- Berechne Wert pro Gewicht für jeden Gegenstand
- Fülle Rucksack beginnend mit Gegenständen mit grösstem Wert pro Gewicht, bis voll
- Funktioniert nicht für 0-1-Problem
 - Brauchen dynamische Programmierung

Beispiel

	i	1	2	3				
	v_i	60	100	120	\$120			
	w_i	10	20	30		+		
	v_i/w_i	6	5	4				

$W = 50.$

The diagram illustrates the knapsack problem with three items and a knapsack. Item 1 has weight 10 and value \$60. Item 2 has weight 20 and value \$100. Item 3 has weight 30 and value \$120. The knapsack has a capacity of 50. To the right, the items are listed with their values and weights, followed by their ratios v_i/w_i . Below the table, the total value and weight are calculated for each item added to the knapsack.

Greedy Lösung:

- Nimmt Objekte 1 + 2.
- Wert: 160
Gewicht: 30

Restkapazität: 20 Pfund

Optimale Lösung:

- Nimmt Objekte 2 + 3.
- Wert: 220
Gewicht: 50

Restkapazität: 0 Pfund

Übersicht

Greedy Algorithmen

- Einführung (Aktivitäten-Auswahl-Problem)
- Greedy Entwurfsstrategie
- Huffman Codierung

Huffman Codierung

- Greedy Algorithmus zur Datenkompression
- Gegeben
 - Daten bestehend aus Zeichenketten
 - Für jedes Zeichen die Häufigkeit, mit der es vorkommt
- Gesucht: **optimaler Binärcode** für jedes Zeichen
 - Variable Länge für jedes Zeichen
 - Code führt zu kleinster totaler Anzahl Bits, um Zeichenkette darzustellen
- Idee
 - Binärcodes für **häufige** Zeichen sind **kürzer** als Codes für seltene Zeichen

Beispiel

- File mit 100'000 Zeichen
- Nur 6 verschiedene Zeichen a,b,c,d,e,f

	a	b	c	d	e	f
Häufigkeit (in Tausend)	45	13	12	16	9	5
Codewort fester Länge	000	001	010	011	100	101
Codewort variabler Länge	0	101	100	111	1101	1100

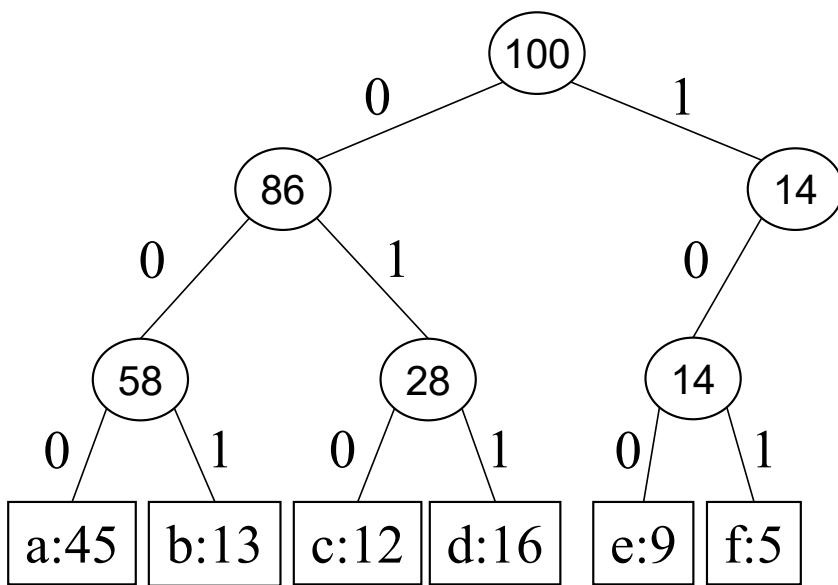
- Total Bits
 - Feste Länge: $100'000 * 3 = 300'000$ Bits
 - Variable Länge:
 $(45*1+13*3+12*3+16*3+9*4+5*4)=224'000$ Bits

