IMPLEMENTATION: Data Structures
Relational Information Systems Chapter 1.2
(Revised 99/9)
Files and Secondary Storage

January 26, 2007

Copyright ©1999 Timothy Howard Merrett
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation in a prominent place. Copyright for components of this work owned by others than T. H. Merrett must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to republish from: T. H. Merrett, School of Computer Science, McGill University, fax 514 398 3883.

The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

All implementations in this book will use secondary memory. The primary memory of a computer is "random access memory" (RAM): the time to find an item of data in RAM is comparable to the time to transfer it to the processing unit of the computer. Secondary memory, on the other hand, sacrifices access time to save money: the time to find an item of data is usually orders of magnitude greater than the time to transfer it to primary memory. (This transfer time for secondary memory, however, is often nearly the same as that for RAM.) In terms of turn-of-the millenium technology, figure 1 shows the tradeoff between these two characteristics for various types of device. We see that the access/transfer ratio increases as the cost decreases.

A significant aspect of secondary memory is size. We will be considering applications for which the data requirements potentially exceed RAM capacity by many times. Most of the secondary memory technologies in figure 1 allow the medium to be removed from the recording equipment, so that the storage capacity is effectively unlimited.

Secondary memory thus gives us the facility for large (because cheap) permanent databases. Despite the plummetting cost of RAM, magnetic medium costs have dropped even more rapidly, and application demands have grown even faster than the primary memory capacity of any computer. For example, a LandSat satellite images the earth in terabytes (10^{12} bytes, or a million megabytes), high-energy physicists talk about database capacities of petabytes (10^{15} bytes, or a thousand terabytes), and the genome of a single human is a record of about

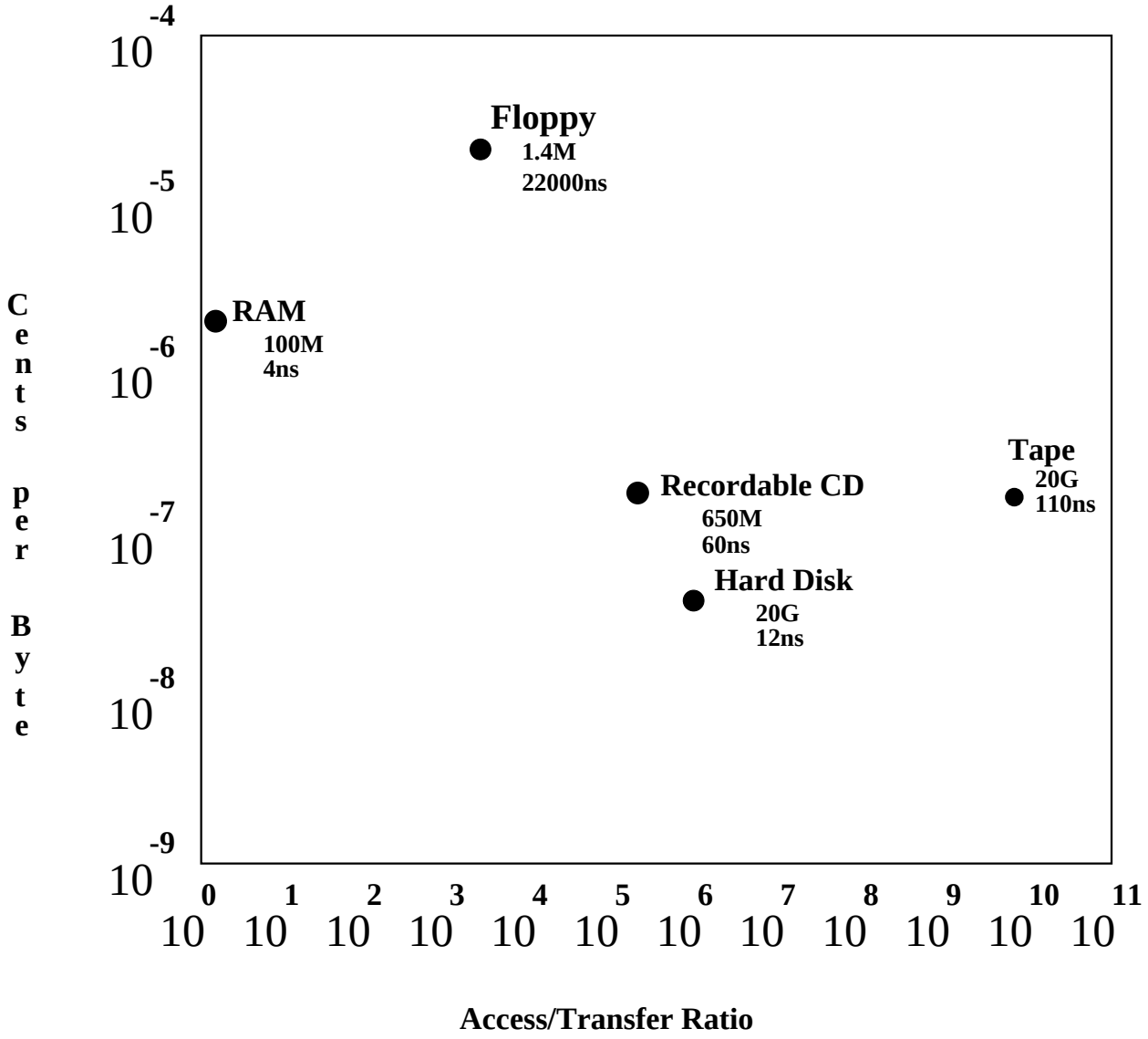
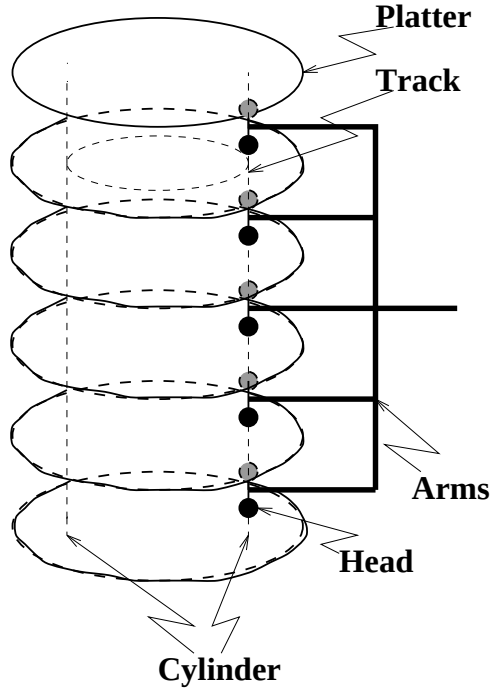


