

# MULTICS DEMONSTRATION PROGRAM #1

March 4, 1970

FUNCTION: to construct an arbitrary part of a 36-level binary tree

What is demonstrated: How use of a large virtual memory can simplify construction of a program.

-----

This one-page program constructs in virtual memory an arbitrarily large part of a 36-level binary tree. Such a tree is an effective way of storing for retrieval (or checking for duplicates among) a large number of short (in this case 36-bit) information items.

In a 3-bit example, the information item "010" traces out the heavily outlined path in the 3-level tree of figure 1. The program would construct in an open-ended storage area a list structure describing just the appropriate portion of the tree, using non-null branches in the list structure to correspond to branches so far placed in the tree. If now the information item "011" is to be stored, the program only needs to add to the tree the single additional branch outlined in Figure 2. To inquire whether the item "010" has previously been stored, one attempts to trace the list structure corresponding to that portion of the tree; success in reaching the end means that the item has been previously stored.

For this demonstration, the program calls on a pseudo-random number generator to generate 36-bit information items. After every 100 items have been stored, it reports this fact to the typewriter and also reports the total amount of virtual storage so far consumed. In Multics, this program runs at a fairly constant rate as the tree builds up, even though the program may be dealing with several million words of virtual addressable memory. This is because the frequently referenced base of the tree tends to be substantially filled out by the first few storage items and thus tends to be concentrated near the beginning of the storage area in a few pages which remain in core; while reference to the extremities of the tree requires that the supervisor fetch only a few pages for each new information item.

The ability of the Multics file system to make mass storage easily accessible to the program is exhibited by the fact that the demonstration program is only about 1 page of code in the PL/I language and includes only a very few lines of "file manipulation".

This program was suggested by Edward Fredkin, and is based on his "trie memory".<sup>1</sup>

---

<sup>1</sup> Fredkin, E.M., "Trie Memory" Communication of the ACM, Sept. 1960, p. 490.

Dev = please retype

Revised  
Sent  
Per Kelly  
Cody

### MULTICS DEMONSTRATION PROGRAM #1

March 14, 1969 <sup>near</sup> <sub>date</sub>

FUNCTION: to construct an arbitrary part of a 36-level binary tree

What is demonstrated: How use of a large virtual memory can simplify construction of a program.

-----  
This one-page program constructs in virtual memory an arbitrarily large part of a 36-level binary tree. Such a tree is an effective way of storing for retrieval (or checking for duplicates among) a large number of short (in this case 36-bit) information items.

In a 3-bit example, the information item "010" traces out the heavily outlined path in the 3-level tree of figure 1. The program would construct in storage just that portion of the tree. If now the information item "011" is to be stored, the program only needs to add to the tree the single additional branch outlined in figure 2. To inquire whether the item "010" has previously been stored, one attempts to trace that portion of the tree; success in reaching the end means that the item has been previously stored.

For this demonstration, the program calls on a non-repeating pseudo-random number generator to generate 36-bit information items. After every 100 items have been stored, it reports this fact to the typewriter and also reports the total amount of virtual storage so far consumed. In Multics, this program runs at a fairly constant rate as the tree builds up, even though the program may be dealing with several million words of virtual addressable memory. This is because the frequently referenced base of the tree is concentrated in a few pages which remain in core; while reference to the extremities of the tree requires that the supervisor fetch only a few pages for each new information item.

The ability of the Multics file system to make mass storage easily accessible to the program is exhibited by the fact that the demonstration program is only about 1 page of code in the ~~EPL~~ (a dialect of PL/I) language and includes only a very few lines of "file manipulation".

tends to be filled out by the first few storage items and is this concentrated near the beginning of the storage area.

This program was suggested by Edward Fredkin, and is based on his "trie memory". 1

1 Fredkin, E.M., "Trie Memory" Communications of the ACM, Sept. 1960, p. 990.

an open-ended storage area

linear  
of bits  
lay out

the list structure corresponds to

using a well branched in the list structure to correspond to branches in the tree.

a list structure describing just the appropriate

end to the

Fig 1

Fig 2

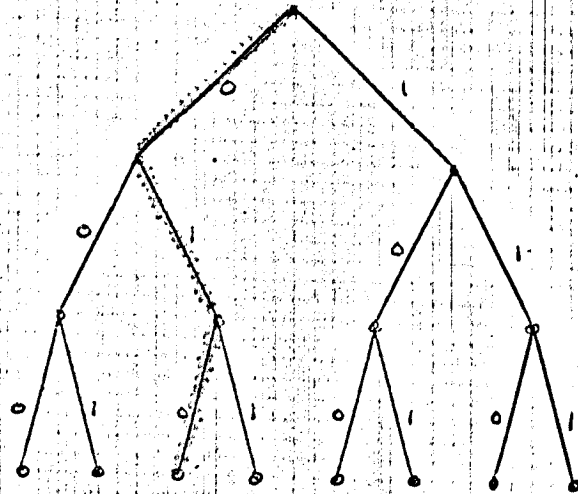


Figure 1--Portion of binary tree required to store information item "010"

