Prof. Dr. T. Härder
Fachbereich Informatik
Arbeitsgruppe Datenbanken und Informationssysteme
Universität Kaiserslautern

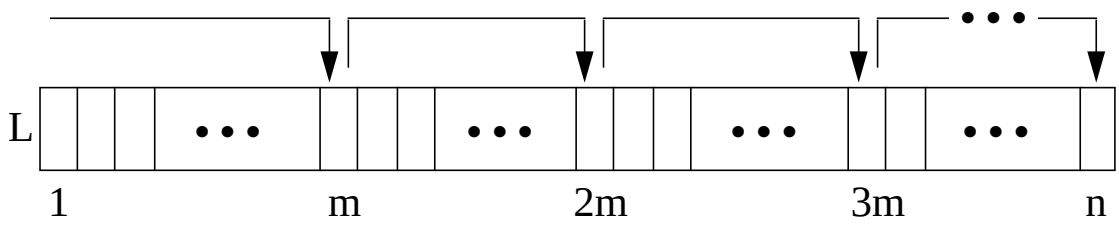
Übungsblatt 5 – Lösungsvorschläge

für die freiwillige Übung

Unterlagen zur Vorlesung: „wwwdvs.informatik.uni-kl.de/courses/DBSREAL/“

Aufgabe 1: Suche in einer Liste - Sprungsuche

Gegeben ist eine Liste L mit n Einträgen fester Länge. Die Sprungsuche soll auf Liste L optimiert werden, wobei folgendes Schema angenommen wird:



a) Bei der einfachen Sprungsuche erfolgen konstante Sprünge zu den Positionen m, 2m, 3m usw. Sobald der Abschnitt, in den der gesuchte Schlüssel fällt, lokalisiert ist, wird sequentiell gesucht. Welche mittleren Suchkosten $C_{avg}(n)$ ergeben sich, wenn ein Sprung a und ein sequentieller Vergleich b Einheiten kostet?

$$C_{avg}(n) = \frac{1}{2}a \cdot \frac{n}{m} + \frac{1}{2}b(m-1)$$

b) Was sind die Kosten $C_{avg}(n)$ bei optimaler Sprungweite m?

$$C_{avg}(n) = a\sqrt{n} - a/2$$

mit optimaler Sprungweite $m = \sqrt{(a/b)n}$ bzw. $m = \sqrt{n}$ falls a=b

c) Bei der Zwei-Ebenen-Sprungsuche wird statt sequentieller Suche im lokalisierten Abschnitt wiederum eine Quadratwurzel-Sprungsuche angewendet, bevor dann sequentiell gesucht wird. Welches $C_{avg}(n)$ ergibt sich mit

a = Kosten eines Sprungs auf der ersten Ebene,
b = Kosten eines Sprungs auf der zweiten Ebene,
c = Kosten für einen sequentiellen Vergleich?

$$C_{avg}(n) \leq \frac{1}{2} \cdot a \cdot \sqrt{n} + \frac{1}{2} \cdot b \cdot n^{\frac{1}{4}} + \frac{1}{2} \cdot c \cdot n^{\frac{1}{4}}$$

Für a = b = c ergibt sich: $C_{avg}(n) \leq a\left(\frac{1}{2}\sqrt{n} + n^{\frac{1}{4}}\right)$

d) Welche Verbesserungen ergeben sich durch optimale Abstimmung der Sprungweiten m_1 und m_2 der beiden Ebenen, wenn a = b = c gesetzt wird?

$$m_1 = n^{\frac{2}{3}} \quad \text{und} \quad m_2 = n^{\frac{1}{3}}$$

$$C_{avg}(n) = \frac{3}{2} \cdot a \cdot n^{\frac{1}{3}}$$

e) Läßt sich die Effizienz der Suche steigern, wenn das Verfahren zu einem n-Ebenen-Verfahren verallgemeinert wird?