Figure 1: Cost vs. Access/Transfer Ratio for 1998 Memory Technology, Showing Typical Unit Capacities and Transfer Times per Byte.

e.g., Disks

- have been around a long time
- recently improved faster than other technologies



- – Latency @ 7200 rpm: $\lambda = 1/(2 \times 120) \sim 4$ ms.
  - – Arm movement, say, $\mu = 8$ ms.
- Access time $\lambda + \mu = 12$ms.
- With, say, $\beta = 100,000$ bytes/track, transfer is $1/\tau = 12$Mbytes/sec

So access/transfer $\rho = (\lambda + \mu)/\tau = 144,000$ (bytes which could have been transferred during time spent seeking the data).

Figure 2: The Difference Between Primary and Secondary Storage

a gigabyte; the file for all humanity is[1] six exabytes (6×10^{18} bytes). It is reasonable to suppose that there will always be a demand for secondary memory.

Figure 2 shows how one form of secondary storage, the magnetic disk, is put together, and, consequently, shows the large access/transfer ratio which is the price we pay for the monetary cheapness of secondary storage.

Based on figure 1, the book assumes that the price to be paid for cheapness and permanence is a high access/transfer ratio. This ratio is central to our treatment of data on secondary memory. Because access is expensive while transfer is cheap, data must be transferred from secondary to primary memory in *blocks* (or *pages*: we will use the words synonymously) of hundreds or thousands of bytes, and it must be organized so that each block is used as fully as possible while in RAM.

Transferring a block in order to learn the contents of only a few bytes is not economical if a great deal of information is to be examined in this way. Thus, much of the processing that we discuss in this book will require data to be *clustered* so that the data can be read and written with a minimum number of accesses to each block.

Because of this fundamental difference in memory organization between secondary and

[1] as of Oct. 12, 1999

primary memory, new data structures, new algorithms, and even new languages are required. In a significant sense, computer science must be reinvented for secondary memory.

We next examine in more detail the characteristics of available types of secondary memory. Then we discuss the logical organization of data into files. Finally, we devote a section to each of the three categories of data structures for secondary storage, according to their access complexity: sequential, logarithmic, and direct.

1 Secondary Storage

Figure 1 shows the types of memory we will consider in this book. There are three categories: RAM, which we are not concerned with; tape; and circulating memories. This last includes the disk memories (hard, floppy, and optical), drum, and even technologies which have not panned out commercially such as magnetic bubble memory. It is a category which has been important since the 1940s and which shows no sign of being bypassed.