Präfix-Codes

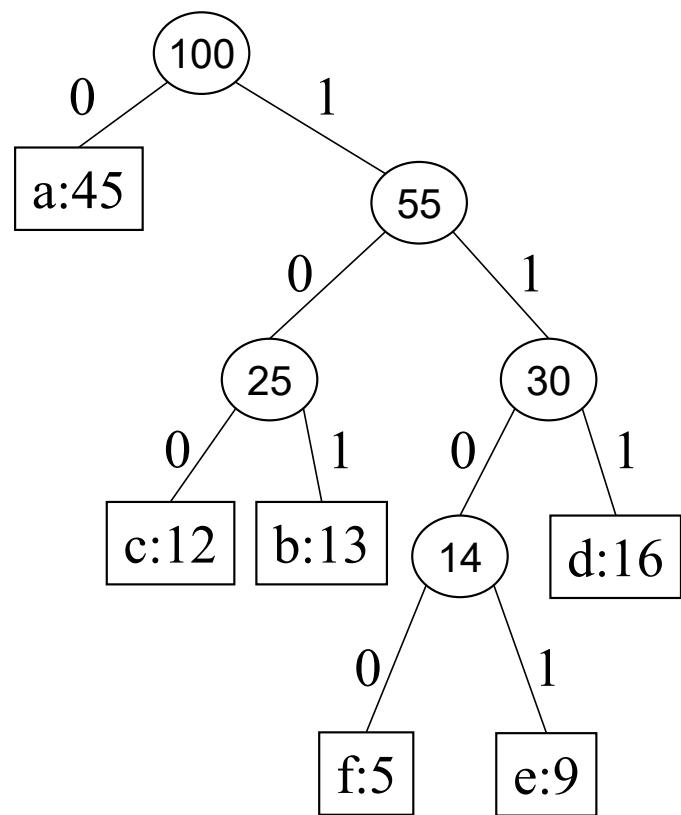
- Präfix-Code: kein Codewort ist gleichzeitig Präfix eines anderen Codewortes
 - Beispiel: zwei Zeichencodes $a = 101$ und $b = 10$ können zusammen in einem Präfix-Code nicht vorkommen
- Kann zeigen: optimale Datenkomprimierung immer mit Präfixcode möglich
- **Codierung:** Konkatenation der binären Zeichencodes
- **Decodierung**
 - Einfach, da kein Codewort Präfix eines anderen
 - Erstes Codewort im codierten File ist eindeutig

Darstellung als Binärbaum

Feste Codelänge



Optimaler Präfix-Code



Eigenschaften

- Optimaler Code entspricht immer einem vollen binären Baum
 - Jeder innere Knoten hat zwei Kinder
- Sei
 - C das Alphabet (Menge der möglichen Zeichen)
 - $|C|$ Anzahl verschiedener Zeichen
- Baum zum optimalen Code hat
 - $|C|$ Blätter
 - $|C| - 1$ innere Knoten
- Totale Anzahl Bits B eines Baumes T
 - $c.freq$ Häufigkeit von Zeichen c
 - $d_T(c)$ Tiefe von Zeichen c in Baum T

