May 20 1957

PROJECT: STRETCH

SUBJECT: A General Technique for Searching Tables with a Non-Uniform Interval

The technique to be described includes binary searching and sequential searching. as special cases. The main principle involved is that of storing the pattern of search within the table itself. Two important advantages of this method are:

1. The process is easy to mechanize

2. The table entries need not be stored in consecutive or evenly spaced locations.

In order to explain the process the following table will be taken as an example:

01 03 09 10 11 15 18 24 25 30 31 37 39 51 54 56 57 71 89

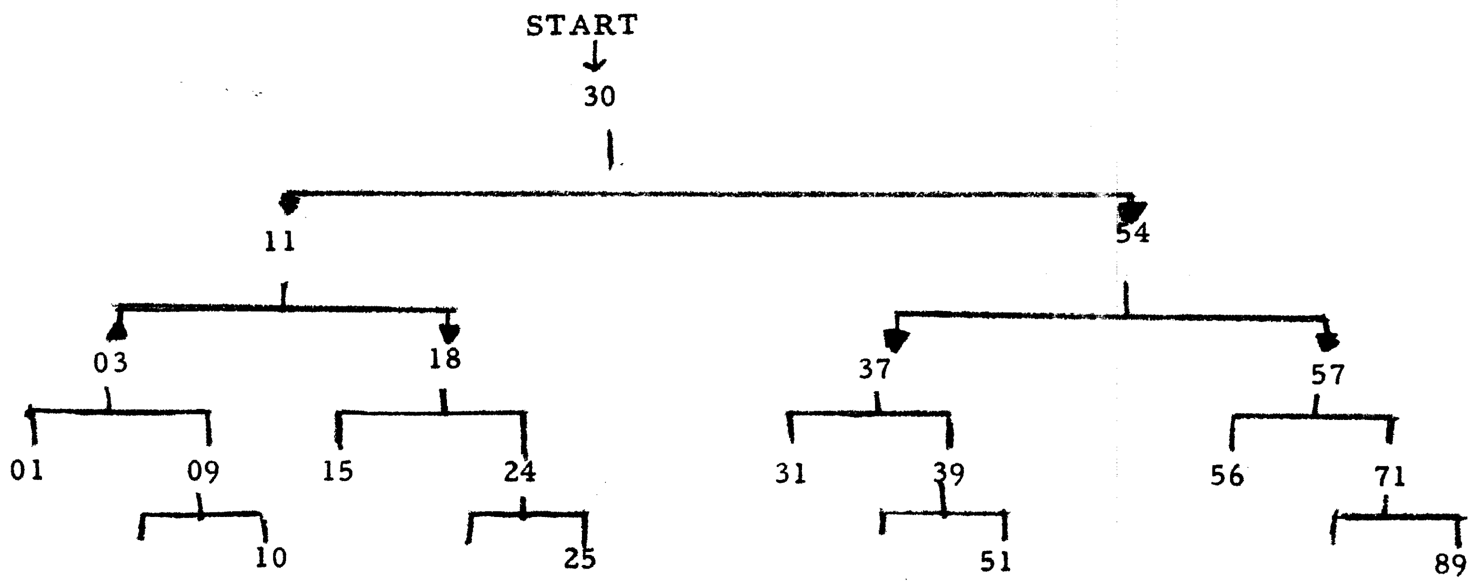
For simplicity these 19 entries will be assumed to occupy 19 consecutive locations in memory with addresses 0 through 18 relative to the table origin. Alongside each of the table entries we now place two addresses which will be used to specify the pattern of searching. One of these is called the LOW ADDRESS and indicates where to look next if the given argument is found to be strictly less than the table entry. The other address is called the HIGH ADDRESS and indicates where to look next if the given argument is found to be strictly greater than the table entry. In case equality occurs the search is ended.

Let us suppose first of all that the desired pattern of search is that used in binary searching. For the table chosen above this pattern can be represented as in Fig. 1. The table is set up as in Fig. 2. Now, given a computed argument say 37, the Search proceeds as follows:

Step 1: Start at location 9. Compare the table argument 30 with the computed argument 37. We find a HIGH condition and therefore, go to the HIGH ADDRESS, namely 14.

Step 2: Pick up the contents of 14 and compare the table argument 54 with the computed argument 37. We find a LOW condition and therefore, go to the LOW ADDRESS, namely 11.