1.1 Circulating Memories

Circulating memories each have a characteristic rate of revolution, which may be a visible revolution as for the disks (300–7200 r.p.m.) or a circulating current of magnetic "bubbles" ($\sim$ 7500 r.p.m.). They therefore have a *latency time*, which is how long one must wait before the data comes around. The *mean latency time* is the time for half a revolution (50–4.2 msec. for disks, $\sim$ 4 msec. for bubbles).

Drums (and bubbles) have no further delay: they are "head-per-track" devices in which many revolving loops of data are examined in parallel. The disk devices, on the other hand, rely on a single head to handle many tracks: each *track* is a circular band on the spinning platter, and the *head* is a reading/writing device on a mechanical arm which can move in and out between the concentric tracks. For disks there is thus a delay associated with the motion of the arm from one track to another. The averge arm delay for disks can be from 7.8 to 95 msec. Hard magnetic disks usually have a more complicated structure than floppy or optical disks, with several disk surfaces on a common spindle, and a rake of arms moving simultaneously from one set of tracks (one track per surface) to another. Thus the natural grouping of tracks is not in terms of the few hundred on each surface, but in terms of tracks of common diameter on the ten or so surfaces. This group of tracks is called a *cylinder*, and the mechanical arms are considered to move from cylinder to cylinder.

The track is thus a notion common to all circulating memories, namely the loop of data that can be read or written by one head in the period of revolution. The *track length* limits the size of the blocks of data that can be transferred to and from primary memory. This can run from a few thousand bytes on a floppy to hundreds of thousands on a hard disk or optical disk. In most cases, this size is so great that other considerations limit the block size. One such consideration is *sectoring* of the track into smaller units, as is usually done with floppy disks and other media for small computers.

Blocks of data are normally transferred to and from a *buffer* in primary memory, and buffer capacity is a limiting factor on the blocksize. The most important consideration is the tradeoff between the amount of data transferred with each block and the number of accesses required.

To fix our ideas for the discussion and calculations of this book, we will describe two magnetic disk devices and one optical disk whose specifications, given in figure 3, are about state-of-the-art for 1999. One is a removable hard disk, DISK2000, one is a two-sided 3 1/2" floppy, FLOPPY2000, and one is an optical disk, RCD2000.

	τ TRANSFER TIME/BYTE	ρ ACCESS/ TRANSFER RATIO	σ ROTATION SPEED	λ AVERAGE LATENCY	μ AVERAGE ARM DELAY
DISK2000	8.3nsec.	1,440,000	7200 rpm	4.2ms	7.8ms
RCD2000	0.7μsec.	240,000	600–300 rpm	75ms	95ms
FLOPPY2000	22μsec.	4,000	600 rpm	50ms	38ms

	β BYTES/ TRACK	γ TRACKS/ CYLINDER	ν CYLINDERS /UNIT	TOTAL CAPACITY
DISK2000	1,000,000	10	20000	20GB
RCD2000	140,000–280,000	1	3095	650MB
FLOPPY2000	4608	2	160	1.4MB

Figure 3: Specifications for Magnetic Disk Units to be Used in This Book

The critical quantity for each of these disks is the *access/transfer ratio*, which is the abscissa of figure 1. It is the ratio of the time required to find the data (rotation latency plus arm movement) to the time required to transfer a byte of data,

$$\rho = (\lambda + \mu)/\tau,$$

and may be thought of as the number of bytes that could be read while the disk is searching for the beginning of the block.

Circulating memories are often called "direct access storage devices", because the heads can move to any cylinder and start reading or writing once the data has come around. (They must not be abused by being treated as RAM: we have seen the great differences in access/transfer ratio.)

Exercise 1.2-1 Confirm that the following relationships hold in figure 3:
$\lambda = 1000 \times 60/(2 \times \sigma); \tau = 60 \times 10^6/(\sigma \times \beta), or\ \tau = 60 \times 10^9/(\sigma \times \beta); \rho = 1000 \times (\lambda + \mu)/\tau, or\ \rho = 10^6 \times (\lambda + \mu)/\tau.$
What is their significance?

Exercise 1.2-2 How do we calculate the total capacities of DISK2000, FLOPPY2000, and RCD2000?

1.2 Tape

Magnetic tape is in a category of its own. It does not have the direct-access capability of circulating memories; the avereage wait to access data somewhere on a 300-foot reel of tape at 80 inches per second is half the tape, or 45 seconds. It is a sequential memory, and characterized by a very high access/transfer ratio. Once the data has been found, however, the transfer rate is very good and can compete with any of the circulating memories. With a recording density of 10 megabits per inch, a nine-track tape (which transfers 9 bits in parallel to give a full byte plus a parity bit) at 80 inches per second transfers 800 megabytes a second, a transfer time of 12.5 nanoseconds per byte. Tapes are simple and cheap, which, together with their data transfer rate, makes it likely that they will continue to be used whenever sequential processing is appropriate.

