TR-78-014
December 1978

A ROUTING NETWORK FOR A MACHINE TO EXECUTE REDUCTION LANGUAGES

by

David R. Kehs

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science

Chapel Hill

1978

Approved by:

Gyula Magó
Adviser

Peter Calingaert
Reader

Reader

DAVID R. KEHS. A Routing Network for a Machine to Execute Reduction Languages. (Under the direction of DR. GYULA A. MAGO.)

ABSTRACT

A network of cells arranged to form a binary tree with connections between horizontally adjacent cells is investigated. Properties of shortest paths between leaf cells in such a network are established.

Algorithms are developed to broadcast copies of a data item from one of the leaf cells to other leaf cells of the network. Using different types of guiding information stored in the cells, these algorithms are modified to avoid broadcasting copies to cells which do not need them. Analysis shows that, at best, these algorithms require $\Omega(n)$ steps to accomplish such patterns as reversal, transposition, or translation of the contents of n leaf cells.

A theoretical bound of $\Theta(n/\log(n))$ steps is established for the problem of reversing the contents of n adjacent leaf cells. This result is generalized to obtain upper bounds for reversal of the contents of non-adjacent leaf cells and for arbitrary movement patterns which do not require multiple copies of data items.

ACKNOWLEDGEMENTS

I would like to thank Dr. Gyula Magó for his invaluable guidance and encouragement throughout the progress of this dissertation and throughout my graduate school career. In addition to suggesting the topic for this research, he read many drafts of this dissertation and made many helpful suggestions to improve its organization and clarity.

The remainder of my committee, Dr. Don Stanat, Dr. Peter Calingaert, Dr. Frederick Brooks, Jr, and Dr. Mehdi Jazayeri, provided many useful comments and criticisms. Don Stanat, in particular, was helpful in pointing me toward clearer and more elegant proofs of many of the mathematical results.

Finally, I would like to thank my family and friends for encouraging and supporting me in this work and especially for distracting me from it.

CONTENTS

1. INTRODUCTION

1.1 The Problem

In a recent paper, Mag6 [1] outlines the organization of a machine which efficiently executes the reduction languages described by Backus [2]. Informal discussions of these languages can be found in [1] as well as the dissertations of Pozefsky [3] and Koster [4]. Figure 1.1 shows a schematic diagram of a portion of the machine as described in [1].

The program text is normally stored in a linear array of cells, L. These cells form the leaves of a binary tree (T) of microprocessors. These processors are provided with enough logic to manipulate the program text according to the rules of the language. Further details can be found in [1].

At certain times during the computation, the machine must move the contents of some of the cells of L to other cells in such a way that the relative positions of some of the symbols must change. For example, the array L may have to be changed from _AB__C_D to B__C_DA_. This process is called data movement. (A similar process in which symbols

move but do not change relative positions, as in a change from _AB__C_D to A__B_CD_, is called storage management. Storage management makes use of horizontal connections between pairs of adjacent L cells. Since these connections are not used for data movement, they are not shown in Figures 1.1-1.3.)

As described in [1], the data items (located in the cells of L) which are to be moved are labeled with integers. (Actually, the labels may be pairs or triples of integers. However, in this paper, we will use only single integers. This restriction does not affect any of the results presented.) The cells to which the data items must move are also assigned integer markers. Then each labeled data item moves through the tree T to the cell which has the same integer marker. It is possible that a particular data item will have to be copied into two or more different locations. To handle this situation, the same integer marker is assigned to each target cell. It is also possible that one data item will have to replace another. In this case, the contents of one cell will be assigned an integer label and the cell will be assigned a different integer target marker. An example of data movement is shown in Figure 1.2. In this example, the data items to be moved are A, B, and C. Notice that B was copied into two different cells, one of which originally contained C.

In the following paragraphs, and, in fact, in the whole dissertation, we assume that the operation of all cells of the machine is synchronized. Though not necessary, this assumption simplifies the descriptions of algorithms. Details about how different data movement patterns could be controlled asynchronously can be found, for example, in [5].

The data movement mechanism described in [1] works as follows. When all of the data items and target cells have been labeled properly, the data items begin to move to the top of the "active area" [1], which is a subtree of T containing the expression being evaluated. (A key feature of this machine is that more than one expression can be evaluated in parallel.) All of the data items move simultaneously unless two items are trying to move up to the same node. In this case, the item with the smaller integer label proceeds first. Each item must move to the top of the active area. At this point, copies of the item are sent downward to each of the T and L cells in the active area. The cells of L which are not looking for a data item with this integer label will simply ignore it. It is possible for a downward-moving data item to "pass" upward-moving items. Thus, at any one time, a tree node may contain two data items, one moving upward (or waiting to do so) and the other moving downward. Figure 1.3 shows how data movement will proceed for the previous example.

The preceding description omits some of the details needed to understand how the algorithm is implemented. Below we will describe how the data items are transferred from cell to cell in the machine of [1]. Figure 1.4 shows a detailed view of three of the cells of T, named A, B, and C. Each cell has six registers which hold data items. These are named INTOP, OUTTOP, INLO, OUTLO, INHI, and OUTHI. In the following description, names of the form A.INLO will be used to refer to the INLO register in cell A, etc. The arrows represent wires along which data can be transferred from register to register. In the algorithm of [1], data will flow only in the direction of the arrows. Thus, each register has only one (or two, in the case of INTOP) other register to which its data item will be copied. The registers can be described as "full" or "empty". When a register holds a data item, it is "full". When the data item is copied into another register, the original register is said to be "empty", i.e. ready to receive another value. Each register contains one bit (or two, in the case of INTOP) which tells if the associated destination register is "full" or "empty". For example, if data are sent from A.OUTLO to B.INTOP, then the "full" bit in A.OUTLO is set. When B.INTOP sends the data (to B.OUTLO and B.OUTHI), it also sends a message to A which causes the "full" bit in A.OUTLO to be turned off.

The complete algorithm takes place in two phases. During the first phase, data items are moved within the cell, according to the arrows of Figure 1.4. Of course, no data can be moved to a "full" register. Instead, the data must wait until the target register is "empty". During the second phase, data items are moved from cell to cell, again according to the arrows shown in Figure 1.4.

The time needed for the entire data movement operation can be expressed as

$$t=2*h + n - 1$$

time units where h is the height of the active area and n is the number of data items. (The first data item reaches the top after h time units, the other n-1 items then pass through the top, and the last item descends to its target in h time units.)

The data movement mechanism just described, to be known henceforth as the VERTICAL algorithm, was chosen in [1] primarily for its simplicity, both of description and of implementation. The purpose of this research is to devise and investigate other data movement mechanisms in an attempt to obtain a significant improvement in efficiency.

We propose to add to the machine horizontal connections between the cells of T, as shown in Figure 1.5. Internally, each cell will be organized as shown in Figure 1.6. This will shorten the paths between many of the pairs of nodes.

For example, the B in Figure 1.7 can move to its destination in four steps, following the path indicated. Going through the top of the tree would require six steps. When more cells are involved, the savings can be even more significant, as shown in Figures 1.8-1.10 which will be discussed below.

In Figure 1.8, five pairs of adjacent symbols are interchanged in only one step. In fact, any number of pairs could be interchanged in just one step. (Recall that in [1], the horizontal connections between pairs of L cells are not used for data movement and all cells rise to the top of the active area. Therefore, all ten items of Figure 1.8 would be sent through the top of the tree.) Figure 1.9 shows the reversal of a ten-element list in seven steps. Since the shortest path between the cell labeled 1 and its target has length seven, we know that at least seven steps are required for this problem. Therefore, the given movement pattern is optimal. In Figure 1.10, two blocks of five symbols are exchanged in six steps. Using the mechanism described in the paper [1], 17 steps would be required for each of these three problems.

When horizontal connections are added, we can expect that the number of steps required for data movement will not necessarily depend directly on n, the number of moving data items. This is because there is no longer a particular node

of T through which all moving data items are required to pass. Also, since data items will no longer be required to rise to the top of the active area, the number of steps will no longer depend on h, the height of the active area. This is particularly fortunate since in a large machine the height of the active area can be rather high, depending on where the expression happens to be in L. Consider, for example, Figure 1.11. Here, the height of the active area containing ABCD (circled in the figure) is either two or five, depending on the location of ABCD. If ABCD had been located elsewhere, the height of the active area could also have been three or four.

In general, then, we can hope that the time required for data movement will depend on the actual distance that the data items must move, rather than the number of items moving and the height of the active area. Of course, as more and more cells become involved in data movement, the distance between a data item's original location and its destination may increase (as in Figure 1.9), and items may interfere with each other (as in Figure 1.10).

The patterns of movement shown in Figures 1.8-1.10 were obtained by inspecting the tree as a whole, including all of the data labels and target markers, to decide which path each data item should follow. The machine itself will not have the advantage of such an overview. When data movement

begins, paths for the data items must be directed by the nodes of T. That is, when a data item arrives at a node of T, there must be enough information in the node to decide where the data item should go next. In the VERTICAL algorithm, there is no information in the cells. The key problem for this research, then, is to decide what information is to be stored in each node of T so that when a data item arrives at a node, it can be routed to its proper destination along a reasonably short path. The information itself must be easily calculated given the integer data labels and target markers, together with their actual positions in L. "Easily" means that the information for all the cells can be computed in O(log(n)) parallel steps. This could be done, for example, by an upward cycle in which information is propagated from son cells to their father and then (perhaps) a downward cycle during which further information is passed from father to sons.

1.2 Related Research

The literature provides little assistance in finding solutions to this problem. A somewhat similar problem has been studied by Sahin [6]. He considers a network of identical cells, one of which wants to communicate with another. However, the cell with the source of the message doesn't know the address of the cell with which it wishes to communicate. Therefore, it broadcasts its message to all of

its neighboring cells. These cells remember the direction of arrival and pass the message on. When the target cell receives the message, it sends back its response. The response is not broadcast throughout the entire network as the original message was. Instead, the response is sent to only one cell, which passes it to another cell, and so on until the response has returned to the source of the original message. Each cell, on receiving the response, must decide which of its neighbors should receive the response next so that it travels to the source on as short a route as possible. If messages can flow in both directions between two cells, as is the case with the reduction language machine, then this decision is trivial. Each cell merely remembers which of its neighbors first passed it the original message. Then, when it receives the response, the cell passes it on to that neighbor. Most of Sahin's work is devoted to the situation where messages can flow in only one direction, so a response could not be sent directly to the neighbor who sent the original message. Therefore, most of Sahin's work is not relevant to the proposed research. Moreover, Sahin considers only one message and response at a time. The reduction language machine must have many data items moving through the network simultaneously. However, it cannot afford to have several copies of one data item moving around unnecessarily since the tree would be quickly saturated.

A number of other researchers have studied problems involving interconnection networks. This work is summarized by Thurber [7] and includes such applications as telephone networks, sorting networks, and permutation networks. The introduction of parallel machines such as ILLIAC IV has generated increased interest in the problem of data routing among the different computing elements of the processor [8,9,10]. This is similar to our problem of routing data among the cells of the array L. However, in all of the schemes reported so far, one assumes that there is some global controller which knows the destination of each data item and which tells each processing element where to send each data item it receives. In contrast, our problem requires each node to decide for itself where to send a data item using only the item's marker and some information stored in the node.

The X-tree of Seguin [11] involves a similar network (a binary tree with horizontal cross connections). However, each leaf cell of the X-tree is assigned a permanent address, and when a message (data item) is routed, it is given the address of its destination. In our problem, the "addresses" (target labels) change with each movement pattern and may appear in any order.

1.3 Organization of the Dissertation

Chapter 1 provided a description of the problem to be studied in this dissertation. In Chapter 2, certain properties concerning shortest paths in the network are established. These properties will be used in later chapters to analyze some of the algorithms for data movement.

In Chapter 3, two groups of algorithms are discussed. Algorithms SS1-SS4 (Single Source data movement) show how to broadcast copies of a single data item throughout the tree. Algorithms MS1-MS4 (Multiple Source data movement) extend SS1-SS4 to the situation in which a number of different data items are broadcast simultaneously.

With Algorithms MS1-MS4, a copy of each data item is sent to each cell of T and L. This involves the creation of many useless copies of data items. The presence of these copies slows the entire movement process. Therefore, in Chapter 4, Algorithms MS1-MS4 are modified to include different types of guiding information which can be stored in the cells of T and used to reduce the production of useless copies of data items. The algorithms presented are MSR1-MSR4 (Multiple Source with Range information), MSER1-MSER4 (Multiple Source with Extended Range Information), and MSH1-MSH4 (Multiple Source with Hashing Information).

In some cases, the guiding information is perfect in the sense that no useless copies of data items are created. Such guiding information is called Complete Path Information and any algorithm which has access to Complete Path Information is called a CP algorithm. In Chapter 5, CP algorithms are analyzed for several data movement patterns such as reversal, translation, and transposition of the contents of L cells. All of these patterns are found to require time which is linear in the number of items moving. This represents no improvement over the order-of-magnitude behavior of the VERTICAL algorithm.

In Chapter 6, some theoretical bounds on the performance of data movement algorithms are established. It is shown that any movement pattern which does not require multiple copies of data items can be accomplished in $\Theta(n*\log\log(n)/\log(n))$ steps where n is the number of L cells in the active area containing the expression in question. (In presenting order-of-magnitude results, we will use the notation of Knuth [12] in which $O(f(n))$ stands for any function whose magnitude is upper-bounded by a constant times $f(n)$ for large n, $\Omega(f(n))$ stands for any function whose magnitude is lower-bounded by a constant times $f(n)$ for large n, and $\Theta(f(n))$ stands for any function lower-bounded by $cf(n)$ and upper-bounded by $df(n)$ for two constants c and d and for large n.)

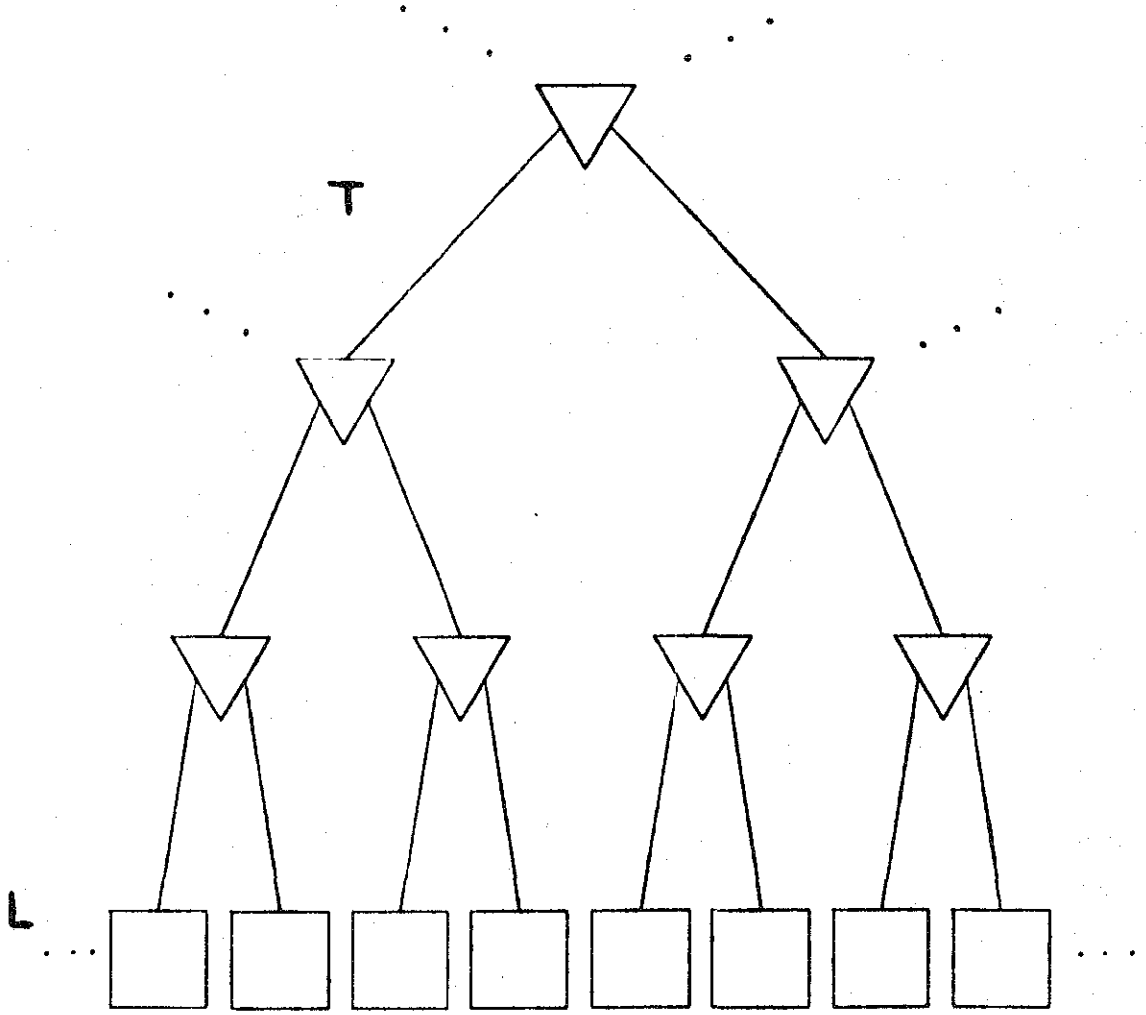
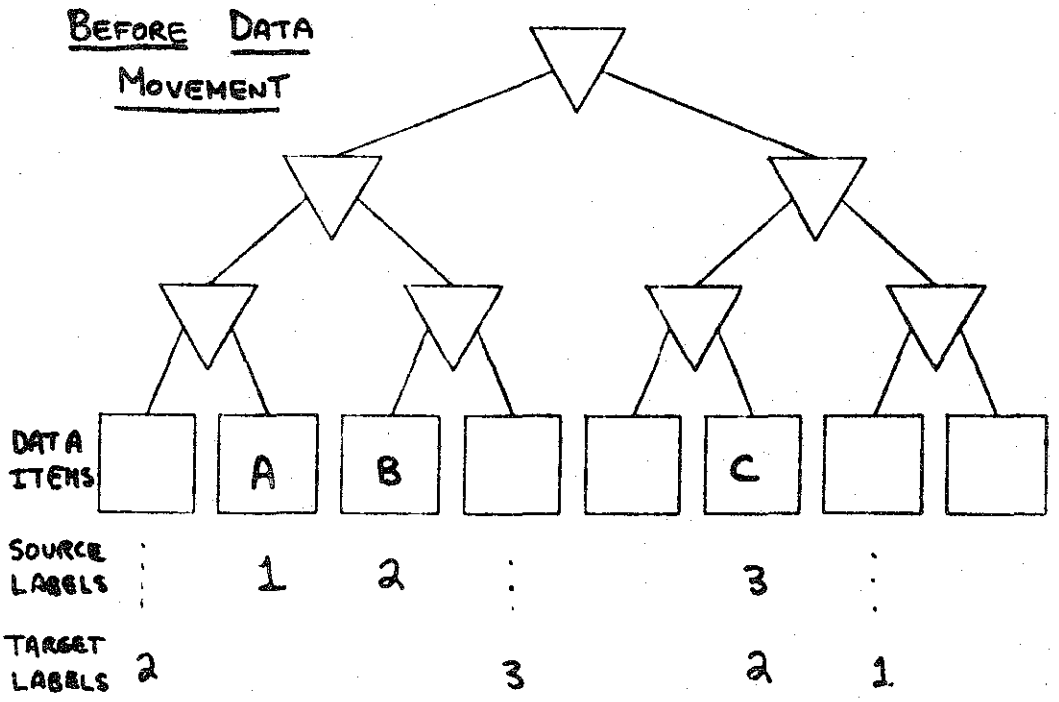


Figure 1.1


BEFORE DATA MOVEMENT
DATA ITEMS
A
B
C
SOURCE LABELS
1
2
3
TARGET LABELS
2
3
2
1

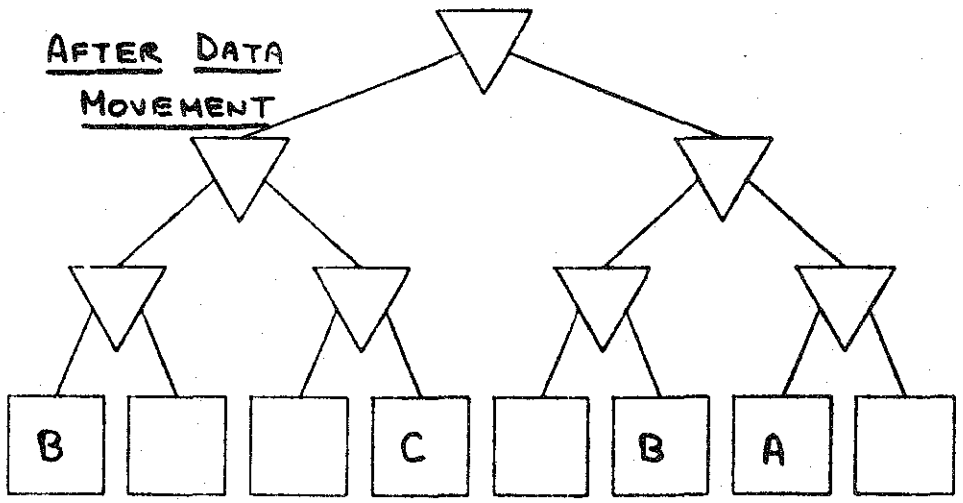

AFTER DATA MOVEMENT
B
C
B
A

Figure 1.2

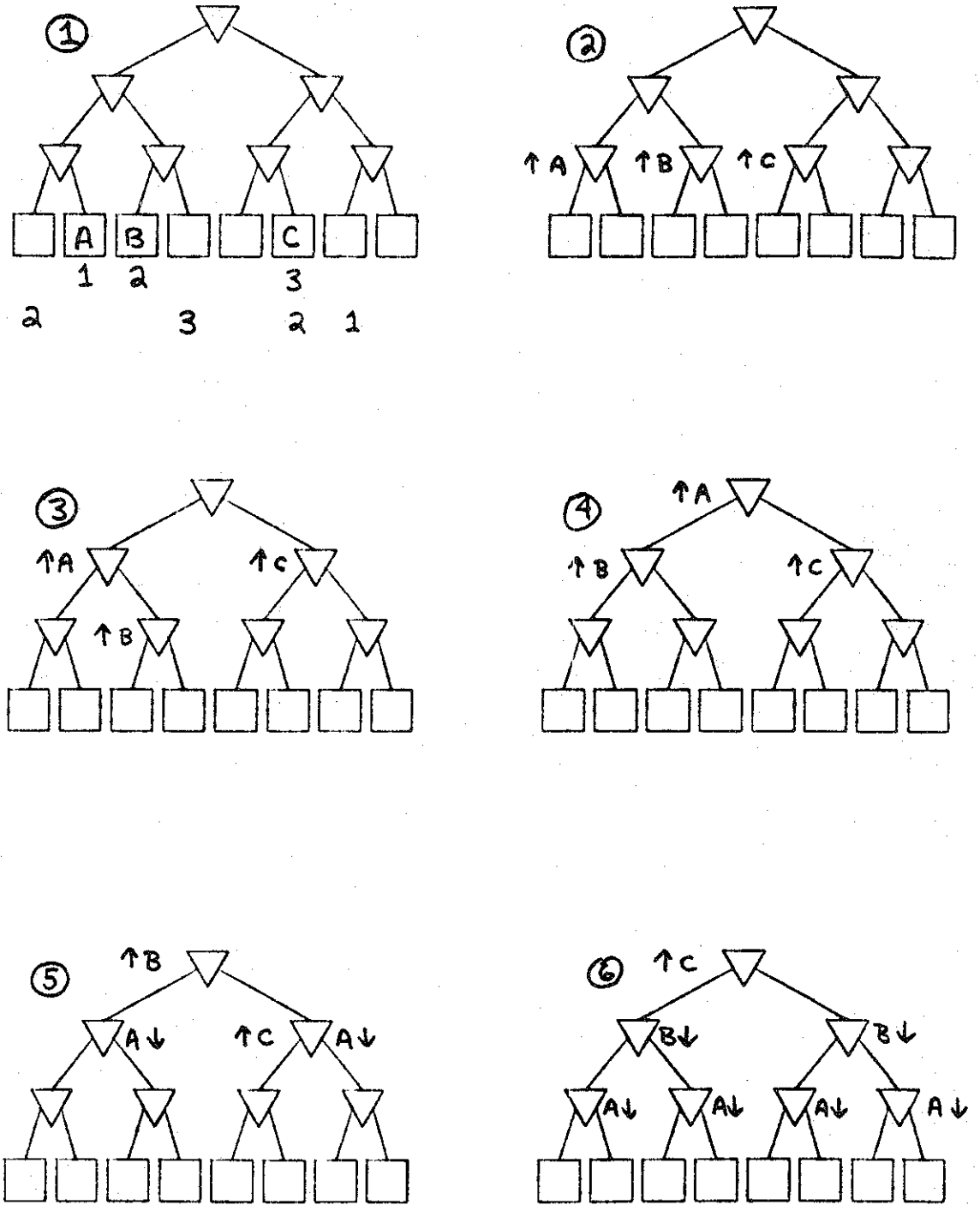
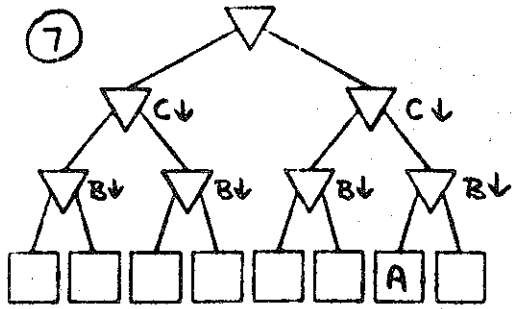
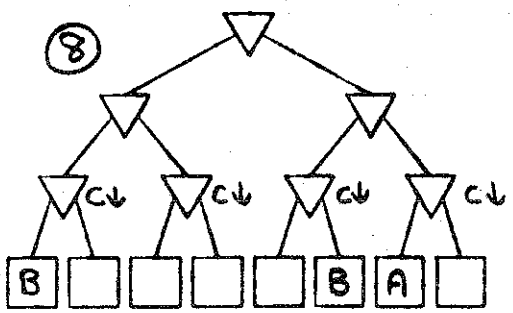


Figure 1.3 (Part 1)





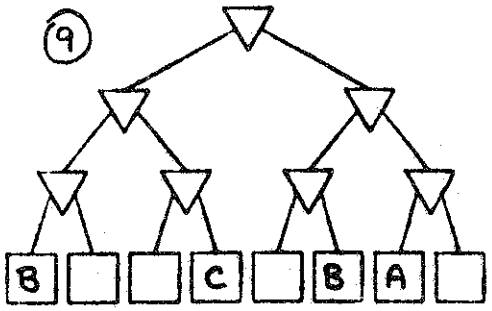


Figure 1.3 (Part 2)

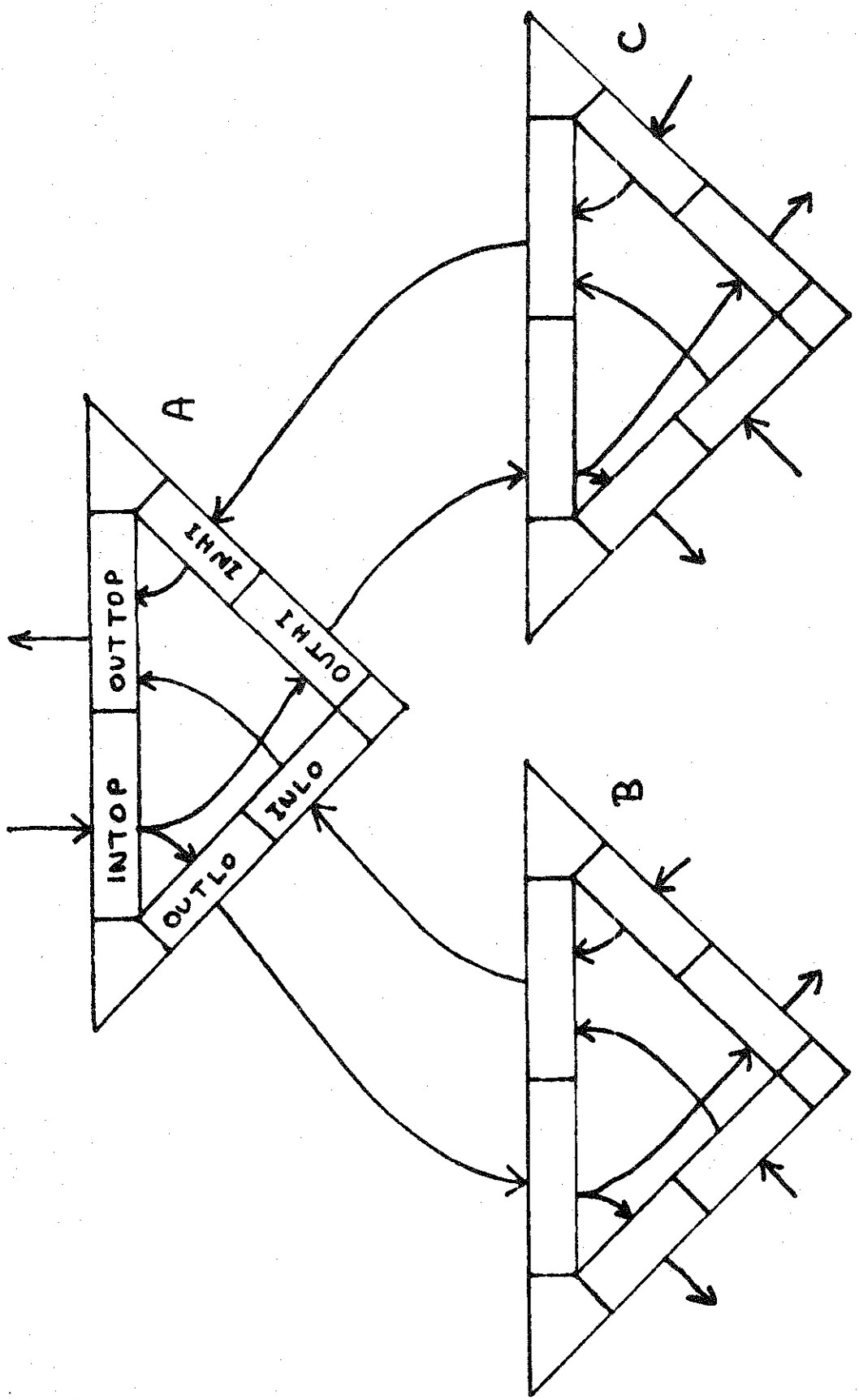


Figure 1.4

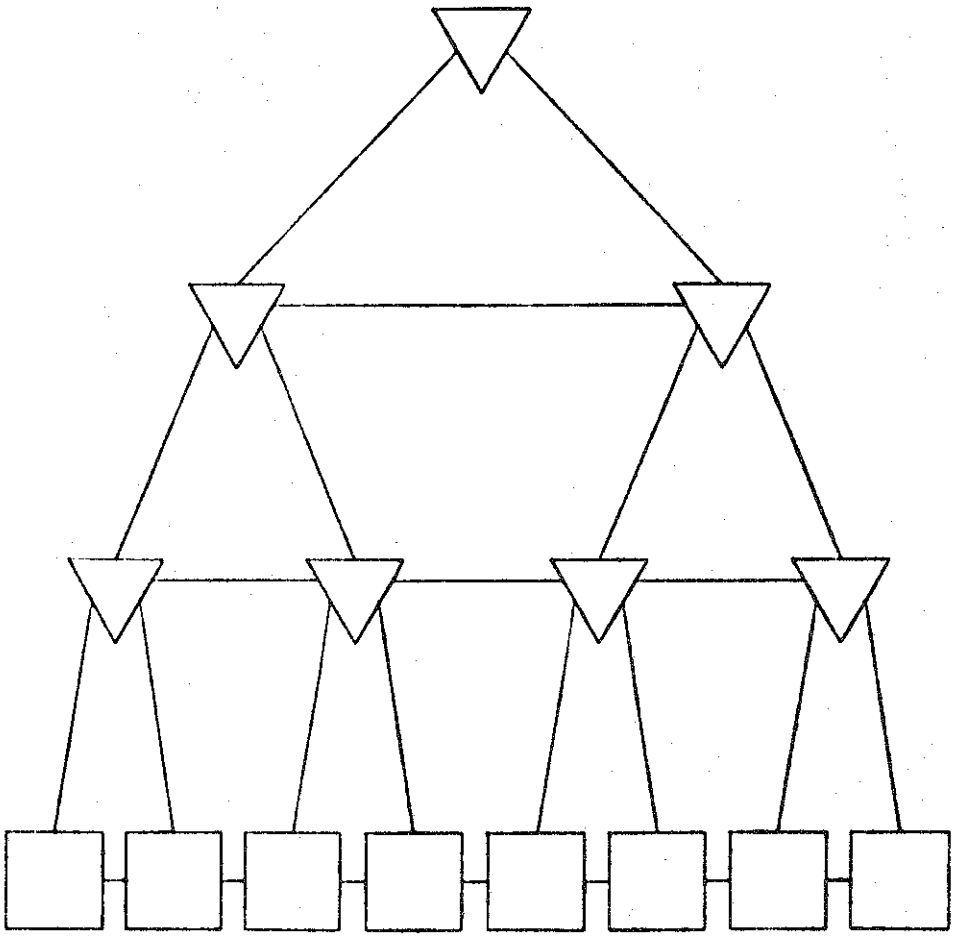


Figure 1.5

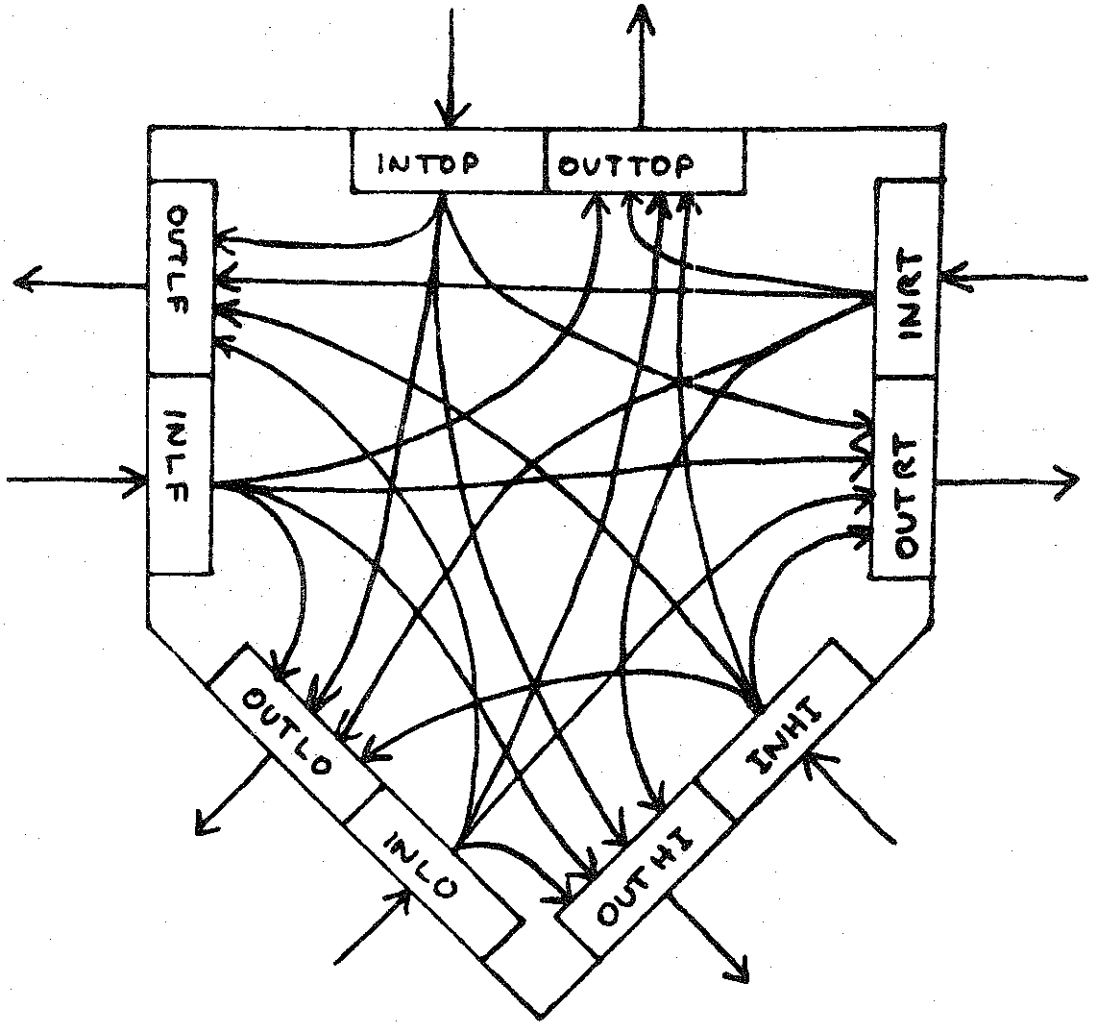


Figure 1.6

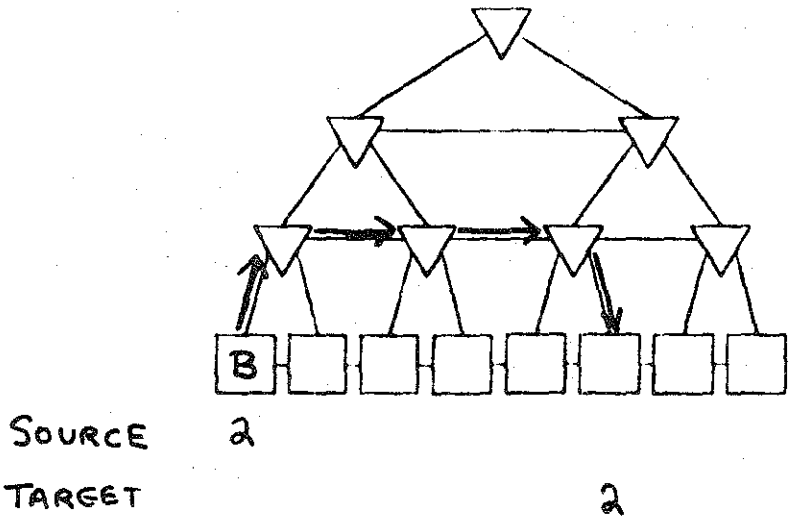
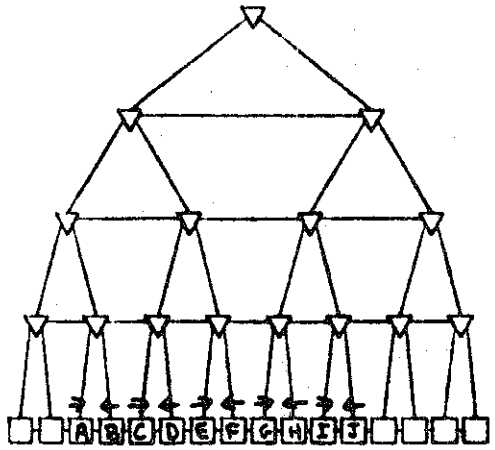


