Paste

Automatic Data Processing

Chapters 6, 7

Chapter 6 A Programming Language

Chapter 7 Programming

Kenneth E. Iverson

© 1960

TABLE OF CONTENTS

CHAPTER 6

CHAPTER 6

A programming language

6.1 Introduction

The systematic analysis and design of complex algorithms must be based upon a suitable means for their description. Since a precise description of an algorithm is called a program, a notational scheme for the description of information processes is called a programming language. A programming language should be concise, precise, consistent over a wide area of application, mnemonic, and economical of symbols; it should exhibit clearly the constraints on the sequence in which operations are performed; and it should permit the description of a process to be independent of the particular representation chosen for the data.

Existing languages prove unsuitable for a variety of reasons. Computer coding specifies sequence constraints adequately and is also comprehensive, since the logical functions provided by the branch instructions can, in principle, be employed to synthesize any finite algorithm. However, the set of basic operations provided are not, in general, directly suited to the execution of commonly-needed processes, and the numeric symbols used for variables have little mnemonic value. Moreover, the description provided by computer coding depends directly upon the particular representation chosen for the data, and it therefore cannot serve as a description of the algorithm per se.

Ordinary English lacks both precision and conciseness. The widely used Goldstine-von Neumann[1] flowcharting provides the conciseness essential to an over-all view of the process, only at the cost of suppressing essential detail. The so-called pseudo-English used as a basis for certain automatic programming systems suffers from the same defect. Moreover, the potential mnemonic advantage in substituting familiar English words and phrases for less familiar but more compact mathematical symbols fails to materialize because of the obvious but unwonted precision required in their use.

Virtually all of the concepts and operations needed in a programming language have already been defined and developed in one or another branch of mathematics. Therefore, much use can and will be made of existing notations. However, since most notations are specialized to a narrow field of discourse, a consistent unification must be provided. For example, separate and conflicting notations have been developed for the treatment of sets, logical variables, vectors, matrices, and trees, all of which may, in the broad universe of discourse of data processing, occur in a single algorithm.

6.2 Programs

A <u>statement</u> is the specification of some quantity or quantities in terms of some finite operation upon specified operands. Specification is symbolized by an arrow directed

toward the specified quantity. Thus "y is specified by sin x" is a statement denoted by

$$y \leftarrow \sin x.$$

A set of statements together with a specified order of execution constitutes a program. The program is finite if the number of executions is finite. The results of the program are some subset of the quantities specified by the program. The sequence or order of execution will be defined by the order of listing and otherwise by arrows connecting any statement to its successor. A cyclic sequence of statements is called a loop. Thus Program 6-1 is a program of two statements defining the result v as the (approximate) area of a circle of radius x, whereas Program 6-2 is an infinite program in which the quantity z is specified as $(2y)^n$ on the nth execution of the two-statement loop. Statements will be numbered at the left for reference.

A number of similar programs may be subsumed under a single more general program as follows. At certain branch points in the program a finite number of alternative statements are specified as possible successors. One of these successors is chosen according to criteria determined in the statement or statements preceding the branch point. These criteria are usually stated as a comparison or test of a specified relation between a specified pair of quantities. A branch is denoted by a set of arrows leading to each of the alternative successors, with each arrow labeled by the comparison condition under which the corresponding successor is chosen. The quantities compared are separated by a colon in the

statement at the branch point, and a labeled branch is followed if and only if the relation indicated by the label holds when substituted for the colon. The conditions on the branches of a properly defined program must be disjoint and exhaustive.

Program 6-3 illustrates the use of a branch point. Statement $\alpha 5$ is a comparison which determines the branch to statements $\beta 1$, $\delta 1$, or $\gamma 1$, according as $z > n$, $z = n$, or $z < n$, respectively. The program represents a crude but effective process for determining $x = z^{2/3}$ for any positive perfect cube n.

Program 6-4 shows the preceding program reorganized into a compact linear array. Two further conventions on the labeling of branch points are used. The listed successor of a branch statement is selected if none of the labeled conditions is met. Thus, statement 5 follows statement 5 if neither of the arrows (to exit or to statement 8) are followed, i.e., if $z \leq n$. Moreover, any unlabeled arrow is always followed; e.g., statement 7 is invariably followed by statement 3, never by statement 8. A program begins at a point indicated by an entry arrow (e.g., step 1), and ends at a point indicated by an exit arrow (e.g., step 5).

Program 6-5 shows an equivalent computer code, the numbers on the left referring to the corresponding steps of Program 6-4. Comparison shows that the former starts at statement 3, skipping statements 1 and 2. This difference occurs because the variables y and k are specified in the computer program by the initial contents of the corresponding registers, and need not be specified by further explicit operations within the program.

A process which is repeated a number of times is said to be iterated, and a process (such as Program 6-4) which includes one or more iterated subprocesses is said to be iterative. A parameter which determines the number of consecutive executions of an iterated process is called a counter. A parameter which designates a particular element of a structured operand such as a vector or matrix is called an index. A simple use of an index occurs in the summation

$$s \longleftarrow \sum_{i=1}^{n} x_i .$$

Program 6-6 shows a particular realization of the process. In this case the index serves also as counter. The practice of beginning with the index equal to n and decrementing to zero allows the comparison to be made with zero and also obviates the need to specify an auxiliary index if, as assumed in Program 6-6, the initial value of n need not be preserved.

A more complex use of indices is shown in Program 6-7, which describes the matrix multiplication $C \longleftarrow AB$ defined as

$$c_j^i = \sum_{k=1}^{\nu(A)} a_k^i \times b_j^k, \qquad i \in \{(1,\mu(A))\}, \qquad j \in \{(1,\nu(B))\}.$$

Each of the indices i, j, and k serves also as a counter.

FP

Program 6-7. Step 1 specifies C as a matrix of zeros, steps 2-4 initialize the indices, and the loop 5-7 continues to add successive products to the partial sum until k reaches zero.

(FP)

When this occurs, the process continues through step 8 to decrement j and to repeat the entire summation for the new value of j, providing that it is not zero. If j is zero, the branch to step 10 decrements i and the entire process over j and k is repeated from j = $\nu(5)$, providing that i is not zero. If i is zero the process is complete, as indicated by the exit arrow.

The programming language will be designed to admit both 0-origin and 1-origin indexing (Sec. 1.9). Examples used in this chapter will, however, be stated in the more familiar 1-origin indexing.

In all examples used in this chapter, emphasis will be placed upon clarity of description of the process, and considerations of efficient execution by a computer or class of computers will be subordinated. These considerations can often be introduced later by relatively routine modifications of the program. For example, since the execution of a computer operation involving an indexed variable is often more costly than the corresponding operation upon a nonindexed variable, the substitution of a variable s for the variable $\binom{i}{j}$ in the first statement of the loop 5-7 would accelerate the execution of the loop. The variable s would, of course, be initialized before each entry to the loop (incidentally obviating step 1) and would be used to specify $\binom{i}{j}$ at each termination.

Since zero often occurs in comparisons, it is convenient to omit it. Thus, if a variable stands alone at a branch point, comparison with zero is implied. Moreover, since a comparison on an index or counter frequently occurs immediately after it is modified,

a branch at the point of modification will denote branching upon comparison of the indicated index with zero; the comparison occurring after modification. Designing programs to execute decisions immediately after modification of the controlling variable results in efficient execution as well as notational elegance, since the variable must be present in a central register for both operations.

Since the sequence of execution of statements is indicated by connecting arrows as well as by the order of listing, the latter can be chosen arbitrarily. This is illustrated by Programs 6-3 and 6-4 which describe functionally identical programs. Certain principles of ordering may yield advantages such as clarity or simplicity of the pattern of connections. Even though the advantages offered by a particular organizing principle are not particularly marked, the uniformity resulting from its consistent application will itself be a boon. The scheme here adopted (for reasons set forth in Sec. 7.1) is called the method of leading decisions. It consists in placing the decision on each parameter as early in the program as practicable, normally just before the operations indexed by the parameter. Program 6-8 shows such a reorganization of Program 6-7.

Although the labeled arrow representation of program branches provides a complete and mnemonic description, it is deficient in the following respects: (1) a routine translation to another language (such as a computer code) would require the

breaking of arrows, and (2) it does not permit programmed modification of the branches.

The following alternative form of a branch statement will therefore be used as well:

$$x : y, \quad x \longrightarrow s.$$

This denotes a branch to statement number s_i of the program if the relation $x\ r_i\ y$ holds, $i \in \{(1, v(x))\}$. The parameters x and y may themselves be defined and redefined in other parts of the program. The null element will be used to denote the relation which complements the remaining relations r_i; in particular, $(\circ) \longrightarrow (s)$, or simply $\longrightarrow s$ will denote an unconditional branch to statement s. Program 6-9 shows the use of these conventions in a reformulation of Program 6-8.

One statement in a program can be modified by another statement which changes certain of its parameters such as indices and selection vectors. More general changes in statements can be effected by considering the program itself as a vector p whose components are the individual serially numbered statements. All of the operations defined upon general vectors can then be brought to bear upon the statements themselves. For example, the jth statement can be respecified by the ith through the occurrence of the statement $p_j \longleftarrow i$.

6.3 Classes of operands

The power of any mathematical notation rests largely on the

use of symbols to represent general quantities which in given instances are further specified by other quantities. Thus Program 6-4 represents a general process which determines $x = n^{2/3}$ for any suitable value of n. In a specific case, say n = 27, the quantity x is specified as the number 9.

Each operand occurring in a meaningful process must be specified ultimately in terms of commonly accepted concepts. The symbols representing such accepted concepts will be called literals. Examples of literals are the integers, the characters of the various alphabets, punctuation marks, and miscellaneous symbols such as $ and %. The literals occurring in program 6-4 are 0, 1, and 2.

It is important to distinguish clearly between general symbols and literals. In ordinary algebra this presents little difficulty, since the only literals occurring are the integers and the decimal point, and each general symbol employed includes an alphabetic character. In describing more general processes, however, alphabetic literals (such as proper names and mnemonic symbols) also appear. Moreover, in a computer code, numeric symbols (register addresses) are used to represent the variables, as illustrated by the right-hand version of Program 6-5.

In general then, alphabetic literals, alphabetic variables, numeric literals, and numeric variables may all appear in a complex process, and it is imperative to distinguish among them. The symbols used for literals will be Roman letters (enclosed in quotes when appearing in text) and standard numerals. The symbols used for variables will be italic letters, italic numerals, and boldface

letters as detailed in Table 6-1. Miscellaneous signs and symbols when used as literals will be enclosed in quotes in both programs and text.

The use of distinct classes of symbols for distinct classes of operands not only aids in the visual interpretation of a program, but also permits significant reduction in the number of distinct operation symbols required, since analogous operations upon different types of quantities may be represented by the same symbol (e.g., $A \wedge B$ for a product of sets and $a \wedge b$ for a product of logical vectors). Potential ambiguity is resolved by the distinct operand symbols. Special operands (such as the unit vectors ϵ^i defined in Chapter 1) or functions will be denoted by Greek letters in the appropriate type face.

In any determinate process, each operand must be specified ultimately in terms of literals. In Program 6-4, for example, the quantity k is specified in terms of known arithmetic operations (multiplication and division) involving the literals 1 and 2. The quantity n, on the other hand, is not determined within the process, and must presumably be specified within some larger process which includes Program 6-4. Such a quantity is referred to as an <u>argument</u> of the process.

6.4 Successor operations

The major advantages accruing from the use of the set operations <u>successor</u> and <u>predecessor</u> (defined in Sec. 1.7 and

summarized in Sec. V of Appendix A) are two: they allow the description of the process and the specification of the representation of the data to be completely divorced, and they eliminate the recurrence of the literal 1 in the incrementation and decrementation of counters and indices. Their use will be illustrated by a single example.

Example 6-1. Consider a process to determine the length of the longest run in any one suit in a hand of thirteen playing cards. A run is defined as a set of cards whose ranks form an interval in the set

$$D \equiv \{2, 3, \ldots, 10, (j), (q), (k), (a)\},$$

where the literals denote deuce, trey, four, $\cdots$, king, and ace. The length of a run is the number of elements it contains.

The hand may be represented by the matrix H of dimension 2×13 whose column vectors H_j each represent one card, H_j^1 and H_j^2 representing respectively the suit and rank of the jth card. Hence,

$$H_j^2 \in D,$$

and

$$H_j^1 \in S \equiv \{(c), (d), (h), (s)\},$$

where the literals denote clubs, diamonds, hearts, and spades, respectively. If

$$H = \begin{pmatrix} (d) & (c) & (h) & (c) & (h) & (s) & (h) & (d) & (c) & (h) & (c) & (h) & (d) \\ (a) & 6 & (k) & (q) & 4 & 3 & 5 & (k) & 8 & 2 & (j) & 9 & 2 \end{pmatrix},$$

then the longest runs in clubs, diamonds, hearts, and spades are 1,3,2, and 1, respectively, and the longest run in any one suit is 3.

The process described by Program 6-10 examines each card H_i as the base (first element) of a run and determines the corresponding run length p. The final result q is determined as the maximum over the values of p.

(FP)

Program 6-10. The maximization over p is performed by steps 4 and 5. The index j determines the card currently examined as a possible successor in any partial run. The variable d is the denomination of the largest element obtained in the partial run. It is initialized by H_i^2 on step 7, and redefined in step 15 by its successor in D if this successor is equal to H_j^2. Each time it is redefined, the count p of the length of the partial run is incremented on step 14. Whenever d is redefined, j is also redefined to the value $\uparrow\nu(H) = 14$ so that the entire hand is scanned for the new successor. A card H_j for which the suit H_j^1 does not agree with H_i^1 is rejected by the decision on step 11. The comparison with the terminal element D_{-1} (step 12) is needed to prevent further continuation of the run when d is an ace.

The incrementation and decrementation of quantities such as the index j which range over a subset of the integers is indicated in Program 6-10 by the successor and predecessor operations $\uparrow j$ and $\downarrow j$. This procedure virtually eliminates the explicit occurrence of literals. The use of the symbol $\uparrow\nu(1)$ (in steps 1 and 9) in lieu of the equivalent but less meaningful literal "14" facilitates interpretation of the program.

6.5 The definition of symbols

Program 6-10 provides a precise and organized description of the required process. It is also important to organize the presentation of auxiliary data such as the ordering of the sets involved, and the significance of the chosen symbols. Table 6-2 presents the data relevant to Program 6-10.

The first column shows the assigned symbols and the second describes their range and dimension. If a variable possesses further structure, its components may themselves be described in further rows. The meaning of the variable is described in column three.

Although additional variables appear in the program, each is specified within the process and need not be defined in the table. In describing complex processes it is, however, often helpful to include these derived variables in an extended or auxiliary table. The distinction between predefined and process-defined variables should, however, be maintained.

In devising a computer program corresponding to Program 6-10, it will be found necessary to choose a specific representation for the variables involved before describing operations such as selecting a successor in a given set. This choice may be displayed in an extension of Table 6-2. For example, the denominations 2, 3, ···, "(k)", and "(a)" might be denoted by 1, 2, ···, 12, and 13. The successor operation then reduces to simple addition of unity. A particular choice of representation

could also be used to obviate the general successor operations in the programming notation, but this would bind the entire description to the chosen representation.

6.6 Reference table of notation

Appendix A summarizes the notation developed in this chapter. Although intended primarily for reference, it supplements the text in several ways. It frequently provides a more concise alternative definition of an operation discussed in the text, and also contains numerous important but easily grasped extensions not treated explicitly in the text. By grouping the operations into related classes it displays their family relationships. Finally, by using the symbol ϕ (I superimposed upon O) to denote the index origin in use, each operation in Appendix A is expressed in terms of both 0-origin and 1-origin index systems. The exposition in the text is limited almost entirely to 1-origin indexing.

Many of the operations defined apply only to certain restricted classes of operands (e.g., arithmetic vector operations do not apply to nonnumerical vectors), and these restrictions will be indicated by adopting the conventions shown in Sec. I of the appendix. The classes of operands appearing in Sec. I are defined as follows. A vector whose component sets X^k are all equal to a given set Y is called <u>homogeneous</u>, and some homogeneous vectors may be further classified as <u>numerical</u>, <u>integral</u>, or <u>logical</u>, according as Y is a finite subset of the real numbers, is a

finite subset of the integers, or is the set $\{0,1\}$, respectively. Each of these classes is a subclass of the classes preceding it, and each operation defined upon a class also applies to each subclass. For example, arithmetic operations defined upon numerical vectors also apply to the subclasses of logical and integral vectors. A special "mapping" operand, which may assume the value $\bullet$ as well as integral values, will be denoted by m, $\mathbb{m}$, or $\mathbb{M}$, or $\underset{\sim}{M}$.

The symbols used in each definition of Appendix A are those of the most general class of suitable operands. Thus the logical operations of Sec. VI are mainly restricted to the logical operands u, v, w, $\mathbb{u}$, $\mathbb{v}$, $\mathbb{w}$, U, V, W, $\underset{\sim}{U}$, $\underset{\sim}{V}$, $\underset{\sim}{W}$, but the logical reduction (Defs. 107-110) apply to arbitrary scalars, vectors, and matrices. The operands must satisfy certain further <u>compatibility</u> conditions (primarily concerning their relative dimensions) which are listed in the final column of Appendix A.

A concise programming language must incorporate <u>families</u> of operations such that the members of a given family are related in a systematic manner. Each family will be denoted by a specific operation symbol, and the particular member of the family will be designated by an associated <u>controlling parameter</u> (scalar, vector, matrix, or tree) which immediately precedes the main operation symbol. The operand is placed immediately after the main operation symbol. For example, the operation $k\uparrow x$ (the kth successor of x) may be viewed as the kth member of the set of successor operations denoted by the symbol $\uparrow$.

In interpreting a compound operation such as $k^{\uparrow}(j\downarrow x)$ it is important to recognize that the operation symbol and its associated controlling parameter together represent an indivisible operation and must not be separated. It would, for example, be incorrect to assume that $j^{\uparrow}(k\downarrow x)$ were equivalent to $k^{\uparrow}(j\downarrow x)$, although it can be shown that the complete operations $j^{\uparrow}$ and $k^{\downarrow}$ do commute, that is, $k^{\uparrow}(j\downarrow x) = j\downarrow(k\uparrow x)$.

In order to reduce the need for parentheses it will be assumed that compound statements are, except for intervening parentheses, executed from right to left. Thus, $k\uparrow j\downarrow x$ is equivalent to $k\uparrow(j\downarrow x)$, not to $(k\uparrow j)\downarrow x$.

Operations involving a single operand and no controlling parameter (such as $\lceil x \rceil$, or $\lfloor x \rfloor$) will be denoted by a pair of operation symbols which enclose the operand. Operations involving two operands and a controlling parameter (such as the mesh operation $\backslash a,u,b\backslash$) will be denoted by a pair of operation symbols enclosing the entire set of variables, and the controlling parameter will appear between the two operands. In these cases the operation symbols themselves serve as grouping symbols.

Certain operators serve a purpose analogous to that served by certain special logical vectors when used as controlling parameters. For mnemonic reasons, such operators will be denoted by the same Greek character used for the analogous vectors; $\omega^{\downarrow}$, for example, is a prefix vector, and $\omega(v)$ denotes the prefix weight of the logical vector v.

6.7 Selection operations

Algorithms, particularly those suited to automatic execution, tend to require the execution of the same operations upon each of a group of operands. It is therefore useful to generalize operations defined upon single operands to corresponding operations defined upon each element of a structured array of operands. The structured arrays employed are primarily the sets, vectors, matrices, and trees introduced in Secs. 1.7 and 1.8. The type of generalization employed is illustrated by the extension of the logical operations and, or, and not, to logical vectors in Sec. 1.8. The execution of a process upon a structured operand can (and in the use of a computer frequently must), however, be executed as an explicit repetition of a given operation upon successive elements of the array.

The effective use of structured operands depends not only upon generalized operations but also upon the ability to specify and select certain elements or groups of elements. The selection of single elements can, for example, be made by specifying indices as in the expressions A_i, v_i, M^i, A_j, M^i_j, and $(\mathcal{T}_i)$. Since selection is a binary operation (i.e., to select or not to select), more general selection is conveniently specified by a logical vector, each unit component indicating selection of the corresponding component of the operand. The logical vectors themselves may be specified in terms of the logical operations defined in Sec. 1.8 and summarized in Sec. VI of Appendix A.

The selection operation (Def. 135) defined upon an ordered set A is denoted by the statement

$$C \leftarrow u/A,$$

and is defined as follows: the set C contains only those elements A_i for which $u_i = 1$, and the set C is ordered on A. In other words, C is obtained by suppressing those elements A_j for which $u_j = 0$. The logical vector u is said to compress the set A. The vector u and the set A must be compatible, i.e., $\nu(u) = \nu(A)$. For example, if $\nu(A) = 5$ and $u = (1,0,1,0,1)$, then $u/A \equiv [A_1, A_3, A_5]$.

The weight of a vector x is denoted by $\sigma(x)$ and defined (Def. 64) as the sum of the components. In the case of a logical vector u, the weight is also the number of unit components and clearly $\sigma(u) + \sigma(\bar{u}) = \nu(u)$. Moreover, $\nu(u/A) = \sigma(u)$.

The compress operation is extended to vectors and matrices as follows. The vector compression u/x selects components of x exactly as the set compression u/A selects elements of A. A matrix A may be compressed in two distinct ways. Row compression, denoted by u/A, compresses each row vector A^i of the matrix A to form a new matrix of dimension $\mu(A) \times \sigma(u)$. Column compression, denoted by u/A compresses each column vector A_i to form a matrix of dimension $\sigma(u) \times \nu(A)$. Compatibility conditions are $\nu(u) = \nu(A)$ for row compression, and $\nu(u) = \mu(A)$ for column compression. For example, if A is an arbitrary 3×4 matrix, $u = (0,1,0,1)$, and $v = (1,0,1)$, then

$$A = \begin{pmatrix} A^1_1 & A^1_2 & A^1_3 & A^1_4 \\ A^2_1 & A^2_2 & A^2_3 & A^2_4 \\ A^3_1 & A^3_2 & A^3_3 & A^3_4 \end{pmatrix},$$

$$u/A = \begin{pmatrix} A^1_2 & A^1_4 \\ A^2_2 & A^2_4 \\ A^3_2 & A^3_4 \end{pmatrix},$$

$$u//A = \begin{pmatrix} A^1_1 & A^1_2 & A^1_3 & A^1_4 \\ A^3_1 & A^3_2 & A^3_3 & A^3_4 \end{pmatrix},$$

and

$$u/v//A = v//u/A = \begin{pmatrix} A^1_2 & A^1_4 \\ A^3_2 & A^3_4 \end{pmatrix}.$$

It is clear that row compression suppresses columns corresponding to zeros of the logical vector and that column compression suppresses rows. This illustrates the type of confusion which can arise in naming operations upon matrices which are obtained by generalizing operations upon vectors. The following nomenclature will be used consistently – an operation is called a row operation if the underlying operation from which it is generalized is applied to the row vectors of the matrix, and a column operation if it is applied to columns.

Example 6-2. A bank which assigns account numbers from a solid set of integers wishes to make a quarterly review of accounts to produce the following four lists:

(1) the name, account number, and balance for each account with a balance less than two dollars;

(2) the name, account number, and balance for each account with a negative balance exceeding one hundred dollars;

(3) the name and account number of each account with a balance exceeding one thousand dollars; and

(4) all unassigned account numbers.

The ledger may be described by a matrix

$$L = (L_1, L_2, L_3) = \begin{bmatrix} \cdot^1 \\ \vdots \\ \cdot^m \end{bmatrix}$$

with column vectors L_1, L_2, and L_3 representing names, account numbers, and balances, respectively, and with row vectors $\cdot^1, \cdot^2, \ldots, \cdot^m$ representing individual accounts. An unassigned account number is identified by the word "none" in the name position. The four output lists will be denoted by the matrices R_1, R_2, R_3 and R_4, respectively. They can be produced by Program 6-11.

Program 6-11. Since L_3 is the vector of balances and $2\cdot$ is a compatible vector each of whose components equals two, the relational statement $(L_3 < 2\cdot)$ defines (Def. 108) a logical

vector having unit components corresponding to those accounts to be included in the list P. Consequently, the column compression of step 1 selects the appropriate rows of L to define P. Step 2 is similar, but step 3 incorporates an additional row compression by the compatible prefix vector $\epsilon^2 = (1,1,0)$ (Def. 127) to select columns one and two of L. Step 4 represents the comparison of the name (in column l_1) with the literal "none", the selection of each row which shows agreement, and the suppression of all columns but the second.

The expression "noneϵ" occurring in step 4 of Program 6-11 illustrates the use of scalar replacement (a useful generalization of the multiplication of a vector by a numeric scalar), which is defined (Def. 112) as follows. For any logical vector u and arbitrary quantity a, the statement $a \leftarrow\!\!- su$ specifies the vector v obtained from u by replacing each unit component by a and leaving the zero components unchanged.

If B is any subset of A which is ordered on A, then there exists a logical vector u such that $u/A \equiv B$. More generally, if C is any subset of A, there exists a vector u such that $u/A = C$, i.e., such that u/A is a permutation of C. Consequently, any subset C can, except for ordering, be represented with respect to A by a logical vector u which is called the subset vector of C on A. For example, if $A = \{(c), (d), (h), (s)\}$, and $C = \{(s), (c)\}$, then $\epsilon_A^C = (1,0,0,1)$, and $\epsilon_A^C/A \equiv \{(c), (s)\} \not\equiv C$, although $\epsilon_A^C/A = C$. If A is the set of integers $\{(1, \nu(A))\}$, and if the value of $\nu(A)$ is clear

from context, then A may be elided. The notation for the unit vectors (Def. 125) is consistent with this convention, the superscript i denoting a set B of one element, that is $B \equiv \{i\}$.

A prefix vector of weight j and a suffix vector of weight j are denoted by ι^j and ω^j, respectively (Sec. 1.9 and Defs. 127-128). A suffix and a prefix of the same weight are clearly related by a rotation (Sec. 1.9 and Defs. 176-177) as follows: $\iota^j = j\uparrow_{-j}\iota^j$, and $\omega^j = j\downarrow_{-j}\omega^j$. Moreover, every infix vector is of the form $i\downarrow_{-j}\iota^j$.

If u is an infix vector and if $B \equiv u/A$, then B is called an infix or interval of the set A. Moreover, if u is a prefix or suffix vector, B is called a prefix or suffix of A.

If $u = \epsilon_A^B$, and $v = \epsilon_A^C$, then clearly $u \wedge v = \epsilon_A^{(B \wedge C)}$, and $u \vee v = \epsilon_A^{(B \vee C)}$. Consequently, sets and set operations can frequently be replaced by the analogous vectors and vector operations with respect to some specified universe of discourse A.

A logical vector u and the two vectors $a = \overline{u}/z$ and $b = u/z$ obtained by compressing a vector z, collectively determine the vector z. The operation which specifies z as a function of a, b, and u is called a mesh and is defined as follows. If a and b are arbitrary vectors and if u is a logical vector such that $\delta(\overline{u}) = \nu(a)$ and $\delta(u) = \nu(b)$, then the mesh of a and b on u is denoted by $\backslash a, u, b\backslash$ and is defined (Def. 145) as the vector z such that $\overline{u}/z = a$ and $u/z = b$. The mesh operation is equivalent to choosing successive components of z from a or b according as the successive components

of α are 0 or 1. If, for example, a = ((s), (e), (k)), b = ((t), (a)), and α = (0,1,0,1,0), then $\backslash a,b,\alpha \backslash$ = ((s), (t), (e), (a), (k)). As a further example, Program 6-12(a) (which describes the merging of the vectors a and b, with the first and every third component thereafter chosen from a) can be described alternatively as shown in Program 6-12(b). Since ι_6 = (1,2,3,4,5,6,$\cdots$) (Def. 166), then $\mid \iota_6, 3 \mid$ = (1,2,0,1,2,0,$\cdots$) (Def. 25), and consequently the vector α specified by the logical reduction (Def. 108) on step 1 is of the form α = (0,1,1,0,1,1,0,$\cdots$).