Tape imposes no limits on the block size of data written on it, except that between each block on the tape there is a gap of empty tape required to accommodate the deceleration of

	τ TRANSFER TIME/BYTE	ρ ACCESS/ TRANSFER	σ TAPE SPEED	λ AVERAGE LATENCY	REWIND TIME
TAPE2000	12.5nsec.	20 G	80 ips	25 sec.	25sec.

	δ RECORDING DENSITY	ι INTER-BLOCK GAP	ϕ TAPE LENGTH	TOTAL CAPACITY
TAPE2000	1MBpi	0.2 in.	3250 ft.	40 GB

Figure 4: Specifications for Magnetic Tape Units to be Used in This Book

the tape when it stops between blocks and the acceleration when it starts up again. Such an interblock gap could be about 0.2 inches, or the equivalent of 2 megabytes. Thus a blocksize of less than this many bytes will result in a half-empty tape. The longer the block the more economical the usage of tape.

As with DISK2000, RCD2000, and FLOPPY2000, we invent a tape unit, TAPE2000, for the discussion of this book. Figure 4 describes the device.

Exercise 1.2-3 Confirm that the following relationships hold in figure 4:
$\lambda = 12 \times \phi/(2 \times \sigma); \tau = 10^6/(\delta \times \sigma); \rho = 10^6 \times \lambda/\tau = 12 \times \phi \times \delta/2.$
What is their significance?

Exercise 1.2-4 How do we calculate the total capacity of TAPE2000

2 Files

Figure 5 shows the basic concepts and quantities associated with files. Logically, a file is a set of *records*. Physically, the records are grouped into *blocks* or, synonymously, *pages*. The example shows a completely full file, with no wasted space (or space available for future insertions): the *load factor* = 1.0. It also shows a file of *fixed-length* records, with an exact integer number (2) of them occupying a block. For such fixed-length records, which are mainly what we will consider in the book, we can work with simple quantities:

N	number of records in the file
n	number of blocks in the file
b	number of records in each block
α	load factor $= N/nb$

The block is the unit of access, and its size is determined by considering the access-transfer ratio, the RAM available for buffers, the characteristics of the secondary storage, and the nature of the application.

The records are divided into *fields*, such as *Ord#, Cust*, and *Sales* in figure 5. Often one or more of these fields is used to identify which record(s) will satisfy a search. For instance, *Ord#* could be used to identify the unique record with *Ord#* `= 1`, or *Cust* could be used to identify the two records with *Cust* `= NYC`. A field, or set of fields, which identifies a unique record is called a *key*. A key or any set of fields used in a search is often called a *search key*.

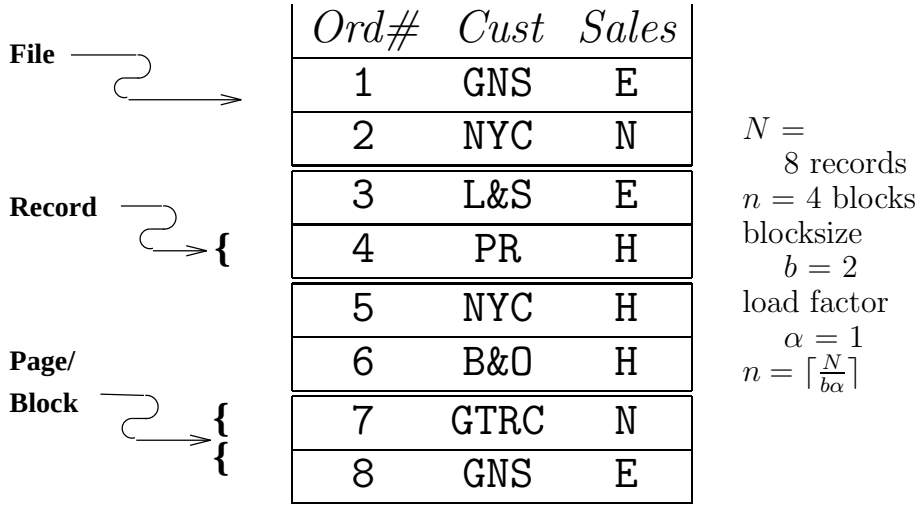


Figure 5: Characteristics and Quantities for Files

2.1 Access Speed

Apart from these notions and quantities which are common to all files, there are four different classifications which one can make on files. These classifications characterize the file structures and the algorithms used to create, search, and modify them. They also characterize applications which require file structures, and so can be used to identify good structures for any particular application.

