In Proc. IFIP'74, pages 647-651, Amsterdam, 1974. North-Holland.

THEORY OF DIRECT-ACCESS STORAGE FUNCTIONS

H.-D. EHRICH

Universität Dortmund, Abteilung Informatik
Dortmund, West Germany

Direct-access storage functions, as usually applied to the implementation of arrays, may be useful in many other cases as well. The theory developed here provides a general method for the effective construction of a one-to-one mapping from a given set of keywords onto an interval of contiguous relative addresses. This mapping is based on a specific ordering of the keywords. A general index formula is derived which represents this mapping. The index formula is interpreted by means of the finite-state acceptor which accepts the set of keywords. Several examples show how this theory can be applied to the derivation of direct-access storage formulas.

1. INTRODUCTION

In practical data processing we often have to organize a set of data in a table, each table entry being characterized and accessed by a so-called keyword. The programmer has to design an access mechanism

keyword $\longrightarrow$ address of table entry

To do this, in most cases some sort of searching is required. In some special situations, however, we can find direct-access storage functions, i.e. procedures that calculate the exact address of the table entry, given only the letters of the corresponding keyword (coded by natural numbers). As it is well known, this is the case with arrays where the index n-tuples play the role of the keywords. This technique is well applicable to some sorts of regularly structured sparse arrays too, as shown by Knuth [3] .

In this contribution a general theoretical framework for the construction of direct-access storage functions is developed. On this basis, structural regularities in the given set of keywords can be detected and, whenever possible, exploited for the effective construction of direct-access storage formulas.

*When preparing this paper the author was with the Institut für Informatik und Praktische Mathematik, Universität Kiel, Kiel, West Germany.

2. THE GENERAL INDEX FORMULA

Let L be a finite set of keywords, i.e. a finite language over the alphabet $X=\{0,1,\ldots,d-1\}$. We thus presume the letters of the keywords to be natural numbers since we want to execute arithmetic calculations on them. We further presuppose that the table entries being in one-to-one correspondence with the keywords are to be stored in consecutive memory cells with the relative addresses $0,1,\ldots,p-1$ where p is the cardinality of L. By mapping the keywords onto these addresses an ordering on L is introduced. The basic idea of the present approach is to fix this ordering in such a way that the address mapping may be expressed by a closed formula.

We arrange the words of L by non-decreasing length. Within a subset of words of equal length the arrangement is made lexicographically increasing, based on the natural less-than-relation on X. The ordering on L thus specified will be denoted by $\prec$.

Let $L=\{x_0,x_1,\ldots,x_{p-1}\}$, $x_i \in X^*$ for $0 \leq i \leq p-1$, such that $x_0 \prec x_1 \prec \ldots \prec x_{p-1}$. X^* is the set of words over the alphabet X, including the empty word ε of length 0. The mapping

$\nu : L \longrightarrow \{0,1,\ldots,p-1\}$

is called index mapping iff for all $i \in \{0,1,\ldots,p-1\}$ we have $\nu(x_i)=i$.

By definition of the ordering $\prec$ on L the index of a word $x \in L$ of length n may be expressed as a sum of two values,

$$\nu(x) = \alpha_n + \nu_n(x) \quad ,$$

where α_n is the number of words in L of length less than n, and ν_n is the index mapping of the subset $L_n \subseteq L$ of all words of L with equal length n.

In practical cases the number of different word lengths is small compared with the number p of all words. Furthermore, the number q_k of words of length k in L, $k=0,1,\dots$, may be assumed to be known in advance. The value of α_n , then, is given by

$$\alpha_n = \sum_{k=0}^{n-1} q_k \quad .$$

The determination of $\nu_n(x)$ is a more difficult problem. In what follows, let $L=L_n$, i.e., all words of L are of equal length n. Then we have $\nu = \nu_n$ too. For a word $x=x_1x_2\dots x_k \in X^*$ let $L(x_1,\dots,x_k)$ be the derivative of L with respect to x (cf. [1]),

$$L(x_1,\dots,x_k) := \{ y \in X^* \mid xy \in L \} \quad .$$

Let $p(x_1,\dots,x_k)$ be the cardinality of $L(x_1,\dots,x_k)$. By means of the cardinalities of the derivatives of L, the index mapping ν can be expressed by a general index formula as the following theorem shows.