Mesh operations on matrices are defined analogously (Defs. 146,148), row mesh and column mesh being denoted by single and double reverse virgules, respectively.

In numerical or other vectors for which addition of two vectors is defined (Def. 28), the effect of the general mesh operation can be produced as the sum of two meshes each involving one zero vector. Specifically,

$$\begin{aligned} \backslash a,b,\alpha \backslash &= \backslash a,b,0 \backslash + \backslash 0,a,\alpha \backslash \\ &= \backslash 0,\bar{b},\alpha \backslash + \backslash 0,a,\alpha \backslash . \end{aligned}$$

The operation $\backslash 0,a,\alpha \backslash$ proves very useful in numerical work and will be called <u>expansion</u> of the vector a. Vector expansion is denoted by $a\backslash\alpha$ and is defined (Def. 154) as $a\backslash\alpha = \backslash 0,a,\alpha \backslash$. Compression of $a\backslash\alpha$ by α and by $\bar{\alpha}$ clearly yield a and 0, respectively, i.e.,

$$a/\backslash\alpha = a, \qquad \bar{\alpha}/a\backslash\alpha = 0.$$

Moreover, any numerical vector x can be <u>decomposed</u> by a compatible vector u according to the relation

$$x = u\backslash\overline{u}/x + u\backslash u/x.$$

The two terms are vectors of the same dimension which have no nonzero components in common. Thus if $u = (1,0,1,0,1)$, the decomposition of x appears as

$$x = \overbrace{(x_1,0,x_3,0,x_5)} + \overbrace{(0,x_2,0,x_4,0)}.$$

Row expansion and column expansion of matrices are defined analogously (Defs. 155,157). Row expansion of X by u is denoted by $u\backslash X$ and column expansion by $u\backslash\backslash X$. The decomposition relations become

$$X = u\backslash\overline{u}/X + u\backslash u/X,$$

and

$$X = u\backslash\backslash\overline{u}/X + u\backslash\backslash u/X.$$

If $u = (0,1,0,1)$ and $v = (1,0,1)$, and if $\mu(X) = \nu(X) = 2$, then

$$u\backslash X = \begin{bmatrix} 0 & x_1^1 & 0 & x_1^2 \\ 0 & x_2^1 & 0 & x_2^2 \end{bmatrix},$$

$$u\backslash\backslash X = \begin{bmatrix} x_1^1 & x_1^2 \\ 0 & 0 \\ x_2^1 & x_2^2 \end{bmatrix},$$

and

$$\backslash\!\backslash = \backslash\!\backslash\!\backslash = \begin{bmatrix} 0 & x_1^1 & 0 & x_1^2 \\ 0 & 0 & 0 & 0 \\ 0 & x_2^1 & 0 & x_2^2 \end{bmatrix}.$$

In expansion operations, the compatibility requirements (shown explicitly in Appendix A) concern the weight of the logical vector rather than its dimension. The latter quantity serves to determine the dimension of the resulting vector or matrix.

Mask operations (Defs. 160,162,163) are generalizations of the computer mask operations defined in Sec. 4.14. The vector mask, denoted by $\leftarrow /a,u,b/$, defines c as follows:

$$c_i = \begin{cases} a_i \text{ if } u_i = 0 \\ b_i \text{ if } u_i = 1. \end{cases}$$

The vectors a, u, and b must all be of the same dimension. The mask operation is also extended to matrices, single and double virgules denoting row mask and column mask, respectively.

The compress, expand, mask, and mesh operations on vectors clearly are related as follows:

$$/a,u,b/ = \backslash\!\backslash/a,u,b/\backslash,$$

$$\backslash u,u,\backslash = /\!\backslash u,u,\backslash\!\backslash/.$$

Analogous relations hold for the row mask and row mesh and for the column mask and column mesh.

Certain selection operations are controlled by logical matrices rather than by logical vectors. The row compression U/A, for example,

selects elements of A corresponding to the nonzero elements of B. Since the nonzero elements of B may occur in an arbitrary pattern, the result must be construed as a vector rather than a matrix. More precisely, B/A denotes the concatenation (Def. 152) of the vectors A^i/B^i obtained by row-by-row compression of A by B.

The column compression B//A (Def. 139) denotes the concatenation of the vectors A_1/B_1. If, for example,

$$B = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix},$$

then $B/A = (A^1_2, A^1_4, A^1_5, A^2_1, A^2_2, A^3_2, A^3_3)$,

and $B//A = (A^2_1, A^1_2, A^2_2, A^3_2, A^3_3, A^1_4, A^1_5)$.

Corresponding mesh, mask, and expansion operations are defined as shown in Sec. VIII of Appendix A.

6.3 Scan operations

Operations, such as the summation of all components of a vector, which require a scan of all components of a structured operand will be called <u>scan operations</u>. The <u>weight</u> of a vector X, denoted by $\sigma(X)$ and defined previously as the sum of the components of X, is a special case of the application of any associative binary operator θ to all components of the vector X. The <u>θ-weight of X</u> will then be denoted by θ/X, and defined (Def. 59)

as follows:

$$z \longleftarrow \theta/x \Longleftrightarrow z = x_1 \theta x_2 \theta \cdots \theta x_{-1}.$$

For example, $y \leftarrow \sigma(x)$, y is the product of the components of x, and

$$\vee/x = x_1 \vee x_2 \vee \cdots \vee x_{-1}$$

is the logical sum of all components of x.

The θ-weight operator is extended to matrices by the established convention of using a single virgule to denote an operation extending over rows and a double virgule to denote an operation extending over columns. Thus

$$x \longleftarrow \theta/[1]\, x \Longleftrightarrow x = x_1 \theta x_2 \theta \cdots \theta x_{-1},$$

and

$$z \longleftarrow \theta/[2]\, x \Longleftrightarrow z = x^1 \theta x^2 \theta \cdots \theta x^{-1}.$$

For example, if

$$x = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

then $\vee/x = (1,1,1)$ and $\vee/[2]\, x = (1,0,1,0)$.

Example 6-3. Using the ledger L defined in Example 6-2, produce a listing R of name, account number, and balance for each account having a quarterly balance less than two dollars for four successive quarters. The required selection can be based upon the

FP

status in each of the previous three quarters obtained by enlarging the matrix L to the form $L = [L_1, L_2, L_3, L_4, L_5, L_6]$, where L_4, L_5, and L_6 are logical vectors defined by $L_4 = u^{i-3}$, $L_5 = u^{i-2}$, $L_6 = u^{i-1}$, and u^j is the logical vector ($L_3 < 2\epsilon$) computed in period j. The production of the matrices T and L is then described by Program 6-13.

FP

Program 6-13. Step 1 shows the addition of a seventh column to L determined by the current balance (See Def.152). Step 2 shows the application of the logical and operation to each of the rows of the matrix formed by the last four columns of L. Step 3 shows the selection performed by the logical vector found in step 2. Step 4 shows the deletion of the oldest status vector L_4 to leave the matrix L in the form appropriate to the succeeding period.

The use of a scan operation on a vector of sets is also instructive. Let a be a vector of sets, that is, $a_i = A^i$. Then since the Cartesian product operator $\otimes$ (Def. 87) is binary and associative, the statement

$$C \leftarrow \otimes/a$$

specifies the set $C \equiv A^1 \otimes A^2 \otimes \cdots A^{\nu(a)}$. In particular, the homogeneous product set $[A]^k$ may be denoted alternatively by $\otimes/(a^j/A\epsilon)$.

For a vector of sets a it is convenient to define (Def. 3) the dimension vector $\nu(a)$ as follows: $\nu_i(a) = \nu(a_i)$, and $\nu(\nu(a)) = \nu(a)$. Then $\nu(\otimes/a) = \times/\nu(a)$, and for the homogeneous

product set $C = \otimes/(c^j/A\epsilon)$, $\nu(C) = \times/(c^j/\nu(A)\epsilon) = (\nu(A))^j$.

If b is any member of the product set $\otimes/a$, then clearly $\nu(b) = \nu(a)$. Moreover, the product sets containing compressed, expanded, meshed, masked, or permuted vectors can also be expressed easily in terms of the original product sets. For example, if $b \epsilon \otimes/a$, then $v/b \epsilon \otimes/(v/a)$. Similarly, if m is any mapping or permutation vector, then $b_m \epsilon \otimes/a_m$.

Since the operation of determining the maximum of several numerical quantities $x,y,\cdots,z$ is associative, maximization over the components of a vector could be treated in the manner described above. However, since it is often necessary to determine the indices of the components for which the maximum occurs instead of (or in addition to) the value of the maximum itself, an alternative treatment is preferred. The maximization operator determines a logical vector whose unit elements indicate the location of the maxima.

Maximization over the entire vector x is denoted by $\epsilon\lceil x$, and is defined as follows: if $v = \epsilon\lceil x$, then there exists a quantity m such that $v/x = m$ and such that all components of $\bar{v}/x$ are strictly less than m. The maximum is assumed by a single component of x if and only if $+/v = 1$. The actual value of the maximum is given by the first (or any) component of v/x. Moreover, the indices of the maximum components are the components of the vector v/ι. (See Def. 166.)

More generally, the maximization operation $v \longleftarrow u\lceil x$ will be defined so as to ignore the components of $\bar{u}/x$. Specifically, $v \longleftarrow u\lceil x$ implies that $v/x = m\epsilon$ and that $((\bar{v} \wedge u)/x < m\epsilon) = \epsilon$. The operation may be visualized as follows - a horizontal plane punched at points corresponding to the zeros of u is lowered over a plot of the components of x, and the positions at which the plane touches them are the positions of the unit components of v. For example, maximization over the negative components of x is denoted by

$$v \longleftarrow (x < 0)\lceil x,$$

and if $x = (2, -3, 7, -5, 4, -3, 6)$, then $v = (0,1,0,0,0,1,0)$, $v/x = (-3,-3)$, $(v/x)_1 = -3$, and $v/\iota = (2,6)$.

Ordinary maximization is defined with respect to order in the real numbers and can be generalized directly to any ordered set. Thus if x is any element of the homogeneous product set $[A]^{\nu(x)}$, the statement

$$v \longleftarrow u \underset{A}{\lceil} x$$

implies that $v/x = m\epsilon$ and that $((\bar{v} \wedge u)/x \underset{A}{\leqslant} m\epsilon) = \epsilon$. For example, if

$$H = \begin{pmatrix} \text{d} & \text{c} & \text{h} & \text{d} & \text{h} & \text{s} & \text{h} & \text{d} & \text{c} & \text{h} & \text{c} & \text{h} & \text{d} \\ 7 & 6 & 4 & 3 & 4 & 3 & 5 & \text{j} & 8 & 2 & \text{j} & 9 & 2 \end{pmatrix},$$

represents a hand of cards according to the conventions of Example 6-1, then

$$\Box_D^2 = (0,0,0,0,0,0,0,0,1,0,0,1,0,0),$$

and

$$\Box_S^1 = (0,0,0,0,0,0,1,0,0,0,0,0,0,0).$$

Maximization and minimization are extended to matrices as indicated by Defs. 68 and 71. Maximization over the entire matrix X is indicated by $\Box X$, where $\Box$ is the full matrix of all ones (Def. 130). Minimization is denoted by $\sqcap_A X$, and is defined analogously.

6.9 Mappings and permutations

A function f which defines for each element B_i of a set B a unique correspondent A_k in a set A is called a mapping from B to A. If $f(B_i) = A_k$, the element B_i is said to map into the element A_k. If the elements $f(B_i)$ exhaust the set A, f is said to map B onto A. If B maps onto A and the elements $f(B_i)$ are all distinct, the mapping is said to be one-to-one or biunique. In this case, $\nu(A) = \nu(B)$ and there exists an inverse mapping from B to A with the same correspondences.

A program for performing the mapping f from B to A must therefore determine for any given element $b \in B$, the correspondent $a \in A$ such that $a = f(b)$. Because of the convenience of operating upon a solid subset (infix) of the integers (e.g., upon register addresses or other numeric symbols) in the automatic execution of programs, the mapping is frequently performed in three successive phases, determining in turn the following quantities:

(1) the index i of the element b in B,

(2) the index k such that $A_k = f(B_i)$,

(3) the element A_k.

The three phases are shown in Program 6-14(a). The first (steps 1-3) is equivalent to mapping B onto the dense set of integers $\{(1,\nu(B))\}$, and is normally performed by scanning the set B in order and comparing each element with the argument b. The second is a permutation of the integers $\{(1,\nu(B))\}$ which may be described by a vector j, such that $j_i = k$. The selection of j_i (Step 4) then defines k which in turn determines the selection of A_k on step 5.

FP

Example 6-4. If $B \equiv \{$apple, booty, dust, eye, night$\}$ is a set of English words in alphabetical order, $A \equiv \{$Apfel, Auge, Beute, Nacht, Staub$\}$ is a set of German words also in alphabetical order, and if the function required is the mapping of a given English word b into its German equivalent a according to the following dictionary correspondences:

English:	apple	booty	dust	eye	night
German:	Apfel	Beute	Staub	Auge	Nacht

then $j = (1,3,5,2,4)$. If b = "night," then i = 5, $j_i = 4$, and $a = A_4$ = Nacht.

A vector (such as the vector j occurring in the definition of a mapping operation) whose components are each distinct elements of the set $\{(1,\nu(j))\}$ is called a permutation vector. If A is any

set and j is any compatible permutation vector, and if B is the set defined by the relation

$$B_i = A_{j_i},$$

then B is called the <u>permutation of A by j</u> and is denoted by $B \equiv A_j$. Because permutation by the vector j is used in the mapping from B to A, the permutation j is said to <u>map B onto A</u>, although the permutation is actually performed upon A. The <u>identity permutation vector</u> will be denoted by ι and defined as $\iota_k = k$. Clearly, $A_\iota \equiv A$.

The operation of permuting a set will be extended analogously to vectors, that is, $b = a_j$ defines the vector b such that $b_i = a_{j_i}$. If two permutation vectors j and k satisfy the relation $j_k = \iota$, then k is called the <u>inverse</u> of j. Since $j_k = \iota \Longleftrightarrow k_j = \iota$ the relation is mutual, and j is also the inverse of k. Permutation is associative (but not commutative) and therefore

$$(v_j)_k = (v)_{j_k}.$$

In particular, if i and j are mutually inverse, then

$$(v_j)_i = (v)_{j_i} = v_\iota = v.$$

If k is a permutation vector inverse to j, then Program 6-14(b) describes a mapping inverse to that of Program 6-14(a). The inverse mapping can also be described in terms of j as is done in Program 6-14(c). The selection of the

ith component of the permutation vector is then necessarily replaced by a scan of its components.

The vector $\underline{k}$ required for the mapping inverse to Example 6-4 is clearly $\underline{k} = (1,4,2,5,3)$. The vector $\underline{k}$ inverse to a given vector $\underline{j}$ can in general be defined by the <u>implicit statement</u>

$$\underline{k} \longleftarrow \underline{i} \text{ wo } \underline{j}$$

which may be read as "$\underline{i}$ specifies $\underline{k}$ with respect to $\underline{j}$", and interpreted as specifying the permutation $\underline{k}$ with respect to an already specified vector $\underline{j}$. The symbol "wo" will be used freely in similar implicit statements to separate the main specifying variable from the following auxiliary variables.

The use of contracurrent indices will be indicated by a negative sign as usual. Thus, the set $B \equiv A_{-\underline{i}}$ is the set A taken in reverse order, that is, $B \equiv \left\{A_{-1}, A_{-2}, \quad , A_{-\nu(A)}\right\}$. More generally, if $\underline{j}$ is any permutation vector, then $A_{-\underline{j}}$ is the set $A_{\underline{j}}$ taken in reverse order.

Permutation is extended to matrices as follows. <u>Row permutation</u> is defined by

$$B \longleftarrow A_{\underline{p}} \Longleftrightarrow B_{\underline{i}} = A_{\underline{p}_{\underline{i}}},$$

and effects a rearrangement among the columns of the matrix. Column permutation is denoted by $A^{\underline{p}}$.

It is convenient to define a more general mapping operation which allows both suppression and repetition of components as well as the rearrangement provided by the permutation operation. If,

for example, A is the set of alphabetic characters, and $j = (4,5,5,4)$, then the vector $t = A_j$ may reasonably be interpreted as the word (ⓓ,ⓞ,ⓔ,ⓒ), although j is not a permutation vector. Moreover, if $A \equiv \{$ⓐ, ⓑ, ⓒ, ⓓ, ⓔ$\}$, $B \equiv \{$ⓑ, ⓐ, ⓛ, ⓓ$\}$, and $m = (2,1,0,4,0)$, then $c = B_m$ may be interpreted as the vector (ⓐ, ⓑ, ⓓ) whose components are clearly the elements (in proper order) of the set $A \wedge B$.

Formally, a <u>mapping vector m</u> is defined as a vector each of whose components is either an integer or a null element. A <u>set mapping</u> is denoted by

$$c \longleftarrow B_m$$

and defined as follows: $c_i = B_{k_L}$, for $i \in \{(1,\nu(k))\}$, where $k = i = |((m \neq 0_\varepsilon)/m) - \varepsilon, \nu(B)\varepsilon|$.

Normally, the nonnull elements of m all belong to the index set of B, in which case k simply comprises the nonnull elements of m, that is, $k = (m \neq 0_\varepsilon)/m$. The reduction mod $\nu(B)$ simply ensures that the mapping operation is defined for any mapping vector m regardless of the dimension of the set B to which it may be applied. If m is a permutation vector and $\nu(m) = \nu(B)$, then $k = m$ and the mapping is a permutation.

Since a mapping vector m may contain two or more identical nonnull components, the entity B_m may contain two or more identical components and must therefore be considered as a vector rather than as a set. Like permutation, mapping may be extended directly to vectors and matrices, as shown in Defs. 168-171.

If A and B are two equal sets, then the association of identical elements establishes a one-to-one correspondence between the sets. The symbol $\mathcal{M}(A \longleftarrow B)$ will denote the mapping vector which maps B onto A, i.e., $B \equiv A_{\mathcal{M}}$. If, for example, $A \equiv \{\textcircled{c}, \textcircled{d}, \textcircled{h}, \textcircled{s}\}$, and $B \equiv \{\textcircled{h}, \textcircled{d}, \textcircled{s}, \textcircled{c}\}$, then $\mathcal{M}(A \longleftarrow B) = (3,2,4,1)$, and $\mathcal{M}(B \longleftarrow A) = (4,2,1,3)$. If A = B, then $\mathcal{M}(A \longleftarrow B)$ is clearly a permutation vector, and $\mathcal{M}(B \longleftarrow A)$ is the inverse permutation.

For arbitrary sets, the mapping vector $m = \mathcal{M}(A \longleftarrow B)$ is defined more generally as a vector of the same dimension as B such that $m_i = k$ if $B_i = A_k$, and $m_i = o$ if $B_i \notin A$. If $c = A_m$, and $C \equiv B \wedge A$, then clearly $c_i = C_i$. For example, if $A \equiv \{\textcircled{a}, \textcircled{b}, \textcircled{c}, \textcircled{d}, \textcircled{e}\}$, and $B \equiv \{\textcircled{b}, \textcircled{a}, \textcircled{\ell}, \textcircled{d}\}$, then $m = \mathcal{M}(A \longleftarrow B) = (2,1,o,4)$, $c = A_m = (\textcircled{b}, \textcircled{a}, \textcircled{d})$, and $C \equiv B \wedge A = \{\textcircled{b}, \textcircled{a}, \textcircled{d}\}$.

In the analogous mapping from a vector b to a vector a (denoted by $\mathcal{M}(a \longleftarrow b)$), the possibility of two or more identical components in a leads to potential ambiguity which will be resolved by choosing the index of the first of any group of identical components. Formally, if $m = \mathcal{M}(a \longleftarrow b)$, then $m_i = o$ if $b_i \notin C$, and m_i is otherwise the smallest integer such that $b_i = a_{m_i}$, where C is the set of distinct components of a. This definition clearly covers all of the simpler cases previously considered, and the mapping vector $m = \mathcal{M}(\alpha \longleftarrow \beta)$ may be defined for any set or vector α and any set or vector β, as shown in Def. 172.

Example 6-5. Let it be required to determine the kth element of the set $B \wedge A$. This may be performed efficiently by Program 6-15 which applies the mapping vector $m = \mathcal{M}(A \longleftarrow B)$ to the set A rather than produce the set $B \wedge A$ explicitly. If the sets A and B are fixed, the first two steps of the program need be performed but once and need not be repeated for each value of k treated.

It is sometimes useful to consider a set A as a vector, that is, to specify a vector a such that $a_i = A_i$. This is denoted by the statement

$$a \longleftarrow A.$$

Vector operations may then be applied to the vector a. For example, Programs 6-14(a-c) may be reformulated more concisely in terms of vector operations as shown in Programs 6-16(a-c).

Because it may contain duplicate components, a vector cannot specify a set so directly as a set specifies a vector. It is, however, useful to denote by the statement

$$C \longleftarrow a,$$

the set C of distinct components of a, specifically the ordered set obtained by suppressing each recurrence of any repeated component of the vector. If, for example, $a = (\text{C}, \text{a}, \text{n}, \text{a}, \text{d}, \text{a})$, then $C \equiv \{\text{C}, \text{a}, \text{n}, \text{d}\}$, and moreover, $\mathcal{M}(C \longleftarrow a) = (1,2,3,2,4,2)$, and $\mathcal{M}(a \longleftarrow C) = (1,2,3,5)$.

It is sometimes necessary to reorder or sort (i.e., to permute) the components of a numerical vector x so as to arrange them in increasing order. The permutation vector j required to effect this will be called the ordering vector of x and will be denoted by $j = \circledcirc x$. Then $y = x_j$ has the property that $y_i \leq y_k$ for $i < k$. Since the components of x need not be distinct, this definition of the ordering vector must be refined so as to remove potential ambiguity. This is done so as to preserve the initial relative order of each set of equal components. Formally, j is defined as follows: j is a permutation vector such that either $y_i < y_{i+1}$ or $y_i = y_{i+1}$ and $j_i < j_{i+1}$. If, for example, $x = (3,17,-2,3,5)$, then $j = \circledcirc x = (3,1,4,5,2)$, and $y = x_j = (-2,3,3,5,17)$.

More generally, any vector a which is an element of the homogeneous product set $[B]^{\nu(a)}$ may be ordered on the set B. Thus $j = \circledcirc_B a$ is defined (Def. 173) as follows. If $c = a_j$, then j is a permutation vector such that either $c_i \underset{B}{<} c_{i+1}$ or $c_i = c_{i+1}$ and $j_i < j_{i+1}$. If, for example, $a = (3, \textcircled{q}, 6, \textcircled{j}, 6, \textcircled{k})$, and $B \equiv \left\{(2,10), \textcircled{j}, \textcircled{q}, \textcircled{k}\right\}$, then $j = \circledcirc_B a = (1,3,5,4,2,6)$, and $a_j = (3,6,6, \textcircled{j}, \textcircled{q}, \textcircled{k})$.

Rotations. The rotation operation $k \uparrow A$ defined in Sec. 1.8 is a special but very important case of permutation. Its definition and its extension to row and column rotations of matrices are shown in Defs. 174-181. Briefly, the row rotation $C \leftarrow k \uparrow A$ rotates each row of A so that $C^i = k_i \uparrow A^i$. For

column rotation, $C \longleftarrow k \uparrow A \iff C_i = B_i \uparrow A_i$.

Any vector rotation $k \uparrow a$ may be expressed as a permutation by $j = k \uparrow \iota$, the corresponding rotation of the identity permutation vector, that is, $k \uparrow a = a_j$. The general row (or column) rotation is more complex than a single permutation, but for the special case $k = k\iota$, $(k\iota) \uparrow A = A_j$.

Ranking. The first step in performing a mapping is, as illustrated by Program 6-14, the determination of the index in a set B of some argument b. Because of its importance, this operation is given the special name of ranking, and the special notation $\iota(b \text{ wo } B)$ defined (Def. 89) as follows:

$$c \longleftarrow (b \text{ wo } B) \iff \begin{cases} c = i \text{ if } b = B_i \\ c = \circ \text{ if } b \notin B \end{cases}$$

More generally, if b is a vector belonging to the product set $\otimes/a$, then the operation $\iota(b \text{ wo } a)$ is defined as follows:

$$c \longleftarrow \iota(b \text{ wo } a) \iff \begin{cases} c_j = i \text{ if } b_j = (a_j)_i \\ c_j = \circ \text{ if } b_j \notin a_j. \end{cases}$$

Similarly,

$$C \longleftarrow \iota(B \text{ wo } A) \iff \begin{cases} C^k_j = i \text{ if } B^k_j = (A^k_j)_i \\ C^k_j = \circ \text{ if } B^k_j \notin A^k_j, \end{cases}$$

where each component A^k_j is a set.

If, for example,

$$H = \begin{pmatrix} d & e & h & d & h & s & h & d & e & h & e & q & d \\ a & 6 & k & q & 4 & 3 & 5 & k & 12 & 2 & e & 9 & 2 \end{pmatrix},$$

S and D are, respectively, the set of suits and the set of denominations of Example 6-1, and $\alpha = (S,D)$, then $\iota(\square_1 \text{ wo } \alpha) = (2,13)$, $\iota(\square_2 \text{ wo } \alpha) = (1,5)$, $\cdots$, $\iota(\square_{13} \text{ wo } \alpha) = (2,1)$. Moreover, since the outer product

$$\square = \square \mid \square = \begin{pmatrix} S,S & & S \\ & \cdots & \\ D,D, & & D \end{pmatrix},$$

(Def. 50), then

$$\iota(\square \text{ wo } \square) = \begin{pmatrix} 2 & 1 & 3 & 2 & 3 & 4 & 3 & 2 & 1 & 3 & 1 & 0 & 2 \\ 13 & 5 & 12 & 11 & 3 & 2 & 4 & 12 & 0 & 1 & 0 & 3 & 1 \end{pmatrix}.$$

The actual value of the rank of an element b in a set B (that is, $\iota(b \text{ wo } B)$) depends upon the origin used for the index set of B. The rank may therefore be referred to as the 0-origin or the 1-origin rank according to the system in use. The quantity

$$\iota(b \text{ wo } B) - \phi(B)$$

is clearly independent of origin.

In operations such as left justification and right justification, it is convenient to be able to specify the weight k of the longest prefix (solid initial sequence of 1's) in a given logical vector u. The quantity k is called the prefix weight of u. It is denoted by

$$k \leftarrow\!\!\!- \alpha(u)$$

and defined (Def. 115) as follows:

$$k = \max_j \left((\iota_{\square} j_{\square}) \times (\iota_{\square} j_{\square}) \right).$$

If, for example, $x = (1,1,1,0,1,0,1)$, then $\alpha(x) = 3$, $x^{3}/x = (1,1,1)$, and $x^{4}/x = (1,1,1,0)$. If x is a numerical vector which is to be compressed so as to eliminate all leading zeros (i.e., all zeros preceding the first significant digit), the compressed vector y may be defined as follows:

$$k \leftarrow \alpha(x = 0)$$
$$y \leftarrow x^{k}/x .$$

Alternatively, x may be rotated left by k so as to produce the left justified vector $z = k\uparrow x$ with the dimension unchanged.