The first classification is according to access speed. Files may be grouped by access complexity: linear, logarithmic, and constant. These groupings correspond to sequential, tree, and direct files, respectively.

Sequential files distinguish the first record, following which all subsequent records may be accessed only by reading sequentially through the file: the cost to read all N records is, of course, to read all n blocks; the cost to read only one record can vary from 1 (to read the first) to n (to read the last), with a mean of $n/2$ (on average, half the file must be read). (This mean cost depends on aspects of the sequential file, and will be refined later.)

Magnetic tape is a good medium for sequential files, but so is circulating memory. A sequential file on disk need not be stored as a sequence of contiguous blocks: the sequence could be maintained by storing addresses (pointers) for subsequent blocks (but *not* pointers to individual records).

Logarithmic files structure the blocks as a tree, which in a search for any one record need only be read from root to leaf (at most), a cost of some $\log n$ accesses. Figure 6 shows a popular example, the *B-tree*. In this example, the blocksize is 2 records, and the *fanout* could vary from 2 to 3. The maximum number of accesses in any search for a single record is 3, which happens to be $\lceil \log_2 n \rceil$ for the $n = 6$ blocks holding the $N = 8$ records. Later analyses will quantify costs much more precisely.

On secondary storage, the base of the logarithm is significant (in RAM it is always 2) and B-trees and other logarithmic files are used with large bases.

Another logarithmic file is the digital tree, or *trie*, in which the decisions about which subtree to go to, in searching from the root for a particular record, are made by looking at the

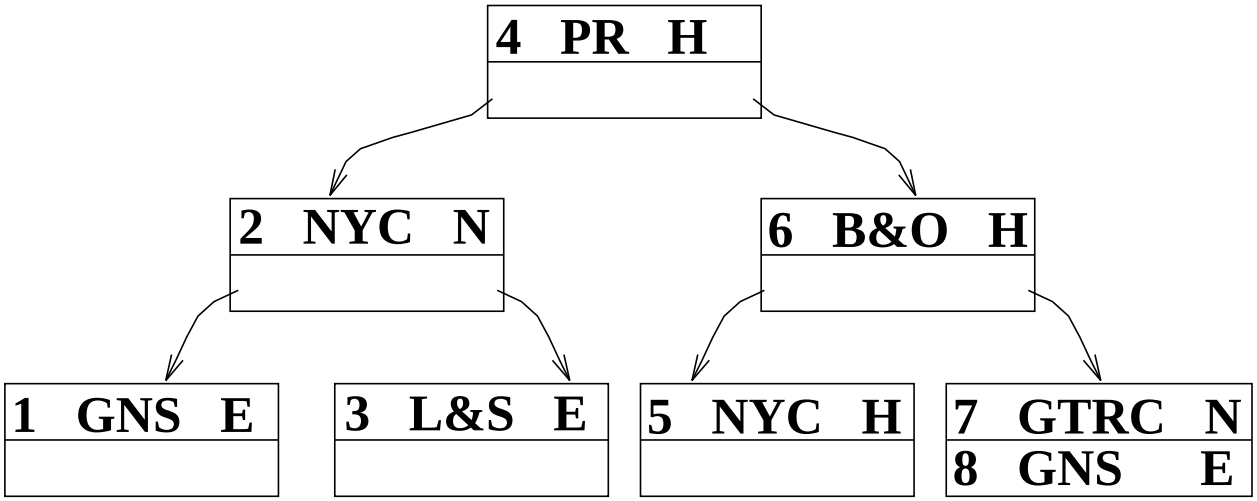


Figure 6: Logarithmic Files: a B-tree

bits of the search key, instead of by comparisons with key values stored in the nodes. Thus, tries have almost no storage overhead, and, because many keys have common prefixes, and therefore share paths from the root, tries can offer tremendous data compression. Figure 7 shows a trie which has been paged for secondary storage, that is, divided into blocks.

Another property of tries is their capability for *variable resolution* when the bits are ordered with the most significant bit at the root. Figure 8 shows a "kd-trie" used for spatial data and the variable-resolution images that result when the trie is scanned to variable depths from the root.

Tries were originally proposed for text data, and figure 9 shows a "truncated" trie and a "PATRICIA" trie representing all the substrings of the (very brief) text, `mocha`.

The third and last group of file structures in the category of access speed is **direct-access files**. An example is multipaging, which views records as points in a space, and attempts to partition the space rectilinearly so that the same number of records fall in each partition, or page. Figure 10 shows a two-dimensional partitioning of eight records into two pages ($N = 8, n = 4$). The records are abbreviated from, say, `1 GNS E`, to `1` in the row headed `GNS` and in the column labelled `E`. The row headings give all the values for the second field, and the column labels give all the values for the third field. In figure 10, there should be two records per page ($b = 2$), but only two of the four pages have two records; one page has three and the fourth page has one. So either the page size will have to be bigger, $b = 3$, or one of the pages will have to overflow in an implementation.