Figure 1.7



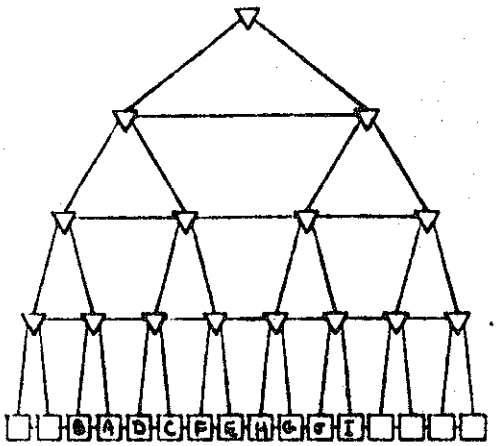


Figure 1.8

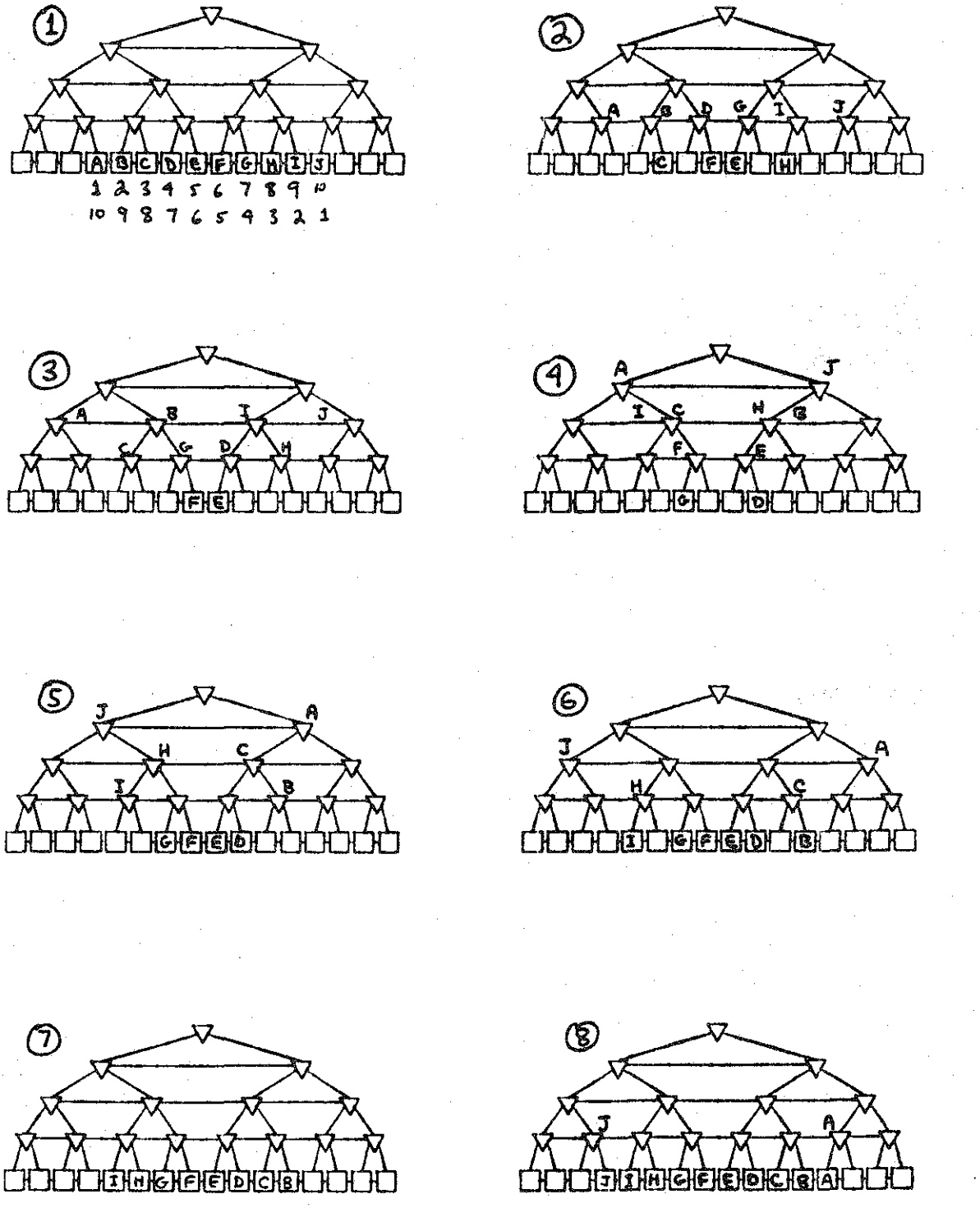

1
A B C D E F G H I J
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
2
A B D G I J
C F E H
3
A B I J
C G D H
F E
4
A J
I C H B
F E
G D
5
J A
H C
I B
G F E D
6
J A
H C
I G F E D B
7
I H G F E D C B
8
J A
J I H G F E D C B A

Figure 1.9

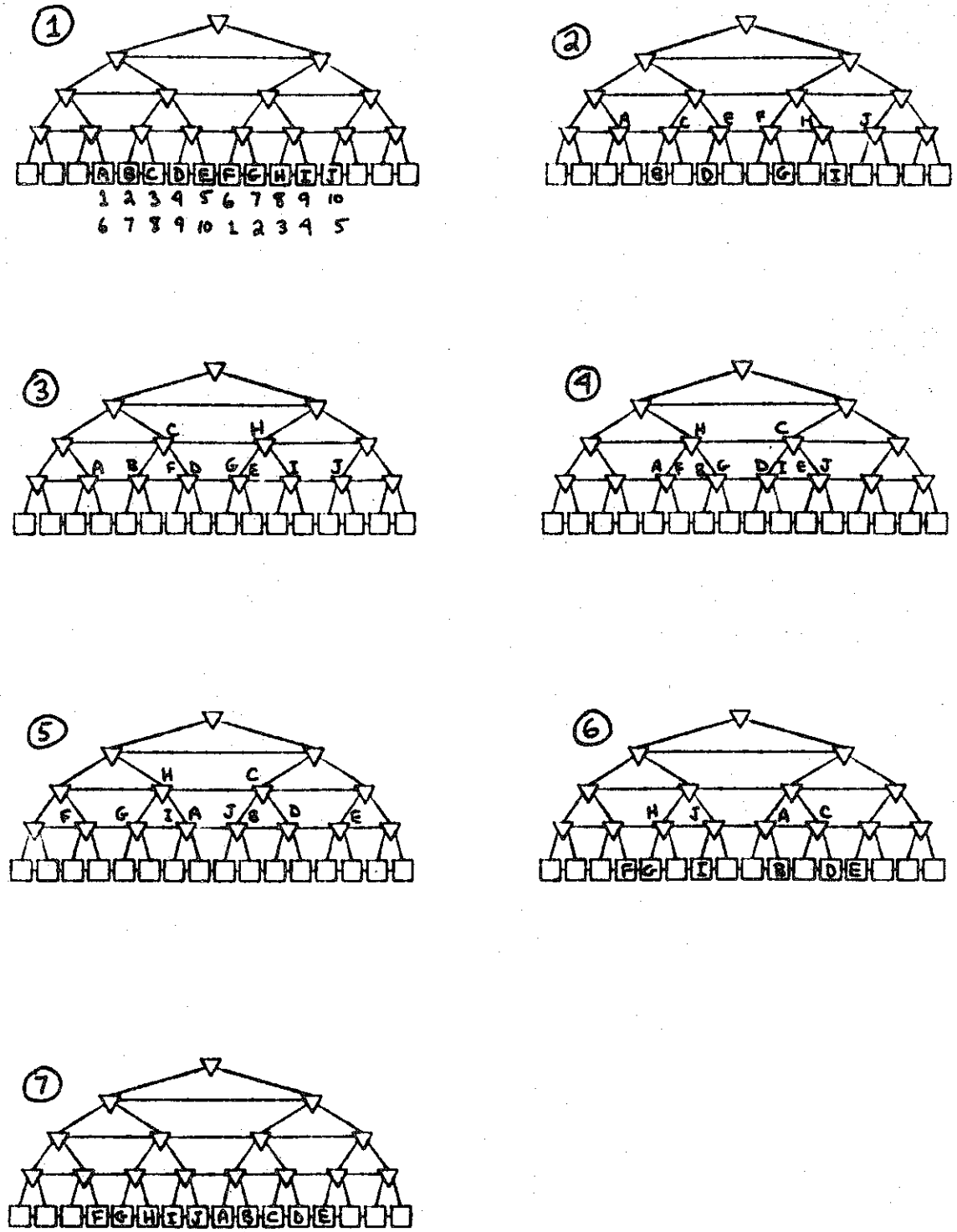


Figure 1.10

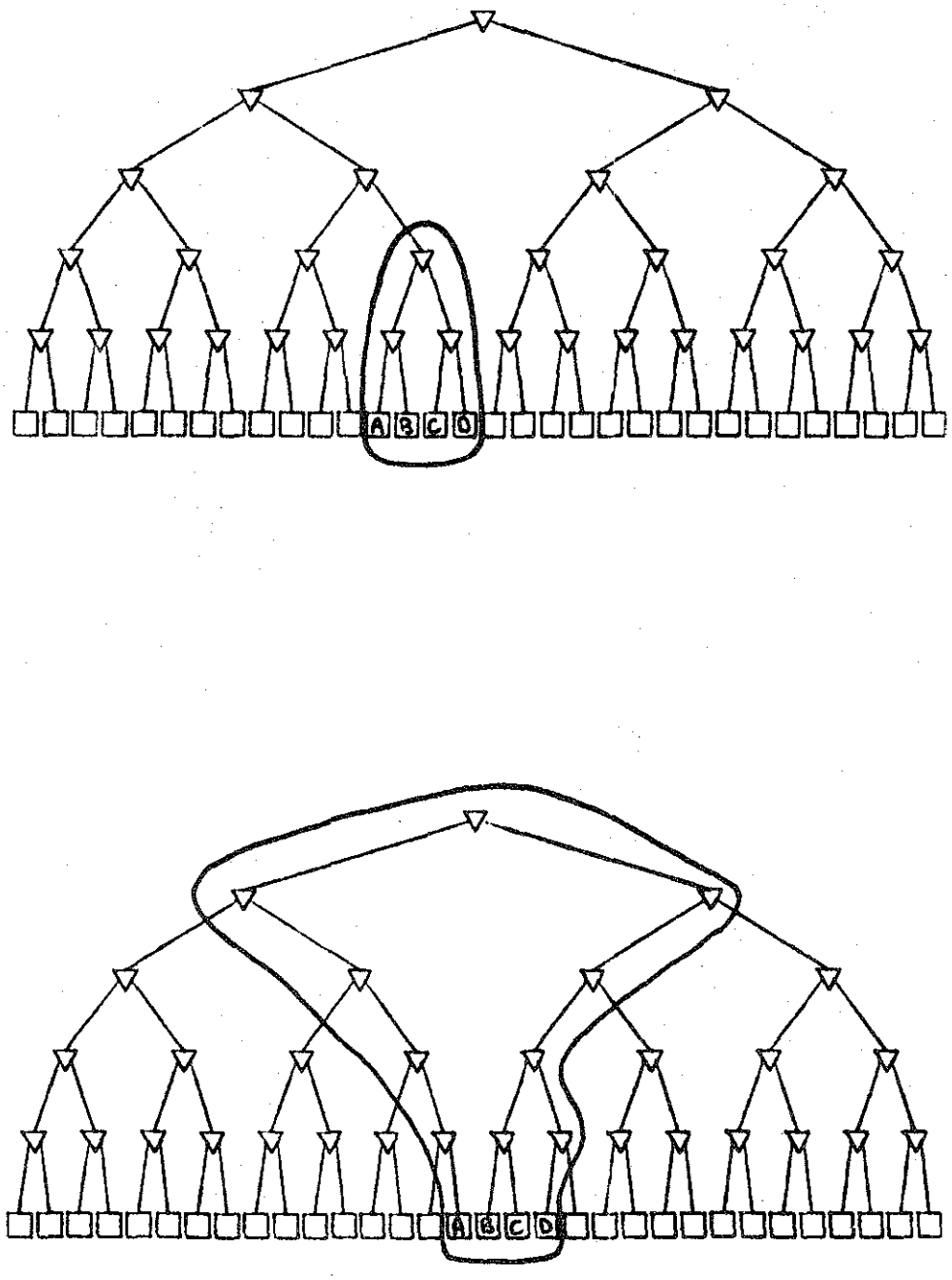


Figure 1.11

2. SHORTEST PATHS IN THE NETWORK

As a first step toward a solution, we might consider the routing for a single source item. In the VERTICAL algorithm, the item is sent to the top node of the tree and then "showered" to every cell of L. Then any cell which is waiting for that item can accept it while the others ignore it.

With horizontal connections, it is no longer necessary to send the data item all the way to the top. Instead, it can be broadcast in waves as shown in Figure 2.1. The numbers in the diagram are included as an aid to the reader and do not represent information which must be stored in the cells. The data item starts in the cell labeled "0". In the first cycle, it spreads to all the adjacent cells (marked by "1"). Then it spreads to the cells marked "2", etc. Eventually, the data item will spread to all the cells of T and L. With this scheme, the number of steps used to reach any cell in L is always a minimum. That is, the data item always travels from the source to the target cell along one of the shortest paths between the two. (There may be

more than one shortest path.)

Chapter 3 will discuss techniques for routing items according to patterns similar to that of Figure 2.1. Since these techniques involve shortest paths, or nearly shortest paths, we will first develop some mathematical results about paths in trees with horizontal connections. The presentation of these results is intended to be rigorous, yet informal. Accordingly, we will use the following informal definitions.

The tree is treated as an undirected graph whose nodes are the cells of T and L and whose edges are the connections between cells as shown in Figure 1.5. A path between two cells is an alternating sequence of cells and edges which connect the two cells. It is assumed that paths contain no loops. That is, a given edge may appear only once in the sequence. The length of a path is the number of edges in the path. In what follows, an edge is sometimes referred to as a "step". Therefore, the number of steps on a path between two cells is the same as the length of the path between the cells. The edges which connect fathers and sons in the tree are called vertical edges while those which connect brother cells are known as horizontal edges. A vertical path is a path which contains only vertical edges. Similarly, a horizontal path is one which contains only horizontal edges. If one cell is a descendant of another,

the length of the path connecting the two cells is called the vertical distance between them. We say that two cells are on the same level if the vertical distance between each of the cells and one of their common ancestors is the same. The levels of the tree are numbered from the bottom up. That is, the cells of L are on level 1, the fathers of the cells of L are on level 2, and so on. When two cells are on the same level, the length of the horizontal path which connects them is known as the horizontal distance between the cells. Finally, the notation ANC(A,k) is defined to mean "cell A's ancestor on level k."

Theorem 1. Let A and B be two cells on the same level of T which are separated by a horizontal distance of n. Then the father of A and the father of B are separated by a horizontal distance of:

n/2 if n is even

FLOOR(n/2) or CEIL(n/2) if n is odd.

Proof:

Case I. n is even.

The proof is by induction on n. For n=2, we have one of the configurations of Figure 2.2. For both of these cases, the assertion holds. Now, suppose the assertion is true for n (n even). Let the horizontal distance between A and B be n+2 and let C be the cell two units from B (in the direction of A), as shown in Figure 2.3. Then the distance from A's

father to B's father

=dist. from A's father to C's father

+ dist. from C's father to B's father

= (n/2) + 1 (by induction hypothesis)

= (n+2)/2.

Case II. n is odd. Let the horizontal distance between A and B be n and let C be the cell adjacent to B (in the direction of A). Then the distance from A to C is n-1, which is even. If B and C have the same father (Figure 2.4) then the distance from A's father to B's father is (n-1)/2 = FLOOR(n/2). If B and C have different fathers (Figure 2.5) then the distance from A's father to B's father is (n-1)/2+1 = (n+1)/2 = CEIL(n/2).

Q.E.D.

Theorem 2. Let A and B both be at level k, separated by a horizontal distance of n. Then there are more than n steps in any path from A to B which uses some cells on levels below k, but no cells on levels above k.

Proof: Since A and B are separated by n horizontal steps, there are n-1 cells between them on level k. Call these X(1), X(2), ... X(n-1). Then the path from A to B which uses only cells on level k or lower must contain the following segments:

-A path from some descendant of A to some descendant of X(1).

-For i=1,2,...,n-2, a path from some descendant of X(i) to some descendant of X(i+1).

-A path from some descendant of X(n-1) to some descendant of B.

Each of these paths must contain at least one horizontal step. Furthermore, any path which uses cells below level k must have at least 2 vertical edges (one edge to get from level k to level k-1 and one edge to return to level k). Therefore, any path from A to B which uses levels below k must have at least n+2 steps.

Q.E.D.

Theorem 3. Suppose A and B are two cells in T or L which are at the same level and are separated by a horizontal distance of n>4. Then there is a path from A to B of length less than n which consists of the edge from A to its father, a path from the father of A to the father of B, and the edge from the father of B to B.

Proof: The proof consists of a construction of the required path. It consists of paths from:

-A to his father (1 edge)

-A's father to B's father (no more than CEIL(n/2) edges by Theorem 1)

-B's father to B (1 edge)

So the total path length is no more than CEIL(n/2)+2. But for n>4, CEIL(n/2) + 2 < n Q.E.D.

Theorem 4. Suppose A and B are as in Theorem 3. Then one of the shortest paths between A and B consists of the edge from A to his father (denoted by C), a path from C to the father of B (denoted by D) and the edge from D to B.

Proof: Suppose A and B are on level k, with A to the left of B. Then by Theorem 2, no shortest path between A and B can use any level below k. The path between A and B which remains on level k has n steps. But Theorem 3 shows how to construct a shorter path using higher levels. Therefore, any shortest path between A and B must use level k+1. Consider an arbitrary shortest path between A and B. Let E be the leftmost level-(k+1) cell on this path and let F be the rightmost level-(k+1) cell. (See Figure 2.6.) If C=E and F=D, then we are finished. If not, then let G be the cell on level k which immediately precedes E on the path. (G will be the left son of E.) By Theorem 2, it follows that the arbitrarily chosen shortest path must begin with a horizontal path from A to G followed by the edge from G to E. We will call this initial sub-path A...GE. But the path which consists of the edge connecting A to C followed by a horizontal path from C to E (we will call this path AC...E) is no longer than A...GE because the horizontal distance between C and E is no longer than the horizontal distance between A and G (by Theorem 1). Therefore, AC...E is a shortest path from A to E. Similarly, we can show that the path consisting of the edge connecting D to B preceded by

the horizontal path from F to D (we will call this path F...DB) is a shortest path from F to B.

Using these facts, we can construct a new path which consists of the path AC...E, followed by the path from E to F which uses the same edges as the original arbitrarily chosen path, followed by the path F...DB. This new path is a shortest path which uses cells C and D.

Q.E.D.

Theorem 5. Let k be the lowest level such that the horizontal distance between ANC(A,k) and ANC(B,k) is less than or equal to 4. Then the path from A to ANC(A,k), from ANC(A,k) to ANC(B,k) (horizontally), and from ANC(B,k) to B is a shortest path from A to B.

Proof: The proof is by induction on the horizontal distance between A and B. If A and B are separated by 4 or fewer edges, then k is such that ANC(A,k)=A and ANC(B,k)=B. In this case, A and B must fit one of the configurations of Figure 2.7, or a mirror image of one. For all of these configurations, it is easy to verify that the horizontal path from A to B is a shortest path.

Now suppose the horizontal distance between A and B is n, where n>4. Let C be the father of A and let D be the father of B. By Theorem 4, we know that a shortest path from A to B contains the edges AC and DB. But the

horizontal distance between C and D is less than n (by Theorem 1) so the induction hypothesis applies to D and C. This means that one shortest path from C to D consists of the vertical path C...ANC(C,k), the horizontal path ANC(C,k)...ANC(D,k) and the vertical path ANC(D,k)...D. But ANC(C,k)=ANC(A,k) and ANC(D,k)=ANC(B,k) so A...ANC(A,k)...ANC(B,k)...B is a shortest path between A and B.

Q.E.D.

It will be useful to examine the value of k, as defined in Theorem 5, given the distance between A and B. The following table shows the value of k for some small values of n (the horizontal distance between A and B).

n	k
1	1
2	1
3	1
4	1
5	2
6	2
7	2
8	2
9	2 or 3
10	3
11	3
12	3
13	3
14	3
15	3
16	3
17	3 or 4
18	3 or 4
19	3 or 4
20	4
21	4
22	4
23	4
24	4

The first four entries are derived from the fact that for $n \leq 4$, the path of Theorem 5 will have no vertical edges. The other entries can be found using Theorem 1. For example, for n=9, the father of A and the father of B are separated by four or five steps so the path from the father of A to the father of B will use one or two levels. Adding one more level to get from A to his father, we get two or three levels for n=9. The general formula for k follows by induction.

Theorem 6. Let A, B, and k be as in Theorem 5, with A and B on level 1 separated by a horizontal distance of n. Then, using the notation m=CEIL(log(n)), the value of k is:

m-1 or m-2 if n < 2**(m-1) + 2**(m-3), and

m-1 otherwise.

Proof: The proof is by induction on k. For small values of k, the theorem can be established by inspection.

Suppose cells A and B are separated by a horizontal distance of n, and k is as in Theorem 5. Consider C, a son of A, and D, a son of B. These sons are separated by a horizontal distance of N, which must equal 2n-1, 2n, or 2n+1. If K is the lowest level at which ANC(C,K) and ANC(D,K) are separated by four or fewer horizontal steps, then K=k+1. Let M=CEIL(N). To establish the theorem, we must compute K in terms of M. There are four cases corresponding to different possible values of n:

I. n < 2**(m-1)+2**(m-3). Then k=m-1 or m-2, by induction.

Multiplying by 2, we get

2n < 2**m+2**(m-2).

Since both sides of this inequality represent even integers, we also have

2n+1 < 2**m+2**(m-2).

But N≤2n+1 and M=m+1, so

N < 2**(M-1)+2**(M-3).

Since k=m-1 or m-2 (by induction) and K=k+1, we have K=M-1 or M-2.

II. n=2**(m-1)+2**(m-3). There are two subcases, depending on the value of N. In both cases, k=m-1 (by induction) and M=m+1.

1. N=2n-1. Then N=2**(M-1)+2**(M-3)-1 and K=M-1.

2. N≥2n. Then N ≥ 2**(M-1)+2**(M-3) and K=M-1.

III. 2**(m-3)+2**(m-1) < n < 2**m. The restriction n<2**m assures that M=m+1. Multiplying by 2, we get

$$2n > 2^{**}m+2^{**}(m-2)$$

which implies

$$2n-1 > 2^{**}m + 2^{**}(m-2)$$

since both sides of the first inequality represent even integers. Since N≥2n-1 and M=m+1, we have

$$N > 2^{**}(M-1)+2^{**}(M-3) \text{ and } K=M-1.$$

IV. n=2**m. Again, there are two sub-cases.

1. N=2n-1 or N=2n. Then since M=m+1, we have N=2**M-1 or N=2**M and since k=m-1 (by induction), K=M-1.

2. N=2n+1. In this case, M=m+2 so N=2**(m+1)+1=2**(M-1)+1 and since k=m-1, K=M-2.

Now, in cases I, II.1, and IV.2, we have N<2**(M-1)+2**(M-3). In each of these cases, K=M-1 or K=M-2. Similarly, in cases II.2, III and IV.2, we have 2**(M-1)+2**(M-3) ≤ N ≤ 2**M. In each of these cases, K=k-1.

Q.E.D.

Theorem 7. Let A and B be two cells in L and let k be the lowest level such that ANC(A,k) and ANC(B,k) are separated by two or fewer steps. Then the path that rises from A to ANC(A,k), moves horizontally to ANC(B,k), and then down to B is at most one step longer than a shortest path from A to B.

Proof: Let C=ANC(A,k), D=ANC(B,k), E=ANC(A,k-1), and F=ANC(B,k-1). Then C, D, E, and F must be arranged in one of the configurations of Figure 2.8. Let A...E...F...B refer to the path which rises from A to E, moves horizontally to F, and descends to B and let A...C...D...B refer to the path which rises from A to C, moves horizontally to D and descends to B.

In cases (i) and (iii), A...E...F...B is a shortest path by Theorem 5. But A...C...D...B has the same length and is therefore a shortest path also. In case (ii), A...C...D...B is a shortest path and in case (iv), A...E...F...B is a shortest path and A...C...D...B is one step longer than A...E...F...B. Therefore, in each case, A...C...D...B is a shortest path or has one more step than a shortest path.

Q.E.D.

Theorem 8. Let A, B, and k be as in Theorem 7, with A and B on level 1 separated by a horizontal distance of n. Then, using the notation m=CEIL(log(n)), the value of k is:

m-1 or m-2 if n < 2**m - 2**(m-2), and

m-1 otherwise.

Proof: As with Theorem 6, the result follows by induction.

Q.E.D.

Theorem 9. Let A and B be two cells in L and let k be the

lowest level such that the horizontal distance between ANC(A,k) and ANC(B,k) is one. Then the path that rises from A to ANC(A,k), moves horizontally to ANC(B,k), and then down to B is at most two steps longer than a shortest path from A to B.

Proof: If $k \leq 2$, then the theorem can be established by inspection. Otherwise, we will use the same notation as in the proof of Theorem 7, with G=ANC(A,k-2) and H=ANC(B,k-2). A...G...H...B will denote the path which rises vertically to G, moves horizontally to H, and descends vertically to B. Then C, D, E, F, G, and H must be configured as in Figure 2.9. In cases (i), (ii), and (iii), the path A...E...F...B is a shortest path by Theorem 5 and the path A...C...D...B has the same length. In case (iv), the path A...C...D...B is one step longer than A...E...F...H, which is a shortest path. In cases (v) and (vi), the path A...C...D...B is one step longer than the shortest path A...G...H...B. In case (vii), the path A...C...D...B is two steps longer than the shortest path A...G...H...B. Therefore, in each case, the path A...C...D...B is a shortest path or has one or two more steps than a shortest path.

Q.E.D.

Theorem 10. Let A, B, and k be as in Theorem 9, with A and B separated by a horizontal distance of n. Then, using the notation m=CEIL(log(n)), the value of k is:

m if n=2**m, and

m or m-1 otherwise (i.e. if n<2**m).

Proof: Again, as with Theorem 8, we omit the details of the inductive proof.

Q.E.D.

Theorem 11. Let A and B be cells of L. Let k be a level such that ANC(A,k) and ANC(B,k) are separated by n edges, n≥4. Then every shortest path between A and B must include some cells on level k.

Proof: If k=1, then the theorem is trivially true because ANC(A,k)=A. Suppose k>1 and consider any path from A to B which does not use level k. Suppose the highest level used by this path is m, m<k. (See Figure 2.10.) Then, since ANC(A,m) and ANC(B,m) are separated by at least (n-1)*2**(k-m)+1 horizontal edges, this path must have at least 2*m+(n-1)*2**(k-m)+1 steps. To get this number, notice that ANC(A,m) and ANC(B,m) are separated by n-1 sub-trees, each of which has 2**(k-m) leaves. But there is a path using ANC(A,k) and ANC(B,k) which takes 2*m + 2*(k-m) + n steps. The path using level k is shorter than the one which used only level m because

$$(n-1)*2**(k-m) + 1 > 2*(k-m) + n$$

for $n \geq 4$ and $k-m > 0$. Therefore, level k must be used for the shortest path.

Q.E.D.

Theorem 12. Let A be in L and let B=ANC(A,k) for some $k \geq 1$. Let C be adjacent (horizontally) to B and suppose D is n horizontal edges away from B on the same side as C, $n \geq 1$. (See Figure 2.11.) Then any shortest path from A to D must use cell B or cell C.

Proof: If k=1, then A=B, so any shortest path must use B. Suppose $k>1$. Then the shortest path between A and D must include some cell on level k (if only D). Without loss of generality, assume B is to the left of D, as in Figure 2.11. Let E be the leftmost cell on level k which is on some shortest path. We will show that E is either B or C. Clearly, E can not be to the left of B. (If it were, we could construct a shorter path which did not use E. But E is assumed to be on some shortest path.) If E is not one of B or C, then there must be $m \geq 2$ horizontal steps from B to E. (See Figure 2.12.) The (supposedly shortest) path A...E must then include k vertical steps plus the following horizontal steps:

- -a descendant of B to a leftmost descendant of C (at least 1 step)
- -leftmost descendant of C to rightmost descendant of C (at least 1 step)

-rightmost descendant of C to a descendant of E (at least m-1 steps since C and E are separated by m-1 steps)

(See Figure 2.12.) This gives a total of at least 1+1+(m-1) = m+1 horizontal steps. But we can easily construct a path from A to E using only m horizontal steps. (This path rises directly to A and then moves horizontally to E.) Therefore, the assumption that E is not one of B or C must be false.

Q.E.D.

Theorems 11 and 12 give us a method of counting the number of shortest paths between two cells of L.

Theorem 13. Let A and B be cells of L which are separated by a horizontal distance of n. Let k be the highest level such that the horizontal distance between ANC(A,k) and ANC(B,k) is greater than or equal to four. If $n<4$, then let k=1. Then every shortest path from A to B consists of three segments:

(1) A segment from A to C where C is either ANC(A,k) or the cell adjacent to ANC(A,k) (towards B)

(2) A segment from C to D, where D is either ANC(B,k) or the cell adjacent to ANC(B,k) (towards A)

(3) A segment from D to B

such that the horizontal distance between ANC(A,k) and ANC(B,k) is greater than or equal to four. (See Figure 2.13.)

Proof: The theorem is trivially true if $n<4$. If $n\geq 4$, then by Theorem 11, every shortest path must use some cells on level k and by Theorem 12, every shortest path must use ANC(A,k) or the cell adjacent to it, and ANC(B,k) or the cell adjacent to it. Therefore, C and D, as defined above must be intermediate cells on any shortest path.

Q.E.D.

Theorem 14. Let A be in L and let C be the right (or left) brother of ANC(A,k)=B, where $k\geq 1$. (See Figure 2.14.) Then the number of shortest paths between C and A is one more than the number of consecutive right (or left) branches beginning at B in the path from B to A.

Proof: Consider the diagram of Figure 2.15. A shortest path from A to C may use any one of the hatched cross edges to get to a descendant of C, and then go up to C. The number of cross paths is the same as the number of consecutive right (or left) branches.

Q.E.D.

Corollary. An algorithm for determining the number of shortest paths between two cells A and B which are separated by a horizontal distance of n in L.

1. If $n<4$, let $k=1$. Otherwise, let k be the highest level such that ANC(A,k) and ANC(B,k) are separated by a horizontal distance greater than or equal to four. Let

C, D, E, and F be as shown in Figure 2.16.

2. Let s(AE) = # of shortest paths from A to E which do not use C, as computed in Theorem 14. (s(AE) could be 0.)
   Let s(BF) = # of shortest paths from B to F which do not use D, as computed in Theorem 14. (s(BF) could be 0.)
   Let s(CF), s(CD), s(EF), s(ED) be the numbers of shortest paths from C to F, C to D etc. (Since C and D are separated by at most seven steps, a finite table could be constructed to compute these values.)
3. Then the total number of shortest paths from A to B is:
   s(CF) * s(FB)
   + s(CD)
   + s(AE) * s(EF) * s(FB)
   + s(AE) * s(ED)

Proof: Any shortest path is of the form A...X...Y...B, where X=C or E, and Y=F or D. The number of such paths is s(AX) * s(XY) * s(YB) . Substituting all possible values for X and Y and using the fact that s(AC)=s(DB)=1, we get the result.

Q.E.D.

