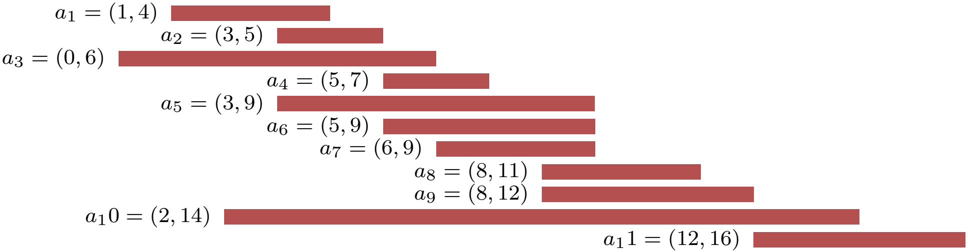
21. Greedy Algorithmen

Aktivitätenauswahl, Fractional Knapsack Problem, Huffman Coding
Cormen et al, Kap. 16.1, 16.3

Aktivitäten Auswahl

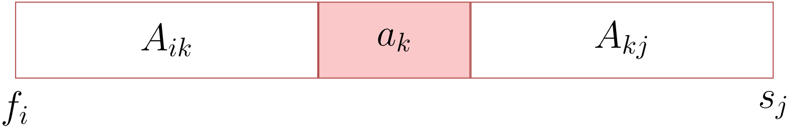
Koordination von Aktivitäten, die gemeinsame Resource exklusiv nutzen. Aktivitäten $S = \{a_1, a_2, \ldots, a_n\}$ mit Start und Endzeiten $0 \leq s_i \leq f_i < \infty$, aufsteigend sortiert nach Endzeiten.



Aktivitäten-Auswahl-Problem: Finde maximale Teilmenge kompatibler (nichtüberlappender) Aktivitäten.

Dynamic Programming Ansatz?

Sei $S_{ij} = \{a_k : f_i \leq s_k \wedge f_k \leq s_j\}$. Sei A_{ij} eine maximale Teilmenge kompatibler Aktivitäten aus S_{ij}. Sei ausserdem $a_k \in A_{ij}$ und $A_{ik} = S_{ik} \cap A_{ij}$, $A_{ki} = S_{kj} \cap A_{ij}$, also $A_{ij} = A_{ik} + \{a_k\} + A_{kj}$.



Klar: A_{ik} und A_{kj} müssen maximal sein, sonst wäre $A_{ij} = A_{ik} + \{a_k\} + A_{kj}$ nicht maximal.

Dynamic Programming Ansatz?

Sei $c_{ij} = |A_{ij}|$. Dann gilt folgende Rekursion $c_{ij} = c_{ik} + c_{kj} + 1$, also

$$c_{ij} = \begin{cases} 0 & \text{falls } S_{ij} = \emptyset, \\ \max_{a_k \in S_{ij}} \{c_{ik} + c_{kj} + 1\} & \text{falls } S_{ij} \neq \emptyset. \end{cases}$$

Könnten nun dynamische Programmierung versuchen.

Greedy

Intuition: Wähle zuerst die Aktivität, die die früheste Endzeit hat (a_1). Das lässt maximal viel Platz für weitere Aktivitäten.

Verbleibendes Teilproblem: Aktivitäten, die starten nachdem a_1 endet. (Es gibt keine Aktivitäten, die vor dem Start von a_1 enden.)

Greedy

Theorem

Gegeben: Teilproblem S_k, a_m eine Aktivität aus S_k mit frühester Endzeit. Dann ist a_m in einer maximalen Teilmenge von kompatiblen Aktivitäten aus S_k enthalten.

Sei A_k maximal grosse Teilmenge mit kompatiblen Aktivitäten aus S_k und a_j eine Aktivität aus A_k mit frühester Endzeit. Wenn $a_j = a_m$ $\Rightarrow$ fertig. Wenn $a_j \neq a_m$. Dann betrachte $A'_k = A_k - \{a_j\} \cup \{a_m\}$. Dann besteht A'_k aus kompatiblen Aktivitäten und ist auch maximal, denn $|A'_k| = |A_k|$.

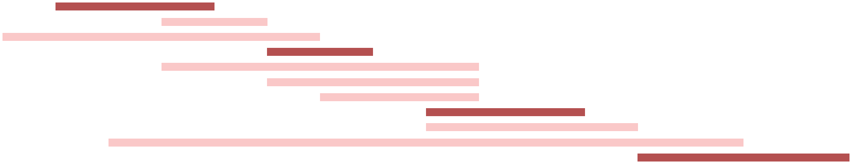
$\blacksquare$

Algorithmus RecursiveActivitySelect(s, f, k, n)

Input : Folge von Start und Endzeiten (s_i, f_i), $1 \le i \le n$, $s_i < f_i$, $f_i \le f_{i+1}$ für alle i. $1 \le k \le n$

Output : Maximale Menge kompatibler Aktivitäten.

