Use of Doubly Chained Tree Structures in File Organisation for Optimal Searching

F. SURAWEERA
Mathematics Department, Kuwait University, P.O. Box 5969, Kuwait

The problem of minimising the search length (time) associated with a variable length, doubly chained tree structure with weighted terminal nodes and its implications on the organisation of a file are investigated. The main result is an algorithm which constructs an optimal doubly chained tree as a static data structure. Also considered are a few ramifications, particularly when the search steps have unequal costs. The supporting activity of file maintenance is also discussed. Two methods of updating a data file organised as a doubly chained tree structure are presented, which allow the realisation of a dynamic data structure.

Received April 1984

1. INTRODUCTION AND LITERATURE

Storage and retrieval problems occur not only in the area of conventional libraries. Many commercial and government file problems are exactly as complex as the conventional library problems and often involve severe time and cost restrictions. Presently, the world around us is full of people and organisations who have become concerned with generating and handling information. There is a much greater dependence on recorded information than a few years ago. Furthermore, we demand easier access to previously recorded information and try to reduce or maintain the amount of total effort involved.

In this paper we discuss the problem of minimising the average search time associated with a variable length, doubly chained tree (d.c.t.) structure when the terminal nodes are weighted. We also discuss the implications of the weights on the organisation of the file, as for example in the case of a subject catalogue. First a static d.c.t. structure is considered, then we study dynamic aspects. We also consider, albeit briefly, a few extensions, particularly when the search mechanism consists of unequal costs. This is a reasonable and justifiable assumption when one recognises that getting the information associated with a node involves two different actions: (*a*) finding the exact node, and (*b*) reading the record associated with that particular node.

Many of the information retrieval applications centre on large data files which must both be searched and updated frequently. It quickly becomes apparent that there are many ways of organising and updating these files, depending upon the particular structure and the application in mind.

The d.c.t. structure for processing large data files was first introduced by De la Briandias.[1] In the following year Fredkin[2] proposed a similar approach for handling multi-attribute keys by using a tree structure which he called a *trie*. Later, in a classic paper Sussenguth[3] analysed various properties of the d.c.t. structure; his solution required that all terminal nodes lie on the same level of the tree, and he assumed equal probabilities of file items being requested. He also showed that an N-record file could be searched and altered in $s * \log_s N$ time, where s is a parameter of the tree.

Patt[4] in a later paper relaxed a restriction which appeared in Sussenguth's paper, namely, that all the terminal nodes should lie on the same level of the tree. He also derived an expression for the minimum average search time as a function of the number of terminal nodes.

Stanfel[5,6] has investigated some aspects of updating of data files in the form of doubly chained trees. Subsequently, combining the concepts of d.c.t. and list, Stanfel[7] was able to produce files with expected search time smaller than that of the pure tree. It is worthwhile to recall that, in a d.c.t., it may take twice as long to go down a sequence of right branches than that of left branches.

A variety of tree-structured schemes have been proposed for organising, searching, and updating data files.[8–15]. For a general reference on indexing the reader is invited to see Knuth.[13] It seems that practically nothing on d.c.t.s has appeared in the literature since 1977. The material reported in this paper is inspired by the earlier works of Sussenguth, Stanfel and Patt.

2. DEFINITIONS AND NOTATIONS

A *rooted tree* is a special type of directed graph which has at most one branch entering each node. Since a tree contains no cycle, the length of a path in a tree is bounded, and there exist maximal paths which are not included in any longer paths. The initial node and final node of a maximal path are called the *root* and the *terminal node* respectively, A terminal node is also called a *leaf*. The root is said to lie on the first level of the tree, and a node which lies at the end of a path with length $k-1$ from the root is on the kth level. The *filial set* of a node x is the set of nodes which lie at the end of a path of length one from node x, and x is called the *parent* node of that set. Every individual member of a filial set is referred to as a *brother*. The set of all nodes reachable from x is said to be governed by x and comprises the nodes of the subtree rooted at x. We do not exclude the possibility of having a collection of rooted trees. In case we have such a collection of trees we require that the roots of such trees are on the same level. Fig. 1 illustrates some of the notions defined so far.

A *file entry* or an *item* is defined as the basic unit which is processed as a single unit. An item consists of two parts: the key field and the data field. The *key* which is associated with an item uniquely identifies the item. A set of items is a *file*. Items of data in the file are stored as terminal nodes in the doubly chained tree structure and

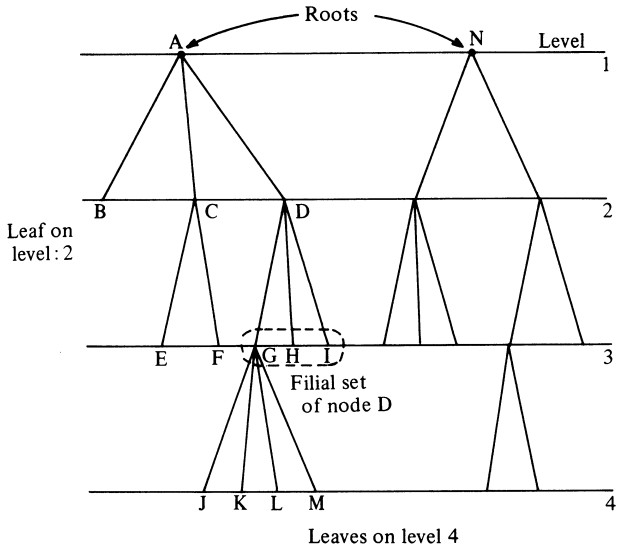

Figure 1. A tree illustrating some terminology.

may be physically implemented as words in the processor storage, tape blocks or disk regions.