The <u>suffix weight</u> is defined analogously and is denoted by $\omega(x)$. Clearly, $\alpha(x^{j}) = \omega(x^{j}) = j$. Moreover, if $x = 0$, then $\alpha(x) = \omega(x) = 0$, and if $x = \varepsilon$, then $\alpha(x) = \omega(x) = \nu(x)$.

The <u>row prefix weight</u> of a logical matrix U is a vector k whose components are the prefix weights of the successive rows of U. It is denoted by $k = \alpha(U)$. The column prefix weight is defined analogously and is denoted by $\alpha((U))$. Row and column suffix weights are also defined as shown in Defs. 119, 120. A numerical matrix X may be left justified by the rotation $\alpha(0 = 0)\uparrow X$, or top justified by the rotation $\alpha((X = 0))\uparrow X$.

6.10 Algebraic vector and matrix operations

The algebraic operations defined upon vectors and matrices appear in Sec. III of Appendix A. Definitions 12-45 represent straightforward extensions of common algebraic operations defined upon scalars. Definitions 46-51 cover the remaining elementary operations of matrix algebra. The incorporation of the compress and expand operations leads to a powerful extension of ordinary algebra whose basic identities are summarized in Appendix B. Proofs of these identities are left as an exercise.

Any positional representation of a number in a base b system can be considered as a numerical vector x whose <u>base b value</u> is the number represented. More generally, x may represent a number in a mixed radix system in which the successive radices (from left to right) are the successive components of a vector y.

The <u>base y value of the vector x</u> is a scalar denoted by $y \perp x$ and defined (Def. 52) as

$$y \perp x = sx = \sigma(y \times x),$$

where $s_{-1} = 1$, and

$$s_{i-1} = s_i \times y_i, \qquad i \in \{(2, \nu(x))\}.$$

If $y = b\mathcal{C}$, then $s_i = b^{\nu(x)-i}$ and $(b\mathcal{C}) \perp x$ is therefore the value of the base b number formed by the components of x taken in natural order. For example, if $x = (1,2,2)$ and $y = 3\mathcal{C}$, then

$y \perp x = (122)_3 = (17)_{10}$.

More generally, if y is not of the form $y\epsilon$, the value of $y \perp x$ is the value in the mixed base system $y_1,\ldots,y_{-1}$ of the number formed by the components of x. The weight assigned to the component x_{i-1} is y_i times the weight assigned to the component x_i. For example, if the successive components of x represent elapsed weeks, days, hours, minutes, and seconds respectively, and if $y = (52,7,24,60,60)$, then $y \perp x$ is the elapsed time in seconds.

The base y value of a vector allows a further useful interpretation, for if y is any number (not necessarily integral) then $(y\epsilon) \perp x$ is the value of the polynomial

$$z = x_1 y^{\nu(x)-1} + x_2 y^{\nu(x)-2} + \cdots + x_{-1}.$$

It is frequently necessary to specify a vector j as the base b representation of an integer k. This specification is implicit, and the auxiliary variables must be indicated as follows:

$$b\epsilon \perp j \longleftarrow k \text{ wo } b.$$

The statement

$$(10\epsilon) \perp j \longleftarrow (2\epsilon) \perp x \text{ wo } \nu(j)$$

denotes a conversion from dyadic to decimal. The dimension of the resulting decimal representation must, of course, be separately specified as indicated by the auxiliary variable $\nu(j)$.

To ensure uniqueness in the statement

$$x \perp j \longleftarrow i \text{ wo } b$$

which specifies j as the representation of i in the mixed base b, it is necessary to restrict the integers in any position of the

representation to less than the relative weight of the next higher component, i.e., $0 \leq j_i < k_i$. Then

$$k^2 \perp_j \longleftarrow k^1 \perp_i w_0 k^2$$

denotes a conversion from the mixed base k^1 to the mixed base k^2. The dimension of j is determined by compatability with the specified vector k^2.

The base ρ operation on vectors is extended to rows and to columns of a matrix in Defs. 53 and 54, respectively. Thus

$$u \longleftarrow v \perp x \iff u_i = v \perp x^i,$$

and

$$u \longleftarrow v \perp\!\!\perp x \iff u_i = v \perp x_i.$$

If x is a numerical vector belonging to the product set $\otimes/a$, where each component a_i is a solid set of integers of the form $\{(0,(k-1)_i)\}$, then $\rho(a) = k$, and the 0-origin rank of x in $\otimes/a$ is equal to the base k value of x. If, for example, $x = (0,2,1,13)$ is the elapsed time in days, hours, minutes, and seconds, then

$$a = \{(0,7-1)\}, \{(0,24-1)\}, \{(0,60-1)\}, \{(0,60-1)\}),$$

$$k = (7,24,60,60),$$

and $z = x \perp x = 7273$ is indeed the 0-origin rank of x in $\otimes/a$, that is, x is the 7274th element of $\otimes/a$.

More generally, for an arbitrary vector b belonging to a product set $\otimes/a$, its 0-origin rank in $\otimes/a$ may be determined as the base $\rho(a)$ value of $\iota(b \text{ in } a)-\phi c$, that is, of the 0-origin index

of ω in α. By analogy, this will be denoted by $\nu(\alpha)\underset{\alpha}{\perp} b$ and defined as follows for all $b \in \Omega/\alpha$:

$$\nu(\alpha)\underset{\alpha}{\perp} b = \nu(\alpha) \perp (\iota(b \text{ in } \alpha)-\phi 0).$$

The definition can be extended immediately to any compatible base β as follows:

$$\nu\underset{\beta}{\perp} b = \nu \perp (\iota(b \text{ in } \omega)-\phi 0).$$

If, for example, $b = (\text{d},\text{q})$, S and D are the sets of Example 6-1, and $\alpha = (S,D)$, then $\nu(\alpha) = (4,13)$, and $\nu(\alpha)\underset{\alpha}{\perp} b = (4,13) \perp (1,10) = 23$. Moreover, if

$$\omega = \begin{pmatrix} \text{d} & \text{c} & \text{h} & \text{d} & \text{h} & \text{s} & \text{h} & \text{d} & \text{c} & \text{h} & \text{c} & \text{h} & \text{d} \\ \text{a} & 6 & \text{k} & \text{q} & 4 & 3 & 5 & \text{k} & 8 & 2 & \text{j} & 9 & 2 \end{pmatrix}$$

and $j = \Theta(\nu(\omega)\underset{\alpha}{\amalg}\omega)$, then (assuming 0-origin indexing for Π also)

$$j = (1,8,10,12,3,7,0,9,4,6,11,2,5)$$

and

$$\omega_j = \begin{pmatrix} \text{c} & \text{c} & \text{c} & \text{d} & \text{d} & \text{d} & \text{d} & \text{h} & \text{h} & \text{h} & \text{h} & \text{h} & \text{s} \\ 6 & 8 & \text{j} & 2 & \text{q} & \text{k} & \text{a} & 2 & 4 & 5 & 9 & \text{k} & 3 \end{pmatrix}$$

is the hand reordered on the product set $S \otimes D$, that is, ordered on denomination within suit.

6.11 Levels of structure

Just as the matrix is defined as a vector each of whose components is a vector, so may further levels of structure be established by defining a vector each of whose components is a

matrix, and so forth. Although in certain fields, such as tensor analysis, it is convenient to define more general arrays whose rank specifies the number of levels of structure (i.e., zero for a scalar, one for a vector of scalars, two for a vector of vectors (matrix), three for a vector of matrices, etc.), the notation will here be limited to the two levels provided by the matrix,[*] and

---

[*] The only essential particularization to two levels occurs in the provision of single and double symbols (e.g., "/" and "//"; "↑" and "⇑"; "⊥" and "⫫") for row and column operations, respectively, and in the use of superscripts and subscripts for denoting rows and columns, respectively. In applications requiring multiple levels, the former can be generalized by adjoining to the single symbol an index which specifies the coordinate (e.g., "$/_1$" and "$/_2$", for row and column compression, and, in general, "$/_j$".) The latter can be generalized by using a vector index subscript possessing one component index for each coordinate.

---

passage to grosser or finer levels of structure will be indicated explicitly.

For example, if a sequence of vectors $\mathbb{y}^1, \mathbb{y}^2, \ldots, \mathbb{y}^m$ are to define the column vectors of the matrix $\mathbb{X}$, this will be denoted by the statement

$$\mathbb{X}_i \leftarrow \mathbb{y}^i, \qquad i \in \{(1,m)\}.$$

On the other hand, the statement

$$X^i \longleftarrow y^i, \qquad i \in \{(1,m)\},$$

defines the matrix X whose row vectors are y^i. Moreover, if ^iM is a member of a sequence of k matrices, the vector v whose components are the matrices ^iM, may be defined by the statement

$$v_i \longleftarrow {}^iM, \qquad i \in \{(1,k)\}.$$

Certain of the selection operations are particularly convenient for describing transitions between different levels of structure. For example, the row compression $c \longleftarrow B/A$, (Def. 138) specifies the components of the vector c as the successive components (in row order)) of the matrix A selected by the logical matrix B. If $B \longleftarrow (A \neq 0)$, then c is the vector of nonzero elements of A which is often used together with the logical matrix B to provide a concise representation of a so-called sparse matrix containing numerous zero elements. Clearly, the row mesh (Def. 147) serves to reconstitute the matrix A, that is, $\backslash c,B,c\backslash = A$. Alternatively, $A = B\backslash c$. Moreover, the row compression $a \longleftarrow I/A$ and the column compression $b \longleftarrow J/A$ can be used to define the row list and column list vectors (Defs. 143,144) obtained by taking the elements of A in row and in column order, respectively. Finally, the diagonal elements of a matrix A define the vector d as follows: $d \longleftarrow I/A$. Other uses of these and related operations are developed in the exercises.

6.12 Files

Many devices used for the storage of information impose certain restrictions upon its insertion or withdrawal. The items recorded on a magnetic tape, for example, may be read from the tape much more quickly in the order in which they appear physically on the tape than in some other prescribed order.

Certain storage devices are also self-indexing in the sense that the item selected in the next read from the device will be determined by the current state or position of the device. The next item read from a magnetic tape, for example, is determined by the position in which the tape was left by the last preceding read operation.

To allow the convenient description of algorithms constrained by the characteristics of storage devices, the following special notation will be adopted. A file of length n is a representation of a vector x of dimension n arranged as follows:

$$\lambda(1), x_1, \lambda(2), x_2, \ldots, \lambda(n), x_{\nu(x)}, \lambda(\nu(x)+1), \ldots, \lambda(-1).$$

The operation of transferring a component from a file to specify a quantity y is called reading the file and is denoted by $y \leftarrow\!\!- \Phi$. The transfer is terminated by the occurrence of a partition symbol, and if this symbol is $\lambda(j)$ the file is then said to be in position j. A file may either be read forward (denoted by ${}_0\Phi$) or backward (denoted by ${}_1\Phi$). If a file originally in position j is read forward it transfers the component x_j and stops in position

$(j+1)$, $j \in \{(1,\nu(x))\}$. A file read backward from position $j+1$ transfers the component x_j and stops in position j, $j \in \{(1,\nu(x))\}$.

The position of a file Φ will be denoted by $\pi(\Phi)$. Thus the statement $j \longleftarrow \pi(\Phi)$ specifies j as the position of Φ, whereas $\pi(\Phi) \longleftarrow j$ positions the file to j. In particular, $\pi(\Phi) \longleftarrow 1$ denotes the rewinding of the file, and $\pi(\Phi) \longleftarrow -1$ denotes winding, i.e., positioning to the end of the file. Any file for which the general positioning operation $\pi(\Phi) \longleftarrow j$ is to be avoided as impossible or inefficient is called a serial or serial-access file.

A file may be produced by a sequence of recording statements, either forward:

$$_0\Phi \longleftarrow x_i, \qquad i = 1,2,\ldots,\nu(x),$$

or backward:

$$_1\Phi \longleftarrow x_i, \qquad i = \nu(x),\nu(x)-1,\ldots,1.$$

As in reading, each forward (backward) record operation increments (decrements) the position of the file by one. A file which is only recorded during a process is called an output file of the process; a file which is only read is called an input file.

Different files occurring in a process will be distinguished by right-hand subscripts and superscripts, the latter being generally reserved to denote major classes of files (e.g., input and output). In a forward read or record, the zero prescript may be elided.

Each terminal partition (that is, $\lambda(1)$ and $\lambda(-1)$) assumes a single fixed value denoted by λ. Each of the remaining partitions

$\lambda(j)$ may assume one of several values denoted by $\lambda_0, \lambda_1, \cdots, \lambda_k$, the partitions with larger indices normally demarking larger subgroups within the file. Thus if x were the row list of a matrix, the last component might be followed by the partition λ_2, the last component of each of the preceding rows by λ_1, and the remaining components by λ_0. In recording an item, the associated partition is indicated by listing it after the item (e.g., $\Phi \longleftarrow y, \lambda_2$), except that the partition λ_0 is usually elided. The indicated partition then follows or precedes the associated item in the file according as the recording is forward or backward.

The indication provided by the k distinct partition symbols is used to control an immediate (k+1)-way branch in the program following each read operation. The branch is determined by the partition symbol which terminates the read.

Example 6-5. A set of m input files Φ_i^1, $i \in \{(1,m)\}$, each terminated by a partition λ_1 are to be copied to a single output file Φ_1^2 as follows. An identification quantity s is to be recorded first and successive items (components) are then chosen in turn from files $\Phi_1^1, \Phi_2^1, \cdots, \Phi_m^1, \Phi_1^1, \Phi_2^1, \cdots$, always omitting from the sequence any exhausted file. The entire process is described by Program 6-17.

Program 6-17. Step 12 indicates a read from the kth input file to specify the variable b. Step 10 cycles k through the values 1 to m, and step 11 allows the read on step 12 to occur only if $\xi_k = 0$. The logical vector ξ is of dimension m and designates the set of exhausted files. It is respecified by step 13

whenever a file is exhausted (as indicated by the occurrence of a partition λ_1 in the read on step 12).

Each read on step 12 is followed (either immediately or after execution of step 13) by step 8, which records on the output file the quantity b just read. Step 9 terminates the process when all files are exhausted. Steps 1-4 perform an initial rewind of all files, and steps 5-7 initialize the variables k, u_j and b.

The selection operation defined upon matrices can be applied in an obvious way to an array of files Φ^i_j. For example, the statement

$$\pi(\Phi^i) \longleftarrow \in$$

denotes the rewinding of the row of files Φ^i_j, $j \in \{(1, \nu(\Phi))\}$; the statement

$$\pi(\Phi_j) \longleftarrow \in$$

denotes the rewinding of the column of files Φ^i_j, $i \in \{(1, \mu(\Phi))\}$; and the statement

$$\upsilon/\Phi^i \longleftarrow \upsilon/p$$

denotes the recording of the vector component p_j on file Φ^i_j for all j such that $u_j = 1$.

As for vectors and matrices, either 1-origin or 0-origin indexing may be used, and will apply to the indexing of the file positions as well as to the array indices. The prescripts, however

(denoting direction) remain the same in either system. Thus if 0-origin indexing is used, the rewind and wind operations become $\pi(\Phi_j^i) \leftarrow\!\!- 0$, and $\pi(\Phi_j^i) \leftarrow\!\!- -0$, respectively. 0-origin indexing is used in the following example.

Example 6-7. Files Φ_0^0 and Φ_1^0 contain the vectors x and y, respectively, each of dimension n. In the first phase, the components are to be merged in the order $x_0, y_0, x_1, y_1, \ldots, x_{-0}, y_{-0}$, and the first n components of the resulting vector are to be recorded on file Φ_0^1, and the last n on file Φ_1^1. In other words, the vectors $x^1 = \omega^n/z$, and $y^1 = \omega^n/z$ are to be recorded on Φ_0^1 and Φ_1^1, respectively, where $z = \backslash x,u,y \backslash$, and $u = (0,1,0,1,\ldots,0,1)$. In the next phase, the roles of input and output files are reversed and the same process is performed upon x^1 and y^1, that is, $x^2 = \omega^n/(\backslash x^1, u, y^1 \backslash)$, and $y^2 = \omega^n/(\backslash x^1, u, y^1 \backslash)$ are recorded on files Φ_0^0 and Φ_1^0, respectively. The process is to be continued through m phases.

Program 6-18. The program for Example 6-7 begins with the rewind of the entire 2×2 array of files. To obviate further rewinding, the second (and each subsequent even numbered) execution is performed by reading and recording all files in the backward direction. Step 6 performs the essential read and record operation under control of the logical vector u whose components u_1, u_2, u_3 determine respectively the subscript of the file to be read, the subscript of the file to be recorded, and the direction of read

and record. The file superscripts (determining which classes serve as input and output in the current repetition) are also determined by u_3, the input being u_3 and the output $\bar{u}_3$. The loop 6-8 copies n items, alternating the input files through the negation of u_1 on step 7. When the loop terminates, u_2 is negated to interchange the outputs, and the loop is repeated unless $u_2 = u_3$. Equality occurs and causes a branch to step 3 if and only if all 2n items of the current repetition have already been copied.

Step 3 decrements m and is followed by the negation of u on step 5. The component u_3 must of course be negated to reverse direction, but the need to negate u_1 and u_2 is not so evident. It arises because the copying order was prescribed for the forward direction, beginning always with the operation

$$ {}_0\Phi_0^{u} \longleftarrow {}_0\Phi_0^{\bar{u}} . $$

An equivalent backward copy must therefore begin with the operation

$$ {}_1\Phi_1^{u} \longleftarrow {}_1\Phi_1^{\bar{u}} . $$

Not all computer files have the very general capabilities indicated by the present notation. Some files, for example, can be read and recorded in the forward direction only and, except for rewind, cannot be positioned directly. Positioning to an arbitrary position k must then be performed by a rewind and a succession of (k-1)

subsequent reads. In some files, recording can be performed in the forward direction only, and the positions are defined only by the recorded data itself. Consequently, recording in position k makes unreliable the data in all subsequent positions, and recording must always proceed through all successive positions until terminated. If the translation from the program to the computer coding is to be kept simple, the file operations employed should be limited to those within the capabilities of the available files.

6.13 Trees

A structured operand is more than the collection of its components, since certain associations between these components are also implied. The associations in a vector, for example, can be described by a single sequence as depicted graphically in Fig. 6-1. The most general type of structured operand is called a directed linear graph, its components are called nodes, and an association from node i to node j is called a branch from node i to node j. A branch from node i to node j is said to leave node i and to enter node j.

A directed linear graph admits of a simple graphical representation as illustrated by Fig. 6-2. The nodes are depicted as circles, and the branches as directed lines. The nodes might, for example, represent places, and the lines, connecting streets. A two-way street is then represented by a pair of oppositely directed

lines as shown between nodes s and y. The structure of the graph is represented entirely by the directed lines.

If p is a vector whose components are nodes of a given graph, and if for each $i \in \{(2, \ell(p))\}$ there is an association from node p_{i-1} to node p_i, then p is called a <u>path of length $\ell(p)$ from node p_1 to node $p_{\ell-1}$</u>. Nodes p_1 and $p_{\ell-1}$ are called the <u>initial</u> and <u>final</u> nodes of the path, respectively, and either may also be called a <u>terminal</u> node. Any infix of a path p is also a path (possessing, in general, terminal nodes different from those of p), and is called a <u>subpath</u> of p. A subpath of p not identical with p is called a <u>proper</u> subpath of p. Any subpath of p is said to be <u>contained</u> in p, and any proper subpath is said to be <u>properly</u> contained. If there exists a path from node i to node j, then node j is said to be <u>reachable</u> from node i.

A path of length one is called a <u>trivial path</u>, and a non-trivial path whose terminal nodes are identical is called a <u>circuit</u>. A path whose proper subpaths do not include a circuit is called a <u>minimal path</u>. In the graph of Fig. 6-2, for example, $p = (s , t , w , u , t , x , v , w)$ is a path of length eight from node (s) to (w) which contains (among others) the proper subpaths $q = (t , w , u)$, and $r = (t , w , u , t)$, the latter of which is a circuit. Removal from p of the infix r^{-1}/q leaves the path $\theta = (s , t , x , v , w)$ which is a minimal path between the initial and final nodes of the path p, but which is <u>not</u> a subpath of p. The minimal path θ from node s to node w is not the

shortest path from  to  , since the path $\gamma = (s, y, s, w)$, though not minimal, is shorter.

A graph (such as Fig. 6-3) which contains no circuits and which has at most one branch entering each node is called a tree. Since each node is entered by at most one branch, a path existing between any two nodes in a tree is unique, and the length of path is likewise unique. Moreover, if any two paths have the same final node, one is a subpath of the other.

Since a tree contains no circuits, the length of path in a finite tree is bounded. There therefore exist maximal paths which are properly contained in no longer paths. The initial and final nodes of a maximal path are called a root and leaf of the tree, respectively. A root is said to lie on the first level of the tree and, in general, a node which lies at the end of a path of length j from a root, lies in the jth level of the tree.

A tree which contains n roots is said to be n-tuply rooted, and if $n = 1$ it is called a rooted tree. If $n > 1$, the sets of nodes reachable from each of the several roots are disjoint, for if any node is reachable by paths from each of two disjoint roots, one is a proper subpath of the other and is therefore not maximal. Likewise, any node of a tree defines a subtree of which it is the root, consisting of itself and all nodes reachable from it, with the same associations as the parent tree. Every subtree of a tree is itself a rooted tree.

If for each level j, a simple ordering is assigned to each of the disjoint sets of nodes reachable from each node of the

preceding level, and if the roots are also simply ordered, then the tree is said to be ordered. Moreover, in such an ordered tree, any path of length k from a root can be uniquely specified by an index vector $\underset{\sim}{i}$ of dimension k, where i_1 specifies the particular root, and the remaining components specify the (unique) path as follows: the path node on level j is the i_jth element of the set of nodes on level j reachable from the path node on level j - 1. Either 0-origin or 1-origin indexing can be used in the description of trees, but the latter will be used exclusively in this chapter.

Attention will henceforth be restricted to ordered trees, which will be denoted by upper-case boldface Roman characters. The height of a tree $\underset{\sim}{T}$ is defined as the length of the longest path in $\underset{\sim}{T}$ and is denoted by $\nu(\underset{\sim}{T})$. The number of nodes on level j is called the moment of level j and is denoted by $\mu_j(\underset{\sim}{T})$; the total number of nodes in $\underset{\sim}{T}$ is called the moment of $\underset{\sim}{T}$ and is denoted by $\mu(\underset{\sim}{T})$. Clearly, $\nu(\mu(\underset{\sim}{T})) = \nu(\underset{\sim}{T})$, and $\sigma(\mu(\underset{\sim}{T})) = \mu(\underset{\sim}{T})$. The number of roots is equal to $\mu_1(\underset{\sim}{T})$, and the number of leaves will be denoted by $\lambda(\underset{\sim}{T})$.

The path in a tree $\underset{\sim}{T}$ specified by an index vector $\underset{\sim}{i}$ will be denoted by $\underset{\sim}{T}^{\underset{\sim}{i}}$. The vector $\underset{\sim}{i}$ is called the index of path $\underset{\sim}{T}^{\underset{\sim}{i}}$ and is also called the index of $(T^{\underset{\sim}{i}})_{-1}$, the node at the end of the path. The node $(T^{\underset{\sim}{i}})_{-1}$ will also be referred to as node $\underset{\sim}{i}$. If, for example, $\underset{\sim}{T}$ is the tree of Fig. 6-3, then $\underset{\sim}{i} = (2,2,2,3)$ is the index of the path (b,h,m,y) and of the node y. An integral vector $\underset{\sim}{i}$ is said to be an index of $\underset{\sim}{T}$ if it is the index of some path in $\underset{\sim}{T}$.

If the nodes of $\underset{\sim}{T}$ all have distinct values, then the index of a node x in $\underset{\sim}{T}$ is unique and will be denoted by $\iota(x \text{ wo } \underset{\sim}{T})$. More generally,

$$i \longleftarrow \iota(x \text{ wo } \underset{\sim}{T})$$

implies that if j is any other index such that $(\underset{\sim}{T}^j)_{-1} = x$, then either $\nu(i) < \nu(j)$ or $\nu(i) = \nu(j)$ and i precedes j in the product set to which they both belong. For example, in the tree of Fig. 6-3, $\iota(h \text{ wo } \underset{\sim}{T}) = (2,2)$, although the final nodes of $\underset{\sim}{T}^{(1,3,1)}$ and $\underset{\sim}{T}^{(3,1)}$ are also equal to h.

The subtrees whose roots are directly reachable from the final node of the path T^i together form a tree which will be denoted by $\underset{\sim}{T}_i$. Thus if $i = (2,2,2)$ in Fig. 6-3, $\underset{\sim}{T}_i$ consists of the subtrees rooted at nodes $(2,2,2,1)$, $(2,2,2,2)$, and $(2,2,2,3)$. If j is any index of the tree T_i, then $(\underset{\sim}{T}_i)_j$ is the tree $\underset{\sim}{T}_k$, where $k = (i,j)$. A path in T_i is denoted by $(T_i)^j$. For example, if $\underset{\sim}{G}$ is an ascending genealogical tree with the sword and distaff sides denoted by the indices 1 and 2, respectively, then the paternal male ancestors of any individual x are represented by the path vector $(\underset{\sim}{G}_i)^\epsilon$, where $i = \iota(x \text{ wo } \underset{\sim}{G})$ and where the dimension of ϵ determines the length of the line considered.

The number of branches leaving a node $(T^i)_{-1}$ is called its <u>branching ratio</u> or <u>degree</u>. The degree of node $(T^i)_{-1}$ is denoted by $\delta((T^i)_{-1})$. Moreover, $\delta(x)$ will denote the value $\delta((T^i)_{-1})$, where $i = \iota(x \text{ wo } \underset{\sim}{T})$. Thus in Fig. 6-3, $\delta(h) = \delta((T^{(2,2)})_{-1}) = 3$, although $(\underset{\sim}{T}^{(3,1)})_{-1} = h$ and $\delta((T^{(3,1)})_{-1}) = 0$.

A vector of dimension $\mu(\underset{\sim}{T})$ whose components give the degrees of each of the nodes of a tree $\underset{\sim}{T}$ is called a <u>degree vector</u> of $\underset{\sim}{T}$. Since each leaf is of degree zero, $\lambda(\underset{\sim}{T}) = \sigma(\quad = 0)$. Moreover, the number of roots is equal to the number of nodes less the total of the degrees, that is, $\quad_1(\underset{\sim}{T}) = \nu(\quad) - \sigma(\quad)$. The maximum degree occurring in $\underset{\sim}{T}$ will be denoted by $\delta(\underset{\sim}{T})$, that is, $\delta(\underset{\sim}{T}) = (\quad)_1$.

<u>Example 6-8</u>. Determine the index such that the path $\underset{\sim}{T}$ is equal to a given argument and such that is of lowest possible rank in its product set. The process used (Program 6-19) will be to trace a path such that successive nodes agree with successive components of the argument.

<u>Program 6-19</u>. The vector represents the path currently traced (that is, $\underset{\sim}{T} = \quad^{\nu(\quad)}/\quad$) and j selects the roots of the successive component subtrees of $\underset{\sim}{T}$ for comparison (Step 11) with the next component of the argument. If agreement occurs, j is appended to (Step 12), j is reset to zero (Step 13), and d is respecified (Step 7) as the degree of the new final node of $\underset{\sim}{T}$. The variable d therefore represents the number of roots of the tree $\underset{\sim}{T}$, and is used (Step 9) to limit the range of the scan performed by the loop 9-11.

