# V. PROCESSING OF THE CONCEPT HIERARCHY

George Shapiro

## 1. Introduction

This report describes a system for setting up and referencing a hierarchic structure consisting of integers which represent English language concepts. These programs are incorporated into the SMART document retrieval system. The programs described here do not depend on the significance attached to the integers (hereafter called "concept numbers"), and confine themselves to the replacement of sets of original concept numbers by new ones found within the hierarchical structure. Thus, search requests producing unsuitable responses might be altered, using such a hierarchy, to produce in some sense "better" requests (i.e., requests resulting in "better" responses).

English words occurring in search requests or in documents are first replaced by concept numbers using a thesaurus lookup operation (several words often have the same number; one word may have several numbers). These concepts are arranged in a treelike structure called the hierarchy where, in general, several related items appear under one item of greater generality. Provision is made for having a given concept number appear in several places within the hierarchy by allowing any concept in the tree to "cross reference" any other. Cross referencing is unidirectional and implies no hierarchic relation. Section III of this report may be consulted for a detailed discussion of the considerations involved in generating the hierarchy.

Two main program blocks, the update and the lookup, are used to process the hierarchy which is normally stored on tape. For each update run, a set of cards is required specifying all changes, additions, and deletions to be made. The update options are described in Part 3 of the present section. It should be noted that the update programs provide diagnostic information to detect inconsistencies in the input data.

The hierarchy lookup program uses as argument a vector of concept numbers and an associated vector of frequencies. The program returns a new vector corresponding to the expansion of the input (request) vector; specifically, each item in the input vector is replaced by or added to (depending on the option chosen) its parent, children, brothers or cross references (depending again on the option chosen). These functions are detailed in Part 6 of this section.

## 2. Structure in Core

To each item in the hierarchy there correspond three consecutive locations in core as follows:

| Prefix | Decrement | Tag | Address |
|---|---|---|---|
| $S_1$ | N | | POP |
| $S_2$ | SON | | BRUB |
| | LEVEL | | REF |

where N is the concept number of the item, POP is the core address of the node corresponding to the parent of the item, SON is the core address of the node corresponding to the first member in the (filial) set of the item, BRUB is the core address of the node corresponding to the next brother in the filial set of which the item is a member, LEVEL is the index of the level (starting from a root) where this item is located in the hierarchy, and REF is the core address of the first item in a chained list of cross references to the item.

N is the only field which can in no case be equal to zero. $S_1 = +$ unless the item is marked deleted, in which case $S_1 = -$. $S_2 = +$ if the item is not the last in the filial set of which it is a member; otherwise $S_2 = -$.

If REF is nonzero, the corresponding address points to the first item in a chained list containing the cross references to the given item. Each element in this list is a 7094 machine word; the decrement part contains in each case the core address of the node in the main structure of the hierarchy corresponding to a cross reference, and the address part, if it is nonzero, points to the next item in the cross reference list; a zero address denotes the end of the list. The sign of a given word in the cross-reference list is negative if and only if the given item has been deleted from that list.

A mechanism must, of course, be available for finding the core address of a node in the hierarchy corresponding to a given concept number. To this end, a chained tree of numeric keys is provided in which each item is of the form

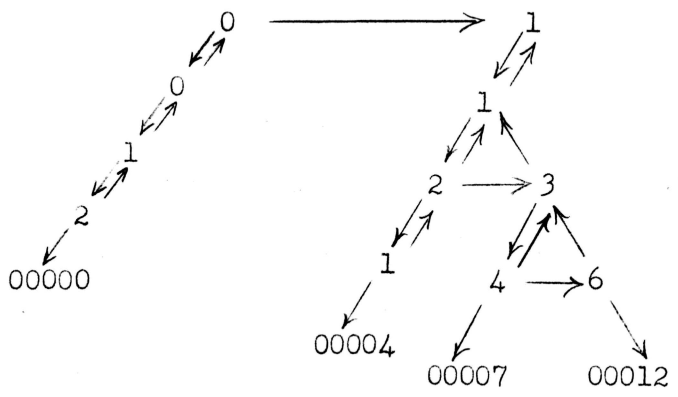
| S | 1 | 2 | Decrement | Tag | Address |
|---|---|---|---|---|---|
|  |  | 0 | R | K | B - P |

.

K represents one (octal) digit of a concept number. Since each concept number is constrained to be less than $8^4 = 4096$ and is thus represented by four octal digits, the key tree includes a total of one for each digit of the concept number. R points to the first element in the set of sons (i.e., the set of key digits one level deeper and under K) of the given word of the key tree. On the fourth level, R points to the core address in the main structure of the hierarchy of the (unique) concept number corresponding to this leaf. B - P points to the brother or parent in the key tree of the given word, depending on whether S is 0 (plus) or 1 (minus), respectively. On the fourth level, if and only if bit 1 is 1, the given item has been deleted.

Lookup in such a tree is a fairly simple procedure (performed by subroutines NFIND and FIND, which are slight modifications of each other) and updating is simpler than in simple lists, because nothing must be physically moved when an insertion is made. Flowchart 1[*] describes subroutine NFIND. N(I) is the Ith octal digit (from the right) of the concept number N which is being looked up. B represents the location of the first root of the key tree. C(J) denotes the contents of location J, and $\left[C(J)\right]_{a-b}$ denotes bits a through b of the contents of location J.

[*]The flowcharts appear in the appendix to this section.



(a) Tree Representation

| | | | |
|---|---|---|---|
| + | 20001 | 0 | 20004 |
| - | 20002 | 0 | 20000 |
| - | 20003 | 1 | 20001 |
| - | 00001 | 1 | 20002 |
| - | 20005 | 1 | 00000 |
| - | 20006 | 1 | 20004 |
| + | 20007 | 2 | 20010 |
| - | 00004 | 1 | 20006 |
| - | 20011 | 3 | 20005 |
| + | 00007 | 4 | 20012 |
| - | 00012 | 6 | 20010 |

(b) Core Image

Example of Key Tree Structure

Figure 1

It is now apparent why all the various pointers in the main structure (and the cross-reference lists) point to node addresses rather than to the concept numbers directly: in the present system, the concept number can be picked up easily, and tests and searches can be made; if, however, the pointers were used to find the concept numbers rather than their core addresses, the item would first have to be looked up in the key tree before tests or searches could be performed.

