22. Dynamic Programming III

Optimale Suchbäume [Ottman/Widmayer, Kap. 5.7]

22.1 Optimale Suchbäume

Optimale binäre Suchbäume

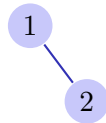
Gegeben: n Schlüssel $k_1, k_2 \ldots k_n$ (oBdA $k_1 < k_2 < ... < k_n$) mit Gewichten (Suchwahrscheinlichkeiten[36]) $p_1, p_2, \ldots, p_n$.
Gesucht: Optimaler Suchbaum T mit Schlüsseltiefen[37] $\mathrm{d}(\cdot)$, welcher die erwarteten Suchkosten

$$C(T) = \sum_{i=1}^{n} (\mathrm{d}(k_i) + 1) \cdot p_i$$

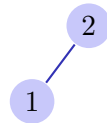
minimiert.

[36] Man kann auch zusätzlich die erfolglose Suche modellieren, hier ausgelassen
[37] $d(k)$: Länge des Pfades von der Wurzel zum Knoten k

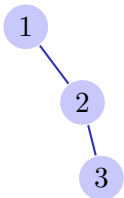
Beispiele



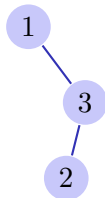
$2p_1 + p_2$



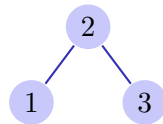
$p_1 + 2p_2$



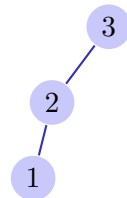
$p_1 + 2p_2 + 3p_3$



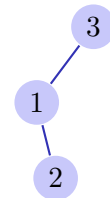
$p_1 + 3p_2 + 2p_2$



$2p_1 + p_2 + 2p_3$



$3p_1 + 2p_2 + p_3$

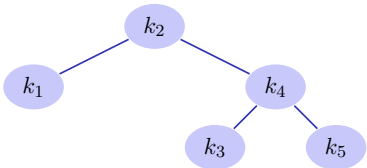


$2p_1 + 3p_3 + 1p_3$

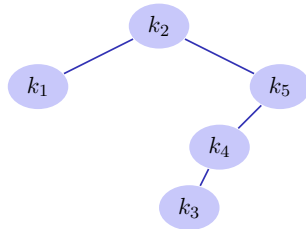
Beispiel

Erwartete Häufigkeiten

i	1	2	3	4	5
p_i	0.25	0.10	0.05	0.20	0.40

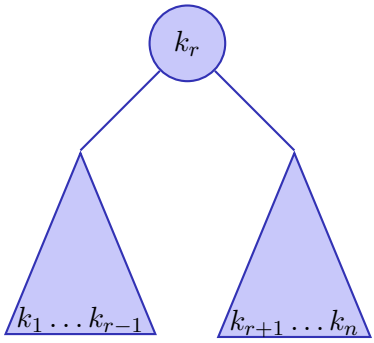


Suchbaum mit erwarteten Kosten 2.35



Suchbaum mit erwarteten Kosten 2.2

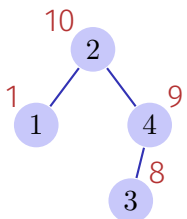
Teilsuchbäume



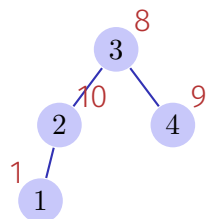
Welches r wählt man?

Greedy?

Szenario $p_1 = 1, p_2 = 10, p_3 = 8, p_4 = 9$



$c(T) = 54$



$c(T) = 49$

Struktur eines optimalen Suchbaumes

- Betrachten aller Teilsuchbäume mit Wurzel k_r, $i \leq r \leq j$ und optimalen Teilbäumen $k_i, \ldots, k_{r-1}$ und $k_{r+1}, \ldots, k_j$
- Teilsuchbäume mit Schlüsseln $k_i, \ldots, k_{r-1}$ und $k_{r+1}, \ldots, k_j$ müssen für die entsprechenden Teilprobleme optimal sein.[38]

$$E(i, j) = \text{Kosten optimaler Suchbaum mit Knoten } k_i, k_{i+1}, \ldots, k_j$$

[38] Das übliche Argument: wäre er nicht optimal, könnte er durch eine bessere Lösung ersetzt werden, welche die Gesamtlösung verbessert.