If the loop fails to find a root of $\underset{\sim}{T}$ agreeing with the argument, the branch to step 2 effects a retraction to the previous level (Step 4), with the new scan of the current level beginning with the next node following the one abandoned (see Step 3). Steps

5 and 6 are needed to specify d as the number of roots of the tree in the event that $\nu(\underset{\sim}{i}) = 0$, e.g., at the outset. The exit on step 2 occurs only if the tree possesses no path equal to the argument.

If each of the $\mu(\underset{\sim}{T})$ index vectors $\underset{\sim}{i}$ of a tree $\underset{\sim}{T}$ is listed together with its associated node $(\underset{\sim}{T})_{-1}$, the list determines the tree completely. Since the index vectors are, in general, of different dimensions, it is convenient to append null components[*] to extend each to the common maximum dimension $\nu(\underset{\sim}{T})$. They may then be combined in an index matrix of dimension $\mu(\underset{\sim}{T}) \times \nu(\underset{\sim}{T})$ which, together with the associated node vector, completely describes the tree. For example, the tree of Fig. 6-3 is described by the node vector n and Matrix I of Fig. 6-4(a).

[*] In the 1-origin indexing system used here it would be possible to use the numeric zero to represent the null. In 0-origin indexing, however, zeros occur as components of index vectors and must be distinguishable from the nulls used.

Certain information which is directly provided by the degree vector is provided only indirectly by the index matrix. Moreover, the degree vector and node vector together can, in certain arrangements, provide a complete description of the tree which is more compact, and for many purposes more convenient, than that provided by the node vector and index matrix. For these reasons, the degree vector will be annexed to the array of node

vector and index matrix as shown in Fig. 6-4(a) to form a full list matrix of the tree. The degree vector and node vector together will be called a list matrix.

Formally, the full list matrix of a tree T is defined as follows: is an index matrix of the tree, is the associated degree vector, and is the associated node vector. Thus for each , , and , where is the nonnull portion of , that is, . The corresponding list matrix is .

Since a full list matrix provides a complete description of a tree regardless of the order in which the nodes occur in the list, any column permutation (that is, any reordering among the rows) is also a list matrix. Two particular arrangements of the full list matrix are of prime interest because each possesses the following properties: (1) the nodes are grouped in useful ways, and (2) the list matrix (i.e., the degree vector and node vector) alone describes the tree without reference to the associated index matrix. They are called the full left list matrix and the full right list matrix, and are denoted by and , respectively. Figure 6-4 shows the full left and full right lists of the tree of Fig. 6-3.

The left list index matrix is left justified, that is, the null elements are appended at the right of each index. The rows are arranged in increasing order on the function

where $A \equiv \{\circ, 1, 2, \ldots, \delta(T)\}$. Equivalently, the rows are in increasing order on their values as decimal (or rather, $(\delta(T)+1)$-ary) numbers, after replacing each null by a zero.*

---

* These statements hold only for 1-origin indexing. In 0-origin indexing, $A \equiv \{\circ, 0, 1, 2, \ldots, \downarrow\delta(T)\}$, and in general $A \equiv \{\circ, (\Phi, \tilde{\phi}\downarrow\delta(T))\}$.

The right list index matrix $\lfloor$ is right justified and is ordered on the same function, namely $(\nu(A)\circ)\frac{\perp}{A\in}I^j$. From the example of Fig. 6-4(b) it is clear that the right list groups the nodes by levels, i.e., level j is represented by the infix $(\iota\downarrow\circ^k)/(\lfloor T)$, where $k = \mu_j(T)$, and $i = \sigma(\circ^{j-1}/\mu(T))$. In Fig. 6-4(b), for example, $\mu(T) = (3,7,8,5,3)$, and if $j = 3$, then $k = 8$, $i = 10$, and level j is represented by rows $i + 1 = 11$ to $i + k = 18$. The right list is therefore useful in executing processes (such as the pth degree selection sort of Sec. 8-5C) which require a scan of successive levels of the tree.

The left list groups the nodes by subtrees, i.e., any node $(T^k)_{-1}$ is followed immediately by the nodes of the subtree T_k. Formally, if $\lfloor = -2/\lfloor T$, and if $i = (I^k = \circ\circ)/I^k$, then the tree T_i is represented by the infix $(k\downarrow\circ^{\mu(T_i)})/\lfloor T$. In Fig. 6-4(a), for example, if $k = 16$, then $i = (2,2,2)$, $\mu(T_i) = 6$, and T_i is represented by rows 17 to 22 of $\lfloor T$. The left list is therefore useful in processes (such as the construction of a Huffman code and the evaluation of a compound statement (Sec. 7.2)) which requires a treatment of successive subtrees.

A matrix which forms the right list of some tree is said to be a well-formed right list. Since the ordering of the nodes in a right list of a given tree is unique, the right list of a given tree is unique. Conversely, any well-formed right list specifies a unique tree according to the algorithm of Program 6-20.

Identical remarks apply to the left list $\overline{\alpha}^2/\lfloor T$, except that Program 6-20 is replaced by Program 6-21. Moreover, the necessary and sufficient conditions for the well-formation of a left list are identical with those for a right list and are derived by virtually identical arguments. The case will be stated for the right list only.

If R is a well-formed right list representing a tree T, the number of roots $\rho_1(T) = \nu(R_1) - \sigma(R_1)$ must be strictly positive. Moreover, if $S = \overline{\alpha}^j /\!/ R$ is any suffix of R, then S is a right list of the tree obtained by deleting from T the first j nodes of the original list. For, such level-by-level deletion always leaves a legitimate tree with the degrees of the remaining nodes unchanged. Consequently, the number of roots determined by every suffix of R_1 must also be strictly positive. In other words, the root vector r defined by

$$r_{j+1} = \nu(\overline{\alpha}^j/R_1) - \sigma(\overline{\alpha}^j/R_1), \qquad j \in \left\{(0, \downarrow \nu(R_1)\right\},$$

must be strictly positive, that is, $(r > 0) = \epsilon$. The condition is also sufficient.

Sufficiency is easily established by induction on the column dimension of R. The condition is clearly sufficient for $\nu(R_1) = 1$.

Assume it sufficient for dimension $\nu(R_1) - 1$. If r, the root vector of R, is strictly positive, then $\overline{u}^1/r$, the root vector of $\overline{u}^1/R$, is also positive, and by hypothesis $\overline{u}^1/R$ represents a tree $\underset{\sim}{G}$ possessing r_2 roots. Moreover,

$$0 < r_1 = r_2 + (1 - R_1^1)$$

implies that $r_2 \geq R_1^1$, and the number of roots possessed by $\underset{\sim}{G}$ therefore fulfills the number of branches required by the added node R_2^1. A legitimate tree corresponding to R can therefore be formed by joining the last R_1^1 roots of $\underset{\sim}{G}$ to the node R_2^1.

Tests for well-formation can therefore be incorporated in any algorithm defined upon a right or left list matrix M by computing the components of the root vector r. The recursion $r_{i-1} = r_i + 1 - M_1^{i-1}$ is convenient in a backward scan of M, and the equivalent recursion $r_i = r_{i-1} - 1 + M_1^{i-1}$ serves for a forward scan. The starting condition for a forward scan is $r_1 = \nu(M_1) - \sigma(M_1)$, and for a backward scan is $r_{-1} = 1 - M_1^{-1}$. Since the criteria of well-formation are identical for right and left lists, a matrix may be characterized simply as well- or ill-formed.

The purpose served by the degree vector d in the description of a tree is sometimes served instead by the <u>reduced degree vector</u>[*]

---

[*] The negative of the reduced degree vector (that is, $\epsilon - d$) is also used. See, for example, Burks, et al (6-2).

$n = d - \epsilon$. This vector is somewhat more convenient in the analysis of well formation, since the expression for the root vector then simplifies to $r_{j+1} = -\sigma(\overline{u}^j/n)$.

The complete determination of the tree corresponding to a given list matrix L is best described as the determination of the associated index matrix I. For both left and right lists this is achieved by a single forward scan of the rows of L and of I.

For a right list R it is first necessary to determine r_1, the number of roots. The first r_1 components of R are then the roots of the tree in order, the next R_1^1 components of R are the second-level nodes reachable from the first root, and so forth. Programs 6-20 and 6-21 describe the process for a right list and a left list, respectively.

Program 6-20. In this exegesis, each node will be referred to by its index in the right list matrix R. In each execution of the main loop (8-16), the ith row of R is examined to determine the index vector of each node on the succeeding level which is directly reachable from it. The number of such nodes is controlled by the parameter d, initialized to the degree of the ith node by step 12. The index of the nodes reachable from node i is determined by j, which is incremented on step 14 as the index vector of each node is determined. The index vectors of the successive nodes reachable from node i have the final components $1,2,3,\ldots$, and each must be prefixed by the index vector of node i. This assignment is effected by the vector v which is initialized by the index vector of node i rotated left by one (step 11), and which is incremented by step 15 before each assignment occurring on step 16. At the outset, v is set to zero and d is set to the number of roots as determined by step 4.

Since j is, at step 10, equal to the number of roots augmented by the cumulative degrees of the first $i-1$ nodes, then $r_i = j - i + 1$ and the exit on step 10 therefore occurs always and only in the event of ill-formation. Alternatively, the test can be viewed as an assurance that each row of the matrix I is specified before it is itself used in specification.

When step 5 is first reached, the index matrix I is complete, but is expressed in 1-origin indexing with zeros representing the null elements. Steps 5-7 translate the matrix to the origin ϕ, and mask in the necessary null elements.

Program 6-21. In this exegesis, each node is referred to by its index in the left list matrix L. The index vectors I^j are determined in order under control of the parameter j. The loop (5-18) traces a continuous path through the tree, determining the index of each successive node of the path by rotating the index of the preceding node (step 17) and adding one to the last component (step 13), and maintaining in the connection vector c a record c_{i+1} of the index j of the successor of node i in the path traced. The path is interrupted by the occurrence of a leaf (that is, $L_1^j = 0$ on step 18), and the degree vector L_1 is then scanned by the loop (19-20) to determine the index i of the last preceding node whose branches remain incompleted. Steps 21-22 then respecify r as the index vector of the node following node i in the path last traced, and step 23 decrements the component L_1^i of the degree vector. The branch from step 19 to step 21 occurs at the completion of each

rooted subtree. The test for well-formation is the same as applied to the right list in Program 6-20, except that the notation for the relevant parameters differs. The concluding operations (6-9) include left justification on step 6.

Any unary operation defined upon each of the nodes of a tree can be generalized immediately to the entire tree in a manner analogous to the generalization to vectors and matrices. If, for example, $\underset{\sim}{U}$ is a logical tree, then $\sim\underset{\sim}{U}$ is determined by node-by-node negation. Binary operations are likewise generalized to any pair of compatible trees, e.g., $\underset{\sim}{X} \times \underset{\sim}{Y}$, $\underset{\sim}{X} \vee \underset{\sim}{Y}$, and $(\underset{\sim}{A} \neq \underset{\sim}{B})$.

Scan operations defined upon vectors may also be extended to trees. For example, $\cup\!\!\int\underset{\sim}{X}$ defines a logical tree whose unit nodes designate the occurrence of maxima in $\underset{\sim}{X}$, where the maximization is restricted to nodes corresponding to the nonzero nodes of the compatible logical tree $\underset{\sim}{U}$. The notation $\odot/\!/\underset{\sim}{T}$ will denote the application of the associative binary operator $\odot$ to the nodes of $\underset{\sim}{T}$ in right list order (i.e., down successive levels), and $\odot/T$ will denote the same application in left list order (i.e., across paths). If the operator is symmetric (i.e., its operands commute), then $\odot/\!/\underset{\sim}{T} = \odot/\underset{\sim}{T}$. The notation $\sigma(\underset{\sim}{T})$ will be adopted for the sum weight $+/\underset{\sim}{T}$.

The selection operations defined upon vectors (compression, expansion, mesh, and mask) can also be extended by adopting the following definition of the compress operation $\underset{\sim}{U}/\underset{\sim}{T}$. The statement $\underset{\sim}{P} \leftarrow\!\!- \underset{\sim}{U}/\underset{\sim}{T}$ implies that the nodes of $\underset{\sim}{P}$ are those nodes of $\underset{\sim}{T}$ for

which the corresponding nodes of $\underset{\sim}{U}$ are unity, and that the structure of $\underset{\sim}{P}$ is determined as follows: if x and y are any two nodes of $\underset{\sim}{P}$, then y belongs to the subtree of $\underset{\sim}{P}$ rooted at x if and only if y belongs to the subtree of $\underset{\sim}{T}$ rooted at x. If, for example, $\underset{\sim}{U}$ is the tree of Fig. 6-5(a) and $\underset{\sim}{T}$ is the tree of Fig. 6-3, then $\underset{\sim}{U}/\underset{\sim}{T}$ is the tree of Fig. 6-5(b).

The compress operation is best executed upon the left list because of the grouping by subtrees. Program 6-22 gives a suitable algorithm which also serves as a formal definition of the compress operation.

Program 6-22. The vector a is specified as the node vector of the left list of the controlling logical tree $\underset{\sim}{U}$, and controls the subsequent process. Step 4 determines j as the index of the first zero component of a. Steps 6 and 7 then delete the corresponding nodes of a and of the left list of $\underset{\sim}{T}$, but only after step 5 has determined d as the change in degree which this deletion will occasion to the root of the smallest subtree containing the deleted node. Steps 8-11 perform a backward scan of the degree vector to determine j as the index of the root of the subtree, and step 12 effects the requisite change in its degree. The exit on step 9 occurs only if the node deleted is a root of the original tree, in which event no change is produced in the degree of any other node.

As in the case of vectors, the remaining selection operations are defined directly in terms of compression. The uses of the selection operations are analogous to their uses in vectors; for

example, $(\underset{\sim}{P} = \underset{\sim}{T})/\underset{\sim}{T}$ denotes a compressed tree formed from those nodes of $\underset{\sim}{T}$ which agree with $\underset{\sim}{P}$.

The symbol $^{u}\underset{\sim}{E}$ will denote a special logical tree (called a level tree) each of whose nodes on level j is equal to u_j, where u is a logical vector. The tree $^{u}\underset{\sim}{E}$ has a height equal to $\nu(u)$ and its structure is otherwise determined by compatibility with associated operands. The trees $^{t}\underset{\sim}{E}$, $^{\alpha j}\underset{\sim}{E}$, and $^{\omega j}\underset{\sim}{E}$ are called full, prefix, and suffix trees, respectively.

Compression by a level tree $^{u}\underset{\sim}{E}$ is called truncation and will be denoted by $u/\underset{\sim}{T}$ as well as by $^{u}\underset{\sim}{E}/\underset{\sim}{T}$. Truncation effects the deletion of specified levels of a tree.

Permutation operations will not be extended directly to trees, but may be applied indirectly as follows. A tree $\underset{\sim}{T}$ comprises the subtrees rooted at $\underset{\sim}{T}^{j}$ for $j \in \{(1, \rho_1(T))\}$, and can therefore be considered as a vector v, each of whose components is a rooted tree. The statement $v \leftarrow\!\!- \underset{\sim}{T}$ will therefore be understood to imply that v_j is the subtree rooted at $\underset{\sim}{T}^{j}$. Analogously, the statement $\underset{\sim}{T} \leftarrow\!\!- v$ can be used to specify a tree by a vector. These transitions between trees and vectors provide a convenient means of designating subtrees. Thus the sequence

$$v \leftarrow\!\!- \underset{\sim}{T}_1$$

$$\underset{\sim}{P} \leftarrow\!\!- v_j$$

specifies $\underset{\sim}{P}$ as the jth (rooted) subtree of the tree $\underset{\sim}{T}_1$. Moreover, these transitions allow permutations and other vector operations

to be applied indirectly to trees. Program 6-23 illustrates the permutation of the tree $\underset{\sim}{T}$ of Fig. 6-3 to produce the tree S of Fig. 6-6.

Homogeneous trees. If, for each level of a tree, all nodes on that level are of the same degree, the tree is said to be homogeneous. The structure of a homogeneous tree $\underset{\sim}{T}$ is completely characterized by the number of its roots and by the degree of each level j. These quantities will be combined into a single descriptor, the dispersion vector $\nu(\underset{\sim}{T})$ defined as follows: $\nu_1(\underset{\sim}{T})$ is the number of roots of $\underset{\sim}{T}$ and $\nu_j(\underset{\sim}{T})$ is the (common) degree of level $(j-1)$, for $j \in \{(2,\nu(\underset{\sim}{T}))\}$. The component $\nu_j(\underset{\sim}{T})$ is called the dispersion of level j, and specifies the number of nodes of level j reachable from each node of level $(j-1)$. All maximal paths of a homogeneous tree are of length $\nu(\underset{\sim}{T})$ and clearly, $\nu(\underset{\sim}{T}) = \nu(\nu(\underset{\sim}{T}))$. Figure 6-7 shows a collection of homogeneous trees and their associated dispersion vectors.

A tree $\underset{\sim}{T}$ for which $\nu(\underset{\sim}{T}) = m\epsilon$ is called a uniform m-way tree, and a tree for which $\nu_1(\underset{\sim}{T}) = 1$ and $\epsilon^{-1}/\nu(\underset{\sim}{T}) = m\epsilon$ is called a uniform m-way rooted tree.

The jth component of the moment vector of a homogeneous tree is clearly equal to the product of the first j components of the dispersion vector, that is, $\mu_j(\underset{\sim}{T}) = \times/\nu^j/\nu(\underset{\sim}{T})$, for $j \in \{(1,\nu(T))\}$. The dispersion vector is in turn uniquely determined by the moment vector according to the relation

$$\nu(\underset{\sim}{T}) = \mu(\underset{\sim}{T}) \div /\epsilon, \epsilon^{-1}, \downarrow \mu(\underset{\sim}{T})/.$$

The total number of nodes is given by $\mu(\underset{\sim}{T}) = \sigma(\nu(\underset{\sim}{T}))$, and it is easily shown that $\mu(\underset{\sim}{T}) = y \perp y$, where $y_{-1} = \nu(\underset{\sim}{T})$.

Although all of the operations on general trees clearly apply to homogeneous trees, not all of them produce homogeneous trees. Because of the importance of homogeneous trees, a closed system of operations will be defined upon them. It should also be remarked that the logical compression operation $\underset{\sim}{U}/\underset{\sim}{T}$ allows any tree $\underset{\sim}{P}$ to be represented by a homogeneous tree $\underset{\sim}{T}$ and a compatible (and therefore homogeneous) logical tree $\underset{\sim}{U}$.

Truncation as defined produces a homogeneous tree when applied to a homogeneous tree, and will therefore be incorporated in the system. If $\underset{\sim}{T}$ is homogeneous and if $\underset{\sim}{P} = u/\underset{\sim}{T}$, then clearly $\mu(\underset{\sim}{P}) = u/\mu(\underset{\sim}{T})$, and $\nu(\underset{\sim}{P})$ may be obtained from $\mu(\underset{\sim}{P})$. The converse operation is called <u>insertion</u>, is denoted by

$$\underset{\sim}{P} \longleftarrow \backslash \underset{\sim}{T}, u, \underset{\sim}{Q} \backslash,$$

and implies that $u/\underset{\sim}{P} = \underset{\sim}{T}$, and $u/\underset{\sim}{P} = \underset{\sim}{Q}$. Clearly $\mu(\underset{\sim}{P}) = \backslash \mu(\underset{\sim}{T}), u, \mu(\underset{\sim}{Q}) \backslash$, and compatibility requires that $\mu(\underset{\sim}{P})$ define an integral dispersion vector. For example, the supposed moment vector $m = (2,5,15)$ defines the supposed dispersion vector $n = (2,2.5,3)$ which is not integral, and m is therefore not a legitimate moment vector.

An operation which in level j deletes a specified group of nodes from each of the groups sharing a common root in level j - 1, and also removes the entire subtrees rooted in the deleted nodes, will be called <u>pruning of the jth level</u>. Thus the pruned tree agrees with the original in the first j - 1 levels, and the dispersion vectors agree in all but the jth component. More precisely, the statement

$$\underset{\sim}{P} \longleftarrow u /\!\!/_{j} \underset{\sim}{T}$$

is defined for a compatible vector u such that $\nu(u) = \nu_j(\underset{\sim}{T})$ and implies that $u^{j-1}/\underset{\sim}{P} = u^{j-1}/\underset{\sim}{T}$, and that $(\overline{u}^{j-1}/\underset{\sim}{P})_k = (\overline{u}^{j-1}/\underset{\sim}{T})_j$, for $k \in \{(1, \sigma(u))\}$, where $j = (u/\iota)_k$. Clearly, $\nu_j(\underset{\sim}{P}) = \sigma(u)$, and $\overline{\epsilon}^j/\nu(\underset{\sim}{P}) = \overline{\epsilon}^j/\nu(\underset{\sim}{T})$. In Fig. 6-7, for example, $\underset{\sim}{T} = \overline{u}/\underset{3}{\sim}\underset{\sim}{P}$, where $u = (1,0,1)$. Moreover, $\underset{\sim}{Q} = \overline{u}^2/(u/\underset{3}{\sim}\underset{\sim}{P})$.

The converse operation is called <u>grafting at the jth level</u> and is denoted by the statement

$$\underset{\sim}{P} \longleftarrow \Vert \underset{j}{\sim}\underset{\sim}{T}, u, \underset{\sim}{Q} \Vert,$$

which implies that $\overline{\iota}/\underset{j}{\sim}\underset{\sim}{P} = \underset{\sim}{T}$, and that $\overline{\epsilon}^{j-1}/(\iota/\underset{j}{\sim}\underset{\sim}{P}) = \underset{\sim}{Q}$. The compatibility requirements are

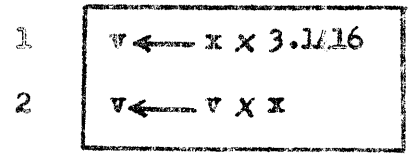
$$\overline{u}^1/\nu(\underset{\sim}{Q}) = \overline{\epsilon}^j/\nu(\underset{\sim}{T}),$$

$$\nu_j(\underset{\sim}{T}) = \sigma(\overline{u}),$$

and

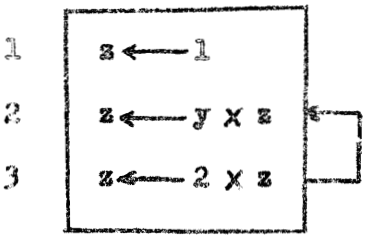
$$\nu_1(\underset{\sim}{Q}) = \sigma(\overline{u}) \times \nu_{j-1}(\underset{\sim}{T}).$$

Clearly, $\nu_j(\underset{\sim}{P}) = \nu(u)$, and $\overline{\epsilon}^j/\nu(\underset{\sim}{P}) = \overline{\epsilon}^j/\nu(\underset{\sim}{T})$. The operation is illustrated by Fig. 6-7.



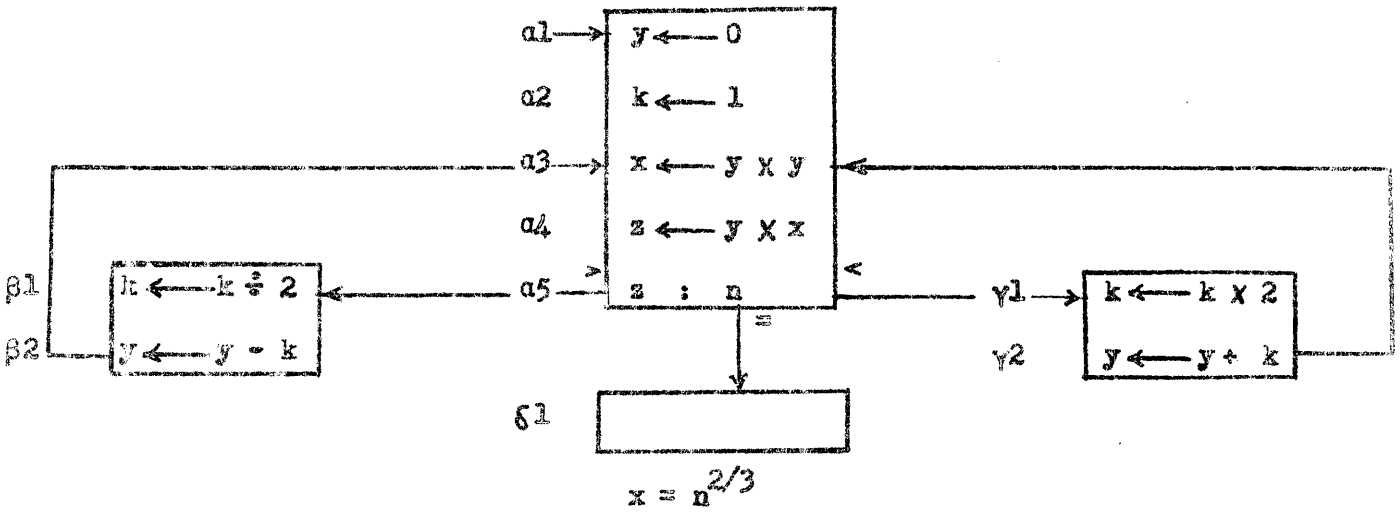
Finite program

Program 6-1

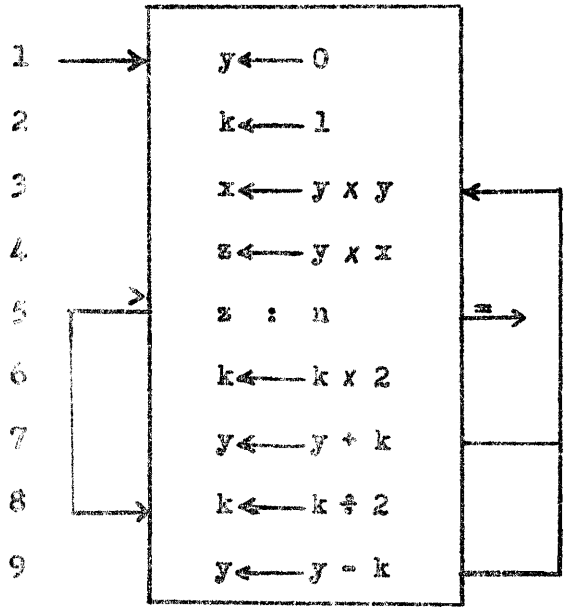


Infinite program

Program 6-2

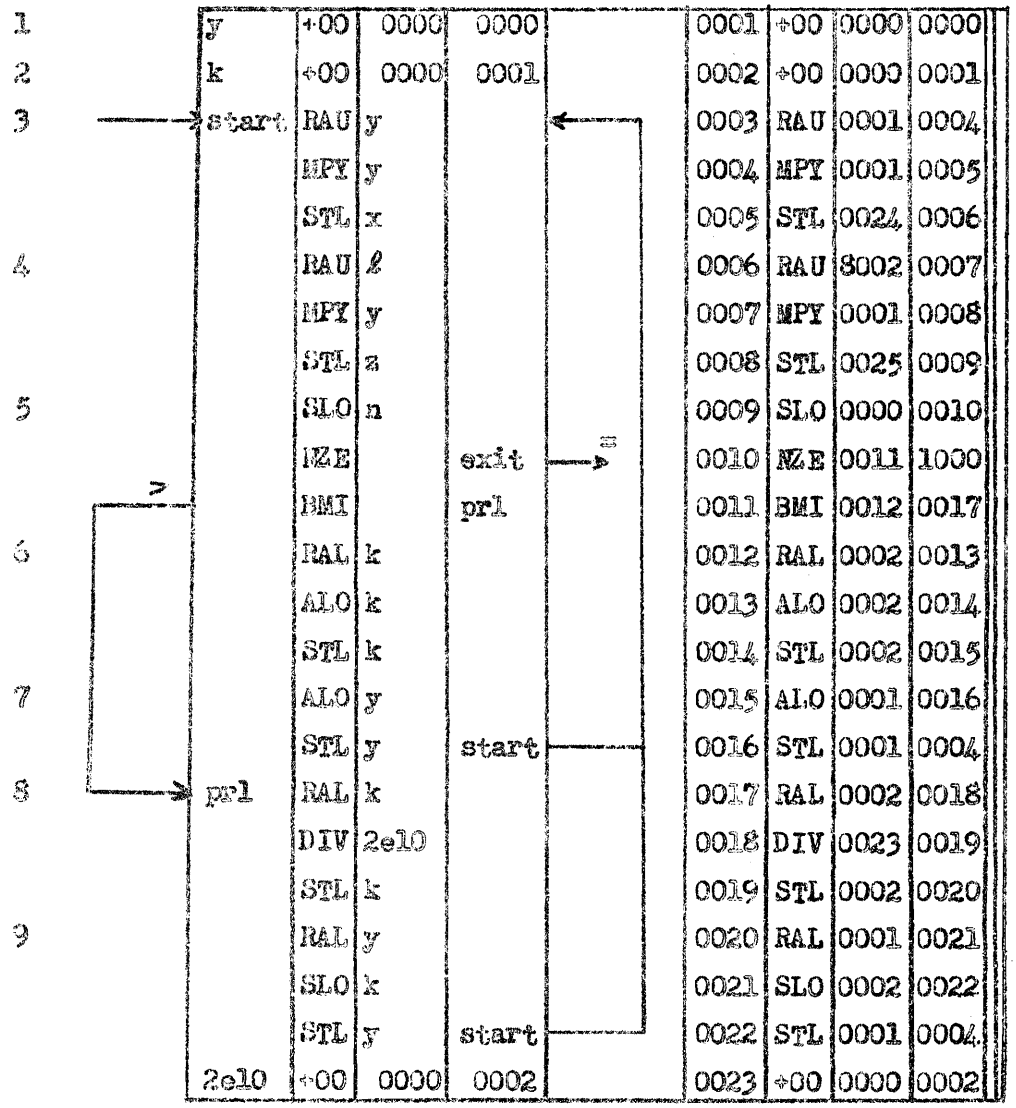

α1
y ← 0
α2
k ← 1
α3
x ← y x y
α4
z ← y x x
α5
z : n
>
<
=
β1
k ← k ÷ 2
β2
y ← y - k
γ1
k ← k x 2
γ2
y ← y + k
δ1
x = n^2/3