Note that no items are ever physically deleted, but rather marked and later skipped by the lookup programs. In this manner the various linked lists are never broken. An option is available to write onto an intermediate tape card images of input cards (see Part 3 of this section) for the hierarchical structure in core, and later to generate an updated hierarchy. This option is used to eliminate inefficiencies in terms of time and space arising from large numbers of deletions or reattachments.

The following illustration should help to illuminate the above discussion of the key list. Suppose the only concept numbers in the hierarchy are (in octal) 0012, 1121, 1134, and 1136, and that the nodes corresponding to these are located respectively at (octal) 00001, 00004, 00007, and 00012. The key tree would then appear as shown in Fig. 1.

## 3. Instructions for Using Subroutine GEORGE

Before giving examples of the main structure of the hierarchy, the present section specifies the necessary instructions for using subroutine

GEORGE, the supervisory program for hierarchy update and setup. The various available options are described in detail.

GEORGE assumes that the input tape containing the hierarchy is correctly positioned on A6. It is further assumed that the output tape (i.e., the tape onto which the hierarchy is to be written) is correctly positioned on B5. An end-of-file mark is written on B5 after the last word of the hierarchy.

Input cards to GEORGE have the following format:

| column: | 1 - 6, | 7 - 10, | 11, | 12 - 15, | 16, | 17 - 20, | 21, | 22 - 25, | 26, | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| | FLAG | NSON | , | NPOP | , | NR(1) | , | NR(2) | , | ... |

where FLAG is a six-character alphabetic flag and NSON,NPOP,NR(I),I= 1,...,12 are right-justified decimal integers. The list NSON,NPOP,NR(1),NR(2),... terminates at the first zero or blank field. Any nonnumeric punch (including "+" or "-") in the numeric fields causes the card to be ignored and printed off-line with a message.

Note that GEORGE, in general, reads in all the input cards and stores the processing requests in the form of so-called "request vectors" before commencing any processing (except of DELETE cards). See, however, the definition of the BYPASS card below.

If

FLAG = INSERT, NSON is inserted as the son of NPOP with NR(1),... as cross references if they are present;

FLAG = TOPMAN, NSON is inserted with no parent and with NPOP,NR(1),... as cross references if they are present;

FLAG = REFERS, NPOP,NR(1),... are inserted as cross references to NSON;

FLAG = OUTREF, NPOP,NR(1),... are deleted from the cross-reference list of NSON;

FLAG = DELETE, NSON is deleted from the hierarchy (its children, if any, are reattached with its old parent as their parent if it had a parent and are left with no parent otherwise.)

FLAG = ATTACH, NSON is removed from its current position in the hierarchy and reattached along with all its children, cross references, etc., as the child of NPOP, or if NPOP is blank or zero with no new parent; note that ATTACH cards are processed <u>after</u> INSERT cards;

FLAG = BYPASS, the hierarchy request vectors up to this point are processed, and the remaining cards in the input deck are then processed;

FLAG = FINISH, the hierarchy request vectors are processed and input ceases. If, further, NSON = 2, a listing of the hierarchy is produced. If NSON = 1, a listing is produced and the hierarchy is rewritten in "tightened" form.

If the first card of the input deck or, for that matter, the first card after a BYPASS card (though in this case the purpose is ill-defined) contains merely FLAG = TAPE, the hierarchy is read in from tape (A6) before updating commences. Otherwise, the hierarchy is written entirely from cards (in which case GEORGE does not require that A6 be mounted).

Note that the structure, as far as it has been set up, is written out on B5 whether or not GEORGE thinks that dangerous errors have been made. Thus, in some cases it might be desirable to correct the errors by updating the new (incorrect or incomplete) tape rather than by attempting to update again the tape originally on A6. This procedure should be followed only by a person who is acquainted with the hierarchy programs and is thus familiar with the structure written onto B5. (Alternatively, this tape might be listed by submitting as an input deck the two cards TAPE and FINISH0002.)

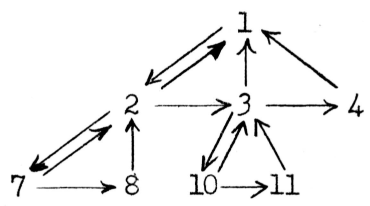
## 4. Hierarchy Setup and Update Processing Examples

The present part describes some typical examples excluding, however, any cross-reference processing which is quite straightforward.

If GEORGE is first given the following input deck

```
TOPMAN0001
INSERT0002,0001
INSERT0003,0001
INSERT0007,0002
INSERT0008,0002
INSERT0010,0003
INSERT0011,0003
INSERT0004,0001,
```

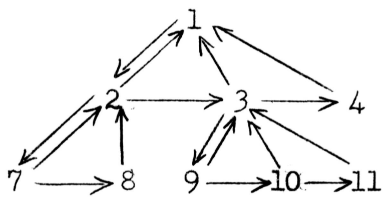
a hierarchical structure is then generated as shown in Fig. 2.



Example of Hierarchy Structure

Figure 2

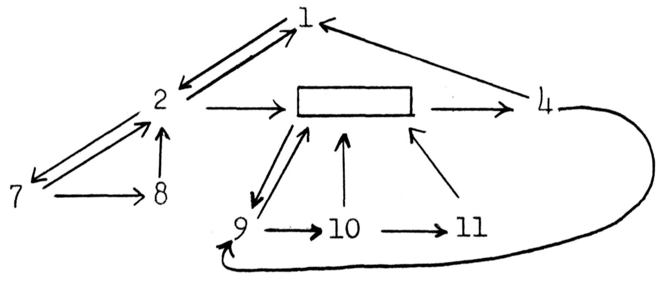
Suppose now that the card INSERT0009,0003 is processed. The procedure used by subroutine CONU to insert the new concept number in the hierarchy is then as follows: three core cells are set aside for concept number 9. Concept number 3 is then looked up and the location of its son is saved. This son pointer is then replaced by the location just set aside for 9, and the address of the old son, now the brother of node 9, is inserted in the "BRUB" location of 9. The hierarcical structure of Fig. 3 then results.