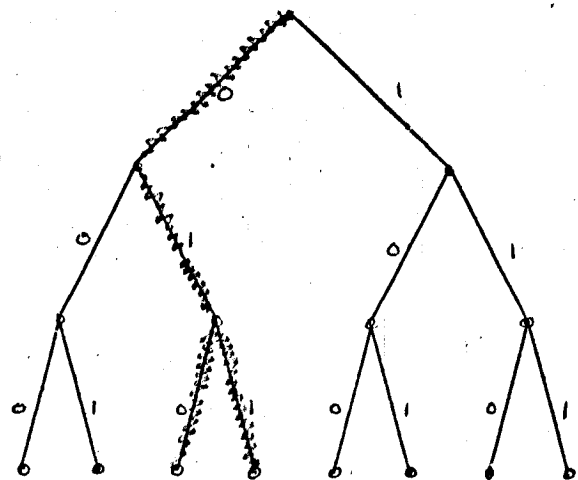
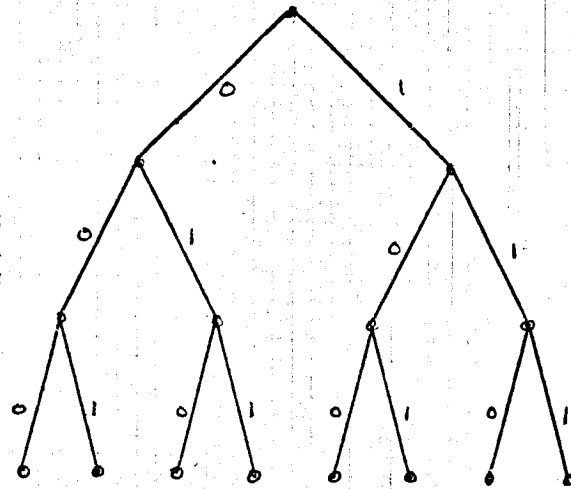
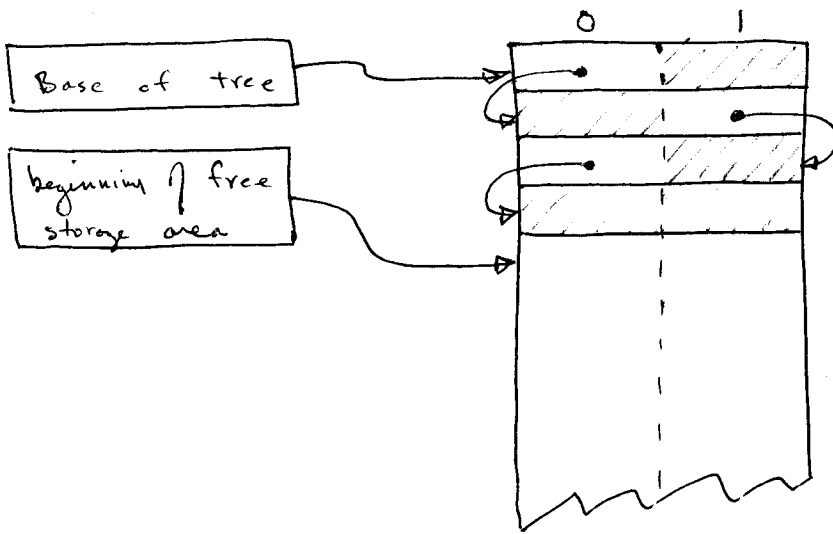


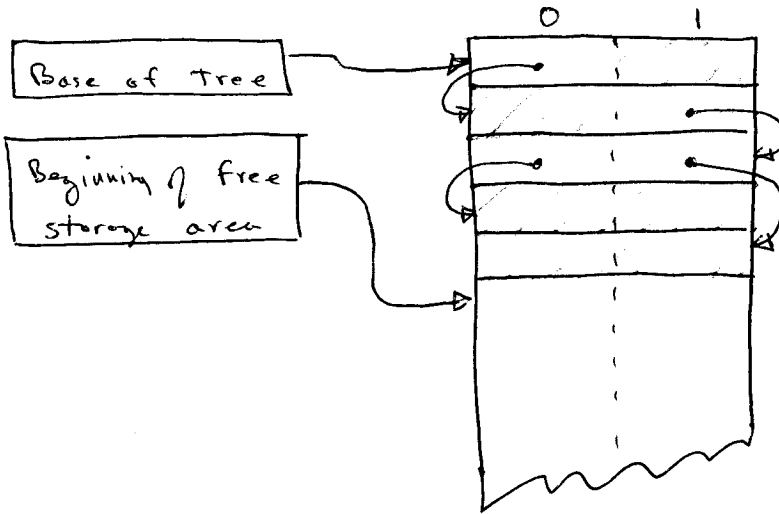
Figure 2--Addition of information item "011" to the binary tree





Corresponding list storage structure

place to  
refer of figure 1



Corresponding list storage structure

place to  
refer of  
figure 2.