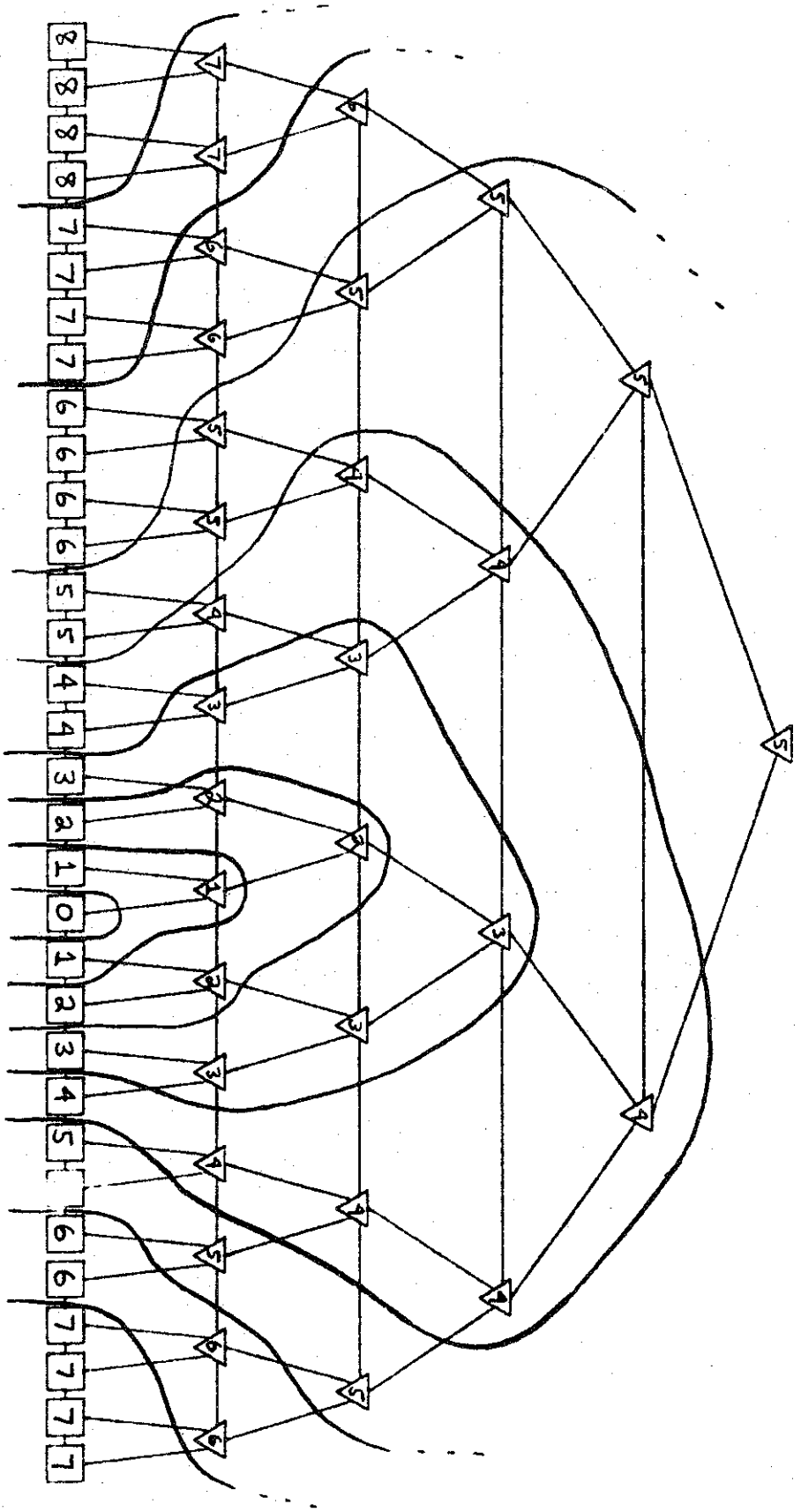


Figure 2.1

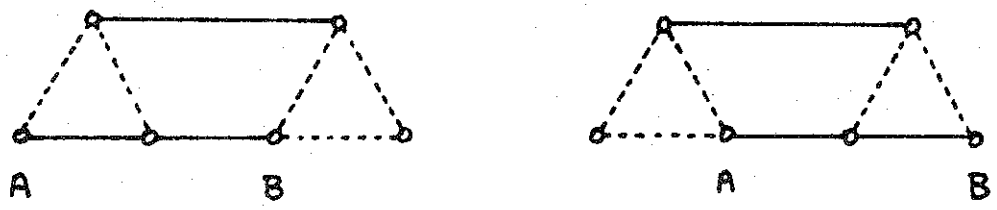


Figure 2.2

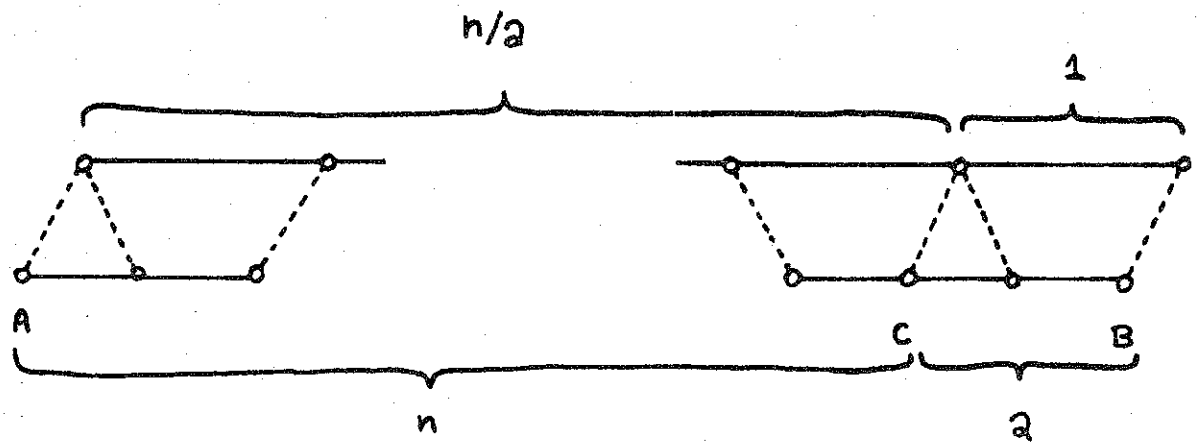


Figure 2.3

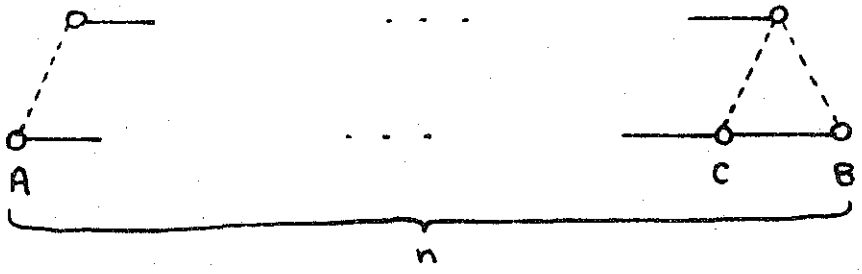


Figure 2.4

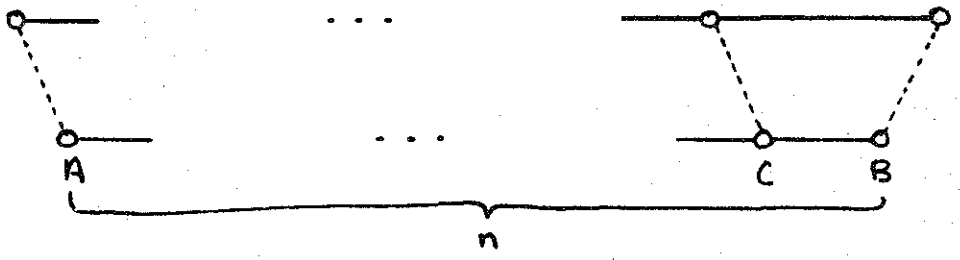


Figure 2.5

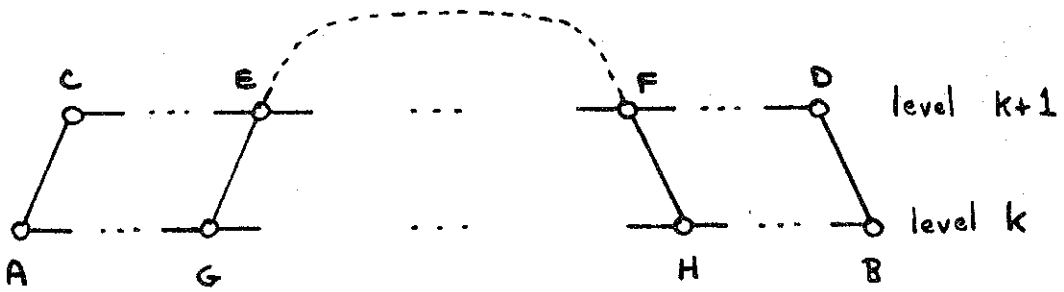


Figure 2.6

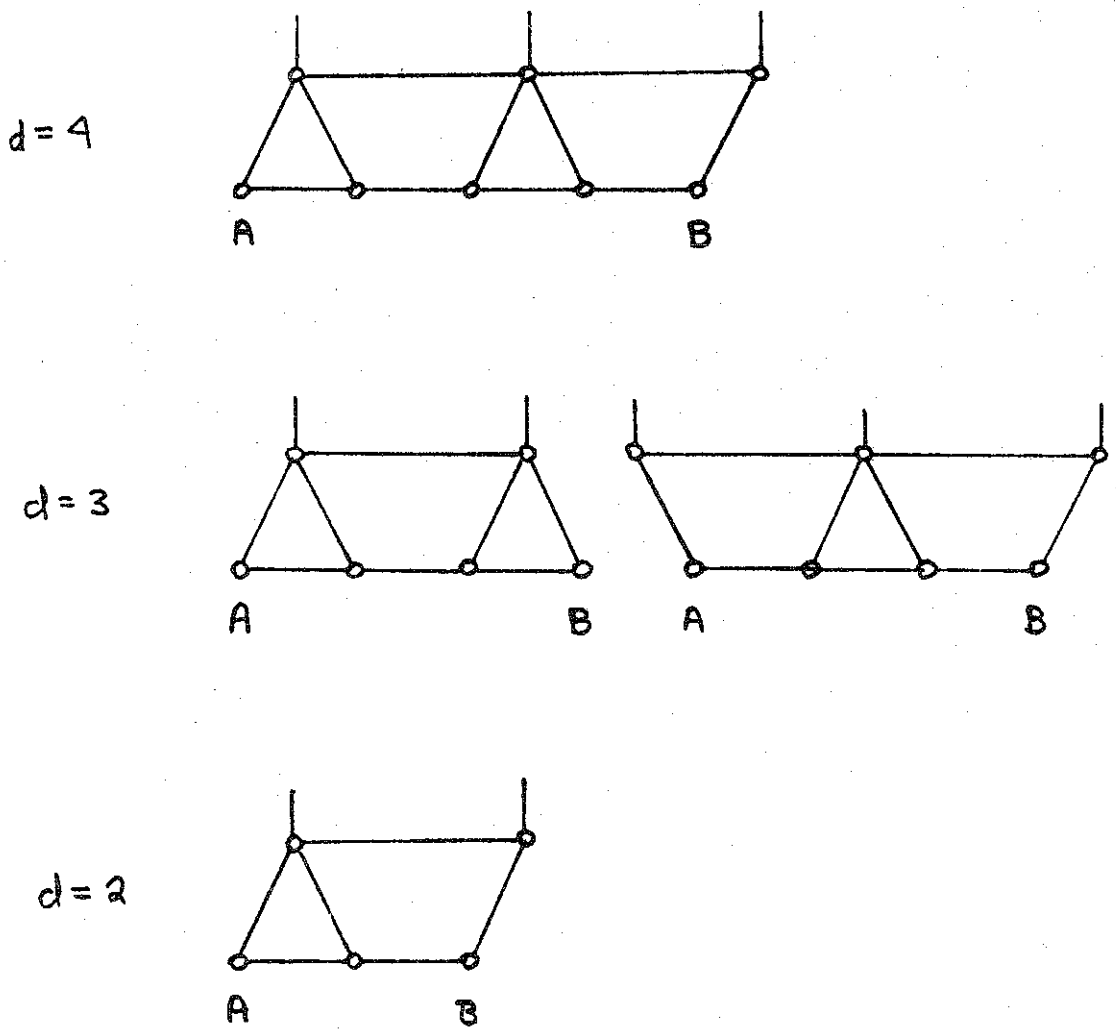
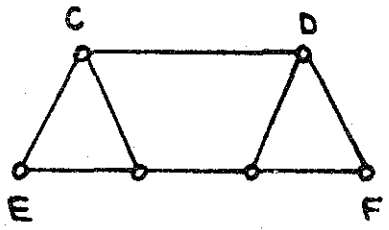
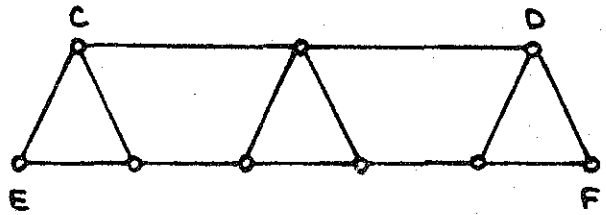


Figure 2.7

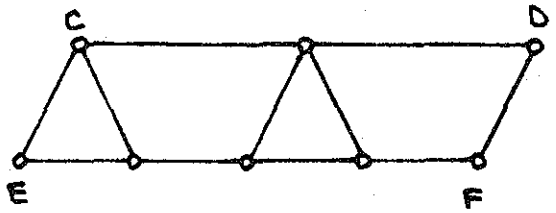
(i)



(ii)



(iii)



(iv)

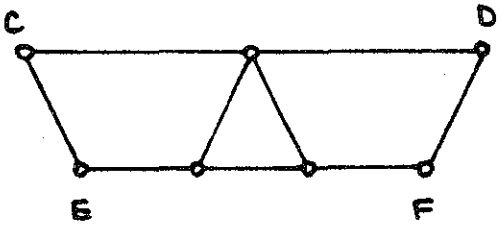
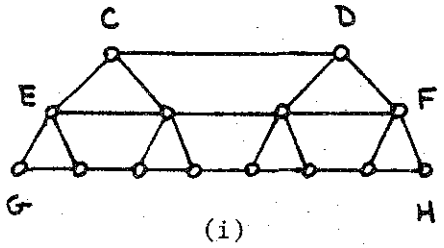
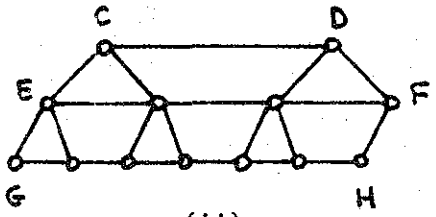


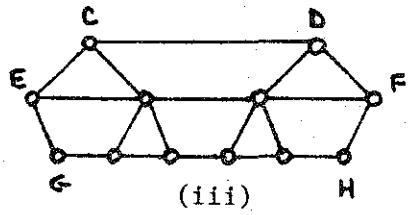
Figure 2.8



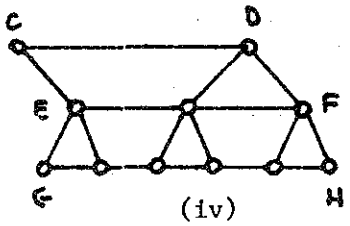
(i)



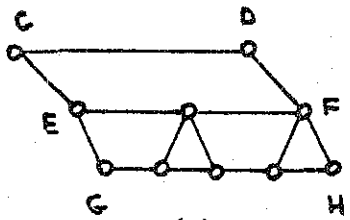
(ii)



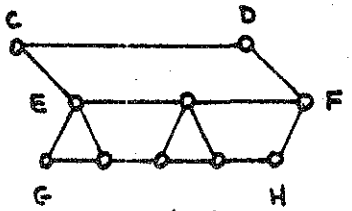
(iii)



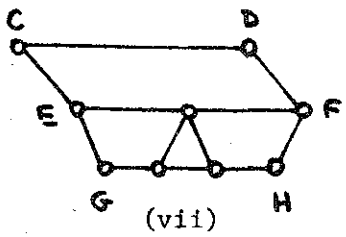
(iv)



(v)



(vi)



(vii)

Figure 2.9

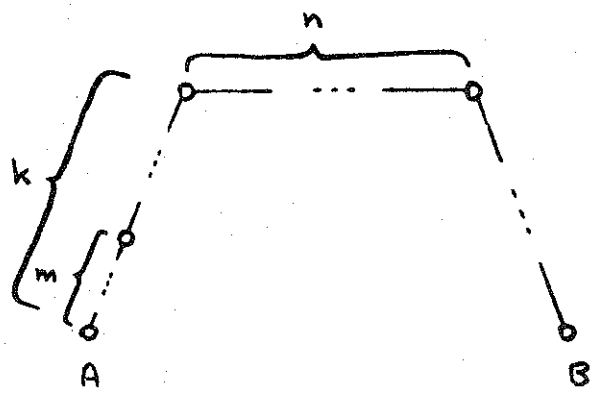


Figure 2.10

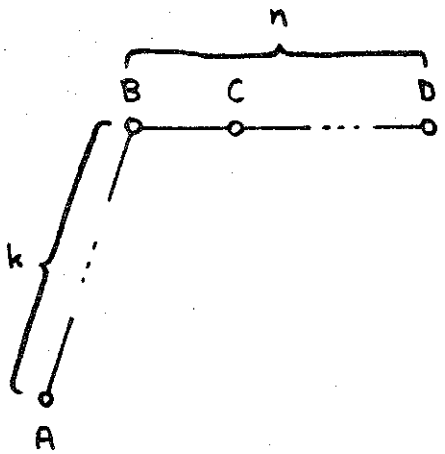


Figure 2.11

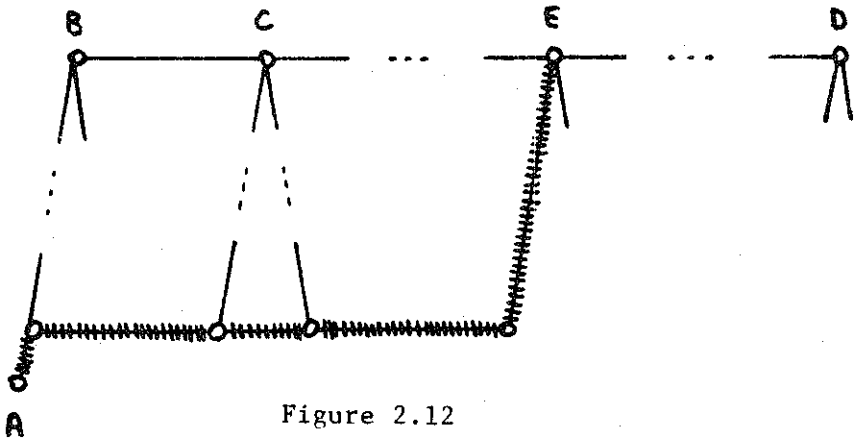


Figure 2.12

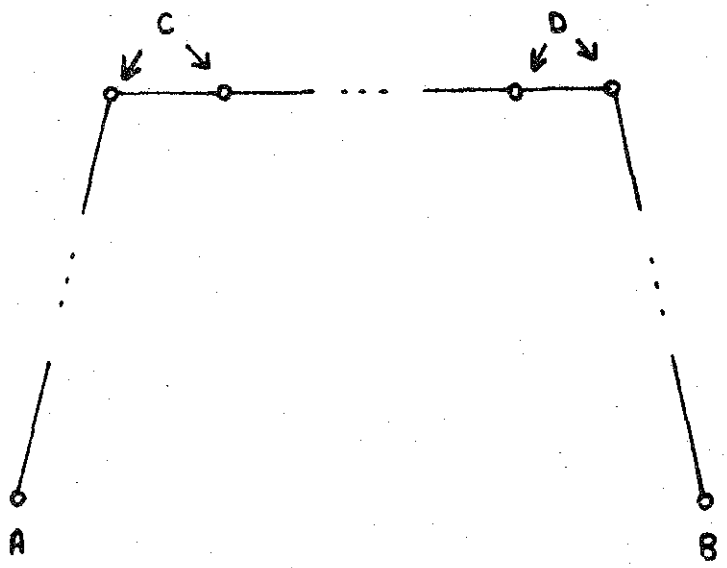


Figure 2.13

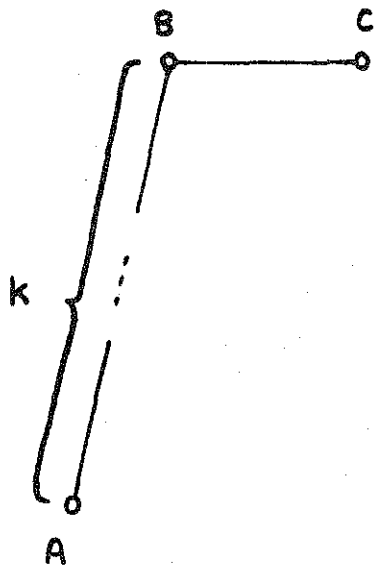


Figure 2.14

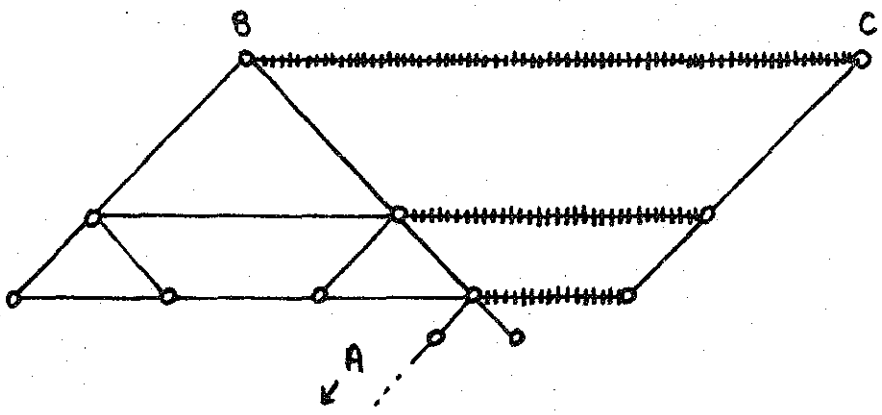


Figure 2.15

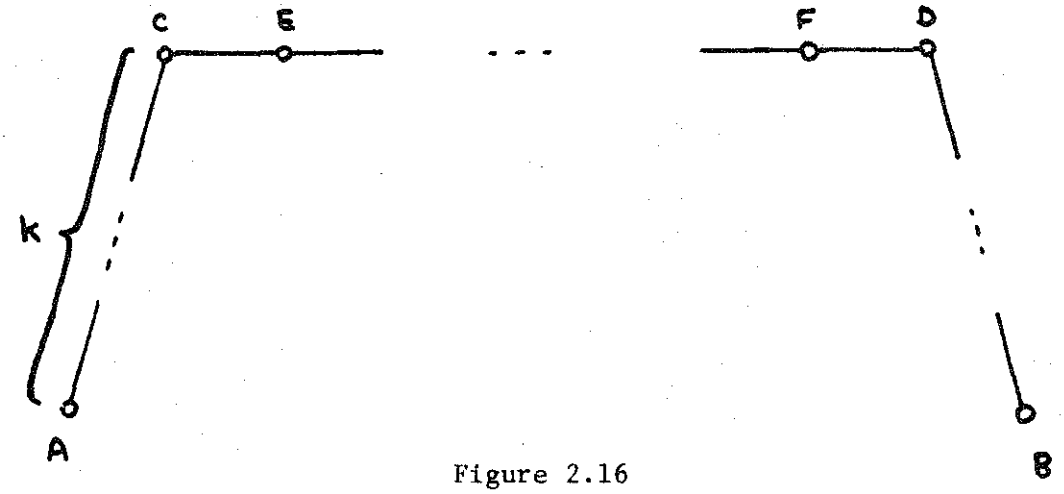


Figure 2.16

3. DATA MOVEMENT IN A NETWORK WITHOUT MEMORY

3.1 Algorithms for One Data Item

In this chapter, we discuss techniques for broadcasting items to every cell of L. We will assume that the cells of T have no memory. That is, there will be no information in these cells to guide a data item to a particular target cell of L. Instead, copies of the item will be broadcast to all of the cells of L. Those cells in L which need the item will accept it while the rest of the cells will discard it. In Chapter 4, we will consider techniques which utilize memory in the cells.

To implement a broadcast algorithm, we need rules which each cell will use for deciding to which of its neighbors it should send the data item. This decision will be based on the direction from which the data item arrived and the previous history of the item. Broadcasting is accomplished by Algorithm SS1, in which each copy of the data item will carry a "state" bit to keep track of its recent history.

Algorithm SS1. (Single Source data movement)
This algorithm broadcasts a data item throughout the tree in

such a way that each cell of T and L receives exactly one copy of the data item.

There are two possible states which will be called "From loson" and "From hison". Initially, the data item occupies a source cell in L and is treated as if it had arrived from the left(lo) son. It is assigned the state "From loson". During each cycle, each cell applies the following rules, which are diagrammed in Figure 3.1, to decide where to send the data item it contains (if any):

(a) If the data item arrived from the left(lo) son, assign it the state "From loson" and send copies of it to the father, the left brother and the right brother.

(b) If the data item arrived from the right(hi) son, assign it the state "From hison" and send copies of it to the father, the left brother, and the right brother.

(c) If the data item arrived from the left brother, then send a copy of it to the right(hi) son. If the state of this item is "From loson", then send another copy to the left(lo) son. In either case, erase the state bit. (Erasing the state implies that the state information is no longer needed. The state bit can be set to an arbitrary value.)

(d) If the data item arrived from the right brother, then send a copy of it to the left(lo) son. If the state of this item is "From hison" then send another copy to the right(hi) son. In either case, erase the state bit.

(e) If the data item arrived from the father, then send copies of the item and its state bit (which will have been erased by this time) to both sons.

The paths created by this algorithm for a tree of size 16 and a data item originating in the 8th cell of L are shown in Figure 3.2. Dotted lines represent connections not used by copies of this data item.

Proposition. Using Algorithm SS1, exactly one copy of the data item will be broadcast to every cell of T and L.

Proof: The proof is by induction on h, the height of the active area. The specific inductive assertion is that each horizontal level of the active area of the tree consists of:

- n cells which receive exactly one copy from above (n may be 0.)
- 0 or 1 cells which receive exactly one copy from the right (if this number is 0, then n=0.)
- 1 cell which receives one copy from below
- 0 or 1 cells which receive exactly one copy from the left.
- m cells which receive exactly one copy from above (m may be 0.)

(See Figure 3.3.)

The assertion can be verified directly for h=1. Assume the assertion holds for the upper h levels and consider the (h+1)th level (from the top). Let A be the cell on level h

(from the top) which receives one copy from below. Let B be the cell which sends the copy to A. Since the rules are symmetrical, we need only treat the case where B is the left son of A. Finally, let C,D,E,F, and G be as in Figure 3.4. (If the active area contains fewer than 6 cells on this level, then some of C,D,E,F, and G may be missing.) Then B must send items according to rule (a) or (b) since a copy is sent upwards only with rules (a) and (b). In either case, C and D will receive exactly one copy from the left and right, respectively (by rule (a) or (b)). Cell E will receive exactly one copy from cell H by rule (d). (If E is part of the active area, then H must be also.) Cells F and G will receive exactly one copy each from cell J by rule (c). Each of the other cells will receive exactly one copy from above by rule (e). This establishes the induction assertion for level h+1 (from the top).

Q.E.D.

Algorithm SS1 then can be used to broadcast a data item to every cell of the tree along a path which has exactly one horizontal edge. That is, the path between the source cell A and a target cell B in L will consist of a path going up from A to one of its ancestors, then across one horizontal edge to an ancestor of B and then down to B. To see this, consider any level k and let C=ANC(A,k) and D=ANC(B,k). If C and D are separated by one horizontal edge, then the path from A to B will use this edge because of rules (a) or (b).

Otherwise, C will send a copy upwards and D will receive its copy from above so the path from A to B will use no horizontal edges on level k.

Proposition. The path followed by a data item in Algorithm SS1 is no more than two steps longer than a shortest path. Proof: This follows immediately from the above discussion and Theorem 9 (Chapter 2).

Q.E.D.

Algorithms which broadcast along shorter paths can be developed by allowing the data items to travel horizontally for more than one step. This will be done in Algorithms SS2, SS3, and SS4. The complete details of these algorithms are rather complicated. However, the operation of the algorithms is easy to understand with the aid of diagrams which show how the data items are routed at the "turning point". For example, Algorithm SS1 can be represented as shown in Figure 3.5. In Figure 3.5(a), an item arrives from the left(lo) son. By rule (a) of Algorithm SS1, copies of the data item will be sent both left and right to B and C respectively. (A copy will also be sent to A's father, not shown.) B will send its copy to its left son by rule (d). And C will send copies to both of its sons by rule (c). (The sons, of course, will send the item to their descendants by rule (e) until L is reached.) Figure 3.5(b) shows arrival from the right son and is symmetric to the

first. In discussing further algorithms, we will present only the diagram corresponding to Figure 3.5(a).

Algorithms SS2, SS3, and SS4. (Single Source data movement.)
These algorithms are represented by Figures 3.6, 3.7, and 3.8.

As with Algorithm SS1, these algorithms all have the property that exactly one copy of the data item will be sent to each cell of T. Furthermore, it follows from Theorems 5 and 7 that the path between a source and a target both in L will be a shortest path if Algorithm SS4 is used and will be no more than one step longer than a shortest path if Algorithm SS2 or Algorithm SS3 is used.

3.2 Algorithms for Multiple Data Items

Algorithms SS1-SS4 show how to broadcast a single data item through the network. In this section and the remaining chapters, we will investigate techniques for moving more than one item simultaneously. That is, different cells of L will contain different data items, each of which is aiming for a different target cell.

3.2.1 The Inevitability of Collisions

One approach to developing a multiple-item data movement algorithm is to use Algorithm SS4 or some other technique to identify a shortest path between each source and its corresponding target(s) and then route the data items along these paths. For some configurations, this may inevitably lead to conflicts, even if there are no extra copies of any of the data items (as there are using Algorithm SS4). That is, two data items may try to occupy the same register of a cell at the same time, thus forcing one of them to wait. For example, consider the pattern of Figure 3.9. By inspection, we see that the minimum number of edges between each source and its corresponding target is six. But:

Proposition. If every data item travels along a shortest path to its target in the configuration of Figure 3.9, then simultaneous data movement cannot be accomplished in six steps.

Proof: (By contradiction) Movement in six steps is possible only if each data item can move along a shortest path. Suppose this is so. Then on the first step, A must move to cell #8 and C must move to cell #9 (because all shortest paths for A go through cell #8 and all shortest paths for B go through cell #9). In order to avoid conflict at step 1, B must choose for its shortest path 17-18-9-10-11-12-25 and

D must choose 19-20-10-11-12-13-27. These paths are shown in Figure 3.10. Therefore, after step 1, we must have the configuration of Figure 3.11. For step 2, B and D are committed to cells #9 and #10 respectively, since there are no other shortest paths given their start in step 1. Therefore, A must move to cell #4 (to follow shortest path 16-8-4-5-6-12-24). But since cells 4 and 10 are occupied, this leaves no second move for C. Therefore, C cannot follow any shortest path without having to wait.

Q.E.D.

This means that, in general, we cannot expect data movement to be completed in the number of steps it would take any one data item to move by itself. That is, for some configurations, some data item will have to wait or follow a longer path.

3.2.2 Algorithms MS1-MS4

One obvious way to handle more than one data item is to allow each item to move according to the rules of one of the Algorithms SS1-SS4. When a conflict occurs and two different items are routed to the same register of a cell, one will be required to wait. There are various possible schemes for deciding which item(s) should have to wait. The one adopted in our implementations is to give first priority to the item which has waited the longest, with ties broken

in favor of the lowest valued data item. The algorithms which can be obtained in this fashion from SS1-SS4 will be referred to as MS1-MS4 (for Multiple Source).

These algorithms solve some of the problems mentioned in the introduction. It is no longer necessary to have each item climb all the way to the top of the active area. As soon as the item gets high enough, it begins its descent. Therefore, the time for data movement no longer depends on the configuration of the active area, as was the case in the VERTICAL algorithm. (Recall Figure 1.11.) Also, if the active area can be partitioned into segments in such a way that the sources for all the targets in a segment are also in that segment, then the time for data movement will depend on the number of items in the largest segment rather than the total number of items moving. As an extreme example, swapping adjacent pairs (Figure 1.8) requires only two cycles using any of algorithms MS1-MS4, no matter how many pairs are to be swapped.

In the general case, however, MS1-MS4 offer little or no improvement over the VERTICAL algorithm. In fact, a simulation showed that MS2 required two more steps than the VERTICAL algorithm to reverse 64 items. It was not immediately obvious that this should happen since both the VERTICAL algorithm and MS2 broadcast every source item to every cell of T and L. The reason for the difference

involves the way in which the item is split into two (or more) copies. In the VERTICAL algorithm, a cell will make two copies of the data item and send them to both of its sons, who will always be ready to receive them. Therefore, the sending cell will always be free to accept another data item. However, in MS2, when a cell makes two copies of a data item, it is possible that one copy will have to wait. This means that the sending cell cannot accept another data item on this cycle. Therefore, it could (and sometimes does) take more than n-1 steps for n data items to move through a given cell.

To avoid these problems, we must find a way to limit the number of useless copies of data items which clog the network. This can be done by giving the cells some memory, as in the algorithms to be discussed in Chapter 4.

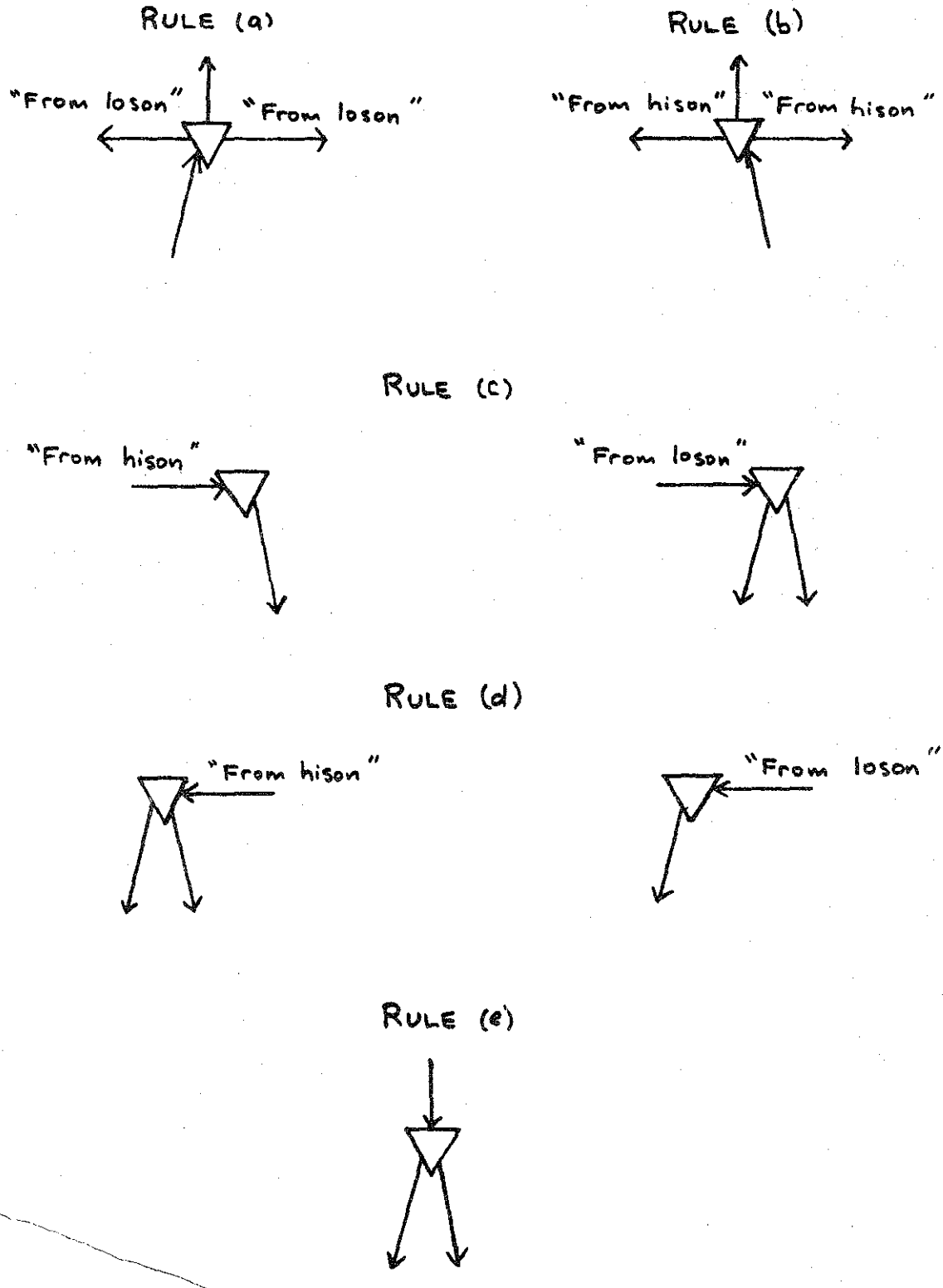

RULE (a)
"From loson"
"From loson"
RULE (b)
"From hison"
"From hison"
RULE (c)
"From hison"
"From loson"
RULE (d)
"From hison"
"From loson"
RULE (e)

Figure 3.1

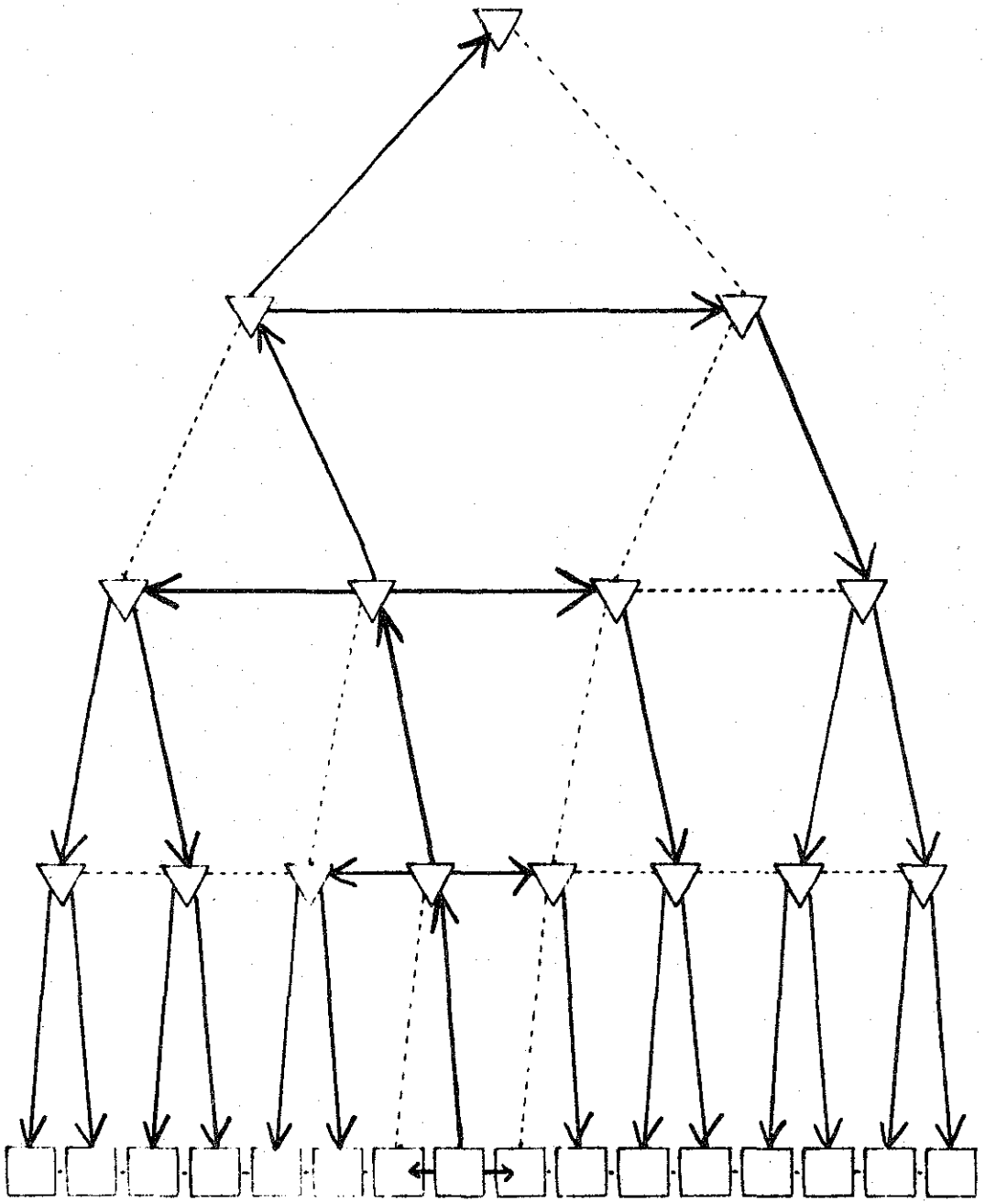


Figure 3.2

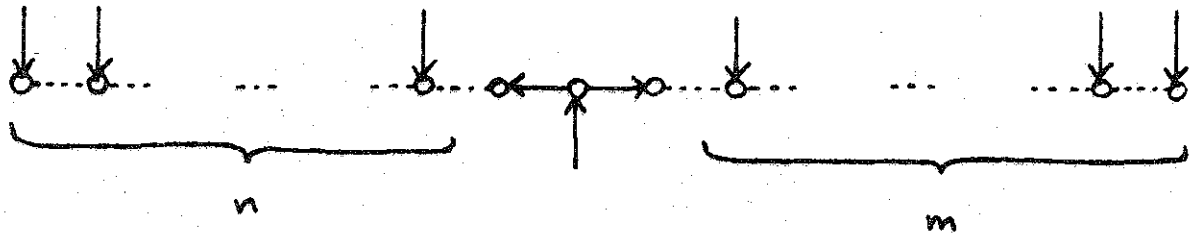


Figure 3.3

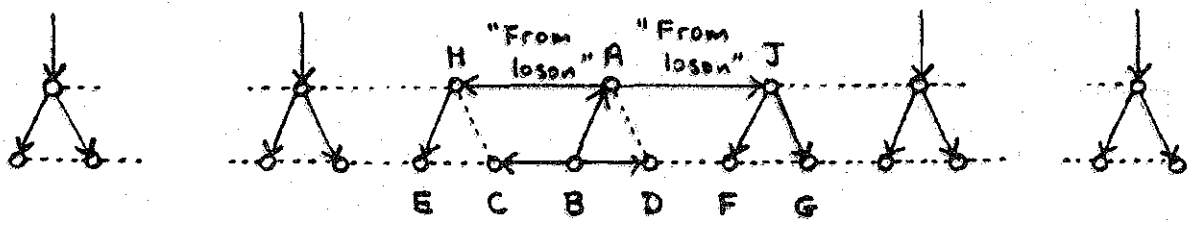
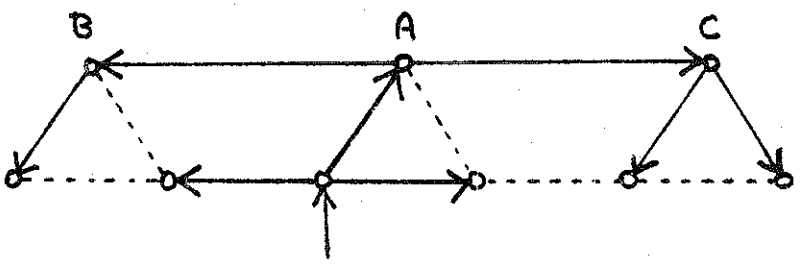
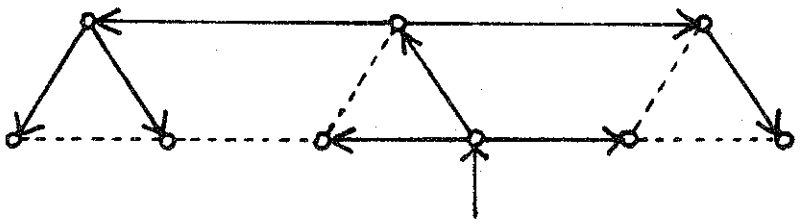