Rekursion

Mit

$$p(i,j) := p_i + p_{i+1} + \cdots + p_j \qquad i \leq j$$

gilt

$$E(i,j) = \begin{cases} 0 & \text{falls } i > j \\ p(i) & \text{falls } i = j \\ p(i,j) + \min\{E(i,k-1) + E(k+1,j), i \leq k \leq j\} & \text{sonst.} \end{cases}$$

DP

0. $E(1,n)$: Kosten optimaler Suchbaum mit Knoten $k_1,\ldots,k_n$ mit Suchfrequenzen $p_1,\ldots,p_n$
1. $E(i,j), 1 \leq i \leq j \leq n$ # Teilprobleme $\Theta(n^2)$
2. Aufzählen: Wurzeln des Teilsuchbaumes von $k_i,\ldots,k_j$, # Möglichkeiten: $j-i+1$
3. Abhängigkeiten $E(i,j)$ hängen ab von $E(i,k), E(k,j)$ $i<k<j$. Berechnung der Nebendiagonalen von E, ausgehend von der Hauptdiagonalen von E
4. Lösung steht in $E(1,n)$, Rekonstruktion: Speichere die Argmins der Rekursion in einer separaten Tabelle V.
5. Laufzeit $\Theta(n^3)$. Speicherplatz $\Theta(n^2)$.

Beispiel

i	1	2	3	4	5
p_i	0.25	0.10	0.05	0.20	0.40

E

i \ j	0	1	2	3	4	5
1	0	0.25	0.45	0.60	1.15	2.00
2		0	0.10	0.20	0.55	1.30
3			0	0.05	0.30	0.95
4				0	0.20	0.80
5					0	0.40
6						0

p

i \ j	1	2	3	4	5
1	0.25	0.35	0.40	0.60	1.00
2		0.10	0.15	0.35	0.75
3			0.05	0.25	0.65
4				0.20	0.60
5					0.40

V

i \ j	1	2	3	4	5
1	1	1	1	1	4
2		2	2	4	5
3			3	4	5
4				4	5
5					5

23. Gierige (Greedy) Algorithmen

Gebrochenes Rucksack Problem, Huffman Coding [Cormen et al, Kap. 16.1, 16.3]

Gierige Auswahl

Ein rekursiv lösbares Optimierungsproblem kann mit einem **gierigen (greedy) Algorithmus** gelöst werden, wenn es die folgende Eigenschaften hat:

- Das Problem hat **optimale Substruktur**: die Lösung eines Problems ergibt sich durch Kombination optimaler Teillösungen.
- Es gilt die **greedy choice property**: Die Lösung eines Problems kann konstruiert werden, indem ein lokales Kriterium herangezogen wird, welches nicht von der Lösung der Teilprobleme abhängt.

Beispiele: Gebrochenes Rucksackproblem, Huffman-Coding (s.u.)
Gegenbeispiele: Rucksackproblem. Optimaler binärer Suchbaum.

Huffman-Codierungen

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

Beispiel

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

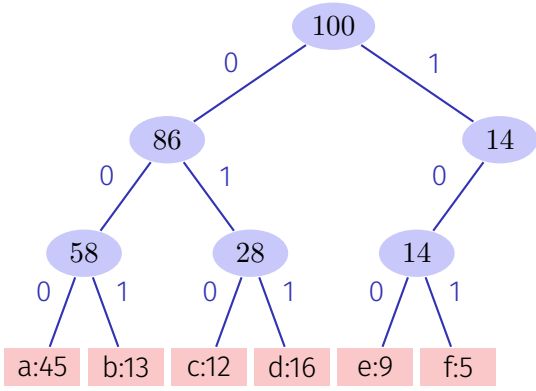
	a	b	c	d	e	f
Häufigkeit (Tausend)	45	13	12	16	9	5
Codewort fester Länge	000	001	010	011	100	101
Codewort variabler Länge	0	101	100	111	1101	1100

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

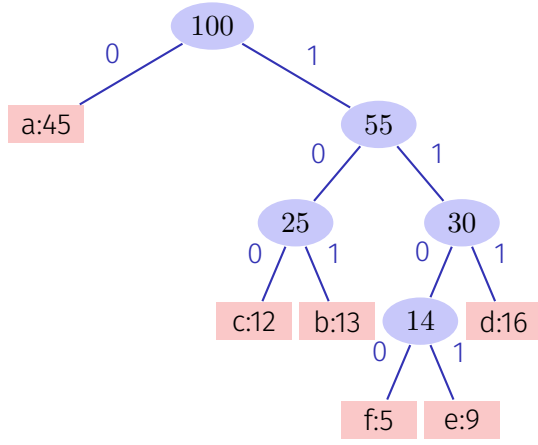
Huffman-Codierungen

- 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

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

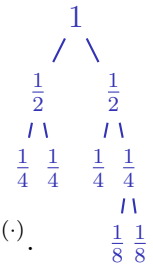
  (Kosten = Anzahl Bits des codierten Files)

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

Wahrscheinlichkeitsverteilungen

Die zu minimierende Summe

$$\sum_{c \in C} f(c) \cdot d_T(c)$$



kann geschrieben werden als

$$-\sum_{c \in C} f(c) \cdot \log_2 g_T(c), \text{ wobei } g_T(\cdot) := 2^{-d_T(\cdot)}.$$

$g_T(\cdot)$ kann als diskrete Wahrscheinlichkeitsverteilung aufgefasst werden, denn es gilt stets $\sum_c g_T(c) = 1$. Das ist eine Eigenschaft eines vollständigen binären Baumes (da an jedem inneren Knoten zwei Kindsknoten haften).

