Trees, Forests and Rearranging

By P. F. Windley

This paper describes a method of rearranging data in a computer, so as to minimize the amount of storage space used. Expressions for the mean and variance of the numbers of comparisons necessary, are produced. These are compared with the number of comparisons required in a merging procedure.

Introduction

Friend (Friend, 1956) considers the influence of access time when merging data stored on magnetic tape and indicates that a special first pass is necessary to produce long sequences, in order to reduce the subsequent number of passes required. He also discusses, as does Bell (Bell, 1958), several techniques for doing the necessary preliminary rearrangement within the computer. All the methods they consider involve a factor of N^2 in the expression for the number of comparisons needed (where N is the number of items) except internal merging, where the expression is $N[\log_2 N]$. The two disadvantages of internal merging are that it needs working space sufficient to hold at least $N/2$ items, which reduces the length of the sequences which can be produced internally, and that at every pass all the data is moved from one set of locations in the machine to another. If the key is short these objections can be partly removed by detached-key techniques. The method of rearranging by use of a "tree," described briefly by Douglas (Douglas, 1959) and attributed to Berners-Lee, does not suffer from these disadvantages. Once the data has been placed in the store, it is not moved until it is picked out in its final order. Sufficient storage space is only required to hold three addresses with each item of data. The number of comparisons necessary depends on the initial sequence. We consider here some variations of this method and derive expressions for the mean and variance of the comparisons, taken over all possible initial sequences.

Trees

A tree may be defined as a network of points arranged in levels. The bottom level contains only one point called the *root point*. Any other point is connected to one and only one point in the next lower level and up to n points in the next higher level. For a binary tree, $n = 2$ and we shall discuss only binary trees in what follows. A direction is associated with each connection from one point to another at the next higher level. These directions are called *left* and *right*. An item of data is associated with every point of the tree. The addresses of those items which are connected direct to a certain item, are stored with that item. These addresses are known as the *left address*, the *right address* and the *back address*. The left address indicates the location in the machine where the item on the next higher level of the tree in direction left is stored. The back address indicates the point on the next lower level to which that item is connected. All points connected to an item through its left address form the *left branch* of that item. Fig. 1 shows how items A–F, together with their connecting addresses are stored in locations 1–6 of a computer to form the tree shown.

The left branch of the root point A contains items B, D, E and F.

The Growth of a Tree

A tree grows by taking the items in their initial sequence and placing them one by one on the tree. The tree is made to grow in such a way that for every item X, all the items on the left branch of X are required before X and all the points on the right branch of X after X, in the final order. For example, the tree in Fig. 1 represents the initial sequence A, B, C, D, E, F and the final order F, E, B, D, A, C. Suppose that it is required to add item G to this tree, and that G occurs between B and D in the final order. G is first compared with the root point A. G comes before A in the final order so it must be stored on the left branch of A. The left address of A specifies the next item, B, to be compared with G.

LOCATION	ITEM	LEFT ADDRESS	RIGHT ADDRESS	BACK ADDRESS
1	A	2	3	
2	B	5	4	1
3	C			1
4	D			2
5	E	6		2
6	F			5

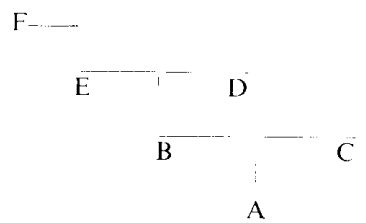

Fig. 1.—Storage of items in the form of a tree

LOCATION	ITEM	LEFT ADDRESS	RIGHT ADDRESS	BACK ADDRESS
1	A	2	3	
2	B	5	4	1
3	C			1
4	D	7		2
5	E	6		2
6	F			5
7	G			4

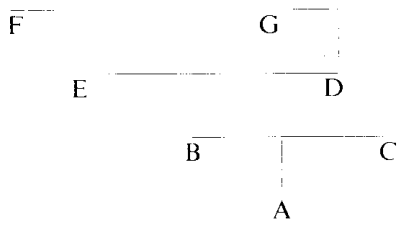

Fig. 2.—The growth of the tree

This comparison shows that G must be stored on the right branch of B. Hence the next comparison is made with D, the item specified by the right address of B. The final comparison shows that G belongs to the left branch of D. D as yet has no left branch so it is given left address 7, the address in which G is going to be stored, and G is given back address 4, the address of D. The new tree is shown in Fig. 2.