The structure of a doubly chained tree is well documented.[3, 6, 7, 14, 15] For the sake of completeness we give below a brief review. The unique key corresponding to a leaf of the tree is partitioned into several disjoint elements. Now each of these elements is made to correspond to a tree node as follows: the first element corresponds to a node on the first level, the second element to a node on the second level, and so on. The *value* of a node is defined to be the element corresponding to that particular node.

More formally, a doubly chained tree is a data structure for representing a tree in a digital computer. In case of processor storage each node is represented by one or more computer words consisting of three fields. The first field contains part of the key which is queried during a search. If the component query key-value and the value at a node does not match, the computer follows the second field, which points to the address of a brother where the same component query key-value will apply. If there is a match, the next node to be queried is obtained by following the third field which is the address of the first son of the given node. Fig. 2 illustrates the organisation of a file, using a basic d.c.t. structure.

Let $k_1, k_2, \ldots, k_N$ be N attributes belonging to a file item. In general, the domain of an attribute will be the set of integers; but we do not rule out other types of values, for example character strings or real numbers. The N-tuple $(k_1, k_2, \ldots, k_N)$ is defined as a *query key value* or simply a key value. In this notation k_i represents the ith component of the query key value.

2.1 Search procedure for a doubly chained tree structure

The search procedure for such a d.c.t. structure is extremely simple. The first component k_1 of the query key value is tested from left to right in the stored tree representation, against the first level nodes (i.e. the first filial set) of the tree; when a match is found, the filial set containing the sons of the matching node on the second level are compared with k_2, and so on. The search continues in this fashion until all the key components are matched, whereupon a terminal node is reached. A simplified flowchart of the matching process is shown in Fig. 3.

3. OPTIMAL TREE STRUCTURES

In this section we consider the determination of optimal tree structures, when there is a weight associated with

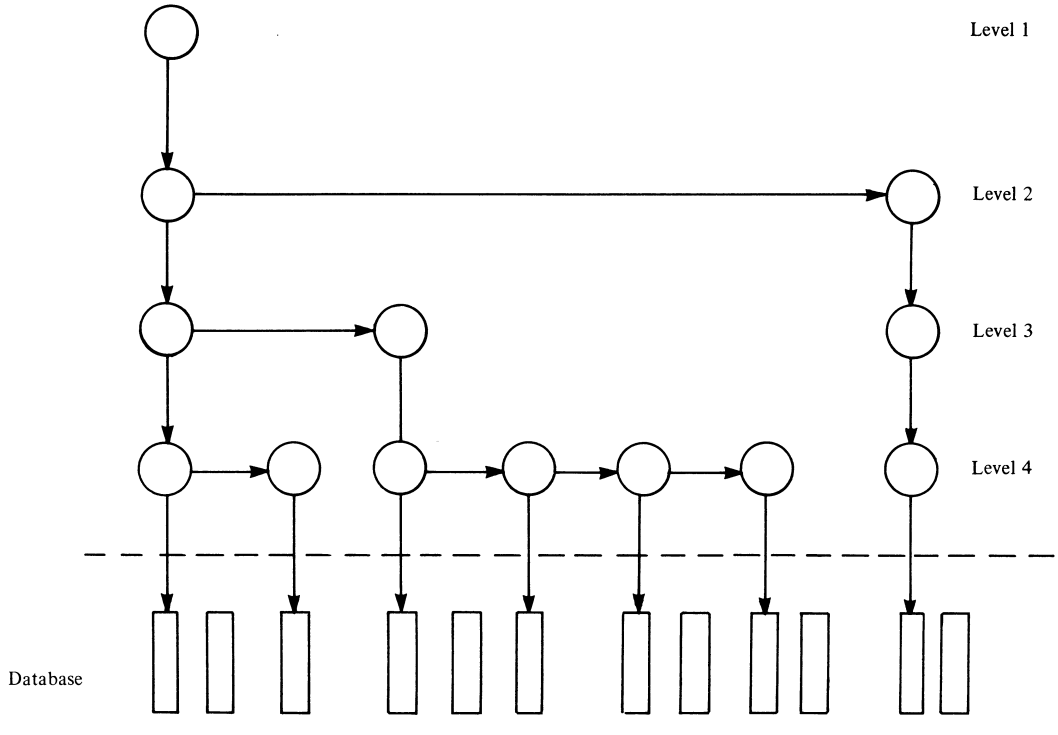

Figure 2. Basic doubly chained tree file organisation (Reproduced by permission of Alfonso F. Cardenas from *The Computer Journal*, Vol. 20).