Theorem 1 : Let L be as defined above. The index of a word $x=x_1x_2\dots x_n \in L$ is given by

$$\nu(x) = \sum_{k=1}^{n} \sum_{j=0}^{x_k-1} p(x_1,x_2,\dots,x_{k-1}, j) \quad (1)$$

Proof: Let $y^k=x_1x_2\dots x_k0\dots0$, $1 \le k \le n$, be the words of length n consisting of the k-letter prefixes of x filled up with k-n letters "0". Let v_k be the number of words of L preceding y^k in the lexicographic ordering $\prec$. For v_k we have the recurrence relation

$$v_k = v_{k-1} + \sum_{j=0}^{x_k-1} p(x_1,x_2,\dots,x_{k-1}, j)$$

since the set of words preceding y^k is the disjoint union of the set of words preceding y^{k-1} and the sets of words with the prefixes $x_1x_2\dots x_{k-1}j$ where $j=0,\dots,x_k-1$. With this relation the index formula of the theorem is easily derived remembering that $\nu(x)=v_n$.

For a word $\bar{x} \in X^*$, let

$$F(\bar{x}) := \{ y \in X \mid \exists\, \bar{z} \in X^* : \bar{x}y\bar{z} \in L \}$$

be the set of letters $y \in X$ which can follow immediately the prefix $\bar{x}$ in a word of L. Then we can write the index formula as follows:

$$\nu(x) = \sum_{k=1}^{n} \sum_{\substack{j \in F(x^{k-1}) \\ j < x_k}} p(x_1,\dots,x_{k-1},j) \quad (2)$$

In this formula x^{k-1} means the prefix of x of length k-1.

For the cardinalities of the derivatives of L evidently the following recurrence relation holds for $k=2,\dots,n$:

$$p(x_1,\dots,x_{k-1}) = \sum_{j \in F(x^{k-1})} p(x_1,\dots,x_{k-1},j) \quad (3)$$

This equation is valid for k=1 too, if we identify $p()=p=|L|$. With this equation further transformations of the index formula can be made.

$$\begin{aligned} \nu(x) &= \sum_{k=1}^{n} \Big(p(x_1,\dots,x_{k-1}) \qquad\qquad (4) \\ &\qquad - \sum_{\substack{j \in F(x^{k-1}) \\ j \ge x_k}} p(x_1,\dots,x_{k-1},j) \Big) \\ &= \sum_{k=1}^{n} \Big(p(x_1,\dots,x_{k-1}) - p(x_1,\dots,x_k) \\ &\qquad - \sum_{\substack{j \in F(x^{k-1}) \\ j > x_k}} p(x_1,\dots,x_{k-1},j) \Big) \\ &= p - p(x_1,\dots,x_n) \\ &\quad - \sum_{k=1}^{n} \sum_{\substack{j \in F(x^{k-1}) \\ j > x_k}} p(x_1,\dots,x_{k-1},j) \end{aligned}$$

Since x is the only word in L with the n-letter prefix $x=x_1\dots x_n$, we have $p(x_1,\dots,x_n)=1$, and another important index formula is established:

$$\nu(x) = |L| - 1 - \sum_{k=1}^{n} \sum_{j=x_k+1}^{d-1} p(x_1,\dots,x_{k-1},j) \quad (5)$$

The right-hand sum counts the number of words succeeding the word x in L with respect to the ordering $\prec$.

3. THE FINITE-STATE ACCEPTOR

A well known construction for the reduced finite-state acceptor $A(L)=(S,X,\delta,s_0,F)$

which accepts the finite (thus regular) language $L \subseteq X^n$ is as follows:

States: $S=\{L(x_1,..,x_k) \mid x=x_1...x_k \in X^*\}$
Transition function: $\delta(L(x_1,..x_k),x_{k+1})=L(x_1,..,x_{k+1})$
Initial state: $s_o = L$
Final states: $F=\{s\in S \mid \varepsilon \in s\}$ (ε = empty word)

The general index formula (1) may well be interpreted by means of the finite-state acceptor. To do this we add the output alphabet $Y=\{0,1,...,p-1\}$ and the output function

$$\lambda(s,x) := \sum_{j=o}^{x-1} |\delta(s,j)| \qquad (6)$$

for all $s\in S$ and $x\in X$ to the acceptor $A(L)$, thus getting the finite-state machine $A^*(L)=(S,X,Y,\delta,\lambda,s_o,F)$ with initial state s_o and the set of final states F.

