addresses of documents indexed by the given descriptor. As we saw, in some cases this address is only the beginning of chain of blocks containing addresses of documents indexed by this descriptor.

As we mentioned earlier, an index file consists of 800 short (about a dozen bytes) records of fixed length. This file is used to search for descriptors contained in a query formulation. Because an index file is needed for a search based on any search request, it is read from disk into primary storage where it remains during the duration of search for all search requests. Any index file is always sorted, which substantially increases the speed of the search (compared to a sequential search). The methods that allow the speed of the search to increase (for example, a binary search or B^+ tree structure) are discussed in great detail in any textbook dealing with data structures (see, for example, Kruse, 1994). Therefore, we will not discuss them in this book. For now it is important (and sufficient) to know that such methods exist.

The occurrence frequency of a descriptor is needed for several purposes. First, it is used in the subsystem for indexing search requests. It also could be used to speech up searches for documents that match subrequests consisting of more than one descriptor. For example, to find an intersection of two sets of document numbers (or addresses) corresponding to two different descriptors and having cardinalities, say, 7 and 100, it is much faster to perform 7 searches on a set of 100 sorted numbers than 100 searches on a set of 7 sorted numbers. For other approaches to improving performance in inverted files see, for example, (Moffat & Bell, 1995).

So, the two files, illustrated in Figure 8.8, implement the idea of inverted file organization and are quite sufficient for a subject search. The third file, as noted previously, is a file of texts, which is indispensable in forming the output. The main ideas connected with its implementation were discussed earlier. It should be emphasized once again that inverted file organization is feasible for a Boolean search. It cannot be used, for example, for calculating the proximity between the search request vector and the document vector. This explains why in the beginning we pointed out that any known file organization can be used in the Boolean system.

8.6 Conclusion

This chapter is intended for those interested in the fundamental ideas of implementing the BSR (block of storage and retrieval) design rather than in the technical details. This is why we focused our attention on such ideas. It should be noted that developers of new IR systems are not always directly involved in the creation of BSR. Since the 1960s several companies (IBM, for example) proposed standard software packages for implementing BSR. With changing technology new ideas have emerged, and new packages as well. Currently, the inversion approach to providing multikey access is being used as the basis for physical database structures in commercially available database systems, including several relational systems (such as IBM's DB2, Relational Technology's Ingress, Oracle, Inc.'s Oracle, Intel's System-2000, and Software AG's Adabas). These systems were designed to provide rapid retrieval to data records via as many inversion keys (descriptors) as the designer cares to identify.

In concluding this chapter, we emphasize once again that any BSR design ultimately results in the same output. In the not-too-distant future the increasing speed of computer operations, particularly the advancement of parallel processing, will certain improve IR system performance, although even now users have no complaints about the speed of a search. It is probably for this reason that researchers emphasize improving the quality of the search by focusing on its recall and precision rather than on search speed in the IR system (as is clearly seen in periodic literature). Currently, one of the most active areas of research is the creation of effective feedback methods to improve the quality of a search. The feedback process and its implementation are the subjects of our next chapter.

References

Folk, M. J., & Zoellick, B. (1992). *File structure* (2nd ed.). New York: Addison Wesley.

Grosshans, D. (1986). *File systems: Design and implementation*. Englewood Cliffs, NJ: Prentice Hall.

Horbron, T. R. (1988). *File systems: structures and algorithms*. Englewood Cliffs, NJ: Prentice-Hall.

Kruse, R. L. (1994). *Data structures and program design*. Englewood Cliffs, NJ: Prentice Hall.

Moffat, A., & Bell, T. A. H. (1995). In Situ generation of compressed inverted files. *Journal of the American Society for Information Science, 46*(7).

Wartik, S., Fox, E. A., Heath, L., & Chen, Q. (1992). Hashing algorithms. In W. B. Frakes & R. Baeza-Yates (Eds.), *Information retrieval: Data structure and algorithms*. Englewood Cliffs, NJ: Prentice-Hall.

Bibliographic Remarks

The readers who are not familiar with the material typically taught in courses on data structures and file organization and who are interested in learning more about the subject are referred to the following books.

Grosshans, D. (1986). *File systems: Design and implementation*. Englewood Cliffs, NJ: Prentice Hall.

Kruse, R. L. (1994). *Data structures and program design*. Englewood Cliffs, NJ: Prentice Hall.