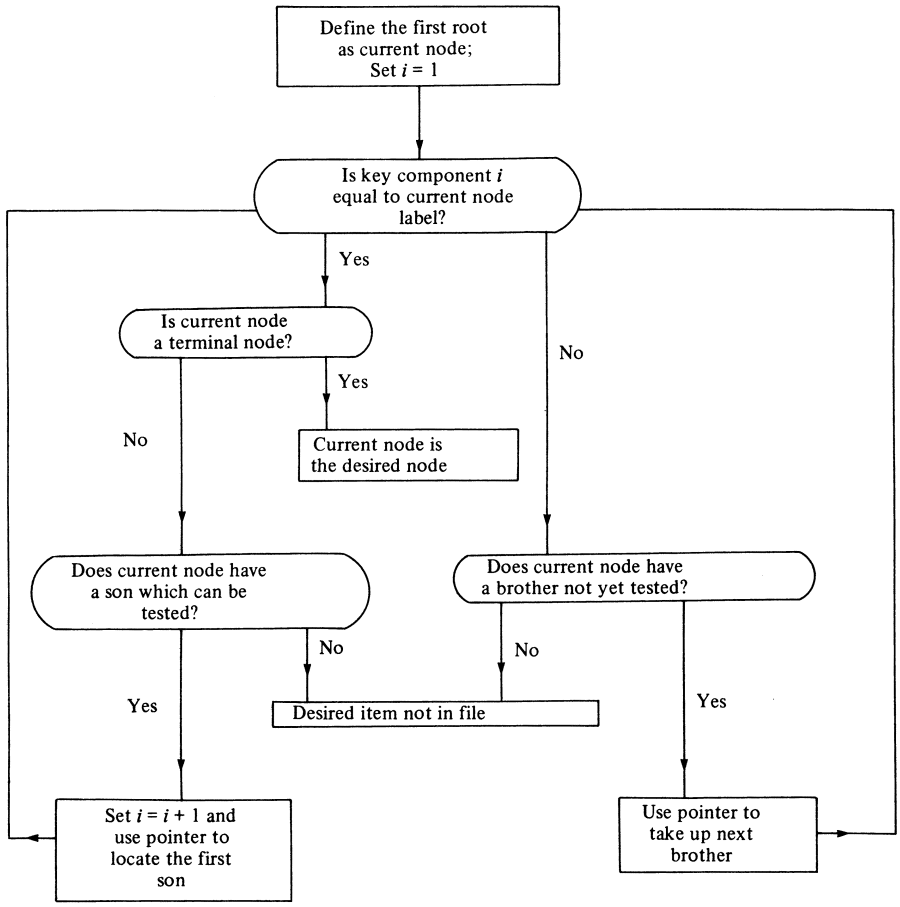

Figure 3. Flowchart for matching process (Reproduced by permission of Gerard Salton from *Automatic Information Organization and Retrieval*, McGraw-Hill (1968)).

each terminal node. These weights could represent or measure such features as the relative importance of the items, the frequency of use and so on. A simple example would be the design of a hierarchical index for the purpose of storing and retrieving records. In particular, we present an algorithm to construct optimal tree structures; the term optimal tree structures means those with minimum expected search time.

Consider a file system organised as a d.c.t. structure in which the filial set of each node is completely know in advance. A typical example of the above type of situation occurs in a special dictionary tree. Clearly the set of all possible letters of words beginning with the letter 'V' is (A, E, I, O, U and Y). In our terminology, the set (A, E, I, O, U, Y) is the filial set of V and we say that the filial set is completely specified *a priori*. It should be noted here that although the filial set of every node is fixed, the order in which the brothers of the filial set are searched is not known in advance.

Consider any terminal node v in the d.c.t. structure T. Recall that in T we may have more than one rooted tree. Let $\pi[\text{root}_1, v]$ denote the unique path from the left-most root, root_1, to the terminal node v. Then the time required to reach v is clearly proportional to the number of nodes queried along the path $\pi[\text{root}_1, v]$. In the present paper this time is referred to as the *search cost* for node v. The search cost $C(v)$ is given by,

$$C(v) = \sum_{r=1}^{h(v)} P_r(v)$$

where $P_r(v)$ denotes the position of the node within the filial set on the rth level, in the path to the terminal node v, and $h(v)$ denotes the level of v. Let V be the set of terminal nodes. Suppose there is a weight $w(v)$ associated with every terminal node v in V. Then the average (or total) search cost $C_T(v)$ of the variable length d.c.t. structure T is defined to be

$$C_T(v) = \sum_{v \in V} w(v)\, C(v) = \sum_{v \in V} w(v) \sum_{r=1}^{h(v)} P_r(v).$$

Fig. 4 illustrates a tree which is searched according to the procedure described in Section 2.1. The circled number near each leaf is the number of chaining branches required to reach that node. The numbers on the edges count the branches at each level. Suppose that we associate a weight with each leaf as shown in the figure. Then the average search cost $C_T(v)$ is given by

$$C_T(v) = 2w.4 + 2w.5 + w.6 + w.7 + w.8 + 4w.6 + 5w.7$$
$$= 98w.$$

In order to prove the main theorem which leads to our algorithm, we note first the following simple but important lemma. For a proof of this result see Ref. 17, p. 261.

Lemma

Let (a) denote the finite set $a_1, a_2, \ldots, a_j, \ldots, a_n$ of non-negatives and $(\bar{a})$ be the set (a) rearranged in

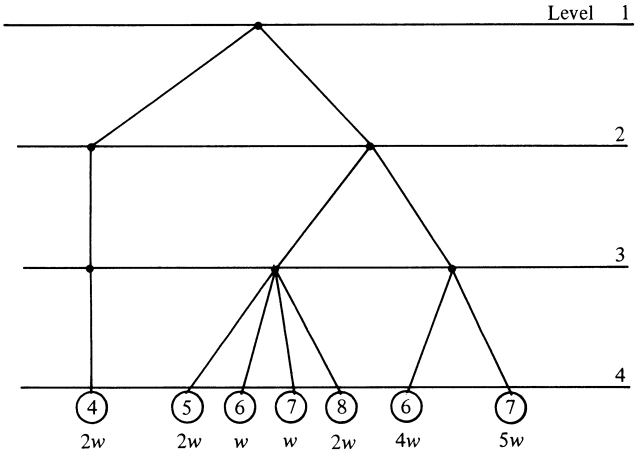

Figure 4. Computation of $C_T(v)$.

ascending order, so that $\bar{a}_1 \leqslant \bar{a}_2 \leqslant \ldots \leqslant \bar{a}_n$. Define the sets (b) and $(\bar{b})$ similarly. If (a) and (b) are given, then Σab is greatest when (a) and (b) are monotonic in the same sense, and least when they are monotonic in opposite senses; that is to say

$$\sum_{j=1}^{n} \bar{a}_j \bar{b}_{n+1-j} \leqslant \sum_{j=1}^{n} a_j b_j \leqslant \sum_{j=1}^{n} \bar{a}_j \bar{b}_j.$$

We define the *weight factor* of a node x to be the sum of the weights of the subtree whose root is the node x. We will denote this by WFACTOR(x).