Wahrscheinlichkeitsverteilungen

Für zwei diskrete Wahrscheinlichkeitsverteilungen f und g über C gilt die **Gibbs'sche Ungleichung**

$$\underbrace{-\sum_{c\in C} f(c)\log f(c)}_{\text{Entropie von } f} \leq -\sum_{c\in C} f(x)\log g(c)$$

mit Gleichheit genau dann wenn $f(c) = g(c)$ für alle $c \in C$.

Folgerung: Wenn $f(c) \in \{2^{-k}, k \in \mathbb{N}\}$ für alle $c \in C$, dann kann der optimale Codebaum einfach gebildet werden mit $d_T(c) = -\log_2 f(c)$.

Shannon Fano Coding

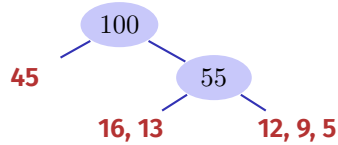
Approximativer Algorithmus von Shannon and Fano:

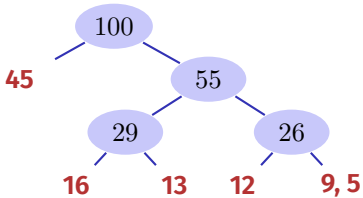
1. Sortiere die Schlüssel nach Frequenzen, oBdA: $p_1 \leq p_2 \leq ... \leq p_n$
2. Teile die Schlüssel in zwei Mengen annähernd gleichen Gewichts auf, also in Mengen $A = \{1, \ldots, k\}$ und $B = \{k+1, \ldots, n\}$ so dass $\sum_{i \in A} p_i \approx \sum_{i \in B} p_i$. Rekursion bis alle Mengen nur noch ein Element enthalten.

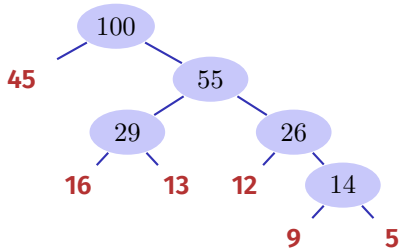
Laufzeit: $\Theta(n \log n)$

Shannon Fano Coding

45, 16, 13, 12, 9, 5



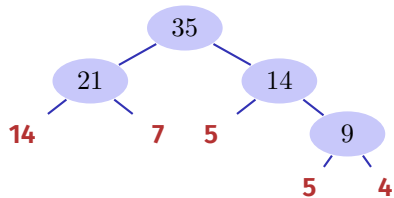




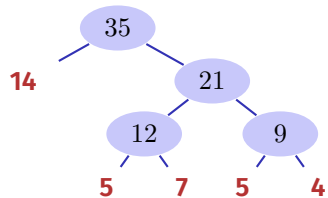
Problem

Das approximative Verfahren von Shannon und Fano liefert nicht immer ein optimales Ergebnis.

Beispiel $\{14, 7, 5, 5, 4\}$ mit unterer Grenze (Entropie) $B(T) \geq 75.35$



Shannon-Fano Coding, $B(T) = 79$

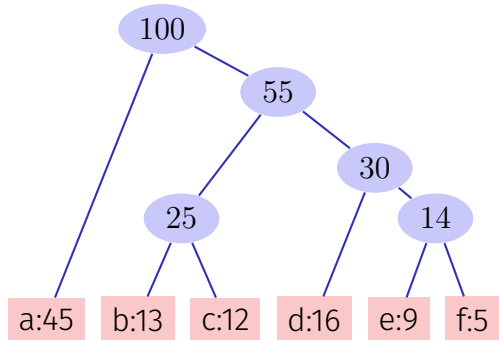


Optimal, $B(T) = 77$

Huffmans 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

$n \leftarrow |C|$
$Q \leftarrow C$
for $i = 1$ **to** $n - 1$ **do**
    Alloziere neuen Knoten z
    z.left $\leftarrow$ ExtractMin(Q) // Extrahiere Wort mit minimaler Häufigkeit.
    z.right $\leftarrow$ ExtractMin(Q)
    z.freq $\leftarrow$ 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 20

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.

Rekursive Problemlösestrategien

Brute Force Enumeration	**Backtracking**	**Divide and Conquer**	**Dynamic Programming**	**Greedy**
Rekursive Aufzählbarkeit	Prüfbare Randbedingung, Partielle Validierung	Optimale Substruktur	Optimale Substruktur, Überlappende Teilprobleme	Optimale Substruktur, Gierige Auswahl Eigenschaft
DFS, BFS, Alle Permutationen, Baumtraversieren	n Damen, Sudoku, m-Färbung, SAT-Solving, naiver TSP	Binäre Suche, Mergesort, Quicksort, Türme von Hanoi, FFT	Bellman Ford, Warshall, Rod-Cutting, LAT, Editierdistanz, Knapsack Problem DP	Dijkstra, Kruskal, Huffmann Coding