Figure 3.4



(a)



(b)

Figure 3.5

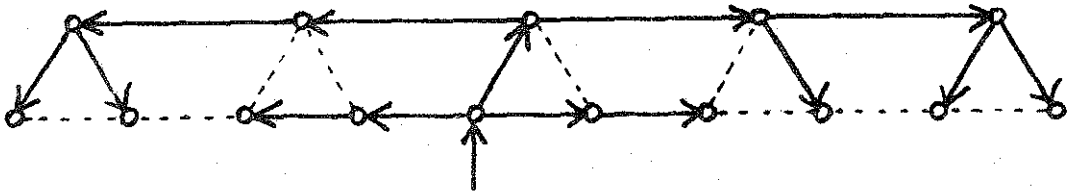


Figure 3.6

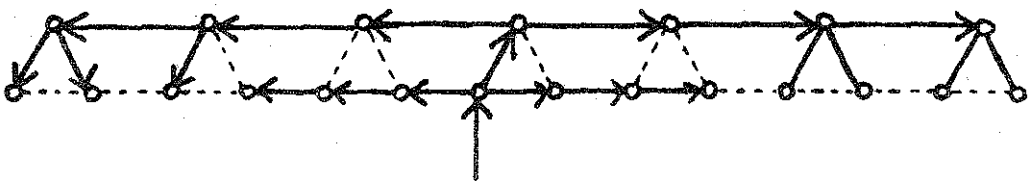


Figure 3.7

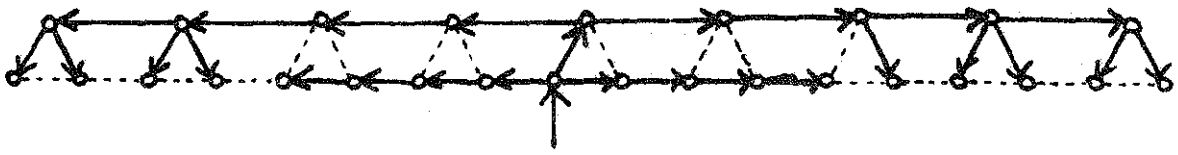


Figure 3.8

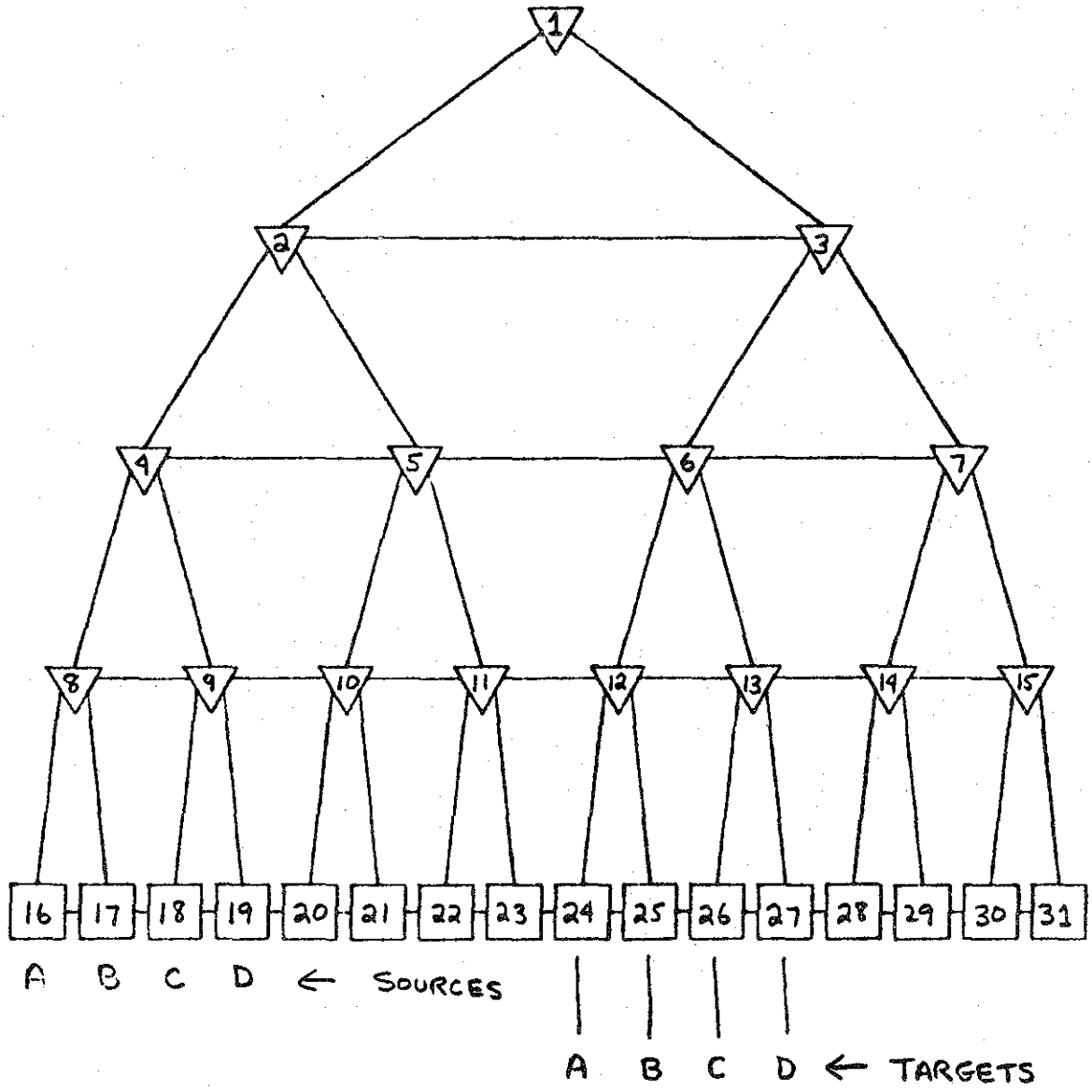

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
A B C D ← SOURCES
A B C D ← TARGETS

Figure 3.9

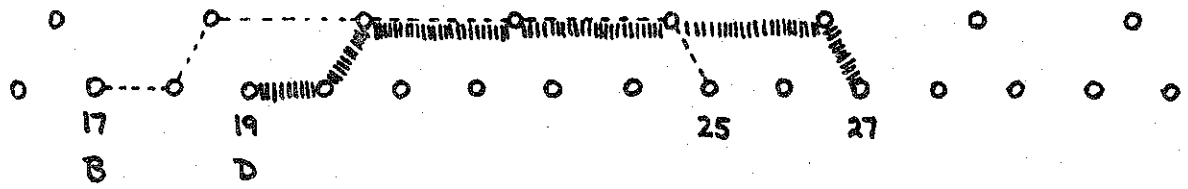


Figure 3.10

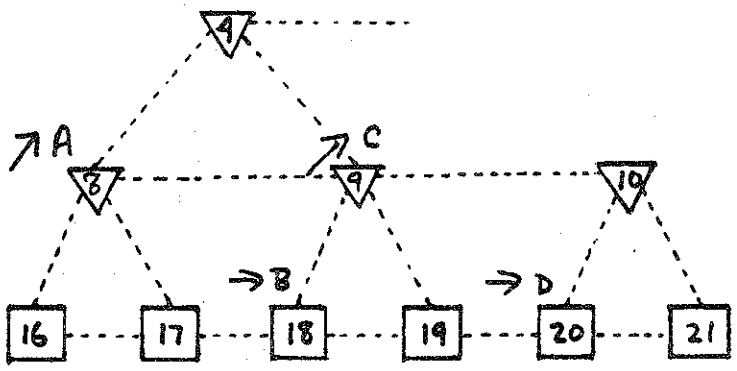


Figure 3.11

4 DATA MOVEMENT ALONG SHORTEST PATHS IN A NETWORK WITH MEMORY

4.1 Complete Path Information--The Goal

In this chapter, we examine techniques for storing information in the cells of T to guide data items to their proper destinations. The items will still follow the paths prescribed by one of the broadcast algorithms SS1-SS4. The information in the cells will be used only to decide when not to make extra copies of the items. Ideally, we would like each item to reach its destination(s) without the production of any unnecessary copies. Consider Figure 4.1. Here a data item originating in cell A is to be sent to cells B and C. The figure shows the paths which the item would take using Algorithm SS1. Of course, with Algorithm SS1, many useless copies of the data item would be created and later discarded. We see from the figure that only at cell D is it necessary to make two copies of the data item. Everywhere else, an item arriving at a cell of T is routed in only one direction, according to the pattern established by SS1.

This chapter describes several types of information which can be stored in each cell so that, upon receiving a data item, the cell can decide where to send the item. The cell must also be able to decide if the item must be copied and must know which way to send the copy (or copies). If the cells always have enough information to avoid generating useless copies, as in Figure 4.1, we will say that the cells have Complete Path Information. The mnemonic CP1 will refer to the algorithm in which all items move according to the rules of SS1 and all of the cells have complete path information. Similarly, CP2, CP3 and CP4 will refer to the algorithms in which the cells have complete path information and move according to the rules of SS2, SS3, and SS4, respectively. Figure 4.2, for example, shows how the item in the configuration of Figure 4.1 would reach its destination using CP4.

Of course, CP1-CP4 are not practical algorithms since we do not know how the machine could compute the information needed for each cell. Instead, they are the goal for this chapter since they represent the best algorithms that can be obtained from the broadcast patterns of SS1-SS4.

4.2 Range Information

The original draft of the Mag0 paper [1] suggests one approach toward the goal of the preceding section. With

each vertical edge, we associate two integers which represent the lowest and highest target values in the cells of L which can be reached vertically from that edge. The term range information will be used in what follows to refer to these two integers. Consider, for example, Figure 4.3. The edge marked A has (4,6) as its range information because the two cells with target values 4 and 6 can be reached from it. Edge B has range information (1,6) because the target values of the cells which it can reach lie between 1 and 6 inclusive.

Of course, the edges have no memory, so the range information must be stored in the cells of T. This can be done if each cell has four registers called MINLO, MAXLO, MINHI, and MAXHI. MINLO and MAXLO will store the range information associated with the edge which connects the cell to its left son, while MINHI and MAXHI will store the information associated with the edge which connects the cell to its right son. Rather than refer to MINLO and MAXLO individually, we will henceforth refer only to the pair (MINLO,MAXLO) which we give the name GLO (for Guiding information for the LO edge.) Similarly, we will use GHI to refer to the pair (MINHI,MAXHI). Figure 4.4 shows the contents of the four registers for cell C of Figure 4.3.

The range information is easy to compute in one upward pass through the tree. This computation proceeds one level

at a time as follows:

Algorithm CVR (Compute Vertical Range Information)

The operation "merge" between two range information pairs is defined as follows:

merge((a,b),(c,d)) ≡ (min(a,c),max(b,d))

I. Initialization.

For each cell of L, if that cell has a target label of k, set GLO to (k,k) and GHI to (k,k). For all other cells of L and T, set GLO and GHI to (n,0), where n is larger than any target label used in this movement pattern.

II. Transfer of information from son to father.

This step is repeated at h levels, where h is the height of the active area. GLO and GHI will refer to the registers in the cell which is sending information upwards. GLO(father) and GHI(father) will refer to the registers in the father of that cell. For each cell of L and T, if the cell is a left(lo) son, then set

GLC(father) ⇐ merge(GLO,GHI)

If the cell is a right(hi) son, then set

GHI(father) ⇐ merge(GLO,GHI)

Given this range information, Algorithms SS1-SS4 can be modified so that the data item is sent downwards only if the range information indicates that there may be a target needing that data item. That is, the item is sent along an edge only if its label falls within the range associated

with that edge. For example, in Figure 4.3, if a data item with label 5 arrives at cell C from above, then it need only be sent to the right son because the range information (3,4) guarantees that there are no targets labeled 5 among the descendants of the left son of cell C.

The algorithms which can be derived from MS1-MS4 in this manner will be referred to as MSR1-MSR4 (for Multiple Sources with Range Information). A few simulations showed that these algorithms performed somewhat (about 30%) better than MS1-MS4 on movement patterns involving 100 or so data items. However, they do not come near the performance of the CP algorithms because useless copies can still be created and passed through the network. For example, in Figure 4.3, if a data item labeled 3 arrives at edge B, it may still pass even though there are no 3's in the area below B. Furthermore, no copies are discarded until they start moving downward. Therefore, there may be many useless upward-moving, left-moving, and right-moving copies. In the next section, we will present an improvement of MSR1-MSR4 which alleviates the latter problem.

4.3 Extended Range Information

In the previous section, range information was associated only with the vertical channels which send data items downward. It is possible, however, to associate range

information with all of the channels. Consider, for example, Figure 4.5, in which target values and cell indices coincide. Suppose a data item arrives at cell A from below. Then with SS1, the item would be sent to cells #7 and #8 via the upper channel. (Copies would be sent to the rest of the cells using other channels.) If we associate the range information (7,8) with this channel, then we know that when an item arrives at A from below, a copy should be sent upward only if the label of the item is in the range 7 to 8.

The horizontal edges can be treated similarly, except for one complication. Consider Figure 4.6 which shows the same configuration as Figure 4.5. The channel to the right of cell A will send a data item to cells #5 and #6 if the item arrived at A from the left son (Figure 4.6(a)). However, if the item arrived at A from the right son (Figure 4.6(b)), then the right channel will send the item only to cell #6. Therefore, we need to store both pairs (5,6) and (6,6) and use one when the item arrives from the left son and the other when the item arrives from the right son. Alternatively, we could apply the merge operation of Algorithm CVR to the two pairs, in this case keeping only (5,6). This alternative is simpler and requires less storage in each cell. However, it may result in the production of more useless copies. For example, if an item labeled 5 arrives at cell A from the right son, then it would be unnecessarily sent to the right if (5,6) were the

only range information stored. In what follows, we will assume that two pairs are stored for horizontal channels.

Figure 4.7 shows all of the range information which must be stored for the configuration of Figure 4.5 and 4.6. As mentioned above, the horizontal connections have two sets of range information for each direction. In Figure 4.7, we have adopted the following convention. For channels which send information to the right, the range information to be used when the item arrived from the left son is written above the horizontal line. For channels which send items to the left, the range information to be used when the item arrived from the right son is written above the horizontal line. This is consistent with Figure 4.6.

Before showing how to compute the range information, let us examine how the information is used as we follow one data item through the tree. Suppose a data item labeled 6 originates at cell #2. (See Figure 4.8(a).) According the Algorithm SS1, the item would be sent upwards, to the left and to the right. However, the range information (1,1) makes it unnecessary to send it to the left and the range information (3,3) makes it unnecessary to send it to the right. Therefore, the item is sent only to the father. At this point (4.8(b)), the range information again dictates that a copy need only be sent upwards, giving 4.8(c). Now since the item is arriving from the left son, we must use

(5,8) as the range information to decide whether or not to send the item to the right. Since 6 is between 5 and 8, we do send it, giving 4.8(d). At this point, Algorithm SS1 would send copies of the item to both sons. However, since (7,8) is the range information associated with the lower right channel, a copy need only be sent to the lower left son (4.8(e)). In the last step, the range information specifies that the item need only be sent to the right son, and the item reaches its destination (4.8(f)).

As with the MSR algorithms, the range information must be stored in the cells of T. This will require fourteen registers in each cell. GLO=(MINLO,MAXLO) and GHI=(MINHI,MAXHI) serve the same purpose as in the MSR algorithms. GUP=(MINUP,MAXUP) will store the range information associated with the upper channel. GRT1=(MINRT1,MAXRT1) will have range information associated with the right channel for a data item arriving from the left son while GRT2=(MINRT2,MAXRT2) will have range information associated with the right channel for a data item arriving from the right son. Similarly, GLF1=(MINLF1,MAXLF1) will have range information associated with the left channel for an item arriving from the right son and GLP2=(MINLF2,MAXLF2) will have range information associated with the left channel for an item arriving from the left son. Figure 4.9 shows the contents of these registers for cell A of Figure 4.7.

The computation of the range information proceeds as follows:

Algorithm CER (Compute Extended Range Information)

I. Initialization

1. For each cell of L, if that cell has a target label of k, set GLO<-(k,k) and GHI<-(k,k). For all other cells cf L and T, set GLO and GHI to (n,0), where n is larger than any target label used in this movement pattern.

2. For all cells of T and L, set GUP, GLF1, GLF2, GRT1, and GRT2 to (n,0).

II. Transfer information to all neighbors.

This step is repeated 2h times, where h is the height of the active area. The neighbors of a cell sending information will be called "father", "left", "right", "loson" and "hison". The registers in these cells will be referred to as GLO(father), GRT(left), and so on. The operation "merge" from the previous section is extended to three arguments by the rule that for pairs X, Y, and Z,

merge(X,Y,Z) = merge(merge(X,Y),Z)

For each cell of T and L, send information to all neighbors as follows:

```
GRT1(left)   <- merge(GLO,GHI)
GRT2(left)   <- GHI
GLF1(right)  <- merge(GLO,GHI)
GLF2(right)  <- GLO
GUP(loson)   <- merge(GUP,GLF2,GRT1)
GUP(hison)   <- merge(GUP,GLF1,GRT2)
```

If the cell sending information is a left(lo) son, then set

GLC(father) ⇐- merge(GLO,GHI)

If the cell sending information is a right(hi) son, then set

GHI(father) ⇐- merge(GLO,GHI)

Figure 4.7 contains an example of the result of this computation of the range information.

The data movement algorithm which uses this information will be known as MSER1, for Multiple Sources with Extended Range Information. The "1" refers to the fact that this algorithm is based on the movement pattern of Algorithm SS1. It is also possible to devise MSER2, MSER3 and MSER4. These require increasing amounts of memory in each cell because the horizontal channels have to serve a larger number of different regions (groups of consecutive cells in L). To see this, consider Figure 4.10, in which information is being sent by Algorithm SS4. A data item can arrive at edge A through any of the cells marked #1-#8. For each of these, the region to which the item will be sent after crossing channel A can be found by consulting Figure 3.8 and is shown in the following table:

Arrival at A from cell #	Region reached
1	9-10
2	9-10

3	9-12
4	9-12
5	10-14
6	11-14
7	12-16
8	13-16

Since there are six distinct regions, (9-10 and 9-12 are repeated), we would need six pairs of registers to store the range information associated with each direction of movement across each horizontal channel. Though we will not present the details, the algorithm for computing all of this range information is a straightforward extension of Algorithm CER.

4.4 Hashing Information

In the example of the previous section (Figure 4.8), the data item reached its destination without being copied unnecessarily. This is because of a fortuitous (and fortunate) arrangement of the target labels and is not inherent in the algorithm. In general, unnecessary copies may be generated. Consider, for example, Figure 4.11. An item labeled 2 which originates at cell A should only be sent to the left. However, the range information is such that an extra copy will be sent upwards and to the right, reaching cell B. Therefore, Algorithms MSER1-MSER4 will not, in general, perform as well as CP1-CP4, and it is still useful to look for other types of guiding information to store in the cells.

One alternative is to replace the two integers of range information associated with a channel with a bit vector which somehow represents all of the target labels which can be reached through that channel (and perhaps other labels as well). For example, we might use a hashing function which maps target labels into bit positions. The vector 0100100000, then, would be associated with a channel which only reaches cells whose target labels hash to 2 or 5. When a data item arrives at this channel, it will be sent across the channel only if its integer label hashes to 2 or 5. Of course, it is still possible that the item will be sent across the channel unnecessarily. The algorithms which use this hashing approach will be called MSH1-MSH4.

Computing the bit vectors requires the same general technique as that used to compute the extended range information. In CER, each register pair GUP, GLO, GHI, GLF1, GLF2, GRT1, GRT2, corresponds to a group of adjacent cells in L. Each register pair contains a condensed (and incomplete) representation of the data item labels which may appear as targets in this group of cells. The merge operation combines the information about two adjacent groups of cells. Using the hashing information, GUP, GLO, GHI, GLF1, GLF2, GRT1, GRT2 will be bit vectors which correspond to the same groups of L-cells, but in a different way. The merge operation must be modified accordingly. The algorithm to compute the bit vectors for MSH1 proceeds as follows:

Algorithm CHV (Compute Hashing Vector)

I. Initialization.

For each cell of L which has a target label k, compute i=h(k), where h is the hashing function chosen. Set the ith bit of GLO and GHI to 1 and all other bits to 0. For the rest of the cells of L and T, set GLO and GHI to all 0's. For all cells of L and T, set GUP, GLF1, GLF2, GRT1, and GRT2 to 0's.

II. Transfer of information between cells.

This section is the same as part II of Algorithm CRE using

merge(X,Y) = X OR Y

The introduction of the hashing function makes it extremely difficult to analyze the performance of MSH1-MSH4. However, it should be clear that if the hashing function is chosen well enough, the performance of MSH1-MSH4 can approach that of CP1-CP4. As an extreme, if the length of the bit vector is equal to the number of distinct targets, then using the identity function for h will provide complete path information. Even with fewer bits, the performance of MSH1-MSH4 can be quite close to CP1-CP4, especially if hashing information is stored in addition to range information, forming hybrid algorithms MSERH1-MSERH4. Therefore, in Chapter 5, we will concentrate analysis efforts on the conceptually simpler CP algorithms.

4.5 Consecutively Numbered Target Labels.

In the previous sections, we made no assumptions about the distribution of the target labels and the location of the source items. An item could originate anywhere in the active area and be sent anywhere. In fact, data movement for many of the operations of the machine of [1] is more orderly. In most cases, data movement involves translations of no more than a fixed number of contiguous regions in L. Each region, in which the non-empty cells are labeled consecutively, may need to be copied an arbitrary number of times. Figure 4.12, for example, shows a 10-cell region which must be copied to two places. The exact percentage of movement patterns which are of this form depends on the particular reduction language dialect of the machine (that is, the choice of primitive operations) and the characteristics of user programs. However, one would expect the case of translation of contiguous regions to be quite common since it is used to implement some of the most basic features of the language. For now, we will consider only one source region, as in Figure 4.12, and assume that data items move according to the rules of SS1.

The region (group of consecutive cells in L) reached by any horizontal or downward-moving vertical channel is always a set of contiguous cells of L. Therefore, the target labels in the area reached by a horizontal or downward-

moving channel is always a group of consecutive integers, as with edge A of Figure 4.13, or two groups of integers, one ending with n (the number of labels in the segment) and the other beginning with 1, as with edge B of Figure 4.13. (It is assumed that the item arrived from the left of cell X.) In the first case, the range information ((3,6) in the example) is sufficient to specify the complete contents of the area. In the other case, we still need only two integers to represent the contents of the area. In the example, we could use (9,2) to mean 1...2 and 9...10. In general, (m,k), where k<m can represent the groups 1..k and m...n.

A channel which sends items upward can reach one group of contiguous cells (edge C in Figure 4.13) or two groups of cells (edge D in Figure 4.13). If the channel reaches one group of cells, then, as with the horizontal and downward-moving vertical channels, only two integers are needed to specify the contents of the area. If the channel reaches two groups, then one group must be at the very left of the active area and the other group must be at the very right. The region on the left must contain labels 1,2,...k and the region on the right must contain labels m,m+1,...n-1,n. If k<m, then the pair (m,k) is sufficient to represent the contents of the entire area reached by the channel. If k≥m, then (1,n) represents the contents of this area. Therefore, it is always possible to use only two integers to represent

all of the target labels in the area reached by an edge. That is, under the assumption of consecutively numbered target cells, it is possible to store complete path information using two integers per vertical channel and four per horizontal channel. (This is the same amount of information required by MSER1.)

The pairs of integers can be computed using an algorithm which is almost the same as CER.

Algorithm CRCT (Compute Range Information for Consecutively Labeled Targets)

I. Initialization.

For each cell of L which has a target label k, set GLO←-(k,k) and GHI←-(k,k). Set all other registers to (0,0).

II. Transfer of information between cells.

This is the same as part II of CER except for the definition of the "merge" function. We define merge((a,b),(c,d))≡(x,y), where x and y are computed by the PL/I-like algorithm below. This algorithm makes use of the fact that we merge information about adjacent regions of L only. In the algorithm, N is the largest target label in the segment. (In Figure 4.12, N=10.) Figure 4.14 shows the different cases which must be handled by the algorithm. In the figure, [A...B] represents a region whose first target label is A and whose last target label is B. For clarity, the

two regions to be merged are drawn on two horizontal levels even though all of the regions are in L.

```
IF A=0 & B=0
   THEN DO; X=C;
            Y=D; END;
IF C=0 & D=0
   THEN DO; X=A;
            Y=B; END;
IF A≤B & C≤D
   THEN DO; /* FIGURE 4.14(A) */
            X=MIN(A,C);
            Y=MAX(B,D); END;

IF A≤B & D<C
   THEN DO; IF A=D+1
               THEN DO; /* FIGURE 4.14(B) */
                        X=C;
                        Y=B; END;
               ELSE DO; /* FIGURE 4.14(C) */
                        X=A;
                        Y=D; END;
            IF X≤Y
               THEN DO; /* FIGURE 4.14(D) */
                        X=1;
                        Y=D; END;
                                 END;

IF B<A & C≤D
   THEN DO; IF C=B+1
               THEN DO; /* FIGURE 4.14(E) */
                        X=A;
                        Y=D; END;
               ELSE DO; /* FIGURE 4.14(F) */
                        X=C;
                        Y=B; END;
            IF X≤Y
               THEN DO; /* FIGURE 4.14(D) */
                        X=1;
                        Y=N; END;
                                 END;

IF B<A & D<C
  THEN DO; /* FIGURE 4.14(G) */
           X=1;
           Y=N; END;
```

The preceding discussion shows how to move the items when one source segment is involved. In general, there may be more than one. Multiple source segments could be handled sequentially in time by reusing the same set of registers for guiding information. This increases the total time needed for data movement while limiting the amount of memory needed in each cell. Alternatively, each cell could have multiple sets of registers so that multiple segments could be handled simultaneously. This would require each data item to carry a segment number with it so that it would know which set of registers to use as guiding information. Since some operations may involve an unbounded number of segments, it will not always be possible to handle all of the segments simultaneously.

For the case of consecutively numbered target labels, we have achieved the goal stated in the introduction of this chapter. Namely, we now have a way of storing complete path information in the cells. That is, we now have a way of moving items in such a way that no useless copies of data items are produced. Therefore, it is now necessary to discuss the performance of the CP (Complete Path Information) algorithms.

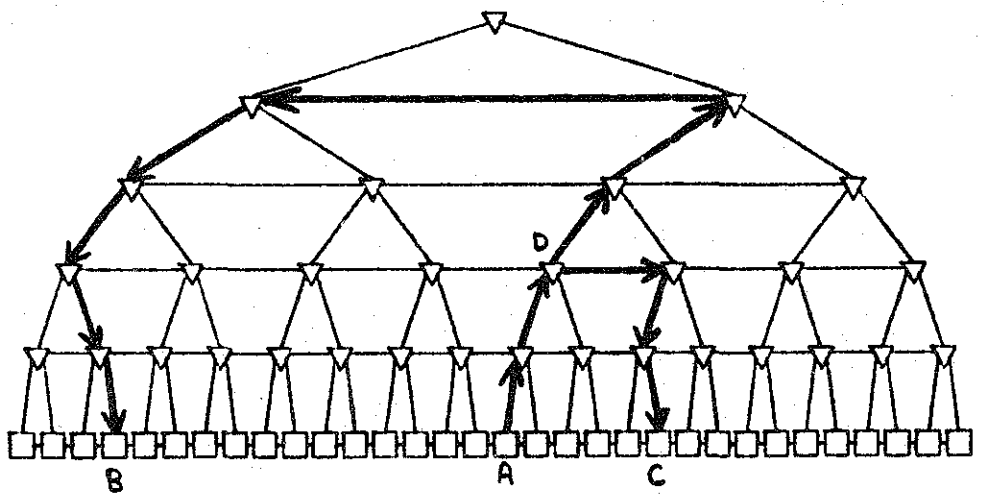


Figure 4.1

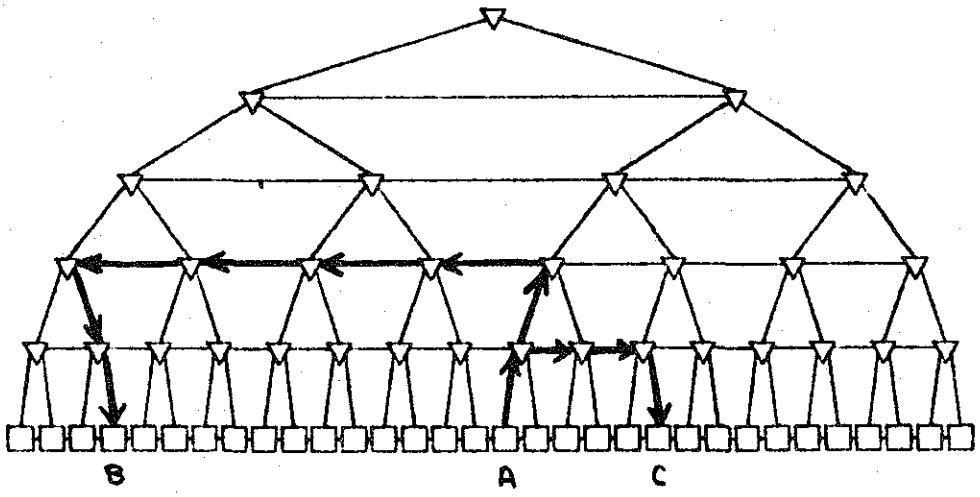


Figure 4.2

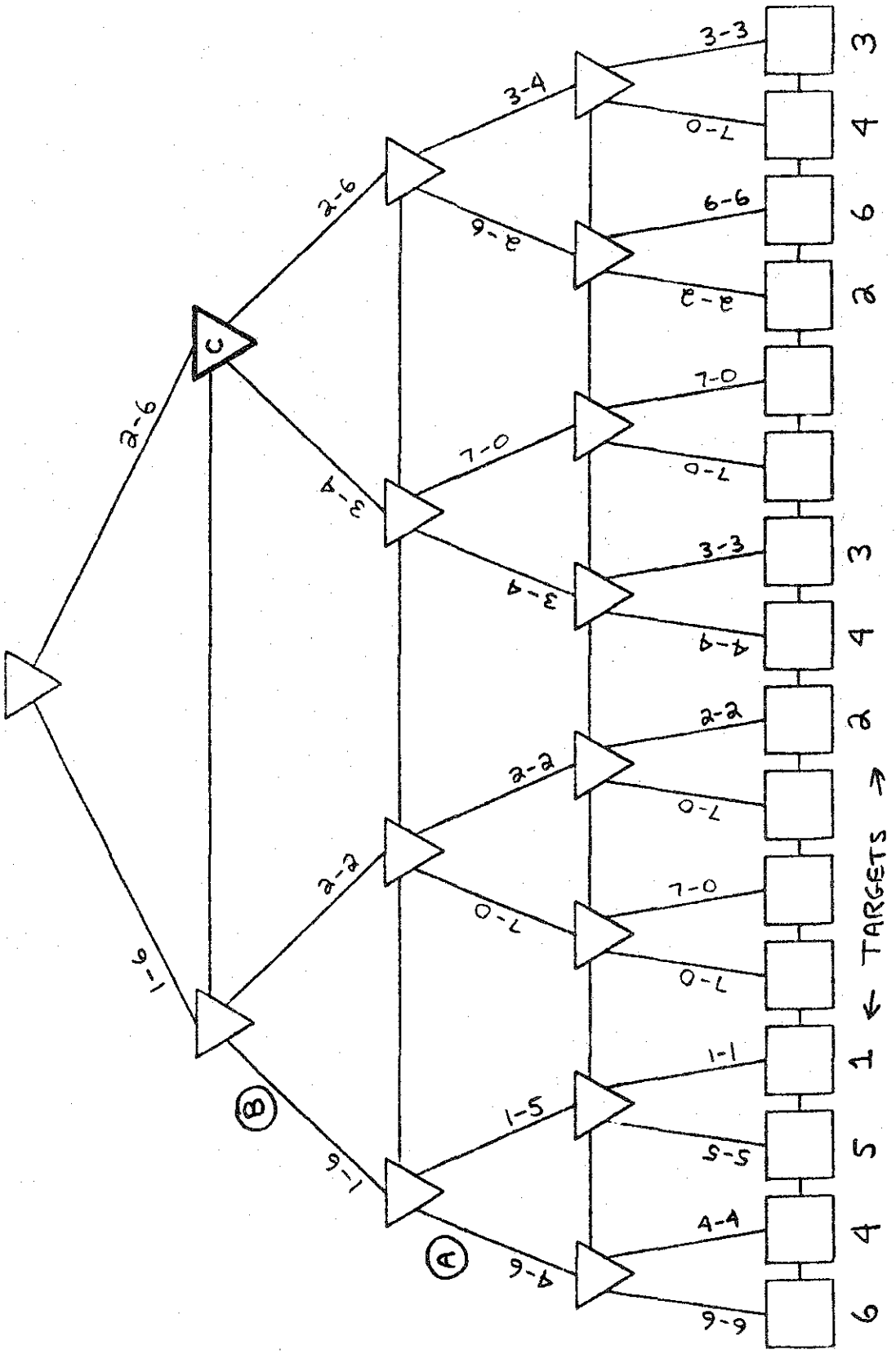


Figure 4.3

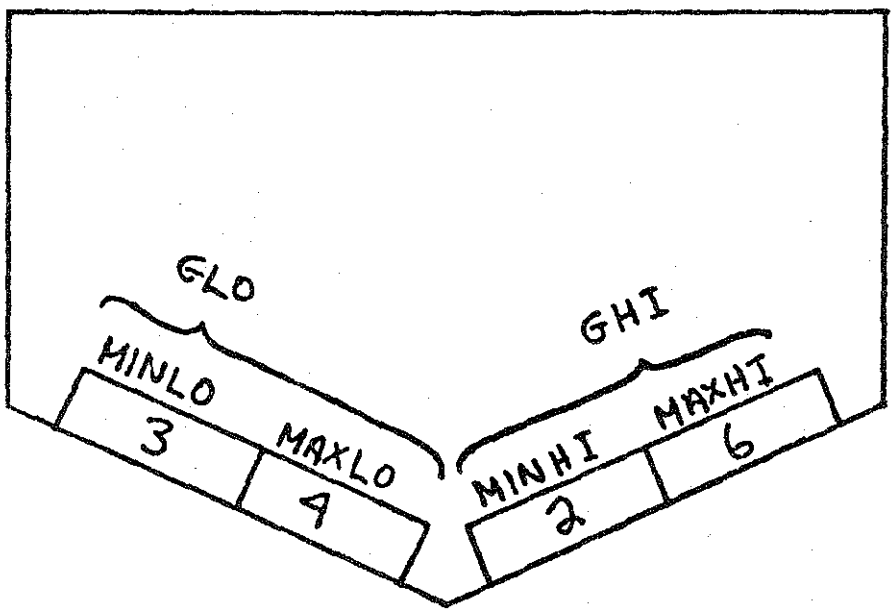


Figure 4.4

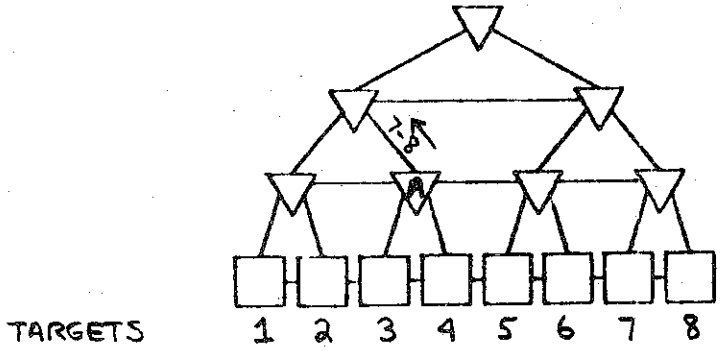
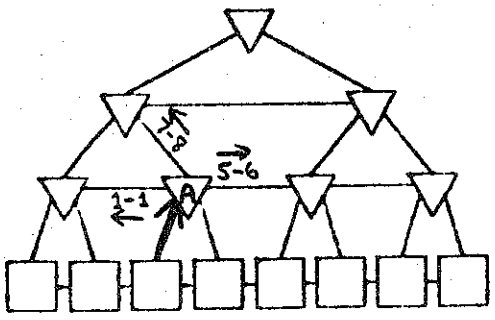
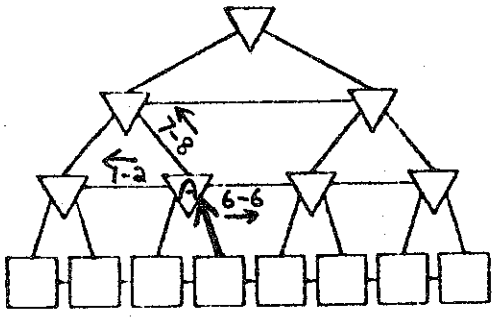