Verallgemeinerung zu n-Ebenen-Verfahren ergibt ähnlich günstige Kosten wie Binärsuche (Übereinstimmung bei $\log_2 n$ Ebenen)

Aufgabe 2: Erweiterbares Hashing

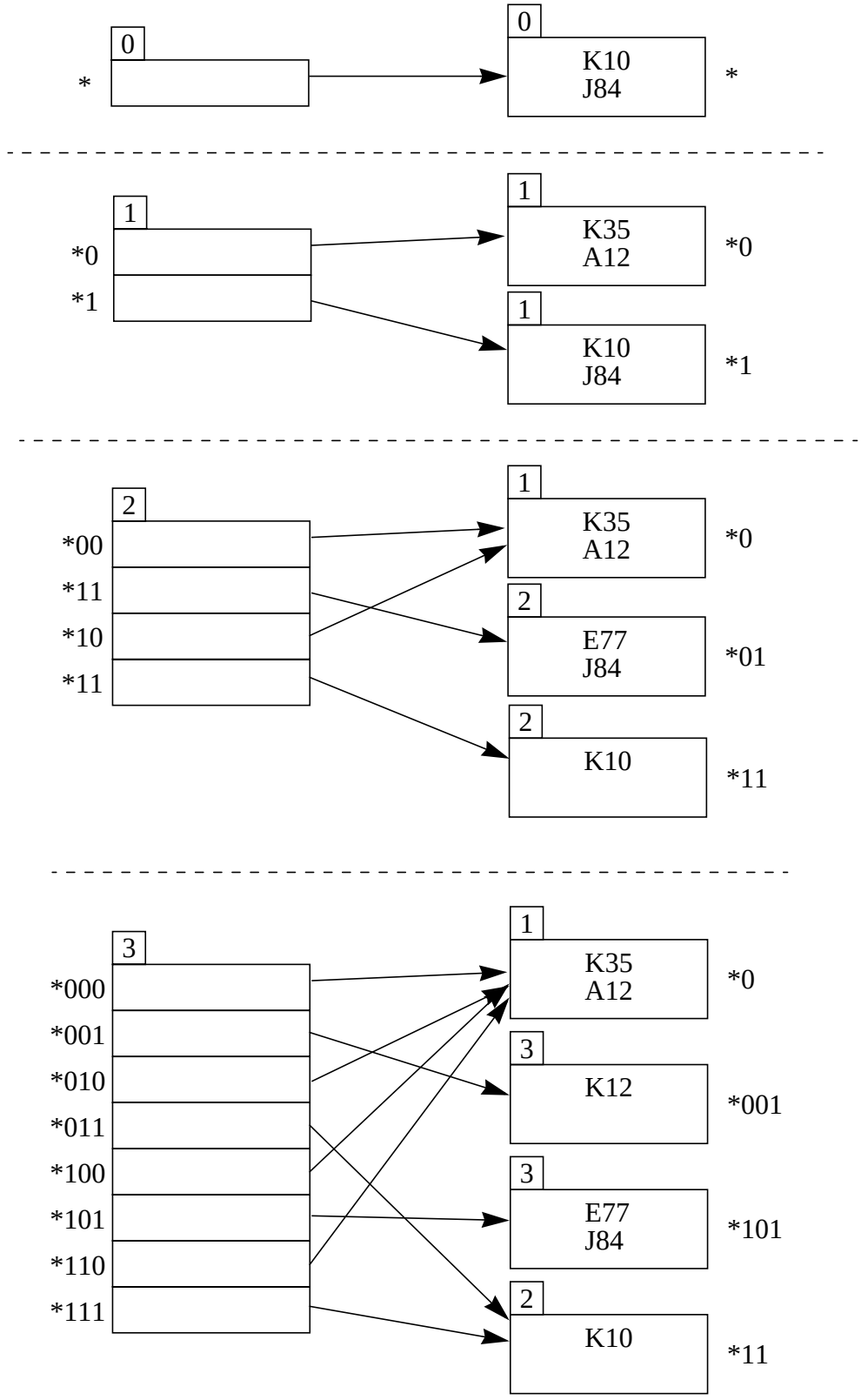
Die Sätze mit den Schlüsseln K10, J84, K35, A12, E77, K12, B22, K08 sollen in der angegebenen Reihenfolge mit dem Verfahren des Erweiterbaren Hashing in Buckets abgespeichert werden, die jeweils 2 Sätze aufnehmen können. Den Pseudoschlüssel des jeweiligen Satzes erhält man durch Verknüpfung der in EBCDIC codierten Zeichen mittels XOR. Zeichnen Sie die Belegung der Buckets und das Directory nach jeder Einfügung.