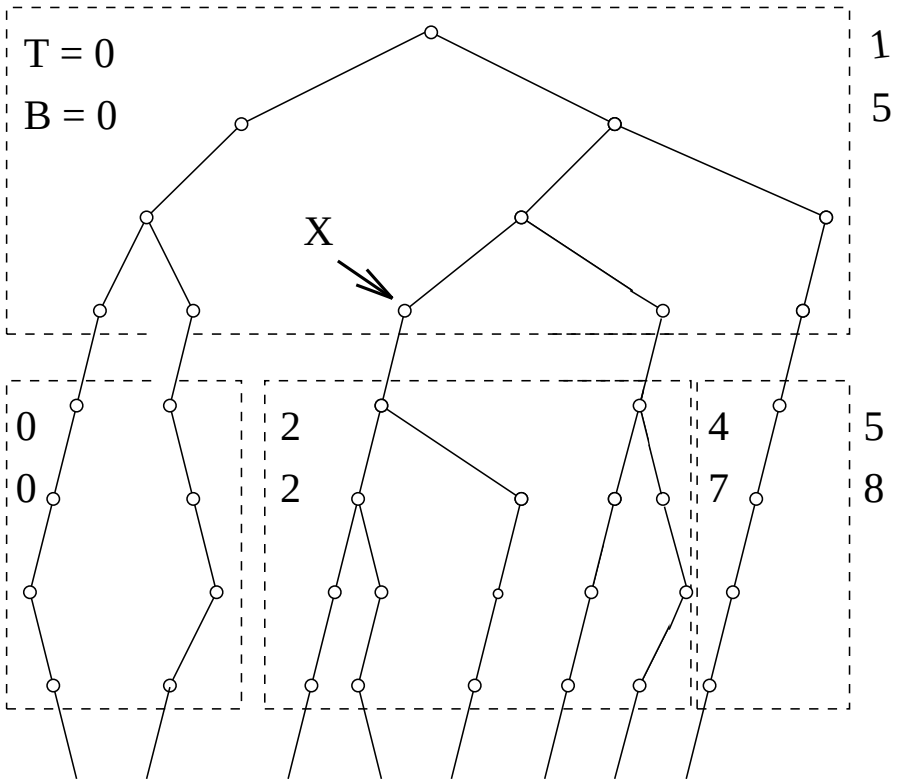
Access to a single record is easy in multipaging, because the partition is rectilinear: look up the desired values along the axes (e.g., `GRTC` and `N`), and retrieve the indicated page. Multipaging also supports "partial match" queries, such as for `GRTC` only: retrieve all the pages in the row; other pages are irrelevant and need not be retrieved.

Finally, files can be **hybrids**. The multipaging of figure 10 does not readily permit retrieval on the first attribute, `1, 2, ...` Figure 11 shows a B-tree "index" to this attribute in the multipage file. Instead of data, the index contains pointers to the page numbers.

Another way of hybridizing is to combine one file structure with *ideas* from another. For example, *z-order* is the one-dimensional ordering of multidimensional data that is induced by traversing a kd-trie from left to right. This idea can be combined with a one-dimensional

Sample data:

```
00000011
00101100
10000000
10000101
10001000
10100000
10101100
11000000
```



(Digital trees Information re*trie*val)