Figure 4.5



(a)



(b)

Figure 4.6

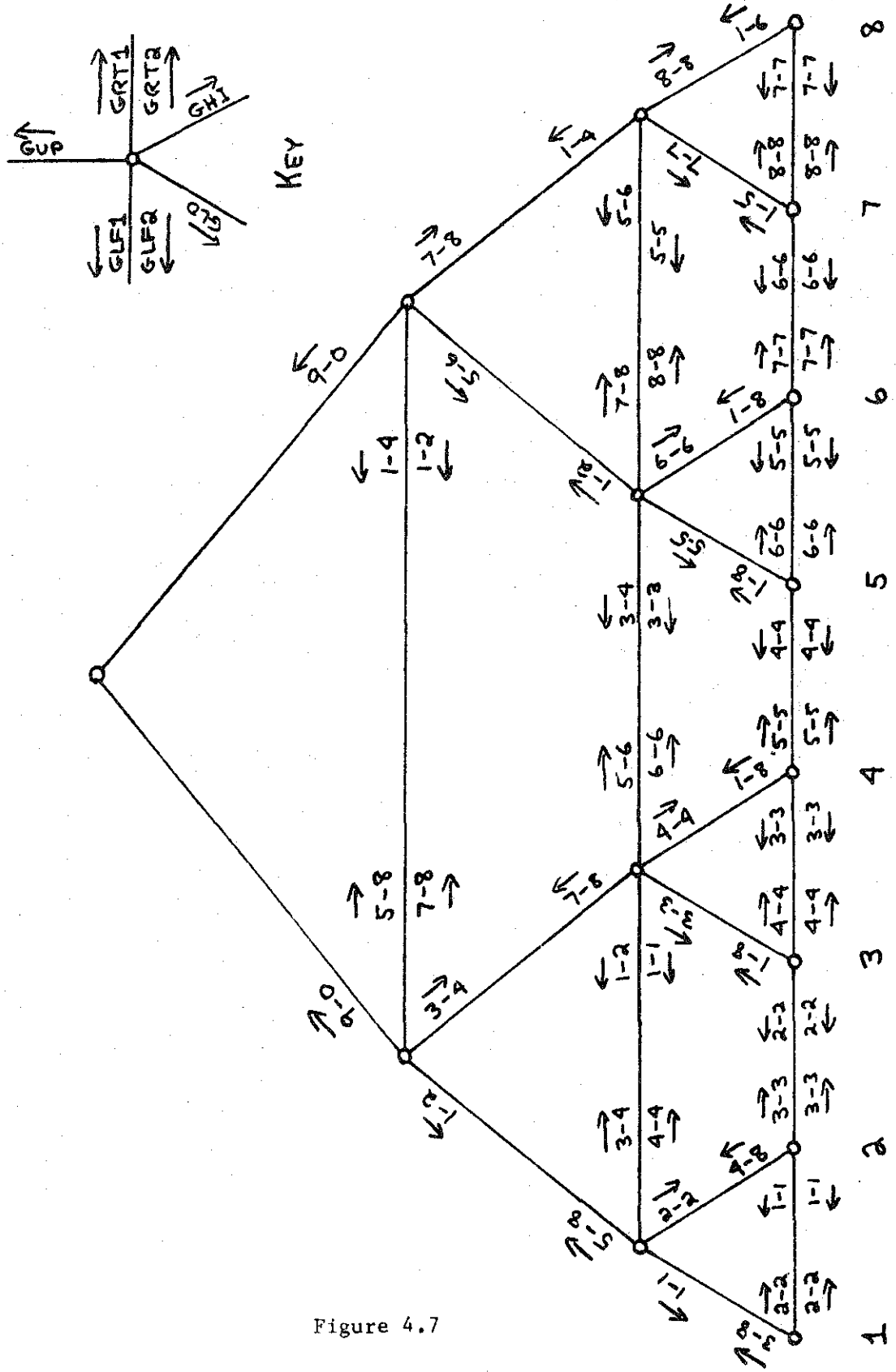


Figure 4.7

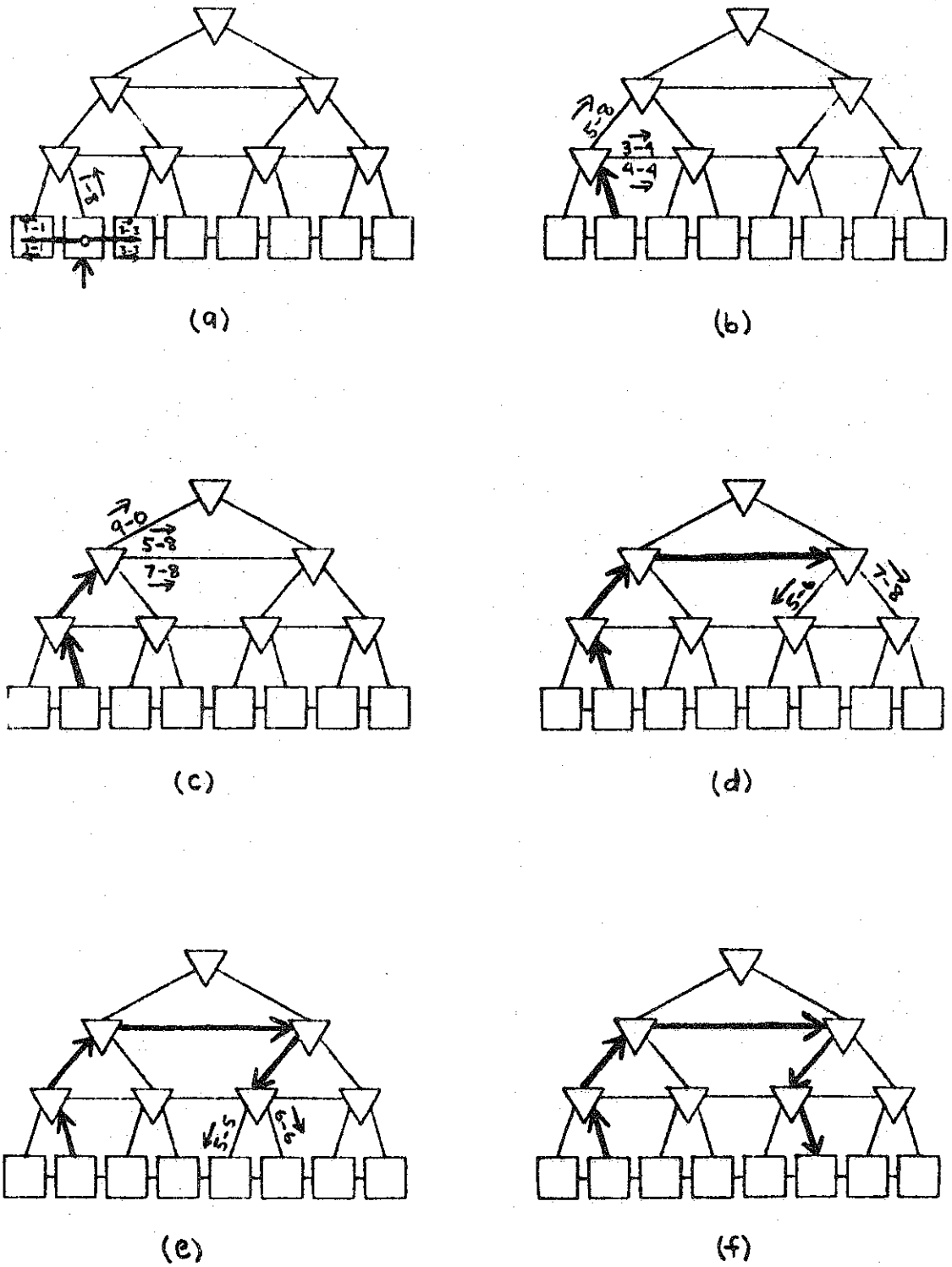


Figure 4.8

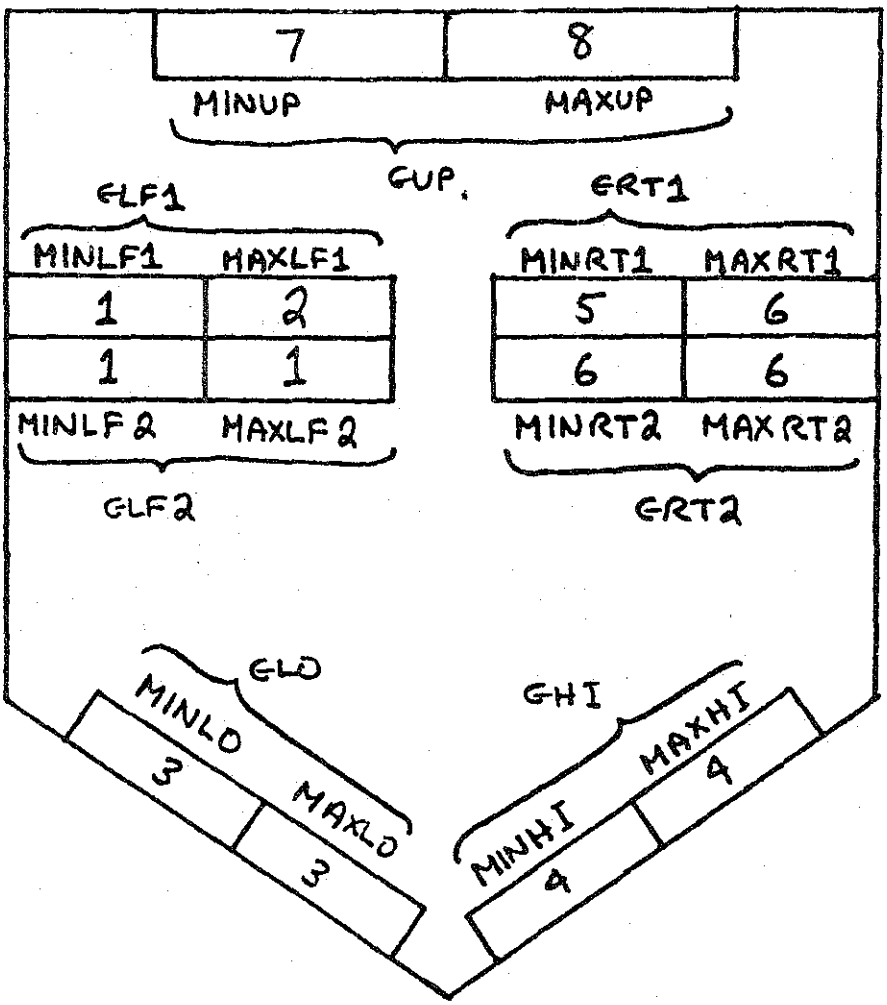

7
8
MINUP
MAXUP
GUP
GLF1
MINLF1
MAXLF1
1
2
1
1
MINLF2
MAXLF2
GLF2
GRT1
MINRT1
MAXRT1
5
6
6
6
MINRT2
MAXRT2
GRT2
GLO
MINLO
MAXLO
3
3
GHI
MINHI
MAXHI
4
4

Figure 4.9

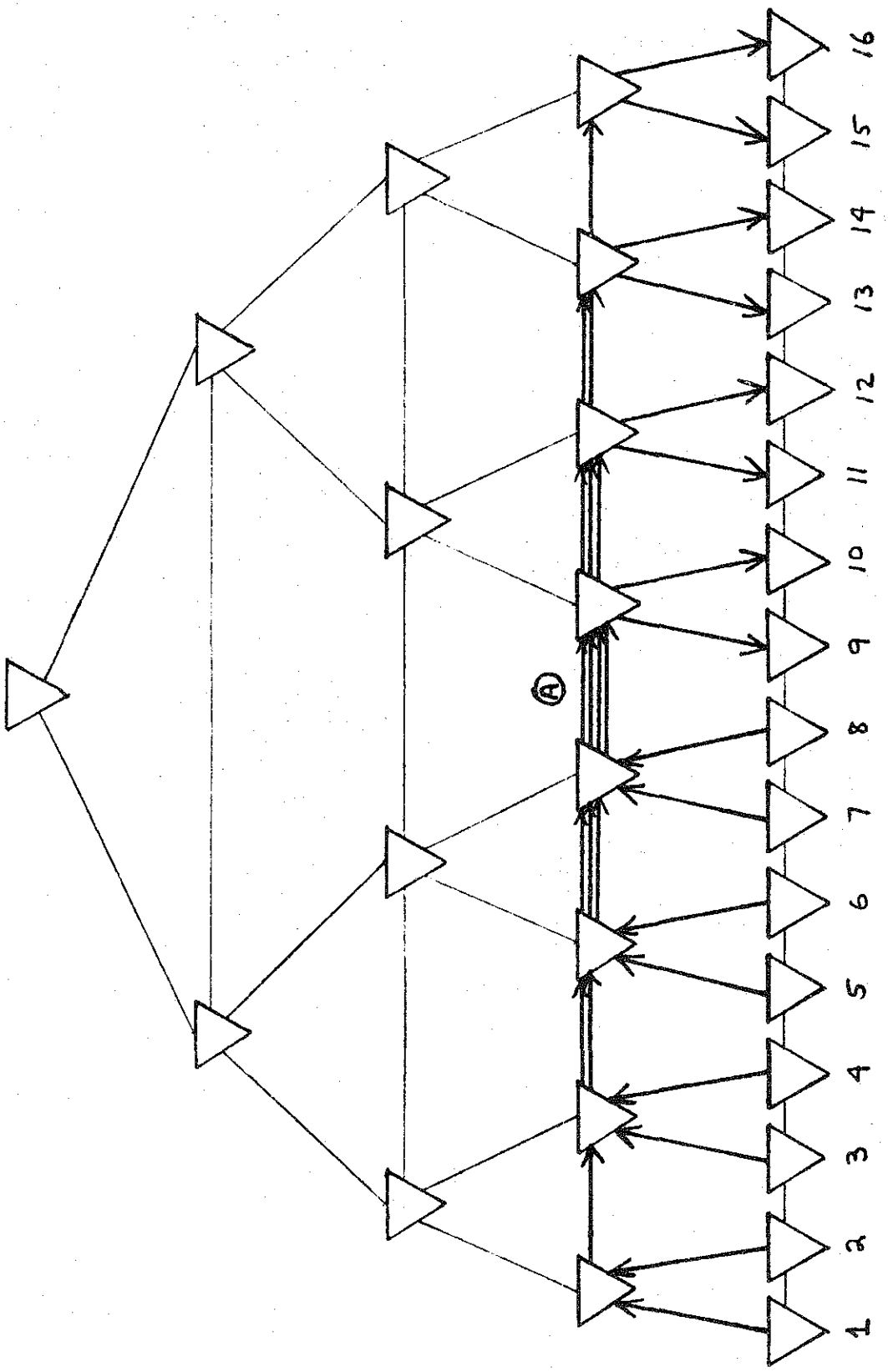


Figure 4.10

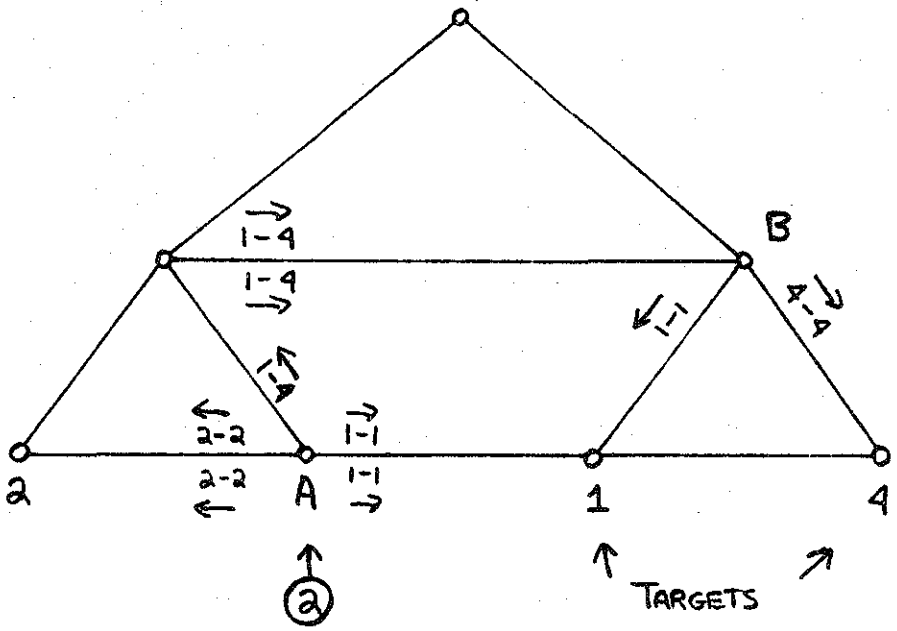


Figure 4.11

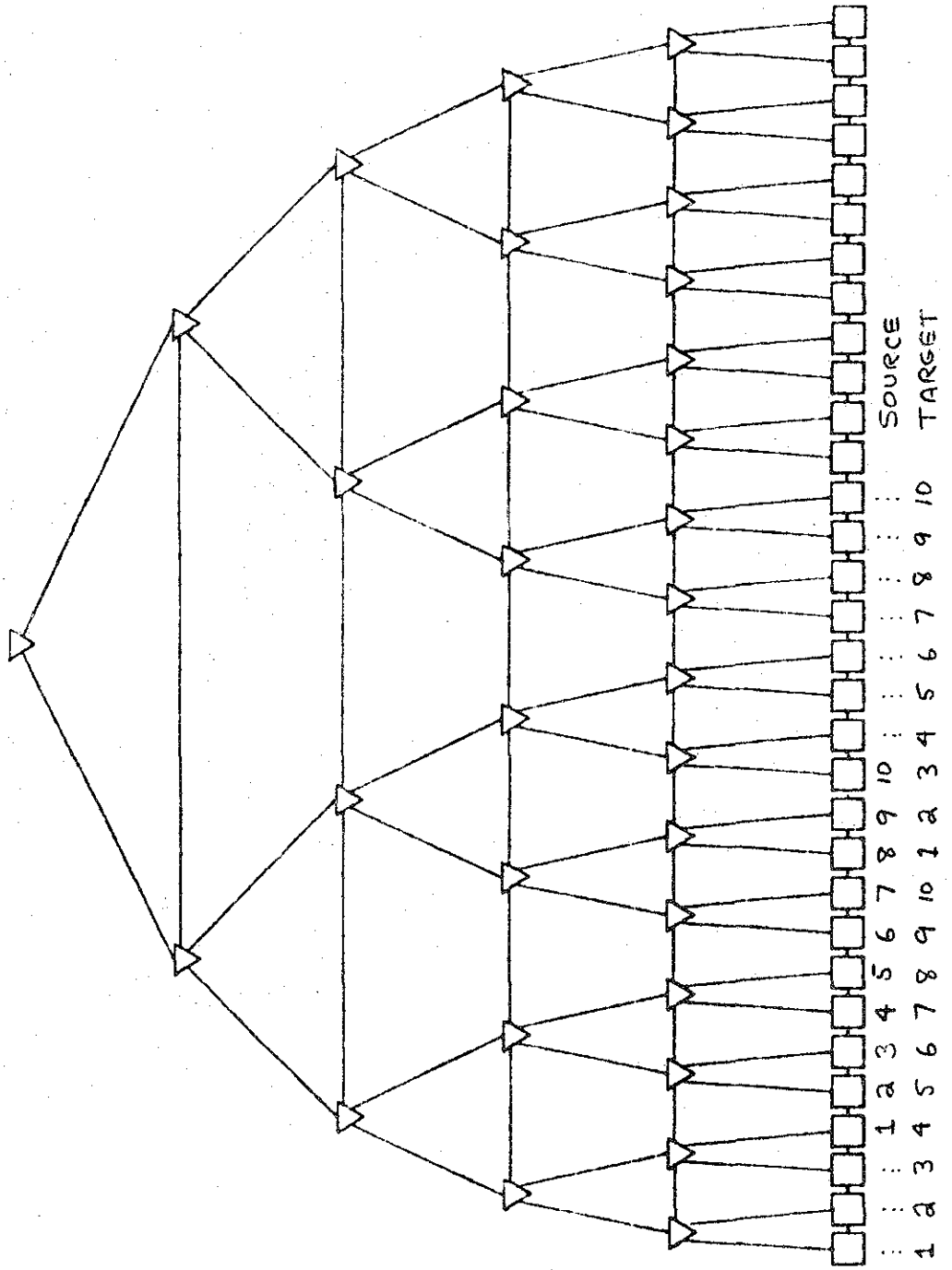

SOURCE
TARGET
1 2 3 4 5 6 7 8 9 10 ... 1 2 3 4 5 6 7 8 9 10 ... 1 2 3 4 5 6 7 8 9 10 ... 1 2 3 4 5 6 7 8 9 10

Figure 4.12

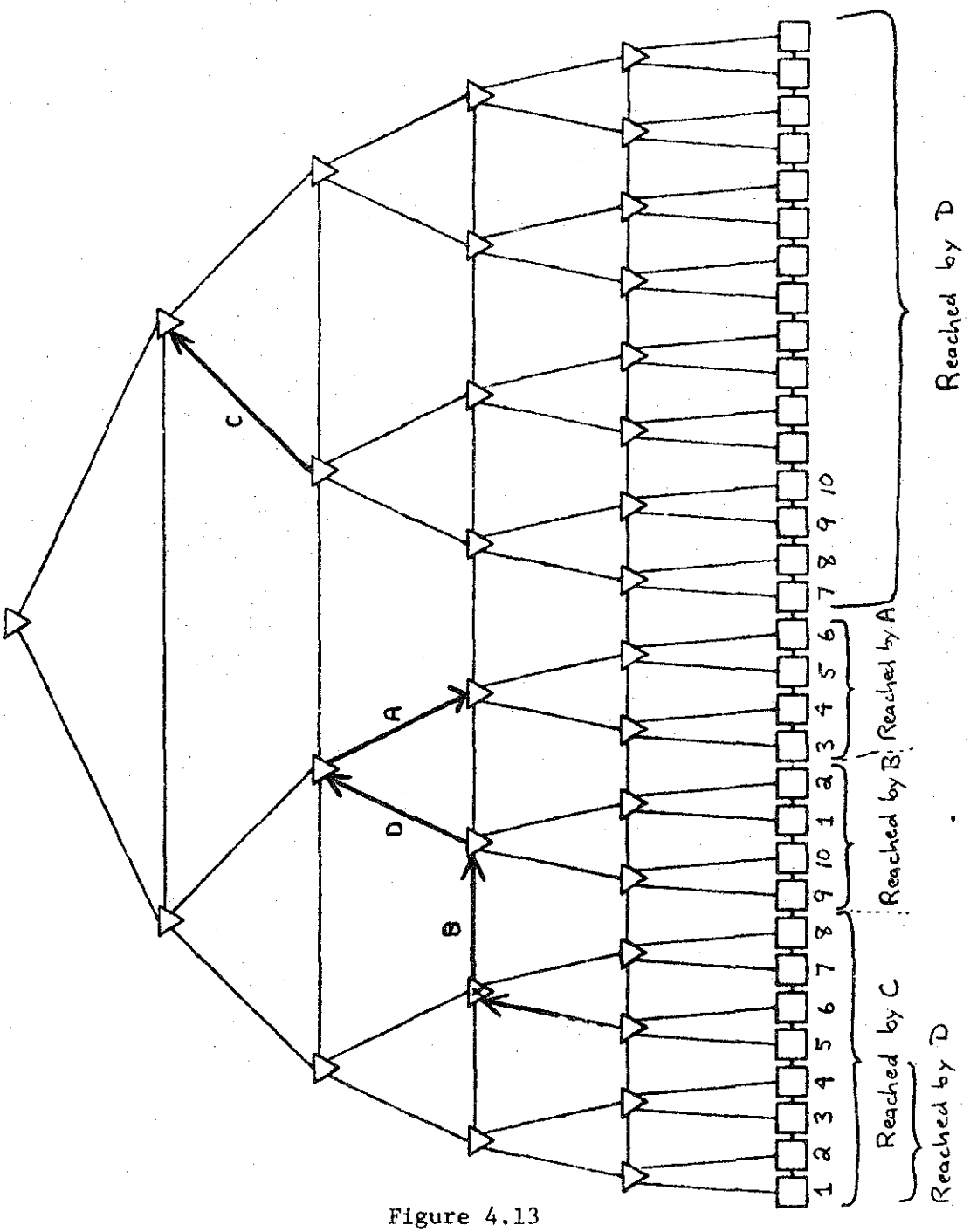
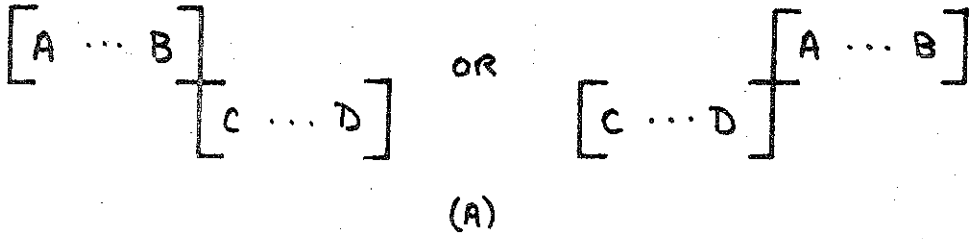
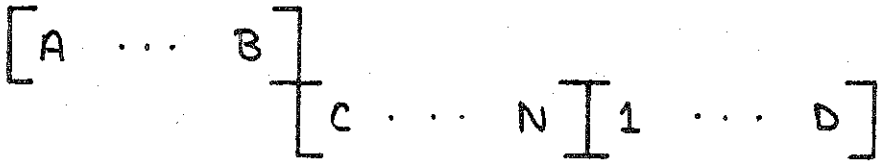

C
A
D
B
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
Reached by C
Reached by D
Reached by B
Reached by A
Reached by D

Figure 4.13

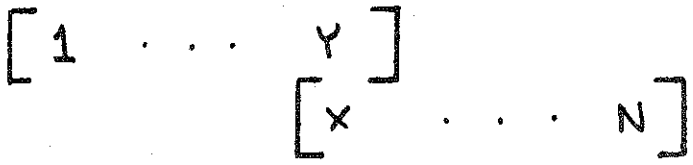


(A)

(B)

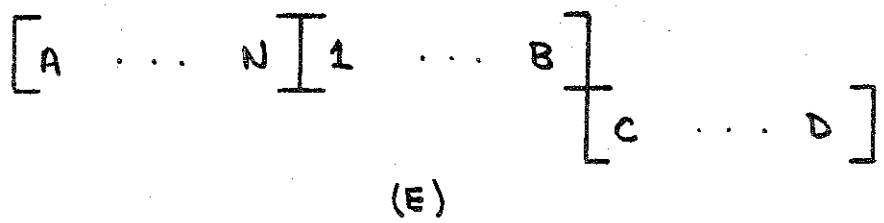


(C)

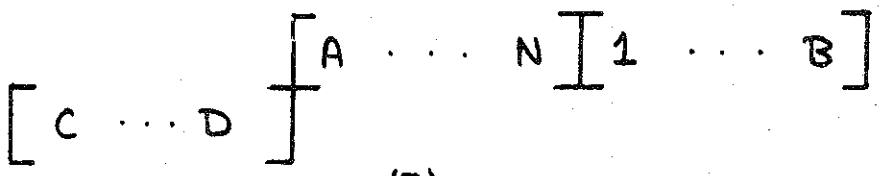


(D)

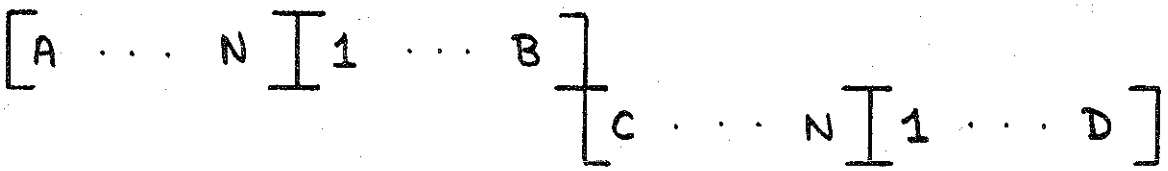
Figure 4.14 (Part 1)

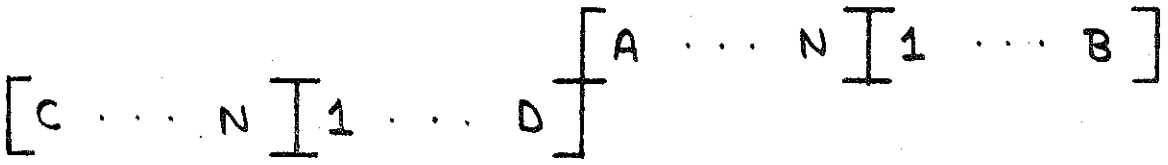


(E)



(F)





(G)

Figure 4.14 (Part 2)

5. PERFORMANCE OF THE CP ALGORITHMS FOR SPECIFIC MOVEMENT PATTERNS

It is difficult to make general statements about the performance of the CP algorithms. Therefore, we will investigate their performance for specific movement patterns.

Certainly, there are some patterns for which CP4, which routes items along shortest paths, is optimal. Consider a right rotation by a single cell (assuming no empty cells) as in Figure 5.1. Notice that the shortest path between the cell containing H (source label #13) and its target has seven steps. The shortest paths between all of the other data items and their respective targets have length one. Therefore, data movement for this pattern must require at least seven steps. But CP4 will require exactly seven steps since it sends each item along a shortest path and none of the paths overlap. Therefore, CP4 is optimal in this example.

In general, CP4 will be optimal whenever the paths between corresponding source and target cells do not

overlap. However, in most interesting patterns, the paths do overlap. In this chapter, we will examine the performance of the CP algorithms on some of these patterns. Chapter 6 will give a negative answer to the question of whether or not the CP algorithms are optimal for these patterns.

5.1 Reversal

Consider the problem of reversing the contents of a group of adjacent cells in L. For simplicity, assume that n, the number of cells, is a power of two and that all of the cells are descendants of a single cell at level log(n). (See Figure 5.2.) Using CP1 or CP2, all of the data items in region X must move along the path C-A-B-F to region Y. Similarly, all of the items in region Y will use path F-B-A-C to reach region X. Since all channels are two-directional, data movement along C-A-B-F and F-B-A-C can take place simultaneously. Now region X and region Y both contain n/4 items. Therefore, the number of steps required to interchange the items in regions X and Y is

$$(n/4) + 2*(\log(n)-1)$$

(The first items reach A and B in (log(n)-1) vertical steps. Then the two sets of n/4 items cross from A to B or from B to A. Finally, it takes (log(n)-1) steps for the last items to descend to L.) The other items (those not in X and Y) can be reversed in fewer steps because they do not use cells

A, B, C, and F and will not interfere with the items from regions X and Y. Therefore, using CP1 or CP2, the total time required for data movement is

$$(n/4) + 2*(\log(n)-1)$$

steps.

Using CP4, each item in region X will be sent to Y along the path C-D-E-F and items in region Y will be sent to X along path F-E-D-C. Again, assuming n>4, the other items will not interfere, so the total time using CP4 will be

$$(n/4) + 2*(\log(n)-2) + 2$$

(The extra +2 accounts for the fact that there are now three horizontal steps instead of one.)

When the items are not all descendants of a single cell, but the number of cells is a power of two, the number of steps required for data movement with CP1 is still linear. Consider Figure 5.3. Cells A and D are separated by a distance of $n-1 \geq d$ while B and C are separated by a distance of $(n/2)+1 \geq d$. But since n is a power of two, $CEIL(\log(d)) = CEIL(\log(c)) \geq m$. Thus, by Theorem 10 (Chapter 2), all of the items between A and B (inclusive) must rise to level m or m-1 before descending to their destinations between cells C and D. But this includes n/2 of the items, with n/4 moving in each direction. Therefore, at least n/8 items must use one edge on level m or level m-1. This

implies that n/8 is a lower bound on the time required by CP1 for reversal.

By a similar argument using Theorems 6 and 8, the time required for reversal of adjacent items with CP2 and CP4 can be shown to be linear in n, the number of items moving. The coefficient on n is less than 1/2, since only half of the items move in each direction, with the exact coefficient depending on the specific configuration.

5.2 Translation

Translation involves moving a group of items to another region of the tree, preserving the original order of the items. An example in which neither sources nor targets have gaps is shown in Figure 5.4. Assuming that the source cells and the target cells are disjoint, as in Figure 5.4, then the time used by the CP algorithms to accomplish translation will be linear in the number of items moving. This is because all of the items are moving the same distance and must therefore rise to the same level, or perhaps the same two levels, and use the same horizontal cross-connections. This follows from Theorems 6, 8, and 10 of Chapter 2.

Consider again Figure 5.4. Using CP1, every item must pass through edge X. Using CP2, every item must pass through edge Y, and using CP4, every item must pass through edge Z. In general, the items may be divided among two

horizontal levels as in Figure 5.5. In this example, eight items are being shifted ten positions each. Using CP1, for example, three of the items (A, B, and C) would use edge W, two items (D and E) would use edge V, and the other three items would use edge X. Using CP2, three items (A, D, and E) would use edge W, four items (B, C, F, and G) would use edge Y, and H would use edge X. Finally, using CP4, all eight items would use edge Y.

In general, using CP4 for translations in which the source cells and the target cells do not overlap, all of the items which use the same level to move horizontally must use at least one edge in common (because the path followed by any item must have at least two horizontal edges on this level). Since no more than two levels are used, at least n/2 items must use each level. Therefore, the time required for translation with CP4 is at least n/2 plus the O(log(n)) steps required for the items to rise to the correct level and then descend.

With CP1 and CP2, there may be a choice between two edges on one level (as with edges W and X in the example of Figure 5.5). Therefore, all of the items must pass through one of three horizontal edges (W, X, or Y in the example) so CP1 and CP2 require at least n/3 + O(log(n)) steps for translation.

If the target cells and source cells do overlap, then the time required for translation depends on the distance by which the items are being moved. As an extreme case, if each item is to be shifted by one position, then any of the CP algorithms will use only one step for data movement. At the other extreme, if each of the n items is to be shifted by n-1 positions, then, as shown above, data movement takes $\Theta(n)$ steps. In between, the number of steps required appears to be linear in the shift distance. If the shift amount is m, as in Figure 5.6 where m=4, then the m items on the end must be translated by m positions, requiring $O(m)$ time. Therefore, $\Theta(m)$ is a lower bound on the time required. My conjecture is that $\Theta(m)$ is an upper bound also.

If there are gaps between source or target items, then the preceding analysis does not hold since all of the items no longer move the same distance. In this case, the time for translation depends on how the occupied cells are distributed among the empty cells.