Theorem 1

Suppose a data file is organised as a variable-length d.c.t. structure and the filial set of each node is specified *a priori*. If the nodes of every filial set are ordered according to decreasing weight factors, then the tree structure has minimum average search cost.

Proof

Let T denote a variable length d.c.t. structure. Let $A_0 = \{a_1, a_2, \ldots, a_i, \ldots, a_n\}$ be the filial set of a node a_0, where a_i is the ith node, ordered from left to right.
Let $R(a_i)$ be the number of terminal nodes reachable from a_i; a_{ij}, v_{ik} the terminal nodes reachable from a_i; w_{ik} the weight associated with terminal node v_{ik}.
Define

$$C(A_0) = \sum_{i=1}^{n} \sum_{k=1}^{R(a_i)} w_{ik}[C(v_{ik}) - C(a_0)]$$

where $C(v_{ik})$ and $C(a_0)$ are the search costs of v_{ik} and a_0 respectively.

$C(A_0)$ can be written as

$$C(A_0) = \sum_{i=1}^{n} \sum_{k=1}^{R(a_i)} w_{ik}\,[C(v_{ik}) - \{C(a_0) + i\} + i]. \quad (1)$$

From the description of the search procedure we have:

$$C(a_0) + i = C(a_i). \quad (2)$$

The rest of the proof is similar to the one given in Ref. 4. Let A_0 be the roots of the n-tuply rooted tree T, so that $C(A_0)$ becomes equal to the average search cost of the tree structure. Finally from (1) with (2) and simplifying we obtain

$$C(A_0) = \sum_{i=1}^{n} iW(a_i) + \sum_{i=1}^{m} iW(a_{1i}) + \ldots + \sum_{i=1}^{t} iW(a_{ni}) + \ldots \quad (3)$$

where $W(a_i) = \Sigma_{k=1}^{R(a_i)} w_{ik}$ and is referred to as the weight factor of the subtree rooted at the node a_i. From the lemma it can be shown that the summation $\Sigma_{i=1}^{n} iW(a_i)$ is minimum if

$W(a_i) > W(a_j)$ implies that $i < j$ for all $a_i, a_j \in A_0$

i.e. if the nodes are ordered according to decreasing weight factors of each. Full details of the proof are given in the Appendix.

If $w_{ik} = 1/N$ where N is the number of terminal nodes, then we get Theorem 1 of Patt[4]; when $w_{ik} = 1/N$, the average search length of d.c.t. structure is minimised if the nodes of every filial set are ordered according to the number of terminal nodes reachable from each.

Next we take an example of a d.c.t. structure in unoptimised form and convert it to the optimised form using an algorithm which is a direct consequence of Theorem 1. Fig. 5 illustrates the various stages behind the re-structuring.

Suppose we are given a tree with the weights as indicated (see Fig. 5a). Let the two nodes on level 2 be x_1 and x_2. WFACTOR(x_1) = $2w$, *and WFACTOR*(x_2) = $15w$. Now rearrange the subtrees rooted at x_1 and x_2 so that the subtree with the larger weight factor becomes the leftmost subtree.

Let x_3, x_4 and x_5 be the three nodes on level 3, in Fig. 5(b). WFACTOR(x_3) = $9w$, WFACTOR(x_4) = $6w$ and WFACTOR(x_5) = $2w$. Since WFACTOR(x_3) > WFACTOR(x_4) > WFACTOR(x_5) no change will be done to the subtrees. Finally, within each filial set belonging to the least level interchange the weight associated with the brothers so that the node with the largest weight becomes the leftmost node and successively smaller ones are placed adjacently. In Fig. 5(c) we have accomplished this, and the tree so obtained is the minimal cost tree.

Although the example shows that all the leaves are on the same level, the algorithm will work equally well for a tree where the leaves are on different levels.

4. PRACTICAL VARIATIONS OF A D.C.T. FILE

Consider a data file organised as a tree structure T, where every filial set is defined *a priori*. Now imagine a user is interested in acquiring the information associated with a terminal node v. Suppose also the information associated with the terminal node v is in the form of a record. For instance, in a library system it may be a list of references. We face a situation where the user searches through the file as if it were a variable length, doubly chained tree. When the user gets to node v, let us suppose he reads the record associated with v. Then the user effort involved consists of two actions, (a) finding the terminal node; (b) reading the record associated with v. The effort expended in (a) could be measured by the number of chaining links and the effort expended in (b) could be measured by the length of the record read. Suppose the steps corresponding