Hierarchy Structure of Fig. 2
After Processing Insertion Request

Figure 3

Assume now that the card processed next is as follows: DELETE0003. Then, the structure generated is that of Fig. 4. The box in Fig. 4 represents the deleted item which merely serves as a pointer, and 4 now

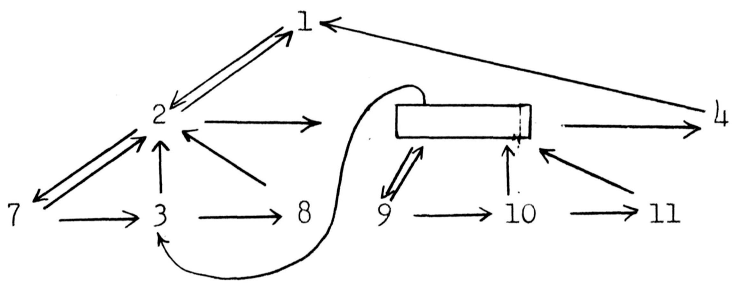


Hierarchy Structure of Fig. 3
After Processing Deletion Request

Figure 4

points to 9 as its brother (rather than being flagged $S_2$ = - as before).

Similarly, given the hierarchical structure of Fig. 2 and the input card ATTACH0003,0002, the structure of Fig. 5 is then generated. The box in Fig. 5 again represents a deleted item, this time the core



Hierarchy Structure of Fig. 3
After Processing Attach Request

Figure 5

location where 3 was previously located; this location now points to (the new) 3 as its parent so that 9, 10, and 11 still exhibit the correct parent. The location now marked 3 is a physically new node. It is the only node which must be newly created in order to effect this change.

## 5. Description of Programs for Setting Up and Updating the Hierarchy

Subroutine GEORGE (cf. Flowcharts 2 through 4 explaining various sections of this program) is the supervisory program for updating the hierarchy. It accepts and processes the control cards described in Part 3 and provides fairly comprehensive diagnostic information about illegal information and inconsistencies in the input cards. In conjunction with SPLASH, options are provided for producing a listing of the hierarchy and for rewriting the hierarchy in "tightened" form (cf. the discussion of deletions and reattachments in Parts 2 and 4).

In order to avoid restrictive conditions regarding the ordering of the input cards, GEORGE has been coded so as to read all input cards before attempting to process any; the single exception to this rule is that deletion requests are processed immediately as they are encountered. (This exception is made because it is assumed that, during a given update run, an item might be deleted and then reinserted elsewhere, but the opposite procedure should never be attempted.)

GEORGE keeps one or two integer request vectors for each type of request (see Appendix A, Table 1 for symbol-definitions of specifics). After all cards for a given run or segment of a run, as indicated by a

FINISH or BYPASS card, have been read, the processing of these vectors commences. During the processing, GEORGE calls a large number of FAP-coded programs which perform "atomic" operations on the hierarchy. The names and calling sequences of these programs appear below. Note that all of these programs return with the argument NTEST equal to zero if and only if they have been able to complete their specified functions successfully.

CONU(NPOP,NSON,NTEST) inserts NSON as the son of NPOP. If NPOP is not in the hierarchy, NTEST is positive on return. If NSON was already in the hierarchy, NTEST is negative on return.

EPON(NSON,NTEST) inserts NSON in the hierarchy with no parent. NTEST is positive on return if NSON was already in the hierarchy.

CHANGE(NSON,NPOP,NTEST) detaches NSON from its old parent (if it had one) and reattaches it as the son of NPOP. If NPOP = 0, NSON is left without a parent. On return, NTEST is positive if NSON is not in the hierarchy and is negative if NPOP is nonzero and is not in the hierarchy.

DELETE(NSON,NTEST) deletes NSON from the hierarchy; if NSON has any children they are attached as children of the parent of NSON if it had one; if it had no parent they are left without a parent. NTEST, on return, is positive if NSON was not in the hierarchy.

REFER(NAME,NCRS,NTEST) enters NCRS as a cross reference to NAME. On return, NTEST is positive if NCRS is not in the hierarchy and is negative if NAME is not in the hierarchy. No test is made to determine whether NTEST is already in the cross-reference list of NAME; if it is, and is then entered a second time, false weights will be generated in expansions during the lookup programs (cf. Part 6).

STILL(DUMMY,NCRS,NTEST) enters NCRS as a cross reference to the last item to which a reference was previously added by REFER (or by STILL). If none had been added previously, the result of calling STILL is undefined. The sequence

```
CALL REFER (NA,NB,M)
CALL STILL (DUMMY,NC,M)
```

should be used in preference to

```
CALL REFER (NA,NB,M)
CALL REFER (NA,NC,M)
```

because NA is not looked up a second time (with FIND) by STILL. All other comments applying to REFER apply also to STILL.

ROUT(NAME,NR,NTEST) removes NR from the cross-reference list of NAME. On return NTEST = 1 if NAME is not in the hierarchy; NTEST = 2 if not in the hierarchy; and NTEST is negative if NR is in the hierarchy but is not in the cross-reference list of NAME.

SROUT(DUMMY,NT,NTEST) acts with respect to ROUT in the same manner as STILL does with respect to REFER.

The algorithm used in connection with CONU may be of interest. The requests for insertions are stored in vectors NDAD,NBOY. In order to avoid specifying a given order among the input cards, the following method is employed: a pass is made through the vectors NDAD,NBOY and an attempt is made to effect each insertion in turn; those successfully handled are marked and a count is kept. GEORGE then continues to scan the vectors until the count of successes for a complete pass is zero (on each successive pass, items previously entered are ignored). If any concept numbers still have not been entered, diagnostic information is printed out. This is also done for any concepts found to be already in the hierarchy. (In this case subroutine XPOP finds the number of its parent if any, and this number is included in the diagnostic.) See Flowchart 4 for a more detailed description of this process.[γ]

After setting up the hierarchy, depending on the FINISH card, SPLASH may be called either to produce a listing or to list and also leave card images of input cards on logical tape 8. There are no DELETE, ATTACH, or OUTREF requests among these data. After control is returned to GEORGE, NTAPE is set to 8, the hierarchy is erased and reinitialized (by subroutine WENT) and processing is recommended.

