データベースシステム 101−10
（１９９５．１．２７）

拡張可能類名表記を用いた類似キー検索ファイル

平出基一　　田中栄一
日本IBM　　神戸大学

本文は拡張可能類名表記とB^+-木を用いて類似キーの検索ができるファイルの一構成法について書いている。拡張類名表記を用いると、従来の類名表記を用いたときに比べて、１つの類名表記に属するキーの数の分散が少なくなり、記憶利用率が良くなる。230,188 個のキーで実験したところ、記憶利用率は約 66 %であった。長さ 6 の入力キーのとき、類似キー検索時のページ読み込み回数は約 30.2 回であった。紙数の都合で理論的検討の記述は省略したが、実験値は理論値とよく一致した。

A SIMILAR KEY SEARCH FILE USING EXTENDIBLE CLASS NAME EXPRESSION

Motoichi Hirade　　**Eiichi Tanaka**
IBM Japan　　**Kobe University**

In information retrieval there is a great demand for a file that can retrieve similar keys to an input key. However, only a few researches have been done for this kind of files. This paper proposes a file of this kind using extendible class name expressions and based on a B^+-tree. Therefore, the file can easily insert and delete keys without lowering its storage utilization. The experiment using 230,188 keys with length $1 \sim 16$ shows the good performance of the file. That is, the file can retrieve, insert and delete keys with less disk access than an original B-tree. The number of page reads for searching similar keys is around 30.2 in case of an input key with length 6. The storage utilization of the file is about 66%. This value is very close to the average storage utilization of an original B-tree. The performance of this file is analyzed and the theoretical evaluation shows a good coincidence to the experimental value.

1 Introduction

In a B-tree and its variants [1] - [4], [6], [7], [14] - [16], [18], retrieval, insertion and deletion of a key can be performed with a small number of disk access. A B-tree keeps storage utilization at least 50% and on average 69%. The analyses [4], [7], [15], [16], [18] of a B-tree and B^+-tree show high performance by analyzing the height of a tree and storage utilization. The method of increasing storage utilization using partial expansions [8] is proposed, and analyzed the expected performance of B^+-trees with partial expansions [17]. A B-tree and its variants cannot search similar keys to an input key. However, the robustness of a file is required in the following cases. (1) Incorrect data are stored in a file. (2) A file has incorrect structures. (3) Users want to list all similar keys to a key.

Among these problems, to cope with the case (2), several methods of detecting and correcting pointers and indexes errors of a B-tree have been proposed [5], [12], [13]. For the problems of (1) and (3), a hierarchical file using class name expressions has been proposed [10], [11]. However, this file is not constructed so that the insertion of a key can not be easily done. Furthermore, the storage utilization of the file becomes lower, when successive deletions occur.

In this paper, we propose a file structure for searching similar keys to an input key, and describe the method of operations. The file structure is a tree structure combined a hierarchical file using class name expressions and a B^+-tree. The proposed file structure has following features. (4) A retrieval, an insertion and a deletion of a key can be performed with a small number of disk access like a B-tree. (5) Search for similar keys can be performed like a hierarchical file using class name expressions.

2 Clustering the Set of Keys

The set of keys should be clustered in order to search similar keys efficiently. In this section we will describe two methods of clustering the set of keys. The one uses class name expressions [10], [11], and the other uses extendible class name expressions.

2.1 Class Name Expressions

Assume that the set Σ of alphabet is divided into the following two classes. A = {a, b, c, d, e, f, g, h, i, j, k, l, m}, B = {n, o, p, q, r, s, t, u, v, w, x, y, z}.

A and B are called *class names.* If we write "apple" using the above class names, we have ABBAA. We call this *the class name expression* (CNE) of "apple". The CNE of "knock" is also ABBAA. A set of keys can be clustered using CNEs as shown in Fig. 1.

2.2 Extendible Class Name Expressions

As we can see in Fig. 1, a clustering a set of keys using CNEs does not always have a good balance in the numbers of keys under CNEs. So we propose *the extendible class name expression* (ECNE) in order to overcome the drawback of CNEs. The basic idea is that we define variable length CNEs. That is, we divide a set that has a large number of keys, and merge sets that have a small number of keys into one.