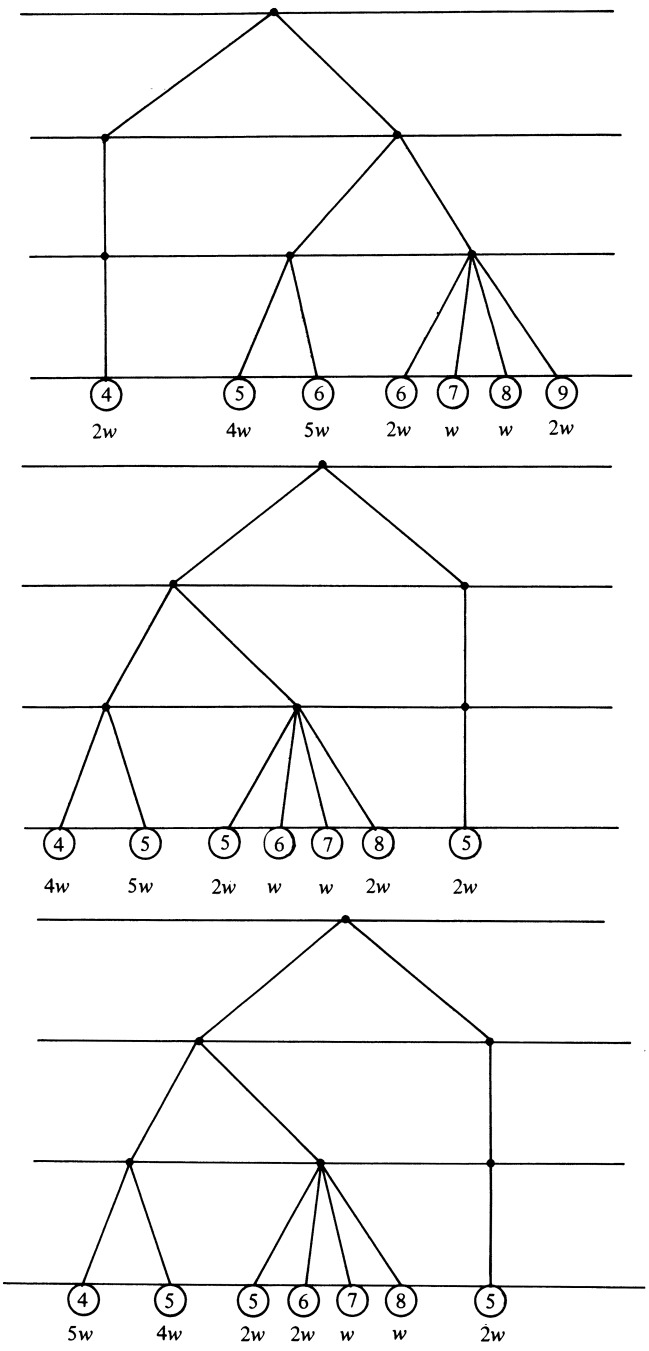

Figure 5. Construction of an optimal d.c.t. structure. (*a*) $C_T(v) = 103w$; (*b*) $C_T(v) = 90w$; (*c*) $C_T(v) = 87w$.

to these two actions have unequal costs. Let p and s denote the cost per chaining link and cost per record item read respectively. Let $C(v)$ denote the number of chaining links to reach v; let $d(v)$ and $u(v)$ be the number of record items and users associated with v respectively. Let V denote the set of all terminal nodes of T and D denote the total number of records associated with V.

User cost for the search pattern just described is given by $pC(v)+sd(v)$. Define the average cost per user per record to be

$$C_u = \frac{1}{D \sum_{v \in V} u(v)} \sum_{v \in V} u(v)\{pC(v)+sd(v)\}.$$

C_u is referred to as the total user effort and may be written as

$$C_u = \frac{1}{D} \sum_{v \in V} w(v)\{pC(v)+sd(v)\}$$

where $w(v) = u(v)/\Sigma_{v \in V} u(v)$, or

$$C_u = \frac{1}{D}[p \sum_{v \in V} w(v)\,C(v) + s \sum_{v \in V} w(v)\,d(v)]. \qquad (4)$$

Let us digress for a moment to consider the following minimisation problem, whose objective function is also the second term of equation (4).

Minimise

$$\sum_{v \in V} w(v)\,d(v)$$

subject to

$$\sum_{v \in V} w(v) = 1$$

$$\sum_{v \in V} d(v) = D,$$

$d(v)$ and $w(v)$ are non-negative.

Three possibilities for $w(v)$ and $d(v)$ are as follows: (i) $w(v)$ = constant, (ii) $d(v)$ = constant, (iii) the distributions $\{w(v)\}$ and $\{d(v)\}$ are known, but arbitrary. Now we shall present three corollaries corresponding to the three cases above.

Corollary 1

Let the user effort C_u be defined as in equation (4) and let $w(v)$ = constant. Then C_u is minimized if the nodes of every filial set are ordered according to the number of terminal nodes reachable from each.

Proof

Let i and j satisfy $1 \leqslant i, j \leqslant |A|$, where $|A|$ is the number of brothers in the filial set A. Since $w(v)$ = constant (w say), C_u becomes

$$C_u = \frac{wp}{D} \sum_{v \in V} C(v) + \frac{ws}{D} \sum_{v \in V} d(v).$$

Consequently

$$C_u = \frac{wp}{D} \sum_{v \in V} C(v) + ws.$$

From Theorem 1 it follows that C_u is minimised if for every filial set A,

$R(a_i) > R(a_j)$ implies that $i < j$ for all $a_i, a_j \in A$

i.e. if the nodes of every filial set are ordered according to the number of terminal nodes reachable from each.

Corollary 2

Let the total user effort be defined as in equation (4) and let $d(v)$ = constant. The C_u is minimised if the nodes of every filial set are ordered according to decreasing weight factors.

Proof

Since $d(v)$ = constant, d say, equation (4) can be written as

$$C_u = \frac{p}{D} \sum_{v \in V} w(v)\,C(v) + sd.$$

The rest follows from Theorem 1.