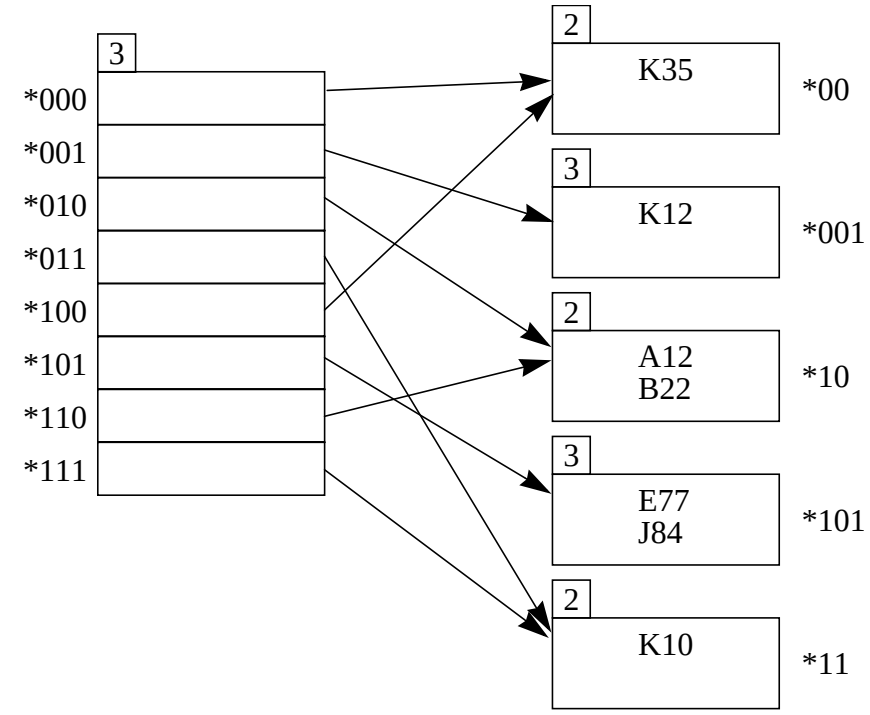
Schlüsselfolge: K10, J84, K35, A12, E77, K12, B22, K08

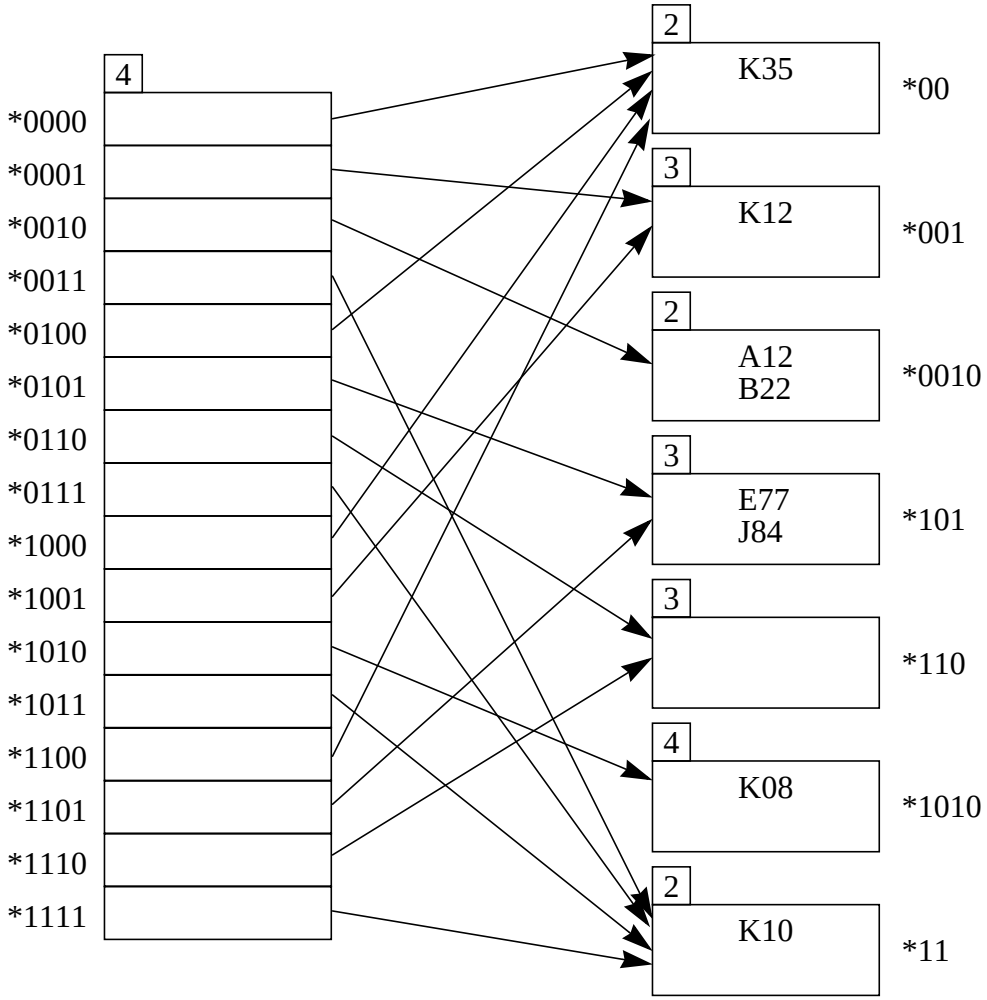
EBCDIC-Codierung: $J \equiv D1_{16}$ $B \equiv C2_{16}$ $K \equiv D2_{16}$ $E \equiv C5_{16}$ $A \equiv C1_{16}$