$m \leftarrow k + 1$
while $m \le n$ and $s_m \le f_k$ **do**
  $m \leftarrow m + 1$
if $m \le n$ **then**
  return $\{a_m\} \cup$ RecursiveActivitySelect(s, f, m, n)
else
  return $\emptyset$



Algorithmus IterativeActivitySelect(s, f, n)

Input : Folge von Start und Endzeiten (s_i, f_i), $1 \leq i \leq n$, $s_i < f_i$, $f_i \leq f_{i+1}$ für alle i.

Output : Maximale Menge kompatibler Aktivitäten.

$A \leftarrow \{a_1\}$
$k \leftarrow 1$
for $m \leftarrow 2$ **to** n **do**
    if $s_m \geq f_k$ **then**
        $A \leftarrow A \cup \{a_m\}$
        $k \leftarrow m$

return A

Laufzeit beider Algorithmen: $\Theta(n)$

Das Gebrochene Rucksackproblem

Menge von $n \in \mathbb{N}$ Gegenständen $\{1, \ldots, n\}$ gegeben. Jeder Gegenstand i hat Nutzwert $v_i \in \mathbb{N}$ und Gewicht $w_i \in \mathbb{N}$. Das Maximalgewicht ist gegeben als $W \in \mathbb{N}$. Bezeichnen die Eingabe mit $E = (v_i, w_i)_{i=1,\ldots,n}$.

Gesucht: Anteile $0 \leq q_i \leq 1$ ($1 \leq i \leq n$) die die Summe $\sum_{i=1}^{n} q_i \cdot v_i$ maximieren unter $\sum_{i=1}^{n} q_i \cdot w_i \leq W$.

Gierige Heuristik

Sortiere die Gegenstände absteigend nach Nutzen pro Gewicht v_i/w_i.

Annahme $v_i/w_i \geq v_{i+1}/w_{i+1}$

Sei $j = \max\{0 \leq k \leq n : \sum_{i=1}^{k} w_i \leq W\}$. Setze

- $q_i = 1$ für alle $1 \leq i \leq j$.
- $q_{j+1} = \frac{W - \sum_{i=1}^{j} w_i}{w_{j+1}}$.
- $q_i = 0$ für alle $i > j + 1$.

Das ist schnell: $\Theta(n \log n)$ für Sortieren und $\Theta(n)$ für die Berechnung der q_i.

Korrektheit

Annahme: Optimale Lösung (r_i) ($1 \le i \le n$).

Der Rucksack wird immer ganz gefüllt: $\sum_i r_i \cdot w_i = \sum_i q_i \cdot w_i = W$.

Betrachte k: kleinstes i mit $r_i \neq q_i$. Die gierige Heuristik nimmt per Definition so viel wie möglich: $q_k > r_k$. Sei $x = q_k - r_k > 0$.

Konstruiere eine neue Lösung (r'_i): $r'_i = r_i \forall i < k$. $r'_k = q_k$. Entferne Gewicht $\sum_{i=k+1}^{n} \delta_i = x \cdot w_k$ von den Gegenständen $k+1$ bis n. Das geht, denn $\sum_{i=k}^{n} r_i \cdot w_i = \sum_{i=k}^{n} q_i \cdot w_i$.

Korrektheit

$$
\begin{aligned}
\sum_{i=k}^{n} r'_i v_i &= r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} (r_i w_i - \delta_i) \frac{v_i}{w_i} \\
&\geq r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} r_i w_i \frac{v_i}{w_i} - \delta_i \frac{v_k}{w_k} \\
&= r_k v_k + x w_k \frac{v_k}{w_k} - x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} r_i w_i \frac{v_i}{w_i} = \sum_{i=k}^{n} r_i v_i .
\end{aligned}
$$

Also ist (r'_i) auch optimal. Iterative Anwendung dieser Idee erzeugt die Lösung (q_i).

Huffman-Codierungen

Ziel: Speicherplatzeffizientes Speichern einer Folge von Zeichen mit einem binären *Zeichencode* aus *Codewörtern*.

Beispiel

File aus 100.000 Buchstaben aus dem Alphabet $\{a, \ldots, f\}$

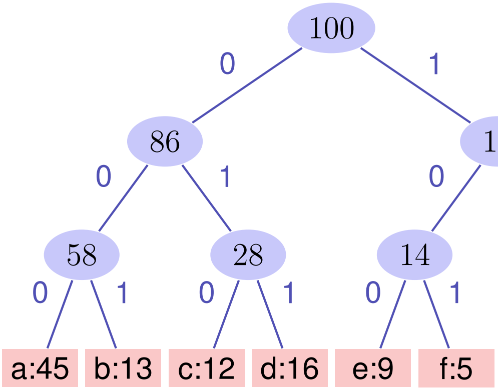
	a	b	c	d	e	f
Häufigkeit (Tausend)	45	13	12	16	9	5
Codewort fester Länge	000	001	010	011	100	101
Codewort variabler Länge	0	101	100	111	1101	1100

Speichergrösse (Code fixe Länge): 300.000 bits.
Speichergrösse (Code variabler Länge): 224.000 bits.