Corollary 3

Suppose the total user effort be defined as in equation (4) and suppose both distributions $\{w(v)\}$ and $\{d(v)\}$ are known. Then C_u is minimized if the nodes of every filial set are ordered according to decreasing weight factors.

Proof

Since both distributions $\{w(v)\}$ and $\{d(v)\}$ are known, the summation $\Sigma_{v\in V}\, w(v)\, d(v)$ is fixed and consequently C_u is minimised if the first summation of equation (4) is minimum. The rest of the corollary follows from Theorem 1.

So far the principal concern has been the total user effort and how the data file should be organised to mimimise the user effort under various conditions. A supporting activity, which we have not considered yet, is the maintenance effort (or the indexer effort). We assume that the maintenance cost is equivalent to the setting-up cost, where the person in charge reads the document first and then traces the path to the corresponding terminal nodes. The effect of including the maintenance effort in a measure of tree performance is considered next. Define the average maintenance cost C_m for a file item to be

$$C_m = \frac{1}{D} \sum_{v\in V} \left\{ s + p \sum_{r=1}^{h(v)} P_r(v) \right\} d(v)$$

and let

$$C = C_u + C_m$$

i.e.

$$C = \frac{1}{D} \sum_{v\in V} \left[sd(v) + sd(v)\, w(v) + p\{w(v) + d(v)\} \sum_{r=1}^{h(v)} P_r(v) \right] \quad (5)$$

C is referred to as the overall cost.

Theorem 2

Suppose C be defined as in equation (5) and both distributions $\{w(v)\}$ and $\{d(v)\}$ are known. Then C is minimised if for every filial set A

$$W(a_i) + D(a_i) > W(a_j) + D(a_j) \Rightarrow i < j \quad \text{for all} \quad a_i, a_j \in A$$

Proof

Let i and j satisfy $1 \leqslant i, j < |A|$. Let $V(a_i)$ denote the set of terminal nodes of the subtree rooted at a_i and $D(a_i)$ denote the number of records associated with this subtree. Alternatively, we shall call $D(a_i)$ as the number of records governed by the node a_i. After separating the terms in equation (5), C can be written as

$$C = \frac{1}{D} \left[s \sum_{v\in V} d(v) + s \sum_{v\in V} d(v)\, w(v) + p \sum_{v\in V} \{w(v) + d(v)\} \sum_{r=1}^{h(v)} P_r(v) \right].$$

Since both distributions $\{w(v)\}$ and $\{d(v)\}$ are known, the summation $\Sigma_{v\in V}\, d(v)\, w(v)$ cannot be controlled. Since D, p and s are constants, C is minimised if

$$\sum_{v\in V} \{w(v) + d(v)\} \sum_{r=1}^{h(v)} P_r(v) \text{ is minimum.}$$

From Theorem 1, the above expression is minimised if for every filial set A,

$$\sum_{v\in V(a_i)} \{w(v) + d(v)\} > \sum_{v\in V(a_j)} \{w(v) + d(v)\} \Rightarrow i < j$$

for all $a_i, a_j \in A$, i.e. if

$$W(a_i) + D(a_i) > W(a_j) + D(a_j) \Rightarrow i < j \quad \text{for all} \quad a_i, a_j \in A \quad (6)$$

Note. Suppose $D(a_i) \neq D(a_j)$. Then the inequality (6) reduces to

$$D(a_i) > D(a_j) \Rightarrow i < j \quad \text{for all} \quad a_i, a_j \in A.$$

The result is immediate because both $W(a_i)$ and $W(a_j)$ are less than unity and $D(a_i)$, $D(a_j)$ are integers; i.e. when $D(a_i) \neq D(a_j)$, the overall cost C as defined in equation (5) is minimised if we arrange the nodes within each filial set with respect to decreasing number of records governed by each.

5. UPDATING TREE STRUCTURES

Updating is necessary to use the d.c.t. as a dynamic data structure and depending on the application in mind there are, conceivably, a variety of ways with which a tree structure can be updated. For instance, in a library system it may be worthwhile to know whether it is cost-effective to augment the structure by splitting on a terminal node. It should be pointed out that the augmentation of the structure would greatly enhance *precision*. The phrase 'splitting on a terminal node' will be defined shortly. In this section we take up the question of updating a tree structure subject to the following restrictions: (i) augmenting the tree structure by splitting on a terminal node; (ii) rearranging or swapping two subtrees, rooted within the same filial set.

Consider a terminal node v_k in T and suppose that the list of records associated with v_k is of considerable size. In a situation like this we may partition the list associated with v_k into m_k sublists and store each sub-list at a node $v_k{}^j$ which lies at the end of a path of length one from v_k. Fig. 6 illustrates the expansion of the tree structure by splitting on the terminal node v_k. It is worthwhile to note that after the splitting v_k becomes a nonterminal node. In fact it becomes the apparent node of the newly created filial set.