CNEs use a fixed classification of letters, but ECNEs use a hierarchical classification of letters as shown in Fig. 2. The Level 0 class consists of one class that has all letters, that is Σ, and its class name is "0". The class at level 0 is divided into u classes and we call them the level 1 classes. Their class names are $1, \cdots, u$. And a level i class ($i = 1, \cdots, r$) is a fine classification of a level $i-1$ class, where r is the maximum level number. Fig. 2 shows an example of a hierarchical classification of letters in case that $r = 2$ and $u = 2$. In Fig. 2, the number on left shoulder of a class is its class name. At level i, u^{i-1} ECNEs have same class names. However, there occurs no confusion as we will show later.

A CNE using a hierarchical classification of letters is called the extendible class name expression. When the maximum level number in a hierarchical classification of letters is r, an ECNE E of key x with length n can be expressed as follows.

$$E(x) = e_{1,1}e_{1,2}\cdots e_{1,n}.e_{2,1}e_{2,2}\cdots e_{2,n}.\cdots.e_{r,1}e_{r,2}\cdots e_{r,n}$$

$e_{i,j}$ is a level i class name of the j-th letter in a key or is the class name 0 at level 0. A period between $e_{i,n}$ and $e_{i+1,1}$ is a separator between class names at level i and $i+1$. For example, 1210.0000 and 212.111.210 are ECNEs.

We define *the initial ECNE* such that $e_{i,j} = 0$ for $i = 1, \cdots, r$ and $j = 1, \cdots, n$, where 0 is the class name at level 0. Any ECNEs are made from the initial ECNE by a series of extension that will be described below. We define the length of an ECNE as the times of extension from the initial ECNE. The length of the initial ECNE is 0. The way of extending an ECNE E with length m $(0 \leq m < nr)$ is as follows. Since $m = n\lfloor m/n \rfloor + (m - n\lfloor m/n \rfloor)$, $e_{\lfloor m/n \rfloor+1, m-n\lfloor m/n \rfloor+1}$ and its right class names are "0". For example, consider ECNE 212.110.000 with $n = 3$ and $m = 5$. Since $5 = 3\lfloor 5/3 \rfloor + (5 - 3\lfloor 5/3 \rfloor)$, $e_{\lfloor m/n \rfloor+1, m-n\lfloor m/n \rfloor+1} = e_{2,3}$ and the right class names of $e_{2,3}$, that is $e_{3,1}$, $e_{3,2}$ and $e_{3,3}$, are "0". Hence, exchanging $e_{\lfloor m/n \rfloor+1, m-n\lfloor m/n \rfloor+1}$ for $1, \cdots, u$, that is, class names at level $(\lfloor m/n \rfloor + 1)$, makes u extended ECNEs. Thus the number of ECNEs increases by $u - 1$. An ECNE whose length is $m = nr$ can not be extended further.

Example: In case of $r = 2$ and $u = 2$, an extension of ECNE 12211.11000 with $n = 5$ and $m = 7$ makes two ECNEs 12211.11100 and 12211.11200.

$E^m(x)$ denotes an ECNE with length m for x. For example, using the hierarchical classification of letters in Fig. 2, the ECNE with length 7 for "apple" is $E^7(\text{apple}) = 12211.11000$. $E^{full}(x)$ denotes an ECNE with the maximum length, that is nr, for x, and $E^{full}(x) = E^{nr}(x)$. An ECNE with length m has $nr - m$ class names of "0" . We define *full extension* by exchanging $nr - m$ class names of "0" in $E^m(x)$ for the maximum class name u, and it is denoted by $F(E^m(x))$. For example, $F(E^7(\text{apple})) = F(12211.11000) = 12211.11222$.

Let E^{m_1} and E^{m_2} be ECNEs with length m_1 and m_2, respectively, and be $m_1 \leq m_2$. If $E^{m_1}(x)$ will become $E^{m_2}(x)$ by more than or equal to 0 times extension, then $E^{m_1}(x)$ is called *a prefix* of $E^{m_2}(x)$. For example, the prefixes of 121.110 are 000.000, 100.000, 120.000, 121.000, 121.100 and 121.110.