Let $x=x_1x_2...x_k\in L$. The output string which results when applying x to the state s of $A^*(L)$ will be denoted by $\bar\lambda(s,x)=y_1y_2...y_k$. With these conventions theorem 1 may be reformulated in terms of the augmented finite-state acceptor $A^*(L)$.

Theorem 2 : Let $x=x_1x_2...x_n\in L$ and $\bar\lambda(s_o,x) =y_1y_2...y_n$. Then equation

$$\nu(x) = \sum_{k=1}^{n} y_k \qquad (7)$$

holds.

Proof: By definition of the δ-function we have $p(x_1,...,x_{k-1},j)=|\delta(s^{k-1},j)|$ where $s^{k-1}=L(x_1,...,x_{k-1})$. Index formula (1) may therefore be written in the following way:

$$\nu(x) = \sum_{k=1}^{n}\sum_{j=o}^{x_k-1} |\delta(s^{k-1},j)| ,$$

and, by definition of the λ-function,

$$\nu(x) = \sum_{k=1}^{n} \lambda(s^{k-1},x_k) .$$

However, the state s^{k-1} may be expressed by $s^{k-1}=\delta(s_o,x_1...x_{k-1})$, and obviously $\lambda(s^{k-1},x_k)=y_k$ holds.

4. SOME APPLICATIONS

The central point for the construction of direct-access storage functions for a given set of keywords L is the output function (6) of the augmented acceptor $A^*(L)$. Whenever we succeed in representing these terms in closed form, we can expect interesting results. This is the case with the following examples, where structural regularities of L in a natural way lead to index formulas in closed form, thus getting direct-access storage functions.

Example 1: Arrays

The set of keywords (index n-tuples) of an n-dimensional array has the form of a complex product $L=X_1X_2...X_n$ where the X_i are intervals, $1\le i\le n$:

$X_i = [l_i,u_i] \subseteq X=[0,d-1]$.

The derivatives and their cardinalities are calculated easily:

$$L(x_1,...,x_k) = \begin{cases} X_{k+1}...X_n , & \text{if } x_i\in X_i \text{ for all } i=1,..,k \\ \emptyset & \text{otherwise}\end{cases}$$

$$p(x_1,...,x_k) = \prod_{i=k+1}^{n} |X_i| = \prod_{i=k+1}^{n} (u_i-l_i+1),$$

if $x_i\in X_i$ for all $i=1,...,k$, and zero otherwise. This implies

$$\begin{aligned} y_k &= \lambda(\, L(x_1,...,x_{k-1}) \, , x_k \,) \\ &= \sum_{j=o}^{x_k-1} p(x_1,...x_{k-1},j) \\ &= \sum_{j=l_k}^{x_k-1} \prod_{i=k+1}^{n} (u_i-l_i+1) \\ &= (x_k-l_k)c_k \quad , \end{aligned}$$

where $c_k = \prod_{i=k+1}^{n} (u_i-l_i+1)$.

By applying theorem 2 we now get the well known storage access formula for n-dimensional arrays:

$$\nu(x_1...x_n) = \sum_{k=1}^{n} (x_k-l_k)c_k \quad .$$

Example 2: Δ_s-arrays

Let $X=\{0,1,...,d-1\}$, $n\in\mathbb{N}-\{0\}$ and $s\in\mathbb{N}$, where $\mathbb{N}=\{0,1,2,...\}$. The definition

$$L_s := \{x_1...x_n \mid x_i\le x_{i-1}-s \; , \; 1\le i\le n\} ,$$

where $x_o:=d-1+s$, specifies a class of sparse arrays which will be called Δ_s-arrays. For n=2,s=0 we have a lower triangular matrix with main diagonal, as shown below.

$$\begin{pmatrix} 0,0 & & & \\ 1,0 & 1,1 & & \\ 2,0 & 2,1 & 2,2 & \\ & & & \cdot \\ & & & \quad\cdot \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

For n=2,s=1 we get a lower triangular matrix without main diagonal, etc.