5.3 Perfect Shuffle

The pattern of perfect shuffle, or 2-by-n transpose, is shown in Figure 5.7. In this pattern, the items are divided into two groups. Those in the left group shift to the right, each item shifting a different distance, so that when

data movement is completed, the items occupy every other cell (Figure 5.7a). Similarly, the items in the group on the right shift to the left (Figure 5.7b). The effect is to interleave the two groups as if each were half of a deck of cards being perfectly shuffled. Suppose n, the number of items to be shuffled, is a power of two and suppose these items occupy adjacent cells which are all descendants of a single cell at level log(n). Then let us examine the time required for CP1 to complete the shuffle. First, consider the right-moving items (Figure 5.8). In this figure, all of the items in region X must move to region Z. Using CP1, all of these items (and there will be n/8 of them) will cross edge A-B. Similarly, all n/8 items in region W will cross edge C-D to reach region Y. The paths for items in these two regions do not interfere with each other. Paths for items in regions W and X do cross at cell C, but we assume that paths from different regions use different registers of C. Therefore, the time required to move all of the items from regions W and X is

$$n/8 + 2*(\log(n)-1)$$

The $2*(\log(n)-1$ term is the number of vertical steps used by items in region X. The other right-moving items (those in region T) can be moved in fewer steps since these items do not move as far and their paths do not interfere with the path for items in regions W and X. Therefore, the right-moving items can be moved in

$$n/8 + 2*(\log(n)-1)$$

steps. By symmetry, the left-moving items require the same amount of time. However, left and right movement can take place simultaneously even though there is some overlap of paths. Consider, for example, Figure 5.9. This shows the path of eight of the items in a shuffle of n=32 items. Notice that one vertical edge carries both left-moving items (G and H) and right-moving items (C and D) so the paths overlap. However, no vertical edge except perhaps W, X, Y, or Z can possibly carry more than n/8=4 items because for any edge other than W, X, Y, or Z, only n/8 or fewer cells of L can be reached vertically via that edge. But edges W and Z carry only n/8=4 items while X and Y carry none. Therefore, no edge -- horizontal or vertical -- carries more than n/8 items. So the longest "waiting line" for any edge is of length n/8 and the total time for data movement is

$$n/8 + 2*(\log(n)-1)$$

Similar analysis yields

$$t = (3/16)*n + 2*(\log(n)-2) + 1$$

for CP2 and

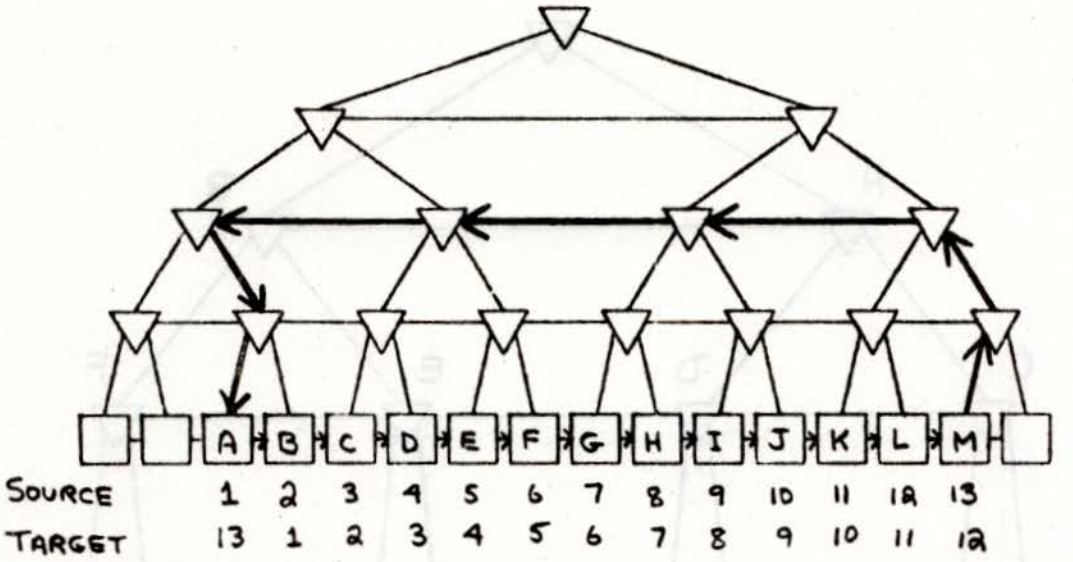
$$t = (7/32)*n + 2*(\log(n)-3) + 3$$

for CP4. Notice that the coefficient on n increases when the individual items travel along shortest paths. This is because with CP4, the paths have more horizontal edges than with CP1 and it is more likely that two paths will have at least one horizontal edge in common. In particular, there

is one edge which must carry (7/32)*n different items.

5.4 Conclusions

The VERTICAL algorithm requires the same amount of time to move n data items, regardless of the movement pattern. By contrast, the performance of the CP algorithms varies greatly with the movement pattern. However, for most of the interesting patterns, the CP algorithms require time proportional to n, as does the VERTICAL algorithm. In some cases, the CP algorithms have a linear coefficient which is less than one, which indicates a linear improvement over the VERTICAL algorithm. However, there is no order of magnitude improvement. That is, the time required is still $\Theta(n)$. In the next chapter, we will study whether it is possible to use a different algorithm to obtain an order-of-magnitude time improvement for these and other patterns.


A B C D E F G H I J K L M
SOURCE 1 2 3 4 5 6 7 8 9 10 11 12 13
TARGET 13 1 2 3 4 5 6 7 8 9 10 11 12
BEFORE ROTATION

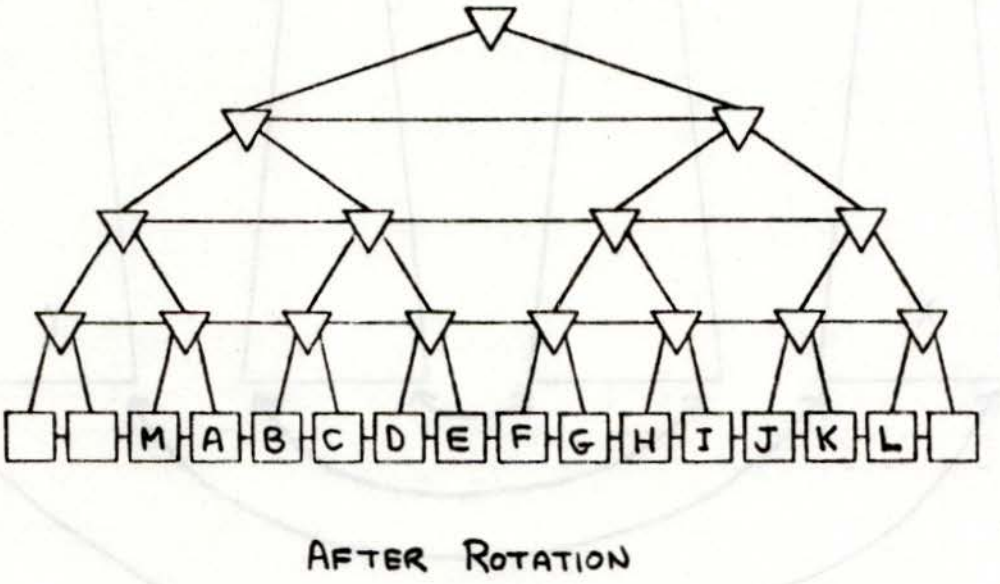

M A B C D E F G H I J K L
AFTER ROTATION

Figure 5.1

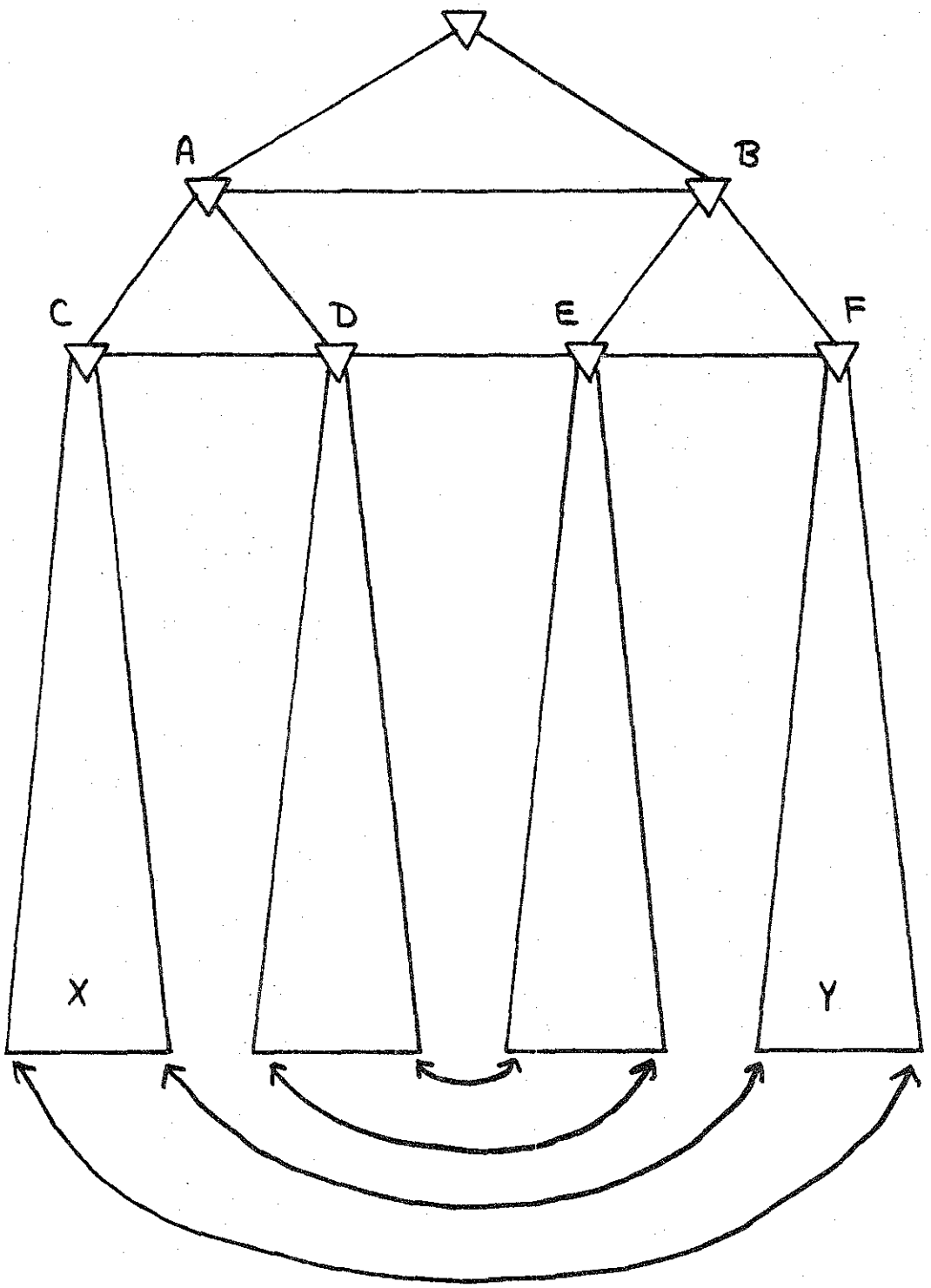


Figure 5.2

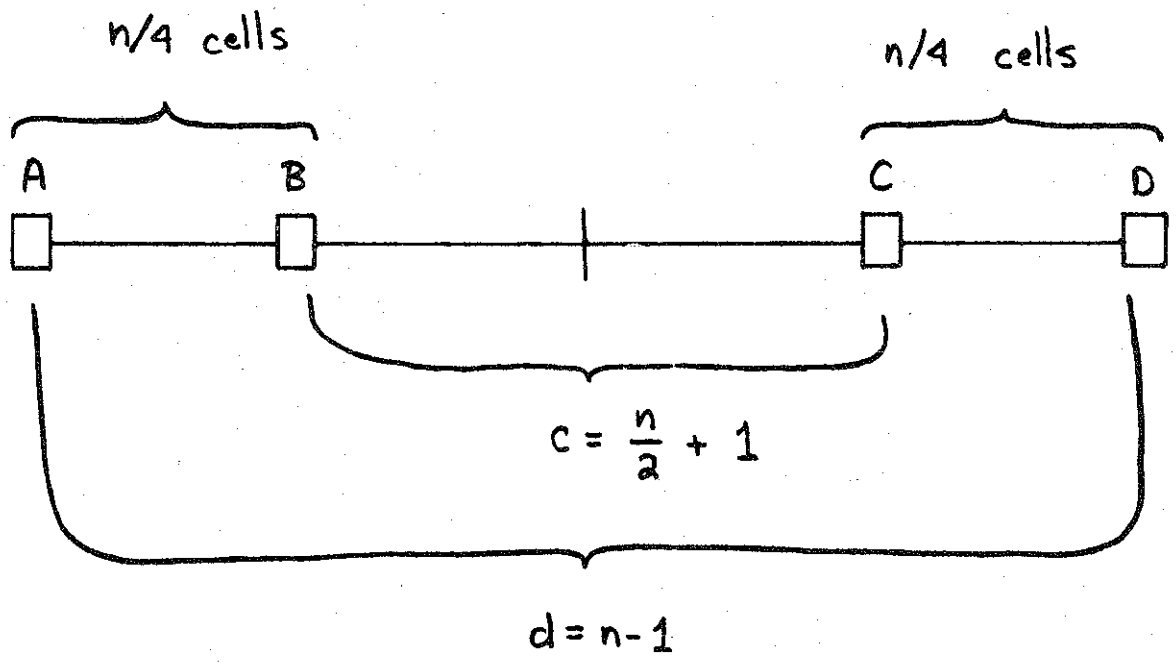


Figure 5.3

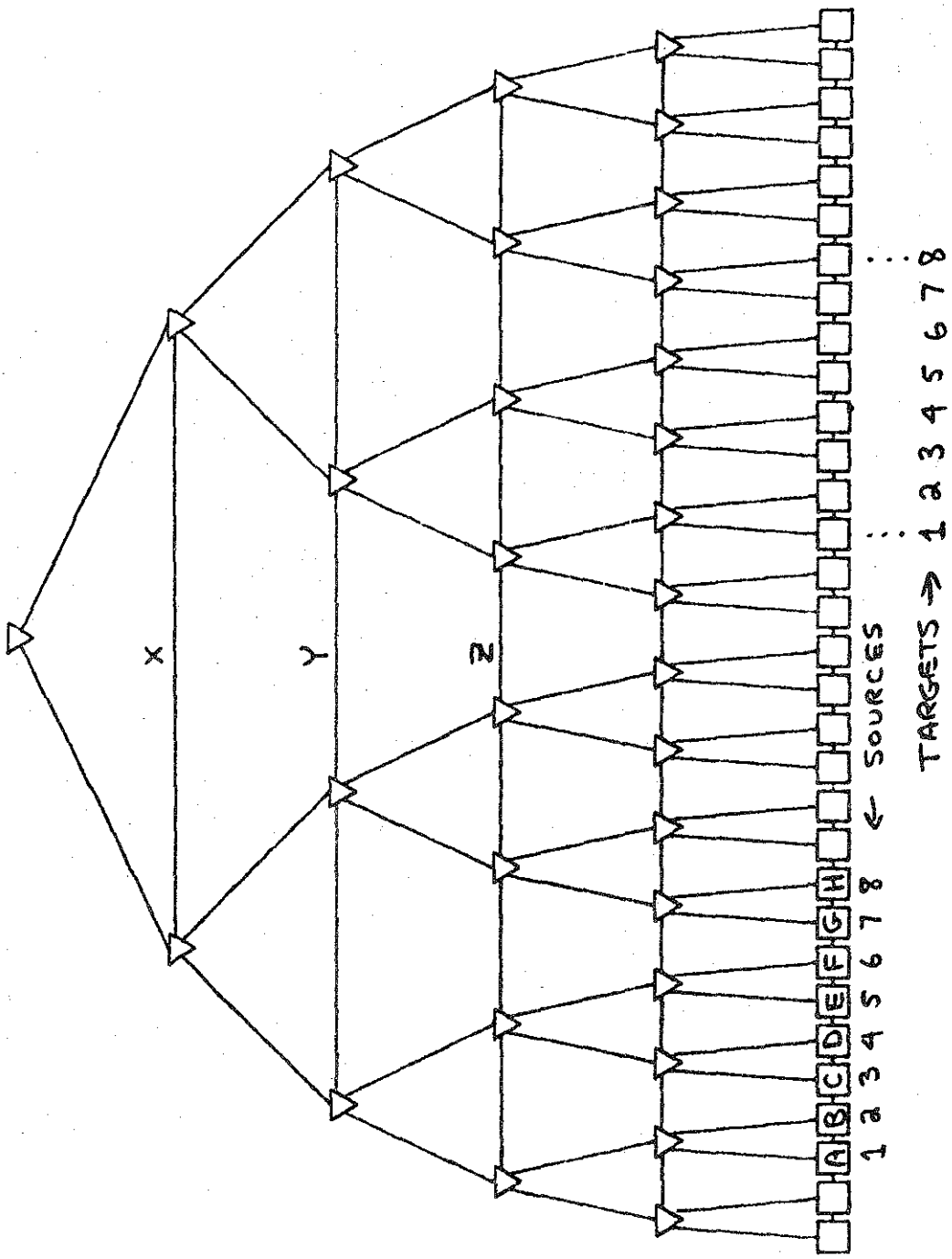


Figure 5.4

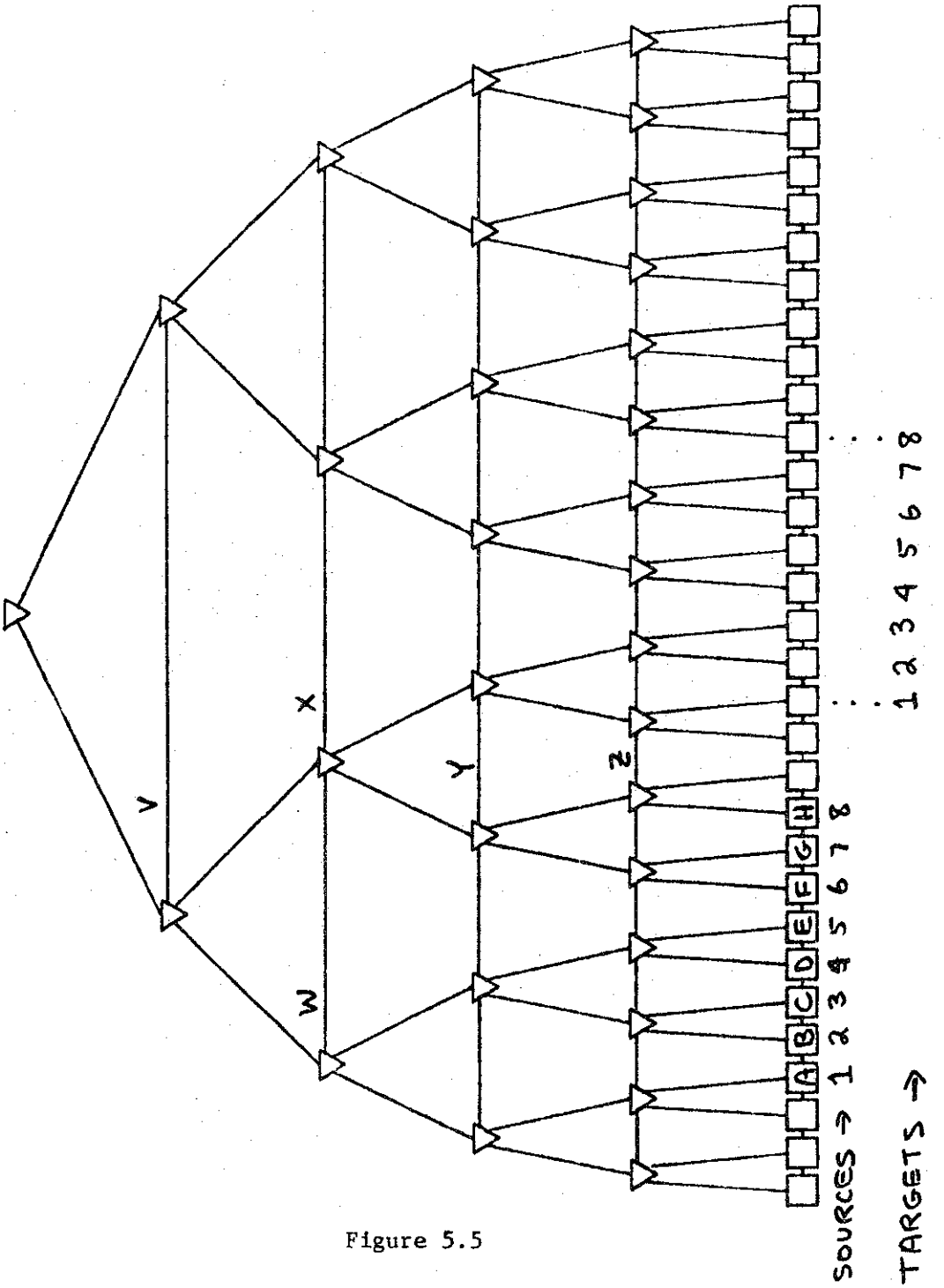


Figure 5.5

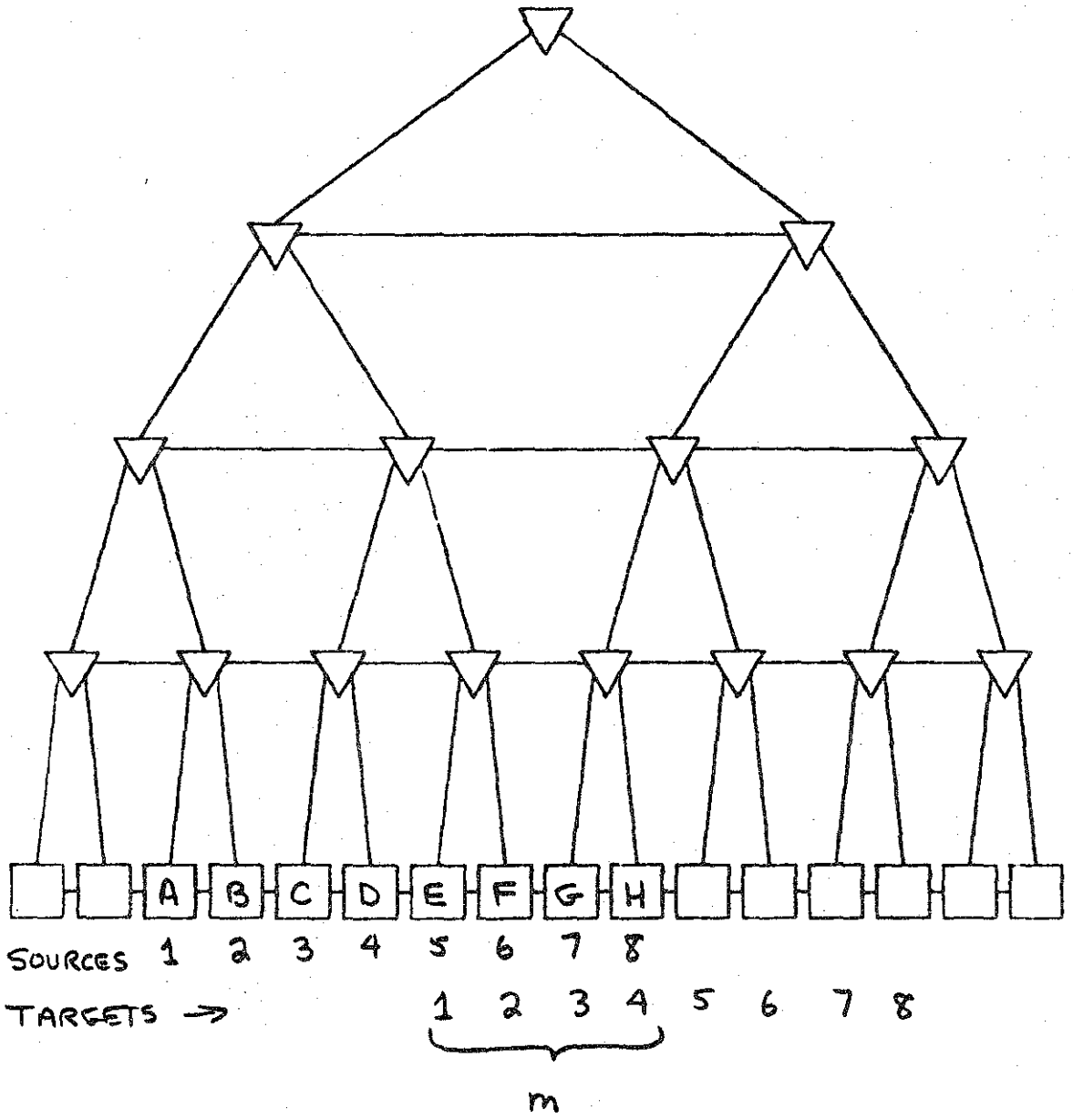
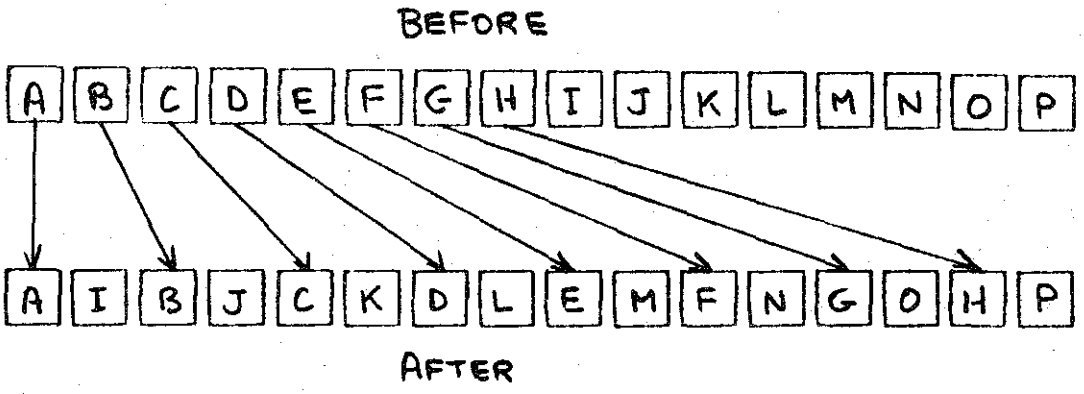
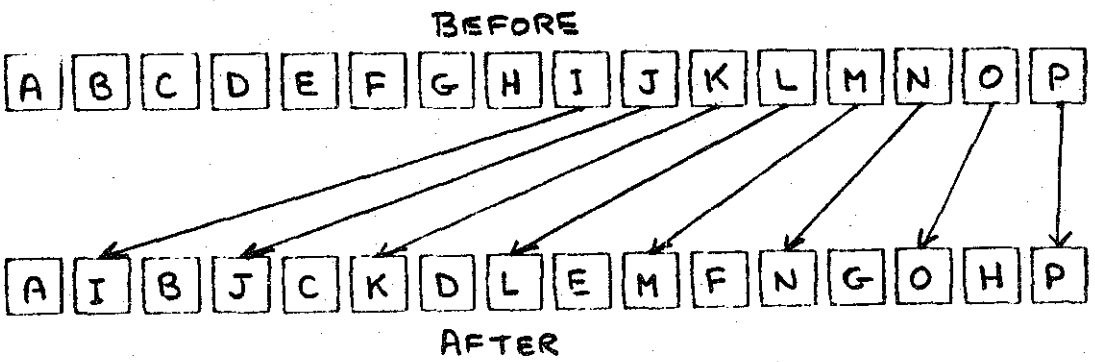


Figure 5.6


BEFORE
A B C D E F G H I J K L M N O P
A I B J C K D L E M F N G O H P
AFTER

(a)


BEFORE
A B C D E F G H I J K L M N O P
A I B J C K D L E M F N G O H P
AFTER

(b)

Figure 5.7

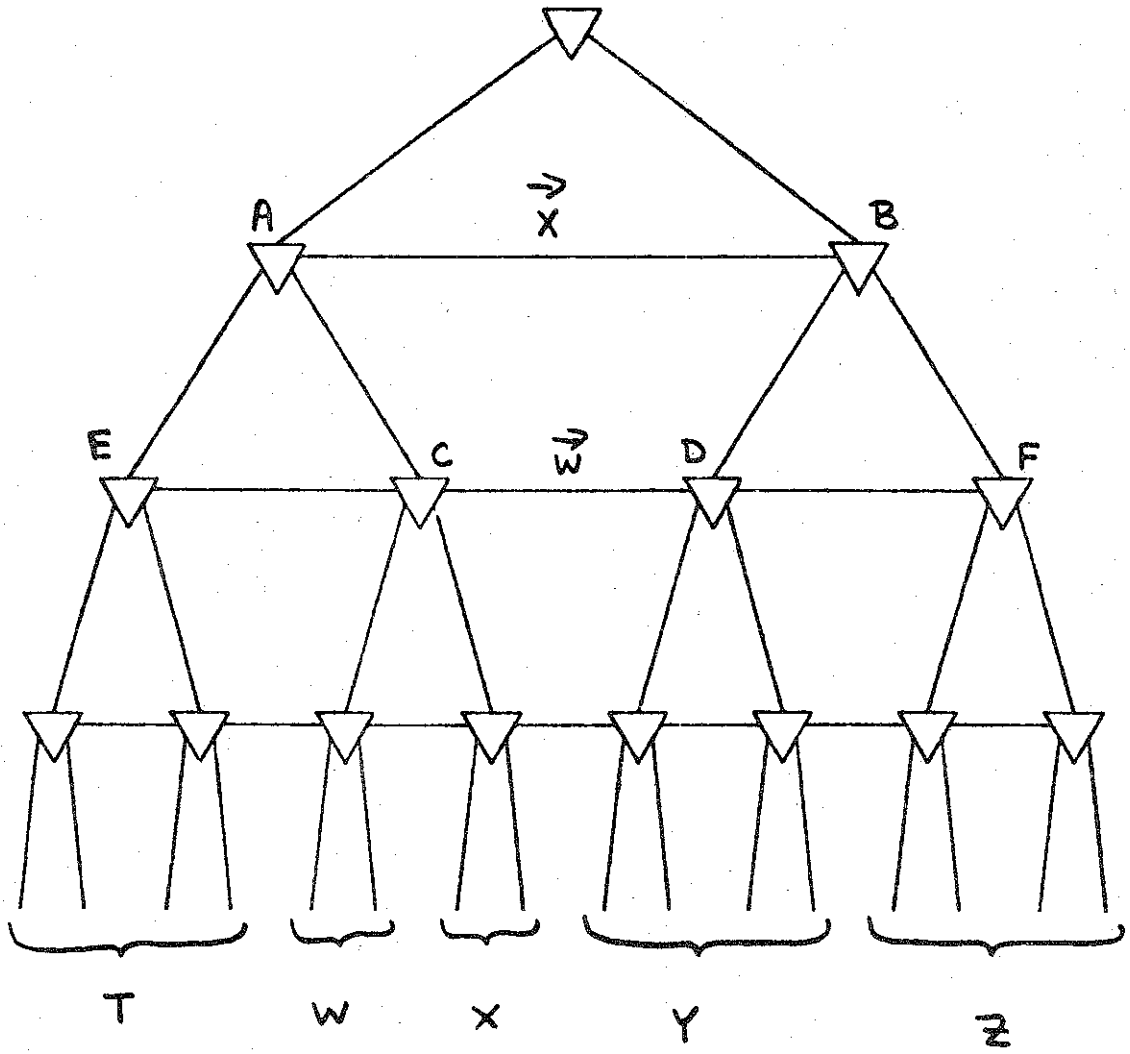


Figure 5.8

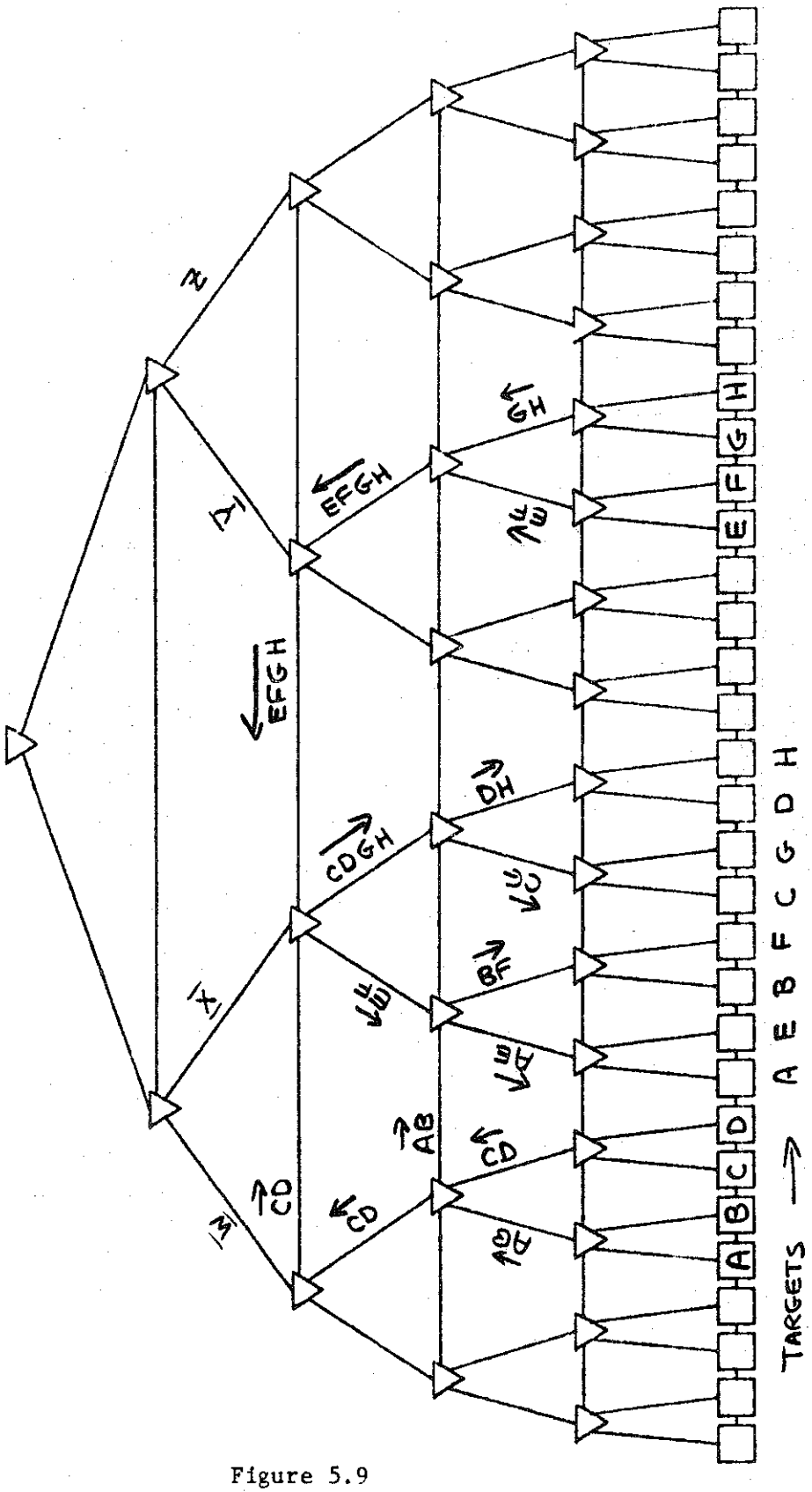

Z
Y
X
W
EFGH
EFGH
CDGH
CD
CD
AB
GH
EF
DH
CD
BF
EF
AB
AG
TARGETS → A E B F C G D H

Figure 5.9

6. BOUNDS ON THE PERFORMANCE OF DATA MOVEMENT ALGORITHMS

The algorithms of Chapter 4 represent a significant improvement over the VERTICAL algorithm. However, they do not improve the order-of-magnitude behavior. That is, they still require $\Theta(n)$ time. In this chapter, we ask whether it is possible to obtain a better order-of-magnitude performance than that of the VERTICAL algorithm. That is, we ask if data movement can be accomplished in less than linear time for any of the patterns investigated in Chapter 5. We will no longer require data items to follow shortest paths or almost shortest paths. We are interested only in the total time required for data movement.

We will approach this question by first concentrating on one problem, namely that of reversing the contents of a group of cells. To simplify the discussion that follows, we will assume in all formal propositions that the data items of interest occupy the leaves of a complete binary tree. In some cases, we will then indicate how the proofs must be modified to handle the more general case.

6.1 The Problem of Reversal

Suppose n=2**k and we wish to reverse the contents of n adjacent cells of L, all leaves of a complete binary tree of height k, as in Figure 6.1. The n/2 items on the left must move to the right side of the tree and the n/2 items on the right must move to the left side of the tree. If only paths below the root are used, then there are only log(n) horizontal connections by which items can get from one side of the tree to the other. This can be seen by drawing a vertical line which divides the n cells and all of their ancestors in half, as in Figure 6.1. This vertical line will cut log(n) horizontal connections. The idea of the algorithm to be presented below is to assign approximately n/(2*log(n)) items to each of the log(n) horizontal cross-connections and then mark out paths by which an item can get to its assigned horizontal connection. The paths will be such that if two items are assigned to a different horizontal cross-connection, then the paths for those two items will be disjoint.

Consider Figure 6.2, which shows the reversal of 32 items. Since log(32)=5, the 16 items in each half have been divided into five regions (groups of horizontally adjacent cells) labeled 1-5. Since the two regions labeled 5 are adjacent, they may be treated as one region. The labels are such that if an item (call it X) is in the kth region from

the left, then the item which is to be interchanged with X is in the kth region from the right. Each pair of items to be interchanged is labeled with the same integer. Some of the channels connecting cells of T have also been given labels between 1 and 5. These labels have been assigned so as to connect each region from the left side of the tree with the corresponding region on the right side. That is, a data item can reach its destination using only channels which are labeled with that item's region number. For all of the data items in one region to reach their destinations, they must queue up at the channel which crosses to the other side of the tree (bold lines in Figure 6.2.) Then these items must cross the channel one at a time and disperse to their destinations in the other half of the tree. This will involve a delay of no more than three steps since there are no more than four items in a region and there is no interference between items in different regions. (There are two cells in Figure 6.2 which must handle data items originating in different regions, namely regions 3 and 4. However, if we examine the internal structure of a node, as in Figure 1.6, we see that an item entering from the left and leaving from the top will not use the same internal registers and connections as an item entering from below and leaving from the right. Therefore, it is possible for two items to pass through the same cell simultaneously without interference.)

An upper bound for the number of time steps required for data movement can be found by adding the number of steps in the longest path to the maximum number of cycles which an item may have to wait to cross the central horizontal channel. Since each region contains no more than four items, no item will have to wait more than three cycles. The longest path (between cells A and B, or between C and D, for example) is nine steps long. Since all channels are two-directional, movement can take place in both directions simultaneously. Therefore, reversing the 32 items will take no more than twelve steps.