Let the maximum class name in a hierarchical classification be u. Let $E^{m,1}, E^{m,2}, \cdots, E^{m,u}$ be ECNEs with length m, and $E^{m-1,1}, E^{m-1,2}, \cdots, E^{m-1,u}$ be their prefixes with length $m - 1$, respectively. If $E^{m-1,1} = E^{m-1,2} = \cdots = E^{m-1,u}$, then the set of $(E^{m,1}, E^{m,2}, \cdots, E^{m,u})$ is said to be reducible and $E^{m-1,1}$ is called a reduced ECNE of $E^{m,1}, E^{m,2}, \cdots, E^{m,u}$. For example, in case of $r = 2$ and $u = 2$, (12211.11100, 12211.11200) is reducible and the reduced ECNE is 12211.11000.

2.3 A Method of Clustering a Set of Keys Using ECNEs

A set of keys can be clustered using ECNEs. When we cluster a set of keys, we establish the upper bound of the number of keys in a subset that have the same ECNE. If the number of keys in a subset becomes greater than the upper bound, the subset is split by extending the ECNE into u ECNEs. Similarly, we establish the lower bound of the number of keys in a subset that have the same ECNE. If the number of keys in a subset become less than the lower bound, the subset will be merged with other subsets by reducing the ECNEs. If the ECNE and other ECNEs can be reduced to an ECNE, their subsets can be merged. Thus, we can cluster a set of keys in good balance, since we establish the upper and lower bounds of the number of keys in a subset. Fig. 3 shows an example of clustering a set of keys using ECNEs made by a hierarchical classification of letters in Fig. 2.

2.4 Property of ECNEs

Let x be a key of length n. Assume that a series of extension of the initial ECNE of a key with length n produces M_n ECNEs, and let $T_n = \{E_{n,1}, E_{n,2}, \cdots, E_{n,M_n}\}$ be the set of them. An ECNE has following properties that is useful to retrieve a key that will be described in Section 3.

(1) There is only one prefix of $E^{full}(x)$ in T_n.

(2) There is no prefix of $E^{full}(x)$ in T_j $(j \geq 1, j \neq n)$.

From property (1) we have property (3).

(3) $E_{n,h}$ can not be a prefix of $E_{n,g}$, where $1 \leq h, g \leq M_n$.

If we omit periods in an ECNE, we can regard the ECNE as a number. The order of ECNEs is equal to that of numbers. Therefore, ECNEs have the following three properties.

(4) Let E_{n_1} and E_{n_2} be ECNEs of keys with length n_1 and n_2, respectively. If $n_1 < n_2$, we have the inequality $E_{n_1} < E_{n_2}$.

(5) Let E and E' be two ECNEs such that $E < E'$. Let $\hat{E}$ be any ECNE made by extending E one or more times. Then there is the inequality $E < \hat{E} < E'$.

(6) If E_n^i is a prefix of E_n^j, there is the inequality $E_n^i \leq E_n^j \leq F(E_n^i)$, and vice versa.

3 The File Structure and Operations

3.1 The File Structure

The file structure of the proposed file is a tree structure as shown in Fig. 4, and it consists of *an index part* and *a data part*. The index part is a B$^+$-tree that stores ECNEs and the data part stores clustered keys. An example of a file is shown in Fig. 5. Each node in the file is a page. A page is the smallest physical unit of storage allocation on the secondary storage device. It is the smallest unit of data that can be read from or written to the secondary storage. Page size, in bytes, is usually defined by the file system. In Fig. 5, the boxes represent pages and the numbers outside the boxes are page numbers.

Fig. 6 shows organizations of pages. A page in the index part is called *an index page* and a page in the data part is called *a data page*. A leaf index page contains ℓ ECNEs $E_1, E_2, \cdots, E_\ell$ and ℓ pointers $p_1, p_2, \cdots, p_\ell$. A non-leaf index page contains ℓ fully extended ECNEs $F_1, F_2, \cdots, F_\ell$ and $\ell + 1$ pointers $p_o, p_1, \cdots, p_\ell$. If E_ℓ denotes the maximum ECNE in $P(p_i)$ that is a leaf index page pointed by p_i, then $F_{i+1} = F(E_\ell)$. For example, in page 1 of Fig. 5, $F_1 = 122.222$ is the fully extended ECNE of $E_\ell = E_3 = 100.000$ in page 2. Tuples $(E_i,\ p_i)$ and $(F_i,\ p_i)$ are called *entries*. A data page, pointed by pointer p_i of entry (E_i, p_i) in a leaf index page, has the set of keys whose ECNEs are E_i. The keys in a data page are stored in lexicographical order.