Program 6-3



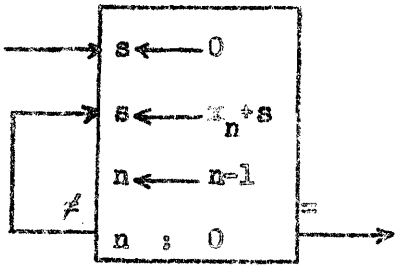
$x = n^{2/3}$

Program 6-4



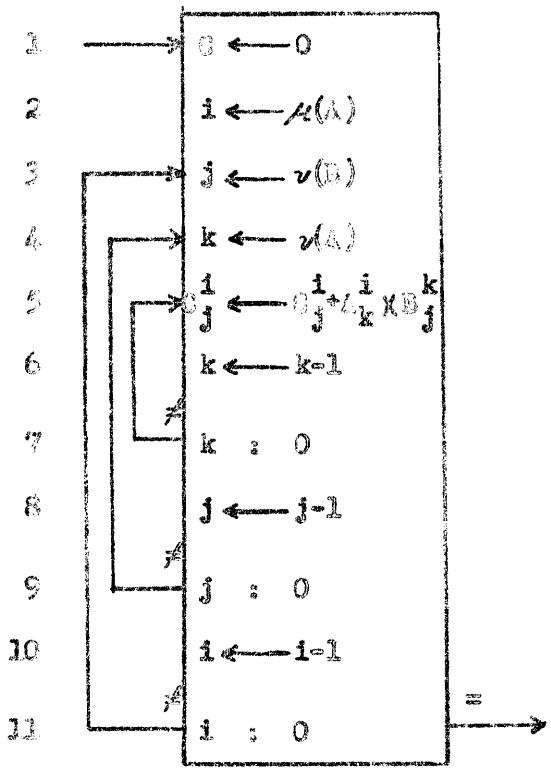
Computer program for $x = n^{2/3}$

Program 6-5



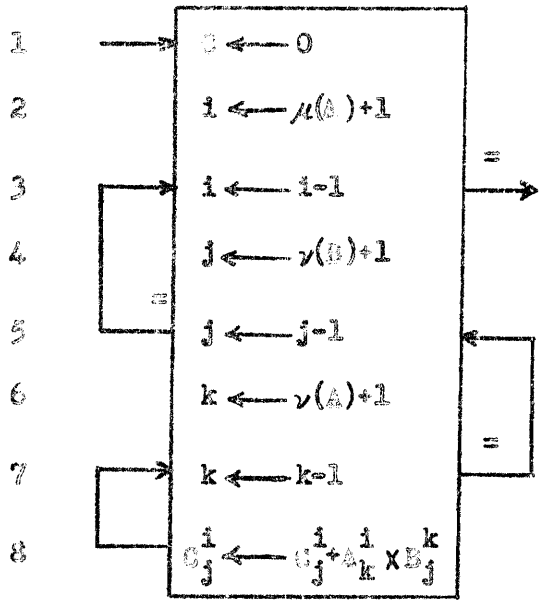
Summation

Program 6-6



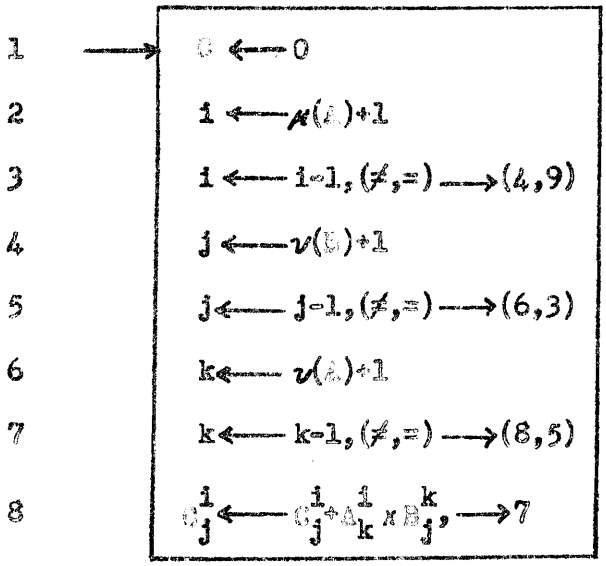
Matrix Multiplication

Program 6-7



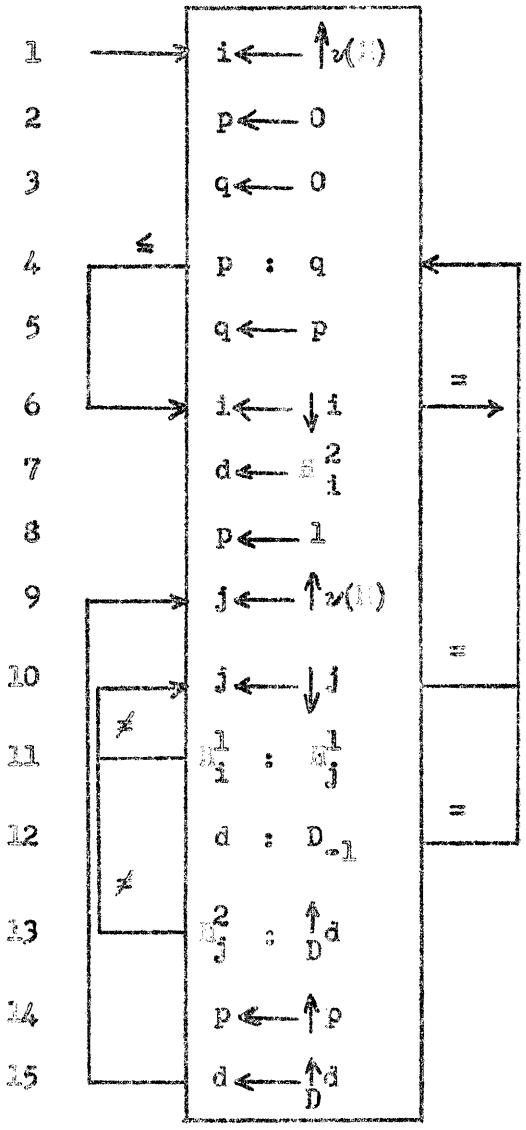
Matrix multiplication using leading decisions

Program 6-8



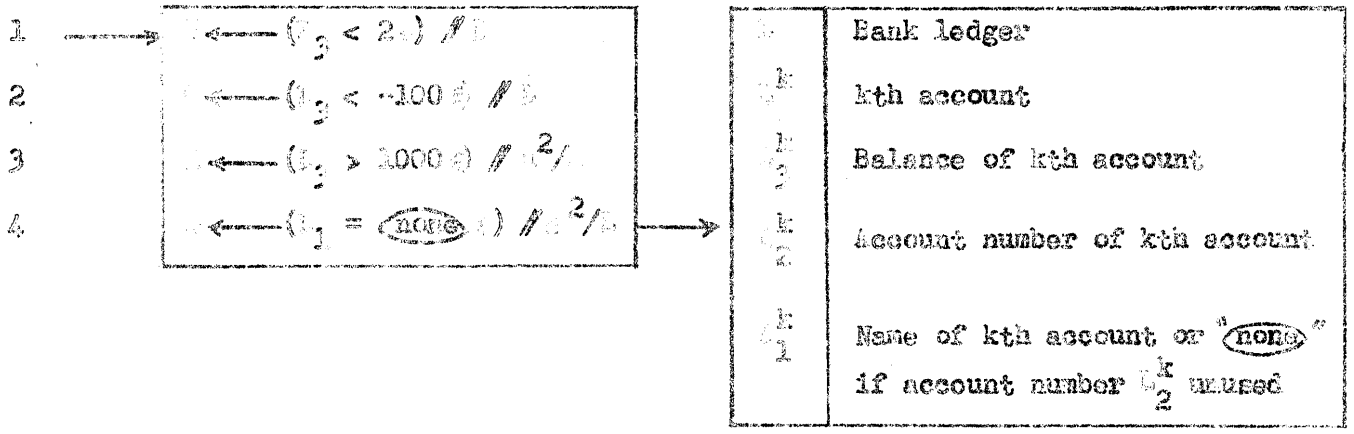
A reformulation of Program 6-8 using an algebraic statement of the branching

Program 6-9



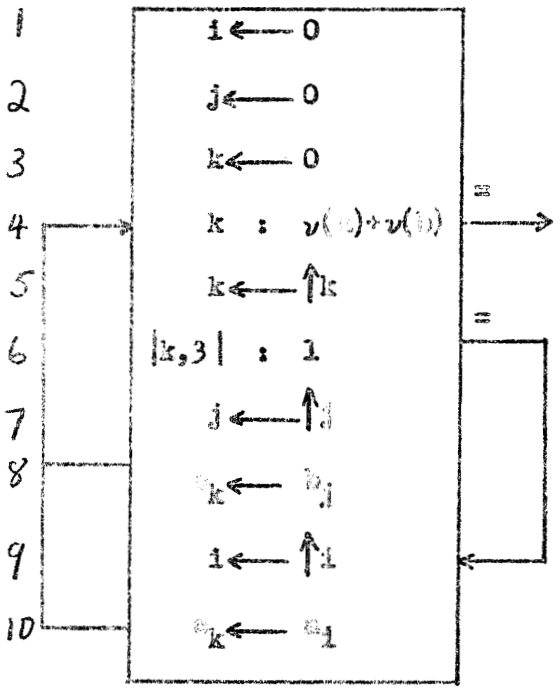
Program for Example 6-1

Program 6-10



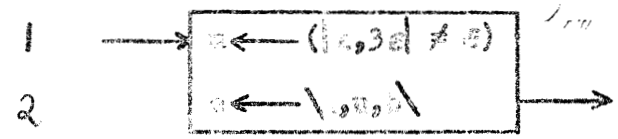
Selection on bank ledger (Example 6-2)

Program 6-11



(a)

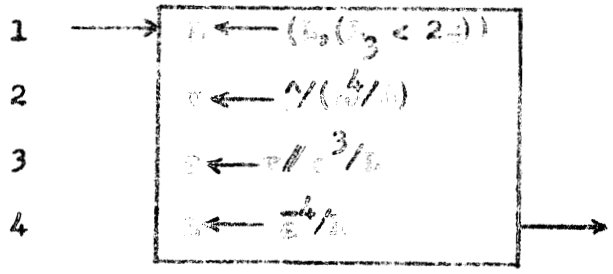
a, b	Given vectors
a	a = $(a_1, b_1, b_2, a_2, b_3, b_4, a_3, \cdots)$
i	index of a
j	index of b
k	index of c
n	n = (0,1,1,0,1,1,0,...)



(b)

Interfiling program

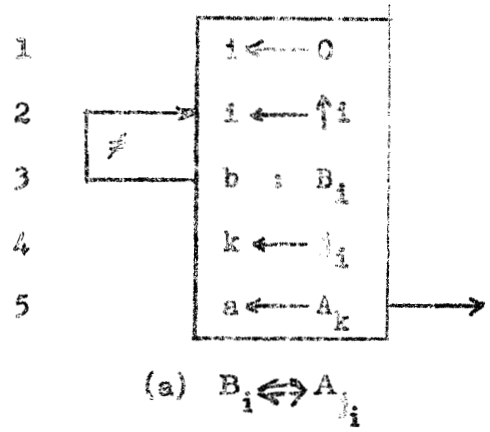
Program 6-12



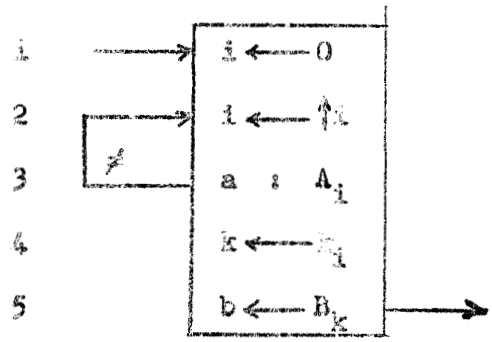
	Bank ledger
L_k	kth account
L_{k3}	Balance of kth account
L_{k2}	Account # of kth account
L_{k1}	Name of kth account
L_{k4}, L_{k5}, L_{k6}	Logical variables giving status (1 if balance ≥ 2) for three preceding periods

Program for Example 6-3

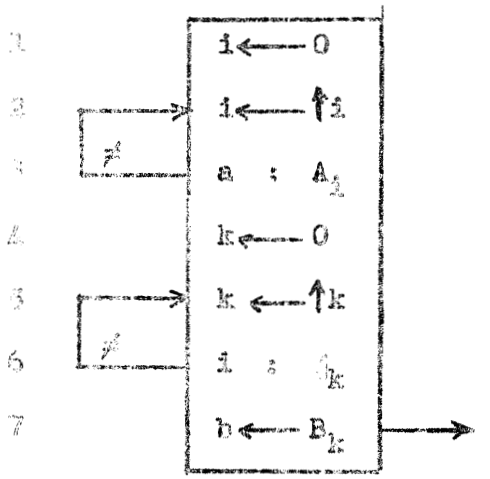
Program 6-13



(a) $B_i \Leftrightarrow A_{j_i}$



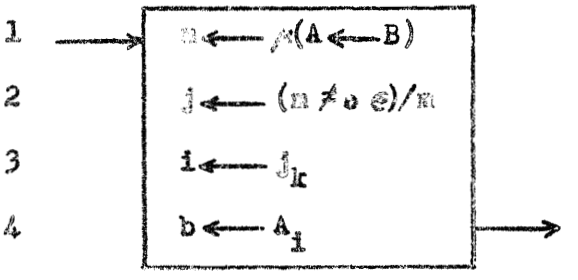
(b) $A_i \Leftrightarrow B_{j_i}$



(c) $A_i \Leftrightarrow B_{j_i}$

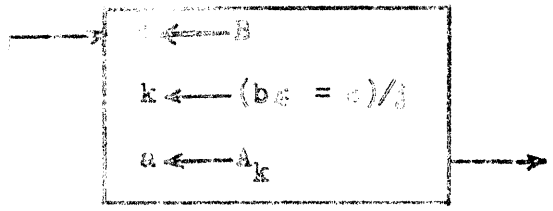
A	Set of correspondents in Program (a) and set of arguments in Programs (b), (c).
B	Set of arguments in Program (a) and set of correspondents in Programs (b), (c).
j, k	Mutually inverse permutation vectors, that is, $j_k = k_j = i$

Mapping defined by a permutation vector
Program 6-14

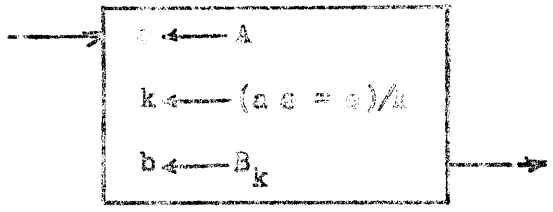


Determination of $b = (B \wedge A)_k$

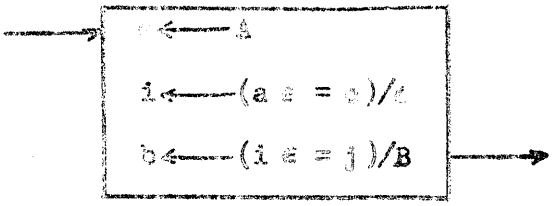
Program 6-15



(a) $B_i \Leftrightarrow A_{j_i}$



(b) $A_i \Leftrightarrow B_{k_i}$

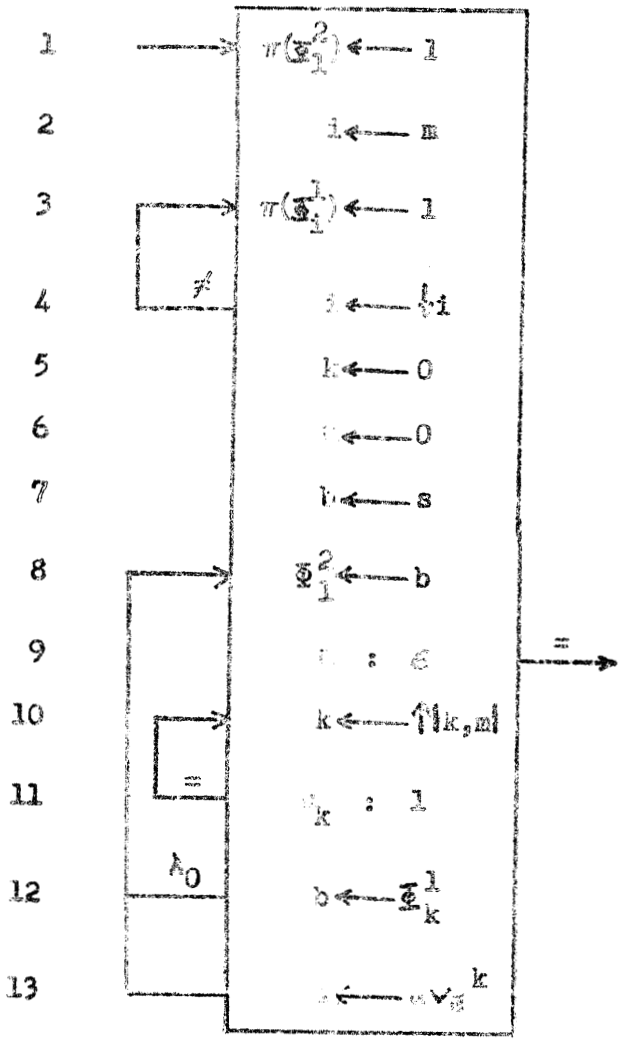


(c) $A_i \Leftrightarrow B_{\ell_i}$

A	Set of correspondents in Program (a) and set of arguments in Programs (b), (c).
B	Set of arguments in Program (a) and set of correspondents in Programs (b), (c).
j, k	Mutually inverse permutation vectors, that is, $j_k = k_j = \iota$

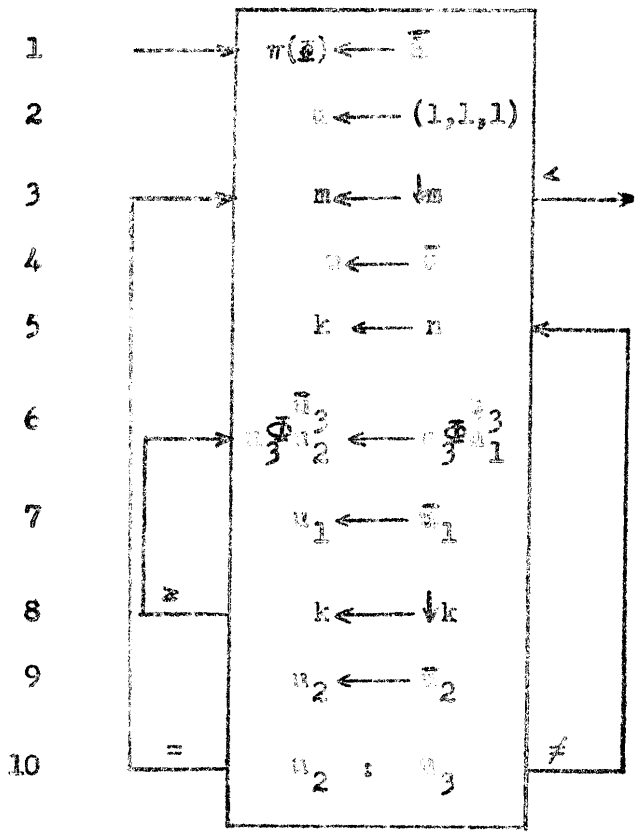
Reformulation of Programs 6-14(a-c)

Program 6-16



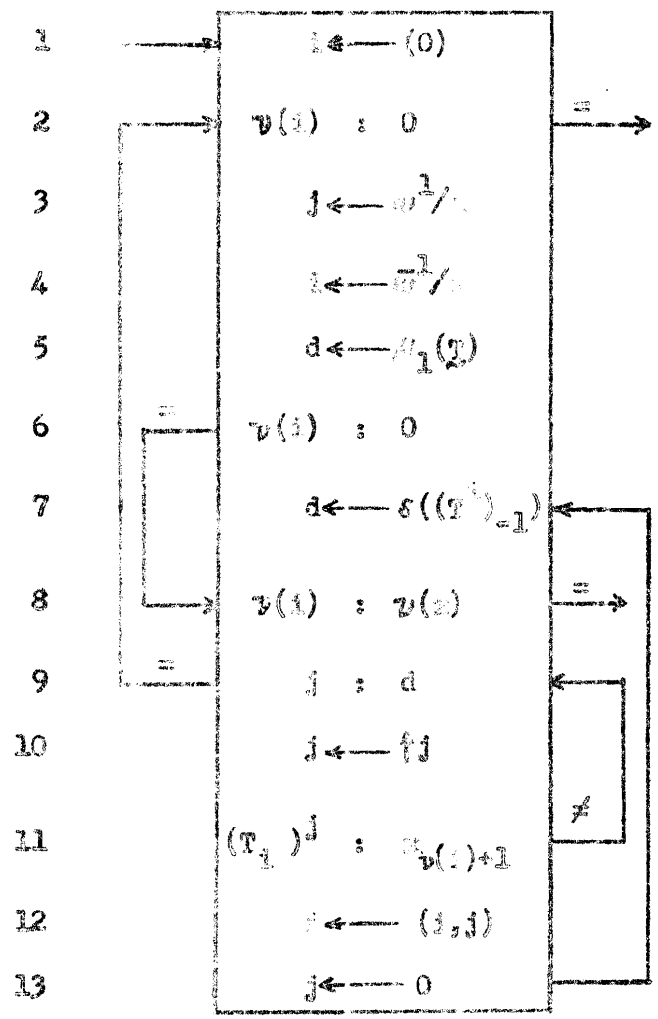
Φ^1	Row of output files of dimension m with terminal partition λ_1
Φ^2_1	Output file
s	Identification quantity for output file
	Files w/ Φ^1 are exhausted
b	Item to be recorded

Program for Example 6-6
Program 6-17



0-origin indexing	
Φ	File array of dimension 2×2; original input Φ^0; original output Φ^1.
u	Control vector
u_1	Column index of input file
u_2	Column index of output file
u_3	Row index of current input file, and direction of read and record
n	Number of items per file
m	Required number of merges

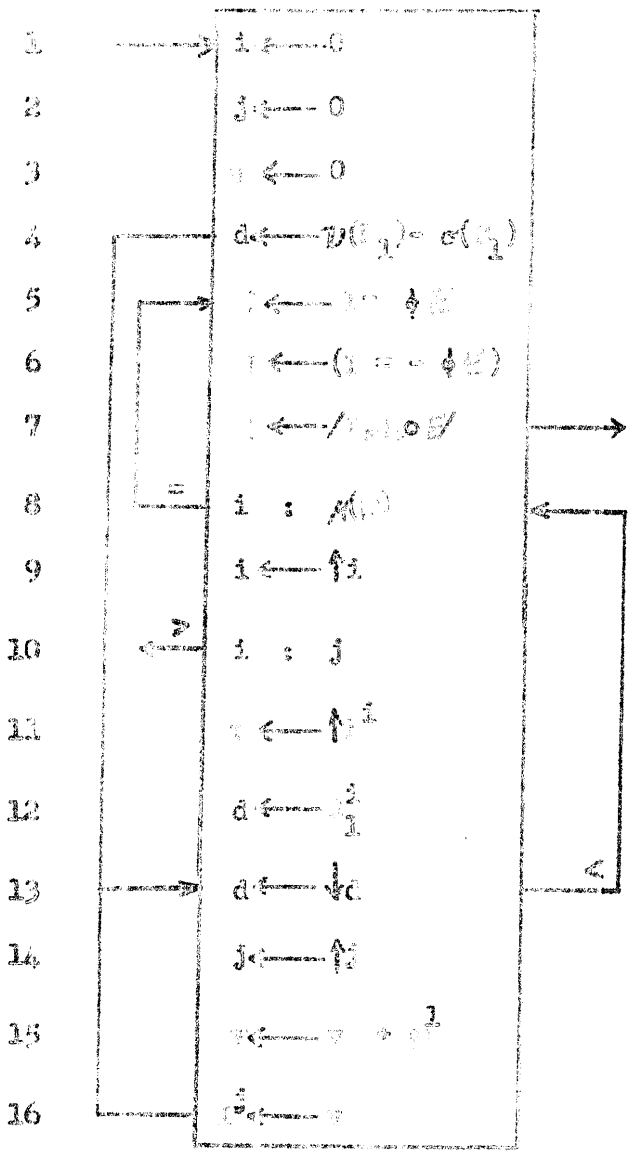
Program for Example 6-7
Program 6-18



$\underset{\sim}{T}$	Given tree
x	Given path vector
i	Path index vector to be determined
$\mu(\underset{\sim}{T})$	Moment vector of $\underset{\sim}{T}$
$\nu_0(\underset{\sim}{T})$	Number of roots of $\underset{\sim}{T}$

Determination of i such that $\underset{\sim}{T}^i = x$

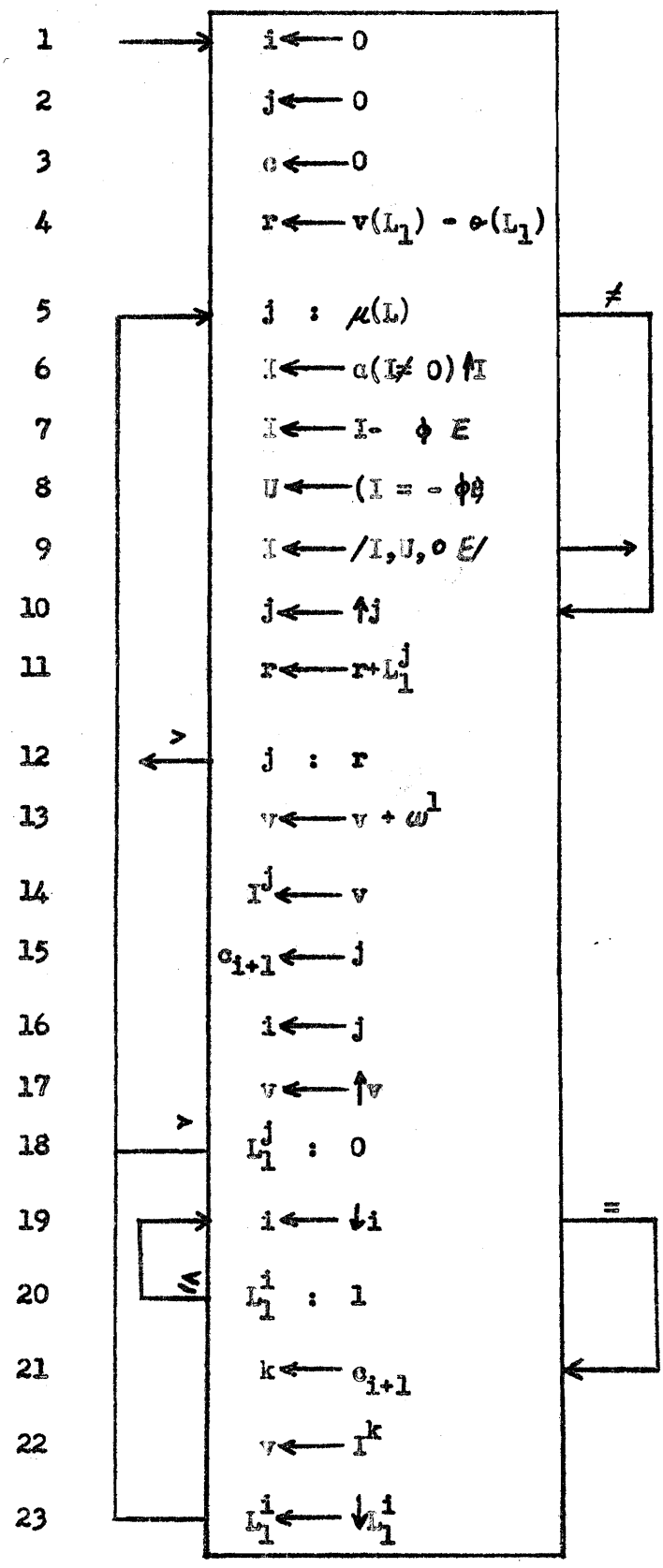
Program 6-19



	Right list of $\underset{\sim}{T}$
	Right index matrix of $\underset{\sim}{T}$
i	Index of row of currently examined
j	Row index of node reachable from node i
	Current index vector
ϕ	Origin of index set in use

Determination of the index matrix $\underset{\sim}{I}$ associated with a right list matrix $\underset{\sim}{R}$.