We will now show that it is always possible to partition nodes and edges of a binary tree in a manner similar to that shown in Figure 6.2. Further, we will show that the total data movement time is $\Theta(n/(\log(n))$.

First, we will specify a way of dividing the n leaf cells into regions of approximately equal size, except for the middle region, which will be about twice as large as the other regions. We will let k=log(n) and divide the n cells into 2*k-1 regions. To get equal sized regions, we would need n/(2*k) cells per region, except for the middle region which would have n/k cells. However, since n/(2*k) may not be an integer, each region except the middle region must contain FLOOR(n/(2*k)) or CEIL(n/(2*k)) cells. In choosing between FLOOR(n/(2*k)) and CEIL(n/(2*k)), we must be sure

that the jth region from the left has the same number of cells as the jth region from the right. For j<k (that is, for regions other than the middle region), we will use the notation S(n,j) to represent the number of cells in the jth region from the end.

Before giving a formal definition for S(n,j), we will describe a physical procedure for computing it. Draw a line of length n and divide it into 2*k equal sub-intervals with dotted lines. Using solid lines, divide the same line into n equal sub-intervals. (See Figure 6.3 with n=32 and k=5.) Then, for each dotted line except the middle line, find the closest solid line and use this solid line as a marker between regions. In Figure 6.3, the markers are represented by arrows. S(n,j) will be the number of solid-lined sub-intervals between the jth and the (j+1)th marker. By symmetry, j can be counted from either end. Formally, if k=log(n), set

$$S(n,1) = \text{FLOOR}(n/(2*k)+1/2)$$

$$S(n,j) = \text{FLOOR}(j*n/(2*k)+1/2) - \text{FLOOR}((j-1)*n/(2*k)+1/2)$$

for $1<j<k$.

We can now show that it is possible to partition a tree of arbitrary size as in Figure 6.2.

Proposition 1. Consider a complete binary tree with n=2**k leaf cells and suppose the n leaf cells are divided into 2*k-1 regions so that for j<k, the jth regions from the left

and right contain S(n,j) cells and the middle region contains the remainder of the cells. Suppose also that each cell in region j is labeled j and each edge which connects cells in region j is labeled j. Then the remainder of the cells and edges in the tree can be labeled with integers 1 thru k so that there is a path between any of the original n cells and any cell in the symmetrically opposite region such that

1. this path includes only cells and edges labeled with the same integer as the source and target regions (except for the two cells in region 4, level 4 from the top, which must be used by paths with edges labeled 3, as in Figure 6.2);
2. the number of horizontal steps in the path is no more than (2*n)/k.

Proof: The proof uses induction on k. For k≤5, Figure 6.2 shows how to label the edges. For k>5, suppose we have n cells and 2*k-1 regions. We must show how to label the edges of the tree into k sets which satisfy the statement of the proposition. The set of edges associated with the innermost region is simply the set of all horizontal connections in the region. By hypothesis, these edges are labeled with the number of the middle region. Thus, any item in the middle region can reach any cell in the middle region by moving along the bottom level. Since the middle region contains no more than 2*CEIL(n/(2*k)) cells, there

will be no more than FLOOR(n/k)+1 horizontal steps on the path for such an item.

To label the edges corresponding to the other k-1 pairs of regions, consider the m immediate ancestors of the original n cells. Since n is even, m=n/2 by Theorem 1 of Chapter 2. If we let h=log(m), then h=k-1. If we divide the m cells into 2*h symmetric regions each of size FLOOR(m/(2*h)) or CEIL(m/(2*h)), then the induction hypothesis holds and we can conclude that the upper portion of the tree (which contains the m cells and their ancestors but not the original n cells) can be labeled with the integers 1 through h.

Figure 6.4 shows the situation for k=4. Notice in the figure that each region on the upper level overlaps the corresponding region in the lower level. That is, the jth region from the left (or right) in the upper level always contains the father of some cell in the jth region from the left (or right) in the lower level. This means that, in this figure, there is always a path connecting cells in opposite regions and this path uses edges only from the set associated with those regions.

For each vertical edge which connects two regions with the same label (region number), we can label that vertical edge with the number of the regions it connects. This gives the required partition. Therefore, any item in region j of

the bottom level can move horizontally in its region towards the middle of the tree until it reaches a vertical edge labeled j. At this point, it can move to the upper level and follow a path to the other side of the tree. Such a path is known to exist by induction. Finally, the item can move to the bottom level and reach its destination by moving horizontally within region j of the other side of the tree.

Therefore, if we can show that corresponding regions on different levels overlap for $k \geq 5$, we will always be able to label the edges as required.

Consider Figure 6.5, in which cells A and B are the end cells of the jth region from the left on one level and cells C and D are the end cells of the jth region from the left on the next upper level. If we assign each cell an integer index 1,2, ... starting at the left, then we can compute q, the index of B, from the definition of S(n,j). Referring again to Figure 6.3, the value of q is the distance between the first arrow marker and the (j+1)th arrow marker. The (j+1)th arrow marker is located at the solid line closest to the (j+1)th dotted line, which is j*k/(2*n) units from the left end. Therefore,

$$q = \text{FLOOR}(j*(n/(2*k))+1/2).$$

That is,

$$j*(n/(2*k))-1/2 < q \leq j*(n/(2*k))+1/2.$$

Similarly, the index of cell C, which is one cell past the

last cell of the (j-1)th region of the upper level, is

1+FLOOR(((j-1)*n/2)/(2*(k-1))+1/2)

and the index of cell D, the last cell of the jth region of the upper level, is

FLOOR(j*(n/2)/(2*(k-1))+1/2).

Now E, the father of B, has index q/2 or (q+1)/2, depending on whether B is the right or left son. Therefore, the index of E is between (j*n/(2*k)-1/2)/2 and (j*n/(2*k)+3/2)/2. To verify that E is between C and D, we need only verify that

1+(j-1)*(n/2)/(2*(k-1))+1/2 ≤ (j*n/(2*k)-1/2)/2

and

(j*n/(2*k)+3/2)/2 ≤ j*(n/2)/(2*(k-1))-1/2.

The first of these inequalities is equivalent to

j ≤ k - 7*k*(k-1)/n

while the second is equivalent to

5*k*(k-1)/n ≤ j.

Since 1≤j≤k-1 and k=log(n), both of these inequalities hold for k≥10 (or n≥1024). Since the inequalities hold only for k≥10, we must use another technique to establish the assertion that corresponding regions on consecutive levels overlap for 6≤k≤10. To see that this is true, consider the following table which shows the indices of the cells in each region of one half of the tree for k=5,6,7,8,9 and 10. These cell indices are computed according to the definition of S(n,j).

region #	k=5	k=6	k=7	k=8	k=9	k=10
1	1-3	1-5	1-9	1-16	1-28	1-51
2	4-6	6-11	10-18	17-32	29-56	52-102
3	7-10	12-16	19-27	33-48	57-85	103-154
4	11-13	17-21	28-36	49-64	86-114	155-205
5	14-16	22-27	37-46	65-80	115-142	206-256
6		28-32	47-55	81-96	142-171	257-307
7			56-64	97-112	172-199	308-358
8				113-128	200-228	359-410
9					229-256	411-461
10						462-512

For example, when k=5, the first region consists of cells 1, 2, and 3, the second region consists of cells 4, 5, and 6, and so on. Now consider some arbitrary region. For example, choose region 5 when k=9. This region includes cells 115-142 (counting from the left), so the rightmost cell in the region has index 142. Therefore, the father of the rightmost cell has index 71. Now for k=8, region 5 consists of cells 65-80 which include cell 71. Therefore, region 5 for k=8 overlaps region 5 for k=9. We can perform a similar process for each table entry and verify that all corresponding regions on consecutive levels overlap for $6 \leq k \leq 10$. Therefore, the regions overlap for all k.

Given this construction, we can compute the number of horizontal steps in the path. Since we are computing only bounds, we will not require all quantities to be integers. For example, one region may contain FLOOR(n/(2*k)) or CEIL(n/(2*k)) cells. Therefore, the horizontal distance between the end cells of this region is either

FLOOR(n/(2*k))-1 or FLOOR(n/(2*k)). In either case, the horizontal distance is no greater than the quotient n/(2*k).

Any path between cells of L using edges labeled with the same integer will have the ziggurat shape of Figure 6.6. Each horizontal line represents a path over one of the 2*k-1 regions each of which contains n/(2*k) items (except the middle region). For the outermost lines, this path has no more than n/(2*k) edges. The path on the second level has no more than (1/2)*(n/(2*k)) edges since any group of n cells has n/2 immediate ancestors in the tree. The path on the third level has (1/2)*(1/2)*(n/(2*k)) edges, and so on. Therefore, for region j, the number of horizontal steps in any path is no more than 2*(n/(2*k)) * (1 + 1/2 + 1/4 + 1/8 + ... + 1/(2**j)). Since (1 + 1/2 + 1/4 + 1/8 + ...) = 2, we find that the number of horizontal steps is less than (2*n)/k.

Q.E.D.

Corollary. There is a labeling according to Proposition 1 which is symmetrical for the line dividing the tree into two equal subtrees.

With this partitioning scheme, we can now discuss how this algorithm could be implemented on the machine described in [1]. In the preparation phase, all the cells and edges are labeled as in Proposition 1. To do this, the cells of each horizontal level must decide which region they are in.

Each cell can compute this easily using its index (distance from the boundary of the area to be reversed) and the definition of S(n,j). Then the edges connecting cells with the same label are given that label. As a special case, the label 3 must be assigned to four edges which are connected to two cells labeled 4, as in Figure 6.2.

When data movement begins, each data item originating in region j moves upwards and toward the middle of the tree, always using edges and cells labeled j. Conflicts among items using edges with the same label can be resolved at random. Eventually, all the elements originating in one region will be waiting to cross the middle channel corresponding to that region. The items cross the channel one by one. Upon reaching the other side, these items are broadcast throughout the corresponding region on the other side of the tree. The target cells pick out the items they are waiting for and discard the rest. Figure 6.7 shows the path of item C moving through the tree partitioned as in Figure 6.2.

We now compute a bound on the time required for data movement using the above technique. The first data item to reach its middle horizontal channel will do so in no more than (k-1)+n/k steps since the path it follows will have no more than k-1 vertical steps and n/k horizontal steps. This item will be followed by no more than CEIL(n/(2*k))-1 items

which originated in the same region. These items must cross the middle channel one at a time. Finally, the last item will reach its destination after crossing the horizontal channel in no more than (k-1)+n/k steps. Therefore, the total number of time units is bounded by (5/2)*n/k + 2*k-2, which is Θ(n/log(n)).

The construction of Proposition 1 does not necessarily give the best partition of the tree and the analysis gives only a bound on the number of steps required, so it may be possible to improve on the coefficient 5/2. However, since n/2 items must somehow be squeezed through the log(n) central horizontal channels, at least (n/2)/log(n) steps are required for reversal. Therefore, no partition will give better than Θ(n/log(n)) asymptotic behavior.

As noted previously, Proposition 1 assumes that n, the number of items to be reversed, is a power of two and that the n items are the leaves of a complete binary tree. If we relax these restrictions and let k=FLOOR(log(n)), then the result of Proposition 1 is still true. To prove this requires the examination of a number of different cases. If n is odd, then m, the number of cells on the next level, is (n+1)/2. If n is even, then the number of cells on the next level is either n/2 or n/2+1. For each of these cases, one must establish the fact that regions of the same number overlap as in Figure 6.4. This can be done with the same

technique as that used in the proof of Proposition 1. We will omit further details.

6.2 Reversal of Non-adjacent Cells

The preceding algorithm is practical in the sense that it can be implemented on a machine of the kind described in [1]. However, it is unrealistic in that it requires the data items being reversed to occupy adjacent cells. If we relax this restriction, we cannot always apply the algorithm directly. Consider Figure 6.8, which shows 16 data items occupying a 32-cell area. If these 32 cells are divided into the regions as in Figure 6.2, then region 2 will include three data items on the left and two items on the right. Therefore, two of the items on the left will have to use edges labeled with a different region number to reach their destinations. This would invalidate the analysis of the previous section.

Therefore, we will generalize Proposition 1 by dividing the n cells of L into regions such that each region contains the same number of occupied cells. Specifically, each region except the middle region will include FLOOR(n/(2*k)) or CEIL(n/(2*k)) occupied cells, where k=FLOOR(log(n)). The total number of cells in a region will depend on how many empty cells surround the occupied cells. Therefore, the number of regions will depend on the percentage of occupied

cells in the active area of L which contains the symbols to be reversed.

Proposition 2. Let n=2**k and consider a complete binary tree of n leaf cells. Suppose the n cells are divided into an odd number of regions so that, with the possible exception of the middle region, the jth regions from the left and right contain at least S(n,j) cells. (Each region of the bottom level will contain S(n,j) occupied cells plus all of the empty cells in between.) Suppose also that each cell in region j is labeled j and each edge which connects cells in region j is labeled j. Then if the number of regions is 2*h-1, then the cells and edges of the tree can be labeled with the integers 1 thru h so that there is a path connecting any two cells in symmetrically opposite regions such that:

1. the path includes only cells and edges labeled with the same integer as the source and target regions (with the possible exception of the two cells in region 4, level 4, which may be used by paths with edges labeled 3);
2. the number of horizontal steps in the path is no more than (2*n)/k.

Proof: (The proof is analogous to that of Proposition 1.) The proof is by induction on k. For k≤5, we can exhibit the partition explicitly. Suppose we have n cells and 2*h-1

regions, $h \leq k$, numbered $1,2,\ldots,h-1,h,h-1,1,\ldots,2,1$. Then, as in the proof of Proposition 1, we will divide the m fathers of the original n cells into regions, and show that corresponding regions on two consecutive levels of the tree can be connected. That is, the ith region from the left of the upper level contains the father of a cell in the ith region from the left on the lower level.

For each side of the tree, we compute the first h-1 regions of the upper level (with m cells) one at a time, starting with the outermost region. To define one region (say region i from the left) start at the cell immediately adjacent to the previous region and include all cells up to and including the father of the rightmost cell of region i in the lower level. Then add to these as many cells as needed to give a total of at least $S(m,i)$ cells in this region. The remaining cells of the upper level, if any, will belong to the middle region, region h. (See Figure 6.9.) Note that there will be cells in region h of the upper levels only if $h<k$, which implies $h \leq k-1=\log(m)$ so the induction hypothesis still holds. If the two regions labeled h-1 overlap (as in Figure 6.10), then they should be merged to give a new middle region.

Before continuing with the proof of Proposition 2, let us examine an example of this construction. Figure 6.11 shows how the partition would be constructed for a tree of

size 64 which has 40 occupied cells and 24 empty cells in L. For the first level (L), n=64, k=6, q=FLOOR(n/(2*k))=5 and r=4. To mark out the regions, we first count off the first 6 occupied cells (since S(64,1)=6). This defines the left and right regions labeled 1. We then do the same for left and right regions 2 and 3. At this point, there are only 4 occupied cells left so they are all assigned to region 4, the middle region. On level 2, we have n=32, k=5, q=3 and r=2. The left region 1 consists of the first 6 cells, all of which are fathers of region 1 cells from the first level. Similarly, the right region 1 consists of the first 5 cells. Now, for region 2 on the left, only the next 3 cells are fathers of region 2 cells in the first level. But S(32,2)=4. Therefore, region 2 on the left must include the next 4 cells. The rest of the regions are defined similarly. On level 3, we have n=16, k=4, q=2, and r=0; on level 4, n=8, k=3, q=1, and r=2; on level 5, n=4, k=2, q=1 and r=0. Finally, on level 6, n=2, k=1, q=1, and r=0. This construction labels each cell with an integer between 1 and 4. To label the edges, we merely apply the following rule: if the connected cells have label i, then the edge connecting them should be given label i. Figure 6.12 shows the edges which would be labeled '2' in the example of Figure 6.11.

We now resume the proof of Proposition 2. By the same argument as in the proof of Proposition 1, we can show that

the ith region on the upper level always contains the father of some cell in region i of the lower level for $k \geq 5$. Furthermore, the method of constructing the regions in the upper level guarantees that only the middle region can contain fewer than $\text{FLOOR}(m/(k-1))$ cells. Since $\text{FLOOR}(m/(k-1)) \leq S(m,j)$, the induction hypothesis holds and edges and cells of the upper portion of the tree can be labeled. Labeling the edges which connect corresponding upper and lower regions on the same side of the tree completes the construction. The path of any item will be a ziggurat shape as in the proof of Proposition 1 (Figure 6.6). The longest horizontal path for one level is shown in Figure 6.13. Here an item on the left end of region j must travel horizontally under region j-1 of the upper level. Then it can move vertically to the next level. We need to calculate the length of this path and the other horizontal paths. But if region j-1 of the lower level contains enough cells, then region j-1 of the upper level may not have to overlap the lower region j. (Recall, for example, the lower left regions (j=2) of Figure 6.11.) In this case, there would be no horizontal steps in going from the lower level to the upper level. Therefore, the longest horizontal path will be needed when region j-1 (and regions $j-2, j-3, \ldots, 1$) of the lower level are as small as possible. That is, the number of horizontal steps will be greatest if each region j has only $S(n,j)$ cells. But this is exactly the case which

was analyzed in Proposition 1. Therefore, the result is the same. Namely, there are no more than (2*n)/k horizontal steps.

Q.E.D.

With Proposition 2, we can develop an algorithm for reversal which does not assume that the items to be reversed occupy adjacent cells. The algorithm follows:

1. Let the items to be reversed occupy some of the leaf cells of a complete binary tree of size n and let k=log(n).
2. Count off the leftmost S(n,1) occupied cells and the rightmost S(n,1) occupied cells and assign them and all intervening empty cells a region number of 1. (This will create two regions, one on the left and one on the right.)
3. For j=2,3,...,k-1 but only as long as there are 2*S(n,j) occupied cells which haven't been labeled, count off the next S(n,j) occupied cells from the left and the next S(n,j) occupied cells from the right and assign them and all of the empty cells between them the region number j.
4. Place the "leftover" cells in one region in the middle.
5. Using these regions, partition the tree as described in Proposition 2.

Once the edges have been labeled, data movement may begin. Each item rises as high as it can, remaining on edges assigned to its region. (By rising as high as possible, the item can follow the shortest path among those

paths which use edges of the same label.) The item then crosses the "middle line" and is broadcast to all the cells in its region on the other side of the tree.

As before, the total number of steps required for reversal is bounded by

```
   2*n/log(n)     horizontal steps
 + 2*log(n)       vertical steps
 + n/(2*log(n))   steps used crossing the middle
                  horizontal channel.
```

Again, the number of steps is $\Theta(n/\log(n))$. Recall that n is the total number of cells of L in the active area, rather than the number of items moving.

6.3 Alternative Generalizations of Proposition 1.

The preceding algorithm based on Proposition 2 is not the only way to generalize the algorithm of Proposition 1. Here we examine some of the other possibilities.

If there are p data items in an n-cell area (leaving n-p empty cells), then the algorithm just presented will divide each side of L into $(p/2)/(n/(2*\log(n))) = (p/n)*\log(n)$ regions since each region contains $n/(2*\log(n))$ occupied cells. Therefore, only $(p/n)*\log(n)$ of the edges which connect one side of the tree to the other will be used. We might try to use all log(n) such edges, dividing each half of the occupied cells into log(n) regions of size

(p/2)/log(n), and connecting corresponding regions. This can work well if the non-empty cells are scattered reasonably well in the n-cell area. (See Figure 6.14.)

However, if the items to be moved are all located on the edges of the area as in Figure 6.15, then some of the paths can get very long. In fact, the path connecting cells in the innermost region can take almost n steps if p<<n. Therefore, while the number of items in each group is smaller, (namely, p/(2*log(n))), the path length could increase almost to n. That is, the time for data movement could be linear in the size of the area.

Another alternative is to make the number of regions depend only on p, the number of occupied cells. We might expect to find an $\Theta(p/\log(p))$ algorithm since if n=p (that is, if there are no empty cells in the area) we have an $\Theta(n/\log(n))$ algorithm. This might be done by using log(p) regions in each half and log(p) of the "middle" edge connections. However, as in the last example, this can give very long paths if all the empty cells are in the middle of the area. In fact, the length of the path could be close to p giving an $\Theta(p)$ algorithm. The reason for the $\Theta(p)$ term is that if there are log(p) levels in use, then the bottom level has 2**log(p)=p cells, but half of these could be empty. Therefore, the middle region of the lowest level could include p/2 empty cells.

To improve on this bound, we must use fewer than log(p) levels, selecting levels in a way which depends on the distribution of the occupied cells. For example, using only log(p)/r levels, with r>1, the longest path would have $\Theta(2^{**}(\log(p)/r)) = \Theta(p^{**}(1/r))$ horizontal steps and $\Theta(\log(n))$ vertical steps and each region would have $r*p/\log(p)$ items, giving an $\Theta(p^{**}(1/r)) + \Theta(p/\log(p)) + \Theta(\log(n))$ algorithm. This idea is workable but it is not a generalization of the algorithm of the previous section. That is, if p=n, it does not give $\Theta(n/\log(n))$ performance.

6.4 Pivoting

The problem of reversal is a special case of a more general problem which we will call pivoting. For pivoting, a position in L is chosen as the pivot point. Then all the data items to the left of the pivot point are moved to the right (in reverse order) and the items to the right of the pivot point are moved to the left. Figure 6.16 shows two such patterns. Notice that the pivot point may be outside the area which contains the data items to be moved. Reversal is a special case of pivoting with the pivot element chosen in the middle of the items to be moved.

Pivoting can be handled using the same techniques as those used for reversal. The only difference is that some items will move to cells which were not previously occupied.

This means that the regions will have to be determined using all of the cells involved in data movement (sources, and targets), not just the source cells. Specifically, we would mark all of the cells which are either sources or targets and apply the reversal algorithm of section 6.2 using "marked cells" in place of "occupied cells."

6.5 Translation

The solution for other patterns can be constructed from a sequence of pivot operations. For example, suppose we want to translate the contents of a group of (not necessarily adjacent) cells to a different area of L, preserving the original order. We can do this in two steps as shown in Figure 6.17. First, reverse the contents of the cells (A ... Z in Figure 6.17). Then pivot the entire group to the target area. The number of steps required for this operation is $\Theta(n/\log(n))$ where n is the number of cells of L in the area where movement is taking place.

6.6 Arbitrary Movement Patterns

In general, any movement pattern which does not involve multiple copies can be decomposed into a sequence of pivot operations. (Here, "can be decomposed" means that there is a way for a global observer to decompose the pattern. It does not imply that there is a practical way for the machine

of [1] to find the proper pivots.) To see that this can be done, consider any movement pattern and suppose all the source and target cells occupy a region of size n. Divide this region in half, giving two smaller regions of size FLOOR(n/2) and CEIL(n/2). Then take each half and mark all the cells which contain items whose targets are in the other half. Suppose we find l items in the left half which must move to the right and r items in the right half which must move to the left. Without loss of generality, assume l>r. Then there must be at least l-r empty spaces somewhere in the right half of the tree. Mark l-r of these also and perform a pivot operation on all marked cells. Use the middle of the area as the pivot point. (See Figure 6.18 where l=5 and r=2.)

At this point, it is quite possible that none of the data items will have reached its final destination. However, all of the items are known to be in the correct half of L. So we can apply the procedure described above to each half, as in Figure 6.19. This requires a pivot operation for each half. But these can be performed simultaneously since pivots take place in different areas of the tree. There will be no conflict at the boundary because the reversal (and therefore the pivot) algorithm presented previously does not use the outermost edges. That is, in Figure 6.19, the F and the E will not try to use any of the same edges even though they may have any number of ancestors

in common. Clearly, the procedure just described can be repeated log(n) times, at which time all of the items will have reached their destinations.

The time required for the first step is roughly C*n/log(n) for some C. The time for the second is C*(n/2)/log(n/2), since both pivots occur simultaneously. Therefore, the time required for any movement pattern is

$$C * (n/\log(n) + (n/2)/\log(n/2) + \ldots + 2/\log(2))$$

As noted by Tolle[13], there is a bound on the sum of this series.

Lemma. Let n=2**k, where k=2**(m+1)-1. Then

$$2^{**}k/k + 2^{**}(k-1)/k-1 + \ldots + 2/\log(2) \leq O(n*\mathrm{loglog}(n)/\log(n))$$

(To increase the readability of some of the formulas in this proof, we will use "loglog(n)" instead of "log(log(n))".

Proof: Denote the sum by R(k). Reversing the order of the summands,

$$R(k)=2/1 + 4/2 + 8/3 + 16/4 + \ldots + 2^{**}(k-1)/k-1 + 2^{**}k/k$$

Replacing each denominator by the largest power of 2 which is not larger than the denominator, we obtain

$$R(k) < (2/1) + (4/2 + 8/2) + (16/4 + 32/4 + 64/4 + 128/4) + \ldots + (2^{**}(2^{**}m)/2^{**}m + \ldots + 2^{**}(2^{**}(m+1)-1)/2^{**}m)$$

since 8/3 < 8/2, 32/5 < 32/4, and so on. Combining terms with like denominators, we get

$$R(k) < 2/1 + (2^4-2^2)/2^1 + (2^8-2^4)/2^2 + (2^{16}-2^8)/2^3$$

$$+ (2^{32}-2^{16})/2^{4} + \ldots$$

$$+ (2**2**(m+1)-2**2**m)/2**m$$

or

$$R(k) < (2^{2}-2^{1}) + (2^{3}-2^{1}) + (2^{6}-2^{2}) + (2^{13}-2^{5}) + 2^{28}-2^{12}) +$$

$$\ldots + (2**(2**(m+1)-m) - 2**(2**m-m)).$$

Rewriting, we get $R(k) < S(m)-U(m)$ where

$S(m) = 0+2^{2}+2^{3}+2^{6}+2^{13}+2^{28}+ \ldots +2**(2**m-m+1)+2**(2**(m+1)-m)$

$U(m) = 2^{1}+2^{1}+2^{2}+2^{5}+2^{12}+2^{27}+ \ldots +2**(2**m-m).$

Subtracting, we get $S(m)-U(m) =$

$(0-2^{1})+(2^{2}-2^{1})+(2^{3}-2^{2})+ \ldots +(2**(2**m-m+1)-2**(2**m-m)))$

$+2**(2**(m+1)-m)$

$= -2^{1}+2^{1}+2^{2}+2^{5}+2^{12}+2^{27}+ \ldots +2**(2**m-m) + 2**(2**(m+1)-m)$

or $S(m)-U(m) =$

$+2^{1}+2^{1}+2^{2}+2^{5}+2^{12}+2^{27}+ \ldots +2**(2**m-m)$

$+ 2**(2**(m+1)-m) -4.$

Since there are m+1 terms on the top line of this expression, we get

$$R(k) < (m+1)*(2**(2**m-m)) + 2**(2**(m+1)-m) -4.$$

Now m+1 is approximately loglog(n), 2**m is approximately log(n), and 2**(2**m-m) is approximately n. Therefore,

$$R(k) < O(\text{loglog}(n)*n/\log(n))$$

Q. E. D.

Therefore, any movement pattern which does not involve

multiple copies can be accomplished in less than linear time.

As noted previously, the preceding argument is an "existence proof." It shows that the movement in question can be accomplished in a certain time but it does not show that it is possible for the machine to figure out how to move the data in that time.

6.7 Summary of Results

1. There is a technique for reversing the contents of a group of n adjacent cells of L in $\Theta(n/\log(n))$ steps. This technique could be implemented on a machine of the type described in [1].
2. The contents of any set of p cells in an active area of n cells (in L) can be reversed in $\Theta(n/\log(n))$ steps.
3. Any movement pattern taking place in an active area of size n and which does not require multiple copies of data items can be completed in $O(n*\log\log(n)/\log(n))$ steps.