When the hierarchy is eventually written on tape, subroutine TAPEU is called which in turn calls HOUT, a routine used to write the hierarchy, including also a code record giving length and location, onto tape B5. Read-in of the hierarchy is performed by subroutine HIER (called in the update link by INPUT and in the lookup link by FOREST). If read-in is to be performed a location other than that from which the hierarchy was previously read out, the various pointers in the hierarchy must be relocated

[γ]XSCHON is a Fortran integer which sets the sign bit and bit one of its argument equal to 1. XTHENF(N) is zero if and only if bit one of N is one. (The items which could not be entered because they were already included are set negative, so as to be ignored on the next pass, and are also flagged so that the appropriate diagnostic information can be written out later.)

by HIER. (It would have been possible to specify that the hierarchy should always be read into the same core locations; this would, however, have impaired the flexibility of the various component subprograms.)

It should be noted that GEORGE keeps count of a number of errors considered dangerous during setup, and if this count (called NF) is positive the program turns on Sense Light 1 before returning. The main program in the update link of SMART interprets this as a signal of failure and writes out an appropriate message.

## 6. The Hierarchy Lookup Programs

The second main set of programs includes those which expand (request) vectors of concept numbers. These routines are called (by the routine TREES in link 1) with a sequence such as CALL NAME (NUMS,NWTS,N,NF), where NUMS is an N-vector of distinct, nonzero concept numbers, and NWTS is an N-vector containing in position I the integer weight associated with NUMS(I). NF = 1 if each item in NUMS is to be <u>replaced</u> by its expansion; NF = 2 if each (expanded) item is to be <u>added</u> to the original in NUMS. Note that, if the expansion of an item is null, the item will be left in the output vector even if NF = 1.

On return, NUMS, NWTS, and N specify the same type of information where NUMS is now the vector of expansions. NUMS is compressed in the sense that if a concept number appears in the expansion of more than one item of the input vector, it still appears only once in the output vector, the corresponding weights being added. Thus, on return, NWTS(I) is equal to the sum of the input weights of all the items in whose expansion NUMS(I) occurs.

| | | | |
|---|---|---|---|
| If name is | CLIMB | each item in NUMS is replaced by or added to its | parent |
| | CHILD | | children |
| | FILIAL | | brothers |
| | CROSS | | cross references |

For example, if the hierarchy appears as in Fig. 3 and CALL FILIAL(NAMES,NWTS,N,NF) is executed with NF = 2, N = 3, and

| | |
|---|---|
| NAMES(1) = 1 | NWTS(1) = 1 |
| NAMES(2) = 9 | NWTS(2) = 1 |
| NAMES(3) = 10 | NWTS(3) = 1; |

then, on return, N = 4 and

| | |
|---|---|
| NAMES(1) = 1 | NWTS(1) = 1 |
| NAMES(2) = 9 | NWTS(2) = 2 |
| NAMES(3) = 10 | NWTS(3) = 2 |
| NAMES(4) = 11 | NWTS(4) = 2. |

As in the setup link there exist, in the hierarchy lookup link, FORTRAN-coded "bookkeeping" programs which call additional "atomic" FAP programs.

CLIMB calls a (function type) subprograms, XDADF(NSON) which returns with the concept number of the parent of NSON if NSON is in the hierarchy and has a parent, and with zero otherwise. The coding of CLIMB is indicated in Flowchart 5. NCON,NFREQ are (integer) vectors used by generated expansions and associated weights before they are condensed. MARC is a subprogram which condenses NCON,NFREQ so that each concept number occurs only once in NCON.

CHILD, FILIAL, and CROSS are similar programs, except that a FAP-coded supervisory program (cards labeled NFUDGE) arranges matters so that different FAP subroutines labeled KID, BRUB, and CRS, respectively, are called. Each of these latter is called with a sequence like CALL NAME(NH,NCON(I),K) and inserts the expansion of NH in the vector NCON in positions NCON(I) through NCON(I+K-1), where K (on return) represents the number of concept numbers in the expansion of NH. None of these routines returns NH as part of its own expansion, thereby permitting the bookkeeping program conditional on the value of NF to decide whether NH is to be included in the output vector.

CHILD, FILIAL, and CROSS all call MARC, except that CHILD does not if NF = 1 (since distinct parents have disjoint sets of sons).

Flowchart 6 describes the coding of CHILD, FILIAL, or CROSS; NCON, NFREQ here serve the same function as in CLIMB.

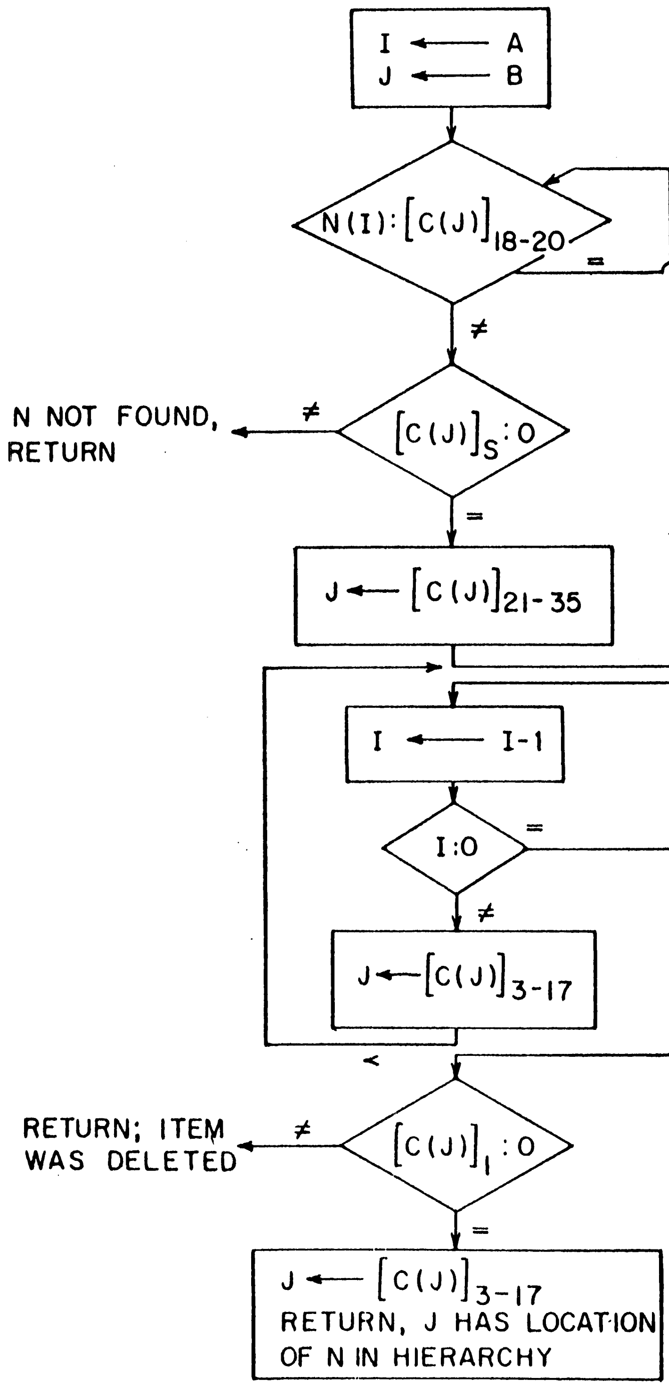
As an example of (a fairly simple) lookup procedure, Flowchart 7 describes the coding of XDAD. The argument is found by the subroutine in the accumulator (Acc), and the value is returned there. Again, C(X) refers to the contents of location X and $Y_{a-b}$ refers to bits a through b of Y.