Note that G does not have to be placed in location 7. It can be placed in any vacant location provided the left address of D is adjusted accordingly. The algorithm for placing a point on the tree is:

(1) Set the address of the root point as the address of the item with which to make the next comparison. Jump to 3.
(2) Set the left (or right) address as the address of the item with which to make the next comparison.
(3) Compare the item about to be placed on the tree with the item specified in 1 or 2.
(4) Decide whether the new item is to be placed on the left or right branch of the item with which it has just been compared.
(5) Extract the next left (or right) address and examine it for zero. If not zero jump to 2.
(6) Place the new item on the tree with its back address referring to the last item with which it was compared.
(7) Set the left (or right) address previously zero, to the address in which the new item was placed in 6.

Picking the Data off the Tree

The data must be picked off the tree in such a way that for every item X, all the data on the left branch of X is picked before X, and all the data on the right branch of X is picked after X. Suppose the program has reached item X, now proceed as follows:

(1) Examine the left address of X. If it has no left address then either it has no left branch or all the points on its left branch have already been picked. In this case proceed to 2. If it has a left address remove this address, use it to specify X′ the item for the next examination, and repeat from 1.
(2) Examine X to see if it has been picked. If it has, jump to 4. If not, pick it and mark it as picked.
(3) Examine the right address of X. If this address exists, remove it, use it to specify X′, the item for the next examination, and repeat from 1. If no right address exists, proceed to 4.
(4) Examine the back address of X. If no back address exists, X is the root point and the procedure finishes. Otherwise the back address specifies the item for the next examination. Repeat from 1.

The procedure starts by examining the root point and moves up and down every connection in the tree. Hence, $2N$ examinations are made. Every time a climb is made up the tree, an address in the main store is removed, also before a climb to the right a mark is made to say that an item has been picked. This mark need not be made if there is no right address, and by suitable packing it can be made at the same time as the right address is removed. Hence this procedure makes in all $3(N-1)$ references to the store: two references for every climb up the tree and one for every climb down.

The Number of Comparisons necessary during the Growth of the Tree

Assign a value to every level of the tree. The root point has value 1, the points on the next level 2, and so forth. Let N be the number of points on the tree. The total value of the tree, which is the sum of the values of all the points on the tree, depends on the shape of the tree and is determined by the initial sequence of the data.

Denote the shape of the tree by S and the total value of the tree by $R(N, S)$. Let $P(N, S)$ be the probability that a tree of N points has shape S. A point on the rth level of the tree must have caused comparisons with the $(r-1)$ points on whose branches it is placed. Therefore the total number of comparisons is $R(N, S) - N$.

The Mean Value of a Tree

The Mean Value $Q(N)$ of a tree of N points is given by $Q(N) = \sum_S P(N, S)R(N, S)$, where the summation is

taken over all possible shapes of tree. Now the left branch of the root point is a tree in its own right. Suppose it has t points and its shape is S'. Its value must be $R(t, S') + t$, (since its root point has value 2). The right branch of the root point is also a tree. It has $(N - 1 - t)$ points and shape S'' (say). Its value is

$$R(N - 1 - t, S'') + (N - 1 - t)$$

$$\therefore\ R(N, S) - 1 + R(t, S') - t + R(N - 1 - t, S'') + (N - 1 - t) = R(t, S') + R(N - 1 - t, S'') + N.$$