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$$

Huffman-Codierung

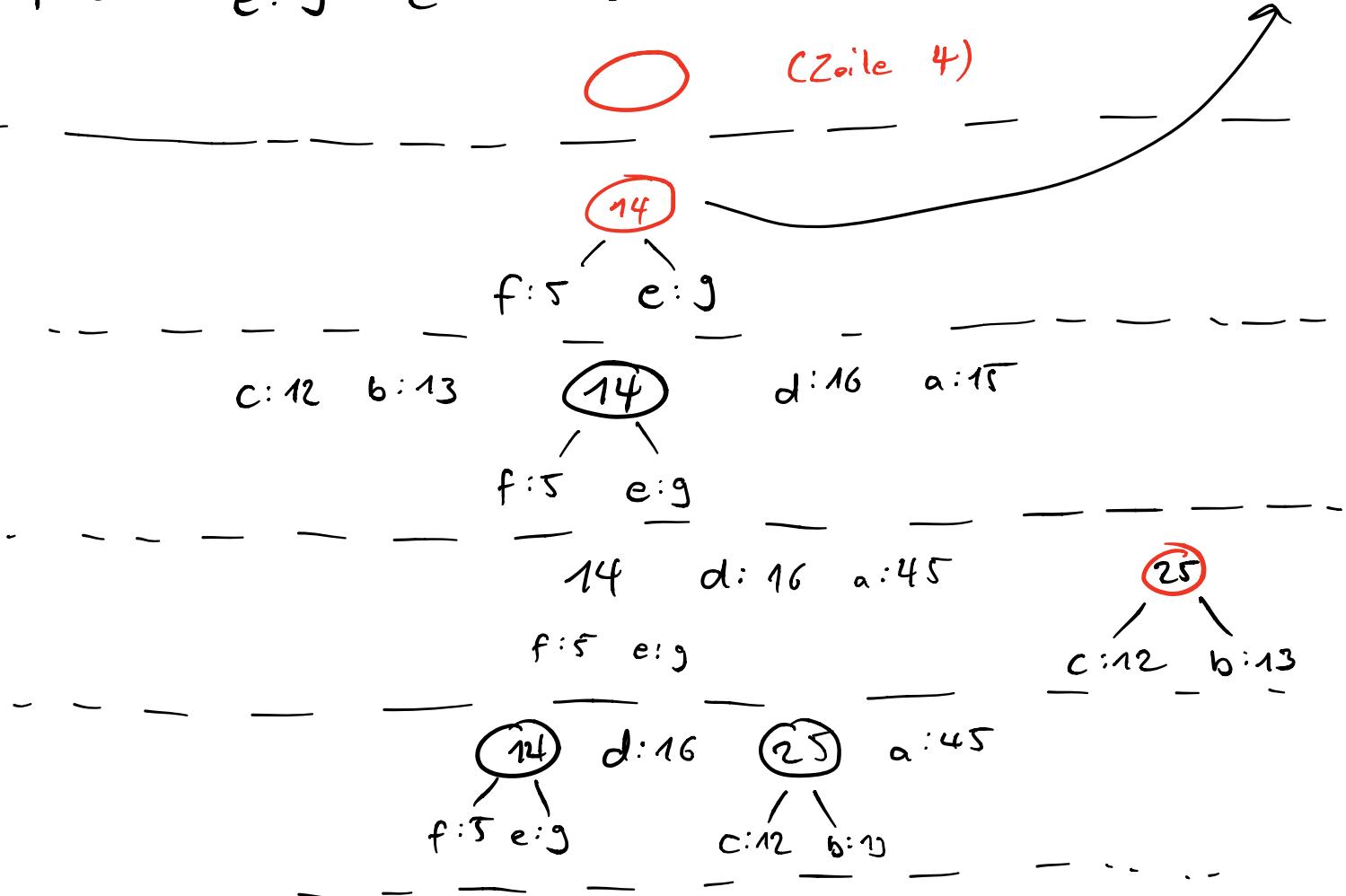
- Greedy Algorithmus
 - Anzahl verschiedener Zeichen n
 - Nach Häufigkeiten sortierte Min-Prioritäts-Warteschlange Q

$\text{HUFFMAN}(C)$

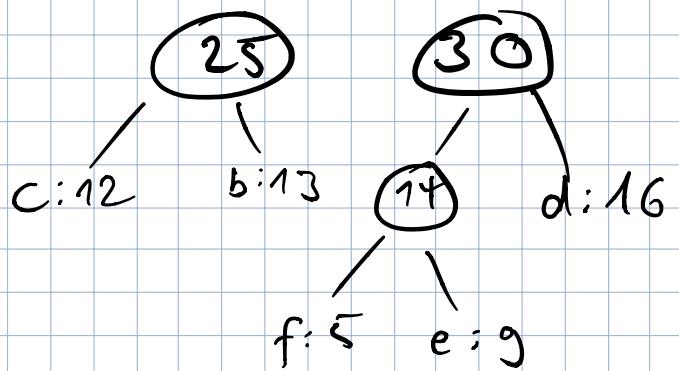
```
1   $n = |C|$ 
2   $Q = C$ 
3  for  $m = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $z.left = x = \text{EXTRACT-MIN}(Q)$ 
6       $z.right = y = \text{EXTRACT-MIN}(Q)$ 
7       $z.freq = x.freq + y.freq$ 
8       $\text{INSERT}(Q, z)$ 
9  return  $\text{EXTRACT-MIN}(Q)$  // return the root of the tree
```