## APPENDIX A

| Symbol | Definition |
|---|---|
| FLAG | the six-character alphabetic flag read in from the input card |
| NAME | a vector containing in position N the element to which the Nth item in the vector NCRS must be added as a cross reference |
| NAT | the length of the vectors NCH and NEW |
| NBOY | a vector containing in the Nth position the Nth item to be attempted to be entered into the hierarchy (it will be added as the son of the Nth item in NDAD) |
| NCH | a vector containing in the Nth position the Nth item to be moved (cf. ATTACH option) |
| NCRS | see definition of NAME |
| NDAD | see definition of NBOY |
| NDL | the length of the vector NOUT |
| NEW | a vector containing in the Nth position the item to which the Nth item in NCH is to be moved |
| NF | the number of so-called "dangerous" errors |
| NINS | the length of the vector NBOY |
| NOR | the length of the vector NROUT |
| NOUT | a vector which contains in the Nth position the Nth item to be inserted without a parent (cf. TOPMAN option) |
| NPL | a vector which contains in the Nth position the item from which the Nth item in the vector NROUT should be deleted as a cross reference |
| NPOP | the second integer read in from the input card |
| NR | a vector of length twelve, the third through fourteenth integers to be read in from the input card |
| NREF | the length of NCRS |
| NROUT | see definition of NPL |
| NSON | the first integer read in from the input card |
| NTAPE | the number of the input tape (either logical 5 or 8) |
| NTEST | a variable used by the various FAP routines to indicate to GEORGE whether or not they have been able to perform their intended task |

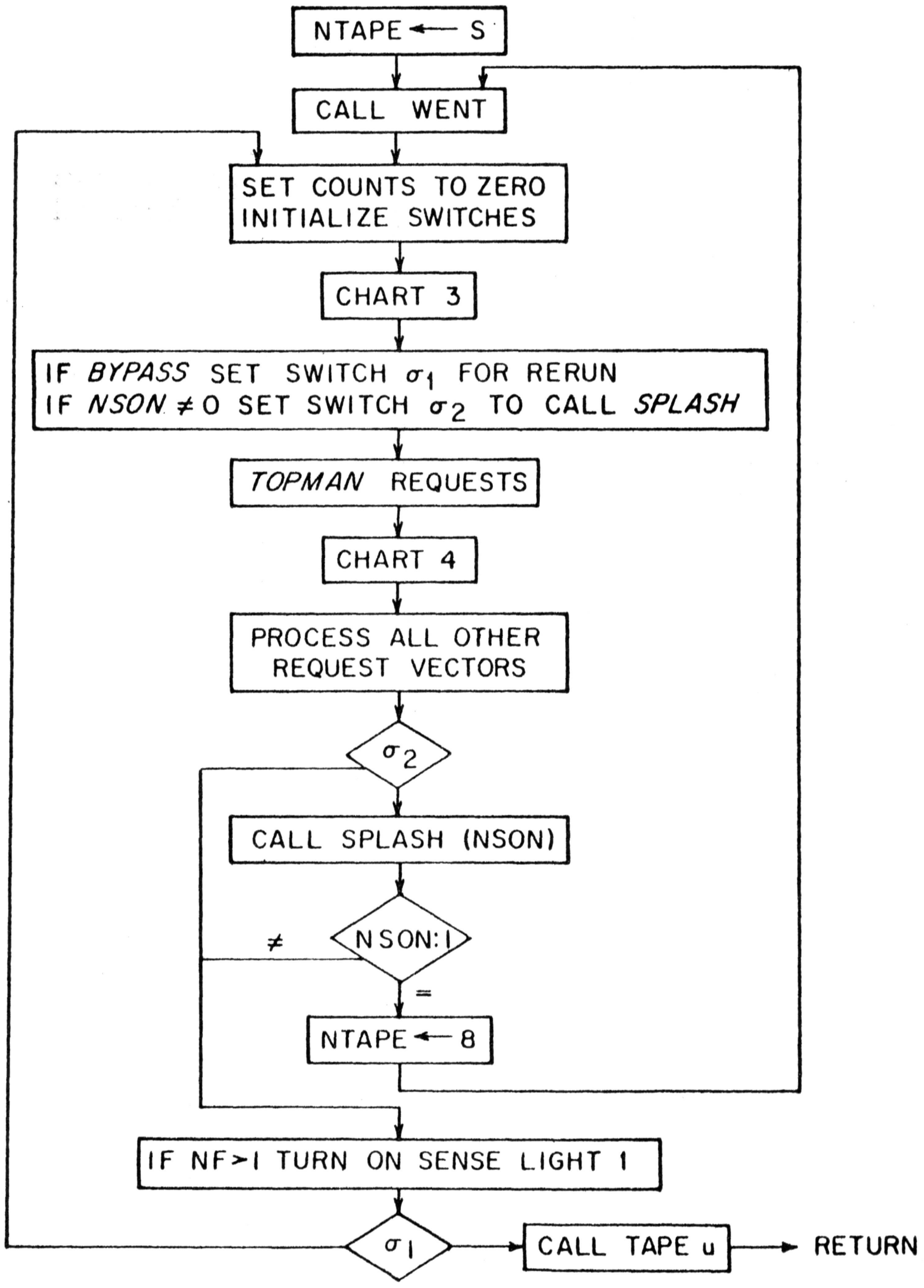
Definition of Symbols in Subroutine GEORGE