The file has the following properties.

(1) A leaf page except the root in the index part has at least $\lfloor (k+1)/2 \rfloor$ and at most k ECNEs. The root and leaf page holds at least one and at most k ECNEs.

(2) A non-leaf and non-root index page has at least $\lfloor k^*/2 \rfloor$ and at most k^* fully extended ECNEs. The root and non-leaf index page has at least one and at most k^* fully extended ECNEs.

(3) A data page has at least one and at most b keys, where $b = \lfloor B/n \rfloor$. B is a page size and n is the length of keys.

3.2 The Retrieval Algorithm

If key x is in a file, we can find only one prefix of $E^{full}(x)$ in the index part due to property (1) of an ECNE. When we retrieve x, we must search the data page that has keys with the prefix of $E^{full}(x)$. Let $P(p)$ be a page to which p is pointing. Then $E_1, \cdots, E_\ell$ are the ECNEs in $P(p)$ and $p_1, \cdots, p_\ell$ are pointers in $P(p)$ if $P(p)$ is an leaf index page, and $F_1, \cdots, F_\ell$ are the fully extended ECNEs in $P(p)$ and $p_0, \cdots, p_\ell$ are pointers in $P(p)$ if $P(p)$ is an non-leaf index page. α and β, that is, an ECNE and a pointer, respectively, are not used in the retrieval algorithm, but will be used in the insertion and deletion algorithms and the algorithm for searching similar keys. The algorithm for retrieving key x is as follows.

(1) Make $E^{full}(x)$.

(2) $p \leftarrow$ the pointer to the root page.

(3) While $P(p)$ is a non-leaf index page, set p as follows.

$$p \leftarrow \begin{cases} p_0 \text{ , if } E^{full}(x) \leq F_1 \\ p_i \text{ , if } F_i < E^{full}(x) \leq F_{i+1} \\ p_\ell \text{ , if } F_\ell < E^{full}(x) \end{cases}$$

(4) In a leaf index page $P(p)$,

  (a) If $E^{full}(x) < E_1$, let $\alpha \leftarrow$ null and $\beta \leftarrow$ null. Quit operations, since there is no prefix of $E^{full}(x)$ in the file due to property (2) of an ECNE.

  (b) If $E_i \leq E^{full}(x) < E_{i+1}$, let $\alpha \leftarrow E_i$ and $\beta \leftarrow p_i$.

  (c) If $E_\ell \leq E^{full}(x)$, let $\alpha \leftarrow E_\ell$ and $\beta \leftarrow p_\ell$.

(5) If the length of a key with α differs from the length of x, there is no prefix of $E^{full}(x)$ in the file due to property (2) of an ECNE. Then let $\alpha \leftarrow$ null and $\beta \leftarrow$ null, and quit operations.

(6) If $\beta =$ null, there is no x in the file. Otherwise, since α is a prefix of $E^{full}(x)$, search x in data page $P(\beta)$.

Example: Consider the case of retrieving "king" in the file of Fig. 5. Since x = king, E^{f*ll}(king) = 1121.2211. In the root page 1 there exist ECNEs 1112.2222 and 1222.2222. Since 1112.2222 < E^{f*ll}(king) = 1121.2211 < 1222.2222, pointer p_2 is taken. p_2 points leaf index page 4. Since 1120.0000 < 1121.2211 < 1200.0000 in page 4, we have E_1 = 1120.0000 that is a prefix of E^{f*ll}(king). Pointer p_1 points page 12. We can find "king" in page 12.

3.3 The Search Algorithm for Similar Keys

We use *the one dimensional weighted Levenshtein distance* (WLD) [9] as a similarity measure between two keys. Let d_t be the predetermined threshold of distance between an input key and keys in a file. We will describe the case that $d_t = 1$, that is, "a key has at most one error" (condition(A)).