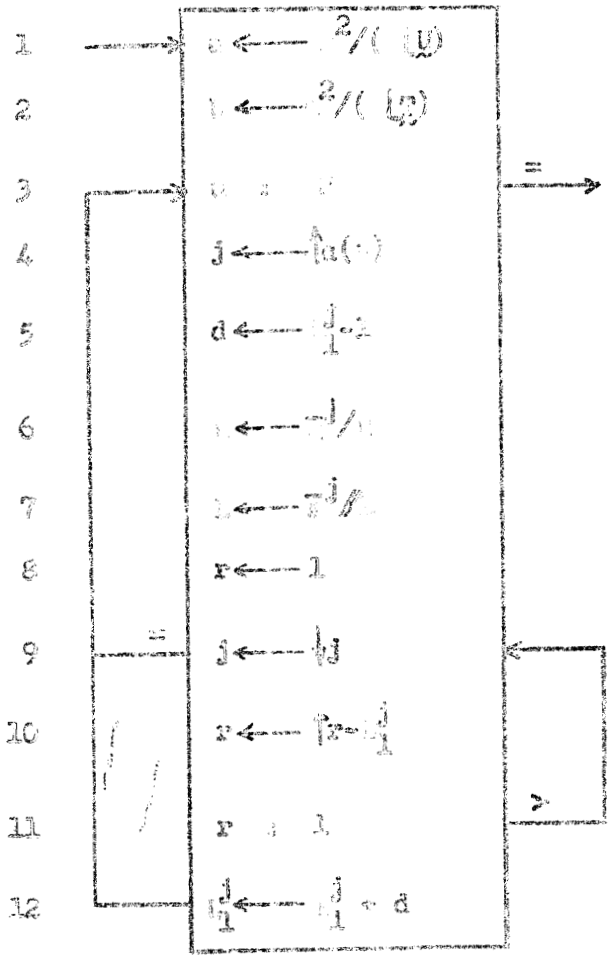
Program 6-20



L	Left list of T
I	Left index matrix of T
j	Index of row of I being determined
i	Index of path node preceding node j in current path (Step 16), or index of last previous node whose branches remain unexhausted (Step 21).
σ_{i+1}	Index of node following node i in last path traced from i.
r	Parameter for testing well-formation
v	Current index vector
φ	Origin of index set in use

Determination of the index matrix I associated with a left list matrix L.

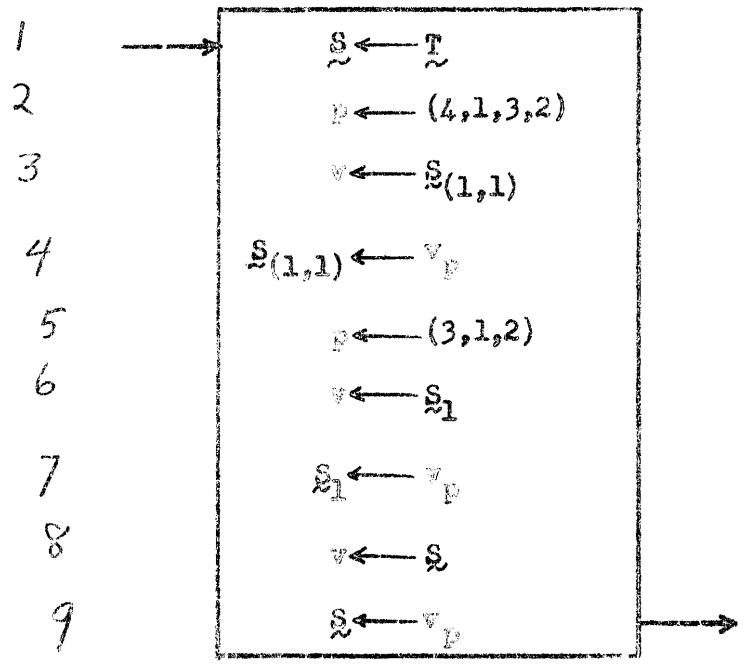
Program 6-21



	Left node vector of $\mathcal{U}$
	Left list of $\mathcal{T}$
j	Index of first zero of (Steps 4-8). Index of root of smallest subtree containing deleted node (Step 12).
d	Change of degree caused by deletion of node j.
r	Number of roots indicated by infix $(\omega^{j} \wedge \overline{\omega}^{k})/\mathcal{U}$, where j is initial value and k+1 is current value of j.

Determination of the left list $\mathcal{L} = \mathcal{L}^{2}/\mathcal{U}(\mathcal{U}/\mathcal{T})$

Program 6-22

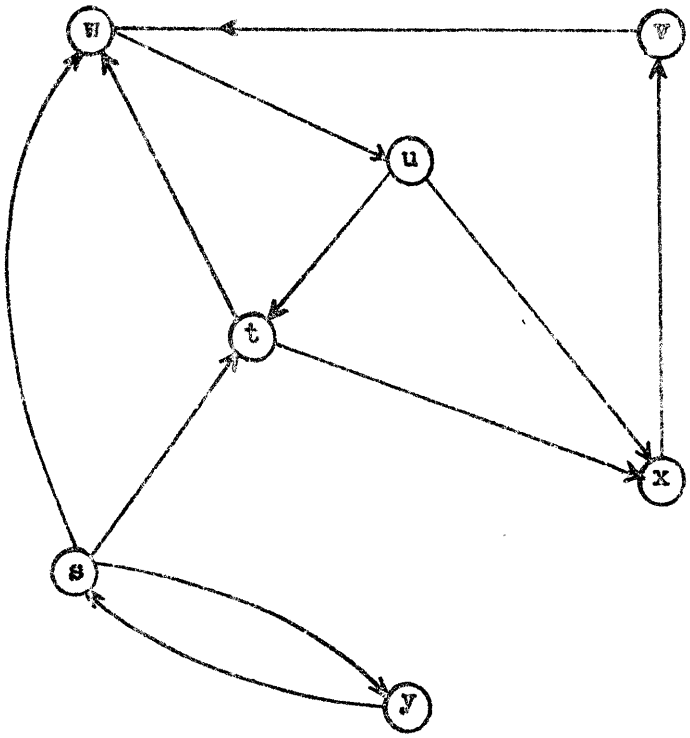


Permutation of $\underset{\sim}{T}$ (Fig. 6-3) to yield $\underset{\sim}{S}$ (Fig. 6-6)

Program 6-23

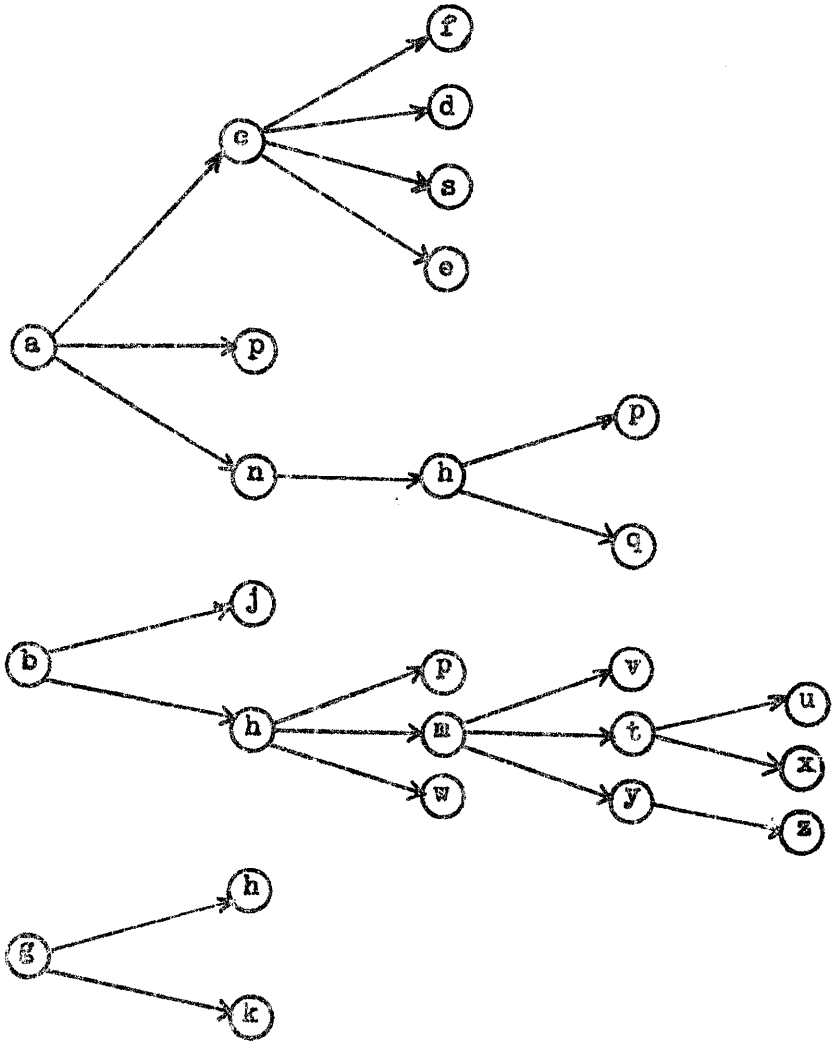
Vector as a directed graph

Fig. 6-1



A general directed graph

Fig. 6-2



A general triply-rooted tree with $\mathcal{D}(\underset{\sim}{T}) = 5$, $\lambda(\underset{\sim}{T}) = 12$, $\mu(\underset{\sim}{T}) = (3,7,8,5,3)$, and $\mu(\underset{\sim}{T}) = 26$

Fig. 6-3

	n	r	l	
1	3	a	1 0 0 0 0	1
2	4	c	1 1 0 0 0	2
3	0	f	1 1 1 0 0	3
4	0	d	1 1 2 0 0	4
5	0	s	1 1 3 0 0	5
6	0	e	1 1 4 0 0	6
7	0	p	1 2 0 0 0	7
8	1	n	1 3 0 0 0	8
9	2	h	1 3 1 0 0	9
10	0	p	1 3 1 1 0	10
11	0	q	1 3 1 2 0	11
12	2	b	2 0 0 0 0	12
13	0	j	2 1 0 0 0	13
14	3	h	2 2 0 0 0	14
15	0	p	2 2 1 0 0	15
16	3	m	2 2 2 0 0	16
17	0	v	2 2 2 1 0	17
18	2	t	2 2 2 2 0	18
19	0	u	2 2 2 2 1	19
20	0	x	2 2 2 2 2	20
21	1	y	2 2 2 3 0	21
22	0	z	2 2 2 3 1	22
23	0	w	2 2 3 0 0	23
24	2	g	3 0 0 0 0	24
25	0	h	3 1 0 0 0	25
26	0	k	3 2 0 0 0	26

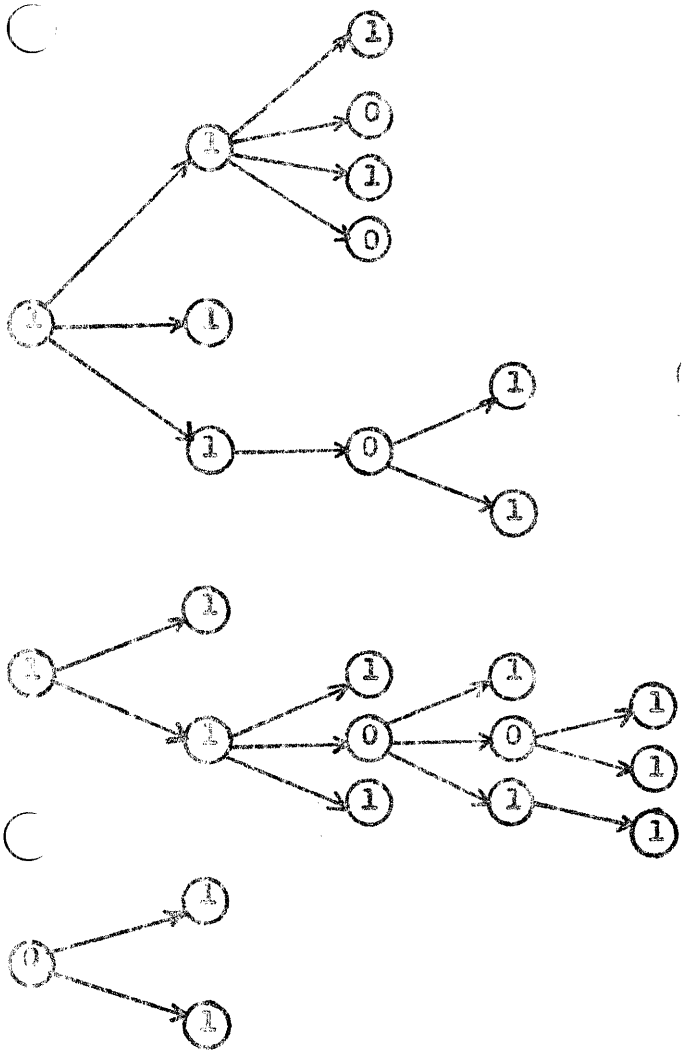
Full left list matrix $\tilde{L}T$
(a)

	n	r	l	
1	3	a	0 0 0 0 1	1
2	2	b	0 0 0 0 2	2
3	2	g	0 0 0 0 3	3
4	4	e	0 0 0 1 1	4
5	0	p	0 0 0 1 2	5
6	1	n	0 0 0 1 3	6
7	0	j	0 0 0 2 1	7
8	3	h	0 0 0 2 2	8
9	0	h	0 0 0 3 1	9
10	0	k	0 0 0 3 2	10
11	0	f	0 0 1 1 1	11
12	0	d	0 0 1 1 2	12
13	0	s	0 0 1 1 3	13
14	0	e	0 0 1 1 4	14
15	2	h	0 0 1 3 1	15
16	0	p	0 0 2 2 1	16
17	3	m	0 0 2 2 2	17
18	0	w	0 0 2 2 3	18
19	0	p	0 1 3 1 1	19
20	0	q	0 1 3 1 2	20
21	0	v	0 2 2 2 1	21
22	2	t	0 2 2 2 2	22
23	1	y	0 2 2 2 3	23
24	0	u	2 2 2 2 1	24
25	0	x	2 2 2 2 2	25
26	0	z	2 2 2 3 1	26

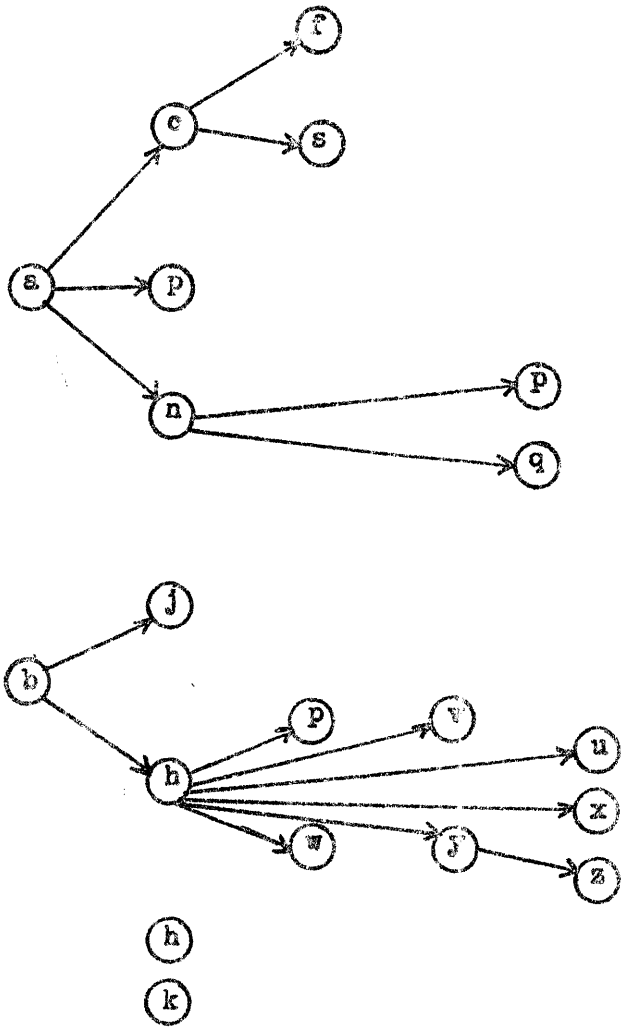
Full right list matrix $\tilde{J}T$
(b)

Full list matrices of the tree of Fig. 6-3

Fig. 6-4



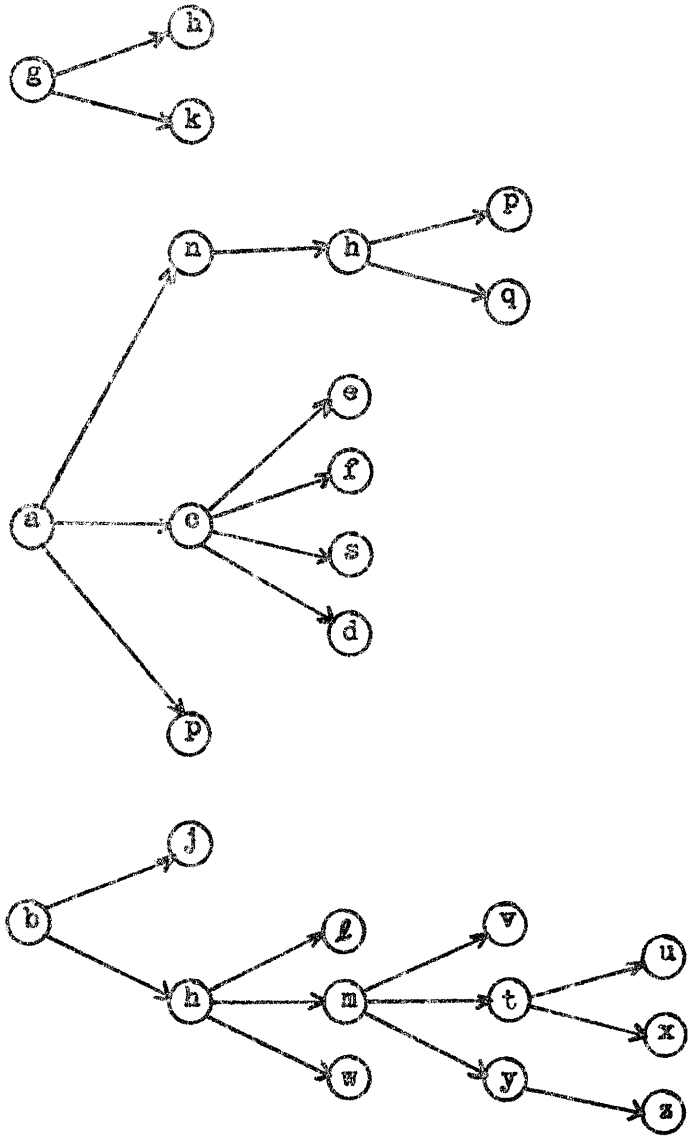
U
(a)



U/T
(b)

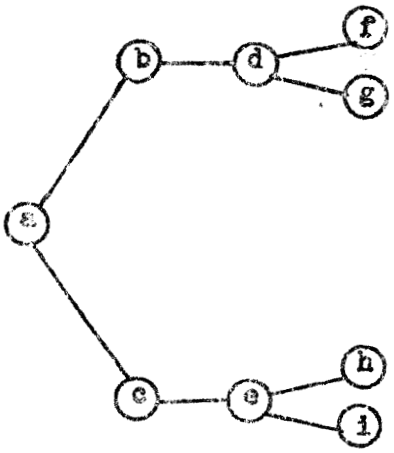
Compression of tree T of Fig. 6-3

Fig. 6-5



The tree of Fig. 6-3 permuted by Program 6-23

Fig. 6-6

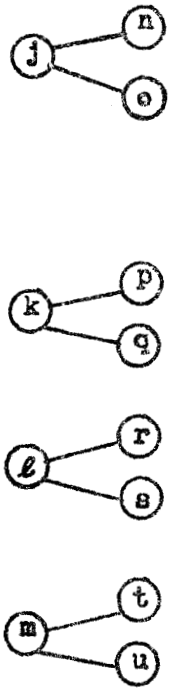


$\underset{\sim}{T}$

$v(\underset{\sim}{T}) = (1,2,1,2)$

$\mathcal{M}(\underset{\sim}{T}) = (1,2,2,4)$

(a)

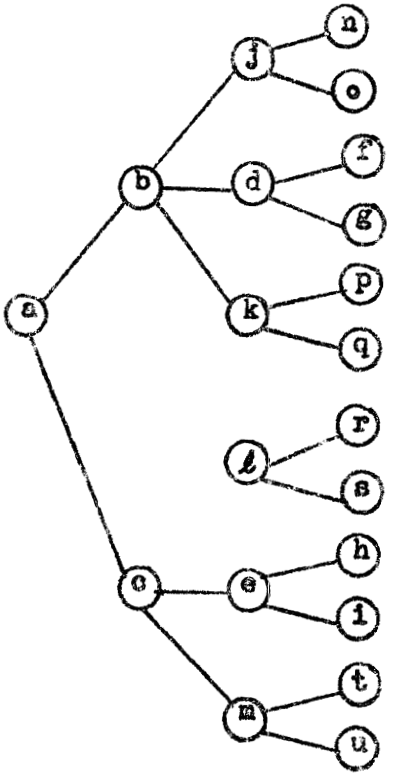


$\underset{\sim}{Q}$

$v(\underset{\sim}{Q}) = (4,2)$

$\mathcal{M}(\underset{\sim}{Q}) = (4,8)$

(b)



$\underset{\sim}{P} = \{\underset{\sim}{T}, u, \underset{\sim}{Q}\}$, for $u = (1,0,1)$

$v(\underset{\sim}{P}) = (1,2,3,2)$

$\mathcal{M}(\underset{\sim}{P}) = (1,2,6,12)$

(c)

Grafting at level three

Fig. 6-7

Meaning	Representation			
	Printed	Typed	Dittoed	Handwritten
Literal Alphabetic	Roman, u.c. and l.c.	Circled l.c. Roman (a), (abc)	As Typed	Printed: a,b,c.
Numeric	Standard numeral	Standard Numerals	As Typed	Standard
Variable Alphabetic	l.c. italic	Unmarked	As Typed	Script, a,b,c.
Numeric	Italic numeral	Underscore	As Typed	Underscore
Set	u.c. italic	Unmarked	As Typed	Unmarked
Vector	l.c. boldface italic	Red	Red Underscore	Underscore
Matrix	u.c. boldface italic	Red	Red Underscore	Underscore
Tree	u.c. boldface Roman	Wavy underscore		Wavy underscore

Typographic conventions for classes of operands

Table 6-1

Variable	Range and dimension	Meaning
H	$\mu(H) = 2,\ \nu(H) = 13$	
H_i^1	S	Suit of card i
H_i^2	D	Denomination of card i
S		Card suits
	(c)	Clubs
	(d)	Diamonds
	(h)	Hearts
	(s)	Spades
D		Card denominations
	2	Deuce
	3	Trey
	.	.
	.	.
	.	.
	(k)	King
	(a)	Ace

The definition of symbols

Table 6-2

Appendix A

Class	Scalar	Vector	Matrix	Tree	Set
arbitrary	a,b,c	a,b,c	A,B,C	$\underset{\sim}{A},\underset{\sim}{B},\underset{\sim}{C}$	A,B,C
numerical	x,y,z	x,y,z	X,Y,Z	$\underset{\sim}{X},\underset{\sim}{Y},\underset{\sim}{Z}$	
integral	i,j,k	i,j,k	I,J,K	$\underset{\sim}{I},\underset{\sim}{J},\underset{\sim}{K}$	
logical	u,v,w	u,v,w	U,V,W	$\underset{\sim}{U},\underset{\sim}{V},\underset{\sim}{W}$	
mapping (integral values and null)	m	m	M	$\underset{\sim}{M}$	