TABLE 1



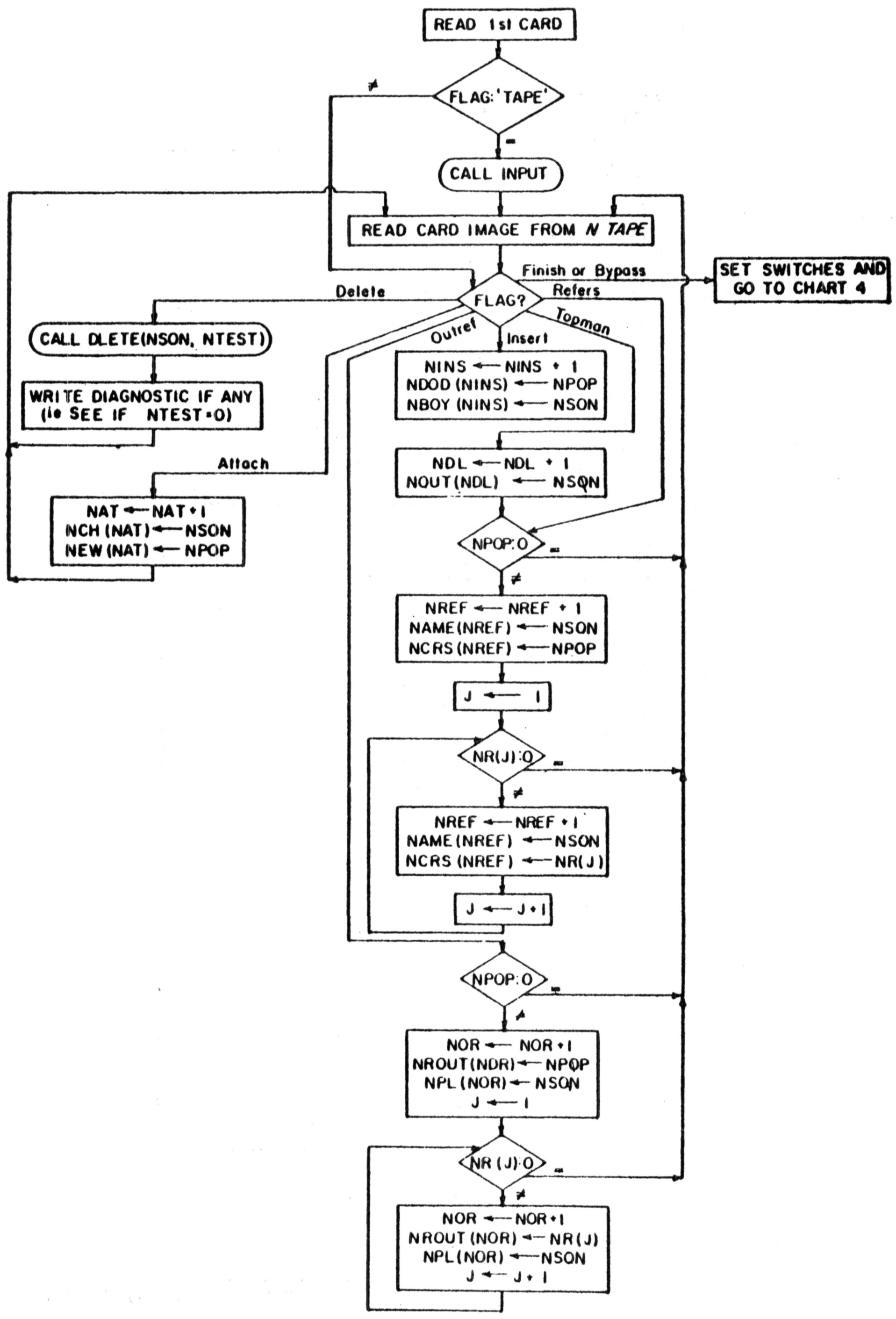
Subroutine NFIND, Looks up Concept Number (N) in Key Tree and Returns Location of Hierarchy Node

Flowchart 1



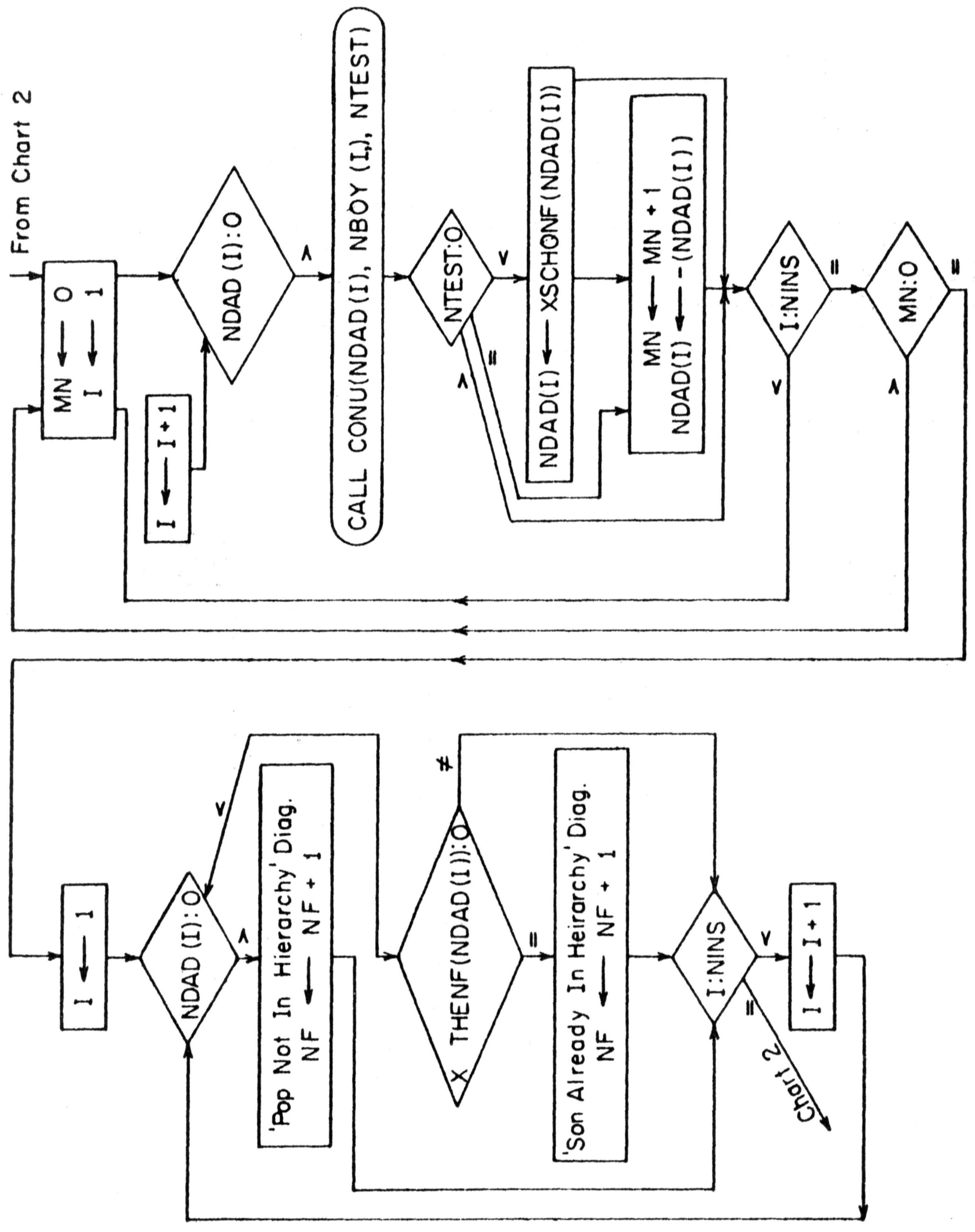
Block Diagram of Subroutine GEORGE; (Processing of Request Vectors Other Than Insertion Requests is Straightforward and is, Therefore Not Included)