In other words the set $v_k{}^j$, where $j = 1, 2, \ldots, m_k$

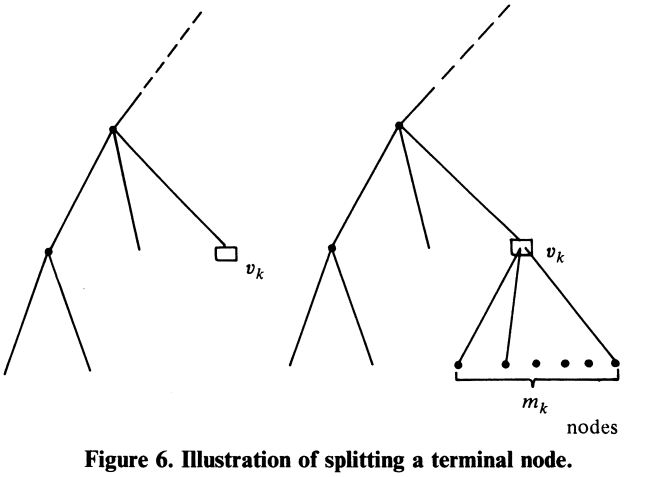

Figure 6. Illustration of splitting a terminal node.

becomes the filial set of v_k and the information items previously associated with v_k are now associated with $v_k{}^j$, so that

$$\sum_{j=1}^{m_k} d(v_k{}^j) = d(v_k)$$

and

$$\sum_{j=1}^{m_k} w(v_k{}^j) = w(v_k).$$

The above procedure is referred to as augmenting of the structure by splitting on v_k. The next result characterises the effect of splitting a terminal node restricted to a uniform distribution of users and records over the new subdivisions created, when the number of records per user is still large.

Theorem 3

Let the tree structure be augmented by splitting on the terminal node v_k. If

(i) $w(v_k{}^j) = w(v_k)/m_k, \quad j = 1, 2, \ldots, m_k,$
(ii) $d(v_k{}^j) = d(v_k)/m_k, \quad j = 1, 2, \ldots, m_k,$

and

(iii) $d(v_k) \gg w(v_k),$

then the overall cost decreases or increases according as

$$\frac{s}{p} \gtrless 0\left(\frac{1}{w(v_k)}\right).$$

Proof

Suppose the search cost for the node v_k be λ. Then the search cost for $v_k{}^j$ becomes equal to $\lambda+j$, where $j = 1, 2, \ldots, m_k$.

The overall cost C before augmenting the structure is given by

$$C = \frac{1}{D}\left[sD + s\sum_{v\in V} w(v)\,d(v) + p\sum_{v\in V}\{w(v)+d(v)\}\sum_{r=1}^{h(v)} P_r(v)\right]$$

where all the symbols are as defined before.

The overall cost C' after augmenting the structure is given by

$$C' = \frac{1}{D}\left[sD + s\sum_{v\in V-\{v_k\}} w(v)\,d(v) + sw(v_k)\,d(v_k) + p\sum_{v\in V-\{v_k\}} w(v)+d(v)\sum_{r=1}^{h(v)} P_r(v) + p\sum_{j=1}^{m_k}\{w(v_k{}^j)+d(v_k{}^j)\}(\lambda+j)\right]$$

where $V-\{v_k\}$ represent the set of terminal nodes without v_k.

Since $w(v_k{}^j) = w(v_k)/m_k$ and $d(v_k{}^j) = d(v_k)/m_k$ } for all $j = 1, 2, \ldots, m_k$

$$C - C' = \frac{1}{D}\left[sw(v_k)\,d(v_k)\left(1-\frac{1}{m_k}\right) - \frac{p}{2}\{w(v_k) + d(v_k)\}(1+m_k)\right].$$

Thus, $C - C' \gtrless 0$ according as

$$\frac{s}{p} \gtrless \frac{(1+m_k)\{w(v_k)+d(v_k)\}}{2\left(1-\frac{1}{m_k}\right)w(v_k)\,d(v_k)}, \quad m_k \neq 1.$$

The right-hand side of the above inequality can be written as

$$\frac{1+m_k}{2}\cdot\frac{m_k}{m_k-1}\left\{\frac{1}{d(v_k)}+\frac{1}{w(v_k)}\right\}.$$

Since $d(v_k) \gg w(v_k)$ the above expression becomes

$$\simeq \frac{1+m_k}{2}\cdot\frac{m_k}{m_k-1}\cdot\frac{1}{w(v_k)} = 0\left(\frac{1}{w(v_k)}\right).$$

This implies that after augmenting the tree structure subject to the conditions stated in the theorem, the overall cost decreases or increases according as

$$\frac{s}{p} \gtrless 0\left(\frac{1}{w(v_k)}\right).$$

The effect of rearranging two subtrees, rooted within the same filial set, has been investigated earlier. We have seen that the search cost could be decreased if the rearrangement is done in a certain prescribed way; i.e. economies can be effected by ordering the file in decreasing weight order. Thus, given a tree structure we can find an optimal tree structure subject to our two restrictions.

6. SUMMARY AND CONCLUSIONS

Theoretical studies on databases facilitate a better understanding of how data can be organised and searched, thereby leading to better implementations of algorithms for accomplishing these two purposes. The doubly chained tree structure provides a compromise between the fast search/slow update characteristics of binary searching and the slow search/fast update characteristics of serial searching. The main result in this paper is the algorithm, which is an immediate consequence of Theorem 1. The algorithm constructs optimal tree structures (minimises average search cost), and is extremely simple to apply. We begin at the first level of the d.c.t. structure and compute the weight factor (WFACTOR) of each root. It should be noted that the only filial set on level one consists of the set of roots – i.e. if there is more than one tree. Then we rearrange the trees/subtrees so that the tree/subtree with that largest WFACTOR becomes the leftmost tree/subtree and successively smaller ones are placed adjacently. Now, within each tree we do the same, level by level, for every filial set recursively until the last but one level is reached. At this stage we have an optimal d.c.t. structure. It is worthwhile recalling that we are not allowed to swap two subtrees if they belong to two different filial sets.

The supporting activity of maintenance effort or

indexer effort required to maintain the store of data is also considered. Finally, the tree updating process has been taken up. This makes it possible to use the d.c.t. as a dynamic data structure. The action of adding a brother to a filial set will not present any serious problems; once the system and the environmental characteristics are known the tree could be ordered to give an optimal tree structure at the time of searching.