Let x' be a garbled key of x with length n under condition(A). Let the maximum level number r in a hierarchical classification be 2, and $E^{f*ll}(x') = e_{1,1}e_{1,2}\cdots e_{1,n}.e_{2,1}e_{2,2}\cdots e_{2,n}$. Let $e'_{i,j}$ be a class name at level i such that $e'_{i,j} \neq e_{i,j}$ and $*_i$ be any class name at level i.

The following is the method of searching the most similar keys to an input key x' with length n.

(1) Let Z be the set of keys that have the minimum distance from x' and $d_{\min}$ be the minimum distance. $Z \leftarrow \phi$ and $d_{\min} \leftarrow d_t$ are given in advance, where ϕ is the empty set.

(2) Make $E^{f*ll}(x')$.

(3) Create the set S of all ECNEs that satisfy condition(A) from $E^{f*ll}(x')$, and sort the set S.

(4) While $S \neq \phi$, repeat (5) ~ (7).

(5) Remove an element s from S.

(6) Search s in the index part by step (2) ~ (5) in the retrieval operation.

   (a) If $\alpha = s$, perform (7) for $P(\beta)$.

   (b) If α is a prefix of s, remove an ECNE that is a prefix of α from S, and perform (7) for $P(\beta)$.

   (c) If α = null, go to (4).

(7) Let h be the number of keys in $P(\beta)$ and x_j $(1 \leq j \leq h)$ be a key in $P(\beta)$. For $j = 1 \sim h$, calculate the WLD from x_j to x'
If $D(x_j, x') < d_{\min}$, let $Z \leftarrow \{x_j\}$ and $d_{\min} = D(x_j, x')$.
If $D(x_j, x') = d_{\min}$, let $Z \leftarrow Z + \{x_j\}$.

After the above operations, Z is the set of the most similar keys to x'.

4 Experimental Results

The file was constructed using 230,188 English words. The experiment was carried out under the following conditions. The page size B was 1,024 bytes. The maximum number of ECNEs in a leaf index page and a non-leaf index page were $k = k^* = 170$. The hierarchical classification of letters of Fig. 2 was used.

The storage utilization U of our file is shown in Fig. 7. As the number N of keys becomes larger, U seems to converge to about 66%. According to the performance analysis [16], [15] of a B-tree, the storage utilization converges to ln 2, when N and b are large. The storage utilization of the proposed file was slightly lower than that of a B-tree, that is, ln 2, since the assumption (3) is not satisfied.

Let x' be a garbled spelling of x and Z be the set of keys with the minimum distance to x'. We estimated the following three rates.

(1) Correction rate : the probability to be $|Z| = 1$ and $Z = \{x\}$.

(2) Miscorrection rate : the probability to be $|Z| = 1$ and $Z \neq \{x\}$.

(3) Rejection rate : the probability to be $|Z| > 1$.

A key in the file was garbled by making a substitution, an insertion or a deletion, and this garbled spelling was input to the system. If a garbled key is equal to another key in the file, the key is not used for an input key. The correction rates of search for the most similar keys are shown in Fig. 8. In the experiment under condition (A), a miscorrection did not occur. As shown in Fig. 8, the longer an input key was, the higher the correction rate was. The correction rate was higher when an input key had an insertion error. On the contrary, it was lower when an input key had a deletion error.

5 Conclusion

In this paper, we proposed a file that can retrieve similar keys to an input key, and insert and delete keys without lowering its storage utilization. The file structure is based on B$^+$-tree. The experiment using 230,188 English words with length 1 $\sim$ 16 indicates that this file can retrieve, insert and delete keys with less disk access than a an original B-tree [1]. The storage utilization of the file was about 66%. We analyzed the performance of searching similar keys. The theoretical values of the numbers of page reads and calculating WLDs were very close to their experimental values. The performance analysis was based on the assumption that a key has at most one error. One of the remained problems is to analyze the performance in more general cases. The details are described in [19] although many topics are omitted.

References

[1] R. Bayer and E. McCreight, "Organization and maintenance of large ordered indexes," *Acta Inform.*, vol.1, pp.173-189, 1972.

