In explaining inverted files, we concentrated on the idea of organizing such files and their usage. We might say that this is a logical presentation rather than a technical one. This presentation is probably quite sufficient to explain the approach in general. Yet because inverted files are used (in fact, probably by all functioning IR systems) and because it provides major resources for enhancing its performance, we consider it expedient to also describe the main techniques of realizing this approach. Actually, inverted files are described in most textbooks containing sections on file structures, but this is done without considering such an important application as the IR system. Here we will adapt well-known descriptions of inverted files to their application in IR systems.

At the beginning of this chapter we showed that document text and its document profile (see the record in Figure 8.1) are sufficient for conducting a search and forming the output. All the ideas considered subsequently only help to reduce the time spent on the given processes. This is true of the very idea of inverted files as well as the techniques used in implementing them. We started explaining the inverted file by describing the record in Figure 8.5. But technically it is inconvenient to create a file of such 800 records, primarily because some records will have only a few fields with numbers (addresses) of respective documents, whereas others will have thousands of such fields. Furthermore, it is impossible to know beforehand how long a particular record will be, and if new documents are added to the existing search file, it is not clear which records are likely to change and to what extent. Therefore, the idea presented earlier is realized in a somewhat different manner.

Recall that the first approaches to increase speed of a search were based on dividing the original record (Figure 8.1) into two parts (Figure 8.3). On the basis of these two types of records, two files were created: a file of document profiles and a file of documents. To realize an inverted file organization, the idea of creating several files is also used but in this case three files are created. The first file is the descriptor file. The record in this file consists of three fields. Field 1 contains a descriptor, field 2 contains the number of documents indexed by this descriptor (this number is the occurrence frequency of the descriptor in the collection of documents), field 3 contains the address of the block containing the addresses of documents whose document profiles include the descriptor from field 1. Returning to our example, it is easy to see that the first file consists of 800 short records (see Figure 8.8a). This is called an index file, and it has many advantages. An index file is quite small and could easily fit into primary storage where a fast search for a needed descriptor could be performed. Also the size of the file does not depend on the number of documents in the collection. The second file consists of a number of blocks where each block contains the addresses of those documents whose document profiles include a given descriptor.

In the simplest case, the second file will contain only 800 blocks (1 block for each descriptor if we assume that there are 800 different descriptors). However, it is possible (and quite likely) that some descriptors have very low occurrence frequencies while others have very high occurrence frequencies. In this case, some blocks might contain several lists of addresses corresponding to different descriptors (to save space), whereas some lists could span more than 1 block (corresponding to descriptors with very high occurrence frequencies). There is a well-known computer science technique to deal with such situations, and it uses linked lists and pointers. We will illustrate the idea of this method on the case where several blocks are used for addresses of documents whose document profiles contain the same descriptor, that is, the case of a descriptor with very high occurrence frequency. In Figure 8.8b every block is represented in the form of a record.

In our example, descriptor C71 has very high occurrence frequency and the addresses of all the documents whose document profiles contain C71 do not fit into one block. The address for the first block containing these addresses is 86, which is pointed to by the third element in the index record of descriptor C71. This element is called a pointer. The continuation of the list of addresses for documents whose document profiles include descriptor C71 is contained in block with address 911. This block is pointed to by a pointer at the end of block 86. The structure of block 911 is the same as that of block 86. The addresses of documents whose document profiles contain descriptor C71 are stored at the beginning of the block and at the end of the block—the pointer to block 1221. Block 1221 has the same structure, but because it is the last block containing addresses for descriptor C71, there is no need to point to another block with addresses. This is typically indicated by including a so-called null pointer, which identifies the end of the list. It is clear now why in the index file only one address is needed for each descriptor. This is the address of the first block containing the

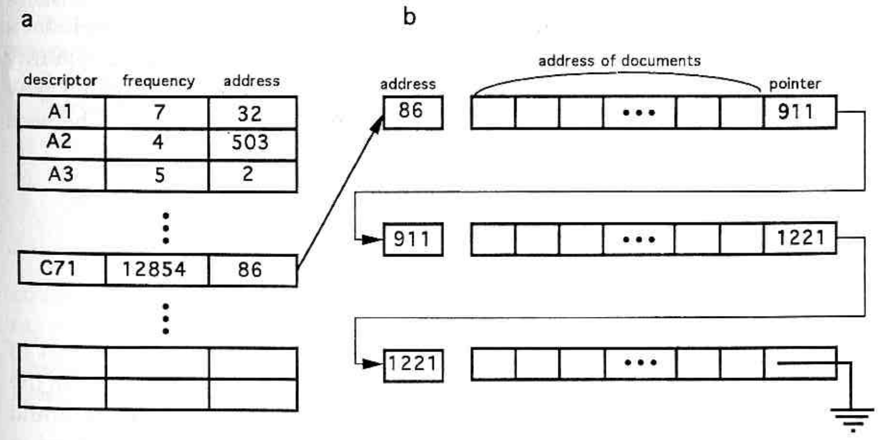

Figure 8.8
(a) Index file and (b) linked list.