The probability that there are t points on the left branch of the root point is the probability that the root point is the $(t + 1)$th point in the final order. But it is equally probable that the root point will be anywhere in the final order.
Hence the probability is $1/N$.

$$\therefore\ Q(N) = \sum_S P(N, S)R(N, S)$$

$$= \frac{1}{N}\sum_{t=0}^{N-1}\sum_{S'}\sum_{S''}\{(P(t, S')P(N - 1 - t, S'')[R(t, S') + R(N - 1 - t, S'') + N]\}$$

$$= N + \frac{2}{N}\sum_{t=0}^{N-1} Q(t). \qquad (1)$$

The Mean Deviation of the Value of the Tree

The mean deviation $V(N)$ is given by

$$V^2(N) = U(N) - Q^2(N)$$

where $U(N)$

$$= \sum_S P(N, S)R^2(N, S)$$

$$= \frac{1}{N}\sum_{t=0}^{N-1}\sum_{S'}\sum_{S''}\{P(t, S')P(N - 1 - t, S'')[R(t, S') + R(N - 1 - t, S'') + N]^2\}$$

$$= \frac{1}{N}\sum_{t=0}^{N-1}\sum_{S'}\sum_{S''} P(t, S')P(N - 1 - t, S'')[R^2(t, S') + R^2(N - 1 - t, S'') + N^2 + 2NR(t, S') + 2NR(N - 1 - t, S'') + 2R(t, S')R(N - 1 - t, S'')]$$

$$= \frac{1}{N}\sum_{t=0}^{N-1}\sum_{S'}\{P(t, S')[R^2(t, S') + U(N - 1 - t) + N^2 + 2NR(t, S') - 2NQ(N - 1 - t) + 2R(t, S')Q(N - 1 - t)]\}$$

$$= \frac{1}{N}\sum_{t=0}^{N-1}[U(t) + U(N - 1 - t) + N^2 + 2NQ(t) + 2NQ(N - 1 - t) + 2Q(t)Q(N - 1 - t)]$$

$$= N^2 + \frac{2}{N}\sum_{t=0}^{N-1}[U(t) + 2NQ(t) + Q(t)Q(N - 1 - t)]$$

$$= 2NQ(N) - N^2 + \frac{2}{N}\sum_{t=0}^{N-1}[U(t) + Q(t)Q(N - 1 - t)] \qquad (2)$$

$$\therefore\ V^2(N) = \frac{2}{N}\sum_{t=0}^{N-1}[U(t) + Q(t)Q(N - 1 - t)] - [Q(N) - N]^2.$$

Determination of $Q(N)$

From (1), $Q(N) = N + \frac{2}{N}\sum_{t=1}^{N-1} Q(t).$

Hence $Q(N - 1) = (N - 1) + \left(\frac{2}{(N - 1)}\right)\sum_{t=1}^{N-2} Q(t)$

$$\therefore\ NQ(N) - (N - 1)Q(N - 1) = (2N - 1) + 2Q(N - 1)$$

$$\therefore\ \frac{Q(N)}{N + 1} - \frac{Q(N - 1)}{N} = \frac{2N - 1}{N(N + 1)} = \frac{3}{N + 1} - \frac{1}{N}$$

$$\therefore\ \frac{Q(N)}{N + 1} - \frac{Q(1)}{2} = \frac{3}{N + 1} + \sum_{t=2}^{N-1}\frac{2}{t + 1} - \frac{1}{2}.$$

Hence $Q(N) = 2(N + 1)\sum_{t=1}^{N}\frac{1}{t} - 3N.$

The formal series for $\sum_{t=1}^{N}\frac{1}{t}$ obtained by the Euler–Maclaurin formula is

$$\log N + \gamma + \frac{1}{2N} - \sum_{i=1}^{\infty}\frac{B_i}{2iN^{2i}}$$

(where γ is Euler's constant and the B_i are the Bernoulli Numbers)

$$\gamma = 0{\cdot}57721566 \text{ approximately.}$$

$$\therefore\ Q(N) = 2(N + 1) \times \left(\log_e N + \gamma + \frac{1}{2N} - \frac{1}{12N^2} + \frac{1}{120N^4} + \ldots\right) - 3N. \qquad (3)$$