[2] R. Bayer and K. Unterauer, "Prefix B-Trees," *ACM Trans. Database Syst.*, vol.2, no.1, pp11-26, 1977.

[3] D. Comer, "The ubiquitous B-tree," *Comput. Surveys*, vol.11, no.2, pp121-137, 1979.

[4] B. Eisenbarth, N. Ziviani, G. Gonnet, K. Mehlhorn, and D. Wood, "The theory of fringe analysis and its application to 2-3 trees and B-trees," *Information and Control*, vol.55, pp.125-174, 1982.

[5] K. Kant and A. Ravichandran, "Synthesizing robust data structures – an introduction," *IEEE Trans. Comput.*, vol.39, no.2, pp.161-173, 1990.

[6] D. E. Knuth, *The Art of Computer Programming, vol.3 : Sorting and Searching*, Reading, MA : Addison-Wesley, 1973.

[7] K. Küspert, "Storage utilization in B*-trees with a generalized overflow technique," *Acta Inform.*, vol.19, pp.35-55, 1983.

[8] D. Lomet, "Partial expansions for file organizations with an index," *ACM Trans. Database Syst.*, vol.12, no.1, pp.65-84, 1987.

[9] T. Okuda, E. Tanaka and T. Kasai, "A method for the correction of garbled words based on the Levenshtein metric," *IEEE Trans. Comput.*, vol.C-25, no.2, pp.172-178, 1976.

[10] E. Tanaka and T. Toyama and S. Kawai, "High speed error correction of phoneme sequences," *Pattern Recognition*, vol.19, no.5, pp.407-412, 1986.

[11] E. Tanaka and Y. Kojima, "A high speed string correction method using hierarchical file," *IEEE Trans. Pattern Anal. & Mach. Intell.*, vol.9, no.6, pp.806-815, 1987.

[12] D. J. Taylor and D. E. Morgan and J. P. Black, "Redundancy in data structures: improving software fault tolerance," *IEEE Trans. Software Eng.*, vol.6, no.6, pp.585-594, 1980.

[13] D. J. Taylor and D. E. Morgan and J. P. Black, "Redundancy in data structures: some theoretical results," *IEEE Trans. Software Eng.*, vol.6, no.6, pp.595-602, 1980.

[14] H. Wedekind, "On the selection of access paths in a data base system," *in Proc. IFIP Working Conf. Data Base Management*, pp385-397, 1974.

[15] W. E. Wright, "Some average performance measures for the B-tree," *Acta Inform.*, vol.21, pp.541-557, 1985.

[16] A. Yao, "On random 2-3 trees," *Acta Inform.*, vol.9, pp.159-170, 1978.

[17] R. Baeza-Yates and P. Larson, "Performance of B^+-trees with partial expansions," *IEEE Trans. Knowledge and Data Eng.*, vol.1, no.2, pp.248-257, 1989.

[18] R. Baeza-Yates, "Expected behaviour of B^+-trees under random insertions," *Acta Inform.*, vol.26, pp.439-471, 1989.

[19] M. Hirade, "A study on a fault tolerant file," *Master theses*, Department of Electrical and Electronics Engineering, Kobe University, 1994.

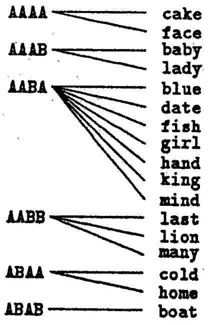

Fig. 1. An example of clustering a set of keys using CNEs.

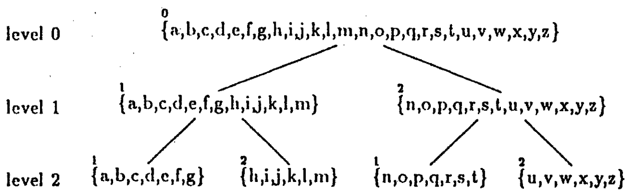
level 0
0
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}
level 1
1
{a,b,c,d,e,f,g,h,i,j,k,l,m}
2
{n,o,p,q,r,s,t,u,v,w,x,y,z}
level 2
1
{a,b,c,d,e,f,g}
2
{h,i,j,k,l,m}
1
{n,o,p,q,r,s,t}
2
{u,v,w,x,y,z}

Fig. 2. A hierarchical classification of letters.

