6 SPEICHERTECHNIKEN ZUR DARSTELLUNG VON DATENSTRUKTUREN

Im Kapitel 4 über zusammengesetzte Datenstrukturen haben wir Möglichkeiten der logischen Beschreibung von Datenstrukturen kennengelernt. Im folgenden wollen wir uns mit den Möglichkeiten der Darstellung von Datenstrukturen in Speichern befassen.

Definition: Datenstruktur

Eine Datenstruktur S ist ein Paar (D, R), worin D eine nichtleere endliche Menge von Datenobjekten d und R eine Menge von Relationen r über D ist.

Eine Relation $r \in R$ ist eine Teilmenge des Cartesischen Produktes $\bigtimes_{i=1}^{n} D = D^n$, $n \in \mathbb{N}$, also eine Menge von (geordneten) n-Tupeln von Datenobjekten.

Jede n-stellige Relation r kann mit Hilfe von binären Relationen dargestellt werden (aber nicht umgekehrt). Hinsichtlich einer binären Relation r' läßt sich die Datenstruktur S durch einen Graphen G = (N, E) veranschaulichen; die Menge N der Knoten repräsentiert die Menge D der Datenobjekte und die Menge E der gerichteten Kanten die Menge r' der geordneten Paare von Datenobjekten.

Um eine Datenstruktur in einem Speicher darzustellen, müssen einerseits die Datenobjekte gespeichert und anderseits ihre Relationen in irgendeiner Form festgehalten werden.

Ein Speicher besteht aus einer nichtleeren endlichen Menge Z von Speicherzellen z. Wir wollen den Begriff "Speicherzelle" abweichend von DIN 44300 so verstehen, daß eine Speicherzelle eine Einheit des Speichers darstellt, die einen Wert aufnehmen und auf die physisch zugegriffen werden kann. Nun ist technologisch bedingt die Menge von Speicherzellen eines Speichers mit einer Struktur versehen. Grundsätzlich läßt sich die Struktur eines beliebigen technischen Speichers auffassen als sequentielle Anordnung seiner Speicherzellen. In diesem Sinne definieren wir einen Speicher wie folgt:

Definition: Speicher

Ein Speicher ist ein Paar (Z, r_S), bei der die Menge Z aus m Speicherzellen $z_1, z_2, \ldots, z_m$ besteht und die Relation $r_S = \{(z_{i-1}, z_i);$ $i = 2, 3, \ldots, m\}$ die sequentielle Anordnung der Speicherzellen festlegt.

Anmerkung

Diese Definition erlaubt es, einen Speicher auch als Folge $(z_1, z_2, \ldots, z_m)$ seiner Speicherzellen zu betrachten.

Da also die zur Verfügung stehenden Speicher von höchst einfacher Struktur sind, gelingt die Darstellung von komplexen Datenstrukturen unmittelbar im Speicher nur bruchstückhaft. Häufig muß die vorliegende Datenstruktur um einfache, im Speicher darstellbare Strukturen ergänzt werden, und die Informationen darüber, wie aus diesen vereinfachten Strukturen die eigentlich darzustellende Datenstruktur zu gewinnen ist, in Form von zusätzlichen Daten und Algorithmen festgehalten werden.

Alle Festlegungen, die es gestatten, den logischen Zugriff auf ein Datenobjekt zu beschreiben, nennt man anschaulich den logischen Zugriffspfad. Ein einfaches Beispiel hierfür ist der Index einer Variablen vom Feldtyp, um auf ein Feldelement zugreifen zu können. Alle Daten und Algorithmen, die unter Einbeziehung der physischen Speicherorganisation den Zugriff auf ein Datenobjekt im Speicher ermöglichen, bezeichnet man zusammenfassend als den physischen Zugriffspfad.

6.1 Formale Beschreibung der Speicherung von Datenobjekten

Für den Inhalt w einer Speicherzelle z gilt zu einem gegebenen Zeitpunkt

$$w = \omega(z), \tag{6.1}$$

worin ω eine eindeutige Abbildung (Funktion) ist. Die Funktion ω ist nicht injektiv. Zwischen der Menge W der Werte und der Menge Z der Speicherzellen besteht also zu einem Zeitpunkt eine "eins-zu-viele Abbildung", d. h. ein

Wert W kann in mehreren Speicherzellen vorhanden sein, aber jede Speicherzelle besitzt genau einen Wert als Inhalt.

Definition: Gespeichertes Datenobjekt

Ein Datenobjekt d heißt gespeichert, wenn es zum Inhalt w einer Speicherzelle z geworden ist, d. h. wenn gilt:

$$w = \omega(z) = d. \tag{6.2}$$

Hierbei haben wir zunächst vorausgesetzt, daß das Datenobjekt in *einer* Speicherzelle gespeichert werden kann. Ist dies nicht der Fall, so muß das Datenobjekt in entsprechend vielen Speicherzellen gespeichert werden. Die zur Speicherung eines Datenobjektes benötigte Anzahl s von Speicherzellen nennt man die Spanne s; man sagt, das Datenobjekt wird mit der Spanne s gespeichert. So werden in einigen Rechenanlagen Gleitkommazahlen mit der Spanne 2 gespeichert.

Die Menge D der Datenobjekte einer Datenstruktur bezeichnet man auch als Datenbestand. Wir nennen einen Datenbestand D gespeichert, wenn gemäß Gl.(6.2) für jedes $d \in D$ gilt: $d = \omega(z)$.

Der Zugriff auf eine Speicherzelle kann ortsorientiert oder inhaltsorientiert erfolgen. Für den ortsorientierten Zugriff wird jeder Speicherzelle eine Adresse zugeordnet, die die Speicherzelle identifiziert. Beim inhaltsorientierten Zugriff wird die Speicherzelle aufgrund ihres Inhalts oder eines Teils davon identifiziert.

Ortsorientierter Zugriff

Für den ortsorientierten Zugriff wird jeder Speicherzelle z durch die injektive Funktion α eine Adresse

$$a = \alpha(z) \tag{6.3}$$

zugeordnet. Zwischen der Menge Z der Speicherzellen und der Menge A der Adressen besteht also eine "eins-zu-eins Abbildung".

Ohne Einschränkung der Allgemeinheit setzen wir voraus, daß die Speicherzellen gemäß ihrer sequentiellen Anordnung im Speicher fortlaufend durchnumeriert werden, so daß gilt

$$a_i = \alpha(z_i) = i-1, \qquad i = 1, 2, \ldots, m.$$

Es ist allerdings anzumerken, daß die physischen Adressen bei einigen Speichern nicht lückenlos aufeinander folgen. So weisen die Spuradressen bei Magnetplattenspeichern beim Übergang von einem zum nächsten Zylinder technisch bedingt Sprünge auf.

Der Inhalt einer Speicherzelle ergibt sich mit Gl.(6.1) und (6.3) zu

$$w = \omega(\alpha^{-1}(a)). \tag{6.4}$$

Diese Gleichung läßt sich vereinfachen, wenn man die Abbildungen α^{-1} und ω zu einer eindeutigen Abbildung C (C steht für "content") zusammenfaßt; der Inhalt der Speicherzelle mit der Adresse a ist dann gegeben durch

$$w = C(a). \tag{6.5}$$

Zusammen mit Gl.(6.2) zeigen die Gln.(6.4) bzw. (6.5), daß auf ein gespeichertes Datenobjekt d über die Adresse a der Speicherzelle zugegriffen werden kann.

Nun greift man in Programmen, die in einer höheren Programmiersprache geschrieben sind, auf Datenobjekte nicht über Adressen von Speicherzellen zu, sondern über Namen von Variablen oder Konstanten. Der Name einer Programmgröße und deren Typ werden im Quellprogramm deklariert.

Die umkehrbar eindeutige Zuordnung des Namens zu der Adresse einer Speicherzelle geschieht im allgemeinen in zwei Stufen. Der Übersetzer weist dem Namen zunächst eine programmrelative Adresse zu, die dann von der Speicherverwaltung des Betriebssystems in eine absolute physische Adresse abgebildet wird (Kapitel 10).

Ist x der Name einer Variablen oder Konstanten, so wird ihm durch die injektive Funktion L (L steht für "location") eine Adresse

$$a = L(x) \tag{6.6}$$

zugeordnet. Zwischen der Menge X der Namen und der Menge A der Adressen besteht also eine "eins-zu-eins Abbildung".

Nun sollen aber durch ein Programm Datenobjekte verarbeitet werden. Für diese hat der Name x einer Programmgröße die Funktion eines Platzhalters. Mit dem Namen x korrespondiert (über eine Adresse) eine Speicherzelle z, die innerhalb der Datenverarbeitungsanlage als Platzhalter für Datenobjekte fungiert. Durch Initialisierung oder Wertzuweisung wird dieser Zelle statisch oder dynamisch das jeweilige Datenobjekt (Wert) zugewiesen.

Für den ortsorientierten Zugriff auf ein Datenobjekt d im Speicher bei gegebenem Namen x erhält man mit Gl.(6.2) und den Gln.(6.5) und (6.6)

$$d = C(L(x)). \tag{6.7}$$

Eine weitere Möglichkeit, auf ein gespeichertes Datenobjekt zuzugreifen, besteht darin, das Datenobjekt aufgrund (eines Teils) seines Inhaltes aufzufinden. Dieser Teil wird Schlüssel (genauer: Schlüsselwert*) k des Datenobjektes d genannt. Man unterscheidet Primär- und Sekundärschlüssel. Primärschlüssel dienen zur Identifikation von Datenobjekten in einem Datenbestand. Ein Beispiel hierfür ist die Verwendung einer Personalnummer zur Identifizierung der Daten eines Mitarbeiters. Bei Verwendung eines Sekundärschlüssels können sich ein Datenobjekt oder mehrere Datenobjekte qualifizieren. Hat man zum Beispiel die Daten von Mitarbeitern einer Firma gespeichert und sucht nach allen Mitarbeitern, die an einem bestimmten Projekt beteiligt sind, so stellt "Projekt" einen Sekundärschlüssel dar (Abschn. 9.3).

Bei der Speicherung eines Datenobjektes d wird diesem mit Hilfe des Schlüssels k durch eine eindeutige Abbildung ε die Adresse

$$a = \varepsilon(k) \tag{6.8}$$

---

*) Soweit sich die Deutung des jeweiligen Begriffes aus dem Zusammenhang ergibt, wird auf die Unterscheidung verzichtet.

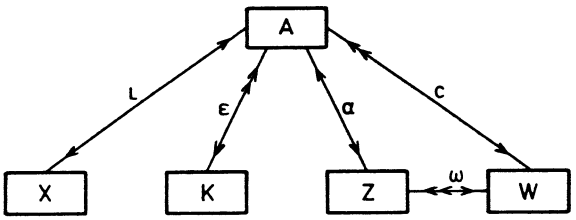
einer Speicherzelle zugeordnet. Die Funktion ε kann injektiv sein. Ist sie injektiv, so läßt sich k als Primärschlüssel verwenden, und die Umkehrabbildung ist eindeutig. Ist die Funktion ε nicht injektiv, so läßt sich k nur als Sekundärschlüssel verwenden. Zwischen der Menge K der Schlüssel und der Menge A der Adressen liegt im ersten Fall eine "eins-zu-eins Abbildung" und im zweiten Fall eine "eins-zu-viele Abbildung" vor.

Für den ortsorientierten Zugriff auf ein Datenobjekt d im Speicher bei gegebenem Schlüssel k erhält man mit Gl.(6.2) und den Gln.(6.5) und (6.6):

$$d = C(\varepsilon(k)). \tag{6.9}$$

In Speichern mit ortsorientiertem Zugriff kann der Schlüssel nur mittelbar zum Zugriff auf ein Datenobjekt herangezogen werden. Ihm muß nach Gl.(6.8) eine Adresse zugeordnet werden. Dies ist eine der wesentlichen Aufgaben der Dateiverwaltung eines Betriebssystems (Kapitel 9) oder eines Datenbankverwaltungssystems. Werden bei der Abbildung des Schlüssels in eine Adresse Methoden verwendet, bei denen die Adresse durch Manipulation eines Primärschlüssels ermittelt wird, so spricht man von gestreuter Speicherung (Abschn. 6.4).

Die im Zusammenhang mit der Speicherung von Datenobjekten bei ortsorientiertem Zugriff besprochenen Relationen sind in Bild 6-1 dargestellt; darin



X : Menge der Namen x
K : Menge der Schlüssel k
A : Menge der Adressen a
Z : Menge der Speicherzellen z
W : Menge der Inhalte w von Speicherzellen
$\leftarrow\!\!\rightarrow$: eins-zu-eins Abbildung
$\leftarrow\!\!\rightarrow\!\!\!\!\rightarrow$: eins-zu-viele Abbildung (schließt eins-zu-eins Abbildung ein)

Bild 6-1 Zur Speicherung von Datenobjekten bei ortsorientiertem Zugriff

repräsentieren die Kästchen die jeweiligen Mengen und die gepfeilten Kanten die Relationen. Bei Datenbankanwendungen sind derartige Darstellungen unter dem Namen Bachmann-Diagramm [BAC 69] bekannt.

Inhaltsorientierter Zugriff

Der inhaltsorientierte Zugriff auf eine Speicherzelle ist bei Assoziativspeichern möglich. Dazu wird jeder Speicherzelle z durch eine eindeutige Abbildung α^* ein sie kennzeichnender Inhalt

$$w^* = \alpha^*(z) \tag{6.10}$$

zugeordnet, wobei w^* i. allg. nicht den gesamten Inhalt der Speicherzelle darstellt. Ist α^* injektiv, so ist auch die Umkehrabbildung eindeutig, und bei Vorgabe eines Wertes w^* qualifiziert sich genau eine Zelle z.

Analog der Vorgehensweise beim ortsorientierten Zugriff könnte man z. B. dem Namen x einer Programmgröße im Quellprogramm durch eine injektive Funktion L^* ein die Speicherzelle z eindeutig kennzeichnenden Inhalt w^* zuordnen. Im einfachsten Fall könnte $w^* = x$ gewählt werden, so daß sich die Speicherzelle unmittelbar über den Namen der Programmgröße qualifiziert.

Im Gegensatz zur entsprechenden injektiven Funktion α (Gl. 6.3) bei ortsorientiertem Zugriff, die einer Speicherzelle eine Adresse zuordnet, muß die Funktion α^* (Gl. 6.10) nicht injektiv sein. Damit bieten Assoziativspeicher eine interessante Zugriffsmöglichkeit: Ist nämlich α^* nicht injektiv, so können sich bei Vorgabe eines Wertes w^* mehrere Speicherzellen qualifizieren, auf die dann nacheinander zugegriffen werden kann.

Das Auffinden eines gespeicherten Datenobjektes mit Hilfe seines Primärschlüssels und insbesondere das Auffinden mehrerer Datenobjekte mit Hilfe eines Sekundärschlüssels ist dadurch in Assoziativspeichern auf einfachste Weise möglich. Bei der Speicherung eines Datenobjektes wird seinem Schlüssel k durch eine Abbildung ε^* ein die Speicherzelle kennzeichnender Inhalt

$$w^* = \varepsilon^*(k) \tag{6.11}$$

zugeordnet. Im einfachsten Fall kann $w^* = k$ gewählt werden. Zusammen mit

Gl.(6.10) ergibt sich dann, daß bei inhaltsorientiertem Zugriff unmittelbar über den Schlüssel auf die Speicherzelle und damit auf das Datenobjekt zugegriffen werden kann. Im Gegensatz dazu findet beim ortsorientierten Zugriff zunächst eine Abbildung des Schlüssels in eine Adresse statt, obwohl der Schlüssel Bestandteil des gespeicherten Datenobjektes ist - er also auch die Speicherzelle kennzeichnet, in der das Datenobjekt zu finden ist.

Insbesondere bei Datenbankanwendungen wird auf Datenobjekte nach den unterschiedlichsten Kriterien (Schlüsseln) zugegriffen. Der Aufwand für die entsprechenden Abbildungen ist bei ortsorientiertem Zugriff daher sehr groß; in diesen Anwendungsfällen würde der Einsatz von Assoziativspeichern von großem Nutzen sein.

Da Assoziativspeicher zur Zeit nur sehr wenig Anwendung finden, wollen wir für die weiteren Ausführungen - falls nicht ausdrücklich anders erwähnt - Speicher mit ortsorientiertem Zugriff voraussetzen.

6.2 Sequentielle Speicherung

6.2.1 Speichertechnik

Bei der Darstellung einer Datenstruktur S=(D,R) in einem Speicher genügt es nicht, allein den Datenbestand D zu speichern, es müssen zusätzlich auch die Relationen $r \in R$ in irgendeiner Form dargestellt werden. In diesem Zusammenhang stellt sich als Teilproblem die Aufgabe, eine der Relationen $r \in R$ darzustellen. Wie zu Anfang dieses Kapitels bereits ausgeführt wurde, dürfen wir bei der Relation r ohne Einschränkung der Allgemeinheit von einer binären Relation ausgehen. Die Problemstellung, die Datenstruktur S=(D,R) in einem Speicher darzustellen, reduziert sich damit auf das Teilproblem der Repräsentation eines Graphen G = (D,r). Dazu sind neben den Knoten d (Datenobjekten) auch die Kanten $e \in r$ zu speichern.

Nun läßt sich ein Speicher als Folge von jeweils adjazenten Speicherzellen z_{i-1}, z_i auffassen. Wenn man jetzt die Kanten eines Graphen als geordnete Paare (d_j, d_k) entsprechender adjazenter Knoten d_j, d_k beschreibt, so lassen sich Kanten in einem Speicher äußerst einfach darstellen, indem adjazente Knoten in adjazenten Speicherzellen gespeichert werden. Die hier

entwickelten Vorstellungen der Speicherung von Kanten wollen wir in folgender Weise präzisieren:

Definition: Sequentiell gespeicherte Kante

Eine Kante $e = (d_j, d_k) \in r$ heißt sequentiell gespeichert genau dann, wenn die adjazenten Knoten d_j, d_k in den adjazenten Speicherzellen z_{i-1}, z_i gespeichert sind, so daß gilt: $d_j = \omega(z_{i-1})$ und $d_k = \omega(z_i)$.

Anmerkung

Diese Definition impliziert, daß die Knoten d_j und d_k mit der Spanne $s = 1$ speicherbar sind. Bei einer Spanne $s > 1$ müßte die Definition entsprechend modifiziert werden.

Definition: Sequentiell gespeicherte Relation

Eine binäre Relation $r \in R$ heißt sequentiell gespeichert genau dann, wenn alle Kanten $e \in r$ sequentiell gespeichert sind.

Damit hätten wir eine Möglichkeit gefunden, den Graphen $G = (D,r)$ in einem Speicher zu repräsentieren:

1. Jeder Knoten d des Datenbestandes D wird gespreichert - man beachte, daß der Graph G auch isolierte Knoten besitzen kann -, und
2. die Relation r wird sequentiell gespeichert.

Bei dieser Form der Speichertechnik von Graphen ist aber häufig eine Mehrfachspeicherung (redundante Speicherung) einzelner Knoten gegeben; insgesamt kann ein Knoten d maximal (1+g(d))-mal gespeichert sein, einmal bei der Speicherung des Datenbestandes D und entsprechend seinem Grad g(d)-mal bei der sequentiellen Speicherung der Relation r. Somit hat diese Speichertechnik eine wenig effiziente Speicherplatzausnutzung zur Folge.

Eine Möglichkeit, die redundante Speicherung der Knoten eines Graphen zu vermeiden, ist dann gegeben, wenn der Graph ein linearer binärer Wurzelbaum $T_1=(D,r_1)$ ist, so daß $D = \{d_1, d_2, \ldots, d_n\}$ und $r_1 = \{(d_{i-1}, d_i); i = 2, 3, \ldots, n\}$ ist. In diesem Falle läßt sich der Graph T_1 auch als Folge $(d_1, d_2, \ldots, d_n)$ interpretieren. Wenn jetzt die Elemente d_i der

Folge (d_1, d_2, ..., d_n) in die korrespondierenden Speicherzellen z_{k+i} der Folge (z_{k+1}, z_{k+2}, ..., z_{k+n}) abgelegt werden, so ist jeder Knoten des Datenbestandes D nur einmal gespeichert und zugleich die Relation r_1 sequentiell gespeichert. Wir wollen diese Form der Speicherung von $T_1 = (D,r_1)$ als sequentielle Speicherung des Datenbestandes D (bez. r_1) bezeichnen.

Ist nun in einer gegebenen Datenstruktur S = (D,R) ein linearer Binärbaum $T_1=(D,r_1)$ enthalten, so kann man sich das gegebenenfalls zunutze machen und den Datenbestand D bez. r_1 sequentiell speichern. In den Fällen, in denen a priori kein linearer Binärbaum als Teilstruktur vorliegt, kann es für die Speicherung des Dabenbestandes D zweckmäßig sein, eine Relation r_1 auf dem Datenstand D zusätzlich einzuführen, so daß er damit sequentiell (bez. r_1) gespeichert werden kann. So kann z. B. durch Einrichten eines Primärschlüssels für einen Datenbestand D eine solche Relation r_1 auf D induziert werden. Dazu wählt man eine total geordnete Menge aus, macht sie zur Menge K der Schlüsselwerte und weist jedem Datenobjekt d_i einen Schlüsselwert $k_i \in K$ zu, der das Datenobjekt identifiziert. Damit Läßt sich für den Datenbestand D eine Folge (d_1, d_2, ..., d_n) konstruieren, die bezüglich der Ordnung r auf K so sortiert ist, daß für alle i=2, 3, ..., n (d_{i-1}, d_i) $\in r_1$ genau dann gilt, wenn (k_{i-1}, k_i) $\in r$ ist. In diesem Fall sagt man, daß der Datenbestand (bez. der Ordnung r) sortiert ist.

Mit der Zunahme an struktureller Gliederung eines Datenbestandes und der damit verbundenen effizienten sequentiellen Speicherung geht ein Teil der Flexibilität hinsichtlich seiner Verarbeitung verloren.

6.2.2 Grundoperationen auf sequentiell gespeicherten Datenbeständen

Bei der Behandlung der Grundoperationen: Auffinden, Einfügen und Entfernen von Datenobjekten bei sequentieller Speicherung des Datenbestandes ist es zweckmäßig, zwischen Speichern mit sequentiellem und solchen mit direktem Zugriff zu unterscheiden.

Grundoperationen bei sequentiellem Zugriff

Um bei gegebenem Suchargument die Speicherzelle eines Datenobjektes mit diesem Wert aufzufinden, müssen die Speicherzellen starr fortlaufend durch-

sucht werden. Man nennt dieses Verfahren sequentielle oder sukzessive Suche. Die Wahrscheinlichkeit, daß ein Datenobjekt d_i, $i = 1, 2, \ldots, n$, mit dem Schlüsselwert k_i gesucht wird, sei P_i. Die mittlere Anzahl S der benötigten Zugriffe bei erfolgreicher Suche - das gesuchte Datenobjekt befindet sich im Datenbestand - beträgt

$$S = \sum_{i=1}^{n} P_i \cdot i, \qquad \text{wobei} \sum_{i=1}^{n} P_i = 1. \tag{6.12}$$

Werden alle Datenobjekte mit gleicher Wahrscheinlichkeit $P_i = 1/n$ gesucht, so ergibt sich

$$\overline{S} = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n+1}{2}. \tag{6.13}$$

Bei erfolgloser Suche - zu einem gegebenen Suchargument k gibt es kein Datenobjekt $d_i \in D$ mit $k_i = k$ - beträgt die Anzahl der Zugriffe

$$S' = n. \tag{6.14}$$

Man beachte, daß bei einem (bez. r_1) sequentiell gespeicherten Datenbestand D zwischen der mittleren Anzahl $\overline{S}$ von Zugriffen und der mittleren Pfadlänge $p(T_1)$ (Gl. 5.2) des entsprechenden linearen Binärbaumes $T_1=(D,r_1)$ folgender Zusammenhang besteht:

$$\overline{S} = \overline{p}(T_1) + 1. \tag{6.15}$$

Dies ist dadurch begründet, daß genau die eine Kante, die dem Zugriff auf das erste Datenobjekt d_1 (Wurzel des Binärbaumes) entsprechen würde, in dem Binärbaum nicht vorkommt. Entsprechend gilt bei erfolgloser Suche für S' und die Höhe $h(T_1)$ (Gl. 5.1) des linearen Binärbaumes

$$S' = h(T_1) + 1. \tag{6.16}$$

Das Einfügen eines Datenobjektes bei einem sequentiell gespeicherten Datenbestand in die Speicherzelle, die ihm bez. der Relation r_1 zukommt, ist i. allg. mit Hilfe eines zeitaufwendigen Kopiervorganges möglich (Bild 6-2). Ist das Datenobjekt am Anfang oder Ende des sequentiell gespeicherten

Datenbestandes einzufügen, so kann der Kopiervorgang u. U. entfallen, nämlich wenn die entsprechenden Speicherzellen nicht belegt sind.

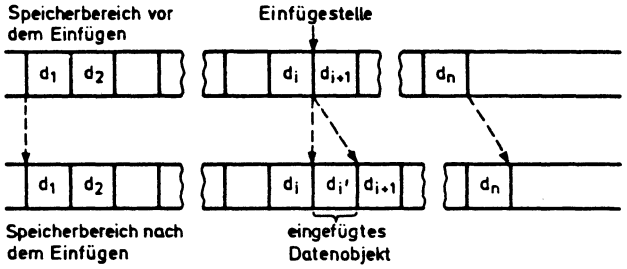


Bild 6-2 Einfügen eines Datenobjektes bei Speichern mit sequentiellem Zugriff

Das Entfernen von Datenobjekten ist nur dann in einfacher Weise möglich, wenn die sequentielle Speicherung der Relation r_1 aufgegeben wird, indem die im Speicher frei gewordene Speicherzelle (Lücke) nicht wieder entsprechend belegt wird. Das hat jedoch bei häufigem Entfernen eine ungünstige Speicherausnutzung zur Folge. Die Lücken können wieder durch Kopieren des Datenbestandes eliminiert werden (Packen, Komprimieren).

Grundoperationen bei direktem Zugriff

Generell lassen sich die Grundoperationen bei Speichern mit direktem Zugriff in der gleichen Weise wie bei Speichern mit sequentiellen Zugriff durchführen. Die dort notwendigen Kopiervorgänge beim Einfügen und Entfernen von Datenobjekten können hier jedoch durch Verschieben eines Teiles des Datenbestandes im Speicher ersetzt werden. Das Verschieben erfolgt durch sukzessives Umspeichern des Inhaltes der adressierbaren Speicherzellen (Bild 6-3).

Zum Auffinden eines Datenobjektes bei Speichern mit direktem Zugriff bieten sich auch effizientere Verfahren an, nämlich dann, wenn es sich um einen sortierten Datenbestand handelt und alle Datenobjekte mit gleicher Spanne s gespeichert sind.

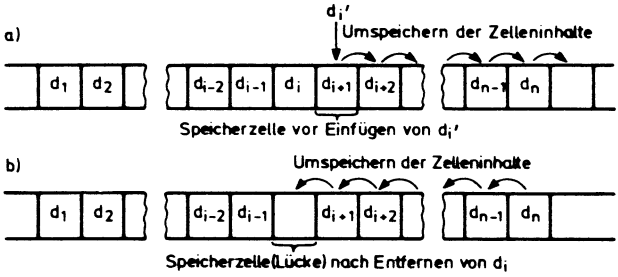
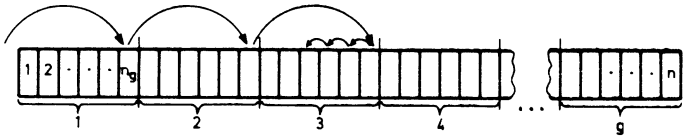


Bild 6-3 Umspeichern a) beim Einfügen eines Datenobjektes
b) beim Entfernen eines Datenobjektes

Sprungsuche

Bei dem Verfahren der Sprungsuche - auch schrittweise Suche oder m-Wege-Suche genannt - werden die n Datenobjekte in g Gruppen zu je n_g Datenobjekten eingeteilt. Die Vorgehensweise ist in Bild 6-4 dargestellt. Die Suche wird so durchgeführt, daß zunächst in jedem Schritt auf das letzte Datenobjekt der nächsten Gruppe zugegriffen wird, bis der Vergleich zwischen Suchargument und entsprechendem Schlüsselwert ergibt, daß das gesuchte Datenobjekt in der zuletzt aufgesuchten Gruppe liegen muß. Sodann werden die Datenobjekte dieser Gruppe rückwärts schreitend sequentiell durchsucht.



$n = g \cdot n_g$, g Gruppen zu je n_g Datenobjekten

Bild 6-4 Sprungsuche in einem sortierten Datenbestand bei direktem Zugriff

Wird jedes Datenobjekt d_i mit der Wahrscheinlichkeit $P_i = \frac{1}{n}$ gesucht und ist die Suche immer erfolgreich, so läßt sich die mittlere Anzahl $\bar{S}$ von Zugriffen einfach berechnen; sie setzt sich additiv zusammen aus der mittleren Anzahl (g+1)/2 von Zugriffen beim Auffinden der entsprechenden Gruppe und der mittleren Anzahl $(n_g+1)/2$ von Zugriffen beim Auffinden des gesuchten Datenobjektes in der zuletzt übersprungenen Gruppe:

$$\bar{S} = \frac{g+1}{2} + \frac{n_g+1}{2} = \frac{n}{2n_g} + \frac{n_g}{2} + 1 \quad . \tag{6.17}$$

Das Minimum von $\bar{S}$ bei gegebener Anzahl n von Datenobjekten erhält man, wenn man den Datenbestand in $g = \sqrt{n}$ Gruppen zu $n_g = \sqrt{n}$ Datenobjekten einteilt, es ist dann

$$\bar{S}_{min} = \sqrt{n}+1. \tag{6.18}$$

Anmerkungen

1. Die mittlere Anzahl $\bar{S}$ von Zugiffen läßt sich gegenüber $\bar{S}_{min} = \sqrt{n}+1$ noch weiter - wenn auch nicht erheblich - minimieren, wenn man auf eine Gruppeneinteilung des Datenbestandes in gleich große Gruppen verzichtet [NOL 72].

2. Eine Variante der m-Wege-Suche wird zur Suche in indexsequentiell gespeicherten Datenbeständen (Kapitel 9) auf Externspeichern mit quasidirektem Zugriff verwendet.

Binäre Suche

Ein noch effizienteres Verfahren als die Sprungsuche ist die binäre Suche. Bild 6-5 illustriert dieses Verfahren für den Fall, daß der sortierte Datenbestand aus $n = 2^l-1$ Datenobjekten besteht.

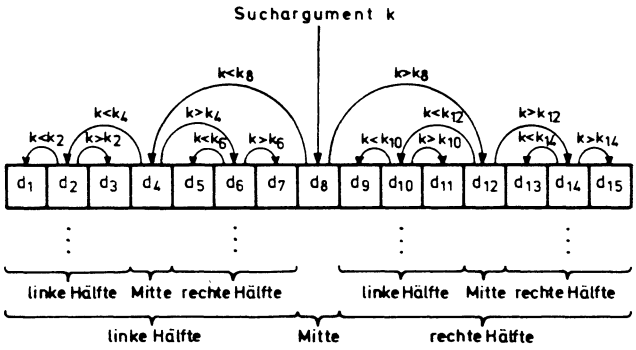


Bild 6-5 Binäre Suche in einem sortierten Datenbestand bei direktem Zugriff

Die binäre Suche läßt sich am einfachsten beschreiben in Form eines rekursiven

Algorithmus: Binäre Suche [1)]

Der Bereich, auf den sich die binäre Suche bei gegebenem Suchargument erstreckt, ist der sortierte Datenbestand.

(1) Greife auf das Datenobjekt in der "Mitte" des Bereiches zu.

(2) Wenn der Vergleich des Suchargumentes mit dem Schlüsselwert des aufgefundenen Datenobjektes ergibt:

(2.1) "=", dann ist das gesuchte Datenobjekt aufgefunden worden. Breche die binäre Suche ab.

(2.2) "<", dann reduziert sich die weitere Suche auf die "linke Hälfte des Bereiches". Ist die "linke Hälfte des Bereiches" leer, gehe nach (3), andernfalls ersetze "Bereich" durch "linke Hälfte des Bereiches" und gehe nach (1).

(2.3) ">", dann reduziert sich die weitere Suche auf die "rechte Hälfte des Bereiches". Ist die "rechte Hälfte des Bereiches" leer, gehe nach (3), andernfalls ersetze "Bereich" durch "rechte Hälfte des Bereiches" und gehe nach (1).

(3) Das gesuchte Datenobjekt befindet sich nicht im Datenbestand. Beende die binäre Suche.

Für den Fall $n + 1 = 2^l$ veranschaulicht Bild 6-5 exemplarisch, daß mit Ausnahme des ersten Zugriffs auf den Datenbestand die Gesamtheit der übrigen Zugriffe auf die "Mitten" bei binärer Suche einem vollständigen Binärbaum entspricht. Für $n + 1 \neq 2^l$ sind aber die "linke Hälfte" und die "rechte Hälfte" eines Bereiches nicht immer gleich groß, so daß in diesem Falle der Graph der Zugriffe auf die jeweiligen "Mitten" ein voller Binärbaum ist. Diese Betrachtungsweise erlaubt es, die mittlere Anzahl $\overline{S}$ der Zugriffe bei binärer Suche über die mittlere Pfadlänge $\overline{p}_n(T_V)$ (Gl.5.4)

eines vollen Binärbaumes mit n Knoten zu bestimmen. Wenn jedes Datenobjekt d_i mit gleicher Wahrscheinlichkeit $P_i = 1/n$ gesucht wird und die Suche immer erfolgreich verläuft, beträgt

$$\overline{S} = \overline{p}_n(T_V) + 1 \approx \mathrm{ld}n - 1 \ . \qquad (6.19)$$

Bei erfolgloser binärer Suche läßt sich die Anzahl S' der Zugriffe aus der Länge des Pfades von der Wurzel bis zu einem Blatt im entsprechenden vollen Binärbaum T_V ermitteln; diese Pfadlänge ist entweder gleich der Höhe $h(T_V)$ oder gleich $h(T_V) - 1$. Mit Gl.(5.3) ergibt sich

$$\lfloor \mathrm{ld}n \rfloor \leq S' \leq \lfloor \mathrm{ld}n \rfloor + 1. \qquad (6.20)$$

Anmerkung

Da die binäre Suche den Direktzugriff auf jedes Datenobjekt erfordert, ist dieses Suchverfahren bei sequentiell gespeicherten Datenbeständen auf Speichern mit quasidirektem Zugriff nicht realisierbar.

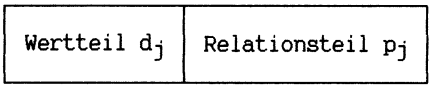
Wie wir gesehen haben, kann die Suche in sequentiell gespeicherten Datenbeständen sehr effizient durchgeführt werden, wenn die Datenbestände sortiert sind. Hingegen erfordert das Einfügen und Entfernen von Datenobjekten bei sequentiell gespeicherten Datenbeständen - sei es durch Kopieren, sei es durch Umspeichern - in der Regel einen enormen Aufwand. Abhilfe schafft erst die Methode der Kettung von Datenobjekten, auf die im folgenden Abschnitt eingegangen wird.

6.3 Gekettete Speicherung

6.3.1 Speichertechnik

Wie zu Beginn von Abschnitt 6.2.1 dargelegt, tritt bei der Darstellung einer Datenstruktur S = (D,R) in einem Speicher das Teilproblem der Repräsentation eines Graphen G = (D,r), $r \in R$, auf. Dazu sind neben den Knoten $d \in D$ auch die Kanten $e \in r$ zu speichern. Bei der Darstellung einer Kante im Speicher ist man nun nicht an die sequentielle Speicherstruktur gebunden. Wenn man jedem Knoten d_j einen Relationsteil p_j zuordnet, so läßt sich

die Kante (d_j, d_k) zweier adjazenter Knoten d_j und d_k dadurch darstellen, daß man im Relationsteil des Anfangsknotens d_j die Beziehung zum Endknoten d_k beschreibt. Da der Relationsteil selbst ein Datenobjekt ist, entsteht so ein neues, zusammengesetztes Datenobjekt (d_j, p_j). Es besteht aus dem "eigentlichen" Datenobjekt d_j, das wir fortan als Wertteil des zusammengesetzten Datenobjektes bezeichnen, und dem Relationsteil p_j.



Wertteil und Relationsteil können wiederum zusammengesetzte Datenobjekte sein. Die Datenobjekte des Wertteils werden auch als Primärdaten bezeichnet, die Datenobjekte des Relationsteils werden zu den Sekundärdaten gezählt.

Der Relationsteil p_j enthält einen Verweis auf ein anderes Datenobjekt, genauer gesagt, auf die Speicherzelle z_r, in der dieses andere Datenobjekt gespeichert ist. Dieser Verweis wird Zeiger (pointer) oder auch Referenz (reference) genannt (Abschn. 4.4). Über den Relationsteil wird also eine Abbildung $z_r = \varsigma(p_j)$ realisiert. In Speichern mit ortsorientiertem Zugriff enthält der Relationsteil häufig die Adresse einer Speicherzelle ($z_r = \alpha^{-1}(p_j)$). Die hier entwickelte Vorstellung der Speicherung von Kanten können wir nun wie folgt präzisieren:

Definition: Gekettet gespeicherte Kante

> Eine Kante $e = (d_j, d_k) \in r$ heißt gekettet gespeichert genau dann, wenn die adjazenten Knoten d_j, d_k in den Speicherzellen z_s, z_t gespeichert sind, so daß mit $(d_j, p_j) = \omega(z_s)$ und $(d_k, p_k) = \omega(z_t)$ gilt: $z_t = \varsigma(p_j)$.

Somit wird die Darstellung einer Kante im Speicher unabhängig davon, in welchen Speicherzellen die adjazenten Knoten gespeichert sind.

Definition: Gekettet gespeicherte Relation

> Eine binäre Relation $r \in R$ heißt gekettet gespeichert genau dann, wenn alle Kanten $e \in r$ gekettet gespeichert sind.

Damit hätten wir eine weitere Möglichkeit gefunden, einen Graphen G = (D,r) in einem Speicher zu repräsentieren:

1. Jeder Knoten d des Datenbestandes D wird gespeichert - man beachte, daß der Graph G auch isolierte Knoten besitzen kann -, und
2. die Relation r wird gekettet gespeichert.

Bild 6-6 zeigt die gekettete Speicherung des Graphen G = (D,r_1) mit $D = \{d_1,d_2, \ldots,d_6\}$ und $r_1 = \{(d_1,d_2),(d_2,d_3),(d_3,d_4),(d_4,d_5),(d_5,d_6)\}$. Wie zu erkennen ist, sind die Datenobjekte entsprechend der Relation r_1 über die Adreßverweise gekettet. Diese Form der Speicherung bezeichnen wir als linear gekettete Speicherung des Datenbestandes D (bez. r_1).

Weitere Beispiele gekettet gespeicherter Graphen sind in Bild 6-7 dargestellt, wobei der Übersichtlichkeit wegen auf die Anordnung der Datenobjekte im Speicher verzichtet worden ist.

Zur Unterstützung der Verarbeitung von gekettet gespeicherten Datenstrukturen wird vielfach noch ein Anker eingerichtet; das ist ein Zeiger, der auf ein ausgezeichnetes Datenobjekt (Kopfelement) der Datenstruktur verweist und damit von "außen" einen Zugang zur Datenstruktur ermöglicht.

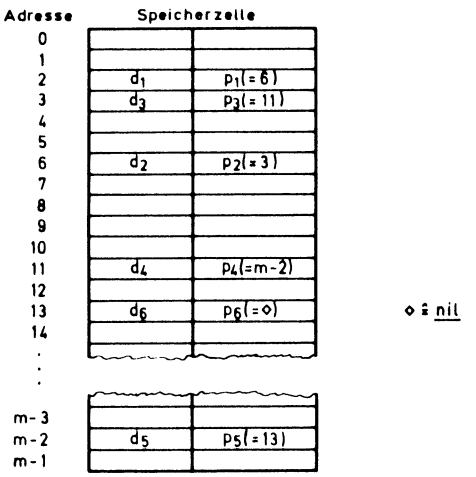


Bild 6-6 Linear gekettete Speicherung eines Datenbestandes

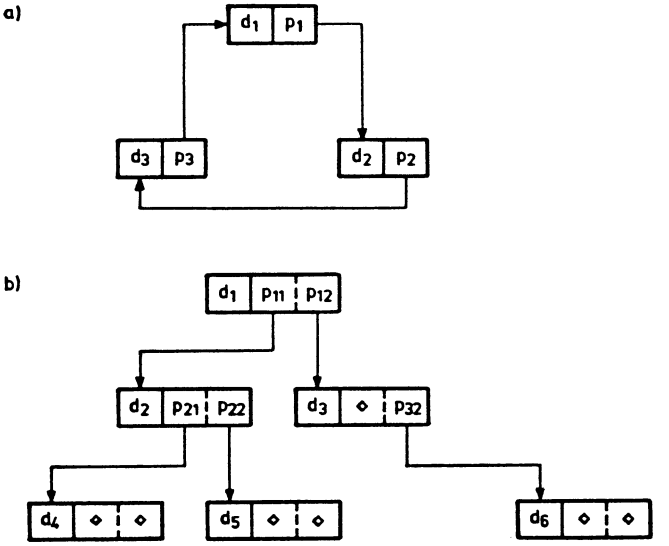


Bild 6-7 Gekettete Speicherung von Graphen
a) Zyklus, b) Binärer Wurzelbaum

6.3.2 Grundoperationen auf linear gekettet gespeicherten Datenbeständen

Ein Datenobjekt innerhalb eines linear gekettet gespeicherten Datenbestandes kann nur in der Weise aufgefunden werden, daß - beginnend mit dem Anker - die Kette sukzessiv durchsucht wird (lineare Suche). Unter der Voraussetzung, daß jedes Datenobjekt d_i mit der gleichen Wahrscheinlichkeit $P_i = \frac{1}{n}$ gesucht wird und die Suche erfolgreich verläuft, beträgt die mittlere Anzahl $\overline{S}$ der Zugriffe

$$\overline{S} = \frac{n+1}{2} , \qquad (6.21)$$

und zwar unabhängig davon, ob der Datenbestand bez. des Suchargumentes sortiert oder unsortiert ist. Bei großen Datenbeständen ist die lineare Suche ineffizient. Techniken, die durch den zusätzlichen Aufbau geeigneter Zugriffspfade die Suche in großen Datenbeständen effizienter gestalten, werden in Kapitel 8 und 9 behandelt.

Die Kettung von Datenobjekten bietet den großen Vorteil, daß beim Einfügen und Entfernen die Datenobjekte nicht umgespeichert werden müssen, sondern daß lediglich die Zeiger zu aktualisieren sind. In Bild 6-8a) ist das

Einfügen eines Datenobjektes d_2 zwischen die Datenobjekte d_1 und d_3 dargestellt. Bild 6-8b) zeigt das Entfernen eines Datenobjektes d_2 aus der Folge (d_1, d_2, d_3).

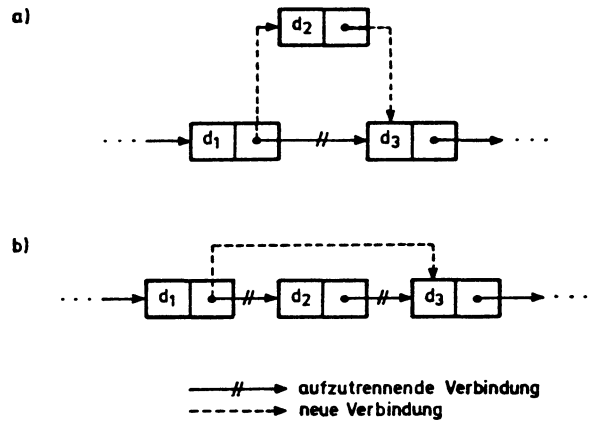


Bild 6-8 Änderungen in linear gekettet gespeicherten Datenbeständen
a) Einfügen eines Datenobjektes
b) Entfernen eines Datenobjektes

Die beim Entfernen von Datenobjekten im Speicher entstehenden Lücken können der Freispeicherverwaltung zugeführt und bei Einfügungen aufs Neue verwendet werden. Die freien Speicherzellen können zum Beispiel dadurch verwaltet werden, daß aus ihnen ebenfalls eine gekettete Folge gebildet wird, deren Anfang freie Speicherzellen entnommen bzw. frei gewordene Speicherzellen hinzugefügt werden (Kapitel 7).

6.3.3 Formen der Kettung

Die Speichertechnik mit Hilfe von Zeigern bezeichnet man als Kettung (chaining). Neben der bisher besprochenen einfach geketteten Speicherung von Datenobjekten gibt es einige erweiterte Formen. Bei der Ringkettung (zyklische Kettung) weist der Zeiger im letzten Datenobjekt der Kette auf den Anfang der Kette. Der Vorteil besteht darin, daß, von einem beliebigen Datenobjekt in der Kette ausgehend, der Zugriff auf jedes andere Datenobjekt möglich ist. Bei der Doppelkettung weist ein Zeiger auf den Nachfolger, ein weiterer Zeiger auf den Vorgänger des Datenobjektes. Die Kette kann also in beiden Richtungen durchsucht werden. Der Algorithmus für das

Einfügen und Entfernen von Datenobjekten gestaltet sich dadurch etwas einfacher, weil beim Aufsuchen der entsprechenden Stelle in der Kette die sonst notwendige Zwischenspeicherung der Vorgänger-Adresse entfällt. Bei der Ankerkettung enthält jedes Datenobjekt einen Verweis auf den Anker der Kette. Ähnlich wie bei der Ringkettung und Doppelkettung kann, ausgehend von einem beliebigen Datenobjekt, jedes andere erreicht werden. Der Zugriffspfad zum Anker ist kürzer.

Werden Ring- und Doppelkettung kombiniert, kann bei Verlust eines Zeigers dennoch jedes Datenobjekt der Kette erreicht werden und die Kette rekonstruiert werden (recovery).

6.4 Gestreute Speicherung

Bei der gestreuten Speicherung (scatter storage) wird nach Gl.(6.8) aus dem Schlüssel k eines Datenobjektes mit Hilfe der Funktion ε die Speicheradresse a bestimmt. Diese Speichertechnik ist daher nur bei Speichern mit Direktzugriff anwendbar.

6.4.1 Methoden

Man unterscheidet:

- Direkte Adressierung *)
  Der Schlüssel wird umkehrbar eindeutig auf eine Speicheradresse abgebildet.

- Indirekte Adressierung *)
  Der Schlüssel wird nicht umkehrbar eindeutig auf eine Speicheradresse abgebildet. Es treten Mehrfachbelegungen (Kollisionen) auf.

Man nennt die Verfahren der indirekten Adressierung auch Hash-Verfahren (hashing; hash coding). Die Sprachregelung ist allerdings nicht einheitlich. Der Bedeutung des Wortes "hash" (zerhacken) entsprechend werden

---

*) Nicht zu verwechseln mit den Adressierungsarten bei Befehlen (direkt: Der Operandenteil des Befehls enthält den Operanden; indirekt: Der Operandenteil enthält die Adresse des Operanden).

häufig im engeren Sinne solche Verfahren als Hash-Verfahren bezeichnet, bei denen durch Manipulation von Teilen des Schlüssels die Adresse ermittelt wird. Im weiteren Sinn werden dagegen die Verfahren der gestreuten Speicherung insgesamt als Hash-Verfahren angesprochen.

Die gestreute Speicherung dient nicht zur Darstellung einer Datenstruktur im Speicher, sondern zur bloßen Speicherung von Datenobjekten in einem bestimmten Adreßbereich. Sie ermöglicht aber über den Schlüssel den quasi-inhaltsorientierten Zugriff auf ein Datenobjekt.

Nennen wir die Menge der vorgesehenen möglichen Schlüssel K'. Häufig wird nur eine Teilmenge $K \subset K'$ aktuell zur Identifizierung von Datenobjekten benötigt (z. B. Menge der aktuellen Kundennummern als Teilmenge aller möglichen Kundennummern). Das Problem besteht nun darin, die Menge K der Schlüsselwerte in die Menge A (Adreßbereich) der Speicheradressen abzubilden.

Die Funktion

$$a = \sigma(k),\ k \in K \text{ und } a \in A \tag{6.22}$$

wird Schlüsseltransformation (auch "Speicherfunktion" oder "Streufunktion") genannt. Die Adresse a bezeichnet man als Hausadresse des Schlüsselwertes k.

Bei der direkten Adressierung ist σ injektiv. Sie wird vorteilhaft dann angewendet, wenn sich die Menge der Schlüsselwerte auf einen Adreßbereich so abbilden läßt, daß die transformierten Schlüsselwerte (Adressen) nahezu lückenlos aufeinanderfolgen. Mit ihr werden unter allen Speicherungsformen die kürzesten Zeiten für das Auffinden der Datenobjekte bei gegebenem Primärschlüssel bei wahlfreier Verarbeitung und für die Datenpflege (Ändern, Entfernen) erreicht. Ebenso ist eine schnelle logisch fortlaufende Verarbeitung der Datenobjekte möglich, wenn durch geeignete Wahl der Schlüsseltransformation die Datenobjekte in der Sortierfolge ihrer Schlüssel gespeichert sind. Häufig wird eine lineare Schlüsseltransformation verwendet:

$$a = s \cdot k + d \tag{6.23}$$

Beispiel:

Die Menge der Schlüssel sei K = {500,501,...,1000}. Die Datenobjekte sollen mit der Spanne s = 2 ab der Speicheradresse 6000 gespeichert werden. Damit ergibt sich d = 5000; es werden die Speicherzellen mit den Adressen 6000 bis 7001 belegt.

$\alpha = 2 \cdot k + 5000$

Bei der indirekten Adressierung ist die Schlüsseltransformation σ nicht injektiv und σ wird Hash-Funktion genannt. Werden außerdem die Adressen a als Indizes eines eindimensionalen Feldes aufgefaßt, so wird der adressierte Speicherbereich als Hash-Tabelle (oder Streutabelle) bezeichnet. Zwei unterschiedliche Schlüssel k_i und k_j können auf dieselbe Adresse abgebildet werden, so daß $\sigma(k_i) = \sigma(k_j)$ ist. Man bezeichnet diese Situation als Kollision. Die Schlüssel k_i und k_j heißen dann Synonyme. Bei indirekter Adressierung ist somit neben der Schlüsseltransformation auch ein Verfahren zur Behandlung von Kollisionen anzugeben.

Die indirekte Adressierung wendet man dann an, wenn bei direkter Adressierung die transformierten Schlüsselwerte (Adressen) nicht nahezu lückenlos aufeinanderfolgen würden. Ein solcher Fall kann z. B. bei nichtnumerischen Schlüsseln vorliegen, wenn man sie als numerische Schlüssel interpretiert (das Bitmuster der Zeichenkette des Schlüsselwertes wird als duale Festkommazahl aufgefaßt). Die Hash-Funktion ist so zu wählen, daß sich die ermittelten Adressen möglichst gleichmäßig über den Adreßbereich verteilen und möglichst keine Kollisionen vorkommen.

Eine Methode, die Anzahl der Kollisionen von vornherein zu reduzieren, besteht darin, mehr Speicherzellen zur Verfügung zu stellen als zur Speicherung der Datenobjekte benötigt werden. Führt man hier den Speicherbelegungsfaktor

$$\beta = \frac{\text{Anzahl der benötigten Speicherzellen}}{\text{Anzahl der zur Verfügung gestellten Speicherzellen}} \qquad (6.24)$$

ein, so wird man $\beta < 1$ wählen. In der Praxis hat sich ein Wert $\beta \approx 0{,}8$ bewährt.

Die logisch fortlaufende Verarbeitung der Datenobjekte ist bei indirekter Adressierung nur möglich, wenn eine Liste mit den sortierten Schlüsselwerten vorliegt.

Im folgenden werden einige Schlüsseltransformationen für die indirekte Adressierung angegeben.

Divisionsrest-Methode

Der Schlüssel k (positive ganze Zahl) wird durch eine ganze Zahl p (in der Regel eine Primzahl) dividiert, wobei der Divisionsrest zur Ermittlung der Adresse a herangezogen wird. Der Divisor p muß in etwa dem Quotienten aus der Anzahl der zur Verfügung gestellten Speicherzellen und der Spanne s entsprechen. Als Hash-Funktion verwendet man dann

$$a = \sigma(k) = s \cdot (k \bmod p) + d. \tag{6.25}$$

Beispiel:

Es seien 8000 Datenobjekte mit der Spanne s = 2 zu speichern. Bei einem Speicherbelegungsfaktor $\beta = 0{,}8$ sollten somit etwa 20000 Speicherzellen zur Verfügung gestellt werden. Der Speicherbereich sei ab Adresse 5000 verfügbar. Es gilt dann mit $p = 9973 \approx 10000$

$$a = \sigma(k) = 2 \cdot (k \bmod 9973) + 5000.$$

Basistransformation

Der Schlüssel k wird als Stellenwertcodierung zur Basis B mit Wert $\sum_{i=0}^{j-1} Z_i B^i$ interpretiert. Bei der Basistransformation wird nun die Ziffernfolge $(Z_{j-1}, \ldots, Z_1, Z_0)$ zur Berechnung der Adresse

$$a = \sigma(k) = \sum_{i=0}^{j-1} Z_i \tilde{B}^i, \qquad \tilde{B} \neq B, \tag{6.26}$$

herangezogen. Eine Basis $\tilde{B} < B$ bedeutet dabei eine Komprimierung des Wertebereiches, während $\tilde{B} > B$ eine Expandierung des Wertebereichs zur Folge hat. Dabei kann es durchaus sinnvoll sein, $\tilde{B} > B$ zu wählen, um

ungleichmäßig dicht belegte Schlüsselbereiche in weitgehend gleichmäßig belegte abzubilden. Um aus dem Ergebnis eine Adresse im vorgesehenen Adreßbereich zu gewinnen, schließt man an die Basistransformation eine weitere geeignete Schlüsseltransformation (z. B. Divisionsrest-Methode) an.

Beispiel:

Gegeben sei k = 264; 264 werde als Oktalzahl interpretiert (B = 8). Zur Komprimierung des Wertebereiches wird $\tilde{B} = 4$ gewählt. Damit ist

$a = \sigma(k) = 2 \cdot 4^2 + 6 \cdot 4^1 + 4 \cdot 4^0 = 74_8$.

Beispiel:

Gegeben sei k = 3264; 3264 werde als Dezimalzahl interpretiert (B = 10). Mit $\tilde{B} = 11$ erhält man zunächst

$k^* = 3 \cdot 11^3 + 2 \cdot 11^2 + 6 \cdot 11 + 4 = 4305_{10}$.

Der Schlüssel k^* wird nun in eine Adresse abgebildet. Soll der Adreßbereich z. B. alle dreistelligen Dezimalzahlen umfassen, so ist

$a = \sigma(k) = k^* \bmod 1000 = 305_{10}$.

Ziffernauswahl (Ziffernanalyse)

Das Prinzip besteht darin, aus den Ziffernstellen der Gesamtheit der Schlüsselwerte (ganze positive Zahlen) diejenigen auszuwählen, in denen die Ziffern möglichst gleichverteilt auftreten. Die Ziffern in diesen ausgewählten Stellen werden zur Bildung der Adresse in Stellenschreibweise herangezogen.

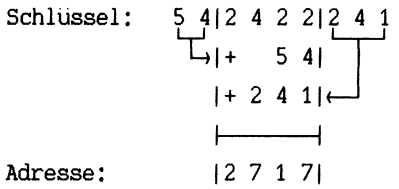
Beispiel:

Der Schlüssel bestehe aus den Ziffernstellen $s_5\ s_4\ s_3\ s_2\ s_1\ s_0$. Eine statistische Untersuchung ergibt für die Ziffernstellen s_4, s_2 und s_1 eine annähernde Gleichverteilung der möglichen Ziffern. In diesem Fall würde man die Adresse aus den Ziffern in den Stellen $s_4\ s_2\ s_1$ bilden.

Faltung

Der Schlüssel (ganze positive Zahl) wird in mehrere Teile zerlegt. Durch arithmetische Verknüpfung der Teile sowie Ausblenden bestimmter Spalten wird eine zulässige Adresse bestimmt.

Beispiel:



Was die Behandlung von Kollisionen betrifft, so kann die Speicherung von kollidierenden Datenobjekten im Adreßbereich selbst - dem Hauptbereich - erfolgen oder aber in einem eigenen Adreßbereich - dem Überlaufbereich. Dabei können die kollidierenden Datenobjekte im Haupt- oder im Überlaufbereich gekettet gespeichert werden (Kettung), oder es wird auf eine Kettung verzichtet und mit Hilfe des Schlüssels eine Folge von Speicheradressen im Hauptbereich festgelegt, die im Falle einer Kollision als Ausweichadressen in Frage kommen (offene Adressierung; open adressing oder open hash). Die festgelegte Adreßfolge - sie sei a_0, a_1, a_2, ... - muß für einen gegebenen Schlüssel natürlich stets dieselbe bleiben. Von den zahlreichen Verfahren der Kollisionsbehandlung [MAU 75], [MOR 69] werden im folgenden einige exemplarisch erläutert.

Dabei wollen wir ohne Einschränkung der Allgemeinheit voraussetzen, daß
- {0,1,2,...,m-1} der Adreßbereich des zur Verfügung stehenden Speicherbereiches ist und
- die Datenobjekte mit der Spanne s = 1 speicherbar sind.

Offene Adressierung

Ist beim Speichern eines Datenobjektes eine Speicherzelle bereits belegt, so wird versucht, das Datenobjekt in der Speicherzelle mit der nächsten Ausweichadresse unterzubringen.

Um eine vollständige Belegung des Adreßbereiches zu ermöglichen, muß jede Adresse des Adreßbereiches genau einmal in der Adreßfolge auftreten (d. h. die Folge muß eine Permutation aller Adressen des Adreßbereiches sein). Beim Auffinden eines Datenobjektes wird ebenfalls, ausgehend von der Hausadresse, die Folge der Adressen durchsucht. Die Suche ist beendet, wenn das Datenobjekt gefunden wird, oder eine freie Speicherzelle erreicht wird, oder die gesamte Adreßfolge durchlaufen worden ist.

Das einfachste Verfahren ist die lineare Sondierung (linear probing), bei der die Adreßfolge folgendermaßen berechnet wird:

$$\begin{array}{ll} & a_i = (\sigma(k) + i) \bmod m; \\ \text{oder} & \qquad\qquad i = 0,1, \ldots, m-1. \\ & a_i = (\sigma(k) - i) \bmod m; \end{array} \tag{6.27}$$

Beispiel:

Es sei ein Speicherbereich mit m = 16 Speicherzellen gegeben, in dem schon einige Datenobjekte gespeichert sind (Bild 6-9). Es soll nun ein Datenobjekt mit der Hausadresse $\sigma(k) = 3$ eingefügt werden.

Da die Speicherzelle mit der Adresse 3 belegt ist, wird mit $a_i = (\sigma(k)+i) \bmod m$ die Ausweichadresse $a_1 = 4$ bestimmt. Diese ist ebenfalls belegt, und so wird als nächste Ausweichadresse $a_2 = 5$ bestimmt und das Datenobjekt dort gespeichert.

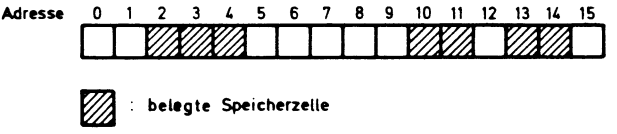


Bild 6-9 Zur linearen Sondierung

Die lineare Sondierung hat den Nachteil, daß sie zu einer Kollisionshäufung (clustering) führt. Soll zum Beispiel in den Speicherbereich gemäß Bild 6-9 ein Datenobjekt mit der Hausadresse $\sigma(k)$ eingefügt werden, so ist der Eintrag in eine der freien Speicherzellen nicht für alle Zellen gleich

wahrscheinlich. Der Eintrag in die Speicherzelle 5 erfolgt nämlich, wenn $2 \leq \sigma(k) \leq 5$ ist, der Eintrag in die Speicherzelle 6 , wenn $\sigma(k) = 6$ ist. Das bedeutet, daß in der gegebenen Situation ein Eintrag in die Speicherzelle 5 viermal wahrscheinlicher ist als ein Eintrag in die Speicherzelle 6. Der Effekt der Kollisionshäufung wird noch verstärkt, wenn sich im Speicher kleinere, bisher getrennte Bereiche mit belegten Zellen zu größeren Bereichen vereinigen (z. B. in Bild 6-9 durch Belegung der Speicherzelle mit der Adresse 12). Durch die Kollisionshäufung wird die Anzahl der notwendigen Zugriffe in der Adreßfolge erhöht (Abschn. 6.4.2).

Eine Möglichkeit, die Kollisionshäufung zu reduzieren, besteht darin, die Schrittweite zwischen den Ausweichadressen der Adreßfolge von deren Stellung i in der Folge abhängig zu machen:

$$\begin{aligned} &a_i = (\sigma(k) \pm \delta(i)) \bmod m; \\ \text{oder auch} \qquad & \qquad i = 0,1,\ldots,m-1. \\ &a_i = (\sigma(k) \pm \delta(i,a_0)) \bmod m; \end{aligned} \tag{6.28}$$

Beispiele für derartige Verfahren sind die zufällige Sondierung (random probing), die als Schrittweite $\delta(i)$ Pseudo-Zufallszahlen aus dem Intervall $[1,m-1]$ benutzt, und die quadratische Sondierung, die als Schrittweite $\delta(i) = i^2$ benutzt. Die Verfahren nach Gl.(6.28) haben hinsichtlich der Kollisionshäufung den Nachteil, daß die Schrittweite für beliebige Schlüssel bzw. für synonyme Schlüssel stets nach der gleichen Gesetzmäßigkeit berechnet werden.

Die Kollisionshäufung kann weiter verringert, praktisch sogar beseitigt werden, wenn die Schrittweite zwischen den Ausweichadressen der Adreßfolge vom Schlüsselwert k des Datenobjektes abhängig gemacht wird. Diese Methode wird Doppel-Hashing (double hashing) genannt.

Es wird eine zweite Hash-Funktion σ' benutzt, um die Schrittweite zu berechnen. Für die Adreßfolge gilt:

$$a_i = (\sigma(k) \pm i \cdot \sigma'(k)) \bmod m, \qquad i = 0,1,\ldots,m-1. \tag{6.29}$$

Um eine vollständige Belegung des Adreßbereiches zu garantieren, muß die Adreßfolge $a_0, a_1, \ldots, a_{m-1}$ eine Permutation der Adressen $0,1, \ldots, m-1$

des Adreßbereiches sein. Das ist dann der Fall, wenn $\sigma'(k)$ und m teilerfremd sind für alle $k \in K$. Die geforderte Teilerfremdheit ist dann gegeben, wenn $\sigma'(k) \in \{1,2, \ldots, m-1\}$ und m eine Primzahl ist. Wendet man zur Berechnung von $\sigma(k)$ und $\sigma'(k)$ die Divisionsrest-Methode nach Gl.(6.25) an, so liegt es nahe, Primzahl-Zwillinge m und (m-2) zu wählen und die Berechnung folgendermaßen durchzuführen:

$$\sigma(k) = k \bmod m$$
$$\sigma'(k) = 1 + (k \bmod (m-2)). \qquad (6.30)$$

Setzt man diese Gleichungen in Gl.(6.29) zur Bestimmung der Adreßfolge ein, so zeigen empirische Tests [KNU 75, Vol. 3], daß praktisch keine Kollisionshäufungen auftreten. Dies gilt unabhängig von dem speziellen Verfahren nach Gl.(6.30) immer dann, wenn $\sigma(k)$ und $\sigma'(k)$ derart unabhängig voneinander sind, daß die Wahrscheinlichkeit, daß für $k_i \neq k_j$ sowohl $\sigma(k_i) = \sigma(k_j)$ als auch $\sigma'(k_i) = \sigma'(k_j)$ ist, von der Ordnung $1/m^2$ ist.

Kettung der kollidierenden Datenobjekte

Für die Kettung kollidierender Datenobjekte im Adreßbereich - dem Hauptbereich - gibt es zwei Varianten:

Bei der Kettung ohne Überschneidung (overlap) werden jeweils die Datenobjekte mit synonymen Schlüsseln linear gekettet gespeichert. Unter der Hausadresse wird der Anker der Kette von Datenobjekten mit dieser Hausadresse abgelegt. Aus diesem Verfahren resultiert, daß beim Einfügen eines Datenobjektes eine Umspeicherung eines bereits gespeicherten Datenobjektes notwendig werden kann, nämlich dann, wenn die Hausadresse des einzufügenden Datenobjektes bereits durch ein nicht synonymes Datenobjekt belegt ist (Abschn. 6.4.3).

Bei der Kettung mit Überschneidung sind solche Umspeicherungen nicht notwendig, weil jedes kollidierende Datenobjekt in die Kette eingefügt wird, in die seine Hausadresse eingebunden ist. In derartigen Ketten können sich also nichtsynonyme Datenobjekte befinden.

Ein Nachteil der gestreuten Speicherung ist, daß aufgrund der Schlüsseltransformation die Größe des Adreßbereiches (Hash-Tabelle) von vornherein

festgelegt werden muß. Nicht immer ist jedoch die maximale Anzahl der zu speichernden Datenobjekte bekannt. Wählt man dann einen großen Adreßbereich, so führt das unter Umständen zu einer schlechten Speicherbelegung. Ist der Adreßbereich zu klein gewählt, kann ein Teil der Datenobjekte nicht gespeichert werden. Dies läßt sich durch die gekettete Speicherung kollidierender Datenobjekte außerhalb des Adreßbereiches in einem besonderen Überlaufbereich vermeiden (Kettung im Überlaufbereich). Ein zu knapp bemessener Hauptbereich führt dann lediglich zu einer steigenden Anzahl von Kollisionen; die Speicherung der Datenobjekte ist aber sichergestellt. Im Bedarfsfall kann der Überlaufbereich vergrößert werden (Kapitel 10), ohne daß hiervon die Schlüsseltransformation berührt wird. Bei dem Verfahren der Kettung im Überlaufbereich wird zweckmäßig jeweils der Anker einer Kette synonymer Datenobjekte unter der Hausadresse im Hauptbereich abgelegt.

Die Kettung erfordert wegen des notwendigen Relationsteiles im Datenobjekt zusätzlichen Speicherplatz gegenüber der offenen Adressierung. Die Vor- und Nachteile der verschiedenen Verfahren werden bei der Behandlung der Grundoperationen ersichtlich.

6.4.2 Grundoperationen auf gestreut gespeicherten Datenbeständen

Bevor wir auf die Ausführung der Grundoperationen bei den oben dargestellten Techniken der gestreuten Speicherung eingehen, wollen wir ein Modell für die offene Adressierung diskutieren, das uns eine Abschätzung der Leistungsfähigkeit der Verfahren der offenen Adressierung bezüglich der Grundoperationen Einfügen und Auffinden erlaubt. Das Modell wird gleichmäßiges Hashing genannt und geht von folgenden idealisierenden Voraussetzungen aus: Die Werte k des Primärschlüssels treten gleichwahrscheinlich auf, und bei jedem Zugriff auf eine Speicherzelle gemäß der durch Hausadressen und Ausweichadressen gegebenen Adreßfolge wird auf die noch nicht aufgesuchten Speicherzellen mit gleicher Wahrscheinlichkeit zugegriffen. Es tritt also keine Kollisionshäufung auf.

Es soll zunächst die zu erwartende Anzahl von Zugriffen ermittelt werden, die nötig sind, um bei n bereits gespeicherten Datenobjekten ein (n+1)-tes Datenobjekt einzufügen. Ein Zugriff ist mindestens notwendig. Die Wahrscheinlichkeit, daß die Speicherzelle beim ersten Zugriff belegt ist und ein zweiter Zugriff notwendig ist, beträgt $P_1 = n/m$ (m: Anzahl der Spei-

cherzellen bzw. der Adressen). Da in den m-1 noch nicht aufgesuchten Speicherzellen noch n-1 belegte Speicherzellen vorhanden sind, beträgt die Wahrscheinlichkeit, nunmehr eine belegte Speicherzelle anzutreffen und somit einen dritten Zugriff zu benötigen, $P_2 = (n-1)/(m-1)$. Die Wahrscheinlichkeit, daß die Speicherzelle beim i-ten Zugriff belegt ist und ein (i+1)-ter Zugriff benötigt wird, beträgt

$$P_i = \frac{n - i + 1}{m - i + 1}, \qquad i = 1,2,\ldots,n. \tag{6.31}$$

Der Erwartungswert für die Anzahl der Zugriffe, die nötig sind, um bei n gespeicherten Datenobjekten ein weiteres einzufügen, beträgt daher

$$\begin{aligned} E_n &= 1 + P_1(1+P_2(1+P_3(1+ \ldots \cdot(1+P_i(1+ \ldots+P_{n-1}(1+P_n) \ldots) \\ &= 1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\left(1 + \frac{n-2}{m-2}\left(1 + \ldots\right.\right.\right. \\ &\quad \cdot\left(1 + \frac{n-i+1}{m-i+1}\left(1 + \ldots+ \frac{2}{m-n+2}\left(1 + \frac{1}{m-n+1}\right) \ldots\right). \end{aligned} \tag{6.32}$$

Da die Folge der Speicherzellen durchsucht wird, bis eine freie Speicherzelle gefunden wird, stellt E_n auch die mittlere Anzahl S' der Zugriffe bei erfolgloser Suche dar:

$$S' = E_n. \tag{6.33}$$

Multipliziert man in Gl.(6.32) die Ausdrücke der Form $P_i(1 + A_{i+1})$, beginnend mit der innersten Klammer, so erhält man rekursiv

$$A_i = P_i(1 + A_{i+1}) \text{ mit } i = n-1, n-2, \ldots, 2,1 \text{ und } A_n = P_n. \tag{6.34}$$

Es ist

$$A_i = \frac{n - i + 1}{m - n + 1} \tag{6.35}$$

und somit

$$E_n = S' = 1 + A_1 = 1 + \frac{n}{m - n + 1} = \frac{m + 1}{m + 1 - n}. \tag{6.36}$$

Wir wollen nun die mittlere Anzahl S von Zugriffen ermitteln, die bei erfolgreicher Suche benötigt werden, um ein Datenobjekt aufzufinden. Dazu ersetzen wir in Gl.(6.36) die Anzahl n der gespeicherten Datenobjekte durch den Laufindex j. E_j stellt also den Erwartungswert für die Anzahl von Zugriffen bei j gespeicherten Datenobjekten und erfolgloser Suche, d. h. für das Auffinden einer freien Speicherzelle, dar. Der Erwartungswert für die erfolgreiche Suche bei j gespeicherten Datenobjekten ist E_{j-1}, weil der jeweils letzte Zugriff auf die freie Speicherzelle nicht berücksichtigt werden darf. Die mittlere Anzahl $\bar{S}$ von Zugriffen bei erfolgreicher Suche ist gleich dem Mittelwert über alle E_{j-1} für j = 1,...,n:

$$\bar{S} = \frac{1}{n} \sum_{j=1}^{n} E_{j-1} = \frac{m+1}{n} \sum_{j=1}^{n} \frac{1}{m+2-j}$$

$$= \frac{m+1}{n} \left(H(m+1) - H(m+1-n)\right), \qquad (6.37)$$

worin H(x) die harmonische Funktion ist.

Somit erhält man näherungsweise

$$\bar{S} \approx \frac{m+1}{n} \left(\ln(m+1) - \ln(m+1-n)\right) = -\frac{m+1}{n} \ln\left(1 - \frac{n}{m+1}\right) . \qquad (6.38)$$

Der Speicherbelegungsfaktor ist $\beta = n/m$, so daß wir für $m \gg 1$ näherungsweise für die Anzahl der Zugriffe bei erfolgreicher Suche angeben können:

$$\bar{S} \approx -\frac{1}{\beta} \ln(1 - \beta). \qquad (6.39)$$

Entsprechend erhalten wir aus Gl.(6.36) für die Anzahl der Zugriffe bei erfolgloser Suche

$$S' \approx \frac{1}{1 - \beta} . \qquad (6.40)$$

Interessant an diesen Ergebnissen ist, daß $\bar{S}$ und S' ausschließlich von β abhängen. In Tabelle 6-1 sind einige Werte für $\bar{S}$ und S' in Abhängigkeit von β angegeben.

Tabelle 6-1 $\overline{S}(\beta)$ und $S'(\beta)$ bei gleichmäßigem Hashing

β	0,5	0,6	0,7	0,8	0,9	0,95
$\overline{S}$	1,39	1,53	1,72	2,01	2,56	3,15
S'	2,00	2,50	3,33	5,00	10,0	20,0

Bei den folgenden Ausführungen wird stets vorausgesetzt, daß alle Schlüssel gleichwahrscheinlich auftreten und daß die Hash-Funktion die Schlüssel auf die Adressen des Adreßbereichs gleichverteilt abbildet.

Grundoperationen bei offener Adressierung

Durch die Kollisionshäufung wird die mittlere Anzahl der Zugriffe beim Auffinden und Einfügen eines Datenobjektes erhöht. Für die Anzahl $\overline{S}$ bzw. S' der Zugriffe für die erfolgreiche bzw. erfolglose Suche bei linearer Sondierung gilt [KNU 75, Vol. 3]:

$$\overline{S} \approx \frac{1}{2}\left(1 + \frac{1}{1 - \beta}\right) \tag{6.41}$$

und

$$S' \approx \frac{1}{2}\left(1 + \frac{1}{(1 - \beta)^2}\right) . \tag{6.42}$$

Tabelle 6-2 zeigt einige Werte. Man erkennt die starke Zunahme von $\overline{S}$ bzw. S' für $\beta \to 1$ infolge der Kollisionshäufung.

Tabelle 6-2 $\overline{S}(\beta)$ und $S'(\beta)$ bei linearer Sondierung

β	0,5	0,6	0,7	0,8	0,9	0,95
$\overline{S}$	1,50	1,75	2,17	3,00	5,50	10,50
S'	2,50	3,63	6,06	13,00	50,50	200,5

Das Entfernen von Datenobjekten ist bei linearer Sondierung nicht ohne weiteres möglich. Denn würde man ein Datenobjekt entfernen, das unter der Adresse a_i gespeichert ist, so könnte auf alle Datenobjekte mit Ausweichadressen a_j, $j > i$, nicht mehr zugegriffen werden. Eine einfache Möglichkeit, dieses Problem zu umgehen, besteht darin, das Datenobjekt zwar zu entfernen, die Speicherzelle aber als gelöscht zu markieren, so daß die Suche bei ihr nicht abgebrochen wird. Bei häufigem Einfügen und Entfernen von Datenobjekten nimmt jedoch durch derart markierte Speicherzellen auch bei günstiger Speicherbelegung die Anzahl der Zugriffe zu. Ein besseres, aber aufwendigeres Verfahren besteht darin, gewisse Datenobjekte in der Suchfolge sukzessiv umzuspeichern, so daß die durch Entfernen entstandene Lücke in der Suchfolge geschlossen wird. Welche Datenobjekte umzuspeichern sind, wollen wir uns anhand von Bild 6-10 klarmachen. Ein Datenobjekt in der Speicherzelle mit der Adresse a_i sei entfernt worden. Die aktuell untersuchte Speicherzelle habe die Adresse a_j, der Schlüssel des darin gespeicherten Datenobjektes d_j sei k_j. Ist $\sigma(k_j) = a_j$, so ist d_j unter seiner Hausadresse gespeichert, eine Umspeicherung ist nicht möglich. Liegt $\sigma(k_j)$ im Bereich B_1, so ist eine Umspeicherung ebenfalls nicht möglich, weil dann die Adresse a_i in der Suchfolge für das Auffinden des Datenobjektes d_j weiter hinten liegt als die Adresse a_j.

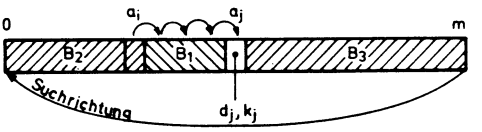


Bild 6-10 Zur Umspeicherung von Datenobjekten beim Entfernen

Liegt dagegen die Hausadresse $\sigma(k_j)$ in den Bereichen B_2 oder B_3, so liegt die Adresse a_i in der Suchfolge weiter vorne als die Adresse a_j. Das Datenobjekt d_j wird umgespeichert, es wird $i = j$ gesetzt und das Verfahren sukzessiv fortgesetzt.

Bei den Verfahren des Doppel-Hashing (zum Beispiel nach Gl.(6.30)) tritt praktisch keine Kollisionshäufung auf, wenn $\sigma(k)$ und $\sigma'(k)$ unabhängig voneinander sind. Empirische Tests zeigen, daß die mittlere Anzahl $\bar{S}$ bzw. S' der Zugriffe bei erfolgreicher bzw. erfolgloser Suche eine gute Übereinstimmung mit den theoretischen Ergebnissen des gleichmäßigen Hashing nach

den Gln.(6.39), (6.40) und Tabelle 6.1 aufweisen. Das Entfernen von Datenobjekten ist jedoch mit vertretbarem Aufwand nur in der Weise möglich, daß die entsprechende Speicherzelle als gelöscht markiert wird - mit den oben beschriebenen Nachteilen bei häufigem Einfügen und Entfernen.

Grundoperationen bei Kettung im Hauptbereich

Zuerst sollen die Grundoperationen bei der Kettung ohne Überschneidung behandelt werden. Das Einfügen eines Datenobjektes wird folgendermaßen vorgenommen: Es wird die Hausadresse $\sigma(k)$ bestimmt. Ist die entsprechende Speicherzelle frei, so wird das Datenobjekt dort gespeichert. Ist die Speicherzelle bereits mit einem synonymen Datenobjekt belegt, so folgt man der hier beginnenden Kette synonymer Datenobjekte bis an deren Ende, ermittelt nach einem geeigneten Verfahren eine freie Speicherzelle im Hauptbereich, speichert das Datenobjekt und fügt diese Speicherzelle an das Ende der Kette an. Ist die Speicherzelle unter der Hausadresse jedoch mit einem nicht synonymen Datenobjekt belegt, so muß dieses in eine freie Speicherzelle umgespeichert und dessen eventuell vorhandene Verkettung entsprechend korrigiert werden. Das einzufügende Datenobjekt wird dann unter seiner Hausadresse gespeichert. Ein geeignetes Verfahren zum Auffinden eines freien Speicherplatzes im Hauptbereich besteht zum Beispiel darin, den Hauptbereich von einem Ende her linear zu ketten und bei Bedarf dieser Kette ein Element zu entnehmen. Beim Auffinden eines gewünschten Datenobjektes beginnt die Suche bei der Hausadresse und erfolgt dann entlang der Kette.

Für die Anzahl $\overline{S}$ bzw. S' der Zugriffe bei erfolgreicher bzw. erfolgloser Suche gilt nach [KNU 75, Vol. 3]:

$$\overline{S} \approx 1 + \frac{\beta}{2}, \tag{6.43}$$

$$S' \approx e^{-\beta} + \beta. \tag{6.44}$$

Hierin ist β wieder der Speicherbelegungsfaktor. Einige Werte von $\overline{S}$ und S' zeigt Tabelle 6-3. Bemerkenswert ist, daß $\overline{S} > S'$ ist.

Die Werte lassen sich jedoch in dieser Form nicht unmittelbar mit denen der offenen Adressierung vergleichen, und auch absolut betrachtet, müssen sie sorgfältig interpretiert werden. Dies hat drei Gründe:

Tabelle 6-3 $\overline{S}(\beta)$ und $S'(\beta)$ bei Kettung im Hauptbereich ohne Überschneidung

β	0,5	0,6	0,7	0,8	0,9	0,95
$\overline{S}$	1,25	1,30	1,35	1,40	1,45	1,48
S'	1,11	1,15	1,20	1,25	1,31	1,34

1. Es entsteht bei der Kettung durch den Relationsteil im Datenobjekt ein Speichermehraufwand. Ist r der Quotient aus dem Speicherplatzbedarf des Relationsteils und dem des Wertteils, so entspricht im Vergleich dem Speicherbelegungsfaktor β bei der offenen Adressierung ein Faktor

$$\beta_k = \beta(1 + r) \qquad (6.45)$$

bei der Kettung (wohlgemerkt: die Berechnung von $\overline{S}$ bzw. S' erfolgt in jedem Fall mit β). Man beachte, daß $\beta_k > 1$ werden kann.

2. In der Anzahl S' der Zugriffe sind die Zugriffe nicht enthalten, die beim Einfügen eines Datenobjektes notwendig sind, um nach Erreichen des Endes der Kette von synonymen Datenobjekten eine freie Speicherzelle im Hauptbereich zu finden.

3. Die Anzahl S' der Zugriffe beim Einfügen eines Datenobjektes sagt nichts über die eventuell notwendige und zeitaufwendige Umspeicherung eines nichtsynonymen Datenobjektes aus. Die Anzahl der notwendigen Umspeicherungen kann verringert werden, wenn die Speicherung der Datenobjekte im Hauptbereich in zwei Stufen erfolgt: Zunächst werden nur diejenigen Datenobjekte gespeichert, für die unter ihrer Hausadresse eine noch nicht belegte Speicherzelle angetroffen wird, danach werden alle übrigen Datenobjekte gespeichert (zweistufiges Laden).

Das Entfernen von Datenobjekten ist wegen der Kettung leicht durchzuführen. Inwieweit allerdings die frei werdende Speicherzelle in den Freispeicherbe-

reich eingefügt werden kann, hängt von dem gewählten Verfahren für die Freispeicherverwaltung ab.

Bei der Kettung mit Überschneidung wird das Einfügen eines Datenobjektes wie folgt vorgenommen: Es wird die Hausadresse $\sigma(k)$ bestimmt. Ist die entsprechende Speicherzelle frei, wird das Datenobjekt dort gespeichert. Ist die Speicherzelle bereits belegt, so folgt man der Kette, in der die belegte Speicherzelle eingegliedert ist, bis an deren Ende, ermittelt eine freie Speicherzelle im Hauptbereich, speichert das Datenobjekt und fügt diese Speicherzelle an das Ende der Kette an. Dies geschieht unabhängig davon, ob die belegte Speicherzelle ein synonymes oder nicht synonymes Datenobjekt enthält. Beim Auffinden eines Datenobjektes beginnt die Suche bei der Hausadresse und erfolgt dann entlang der Kette. Für die Anzahl $\overline{S}$ bzw. S' der Suchschritte gilt nach [KNU 75, Vol. 3]:

$$\overline{S} \approx 1 + \frac{1}{8\beta}(e^{2\beta} - 1) - \frac{1}{4}(1 - \beta), \tag{6.46}$$

$$S' \approx 1 + \frac{1}{4}(e^{2\beta} - 1 - 2\beta). \tag{6.47}$$

Tabelle 6-4 zeigt einige Werte. Die Anmerkungen, die oben zur Interpretation der Ergebnisse bei der Kettung ohne Überschneidung gemacht worden sind, gelten auch hier - mit der Ausnahme, daß keine Umspeicherungen erfolgen.

Tabelle 6-4 $\overline{S}(\beta)$ und $S'(\beta)$ bei Kettung im Hauptbereich mit Überschneidung

β	0,5	0,6	0,7	0,8	0,9	0,95
$\overline{S}$	1,30	1,38	1,47	1,57	1,68	1,74
S'	1,18	1,28	1,41	1,59	1,81	1,95

Das Entfernen von Datenobjekten kann wie bei der Kettung ohne Überschneidung vorgenommen werden.

Grundoperationen bei Kettung im Überlaufbereich

Ist beim Einfügen eines Datenobjektes die Speicherzelle unter der Hausadresse $\sigma(k)$ im Hauptbereich frei, so wird das Datenobjekt dort gespeichert. Ist die Speicherzelle belegt, so folgt man der eventuell hier beginnenden Kette synonymer Datenobjekte bis an das Ende der Kette, ermittelt im Überlaufbereich eine freie Speicherzelle (freie Speicherzellen sind dort linear gekettet), speichert das Datenobjekt und fügt es dem Ende der Kette an.

Beim Auffinden eines Datenobjektes beginnt die Suche bei der Adresse im Hauptbereich und führt eventuell entlang der Kette im Überlaufbereich. Mit einem Faktor $\beta^* = \frac{n}{m}$ (n: Anzahl der gespeicherten Datenobjekte; m: Anzahl der Speicherzellen im Hauptbereich), der hier nicht den Speicherbelegungsfaktor darstellt, gilt analog den Gln.(6.43) und (6.44) für die mittlere Anzahl der Zugriffe $\overline{S}$ bzw. S' bei erfolgreicher bzw. erfolgloser Suche:

$$\overline{S} \approx 1 + \frac{\beta^*}{2}, \tag{6.48}$$

$$S' \approx e^{-\beta^*} + \beta^*. \tag{6.49}$$

Von den n gespeicherten Datenobjekten ist ein Teil im Hauptbereich und der Rest im Überlaufbereich gespeichert. Die Anzahl m der bei diesem Verfahren insgesamt benötigten Speicherzellen setzt sich aus den n Speicherzellen für die Datenobjekte und den im Hauptbereich nicht belegten Speicherzellen zusammen. Diese lassen sich wie folgt ermitteln: Die Wahrscheinlichkeit, daß für einen gegebenen Schlüssel eine bestimmte Hausadresse ermittelt wird, ist 1/m; die Wahrscheinlichkeit, daß die entsprechende Speicherzelle im Hauptbereich bei bereits n gespeicherten Datenobjekten noch frei ist, beträgt somit $(1 - 1/m)^n$. Der Erwartungswert für die Anzahl freier Speicherzellen im Hauptbereich ist daher $m(1 - 1/m)^n$. Damit ist

$$\overline{m} = n + m\,(1 - \frac{1}{m})^n. \tag{6.50}$$

Für den Speicherbelegungsfaktor gilt dann

$$\beta = \frac{n}{\overline{m}} = \frac{n/m}{n/m + (1 - 1/m)^n} \approx \frac{\beta^*}{\beta^* + e^{-\beta^*}}. \tag{6.51}$$

In Tabelle 6-5 sind einige Werte für $\overline{S}$ bzw. S' als Funktion von β angegeben. Sollen diese Werte mit den entsprechenden Werten bei den Verfahren der offenen Adressierung verglichen werden, so muß der durch die Kettung entstehende Speichermehraufwand entsprechend Gl.(6.45) berücksichtigt werden.

Tabelle 6-5 $\overline{S}(\beta)$ und $S'(\beta)$ bei Kettung im Überlaufbereich

β	0,5	0,6	0,7	0,8	0,9	0,95
$\overline{S}$	1,28	1,36	1,46	1,60	1,84	2,08
S'	1,13	1,21	1,32	1,50	1,87	2,28

Das Entfernen von Datenobjekten ist wegen der Kettung im Überlaufbereich recht einfach. Die frei gewordene Speicherzelle wird wieder in die Kette der freien Speicherzellen im Überlaufbereich eingegliedert.

6.4.3 Bewertung und Anwendung

Wie wir gesehen haben, sind die Verfahren der Schlüsseltransformation besonders gut geeignet, Datenobjekte in einem Adreßbereich wieder aufzufinden, wenn deren Primärschlüssel bekannt ist. Bei direkter Adressierung kann auf jedes Datenobjekt unmittelbar zugegriffen werden. Aber auch bei indirekter Adressierung ist die Größenordung der mittleren Anzahl von Zugriffen unabhängig von der Anzahl der gespeicherten Datenobjekte. Dies wird von keiner anderen Speichertechnik bzw. von keinem anderen Suchverfahren erreicht. Gerade deshalb ist es wichtig, sich auch über die Nachteile der Schlüsseltransformation im klaren zu sein:

- Im ungünstigsten Fall müssen alle Speicherzellen durchsucht werden. Ist eine definierte Reaktionszeit beim Auffinden von Datenobjekten gefordert, so scheiden die Verfahren der Schlüsseltransformation aus. Ausnahme: direkte Adressierung.

Die Größe des Adreßbereiches bzw. der Hash-Tabelle muß von vornherein festgelegt werden. Ausnahme: indirekte Adressierung mit Kettung im Überlaufbereich.

- Bei den meisten Verfahren ist das Entfernen von Datenobjekten aufwendig. Ausnahme: direkte Adressierung oder indirekte Adressierung mit Kettung im Überlaufbereich.

- Eine logisch fortlaufende Verarbeitung ist zumindest bei den Verfahren der indirekten Adressierung nicht ohne beträchtlichen Mehraufwand (Sortierung der Datensätze) möglich.

- Beruht die Speicherverwaltung in einem Rechensystem auf dem Konzept der virtuellen Speicherung (Abschn. 10.2), so werden bei gestreuter Speicherung der Datenobjekte in einem großen virtuellen Adreßbereich häufige Ein- und Auslagerungen von Teilen dieses Adreßbereiches in den bzw. aus dem Hauptspeicher notwendig.

Ein wichtiges Anwendungsgebiet der gestreuten Speicherung ist die direkte Suche in Tabellen (table lookup), die in vielen Programmier-Anwendungen benötigt wird. So sind z. B. bei der Übersetzung von Programmen durch Assembler oder Kompilierer Symbol-Tabellen anzulegen. Die Effizienz des Übersetzers hängt unter anderem von einer kurzen Zugriffszeit auf die Symbole in der Tabelle und die ihnen zugeordneten Werte ab (zum Beispiel Variablenname und zugeordnete Speicheradresse).

Die gestreute Speicherung kann auch dazu benutzt werden, eine gegebene Menge von Datenobjekten nach einem Ordnungskriterium (Sortierschlüssel) zu sortieren [KNU 75, Vol.3].