Table 1

N	$Q(N)$	$V(N)$
10	34	4
20	91	10
30	158	16
40	231	22
50	309	28
100	748	59
150	1,238	91
200	1,763	123
250	2,313	155

The Evaluation of the Mean and Variance

Great care must be taken in computing $Q(N)$ and $V(N)$ since the recurrence relations derived from (1) and (2) are unstable. The values given in Table 1 were evaluated "double-length" on the computer. The accuracy of the values of $Q(N)$ was checked by comparing them with the values obtained from the asymptotic expansion (3). It will be seen that the variance is small compared to the mean.

Time taken by the Method

Consider first a machine of the Pegasus type, with a magnetic drum main store and a small high-speed computing store. Assume that a complete item of data can be held in the computing store and that it can be transferred to and from the drum by one or more block transfers. It may be assumed that, for the second and subsequent block transfers, the drum is in the correct position so that no waiting time is involved. Each comparison causes the key and the left or right address to be brought into the computing store. When each item is placed in the main store, it is given a back address, and an address which is stored with another item is adjusted to connect the new item on to the tree. This causes $2N$ transfers. As there are on average $Q(N) - N$ comparisons, the total number of transfers needed is $Q(N) + N$. As seen previously $3N$ transfers are necessary to pick the data off the tree. Therefore, in all, the number of transfers necessary for the complete process is $Q(N) + 4N \simeq (N + 1)\log_e N + (1 + \gamma)N = 1{\cdot}4(N - 1)\log_2 N - (1 - \gamma)N$ approximately. A merging process requires $2N[\log_2 N]$ transfers.

On a machine whose main store is made of magnetic cores, a comparison of the two methods must not be made simply on the number of transfers to and from the store. The number of times the inner loops are obeyed by the two programs must be examined, and the comparative lengths of these loops estimated. For tree sorting there are two inner loops. The first is used during the growth of the tree and is obeyed $Q(N) - N$ times. The second is used during the picking-off process and is used $2N$ times. Let us suppose that α is the time taken to make a comparison, β the time taken to move a complete item of data from one location or set of locations in the machine to another place, λ the time taken by the administrative orders necessary to connect a new item on to the tree, and λ^* the average time taken by the picking-off routine to find the next item. Ignoring the time taken initially to place all the data in the store, which is the same for all methods, the total time taken by the tree method is therefore approximately

$$\alpha[Q(N) - N] + N\lambda + 2N\lambda^*.$$

The time taken by merging is

$$(\alpha + \beta)N[\log_2 N].$$

It is sufficient to compare

$$1{\cdot}4\alpha N[\log_2 N] \text{ with } (\alpha + \beta)N\log_2 N.$$

Hence, provided $\alpha < 2{\cdot}5\beta$, the tree method will be the faster.

Retrieval of Information

Trees may be used to hold data in the main store of a machine so as to facilitate the retrieval of information belonging to a given key. When this is done an average of $Q(N)/N \simeq \log_e N$ comparisons are required to identify each item.

A Forest of Trees

It has already been mentioned that, when an item is placed on a tree, it can be placed anywhere in the store not already used to hold an item of the tree. Both for the growth of the tree and for the picking of items off the tree, the address of the root point is the only parameter necessary to start the process. All other addresses required are already stored on the tree. Hence it is possible to have two trees growing in the machine at the same time without interfering with each other. The trees have distinct root points. Each location in the machine is allotted, as the trees grow, to either tree as required. When an item is to be placed on a tree, it is first examined to see to which tree it belongs. The appropriate root point is then selected to start the comparisons.