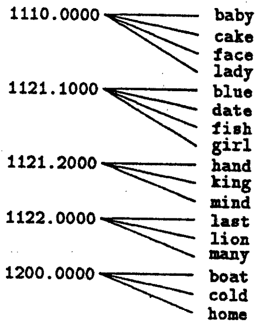

Fig. 3. An example of clustering a set of keys using ECNEs.

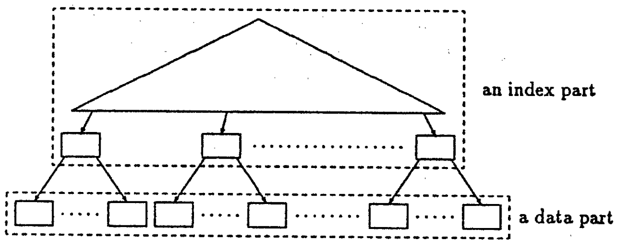

Fig. 4. A file structure.

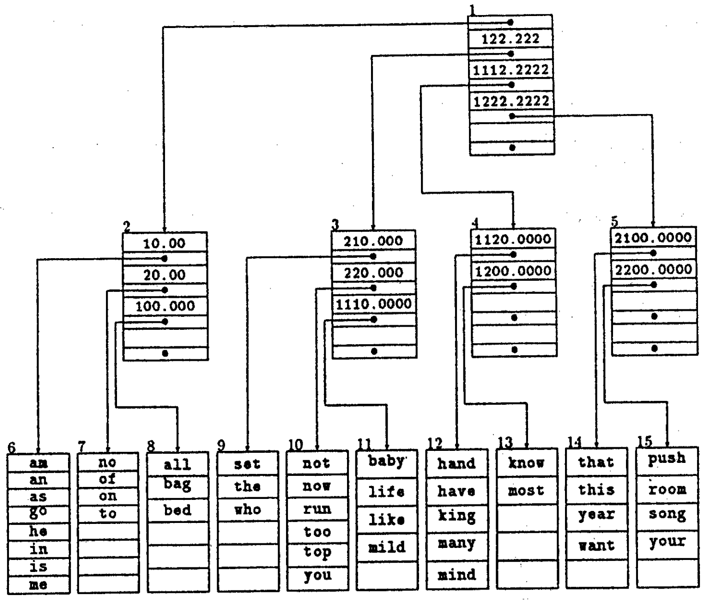

Fig. 5. An example of a file.

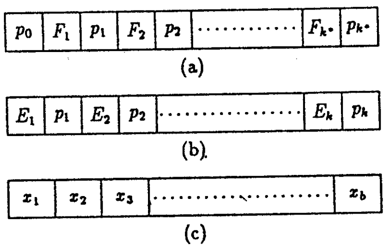
p_0 F_1 p_1 F_2 p_2 .... F_{k^*} p_{k^*}
(a)
E_1 p_1 E_2 p_2 .... E_k p_k
(b)
x_1 x_2 x_3 .... x_b
(c)

Fig. 6. Organizations of pages. (a) A non-leaf page in an index page, (b) a leaf page in an index page, and (c) a page in a data page.

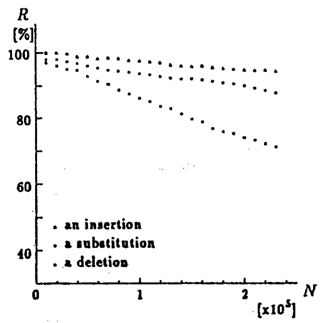

Fig. 8. Correction rates R of searching the most similar keys for length 8 inputs.

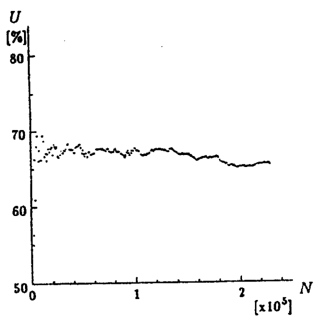

Fig. 7. Storage utilization U versus the number N of keys.