The derivatives depend only on the last letter x_k, whenever the condition $x_i \le x_{i-1}-s$ is fulfilled for $1 \le i \le n$. Hence we simplify by writing

$$\varphi_k(x_k) := p(x_1,\ldots,x_k) \quad .$$

From eq. (3) we get a recurrence relation for φ :

$$\varphi_k(x_k) = \sum_{j=0}^{x_k-s} \varphi_{k+1}(j) \qquad (8)$$

For k=n we have $\varphi_n(x_n)=1$, and by using induction on k, we verify the solution

$$\varphi_k(x_k) = \binom{x_k-(n-k)(s-1)}{n-k}$$

The output function (6) of the machine $A^*(L)$ is given by

$$\begin{aligned} y_k &= \lambda(\ L(x_1,\ldots,x_k)\ ,\ x_{k+1}\ ) \\ &= \sum_{j=0}^{x_k-1} \varphi_k(j) \end{aligned}$$

From eq. (8) we obtain

$$y_k = \varphi_{k-1}(x_k+s-1) \quad .$$

Application of theorem 2 yields

$$\begin{aligned} \nu(x_1\ldots x_n) &= \sum_{k=1}^{n} \varphi_{k-1}(x_k+s-1) \\ &= \sum_{k=1}^{n} \binom{x_k+s-1-(n-k+1)(s-1)}{n-k+1} \\ &= \sum_{k=1}^{n} \binom{x_k+(n-k)(s-1)}{n-k+1} \end{aligned}$$

The special case of this result when s=0 can be found in [3] .

Example 3: ∇_t-arrays

In analogy to Δ_s-arrays, upper triangular arrays to be defined here can be treated in nearly the same way. The main difference is that applying index formula (5) is much more advantageous than applying eq.(1) or theorem 2 . We therefore carry out the construction coarsely.

∇_t-arrays are specified by the set of keywords

$$L_t := \{\ x_1\ldots x_n \mid x_i \ge x_{i-1}+t\ ,\ 1 \le i \le n\ \}\ ,$$

where $x_0:=-t$ and $t \in \mathbb{N}$.

As in the previous case, the derivatives depend only on the last letter. The cardinalities will therefore be denoted by $\psi_k(x_k)$. Eq.(3) then becomes

$$\psi_k(x_k) = \sum_{j=x_k+t}^{d-1} \psi_{k+1}(j) \qquad (9)$$

The solution is

$$\psi_k(x_k) = \binom{d-1-x_k-(n-k)(t-1)}{n-k}$$

The easiest way to proceed now is to transform the right-hand sum of index formula (5) by application of eq.(9):

$$\sum_{j=x_k+1}^{d-1} \psi_k(j) = \psi_{k-1}(x_k-t+1)$$

Furthermore, we have

$$|L| = \psi_0(x_0) = \binom{d-(n-1)(t-1)}{n} \quad ,$$

yielding the result

$$\begin{aligned} \nu(x_1\ldots x_n) = &\binom{d-(n-1)(t-1)}{n} - 1 \\ &- \sum_{k=1}^{n} \binom{d-1-x_k-(n-k)(t-1)}{n-k+1} \end{aligned}$$

5. CONCLUDING REMARKS

Direct-access storage functions are a very attractive means of storage access, both with regard to storage space and access speed. The keywords need not be stored with the associated data elements, since the mapping to the addresses is one-to-one, and the proper address can be calculated directly from the keyword without any searching through storage cells belonging to other keywords. On the other hand, this technique has, of course, some serious drawbacks. It should be clear that its application is restricted to static or nearly static data sets, with very low rates of insertion and deletion (cf. [2]) .

However, where this preliminary condition is fulfilled, the theory developed here may prove useful to extend the application of the direct-access method. The only storage formula used widely is that of example 1 for arrays. From the general formulas of examples 2 and 3, only special cases were known before. Even if the given set of keywords has no apparent regularities as in the cases discussed, it is sometimes possible to construct a reasonable direct-access storage procedure, especially if there is only a small number of keywords.

REFERENCES

[1] J.A. Brzozowski, Derivatives of regular expressions, Journal of the ACM, vol.11, October 1964, 481-494.

[2] D. Ferrari, The use of analytical access functions in data structure implementation, Proc. 4th Hawaii Intern. Conf. on Sys. Sc., Shu Lin (ed), Western Periodicals Comp., North Hollywood, 1971, 707-709.

[3] D.E. Knuth, The art of computer programming, vol. 1, Addison Wesley, Reading, Mass.,1968.