Huffman-Codierung

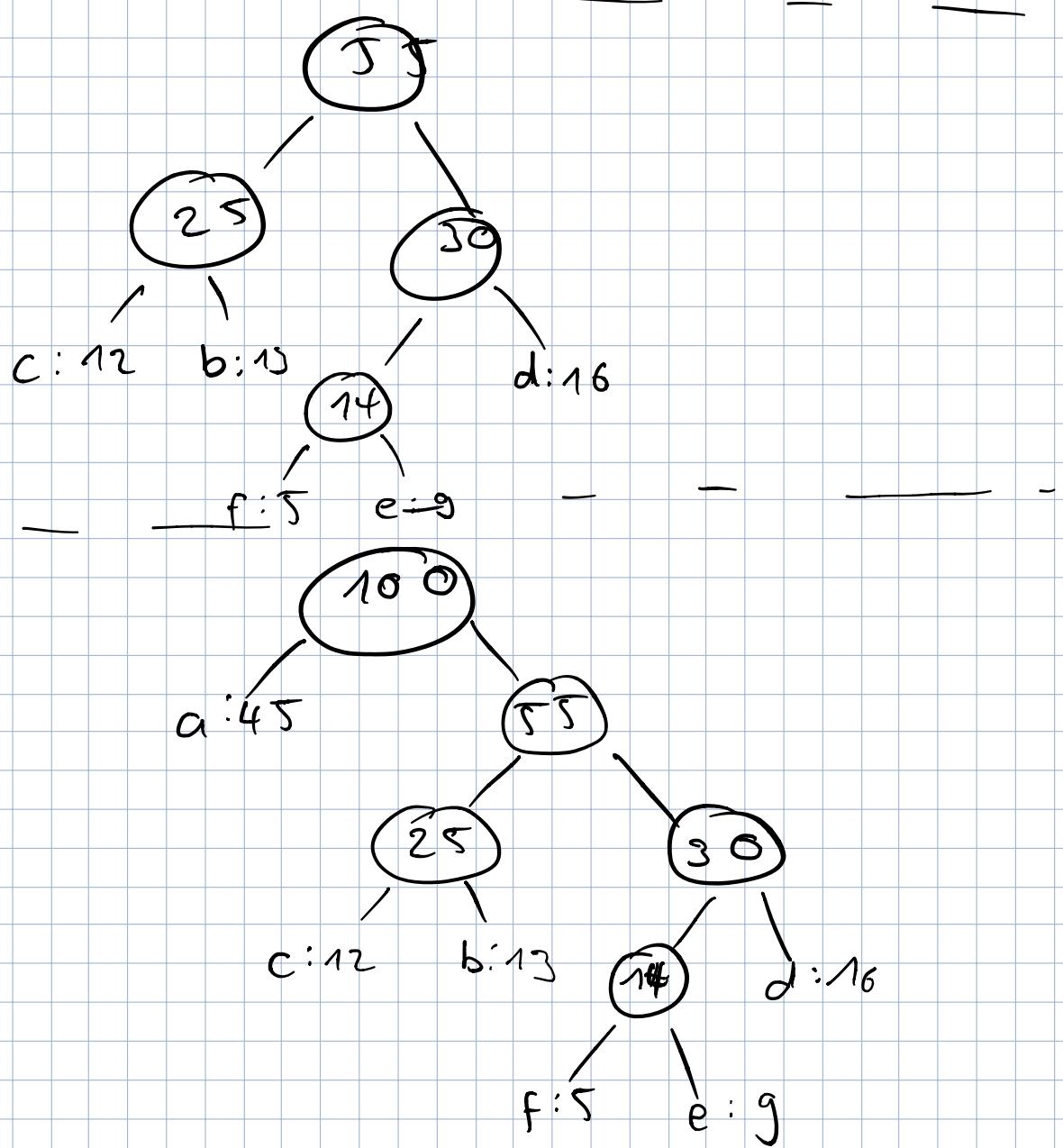
f: 5 e: 9 c: 12 b: 13 d: 13 a: 45



14 d:16 25 a:45 30



a:45



Korrektheit

- Beweise **gierige Auswahl-Eigenschaft** und optimale Teilstruktur

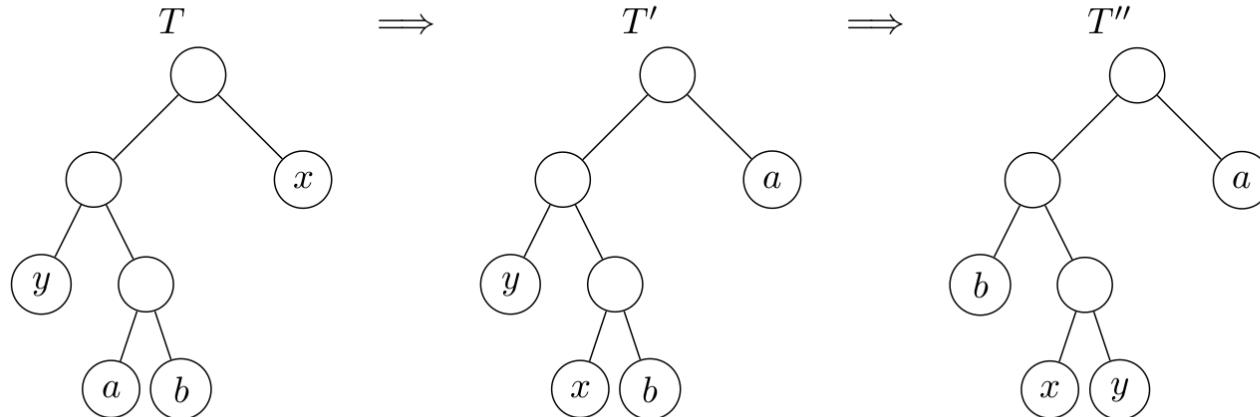
Lemma (gierige Auswahl-Eigenschaft)

- Gegeben
 - C ein Alphabet, $c.freq$ Häufigkeiten der Zeichen
 - x und y Zeichen mit niedrigsten Häufigkeiten
- Behauptung: es existiert ein optimaler Präfix-Code, in dem die Codewörter für x und y gleiche Länge haben und sich nur im letzten Bit unterscheiden
 - Codewörter für x und y sind Geschwister in der Baumrepräsentation

Beweis

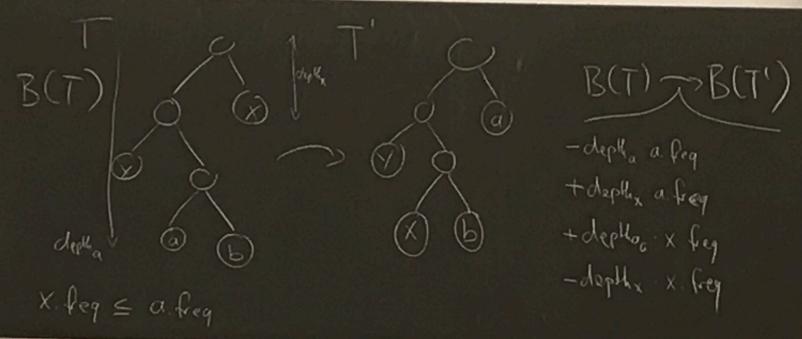