After the necessary comparisons have been made, the new item is placed in the next vacant location in the machine and is linked through its back address to the correct tree. Not just two but any number of trees may be used provided that each tree has a location in the machine designated for its root point. To pick off the items the same routine is used. It is re-entered for every tree, starting at each root point. The advantage of this multi-tree or *forest* technique is that the value of two trees is less than the value of a tree equal in size to the two trees put together. The selection of the correct tree for each item may be done by splitting the range of the key into areas and allotting a tree to each area. It will be noticed that this selection is equivalent to one pass of a distribution sort, but no estimate is needed for the size of the pockets.

Theoretically, as many trees as possible should be used in the forest, but if any tree is void (i.e. it does not have any items on it) the space allotted to its root point is wasted. In practice, the optimum number depends on the distribution of the keys over their possible range, and how much is known about it. In the limit, when every key defines a tree, the method degenerates into a pigeon-hole sort. The method is ideally suited for cases where the data has to be arranged in sequence within a number of groups and the criterion for the determination of the groups is different from the criterion to be used for the ordering of the data within those groups. These,

indeed, were the conditions for which this technique was developed.

The data referred to students in the University of Leeds (see Windley, Kay and Rowland-Jones, 1960) and it was required to compile a list of students arranged alphabetically under courses. The data was first examined for the student's course; this was used to select a tree, and the comparisons were then made on the student's name.

Trees and Sorting

Trees can be used very effectively for sorting. By sorting is meant the accumulating of items with the same key. To use a pigeon-hole sort it is necessary to be able to assign a location in the machine to every possible key over its complete range. In tree sorting, space is only required for the accumulated items actually encountered during the sort, and for three block addresses for each distinct key. By marking the keys which are known to occur most often, and assigning a separate tree to all marked keys, it is possible to utilize the speed of pigeon-hole sorting to deal with the most frequent keys. This may introduce a considerable gain in speed in applications to accounting where some accounts are much more active than others. In tree sorting, items with the same key are accumulated as they are placed on the tree. Thus at no time is it necessary to store the same key twice. This is a great improvement on sorting by merging, where the items are only gradually accumulated.

It is also possible to use trees, in a slightly different form, to generate the best strategies for other sorting and rearranging methods. These techniques are discussed in a paper by Burge (Burge, 1958).

Conclusion

The method of rearrangement by trees was first considered by the author in order to produce the longest possible sequences of data inside a computer. It has been shown that, given random initial sequence, the method is, on average, faster than rearrangement by merging. The method has been used successfully in two programs relating to student registration, mentioned above. First it was used to rearrange the data into various orders as required by the University office. This program only used 6% of the store as working space. Secondly it was used to count the number of students attending each course at a time when the data was not rearranged in order under courses. Pigeon-hole sorting could not be used for this because of the large number of possible course codes, only a few of which were assigned to actual courses. This use of the method as a counter-sorter may well have applications in the market-research field, where it is required to compute a weighted count, under a number of different headings, from a large amount of data. By using a forest of trees it will be possible to perform several counts at the same pass of the complete data. This technique is useful where the coding of the main data is such that a pigeon-hole sort cannot be used because of the wide range of possible keys.

References

BELL, D. A. (1958). "The Principles of Sorting," *The Computer Journal*, Vol. 1, p. 71.

BURGE, W. H. (1958). "Sorting, Trees and Measures of Order," *Information and Control*, Vol. 1, p. 181.

DOUGLAS, A. S. (1959). "Techniques for the Recording of, and Reference to Data in a Computer," *The Computer Journal*, Vol. 2, p. 1.

FRIEND, E. H. (1956). "Sorting on Electronic Computer Systems," *J. Assoc. Comp. Mach.*, Vol. 3, p. 134.

WINDLEY, P. F., KAY, L. F., and ROWLAND-JONES, A. (1960). "Data Processing in University Administration," *The Computer Journal*, Vol. 3, p. 15.