Figure 7: Logarithmic Files: a Paged Trie

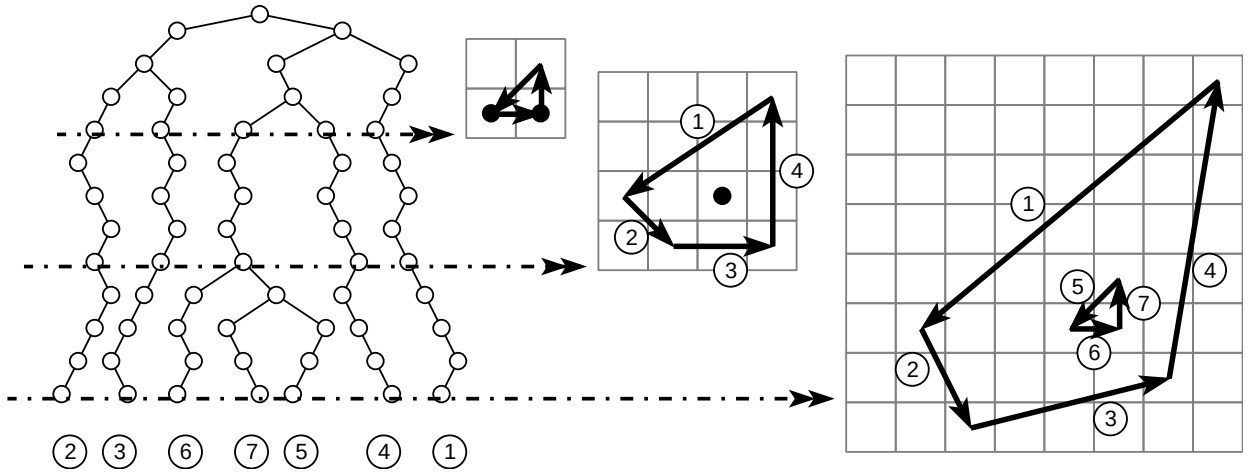
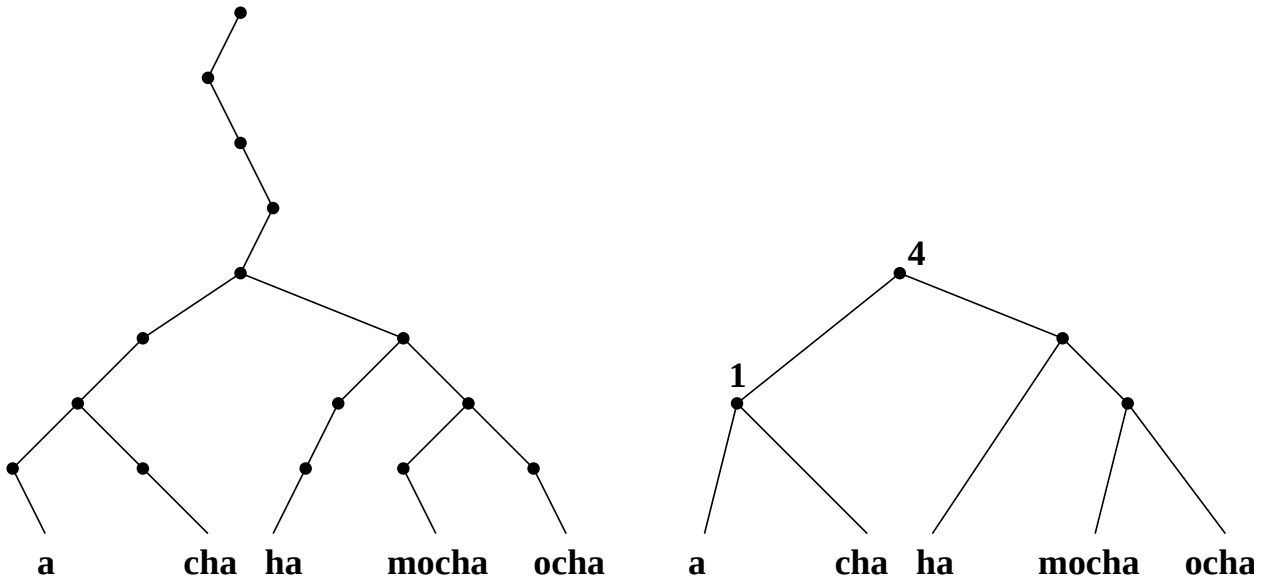