Acknowledgements

The author wishes to express his gratitude to Dr Christopher D. Green, University of Dundee, for his many valuable suggestions and encouragement. This research was supported in full by the Department of Mathematics, University of Dundee, and is part of the author's Ph.D. Thesis.

Many thanks are due to the referee, whose suggestions considerably improved the quality of exposition of this paper.

REFERENCES

1. René de la Briandais, File searching using variable length keys. *Proceedings of the Western Joint Computer Conference*, 295–298 (1959).
2. E. Fredkin, Trie memory, *Communications of the ACM* **3**, 490–499 (1960).
3. E. H. Sussenguth Jr, Use of tree structures for processing files. *Communications of the ACM* **6**, 272–279 (1963).
4. Y. N. Patt, Variable length tree structures having minimum average search time. *Communications of the ACM* **12**, 72–76 (1969).
5. L. E. Stanfel, Tree structures for optimal searching. *Journal of the ACM* **17**, 508–517 (1970).
6. L. E. Stanfel, Practical aspects of doubly chained trees for retrieval. *Journal of the ACM* **19**, 425–436 (1972).
7. L. E. Stanfel, Optimal tree lists for information storage and retrieval. *Information Systems* **2**, 65–70 (1976).
8. H. A. Clampett, Randomized binary searching with tree structures, *Communications of the ACM* **7**, 163–165 (1964).
9. R. G. Casey, Design of tree structures for efficient querying. *Communications of the ACM* **16**, 549–556 (1973).
10. J. Nievergelt, Binary search trees and file organization. *Computing Surveys* **6**, 194–206 (1974).
11. S. B. Yao, Tree structures construction using key densities. *ACM Annual Conference, Minneapolis, Minn.* 337–340 (1975).
12. F. Suraweera, Classification, searching and graphs in information retrieval. *Ph.D. Thesis*, University of Dundee (1976).
13. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, *Searching and Sorting*. Addison & Wesley, Reading, Mass. (1973).
14. G. Salton, *Automatic Information Organization and Retrieval*. McGraw-Hill, New York (1968).
15. Alfonso F. Cardenas, Doubly chained tree database organisation – analysis and design strategies. *The Computer Journal* **20**, 15–26 (1977).
16. Douglas Comer and Ravi Sethi, The complexity of trie index construction. *Journal of the ACM* **24**, 428–440 (1977).
17. G. H. Hardy, J. E. Littlewood and G. Polya, *Inequalities*, Cambridge, Cambridge University Press, 261–262 (1934).

APPENDIX

Details of the proof of Theorem 1.

Define

$$C(A_0) = \sum_{i=1}^{n} \sum_{k=1}^{R(a_i)} w_{ik}[C(v_{ik}) - C(a_0)]$$

$C(A_0)$ can be written as

$$C(A_0) = \sum_{i=1}^{n} \sum_{k=1}^{R(a_i)} w_{ik}[C(v_{ik}) - \{C(a_0) + i\} + i] \qquad (1)$$

and

$$C(a_0) + i = C(a_i) \qquad (2)$$

After substituting from (2) and separating the terms under the double summation, (1) becomes

$$C(A_0) = \sum_{i=1}^{n} \sum_{k=1}^{R(a_i)} w_{ik}[C(v_{ik}) - C(a_i)] + \sum_{i=1}^{n} \sum_{k=1}^{R(a_i)} i w_{ik}$$

But $\Sigma_{k=1}^{R(a_i)} w_{ik}[C(v_{ik}) - C(a_i)]$ satisfies the defining equation of $C(A_i)$, where A_i is the filial set of node a_i. Thus,

$$C(A_0) = \sum_{i=1}^{n} C(A_i) + \sum_{i=1}^{n} \sum_{k=1}^{R(a_i)} i w_{ik}$$

or,

$$C(A_0) = \sum_{i=1}^{n} C(A_i) + \sum_{i=1}^{n} iW(a_i)$$

where $W(a_i) = \Sigma_{k=1}^{R(a_i)} w_{ik}$ and is referred to as the weight factor of the subtree rooted at node a_i.

Now let A_0 be the roots of the n-tuply rooted tree T, so that $C(A_0)$ becomes equal to the average search length of the tree structure. Then $C(a_i) = i$ and consequently $C(a_0) = 0$. We can write $C(A_0)$ in terms of its filial set:

$$C(A_0) = C(A_1) + C(A_2) + \ldots + C(A_n) + \sum_{i=1}^{n} iW(a_i)$$

Similarly for any filial set A we can write $C(A)$ in terms of its filial set. Thus we can systematically obtain the following set of equations:

$$C(A_0) = C(A_1) + C(A_2) + \ldots + C(A_n) + \sum_{i=1}^{n} iW(a_i)$$

$$C(A_1) = C(A_{11}) + C(A_{12}) + \ldots + C(A_{1m}) + \sum_{i=1}^{m} iW(a_{1i})$$

$$\vdots$$

$$C(A_n) = C(A_{n1}) + C(A_{n2}) + \ldots + C(A_{nt}) + \sum_{i=1}^{t} iW(a_{ni})$$

Since T is finite the above set of equations is also finite. Combining the above equations we obtain

$$C(A_0) = \sum_{i=1}^{n} iW(a_i) + \sum_{i=1}^{m} iW(a_{1i}) + \ldots + \sum_{i=1}^{t} iW(a_{ni}) + \ldots$$

where each filial set A contributes one term to the right-hand side of the composite equation. The average search length (cost), therefore, is minimised if for every filial set A, the summation $\Sigma iW(a_i)$ is minimised. Now from the lemma the result follows.