Figure 1: BINARY SEARCH PATTERN



Location	Table Arg.	Low Addr	High Addr.
0	01	STOP	STOP
1	03	00	02
2	09	STOP	03
3	10	STOP	STOP
4	11	01	06
5	15	STOP	STOP
6	18	05	07
7	24	STOP	08
8	25	STOP	STOP
Search Starts Here → 9	30	04	14
10	31	STOP	STOP
11	37	10	12
12	39	STOP	13
13	51	STOP	STOP
14	54	11	16
15	56	STOP	STOP
16	57	15	17
17	71	STOP	18
18	89	STOP	STOP

Figure 2: TABLE LAYOUT FOR BINARY SEARCH

Step 3: Pick up the contents of 11 and compare again. This time we get an EQUAL condition, which automatically ends the search.

The Search has yielded two items of information: the address 11 and a status indication of the EQUAL condition.

Although this example illustrates the main features of the process one might well ask:

1. How is the case handled in which the computed argument is not equal to any of the table arguments?

2. Under what circumstances would a search pattern other than the binary type be advantageous?

3. How may this technique be applied to a computer with a 64-bit word?

These questions are now treated in turn.

When the computed argument Y is not equal to any of the table arguments X_i, the search ends in one of two ways: the current address is either that of X_j <u>or</u> that of X_{j+i} (where $X_j < Y < X_{j+i}$). In the first case the LOW indicator is turned on, in the second the HIGH. By referring to the example given earlier it will readily be observed that the specified pattern of search determines for any given computed argument which of the two end results is obtained.

Regarding choice of search pattern, if nothing is known about the distribution of frequency of references over the given table, binary searching will be the natural choice. If, on the other hand, the table has a known non-uniform distribution, the pattern of search can be biased to take advantage of this situation.

One method of incorporating this technique into the instruction repertoire of a computer with a 64-bit word is as follows. The instruction SEARCH TABLE specifies the starting location relatively or absolutely. Table words are assumed to have the format:

TABLE ARGUMENT	39 bits (inc. Sign, if present)
LOW ADDRESS	12 bits
HIGH ADDRESS	12 bits
STOP BIT	1 bit

The sign modifier portion of the instruction indicates how signs will be handled during the search. The computed argument is assumed to be pre-loaded into one of the LU registers,, the accumulator for example. The process then follows the pattern indicated in the flow chart on the next page.

The 12-bit limitation on addresses in the table word can be readily overcome by dividing a table into sections not larger than 4096 words, and setting up a higher table to determine which section to search in.

FLOW CHART FOR SEARCHING PROCESS

Copy effective address into LU register A

↓

→ Fetch table word from effective address and place in B

↓

Compare computed argument in ACCUMULATOR with the table in B and set the HIGH, EQUAL or LOW status indicator. If the equal condition arises, stop the search

↓

Inspect the STOP BIT in B. If it is 1, stop the search

↓

According to the setting of the LOW and HIGH status indicators, copy the LOW or HIGH ADDRESS in B into A and into the instruction register.

↓

Index if necessary. (return to "Fetch table word")

APPLICATION OF SEARCHING TECHNIQUE

It has been stated that the result of searching a set of table arguments X_i with a computed argument Y is to obtain:

1) L (X_j) and EQUAL status bit on, If $X_j = Y$.

OR 2) L (X_j) and LOW status bit on } if $X_j < Y < X_{j+i}$

OR 3) L (X_{j+i}) and HIGH status bit on }

Although the L (X_j) and L (X_{j+i}) enable the computer to pick up the corresponding table arguments, this is rarely adequate. If the table represents a mathematical or empirical function, it is likely that one or more tabulated function values are required along with their corresponding table arguments in order to carry out an N^{th} order interpretation. A simple recipe for finding F (X_j) is to index L (X_j) by an increment p say, where L (X_j) + p = L (F (X_j)) for all j. A more general solution is to place in L (X_j) +p addresses which guide the computer to L (X_{j+i}), L (F (X_j)), etc.

This latter approach to finding information associated with a set of table arguments enables one to use two or more sets of table arguments (each set having a different meaning and being differently ordered) to act as signposts to a single file which is required to be searched at any time on any one of several items.

In an increasing number of applications it is found inconvenient or impossible to store information in a manner which would enable a simple table look-up operation to find the information required. The incorporation of a fast table search operation in a computer would considerably increase its effectiveness on such applications.

E. F. Codd

EFC/jv