Flowchart 2



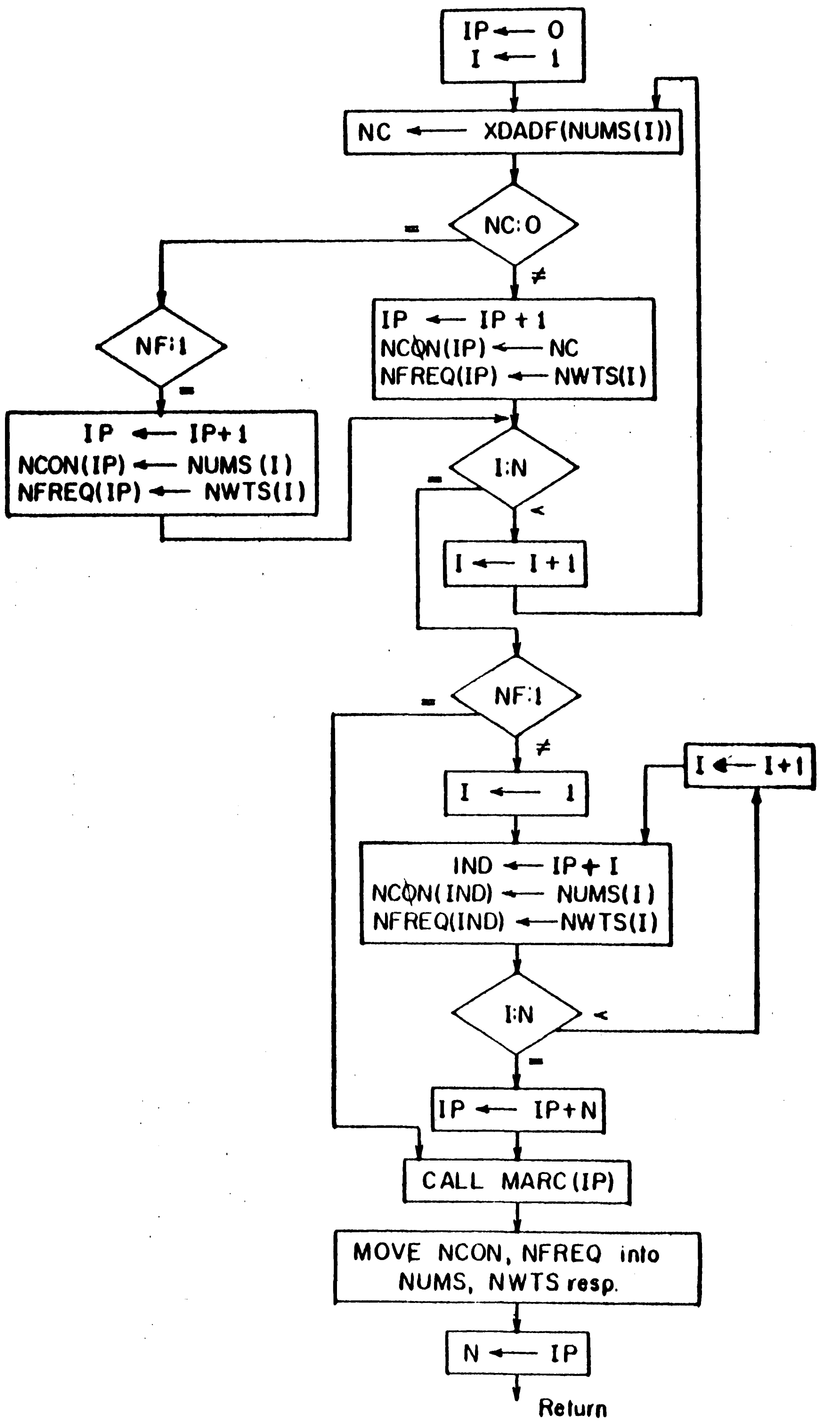
Setup of Request Vectors in Hierarchy Update Program