Operand conventions

Section I

	Operation	Notation	Definition	Compatibility
1	Dimension	$j \leftarrow \nu(A)$	Number of elements in set A	
2		$j \leftarrow \nu(a)$	Number of components in vector a	
3		$j \leftarrow \nu(a)$	$j_i = \nu(a_i)$	a is a vector of sets, i.e., each component is a set
4	Row dimension	$j \leftarrow \nu(A)$	Number of components in each row vector of A	
5	Column dimension	$j \leftarrow \mu(A)$	Number of components in each column vector of A	
6	Height	$j \leftarrow \nu(\underset{\sim}{T})$	Length of longest maximal path in $\underset{\sim}{T}$	
7	Moment	$j \leftarrow \mu(\underset{\sim}{T})$	Number of nodes of $\underset{\sim}{T}$	
8	Null element	o		
9	Null element of set A	o(A)		
10	Index origin	ϕ	$\phi = 0$ for 0-origin indexing $= 1$ for 1-origin indexing	
11		$\phi(a)$	Index origin of a, where a is a set, vector, matrix, or tree	

Dimensions, index origins, and null elements

Section II

	Operation	Notation	Definition	Compatibility
12	Magnitude	$z \leftarrow \lvert x \rvert$	$z = (1-2(x<0)) \times x$	
13		$z \leftarrow \lvert x \rvert$	$z_i = \lvert x_i \rvert$	$\nu(z) = \nu(x)$
14		$Z \leftarrow \lvert X \rvert$	$Z_j^i = \lvert X_j^i \rvert$	$\nu(Z) = \nu(X)$; $\mu(Z) = \mu(X)$
15		$\underset{\sim}{Z} \leftarrow \lvert \underset{\sim}{X} \rvert$	$(\underset{\sim}{Z}_i)^j = \lvert (\underset{\sim}{X}_i)^j \rvert$	$(\lfloor \underset{\sim}{Z} \rfloor)_\phi = (\lfloor \underset{\sim}{X} \rfloor)_\phi$
16	Floor	$k \leftarrow \lfloor x \rfloor$	$k \leq x < k + 1$	
17		$k \leftarrow \lfloor x \rfloor$	$k_i = \lfloor x_i \rfloor$	$\nu(k) = \nu(x)$
18		$K \leftarrow \lfloor X \rfloor$	$K_j^i = \lfloor X_j^i \rfloor$	$\nu(K) = \nu(X)$; $\mu(K) = \mu(X)$
19		$\underset{\sim}{K} \leftarrow \lfloor \underset{\sim}{X} \rfloor$	$(\underset{\sim}{K}_i)^j = \lfloor (\underset{\sim}{X}_i)^j \rfloor$	$(\lfloor \underset{\sim}{K} \rfloor)_\phi = (\lfloor \underset{\sim}{X} \rfloor)_\phi$
20	Ceiling	$k \leftarrow \lceil x \rceil$	$k \geq x > k - 1$	
21		$k \leftarrow \lceil x \rceil$	$k_i = \lceil x_i \rceil$	$\nu(k) = \nu(x)$
22		$K \leftarrow \lceil X \rceil$	$K_j^i = \lceil X_j^i \rceil$	$\nu(K) = \nu(X)$; $\mu(K) = \mu(X)$
23		$\underset{\sim}{K} \leftarrow \lceil \underset{\sim}{X} \rceil$	$(\underset{\sim}{K}_i)^j = \lceil (\underset{\sim}{X}_i)^j \rceil$	$(\lfloor \underset{\sim}{K} \rfloor)_\phi = (\lfloor \underset{\sim}{X} \rfloor)_\phi$
24	Residue	$k \leftarrow \lvert i,j \rvert$	$k \equiv i \pmod{j}$; $0 \leq k < j$	
25		$k \leftarrow \lvert i,j \rvert$	$k_i = \lvert i_i, j_i \rvert$	$\nu(k) = \nu(i) = \nu(j)$
26		$K \leftarrow \lvert I,J \rvert$	$K_j^i = \lvert I_j^i, J_j^i \rvert$	$\nu(K) = \nu(I) = \nu(J)$; $\mu(K) = \mu(I) = \mu(J)$
27		$\underset{\sim}{K} \leftarrow \lvert \underset{\sim}{I}, \underset{\sim}{J} \rvert$	$(\underset{\sim}{K}_i)^j = \lvert (\underset{\sim}{I}_i)^j, (\underset{\sim}{J}_i)^j \rvert$	$(\lfloor \underset{\sim}{K} \rfloor)_\phi = (\lfloor \underset{\sim}{I} \rfloor)_\phi = (\lfloor \underset{\sim}{J} \rfloor)_\phi$

	Operation	Notation	Definition	Compatibility
28		$z \leftarrow x + y$	$z_i = x_i + y_i$	$\nu(z) = \nu(x) = \nu(y)$
29	Sum	$Z \leftarrow X + Y$	$Z_j^i = X_j^i + Y_j^i$	$\nu(Z) = \nu(X) = \nu(Y)$; $\mu(Z) = \mu(X) = \mu(Y)$
30		$\underset{\sim}{Z} \leftarrow \underset{\sim}{X} + \underset{\sim}{Y}$	$(\underset{\sim}{Z}_i)^j = (\underset{\sim}{X}_i)^j + (\underset{\sim}{Y}_i)^j$	$(\lfloor\underset{\sim}{Z})_\phi = (\lfloor\underset{\sim}{X})_\phi = (\lfloor\underset{\sim}{Y})_\phi$
31		$z \leftarrow x - y$	$z_i = x_i - y_i$	$\nu(z) = \nu(x) = \nu(y)$
32	Difference	$Z \leftarrow X - Y$	$Z_j^i = X_j^i - Y_j^i$	$\nu(Z) = \nu(X) = \nu(Y)$; $\mu(Z) = \mu(X) = \mu(Y)$
33		$\underset{\sim}{Z} \leftarrow \underset{\sim}{X} - \underset{\sim}{Y}$	$(\underset{\sim}{Z}_i)^j = (\underset{\sim}{X}_i)^j - (\underset{\sim}{Y}_i)^j$	$(\lfloor\underset{\sim}{Z})_\phi = (\lfloor\underset{\sim}{X})_\phi = (\lfloor\underset{\sim}{Y})_\phi$
34		$z \leftarrow xy$	$z_i = x \times y_i$	$\nu(z) = \nu(y)$
35	Scalar multiple	$Z \leftarrow xY$	$Z_j^i = x \times Y_j^i$	$\nu(Z) = \nu(Y)$; $\mu(Z) = \mu(Y)$
36		$\underset{\sim}{Z} = x\underset{\sim}{Y}$	$(\underset{\sim}{Z}_i)^j = x \times (\underset{\sim}{Y}_i)^j$	$(\lfloor\underset{\sim}{Z})_\phi = (\lfloor\underset{\sim}{Y})_\phi$
37		$z \leftarrow y \div x$	$z_i = y_i \div x$	$\nu(z) = \nu(y)$
38	Scalar quotient	$Z \leftarrow Y \div x$	$Z_j^i = Y_j^i \div x$	$\nu(Z) = \nu(Y)$; $\mu(Z) = \mu(Y)$
39		$\underset{\sim}{Z} \leftarrow \underset{\sim}{Y} \div x$	$(\underset{\sim}{Z}_i)^j = (\underset{\sim}{Y}_i)^j \div x$	$(\lfloor\underset{\sim}{Z})_\phi = (\lfloor\underset{\sim}{Y})_\phi$

	Operation	Notation	Definition	Compatibility
40		$z \leftarrow x \times y$	$z_i = x_i \times y_i$	$\nu(z) = \nu(x) = \nu(y)$
41	Direct product	$Z \leftarrow X \times Y$	$Z^i_j = X^i_j \times Y^i_j$	$\nu(Z) = \nu(X) = \nu(Y)$; $\mu(Z) = \mu(X) = \mu(Y)$
42		$\underset{\sim}{Z} \leftarrow \underset{\sim}{X} \times \underset{\sim}{Y}$	$(\underset{\sim}{Z}_i)^j = (\underset{\sim}{X}_i)^j \times (\underset{\sim}{Y}_i)^j$	$(\lfloor \underset{\sim}{Z})_\phi = (\lfloor \underset{\sim}{X})_\phi = (\lfloor \underset{\sim}{Y})_\phi$
43		$z \leftarrow x \div y$	$z_i = x_i \div y_i$	$\nu(z) = \nu(x) = \nu(y)$
44	Direct quotient	$Z \leftarrow X \div Y$	$Z^i_j = X^i_j \div Y^i_j$	$\nu(Z) = \nu(X) = \nu(Y)$; $\mu(Z) = \mu(X) = \mu(Y)$
45		$\underset{\sim}{Z} \leftarrow \underset{\sim}{X} \div \underset{\sim}{Y}$	$(\underset{\sim}{Z}_i)^j = (\underset{\sim}{X}_i)^j \div (\underset{\sim}{Y}_i)^j$	$(\lfloor \underset{\sim}{Z})_\phi = (\lfloor \underset{\sim}{X})_\phi = (\lfloor \underset{\sim}{Y})_\phi$
46		$z \leftarrow xy$	$z = \sigma(x \times y)$	$\nu(x) = \nu(y)$
47	Inner product	$z \leftarrow Xy$	$z_i = X^i y$	$\nu(z) = \mu(X)$; $\nu(X) = \nu(y)$
48		$z \leftarrow xY$	$z_i = xY_i$	$\nu(z) = \nu(Y)$; $\nu(x) = \mu(Y)$
49		$Z \leftarrow XY$	$Z^i_j = X^i Y_j$	$\nu(Z) = \nu(Y)$; $\mu(Z) = \mu(X)$; $\nu(X) = \mu(Y)$
50	Outer product	$Z \leftarrow x \mid y$	$Z^i_j = x_i \times y_j$	$\nu(Z) = \nu(y)$; $\mu(Z) = \nu(x)$
51	Vector cross product	$z \leftarrow x \cdot y$	$z = \uparrow x \times \downarrow y - \downarrow x \times \uparrow y$	$\nu(z) = \nu(x) = \nu(y) = 3$

	Operation	Notation	Definition	Compatibility
52	Base y value — Vector	$z \longleftarrow y \perp x$	$z = sx;$ $z_{\phi+i} = (x/(\bar{a}^{i+1}/y)) + (i = \nu(y) - \bar{\phi})$	$\nu(y) = \nu(x)$
53	Base y value — Row	$z \longleftarrow y \perp X$	$z_i = y \perp X^i$	$\nu(Y) = \nu(X);$ $\nu(z) = \mu(Y) = \mu(X)$
54	Base y value — Column	$z \longleftarrow y \perp\!\!\!\perp X$	$z_i = y \perp X_i$	$\mu(Y) = \mu(X);$ $\nu(z) = \nu(Y) = \nu(X)$
55	Base y value (Polynomial in y)	$z \longleftarrow (y\epsilon) \perp x$	$z = (y\epsilon) \perp x$	
56	Base y value — Vector	$z \longleftarrow y \underset{a}{\perp} b$	$z = y \perp (\iota(b \text{ wo } a) - \phi\epsilon)$	$\nu(y) = \nu(b) = \nu(a);$ $b \in {}^{\otimes}/_{a}$
57	Base y value — Row	$z \longleftarrow y \underset{a}{\perp} B$	$z_i = y \underset{a}{\perp} B^i$	$\nu(z) = \mu(B);$ $\nu(y) = \nu(B) = \nu(a);$ $B^i \in {}^{\otimes}/_{a}$
58	Base y value — Column	$z \longleftarrow y \underset{a}{\perp\!\!\!\perp} B$	$z_i = y \underset{a}{\perp} B_i$	$\nu(z) = \nu(B);$ $\nu(y) = \mu(B) = \nu(a);$ $B_i \in {}^{\otimes}/_{a}$

	Operation	Notation	Definition	Compatibility
59		$b \leftarrow \odot/a$	$b = \begin{cases} a_\phi \odot a_{\phi+1} \odot \cdots \odot a_{-\phi} & \text{if } \nu(a) \neq 0 \\ 0 & \text{if } \nu(a) = 0 \end{cases}$	$\odot$ is any binary associative operator (such as $\times, \wedge, \vee, \circledast$) defined upon the components of a.
60		$b \leftarrow \odot/A$	$b_i = \odot/A^i$	$\nu(b) = \mu(A)$
61	$\odot$-weight	$b \leftarrow \odot//A$	$b_j = \odot/A_j$	$\nu(b) = \nu(A)$
62		$b \leftarrow \odot/\underset{\sim}{A}$	$b = \odot/(\lfloor A)_{\phi+1}$	
63		$b \leftarrow \odot//\underset{\sim}{A}$	$b = \odot/(\lfloor \underset{\sim}{A})_{\phi+1}$	
64		$x \leftarrow \sigma(y)$	$x = {}^+/y = \epsilon y$	
65	Weight	$x \leftarrow \sigma(Y)$	$x = {}^+/({}^+//Y) = \epsilon Y \epsilon$	
66		$x \leftarrow \sigma(\underset{\sim}{Y})$	$x = {}^+/\underset{\sim}{Y} = {}^+// \underset{\sim}{Y}$	
67		$v \leftarrow u \underset{A}{\lceil} x$	$v/x = m\epsilon$, and $((\bar{v} \wedge u)/x \underset{A}{<} m\epsilon) = \epsilon$	$\nu(v) = \nu(u) = \nu(x)$
68	Maximum	$V \leftarrow U \underset{A}{\lceil} X$	$V/X = m\epsilon$, and $((\bar{V} \wedge U)/X \underset{A}{<} m\epsilon) = \epsilon$	$\nu(V) = \nu(U) = \nu(X)$; $\mu(V) = \mu(U) = \mu(X)$
69		$\underset{\sim}{V} \leftarrow \underset{\sim}{U} \underset{A}{\lceil} \underset{\sim}{X}$	$\underset{\sim}{V}/\underset{\sim}{X} = m^\epsilon \underset{\sim}{E}$, and $((\bar{\underset{\sim}{V}} \wedge \underset{\sim}{U})/\underset{\sim}{X} \underset{A}{<} m^\epsilon \underset{\sim}{E}) = {}^\epsilon \underset{\sim}{E}$	$(\lfloor \underset{\sim}{V})_\phi = (\lfloor \underset{\sim}{U})_\phi = (\lfloor \underset{\sim}{X})_\phi$
70		$v \leftarrow u \underset{A}{\rfloor} x$	$v/x = m\epsilon$, and $((\bar{v} \wedge u)/x \underset{A}{>} m\epsilon) = \epsilon$	$\nu(v) = \nu(u) = \nu(x)$
71	Minimum	$V \leftarrow U \underset{A}{\rfloor} X$	$V/X = m\epsilon$, and $((\bar{V} \wedge U)/X \underset{A}{} m\epsilon) = \epsilon$	$\nu(V) = \nu(U) = \nu(X)$; $\mu(V) = \mu(U) = \mu(X)$
72		$\underset{\sim}{V} \leftarrow \underset{\sim}{U} \underset{A}{\rfloor} \underset{\sim}{X}$	$\underset{\sim}{V}/\underset{\sim}{X} = m^\epsilon \underset{\sim}{E}$, and $((\bar{\underset{\sim}{V}} \wedge \underset{\sim}{U})/\underset{\sim}{X} \underset{A}{>} m^\epsilon \underset{\sim}{E}) = {}^\epsilon \underset{\sim}{E}$	$(\lfloor \underset{\sim}{V})_\phi = (\lfloor \underset{\sim}{U})_\phi = (\lfloor \underset{\sim}{X})_\phi$

	Operation	Notation	Definition	Compatibility
73	Member-ship	$x \in A$	x is an element of the set A	
74	Precedence	$x \underset{A}{\prec} y$	element x precedes element y in A	$x \in A \quad y \in A$
75	Inclusion	$B \subseteq A$	$x \in B$ implies that $x \in A$	
76	Inter-section	$C \leftarrow A \wedge B$	C is formed from A by suppressing each A_i for which $A_i \notin B$.	
77	Comple-mentation	$C \leftarrow \overline{A}$	C is formed from the universe of discourse U by suppressing each U_i for which $U_i \in A$.	
78	Concate-nation	$C \leftarrow (A,B)$	$C \equiv \{A_\phi, A_{\phi+1}, \cdots, A_{-\phi}, B_\phi, B_{\phi+1}, \cdots, B_{-\phi}\}$	$A \wedge B = 0$
79	Union	$C \leftarrow A \vee B$	$C \equiv (A, B \wedge \overline{A})$	
80	Infix or interval of A	$C \underset{A}{\leftarrow} \{(a,b)\}$	$C \equiv \{A_i, A_{i+1}, \cdots, A_j\}$, where $a = A_i$, $b = A_j$	$a,b \in A$; $i \leq j$
81	kth successor in A	$c \leftarrow k \underset{A}{\uparrow} b$	$b = A_i \Longrightarrow c = A_j$; $j - \phi = \lvert i + k - \phi, \nu(A) \rvert$	$b \in A$
82		$C \leftarrow k \underset{A}{\uparrow} B$	$C_i = k \underset{A}{\uparrow} B_i$	$B \subseteq A$
83	kth predecessor in A	$c \leftarrow k \underset{A}{\downarrow} b$	$b = A_i \Longrightarrow c = A_j$; $j - \phi = \lvert i - k - \phi, \nu(A) \rvert$	$b \in A$
84		$C \leftarrow k \underset{A}{\downarrow} B$	$C_i = k \underset{A}{\downarrow} B_i$	$B \subseteq A$

A may be elided if it is an infix of the set of integers

k may be elided if k = 1

	Operation	Notation	Definition	Compatibility
85	Left rotation	$C \leftarrow k \uparrow A$	$C \equiv k \underset{A}{\uparrow} A$	k may be elided if k = 1
86	Right rotation	$C \leftarrow k \downarrow A$	$C \equiv k \underset{A}{\downarrow} A$	
87	Cartesian product	$C \leftarrow A^1 \otimes A^2 \otimes \cdots \otimes A^n$	The elements of C are vectors c with $\nu(c) = n$, and $c_i \in A^i$, so ordered that if $a \neq b$, then $a \underset{C}{<} b \Longleftrightarrow a_k \underset{A^k}{<} b_k$, for $k = \phi + \alpha(a=b)$	$\nu(C) = \nu(A^1) \times \nu(A^2) \times \ldots \times \nu(A^n)$
88	Homogeneous product set	$C \leftarrow [A]^K$	$C \equiv \otimes/(\iota^K / A\epsilon)$	$\nu(C) = (\nu(A))^K$
89	Ranking: element	$m \leftarrow \iota(b \text{ wo } B)$	$m = i$ if $b = B_i$; $m = o$ if $b \notin B$	
90	Ranking: vector	$m \leftarrow \iota(b \text{ wo } a)$	$m_k = i$ if $b_k = (a_k)_i$; $m_k = o$ if $b_k \notin a_k$	a is a vector of sets $\nu(m) = \nu(b) = \nu(a)$
91	Ranking: matrix	$M \leftarrow \iota(B \text{ wo } A)$	$M^k_j = i$ if $B^k_j = (A^k_j)_i$; $M^k_j = o$ if $B^k_j \notin A^k_j$	A is a matrix of sets $\nu(M) = \nu(B) = \nu(A)$ $\mu(M) = \mu(B) = \mu(A)$
92	Ranking: tree	$\underset{\sim}{M} \leftarrow \iota(\underset{\sim}{B} \text{ wo } \underset{\sim}{A})$	$(\underset{\sim}{M}_k)^j = i$ if $(\underset{\sim}{B}_k)^j = (\underset{\sim}{A}_k)^j$; $(\underset{\sim}{M}_k)^j = o$ if $(\underset{\sim}{B}_k)^j \notin (\underset{\sim}{A}_k)^j$	$\underset{\sim}{A}$ is a tree of sets $(\lfloor \underset{\sim}{M})_\phi = (\lfloor \underset{\sim}{B})_\phi = (\lfloor \underset{\sim}{A})_\phi$
93	Set to vector	$a \leftarrow A$	$a_i = A_i$	$\nu(a) = \nu(A)$
94	Vector to set	$A \leftarrow a$	$A_i = b_i$, where $b = u/a$, and u is the logical vector of greatest base 2 value for which all components of b are distinct	$\nu(A)$ = number of distinct components in a

	Operation	Notation	Definition	Compatibility
95		$w \leftarrow \bar{u}$	$w = 1 \Leftrightarrow u = 0$	
96	Negation (not)	$w \leftarrow \bar{u}$	$w_i = \bar{u}_i$	$\nu(w) = \nu(u)$
97		$W \leftarrow \bar{U}$	$W^i_j = \bar{U}^i_j$	$\nu(W) = \nu(U)$; $\mu(W) = \mu(U)$
98		$\underset{\sim}{W} \leftarrow \bar{\underset{\sim}{U}}$	$(\underset{\sim}{W}_i)^j = (\overline{\underset{\sim}{U}_i})^j$	$(\lfloor W \rfloor)_\phi = (\lfloor U \rfloor)_\phi$
99		$w \leftarrow u \wedge v$	$w = 1 \Leftrightarrow u = 1$ and $v = 1$	
100	Logical product (and)	$w \leftarrow u \wedge v$	$w_i = u_i \wedge v_i$	$\nu(w) = \nu(u) = \nu(v)$
101		$W \leftarrow U \wedge V$	$W^i_j = U^i_j \wedge V^i_j$	$\nu(W) = \nu(U) = \nu(V)$; $\mu(W) = \mu(U) = \mu(V)$
102		$\underset{\sim}{W} \leftarrow \underset{\sim}{U} \wedge \underset{\sim}{V}$	$(\underset{\sim}{W}_i)^j = (\underset{\sim}{U}_i)^j \wedge (\underset{\sim}{V}_i)^j$	$(\lfloor W \rfloor)_\phi = (\lfloor U \rfloor)_\phi = (\lfloor V \rfloor)_\phi$
103		$w \leftarrow u \vee v$	$w = 1 \Leftrightarrow u = 1$ or $v = 1$	
104	Logical sum (or)	$w \leftarrow u \vee v$	$w_i = u_i \vee v_i$	$\nu(w) = \nu(u) = \nu(v)$
105		$W \leftarrow U \vee V$	$W^i_j = U^i_j \vee V^i_j$	$\nu(W) = \nu(U) = \nu(V)$; $\mu(W) = \mu(U) = \mu(V)$
106		$\underset{\sim}{W} \leftarrow \underset{\sim}{U} \vee \underset{\sim}{V}$	$(\underset{\sim}{W}_i)^j = (\underset{\sim}{U}_i)^j \vee (\underset{\sim}{V}_i)^j$	$(\lfloor W \rfloor)_\phi = (\lfloor U \rfloor)_\phi = (\lfloor V \rfloor)_\phi$

	Operation	Notation	Definition	Compatibility
107		$w \leftarrow (a \mathcal{R} b)$	$w = 1 \Leftrightarrow a \mathcal{R} b$	$\mathcal{R}$ is a binary relation on a and b
108		$w \leftarrow (a \mathcal{R} b)$	$w_i = (a_i \mathcal{R} b_i)$	$\nu(w) = \nu(a) = \nu(b)$
109	Logical reduction	$W \leftarrow (A \mathcal{R} B)$	$W^i_j = (A^i_j \mathcal{R} B^i_j)$	$\nu(W) = \nu(A) = \nu(B)$; $\mu(W) = \mu(A) = \mu(B)$
110		$\underset{\sim}{W} \leftarrow (\underset{\sim}{A} \mathcal{R} \underset{\sim}{B})$	$(\underset{\sim}{W}_i)^j = ((\underset{\sim}{A}_i)^j \mathcal{R} (\underset{\sim}{B}_i)^j)$	$(\lfloor \underset{\sim}{W})_\phi = (\lfloor \underset{\sim}{A})_\phi = (\lfloor \underset{\sim}{B})_\phi$
111		$c \leftarrow au$	$c = a$ if $u = 1$; $c = 0$ if $u = 0$	
112	Scalar replacement	$c \leftarrow au$	$c_i = au_i$	$\nu(u) = \nu(c)$
113		$C \leftarrow aU$	$C^i_j = aU^i_j$	$\nu(U) = \nu(C)$; $\mu(U) = \mu(C)$
114		$\underset{\sim}{C} \leftarrow a\underset{\sim}{U}$	$(\underset{\sim}{C}_i)^j = a(\underset{\sim}{U}_i)^j$	$(\lfloor \underset{\sim}{C})_\phi = (\lfloor \underset{\sim}{U})_\phi$
115	Vector	$k \leftarrow a(u)$	$k = \max_j((\times/a^j/u) \times (^+/a^j/u))$	
116	Row (Prefix weight)	$k \leftarrow a(U)$	$k_i = a(U^i)$	$\nu(k) = \mu(U)$
117	Column (Prefix weight)	$k \leftarrow a((U))$	$k_i = a(U_i)$	$\nu(k) = \nu(U)$

	Operation	Notation	Definition	Compatibility
118	Vector (Suffix weight)	$k \leftarrow \omega(u)$	$k = \max_j ((^{\times}/\omega^j/u) \times (^{+}/\omega^j/u))$	
119	Row (Suffix weight)	$k \leftarrow \omega(U)$	$k_i = \omega(U^i)$	$\nu(k) = \mu(U)$
120	Column (Suffix weight)	$k \leftarrow \omega((U))$	$k_i = \omega(U_j)$	$\nu(k) = \nu(U)$
121	Subset vector	$w \leftarrow \epsilon_A^C$	$w_i = 1 \Leftrightarrow A_i \in C$	$C \subseteq A$, $\nu(w) = \nu(A)$, $\sigma(w) = \nu(C)$
122	i^{th} Boolean function	$w \leftarrow \beta^i(u)$	$w = \varepsilon_j$: where $j = ((2\epsilon) \perp u) + \phi$, $\nu(\varepsilon) = 2^{\nu(u)}$, and $(2\epsilon) \perp i = i$	$0 \leq i < 2^{\nu(u)}$
123		$w \leftarrow \beta^i(U)$	$w_j = \beta^i(U^j)$	$0 \leq i < 2^{\nu(U)}$
124		$w \leftarrow \beta^i((U))$	$w_j = \beta^i(U_j)$	$0 \leq i < 2^{\nu(U)}$

	Operation	Notation	Definition	Compatibility	
125	Unit vector	ϵ^j	$(\epsilon^j)_k = (j = k)$		Dimension or structure defined by context
126	Full vector	ϵ	$\epsilon_k = 1$		
127	Prefix vector of weight j	α^j	First j components of α^j are each 1; rest are 0		
128	Suffix vector of weight j	ω^j	Last j components of ω^j are each 1; rest are 0		
129	Identity matrix	I	$I^j_k = (j = k)$		
130	Full matrix	E	$E^j_k = 1$		
131	Level tree	${}^uE$	$({}^uE^\lambda)_{-1} = \nu(u)$	$\nu({}^uE) = \nu(u)$	
132	Full tree	${}^\epsilon E$			
133	Prefix tree of weight j	${}^{\alpha^j}E$			
134	Suffix tree of weight j	${}^{\omega^j}E$			