Idee: Sei T Baum, der optimalen Code darstellt. Modifiziere T so, dass neuer Baum wiederum optimalen Code darstellt und x und y Geschwister sind.

Ablauf: Wähle Blätter a und b maximaler Tiefe in T und vertausche sie mit x und y .



Es gilt: $B(T'') \leq B(T)$, da x und y die Blätter mit der kleinsten Häufigkeit sind (s. unten). Weil aber T optimal ist, muss $B(T'') = B(T)$. Somit ist auch T'' optimal.

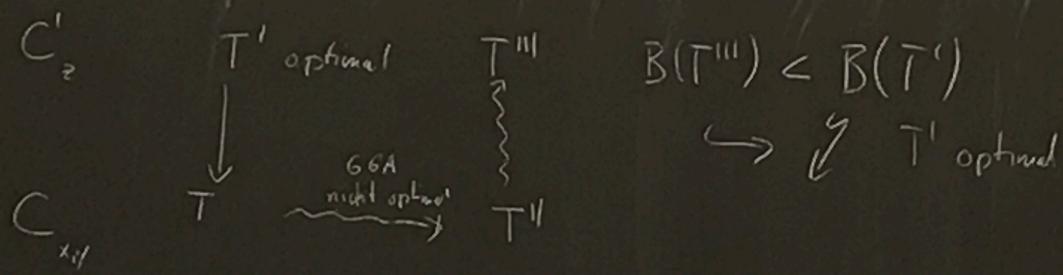
$$\begin{aligned}
 B(T') &= B(T) - a.freq \cdot d_T(a) + a.freq \cdot d_T(x) - x.freq \cdot d_T(x) + x.freq \cdot d_T(a) \\
 &= B(T) - a.freq(d_T(a) - d_T(x)) + x.freq(d_T(a) - d_T(x)) \\
 &= B(T) - \underbrace{(a.freq - x.freq)}_{\geq 0} \cdot \underbrace{(d_T(a) - d_T(x))}_{\geq 0} \\
 &\quad \underbrace{_{\geq 0}}
 \end{aligned}$$