Flowchart 3



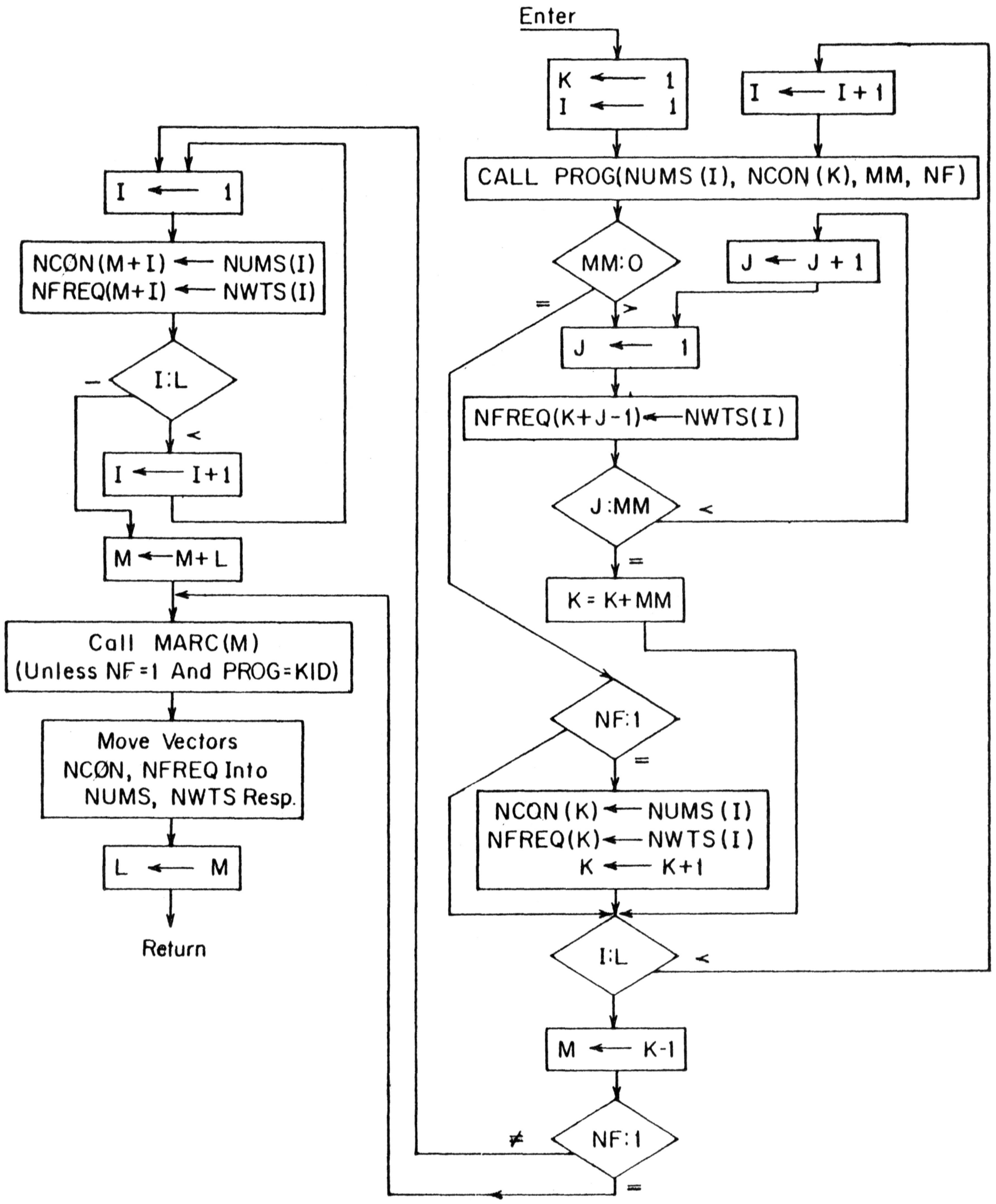
Processing of Insertion Requests by GEORGE

Flowchart 4



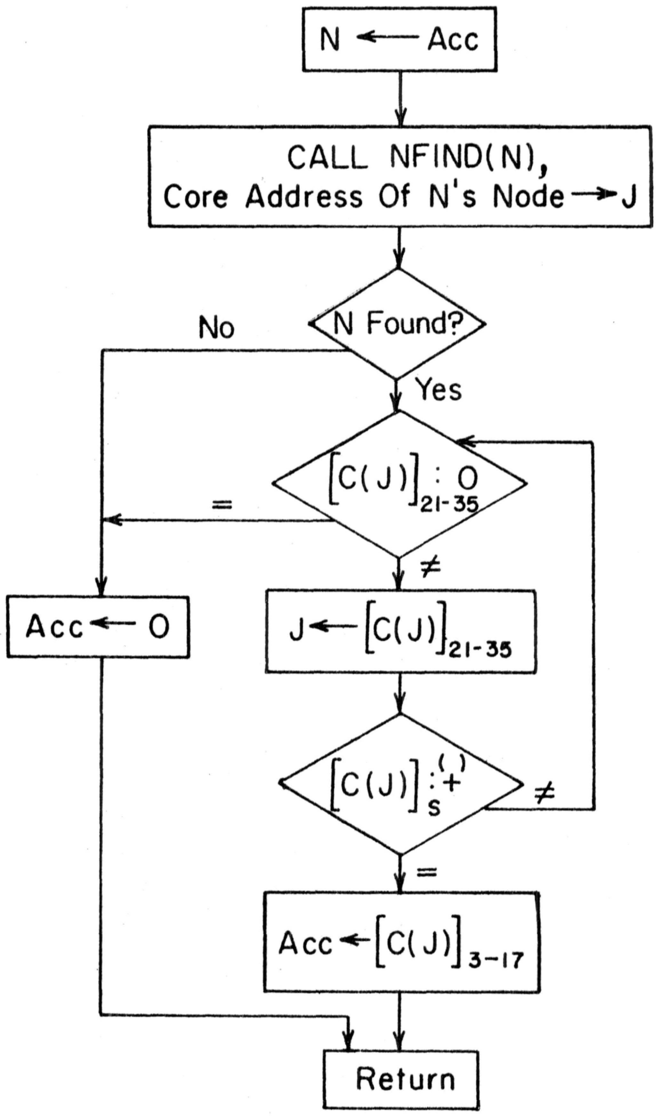
Subroutine CLIMB (NUMS,NWTS,N,NF)

Flowchart 5



Flowchart of CHILD, FILIAL or CROSS;
PROG is KID, BRUB or CRS, Respectively

Flowchart 6



N ← Acc
CALL NFIND(N), Core Address Of N's Node → J
N Found?
No
Yes
$[C(J)]_{21-35} : 0$
=
≠
Acc ← 0
$J \leftarrow [C(J)]_{21-35}$
$[C(J)]_S : (+)$
≠
=
$Acc \leftarrow [C(J)]_{3-17}$
Return

Subroutine XDAD

Flowchart 7