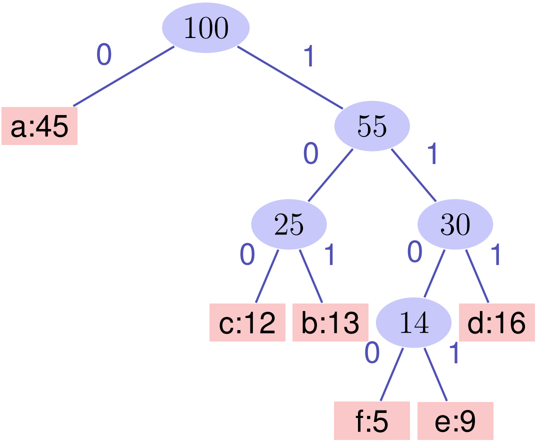
Huffman-Codierungen

- Betrachten *Präfixcodes*: kein Codewort kann mit einem anderen Codewort beginnen.
- Präfixcodes können im Vergleich mit allen Codes die optimale *Datenkompression* erreichen (hier ohne Beweis).
- Codierung: Verkettung der Codewörter ohne Zwischenzeichen (Unterschied zum Morsen!)
  $affe \rightarrow 0 \cdot 1100 \cdot 1100 \cdot 1101 \rightarrow 0110011001101$
- Decodierung einfach da Präfixcode
  $0110011001101 \rightarrow 0 \cdot 1100 \cdot 1100 \cdot 1101 \rightarrow affe$

Codebäume



Codewörter fixer Länge



Codewörter variabler Länge

Eigenschaften der Codebäume

- Optimale Codierung eines Files wird immer durch vollständigen binären Baum dargestellt: jeder innere Knoten hat zwei Kinder.
- Sei C die Menge der Codewörter, $f(c)$ die Häufigkeit eines Codeworts c und $d_T(c)$ die Tiefe eines Wortes im Baum T. Definieren die *Kosten* eines Baumes als

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

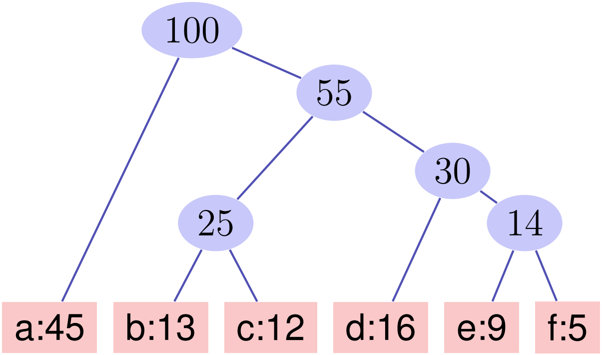
(Kosten = Anzahl Bits des codierten Files)

Bezeichnen im folgenden einen Codebaum als optimal, wenn er die Kosten minimiert.

Algorithmus Idee

Baum Konstruktion von unten nach oben

- Starte mit der Menge C der Codewörter
- Ersetze iterativ die beiden Knoten mit kleinster Häufigkeit durch ihren neuen Vaterknoten.



Algorithmus Huffman(C)

Input : Codewörter $c \in C$
Output : Wurzel eines optimalen Codebaums

```
n ← |C|
Q ← C
for i = 1 to n − 1 do
    Alloziere neuen Knoten z
    z.left ← ExtractMin(Q)              // Extrahiere Wort mit minimaler Häufigkeit.
    z.right ← ExtractMin(Q)
    z.freq ← z.left.freq + z.right.freq
    Insert(Q, z)
return ExtractMin(Q)
```

Analyse

Verwendung eines Heaps: Heap bauen in $\mathcal{O}(n)$. Extract-Min in $O(\log n)$ für n Elemente. Somit Laufzeit $O(n \log n)$.

Das gierige Verfahren ist korrekt

Theorem

Seien x, y *zwei Symbole mit kleinsten Frequenzen in* C *und sei* $T'(C')$ *der optimale Baum zum Alphabet* $C' = C - \{x, y\} + \{z\}$ *mit neuem Symbol* z *mit* $f(z) = f(x) + f(y)$. *Dann ist der Baum* $T(C)$ *der aus* $T'(C')$ *entsteht, indem der Knoten* z *durch einen inneren Knoten mit Kindern* x *und* y *ersetzt wird, ein optimaler Codebaum zum Alphabet* C.

Beweis

Es gilt $f(x) \cdot d_T(x) + f(y) \cdot d_T(y) = (f(x) + f(y)) \cdot (d_{T'}(z) + 1) = f(z) \cdot d_{T'}(x) + f(x) + f(y)$. Also $B(T') = B(T) - f(x) - f(y)$.

Annahme: T sei nicht optimal. Dann existiert ein optimaler Baum T'' mit $B(T'') < B(T)$. Annahme: x und y Brüder in T''. T''' sei der Baum T'' in dem der innere Knoten mit Kindern x und y gegen z getauscht wird. Dann gilt
$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$.
Widerspruch zur Optimalität von T'.

Die Annahme, dass x und y Brüder sind in T'' kann man rechtfertigen, da ein Tausch der Elemente mit kleinster Häufigkeit auf die unterste Ebene den Wert von B höchstens verkleinern kann.