Korrektheit

Lemma (optimale Teilstruktur)

- Gegeben
 - C ein Alphabet, $c.freq$ Häufigkeiten der Zeichen
 - x und y Zeichen mit niedrigsten Häufigkeiten
- Sei
 - $C' = (C \setminus \{x, y\}) \cup \{z\}$
 - $z.freq = x.freq + y.freq$
 - T' Baum, der optimalen Präfix-Code für C' darstellt
- Behauptung: T ist optimaler Präfix Code für C , wenn wir T aus T' erhalten, indem wir z durch internen Knoten ersetzen, der x und y als Kinder besitzt



Beweis

Zuerst: Berechne $B(T)$ aus $B(T')$:

1. Für alle $c \in C \setminus \{x, y\}$ gilt $d_T(c) = d_{T'}(c)$.

Also $c.freq \cdot d_T(c) = c.freq \cdot d_{T'}(c)$

T' ist der Baum, der aus T entsteht, indem x und y durch Knoten z ersetzt werden. $B(T)$ und $B(T')$ unterscheiden sich also nur in den Summanden bezüglich x, y und z . Das heisst

$$B(T) = B(T') - z.freq \cdot d_{T'}(z) + \underline{x.freq \cdot d_T(x) + y.freq \cdot d_T(y)} \quad (1)$$

2. Weil $d_T(x) = d_T(y) = d_{T'}(z) + 1$ gilt

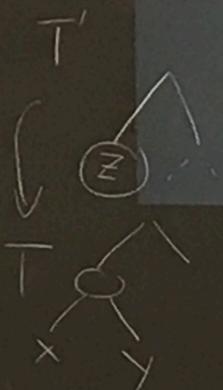
$$\begin{aligned} \underline{x.freq \cdot d_T(x) + y.freq \cdot d_T(y)} &= (x.freq + y.freq) \cdot (d_{T'}(z) + 1) \\ &= \underbrace{(x.freq + y.freq)}_{z.freq} \cdot d_{T'}(z) + x.freq + y.freq \\ &= \underline{z.freq \cdot d_{T'}(z) + x.freq + y.freq} \end{aligned}$$

Aus (1) folgt damit: $B(T) = B(T') + x.freq + y.freq$.

- z. Jreq = x.Jreq + y.Jreq

- T' Baum, der optimalen Präfix-Code für C darstellt

- Behauptung: T ist optimaler Präfix Cod



T , wenn wir T aus T' erhalten, indem wir durch internen Knoten ersetzen, der x als Kinder besitzt

a : 45

$$B(T') = B(T) - x.freq - y.freq$$

$$B(T) = -\frac{1}{P} \sum p_i f_i \log_2 f_i$$

Beweis

Sei T' optimal, zeige T optimal.

Gegenannahme: T sei nicht optimal. Dann gibt es T'' mit $B(T'') < B(T)$

- Wegen des vorhergehenden Lemmas dürfen wir annehmen, dass x und y in T'' als Geschwister vorkommen.
- Konstruiere T''' aus T'' indem x und y durch z ersetzt werden mit $z.freq = x.freq + y.freq$. Dann gilt

$$\begin{aligned}B(T''') &= B(T'') - x.freq - y.freq \\&< B(T) - x.freq - y.freq \\&= B(T')\end{aligned}$$

Dies ist ein Widerspruch dazu, dass T' optimal ist. Also stimmt die Gegenannahme nicht und T ist demzufolge optimal.

Zusammenfassung

- Huffman Algorithmus erzeugt optimalen Präfix-Code
- Beweis
 - Zwei Lemmata von vorher: Das Problem hat Gierige-Auswahl-Eigenschaft und optimale Teilstruktur
 - Huffman Algorithmus trifft in jedem Schritt gierige Auswahl

Nächstes Mal

- Graphenalgorithmen