Special logical vectors, matrices, and trees

Section VII

	Operation	Notation	Definition	Compatibility
135	Set compression	$C \leftarrow u/A$	$A_i \in C \Leftrightarrow u_i = 1$; $C \equiv A \wedge C$	$\nu(A) = \nu(u)$; $\nu(C) = \sigma(u)$
136	Vector compression	$c \leftarrow u/a$ (or $c \leftarrow a//u$)	Components selected like elements in set compression	$\nu(c) = \nu(u)$; $\nu(c) = \sigma(u)$
137	Row compression	$C \leftarrow v/A$	$C^i = v/A^i$	$\nu(C) = \nu(u)$; $\nu(C) = \sigma(v)$
138		$c \leftarrow U/A$	$c = (U^\phi/A^\phi\ U^{\phi+1}/A^{\phi+1}, \cdots, U^{-\phi}/A^{-\phi})$	$\nu(A) = \nu(U)$; $\mu(A) = \mu(U)$; $\nu(c) = \sigma(U)$
139	Column compression	$C \leftarrow u//A$	$C_i = u//A_i$	$\mu(A) = \nu(u)$; $\mu(C) = \sigma(u)$
140		$c \leftarrow U//A$	$c = (U_\phi//A_\phi, U_{\phi+1}//A_{\phi+1}, \cdots, U_{-\phi}//A_{-\phi})$	$\nu(c) = \nu(U)$; $\mu(A) = \mu(U)$; $\nu(c) = \sigma(U)$
141	Tree compression	$\underset{\sim}{C} \leftarrow \underset{\sim}{U}/\underset{\sim}{A}$	$\underset{\sim}{C}$ is obtained by suppressing from $\underset{\sim}{A}$ those nodes $(A_{ij})^j$ for which $(U_{ij})^j = 0$ and reconnecting so that node a lies in the subtree of $\underset{\sim}{C}$ rooted at b $\Leftrightarrow$ a lies in the subtree of $\underset{\sim}{A}$ rooted at b.	$(\lfloor C \rfloor)_\phi = (\lfloor U \rfloor)_\phi = (\lfloor A \rfloor)_\phi$
142	Truncation	$\underset{\sim}{C} \leftarrow v/\underset{\sim}{A}$	$\underset{\sim}{C} = \ulcorner v \urcorner/\underset{\sim}{A}$	$\nu(C) = \sigma(v)$; $\nu(\underset{\sim}{A}) = \nu(v)$
143	Row list	$c \leftarrow \lceil A$		
144	Column list	$c \leftarrow \lceil\!\lceil A$		

	Operation	Notation	Definition	Compatibility
145	Vector mesh	$c \leftarrow \backslash a,u,b \backslash$	$\overline{u}/a = a$; $a/u = b$	$\nu(a) = \sigma(\overline{u})$; $\nu(c) = \sigma(u)$; $\nu(c) = \nu(u)$
146	Row mesh	$C \leftarrow \backslash A,\dot{u},B \backslash$	$\overline{u}/C = A$; $u/C = B$	$\nu(A) = \sigma(\overline{u})$; $\nu(B) = \sigma(u)$; $\nu(C) = \nu(u)$; $\mu(A) = \mu(C) = \mu(B)$
147		$C \leftarrow \backslash a,U,b \backslash$	$\overline{U}/C = a$; $U/C = b$	$\nu(C) = \nu(U)$; $\mu(c) = \mu(U)$; $\nu(a) = \sigma(\overline{U})$; $\nu(b) = \sigma(U)$
148	Column mesh	$C \leftarrow \Vert A,u,B \Vert$	$\overline{u}/C = A$; $u/C = B$	$\mu(A) = \sigma(\overline{u})$; $\mu(B) = \sigma(u)$; $\mu(C) = \nu(u)$; $\nu(A) = \nu(B) = \nu(C)$
149		$C \leftarrow \Vert a,U,b \Vert$	$\overline{U}/C = a$; $U/C = b$	$\nu(C) = \nu(U)$; $\mu(C) = \mu(U)$; $\nu(a) = \sigma(\overline{U})$; $\nu(b) = \sigma(U)$
150	Tree mesh	$\underset{\sim}{C} \leftarrow \backslash \underset{\sim}{A},\underset{\sim}{U},\underset{\sim}{B} \backslash$	$\overline{\underset{\sim}{U}}/\underset{\sim}{C} = \underset{\sim}{A}$; $\underset{\sim}{U}/\underset{\sim}{C} = \underset{\sim}{B}$	$(\lfloor \underset{\sim}{C} \rfloor)_\phi = (\lfloor \underset{\sim}{U} \rfloor)_\phi$; $(\lfloor \underset{\sim}{A} \rfloor)_\phi = (\lfloor (\overline{\underset{\sim}{U}}/\underset{\sim}{U}) \rfloor)_\phi$; $(\lfloor \underset{\sim}{B} \rfloor)_\phi = (\lfloor (\underset{\sim}{U}/\underset{\sim}{U}) \rfloor)_\phi$
151	Concatenation	$c \leftarrow (a,b)$	$c = \backslash a, a^{\nu(b)}, b \backslash$	$\nu(c) = \nu(a) + \nu(b)$
152		$C \leftarrow (A,B)$	$C = \backslash A, a^{\nu(B)}, B \backslash$	$\nu(C) = \nu(A) + \nu(B)$; $\mu(C) = \mu(A) = \mu(B)$
153		$C \leftarrow \binom{A}{B}$	$C = \Vert A, a^{\mu(B)}, B \Vert$	$\mu(C) = \mu(A) + \mu(B)$; $\nu(C) = \nu(A) = \nu(B)$

	Operation	Notation	Definition	Compatibility
155	Vector expansion	$c \leftarrow\!\!- a\backslash b$	$c = \backslash 0, a, b \backslash$	$\nu(a) = \nu(c);\ \nu(b) = \sigma(c)$
156	Row expansion	$C \leftarrow\!\!- A\backslash B$	$C = \backslash 0, A, B \backslash$	$\nu(A) = \nu(a);\ \nu(B) = \sigma(a);$ $\mu(A) = \mu(B)$
157		$C \leftarrow\!\!- U\backslash b$	$C = \backslash 0, U, b \backslash$	$\nu(A) = \nu(B);\ \nu(b) = \sigma(B);$ $\mu(A) = \mu(U)$
158	Column expansion	$C \leftarrow\!\!- a \backslash\backslash B$	$C = \backslash\backslash 0, a, B \backslash\backslash$	$\mu(A) = \nu(a);\ \mu(B) = \sigma(B);$ $\nu(A) = \nu(B)$
		$C \leftarrow\!\!- U \backslash\backslash b$	$C = \backslash\backslash 0, U, b \backslash\backslash$	$\nu(A) = \nu(B);\ \nu(b) = \sigma(B);$ $\mu(A) = \mu(U)$
159	Tree expansion	$\underset{\sim}{C} \leftarrow\!\!- \underset{\sim}{U}\backslash\underset{\sim}{A}$	$\underset{\sim}{C} = \backslash 0, \underset{\sim}{J}, \underset{\sim}{A} \backslash$	$(\lfloor \underset{\sim}{C})_\phi = (\lfloor \underset{\sim}{U})_\phi;$ $(\lfloor \underset{\sim}{A})_\phi = (\lfloor (\underset{\sim}{U}/\underset{\sim}{U}))_\phi$
160	Vector mask	$c \leftarrow\!\!- /a, c, b/$	$\bar{c}/c = \bar{a}/c;\ a/c = c/b$	$\nu(c) = \nu(a) = \nu(c) = \nu(b)$
161	Matrix mask	$C \leftarrow\!\!- /A, C, B/$	$\bar{C}/C = \bar{A}/A;\ A/C = C/B$	$\nu(A) = \nu(A) = \nu(C) = \nu(B);$ $\mu(A) = \mu(A) = \mu(C) = \mu(B)$
162	Row mask	$C \leftarrow\!\!- /A, c, B/$	$\bar{A}/C = \bar{C}/A;\ a/C = c/B$	$\nu(C) = \nu(A) = \nu(c) = \nu(C);$ $\mu(C) = \mu(B) = \mu(C)$
163	Column mask	$C \leftarrow\!\!- //A, a, B//$	$\bar{A}//C = \bar{a}//b;\ a//C = c//B$	$\mu(C) = \mu(A) = \nu(c) = \mu(C);$ $\nu(C) = \nu(B) = \nu(C)$
164	Tree mask	$\underset{\sim}{C} \leftarrow\!\!- /\underset{\sim}{A}, \underset{\sim}{U}, \underset{\sim}{B}/$	$\bar{\underset{\sim}{U}}/\underset{\sim}{C} = \bar{\underset{\sim}{U}}/\underset{\sim}{A};\ \underset{\sim}{U}/\underset{\sim}{C} = \underset{\sim}{U}/\underset{\sim}{B}$	$(\lfloor \underset{\sim}{C})_\phi = (\lfloor \underset{\sim}{A})_\phi = (\lfloor \underset{\sim}{U})_\phi = (\lfloor \underset{\sim}{B})_\phi$

	Operation		Notation	Definition		Compatibility
165	Permutation vector		j	Each j_i is a distinct element of $\{(\phi(j), \nu(\ell) = \ell(j))\}$		Dimension defined by context
166	Identity permutation vector		ℓ	$(\ell)_k = k$		
167	Mapping vector		a	Each a_i is either a null element or an integer		
168	Set	Mapping or Permutation	$a \leftarrow\!\!- A_m$	$a_i = A_{k_i}; k = \phi i = \lvert u \mid u = \phi \varepsilon, \nu(\ell) \varepsilon \rvert$	$\nu(c) = \sigma(u)$	a is a mapping vector. If it is also a permutation vector and $\nu(a)$ equals the dimension of the permuted operand then this and the operation is called a permutation.
169	Vector		$a \leftarrow\!\!- a_m$	$a_i = a_{k_i}; k = \phi i = \lvert u \mid u = \phi \varepsilon, \nu(a) \varepsilon \rvert$	$\nu(c) = \sigma(u)$	
170	Row		$a \leftarrow\!\!- A_m$	$a_i = A_{k_i}; k = \phi i = \lvert u \mid u = \phi \varepsilon, \nu(a) \varepsilon \rvert$	$\nu(C) = \sigma(u)$ $\mu(C) = \mu(A)$	
171	Column		$a \leftarrow\!\!- A^m$	$a^i = A^{k_i}; k = \phi i = \lvert u \mid u = \phi \varepsilon, \mu(\ell) \varepsilon \rvert$	$\mu(C) = \sigma(u)$ $\nu(C) = \nu(A)$	
172	Mapping vector		$a \leftarrow\!\!- \beta(a \leftarrow\!\!- \beta)$	If $\beta_i \notin C_\beta$, $a_i = o$ If $\beta_i \varepsilon C_\beta$, a_i is the smallest integer such that $\beta_i = \alpha_{a_i}$, where C is the set of distinct components of α.		α is any set or vector β is any set or vector
173	Ordering vector		$j \leftarrow\!\!-^{\alpha}_{B} a$	j is a permutation vector such that for $\alpha = \alpha_j$, either $j_i = a_{i+1}$ and $j_i < j_{i+1}$, or $j_i \underset{B}{<} j_{i+1}$		$a \varepsilon \left[F \right]^{\nu(\alpha)}$

	Operation		Notation	Definition	Compatibility
174	Set rotation	Left	$C \leftarrow k\uparrow A$	$C_i = A_j$; $j - \phi = \lvert i + k - \phi, \nu(A) \rvert$ See also Defs. 56-57	$\nu(C) = \nu(A)$
175		Right	$C \leftarrow k\downarrow A$	$C_i = A_j$; $j - \phi = \lvert i - k - \phi, \nu(A) \rvert$	
176	Vector rotation	Left	$c \leftarrow k\uparrow a$	$c_i = a_j$; $j - \phi = \lvert i + k - \phi, \nu(a) \rvert$	$\nu(c) = \nu(a)$
177		Right	$c \leftarrow k\downarrow a$	$c_i = a_j$; $j - \phi = \lvert i - k - \phi, \nu(a) \rvert$	
178	Row rotation	Left	$C \leftarrow k\uparrow A$	$C^i = k_j \uparrow A^i$	$\nu(k) = \mu(A)$ $\nu(C) = \nu(A)$ $\mu(C) = \mu(A)$
179		Right	$C \leftarrow k\downarrow A$	$C^i = k_j \downarrow A^i$	
180	Column rotation	Up	$C \leftarrow k\uparrow\uparrow A$	$C_j = k_j \uparrow A_j$	$\nu(k) = \nu(A)$
181		Down	$C \leftarrow k\downarrow\downarrow A$	$C_j = k_j \downarrow A_j$	
182	Transpose		$C \leftarrow \tilde{A}$	$C_j = A^j$	$\nu(C) = \mu(A)$, $\mu(C) = \nu(A)$

	Operation		Notation	Definition	Compatibility
183	File array	Full	$\underline{\underline{\Phi}}$	Set of files $\underline{\Phi}^i_j$: $i \in I$, $j \in J$; $I \equiv \{(\phi, \lambda(\underline{\Phi})+\bar{\phi})\}$; $J \equiv \{(\phi, \nu(\underline{\Phi})+\bar{\phi})\}$	
184		Row	$\underline{\Phi}^i$	$j \in J$	
185		Column	$\underline{\Phi}_j$	$i \in I$	
186	File		$\underline{\Phi}^i_j$	A partitioned representation of a vector x of the form $\lambda(\phi), x_\phi, \lambda(\phi+1), x_{\phi+1}, \cdots, x_{\nu(x)+\bar{\phi}}, \lambda(\nu(x)+\bar{\phi}), \cdots, \lambda(-\phi)$, where $\lambda(r)$ is the partition at position r; $\lambda(\phi) = \lambda(-\phi) = \lambda$; All other $\lambda(r) \in \{\lambda_0, \lambda_1, \lambda_2, \cdots\}$	
187	Position file		$\pi(\underline{\Phi}^i_j) \leftarrow r$	Set file $\underline{\Phi}^i_j$ to position r	
188	(called rewind if $r=\phi$)		$\pi(\underline{\Phi}^i) \leftarrow r\epsilon$	Set each file of row $\underline{\Phi}^i$ to position r	
189	(called wind if $r=-\phi$)		$\pi(\underline{\Phi}_j) \leftarrow r\epsilon$	Set each file of column $\underline{\Phi}_j$ to position r	
190			$\pi(\underline{\Phi}) \leftarrow r\underline{\underline{\epsilon}}$	Set each file of $\underline{\Phi}$ to position r	

	Operation		Notation	Definition		Compatibility	
191	Record	Forward	$\underset{\sigma}{\Phi}{}^{i}_{j} \leftarrow y, \lambda_k$	Specify x_r by y, partition $\lambda(r+1)$ by λ_k, and stop at position $(r+1)$	r is initial position	$r \neq -\phi$. Zero prescript may be elided	λ_k may be elided if k=0
192		Backward	$\underset{1}{\Phi}{}^{i}_{j} \leftarrow y, \lambda_k$	Specify x_{r-1} by y, partition $\lambda(r-1)$ by λ_k, and stop at position $(r-1)$		$r \neq \phi$	
193	Read	Forward	$y \leftarrow \underset{o}{\Phi}{}^{i}_{j}$	Specify y by x_r, stop at position $(r+1)$, branch according to $\lambda(r+1)$	r is initial position	$r \neq -\phi$. Zero prescript may be elided	
194		Backward	$y \leftarrow \underset{1}{\Phi}{}^{i}_{j}$	Specify y by x_{r-1}, stop at position $(r-1)$, branch according to $\lambda(r-1)$		$r = \phi$	
195	Compress	Row	w/Φ	The compress operations defined upon matrices and vectors may be extended directly to file arrays, e.g., $P \leftarrow w/\Phi$, or $\pi(w/\Phi) \leftarrow \phi \in$		$\nu(u) = \nu(\Phi)$	
196		Column	u/Φ			$\nu(u) = \mu(\Phi)$ $\sigma(u) = \nu(P)$	

	Operation	Notation	Definition	Compatibility
197	Height	$\nu(\underset{\sim}{A})$	Dimension of longest path in $\underset{\sim}{A}$	
198	Moment vector	$\mathcal{M}(\underset{\sim}{A})$	$\mathcal{M}_j(\underset{\sim}{A})$ is the number of nodes on level j.	$\nu(\mathcal{M}(\underset{\sim}{A})) = \nu(\underset{\sim}{A})$
199	Number of roots	$\mathcal{M}_1(\underset{\sim}{A})$		
200	Number of nodes	$\mu(\underset{\sim}{A})$	$\mu(\underset{\sim}{A}) = \sigma(\mathcal{M}(\underset{\sim}{A}))$	
201	Number of leaves	$\lambda(\underset{\sim}{A})$		
202	Degree of node $(A^i)_{-\phi}$	$\delta((\underset{\sim}{A}^i)_{-\phi})$	Number of nodes on level $\nu(i)+1$ reachable from node $(\underset{\sim}{A}^i)_{-\phi}$	
203	Degree of node x	$\delta(x)$	$\delta(x) = \delta(A^i)_{-\phi})$ for $i = \ell(x \text{ in } \underset{\sim}{A})$	
204	Maximum degree	$\delta(\underset{\sim}{A})$	$\delta(\underset{\sim}{A}) = \max_i \delta((A^i)_{-\phi})$	
205	Path i	$c \leftarrow \underset{\sim}{A}^i$	c_ϕ = root i_ϕ of $\underset{\sim}{A}$; c_j = node i_j of the group on level j reachable from node c_{j-1}, for $j \in \{(\phi+1, \nu(i)-\bar{\phi})\}$	i is an index vector of $\underset{\sim}{A}$, that is, $i_\phi \in \{(\phi, \mathcal{M}_1(\underset{\sim}{A})-\bar{\phi})\}$, and $i_j \in \{(\phi, \delta(c_{j-1})-\bar{\phi})\}$.
206	Node i	$c \leftarrow (\underset{\sim}{A}^i)_{-\phi}$	c is the terminal node of path $\underset{\sim}{A}^i$	i is an index vector of $\underset{\sim}{A}$

	Operation	Notation	Definition	Compatibility
207	Subtree i	$\underset{\sim}{B} \leftarrow \underset{\sim}{A}_i$	$\underset{\sim}{B}$ comprises all subtrees of $\underset{\sim}{A}$ whose roots are directly reachable from $(\underset{\sim}{A}^i)_{-\phi}$	i is an index vector of $\underset{\sim}{A}$
208	Index of b in $\underset{\sim}{A}$	$\ell(b \text{ wo } \underset{\sim}{A})$	If $u = ((\jmath\underset{\sim}{A})_{\phi+1} = b\epsilon) = 0$, then $\nu(i) = 0$; if $u \neq 0$, then $i = (m \neq \circ \epsilon)/m$, where $m = (u/(\overline{\alpha}^2/\jmath\underset{\sim}{A}))^{\phi}$	
209	Tree to vector	$a \leftarrow \underset{\sim}{B}$	a_j is the subtree rooted at the jth root of $\underset{\sim}{B}$	$\nu(a) = \mu_1(B)$
210	Vector to tree	$\underset{\sim}{B} \leftarrow a$		
211	Right (justified) index matrix	$I \leftarrow \overline{\alpha}^2/\jmath A$	I^k is of the form $\lvert \circ \epsilon, \omega^{\nu(i)}, i \rvert$	i is an index vector of $\underset{\sim}{A}$, and $j < k \iff (\nu(B)\epsilon)\underset{B\epsilon}{\perp} I^j < (\nu(B)\epsilon)\underset{B\epsilon}{\perp} I^k$, where $B \equiv \{\circ, \phi, \phi+1, \cdots, \delta(A)-\phi\}$	$\nu(I) = \nu(A)$; $\mu(I) = \mu(A)$
212	Left (justified) index matrix	$I \leftarrow \overline{\alpha}^2/\backslash_{\sim}A$	I^k is of the form $\lvert \rho\epsilon, \alpha^{\nu(i)}, i \rvert$	

	Operation	Notation	Definition	Compatibility
213	Right degree vector	$k \leftarrow\!\!- \alpha^1/\rfloor \underset{\sim}{A}$	$k_j=\delta((A^i)_{-\phi})$	$\nu(K)=\mu(\underset{\sim}{A})$
214	Left degree vector	$k \leftarrow\!\!- \alpha^1/\lfloor \underset{\sim}{A}$	$i = (\Gamma^j \neq_\theta \theta)/\Gamma^j$, and Γ is the right (left) index matrix of $\underset{\sim}{A}$	
215	Right node vector	$b \leftarrow\!\!- \varepsilon^{\phi+1}/\rfloor \underset{\sim}{A}$	$b_j=(A^i)_{-\phi}$	$\nu(b)=\mu(\underset{\sim}{A})$
216	Left node vector	$b \leftarrow\!\!- \varepsilon^{\phi+1}/\lfloor \underset{\sim}{A}$		
217	Full right list matrix	$C \leftarrow\!\!- \rfloor \underset{\sim}{A}$	C_ϕ is the right (left) degree vector of $\underset{\sim}{A}$, $C_{\phi+1}$ is the right (left) node vector of $\underset{\sim}{A}$, Γ^2/C is the right (left) index matrix of $\underset{\sim}{A}$	$\nu(C)=\nu(\underset{\sim}{A})+2$, $\mu(C)=\mu(\underset{\sim}{A})$
218	Full left list matrix	$C \leftarrow\!\!- \lfloor A$		
219	Right list matrix	$C \leftarrow\!\!- \alpha^2/\rfloor A$		
220	Left list matrix	$C \leftarrow\!\!- \alpha^2/\lfloor A$		
221	Path list matrix	$C \leftarrow\!\!- \rfloor \underset{\sim}{A}$	$\bar{\alpha}^1/C = \bar{\alpha}^1/\rfloor \underset{\sim}{A}$, and $k = \alpha^1/C$ is the <u>path list vector</u> such that k_j is the row index in C of the first node emanating from node $C^j_{\phi+1}$	$\nu(C)=\nu(\underset{\sim}{A})+2$, $\mu(C)=\mu(\underset{\sim}{A})$

number of roots of $\underset{\sim}{T}$

	Operation	Notation	Definition	Compatibility
222	Dispersion vector	$\nu(\underset{\sim}{T})$	$\nu_\phi(\underset{\sim}{T})=\overset{\wedge}{\ }$; $\nu_j(\underset{\sim}{T})$ = common degree of nodes on level j-1, for $j\in\{(\phi+1,\nu(\underset{\sim}{T})-\bar{\phi})\}$	$\nu(\nu(\underset{\sim}{T}))=\nu(\underset{\sim}{T})$
223	Moment vector (See also Def. 198)	$\mu(\underset{\sim}{T})$	$\mu_j(\underset{\sim}{T})=\gamma(a^j/\nu(\underset{\sim}{T}))$ = number of nodes on level j, for $j\in\{(\phi,\nu(T)-\bar{\phi})\}$	$\nu(\mu(\underset{\sim}{T}))=\nu(\underset{\sim}{T})$
224	Truncation	$\underset{\sim}{P}\leftarrow u/\underset{\sim}{T}$	$\underset{\sim}{P}={}^{u}\underset{\sim}{E}/\underset{\sim}{T}$ (See Defs. 131 and 142)	$\nu(u)=\nu(\underset{\sim}{T})$
225	Insertion	$P\leftarrow\backslash\underset{\sim}{T},u,\underset{\sim}{S}\backslash$	$\bar{u}/\underset{\sim}{P}=\underset{\sim}{T}$; $u/\underset{\sim}{P}=\underset{\sim}{S}$	$\sigma(\bar{u})=\nu(\underset{\sim}{T})$, $\sigma(u)=\nu(\underset{\sim}{S})$, and the moment vector $\mu(\underset{\sim}{P})=\backslash\mu(\underset{\sim}{T}),u,\mu(\underset{\sim}{S})\backslash$ must determine an integral dispersion vector $\nu(\underset{\sim}{P})$
226	Pruning at level j	$\underset{\sim}{P}\leftarrow u/\!\!/_j\underset{\sim}{T}$	$\underset{\sim}{P}$ is obtained by deleting all subtrees $(\underset{\sim}{T}_i)_k$ for which $u_k=0$ and $\nu(i)=j-1$	$\nu(u)=\nu_j(\underset{\sim}{T})$
227	Grafting at level j	$\underset{\sim}{P}\leftarrow\backslash\!\backslash_j\underset{\sim}{T},u,\underset{\sim}{S}\backslash$	$a^{j-1}/\underset{\sim}{P}=a^{j-1}/\underset{\sim}{T}$; $\bar{a}^{j-1}/(\bar{u}/\!\!/_j\underset{\sim}{P})=\bar{a}^{j-1}/\underset{\sim}{T}$; $a^{j-1}/(u/\!\!/_j\underset{\sim}{P})=\underset{\sim}{S}$	$\bar{a}^j/\nu(\underset{\sim}{T})=\bar{a}^{-1}/\nu(\underset{\sim}{S})$ $\nu_j(\underset{\sim}{T})=\sigma(\bar{u})$; $\nu_1(\underset{\sim}{S})=\sigma(u)$; $\nu_1(\underset{\sim}{S})=\sigma(u)\times\mu_{j-1}(\underset{\sim}{T})$ $\nu(\underset{\sim}{P})=/\nu(\underset{\sim}{T}),\epsilon^{j-\bar{\phi}},\nu(u)/$

	Identities	Compatibility Requirements
1	$u \,/\, u \setminus X = X$	$\sigma(u) = \nu(X)$
2	$u /\!\!/ u \setminus\!\!\setminus X = X$	$\sigma(u) = \mu(X)$
3	$\bar{u} \,/\, u \setminus X = 0$	$\sigma(u) = \nu(X)$
4	$\bar{u} /\!\!/ u \setminus\!\!\setminus X = 0$	$\sigma(u) = \mu(X)$
5	$X = u \setminus u \,/\, X + \bar{u} \setminus \bar{u} \,/\, X$	$\nu(u) = \nu(X)$
6	$X = u \setminus\!\!\setminus u /\!\!/ X + \bar{u} \setminus\!\!\setminus \bar{u} /\!\!/ X$	$\nu(u) = \mu(X)$
7	$X = u \setminus v \setminus\!\!\setminus u \,/\, v /\!\!/ X + u \setminus \bar{v} \setminus\!\!\setminus u \,/\, \bar{v} /\!\!/ X$ $+ \bar{u} \setminus v \setminus\!\!\setminus \bar{u} \,/\, v /\!\!/ X + \bar{u} \setminus \bar{v} \setminus\!\!\setminus \bar{u} \,/\, \bar{v} /\!\!/ X$	$\nu(u) = \nu(X)$ $\nu(v) = \mu(X)$
8	$(u \,/\, Z)\, Y = X\, (u \setminus Y)$	$\nu(u) = \nu(Z)$ $\sigma(u) = \mu(Y)$ $\sigma(u) = \nu(X)$ $\nu(u) = \mu(Y)$
9	$(u \setminus Z)\, Y = X\, (u /\!\!/ Y)$	
10	$u \,/\, (XY) = X(u \,/\, Y)$	$\nu(u) = \nu(Y)$
11	$u \setminus (XY) = X(u \setminus Y)$	$\sigma(u) = \nu(Y)$
12	$u /\!\!/ (XY) = (u /\!\!/ X)Y$	$\nu(u) = \mu(Y)$
13	$u \setminus\!\!\setminus (XY) = (u \setminus\!\!\setminus X)Y$	$\sigma(u) = \mu(Y)$
14	$(u \,/\, X)\, (u /\!\!/ Y) + (\bar{u} \,/\, X)\, (\bar{u} /\!\!/ Y) = XY$	$\nu(u) = \nu(X) = \mu(Y)$
15	$(u \,/\, v) /\!\!/ (u /\!\!/ X) = (u \wedge v) /\!\!/ X = (v \,/\, u) /\!\!/ (v /\!\!/ X)$	$\nu(u) = \nu(v) = \mu(X)$
16	$(u \,/\, v) \,/\, (u \,/\, X) = (u \wedge v) \,/\, X = (v \,/\, u) \,/\, (v \,/\, X)$	$\nu(u) = \nu(v) = \nu(X)$

Elementary identities of matrix algebra

Appendix B