Figure 8: Kd-Tries and Variable Resolution



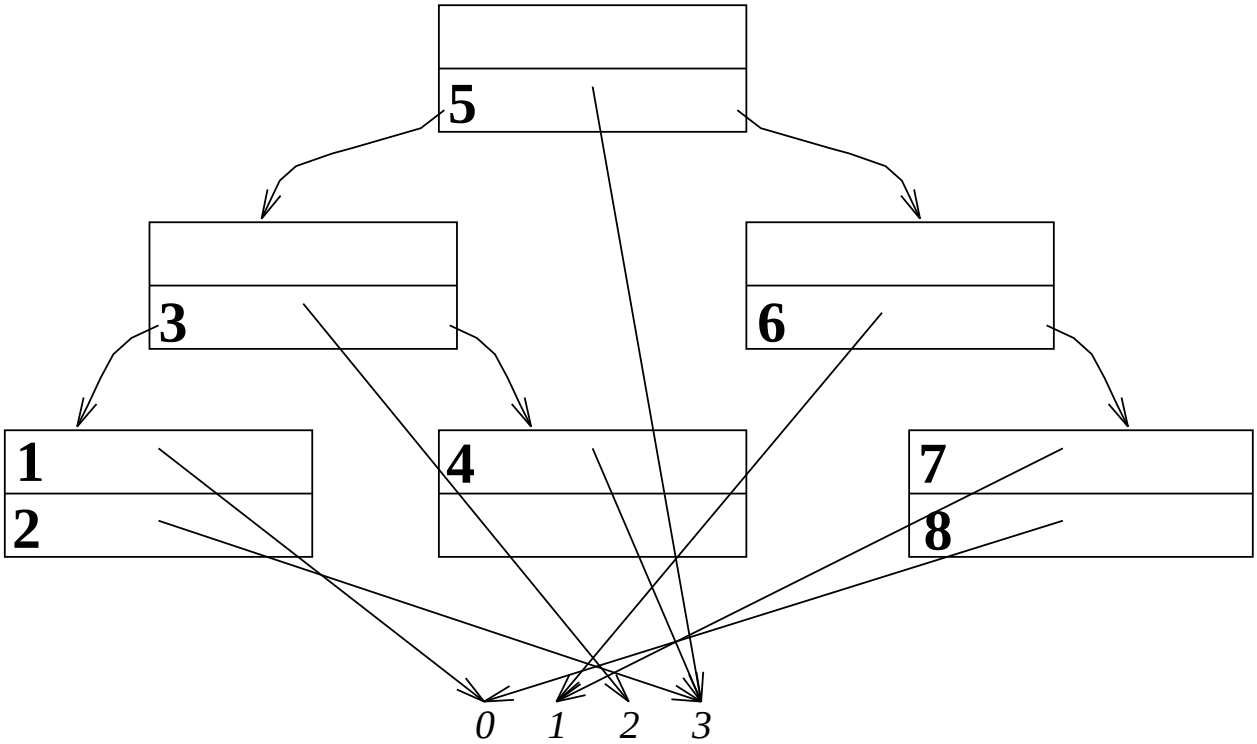
1) Truncated Trie **2) PATRICIA Trie**

Sample "text":
`mocha :` `01101101011011110110001101101000011000 01` with "starts" every eight bits.

Figure 9: Truncated Tries and Text Data

B&O		6	
GNS	1,8		
GTRC			7
L&S	3		
NYC		5	2
PR		4	
	E	H	N

Figure 10: Direct Access Files: Multipaging



B&O	*0*	6	*1*
GNS	1,8		
GTRC			7
L&S	3 *2*		*3*
NYC		5	2
PR		4	
	E	H	N

Figure 11: A Hybrid of B-Tree and Multipaging

access method, such as a B-tree, to give it multidimensional capability.

In this book we will use this spectrum of access speeds to organize the discussion of files.

2.2 Activity, Volatility, and Symmetry

The remaining three classifications are simpler. **Activity**, often known as *selectivity*, which we take to be a synonym, is a precise quantity, the ratio of how many records are needed to satisfy a particular query or transaction to the total number of records in the file. Thus, if a student file contains 30,000 records for 30,000 students, and an application wishes to know the particulars for one student, the activity is 1/30,000, or 0.003%. Another application, using the same file, may need to calculate the mean GPA (grade point average) for all students: here the activity is 1 (100%), since all records are needed. Thus, a file designed to support both of these needs must be capable of both low activity and high activity.

The practical distinction between low and high activity is to consider that a sequential file supports high activity while a direct file supports low activity. It is remarkable that the break-even activity, at which both types of file are equally good, is only a small fraction of a percent, a result we will show as soon as we can analyze both file structures. This says that sequential files are better than anything else over a surprisingly large range of activity; it is a fortunate fact, if obscure to many professionals, because sequential files are also by far the easiest to build.

Volatility is less quantifiable. It refers to the rate of additions to, deletions from, or changes in the file required by an application. We can loosely characterize volatility as low or high, and even suggest that "low" often means "zero" and "high" means anything else (because file structures are sometimes either completely intolerant of volatility or else entirely flexible). But this is simplistic, and there is no silver bullet that we can use to avoid thinking about the particular circumstances of file and application. For instance, a file of stock market prices may grow rapidly (high volatility) as end-of-day prices or daily highs and lows are added, but the application may require only a running average, in which case a very simple file structure could be used which would not normally be thought of as a structure which supports high volatility.

Symmetry pertains to the number of different search keys supported by a file structure, and their search costs. For example, a printed telephone book is not symmetrical because it may be searched by name logarithmically, but may be searched by address or by phone number only linearly. (This difference is enough to make the latter two impossible in practice.) A single file structure which would support equally rapid access by both name and phone number, independently, would have higher symmetry. This measure is also not quantifiable and we tend to distinguish only low and high symmetry, with "low" usually meaning that the file is accessible on only one key (which may be a single field or a combination of fields), and "high" meaning that more than one key may be used with equal effectiveness. Clearly, file structures such as multipaging and kd-tries play a role here.