K10	J84	K35	A12
F0	F4	F5	F2
F1	F8	F3	F1
01	0C	06	03
D2	D1	D2	C1
D3	DD	D4	C2
1101 0011	1101 1101	1101 0100	1100 0010

E77	K12	B22	K08
F7	F2	F2	F8
F7	F1	F2	F0
00	03	00	08
C5	D2	C2	D2
C5	D1	C2	DA
1100 0101	1101 0001	1100 0010	1101 1010


0
*
0
K10
J84
*
1
*0
*1
1
K35
A12
*0
1
K10
J84
*1
2
*00
*11
*10
*11
1
K35
A12
*0
2
E77
J84
*01
2
K10
*11
3
*000
*001
*010
*011
*100
*101
*110
*111
1
K35
A12
*0
3
K12
*001
3
E77
J84
*101
2
K10
*11


3
*000
*001
*010
*011
*100
*101
*110
*111
2
K35
*00
3
K12
*001
2
A12
B22
*10
3
E77
J84
*101
2
K10
*11


4
*0000
*0001
*0010
*0011
*0100
*0101
*0110
*0111
*1000
*1001
*1010
*1011
*1100
*1101
*1110
*1111
2
K35
*00
3
K12
*001
2
A12
B22
*0010
3
E77
J84
*101
3
*110
4
K08
*1010
2
K10
*11

Aufgabe 3: Verweislisten und Bitlisten

Gegeben seien folgende Datensätze von Bestellungsinformationen, die die Adressen Zn mit $n \in \{1, 2, .., 10\}$ besitzen:

Adressen	Bestellnummer	Bestelldatum	Kundennummer
Z1	711	01.02.1999	100
Z2	600	03.01.1999	102
Z3	801	01.02.1999	141
Z4	505	02.01.1999	100
Z5	475	02.01.1999	102
Z6	730	01.02.1999	102
Z7	621	05.01.1999	141
Z8	699	09.01.1999	102
Z9	670	09.01.1999	100
Z10	515	02.01.1999	180

Invertieren Sie die oben vorgegebenen Datensätze jeweils nach den Attributen Bestelldatum und Kundennummer mit folgenden Verfahren:

a) Verweislisten

b) Bitlisten

a) Verweislisten

Invertierung nach Bestelldatum:

02.01.1999	Z4	Z5	Z10
03.01.1999	Z2		
05.01.1999	Z7		
09.01.1999	Z8	Z9	
01.02.1999	Z1	Z3	Z6

Invertierung nach Kundennummer:

100	Z1	Z4	Z9	
102	Z2	Z5	Z6	Z8
141	Z3	Z7		
180	Z10			

b) Bitlisten

Invertierung nach Bestelldatum:

02.01.1999	0 0 0 1 1 0 0 0 0 1
03.01.1999	0 1 0 0 0 0 0 0 0 0
05.01.1999	0 0 0 0 0 0 1 0 0 0
09.01.1999	0 0 0 0 0 0 0 1 1 0
01.02.1999	1 0 1 0 0 1 0 0 0 0

Invertierung nach Kundennummer:

100	1 0 0 1 0 0 0 0 1 0
101	0 1 0 0 1 1 0 1 0 0
141	0 0 1 0 0 0 1 0 0 0
180	0 0 0 0 0 0 0 0 0 1

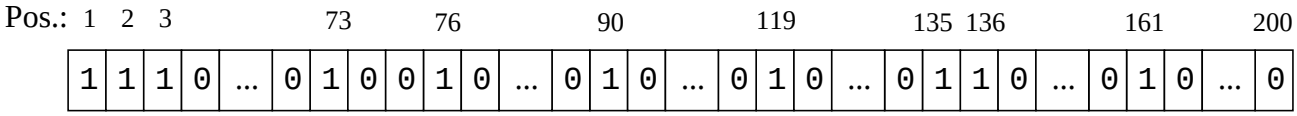
Aufgabe 4: Bitlistenkomprimierung

Gegeben sei eine Bitliste der Länge 200 mit folgenden Eigenschaften:

An den Positionen 1, 2, 3, 73, 76, 90, 119, 135, 136, 161 tritt der Wert ‘1’ und sonst ‘0’ auf.

Führen Sie für die oben angegebene Bitliste eine Komprimierung nach jedem der folgenden Verfahren:

a) *Laufkomprimierung* mit k = 6
b) *Nullfolgenkomprimierung* mit Codiereinheiten fester Länge mit k = 6
c) *Nullfolgenkomprimierung* unter der Anwendung der *Golomb-Codierung* mit m = 4

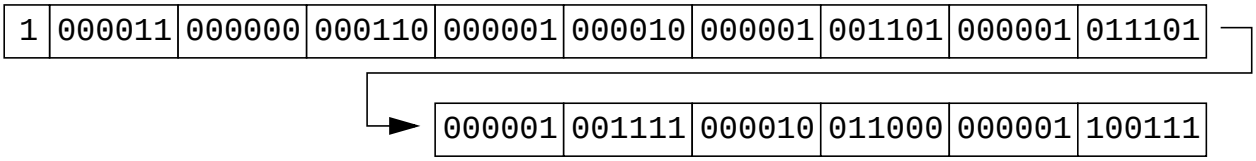


a) Laufkomprimierung mit k = 6:

Man hat folgende Eins- und Null-Folgen:

3 Einsen, 69 Nullen, 1, 2 Nullen, 1, 13 Nullen, 1, 28 Nullen, 1, 15 Nullen, 2 Einsen, 24 Nullen, 1 und

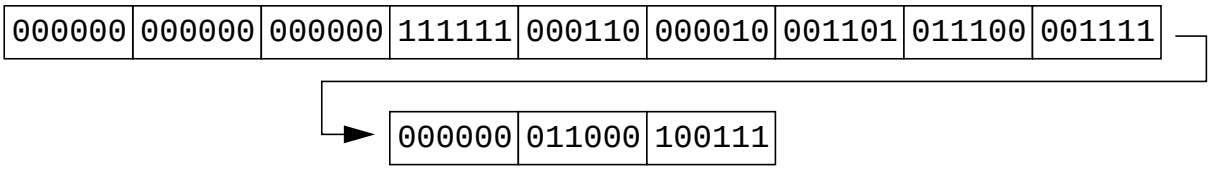
39 Nullen



b) Nullfolgenkomprimierung mit Codiereinheiten fester Länge mit k =6:

Man hat folgende Nullfolgen:

Drei Nullfolgen der Länge 0, 69 Nullen, 2 Nullen, 13 Nullen, 28 Nullen, 15 Nullen, eine Null-foge der Länge 0, 24 Nullen und 39 Nullen



c) Nullfolgenkomprimierung unter der Anwendung der *Golomb-Codierung* mit m = 4: