Mit **Ordnung** (Grad) wird die maximale Anzahl der unmittelbaren Nachfolger eines Knotens bezeichnet. Ein *Baum der Ordnung zwei* (= ein Baum vom Grad zwei) heißt **binärer Baum**. Eine *Kette* läßt sich als *Baum der Ordnung eins* darstellen.

In der Folge werden wir uns ausschließlich mit binären Bäumen, der wichtigsten Kategorie, beschäftigen. Realisiert werden binäre Bäume mit einem *Index, der aus dem Schlüssel und zwei Kettenfeldern besteht.* Ein Kettenfeld steht für den linken Nachfolger, das andere für den rechten Nachfolger. Abb. 14.1.1.2/4 zeigt die zu Abb. 14.1.1.2/3 gehörende Darstellung.

INDEXDATEI

RA	Schlüssel	KF li	KF re
1	Hansen H.R.	4	5
2	Maurer H.	-	7
3	Wirth N.	-	-
4	Date C.J.	-	6
5	Wedekind H.	2	3
6	Elson M.	-	-
7	Müller G.	-	-

HAUPTDATEI

INR	Autor	Titel
4711	Hansen H.R.	Wirtschaftsinformatik I ...
3812	Maurer H.	Datenstrukturen und ...
1029	Wirth N.	Algorithmen und Datenstrukturen ...
1744	Date C.J.	An Introduction to Database Systems ...
1222	Wedekind H.	Datenbanksysteme ...
1313	Elson M.	Data Structures ...
4712	Müller G.	Informationsstrukturierung in ...

Abkürzungen: INR = Inventarnummer; KF li = linkes Kettenfeld; KF re = rechtes Kettenfeld; RA = relative Adresse

Anmerkung:

Bei dieser Lösung entspricht die relative Adresse jeder Indexeintragung der relativen Adresse des dazugehörigen Datensatzes der Hauptdatei. Die Zahlen in den Kettenfeldern sind die relativen Adressen der logisch nachfolgenden Indexeintragungen, wobei „li" für den linken Nachfolger und „re" für den rechten Nachfolger steht. Der Strich („–") ist eine Endemarke.

Abb. 14.1.1.2/4: Realisierung eines binären Baumes durch einen Index mit zwei Kettenfeldern

Ein Baum heißt *sortiert*, wenn alle Knoten im linken Teilbaum kleiner und alle Knoten im rechten Teilbaum größer als ihre unmittelbaren Vorgängerknoten sind. Der Baum in Abb. 14.1.1.2/3 ist demnach sortiert.

Das *Suchen und Einfügen in sortierten binären Bäumen* sind überaus einfache Operationen. Beginnen wir mit dem *Suchen*. Wir suchen im

unten abgebildeten Baum den Knoten mit dem Schlüssel „Knuth D.E.". Zuerst vergleichen wir den Suchschlüssel mit der Wurzel. Es können vier verschiedene Situationen auftreten:

Situation	Aktion
* Der gesuchte Schlüssel ist kleiner:	Suche im linken Teilbaum.
* Der gesuchte Schlüssel ist größer:	Suche im rechten Teilbaum.
* Der gesuchte Schlüssel wird gefunden:	Beende Suchprozeß.
* Der gesuchte Schlüssel ist nicht vorhanden:	Beende Suchprozeß.

Die letztgenannte Situation kann nur dann auftreten, wenn der gesuchte Schlüssel kleiner oder größer ist und der Zeiger auf die Endemarke weist.

Bei der Operation *Einfügen* wird so die logische Position des Knotens ermittelt, und der Knoten wird an dieser Stelle eingefügt.

In Abb. 14.1.1.2/5 wird nach dem Knoten „Knuth D.E." gesucht. Der erste verglichene Knoten (Wurzel „Hansen H.R.") ist kleiner als das gesuchte Element; infolgedessen wird im rechten Teilbaum weitergesucht. Die nächsten beiden verglichenen Knoten („Wedekind H." und „Maurer H.") sind beide größer; es wird in den linken Teilbäumen (in den Teilbäumen mit den kleineren Knoten) weitergesucht. „Maurer H." hat keinen linken Nachfolger, der gesuchte Knoten ist nicht im Baum; „Knuth D.E." wird als linker Nachfolger dieses Knotens eingetragen (siehe Abb. 14.1.1.2/6)

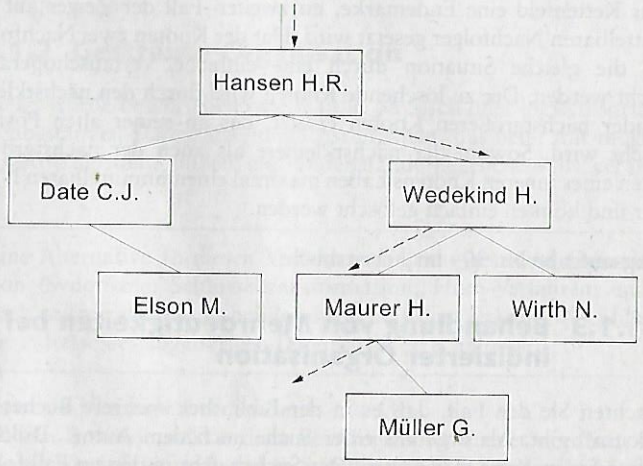

Abb. 14.1.1.2/5: Suchen in binären Bäumen (Autor „Knuth D.E.")

Physisch wird der neue Knoten am Ende des Datenbereiches hinzugefügt, seine logische Position im Baum wird durch Zeiger (Kettenfelder) realisiert.

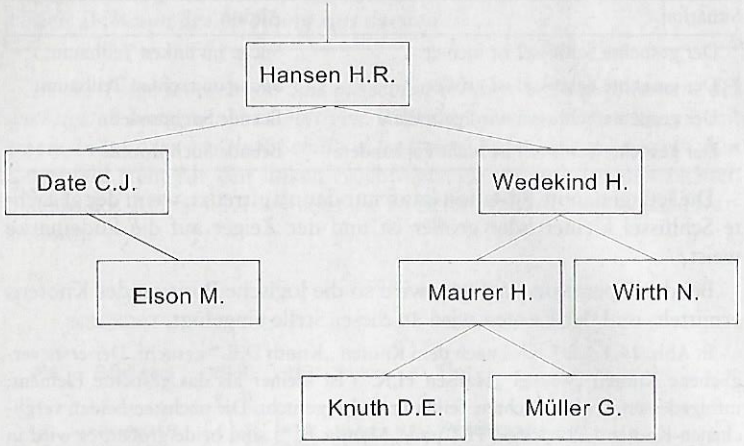

Abb. 14.1.1.2/6: Einfügen in binären Bäumen (Autor „Knuth D.E.")

Das *Löschen* in binären Bäumen ist dann einfach, wenn der zu löschende Knoten keinen oder nur einen unmittelbaren Nachfolger hat. In diesen Fällen kann der Knoten einfach eliminiert werden, indem im ersten Fall in das Kettenfeld eine Endemarke, im zweiten Fall der Zeiger auf den unmittelbaren Nachfolger gesetzt wird. Hat der Knoten zwei Nachfolger, kann die gleiche Situation durch eine einfache Vertauschoperation erreicht werden: Der zu löschende Knoten wird durch den nächstkleineren oder nächstgrößeren Knoten ersetzt, der an seiner alten Position gelöscht wird. Sowohl der nächstkleinere als auch der nächstgrößere Knoten eines inneren Knotens haben maximal einen unmittelbaren Nachfolger und können einfach gelöscht werden.

→ Übungsaufgabe Nr. 304 im Arbeitsbuch

14.1.1.3 Behandlung von Mehrdeutigkeiten bei indizierter Organisation

Betrachten Sie den Fall, daß es in der Bibliothek mehrere Bücher von „Dijkstra" gibt. Als Ergebnis einer Suche nach dem Autor „Dijkstra" erhalten Sie mehrere Datensätze. Das Suchen geht in diesem Fall folgendermaßen vor sich:

1. *Physisch sortierter Index*

   a) Binäres Suchen:
   Durch sequentielles Suchen im Index (auf- und abwärts) ab dem ersten zutreffenden Index und im Überlaufbereich findet man alle entsprechenden Datensätze. Die Größe des Überlaufbereichs muß hier besonders klein gehalten werden, da bei jedem Zugriff der ganze Überlaufbereich durchsucht werden muß. Reorganisationen in kurzen Zeitabständen sind daher unvermeidbar.

   b) m-Wege-Suchen:
   Hier werden sinnvollerweise zusammengehörige Indexeintragungen in denselben Blöcken liegen. Sonst gilt das gleiche wie im Fall 1. a).

2. *Logisch sortierter Index*

   a) Ketten:
   Bei Ketten kann nur sequentiell gesucht werden. Sobald der erste zutreffende Datensatz gefunden ist, wird sequentiell weitergearbeitet, bis sich der Schlüssel ändert.

   b) Bäume:
   Bei Bäumen sind gleiche Schlüsseleinträge erlaubt. Es muß nur sichergestellt werden, daß eventuell gleiche Schlüssel in benachbarten Knoten liegen. Der Suchvorgang kann, sobald der erste zutreffende Knoten gefunden wurde, in den darunterliegenden, benachbarten Knoten fortgesetzt werden.

Übungsaufgabe Nr. 305 im Arbeitsbuch ←

14.1.2 Gestreute Organisation

Bis jetzt wurde beschrieben, wie man in physisch oder logisch sortierten Datenbeständen Datensätze möglichst effizient, das heißt mit möglichst wenigen Zugriffen, finden und Datenstände gegebenenfalls verändern kann.

Eine Alternative zu diesen Verfahren stellt die **gestreute Organisation** (Synonyme: **Schlüsseltransformation, Hash-Verfahren;** engl.: hash organization) dar, bei der direkt aus dem jeweiligen Schlüssel die Adresse des zugehörigen Datensatzes errechnet wird.

Es treten hier *zwei Probleme* auf:

1. *Welche Funktion* soll für diese Berechnung verwendet werden?
2. Da die Anzahl der verfügbaren Speicherplätze in der Regel geringer ist als die der möglichen Schlüssel, muß eine Funktion gewählt werden,

die eine *Doppelbelegung* (Kollision; engl.: collision) einer Adresse zuläßt. Die Art der *Behandlung derartiger Kollisionen* beeinflußt die Zugriffsdauer wesentlich.

Durch die **Schlüsseltransformationsfunktion** (Synonym: **Hash-Algorithmus**; engl.: hash algorithm, hash code) wird aus alphabetischen, numerischen oder alphanumerischen Schlüsseln eine Menge von Adressen berechnet.

Abkürzungen: INR = Inventarnummer; RA = relative Adresse

Abb. 14.1.2/1: Gestreute Organisation

Bei nahezu allen Programmiersprachen ist es möglich, den internen Code (ASCII, EBCDIC ...; vgl. Abschnitt 9.2.2.2) jedes einzelnen Zeichens abzufragen. *Versuchen wir aus dem Schlüssel einen eineindeutigen (= umkehrbar eindeutigen) dezimalen Wert zu errechnen*, also einen Wert, aus dem wiederum der Schlüssel berechnet werden kann.

Da die Zeichen im allgemeinen im 8-Bit-Code dargestellt werden, können sie 256 verschiedene Werte (die Werte 0 bis 255) annehmen. Man kann also einen alphanumerischen Schlüssel direkt in ein Zahlensystem (vgl. Abschnitt 2.1.3.3) mit der Basis 256 übertragen.

Beispiel:

10stelliges alphabetisches Schlüsselfeld mit dem Inhalt |*HANSEN* |:

Buchstabe	|H|	|A|	|N|	|S|	|E|	|N|	|	|	|	|
EBCDIC-Wert	100	193	213	226	197	213	64	64	64	64

Wert: $= 200 \times 256^9 + 193 \times 256^8 + 213 \times 256^7 + 226 \times 256^6 + 197 \times 256^5 + 213 \times 256^4 + 64 \times 256^3 + 64 \times 256^2 + 64 \times 256^1 + 64 \times 256^0 =$

$9480489302785494981182 08$

Der errechnete Wert 9480489... entspricht dem Schlüssel |HANSEN |. Die Anzahl der möglichen Schlüssel kann berechnet werden, indem man für jeden Buchstaben den höchsten Wert einsetzt, den dieser annehmen kann (255). Bei einer Schlüssellänge von zehn ist das 255^{10}-1. Sie sehen, daß die Anzahl der möglichen Schlüssel weit größer ist als die Anzahl der in der Regel zur Verfügung stehenden Speicherplätze. Man braucht also eine Funktion (= Schlüsseltransformationsfunktion), welche die riesige Anzahl von möglichen Schlüsseln auf eine akzeptable Menge von Adressen abbildet.

Hierfür werden in der Literatur eine *Vielzahl von Hash-Algorithmen* (unter anderem Faltung, Abschneiden, Extrahieren) vorgeschlagen. Eine der bekanntesten Funktionen ist das *Divisionsrestverfahren*.

Beim *Divisionsrestverfahren* wird der (etwa auf obige Weise) errechnete, dem jeweiligen Schlüssel entsprechende numerische Wert durch die Anzahl der für die betreffende Datei reservierten Speicherplätze (n) dividiert. Der Rest dieser Division liegt zwischen null und n – 1 und wird als Speicheradresse verwendet. Es hat sich als empfehlenswert erwiesen, für n eine Primzahl zu wählen.

Nehmen wir zum *Beispiel* n = 1117. Dann wird dem Schlüssel |HANSEN | eine Adresse zwischen 0 und 1116 zugeordnet. Diese Adresse ist der Rest der Division von dem oben berechnetem Wert (9480489...) durch 1117, das ist die Zahl 403.

Da durch die Verwendung einer Hash-Funktion die Anzahl der errechneten Adressen geringer ist als die der möglichen Schlüssel (diese Reduzierung ist Sinn und Zweck dieser Funktion), kann es vorkommen, daß mehreren verschiedenen Schlüsseln ein und dieselbe Adresse zugewiesen wird. Die *Behandlung solcher Kollisionen* kann auf verschiedenste Art und Weise geregelt werden.

Eine Möglichkeit der Kollisionsbehandlung ist die sogenannte *direkte Verkettung* (engl.: chaining), bei der die kollidierenden Datensätze in einen Überlaufbereich der Datei geschrieben und dort untereinander verkettet werden. Der Nachteil dieser Methode ist, daß eine zusätzliche Kette zu führen ist, deren Durchsuchung zeitaufwendig werden kann.

Eine andere Lösung, die auf solche Verkettungen verzichtet, heißt *offene Adressierung* (engl.: open addressing). Hier wird im Kollisionsfall im Adreßbereich entweder in konstanten (lineares Sondieren) oder in quadratisch ansteigenden (quadratisches Sondieren) Abständen nach freien Speicherplätzen gesucht.

Beim *Hash-hash-Verfahren* (engl.: double hashing) wird im Kollisionsfall zur Suche eines freien Speicherplatzes wieder eine Hash-Funktion verwendet. Diese zweite Hash-Funktion muß sich von der ersten unterscheiden, da sonst eine neuerliche Kollision auftreten würde. Ansonsten ist

dieses Verfahren das geeignetste, um die Anzahl der Kollisionen gering zu halten.

→ Übungsaufgabe Nr. 306 im Arbeitsbuch

Dem *Vorteil des schnellen Zugriffs* (meist schneller als die „besten" Baumstrukturen) stehen *folgende Nachteile der gestreuten Organisation* gegenüber:

1. Sie ist fast *ausschließlich auf Attribute mit Primärschlüsseleigenschaft anzuwenden.* Die Anzahl der Kollisionen würde bei Mehrdeutigkeiten (wie bei Sekundärschlüsseln möglich) so enorm steigen (die Adreßberechnung von „IHANSEN |" durch die Hash-Funktion bringt immer den gleichen Wert), daß die Suche unvertretbar lange dauern würde.
   Deshalb ist zum *Beispiel* die gestreute Organisation in unserem *Bibliotheksverwaltungssystem* für die Realisierung eines direkten Zugriffs über das Attribut „Autor" nicht geeignet.
2. Die Größe der Hash-Tabelle, das ist der für die Datensätze reservierte Speicherbereich, kann nur mit erheblichem Aufwand geändert werden. Deshalb ist es *notwendig, die Anzahl der zu erwartenden Datensätze relativ genau zu kennen.* Bei einer schlecht gewählten Größe ist die Folge entweder eine Speicherplatzverschwendung, weil die Hash-Tabelle ist nur sehr dünn besetzt ist, oder schlechte Zugriffszeiten, da um so öfter Kollisionen auftreten, je dichter die Datensätze in der Tabelle liegen.
3. Die Datensätze können nur nach einem zeitaufwendigen *Sortiervorgang* sortiert ausgegeben werden; bei einer Baumorganisation ist hingegen jederzeit die Ausgabe der Datensätze in auf- oder absteigender Folge der Schlüssel möglich.
4. Bei allen Formen der sortierten Speicherung (physisch oder logisch sortiert) ist auch ein *„teilqualifizierter" Zugriff* möglich, das heißt, es kann auf einen Datensatz, von dessen Schlüssel nur ein Teil bekannt ist, zugegriffen werden. Es können zum Beispiel alle Datensätze, deren Schlüssel mit „A" beginnen, aufgelistet werden. Diese vor allem bei sehr langen Schlüsseln vorteilhafte Möglichkeit ist *bei gestreuter Organisation nicht gegeben.*

Beispiel:

Betrachten Sie wieder das *Bibliotheksverwaltungssystem*, bei dem über den Titel der Publikation zugegriffen werden soll. Für den Titel müssen mindestens hundert Zeichen reserviert werden. Bei Hash-Verfahren müssen immer diese hundert Zeichen eingegeben werden; bei jedem Tippfehler oder bei jedem fehlenden Leerzeichen wird kein Datensatz, kein Buch mit diesem Titel, gefunden. Bei sortierten Speicherverfahren genügen unter Umständen die ersten fünf oder zehn signifikanten Buchstaben, um die richtige Eintragung zu finden.

Übungsaufgabe Nr. 307 im Arbeitsbuch ←

14.1.3 Vergleich der beschriebenen Dateiorganisationsmethoden

In der folgenden *Übersicht* (Abb. 14.1.3/1) werden die *wichtigsten Merkmale der verschiedenen Dateiorganisationsmethoden* nochmals zusammenfassend dargestellt.

Dateiorganisationsmethoden	SEQUENTIELL	INDIZIERT			GESTREUT
		Relativ	Logisch sort. Index	Physisch sort. Index	
Charakteristische Merkmale	Es gibt nur die Hauptdatei	Auf die Hauptdatei kann mittels eines numerischen Wertes direkt zugegriffen werden.	Es gibt neben der Hauptdatei immer eine vorgelagerte Zugriffsdatei (indexdatei)		Die Adressen der Datensätze der Hauptdatei werden über einen speziellen Algorithmus aus dem Schlüssel berechnet.
Mögliche Speichermedien	Sequentiell oder direkt adressierbar	Direkt adressierbar			Direkt adressierbar
Suchstrategien	Sequentiell	Ermittlung des numerischen Wertes	Ketten: Sequentiell Bäume: ähnlich binärem Suchen	Sequentiell, binär, m-Wege	Suche nur bei Kollision notwendig
Reorganisation notwendig?	Nein	Nein	Nein	Ja	Nein
Für die Primär und/oder Sekundärschlüssel verwendbar?	Kein Schlüsselzugriff möglich	Primärschlüssel	Primärschlüssel und/oder Sekundärschlüssel		Primärschlüssel

Abb. 14.1.3/1: Gegenüberstellung der wichtigsten Merkmale der beschriebenen Dateiorganisationsmethoden

14.2 Datenbanksysteme

Betrachten wir wieder unser *Beispiel einer Bibliotheksverwaltung.* Die Buchrecherche, das Entlehnwesen und die Budgetverwaltung der Bibliothek sollen automatisiert werden. Werden nun diese drei Aufgabenstellungen unabhängig voneinander entwickelt, passiert folgendes:

1. Die Datenstrukturen der Programme sind verschieden, da sie optimal an die einzelnen Problemstellungen angepaßt wurden. Auch die Zugriffspfade unterscheiden sich. So wird bei der Buchrecherche sinnvollerweise die Möglichkeit bestehen, über den Autor direkt zuzugreifen. Realisiert werden kann dieser direkte