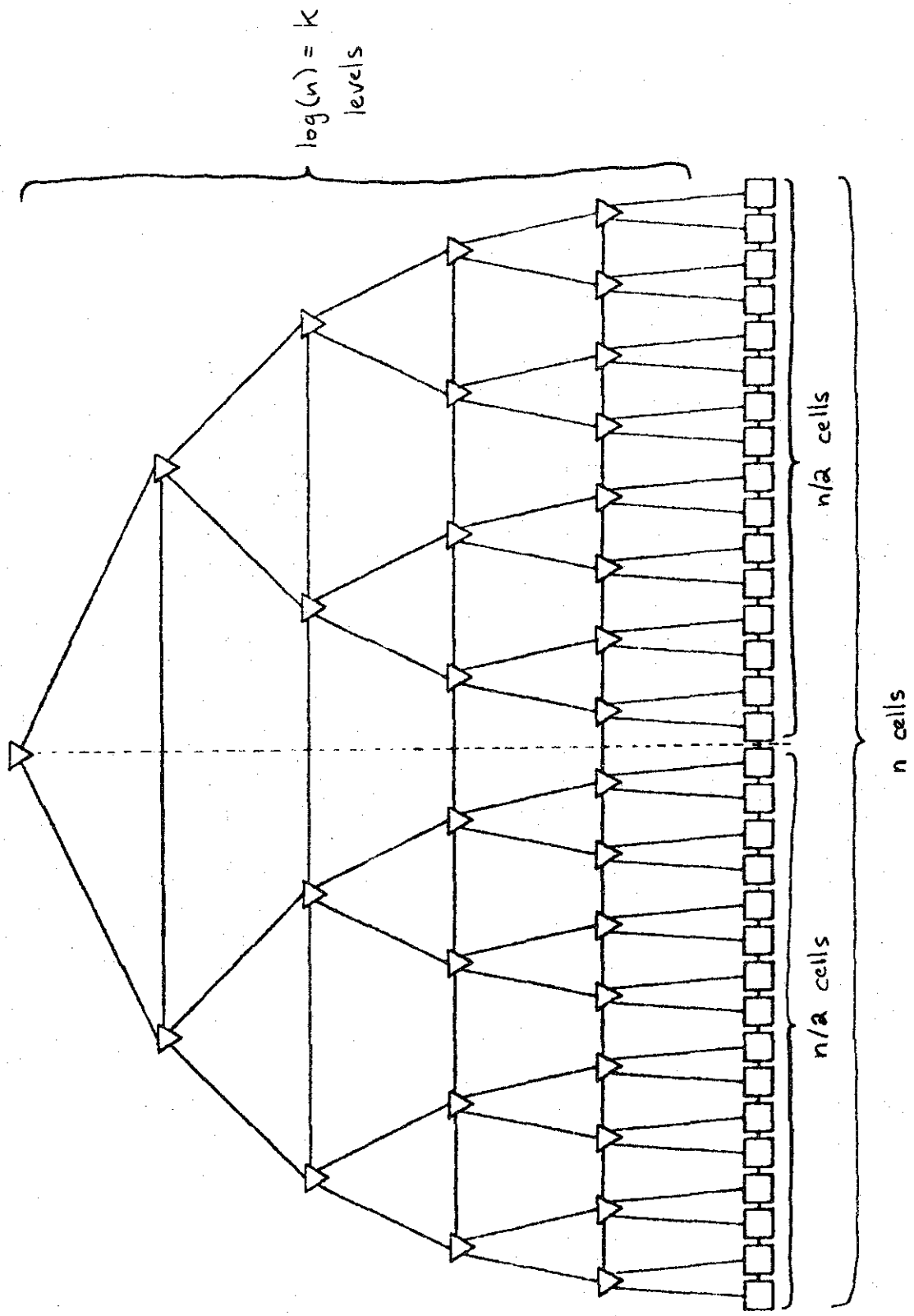


Figure 6.1

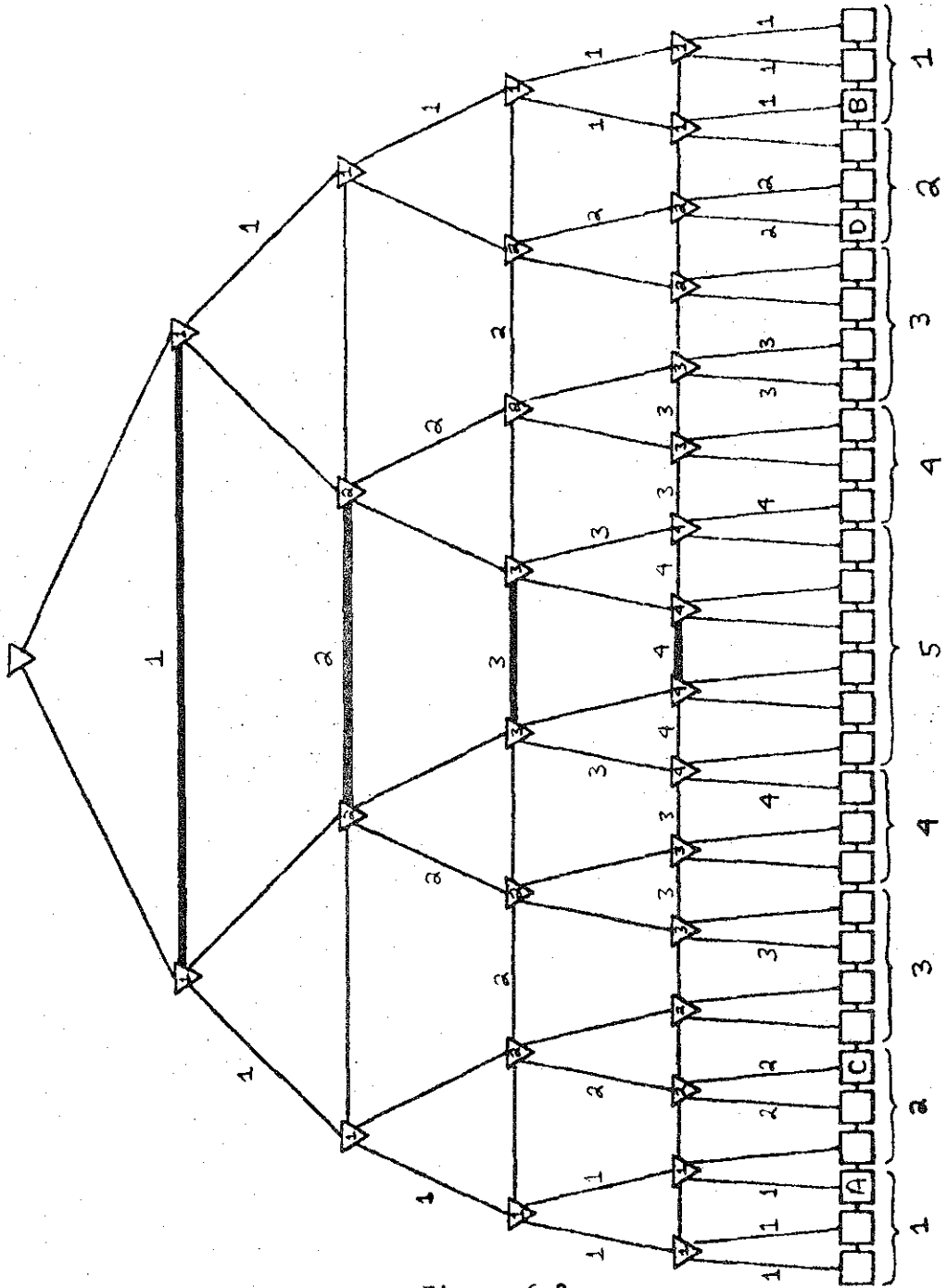


Figure 6.2

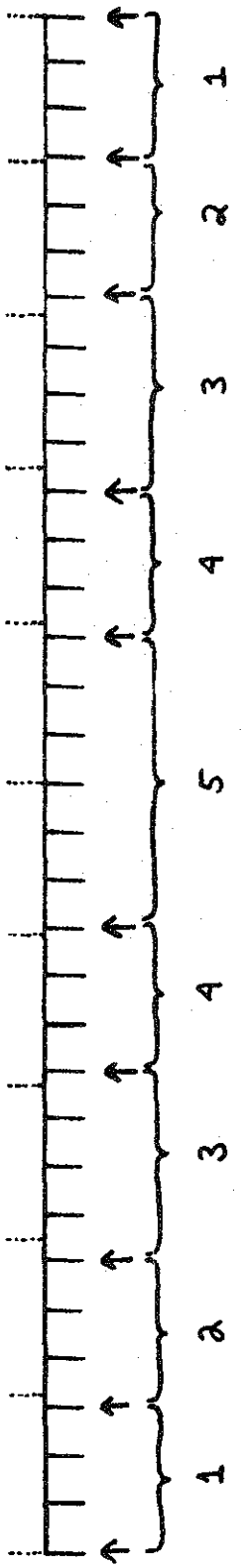


Figure 6.3

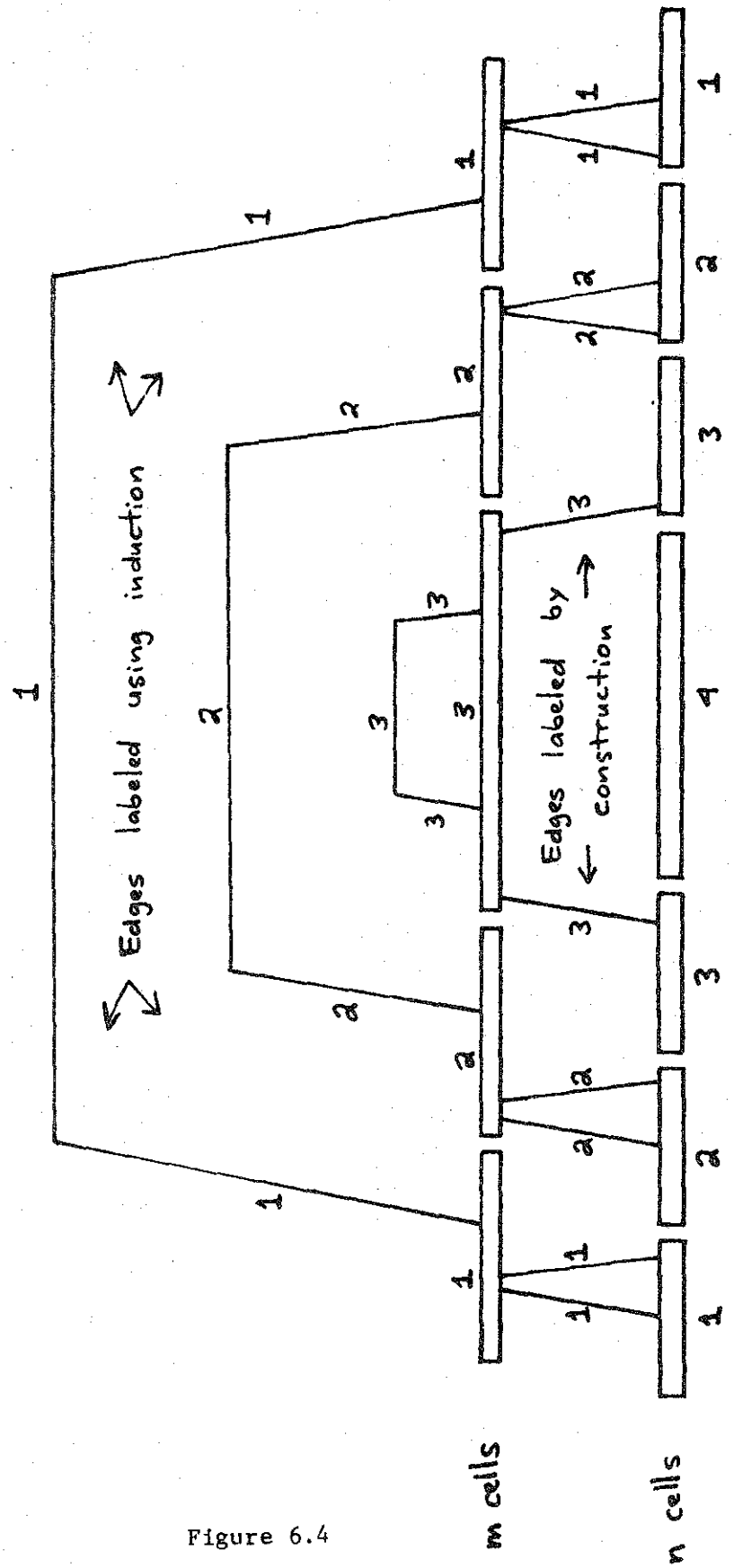

Edges labeled using induction
Edges labeled by construction
m cells
n cells

Figure 6.4

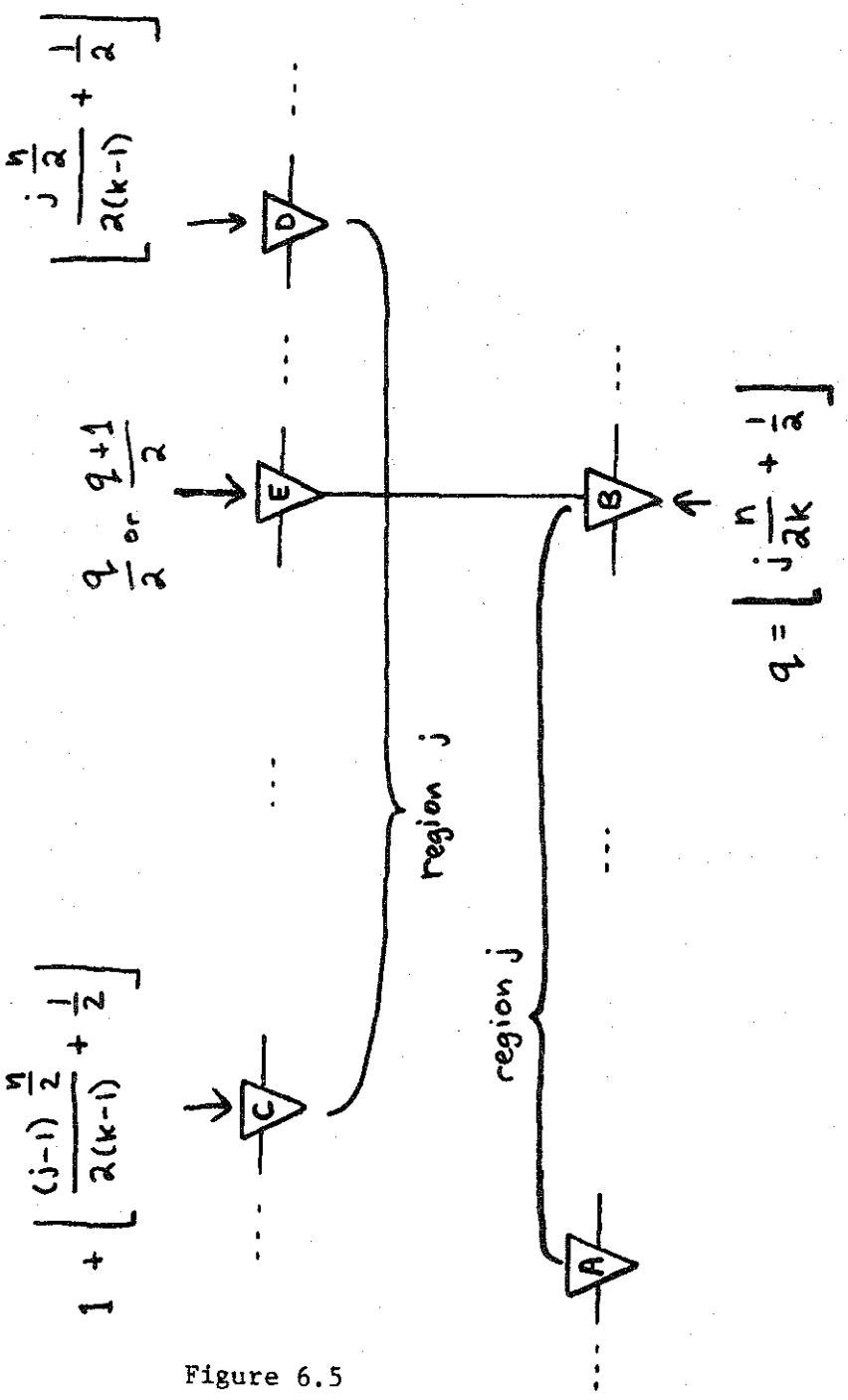

$\left\lfloor \frac{j\frac{n}{2}}{2(k-1)} + \frac{1}{2} \right\rfloor$
D
$\frac{q}{2}$ or $\frac{q+1}{2}$
E
B
$q = \left\lfloor j\frac{n}{2k} + \frac{1}{2} \right\rfloor$
region j
region j
$1 + \left\lfloor \frac{(j-1)\frac{n}{2}}{2(k-1)} + \frac{1}{2} \right\rfloor$
C
A

Figure 6.5

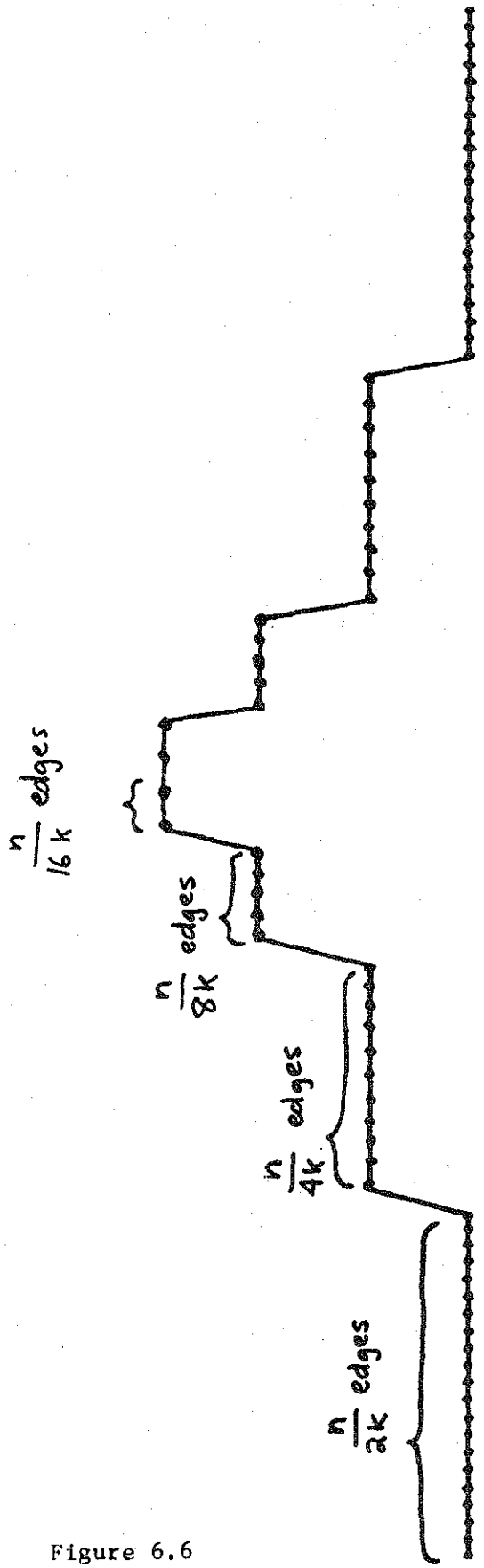


Figure 6.6

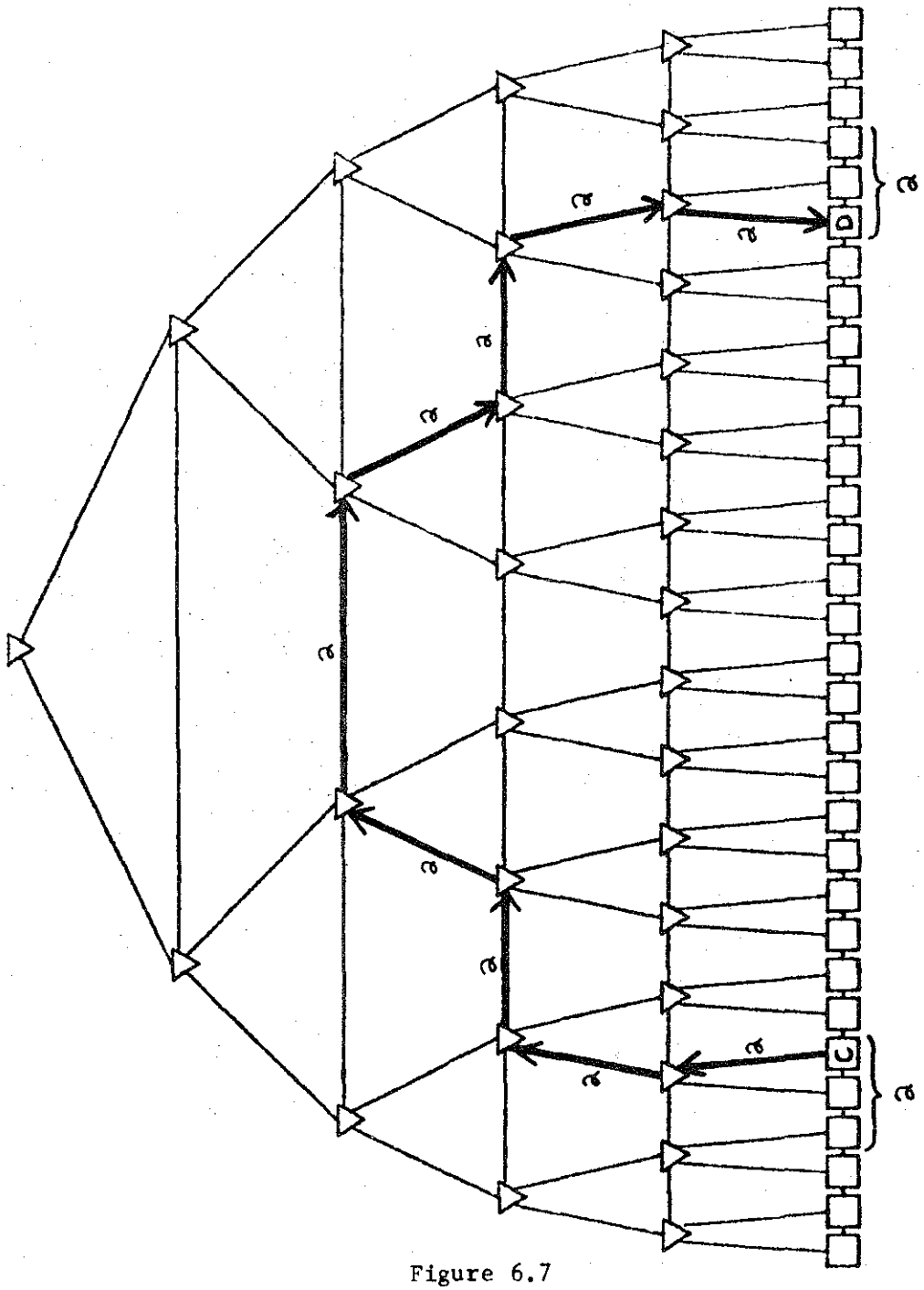


Figure 6.7

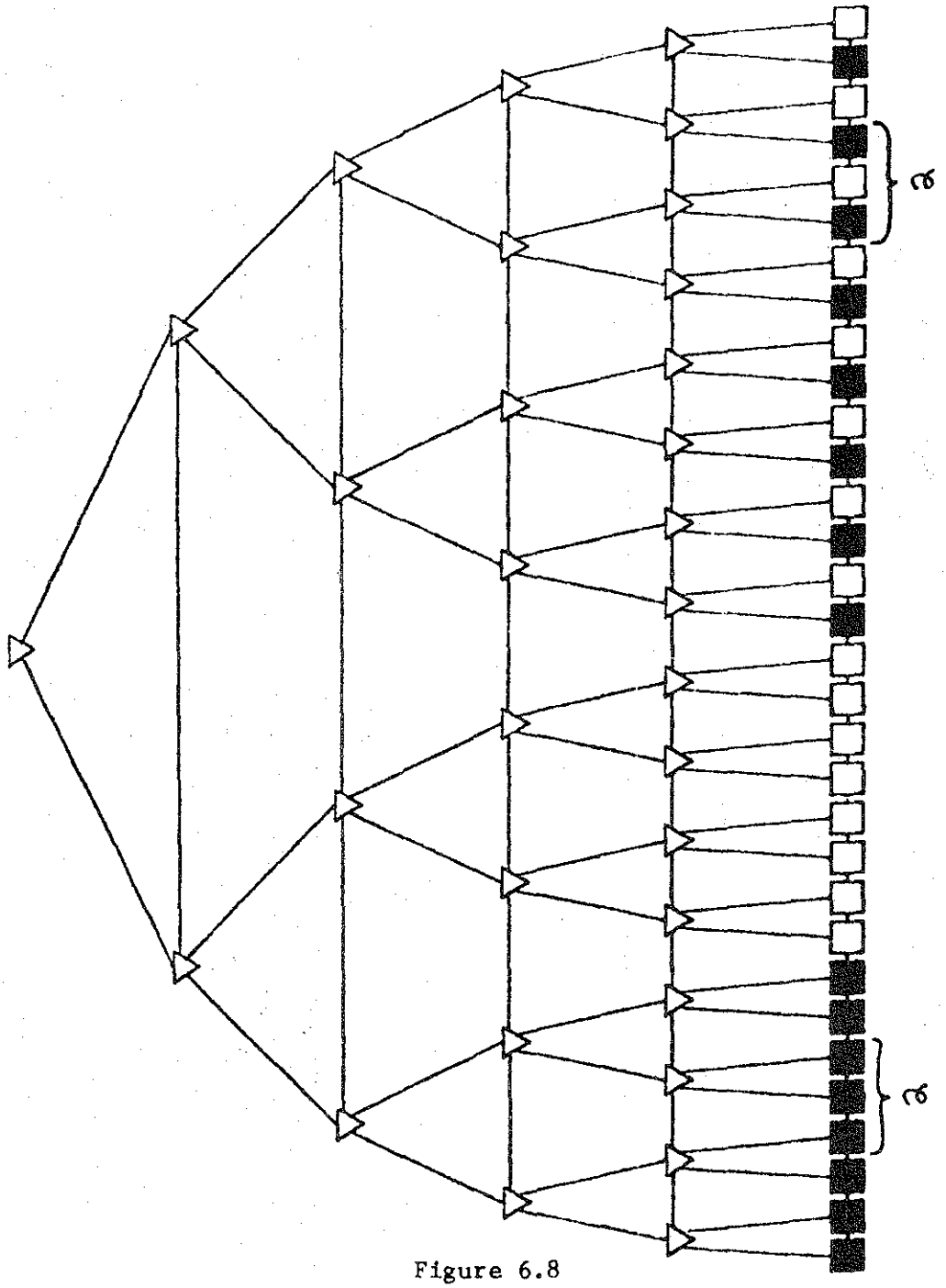


Figure 6.8

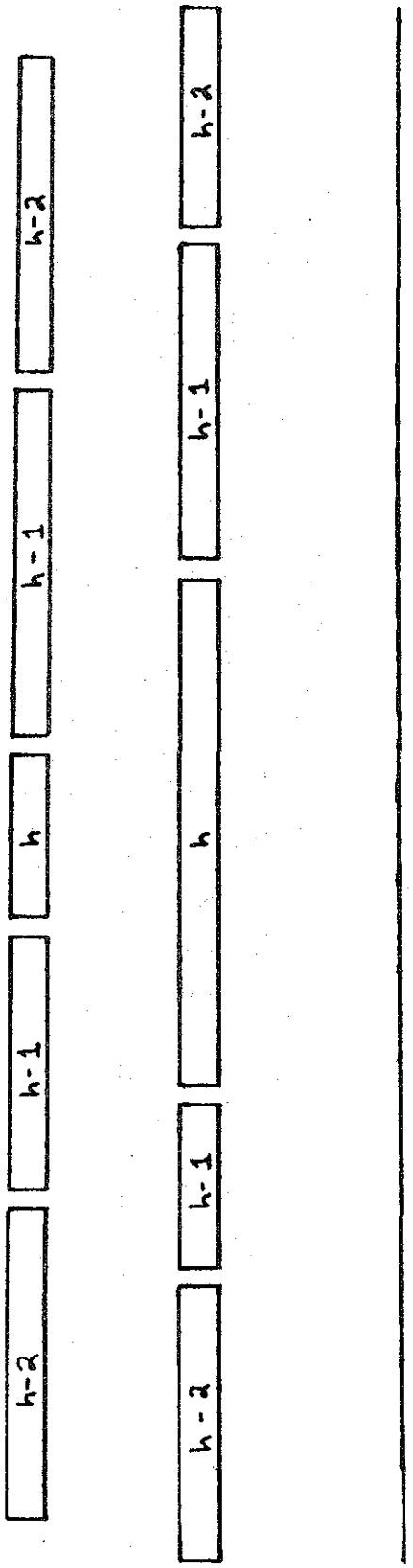


Figure 6.9

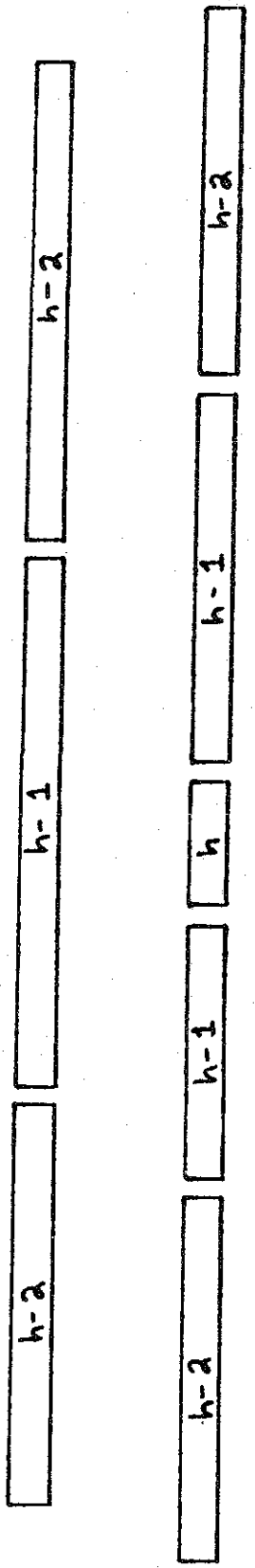


Figure 6.10

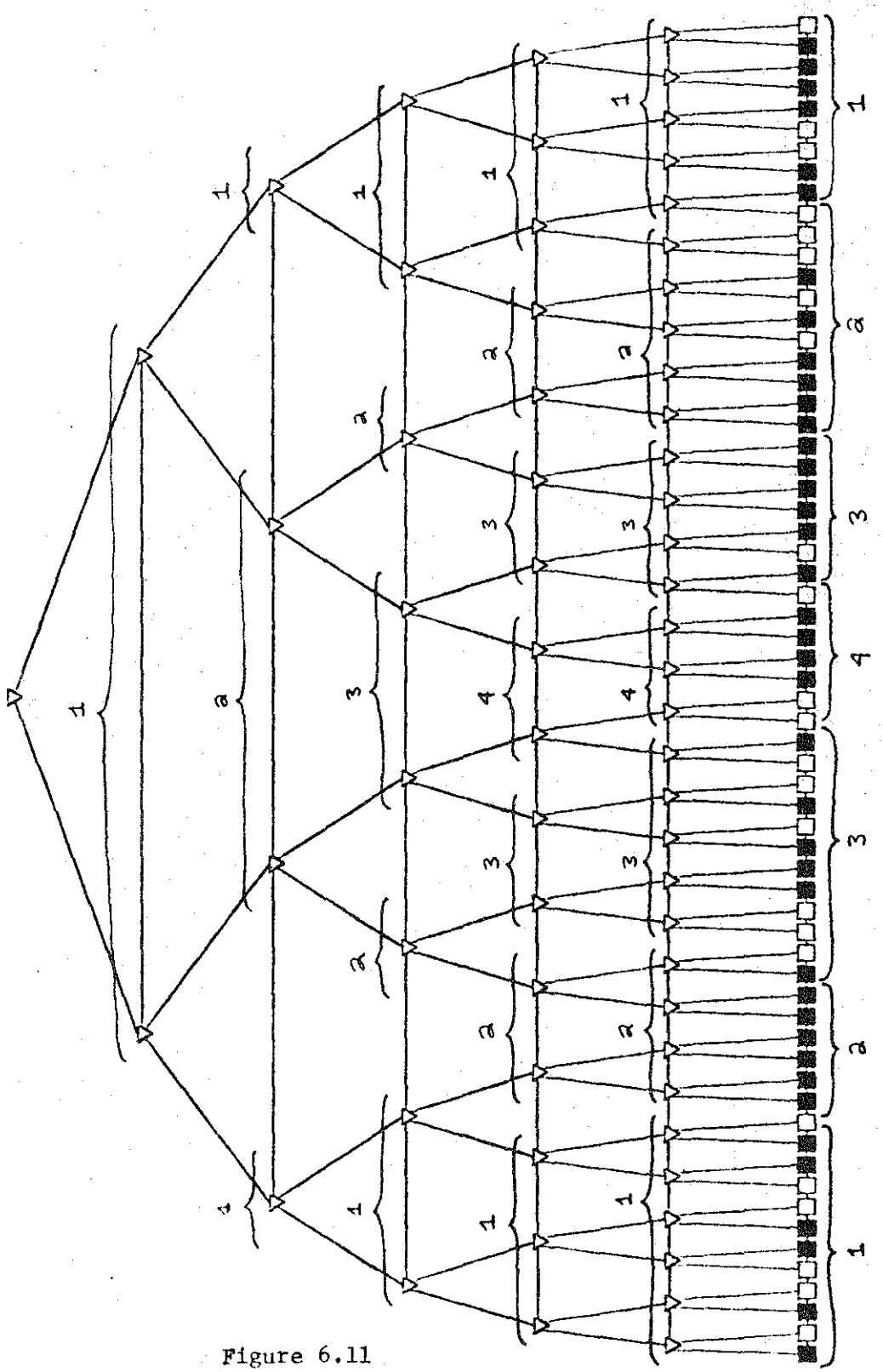


Figure 6.11

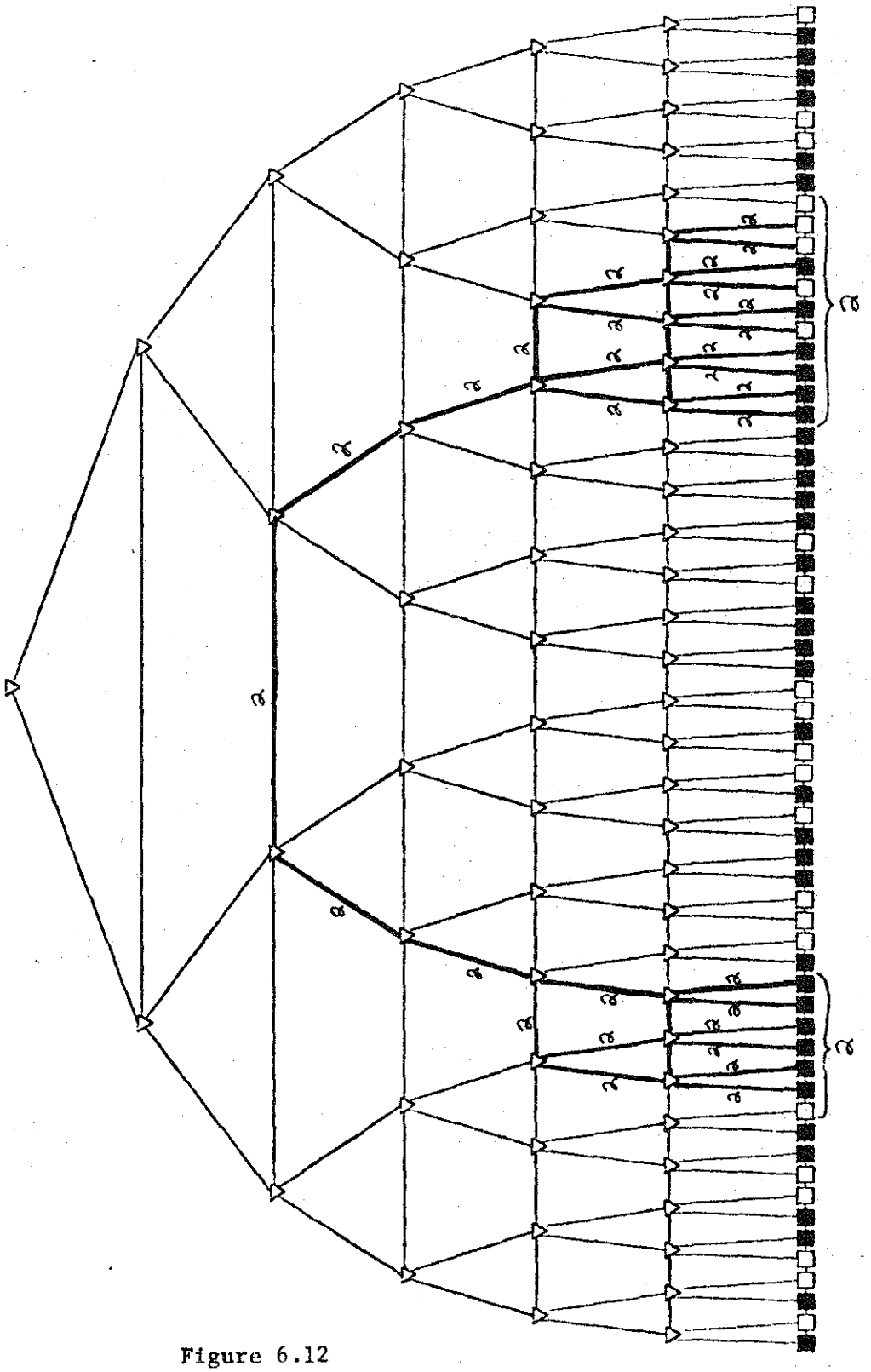


Figure 6.12

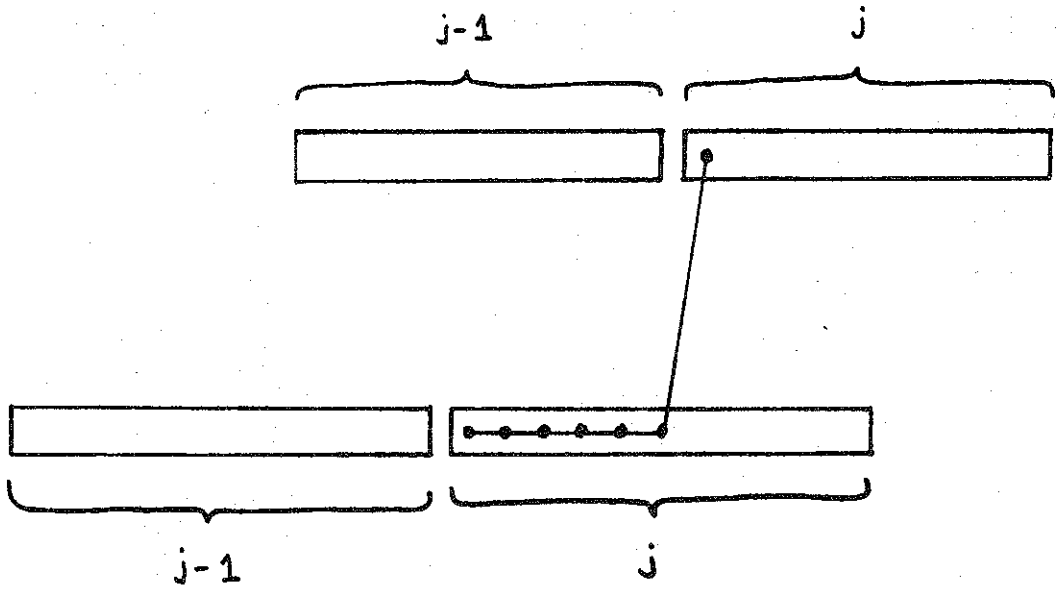


Figure 6.13

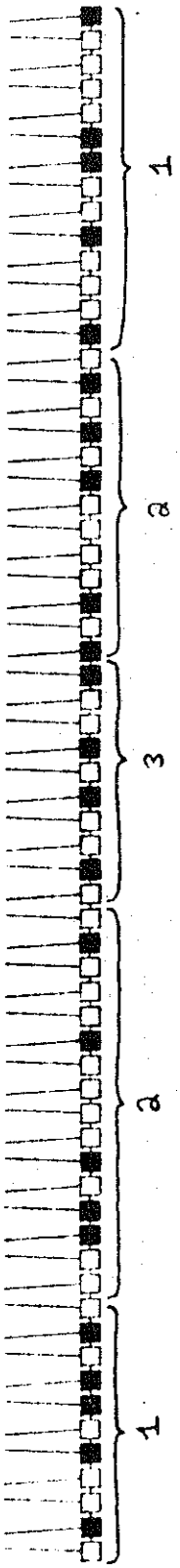


Figure 6.14

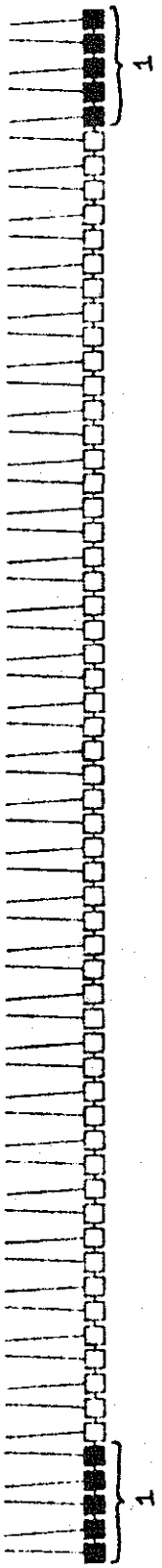
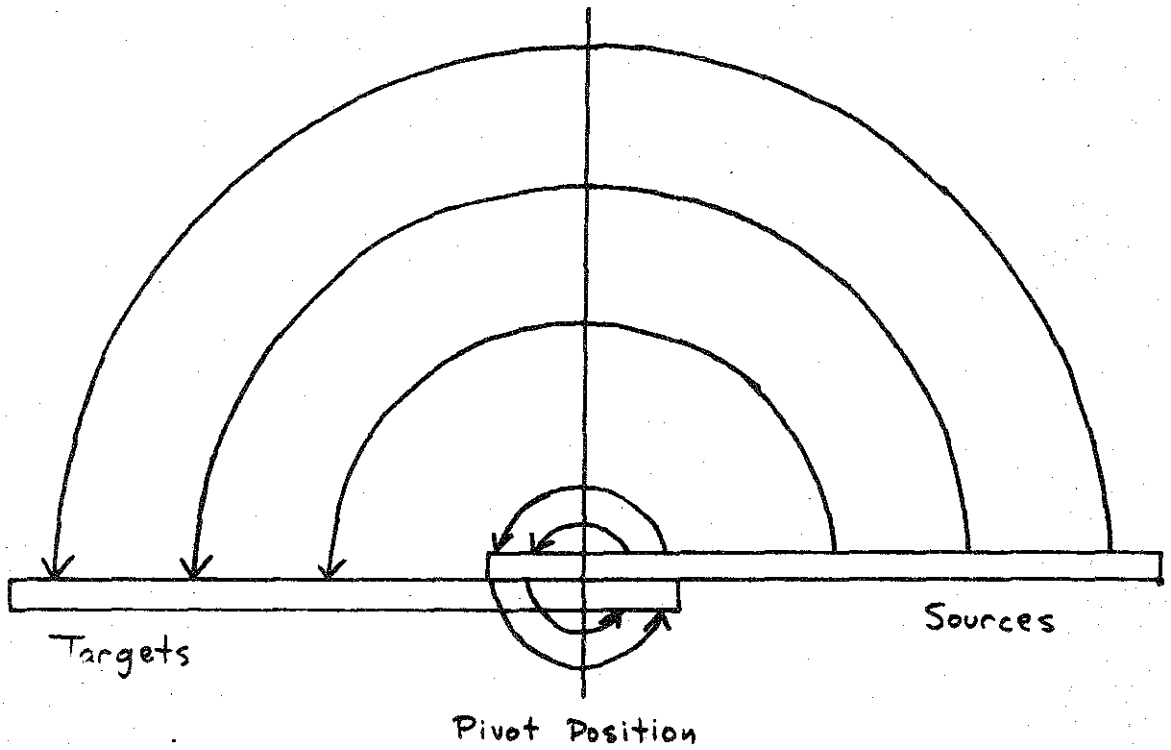


Figure 6.15


Targets
Sources
Pivot Position

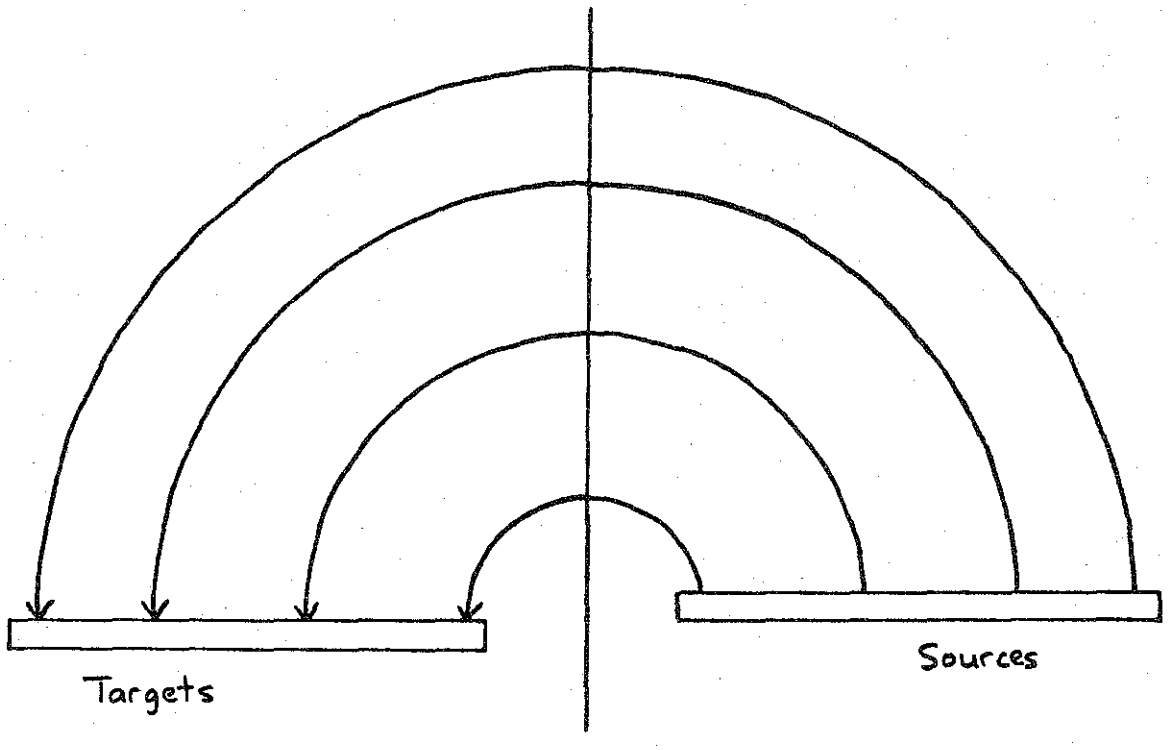

Targets
Sources
Pivot Position

Figure 6.16

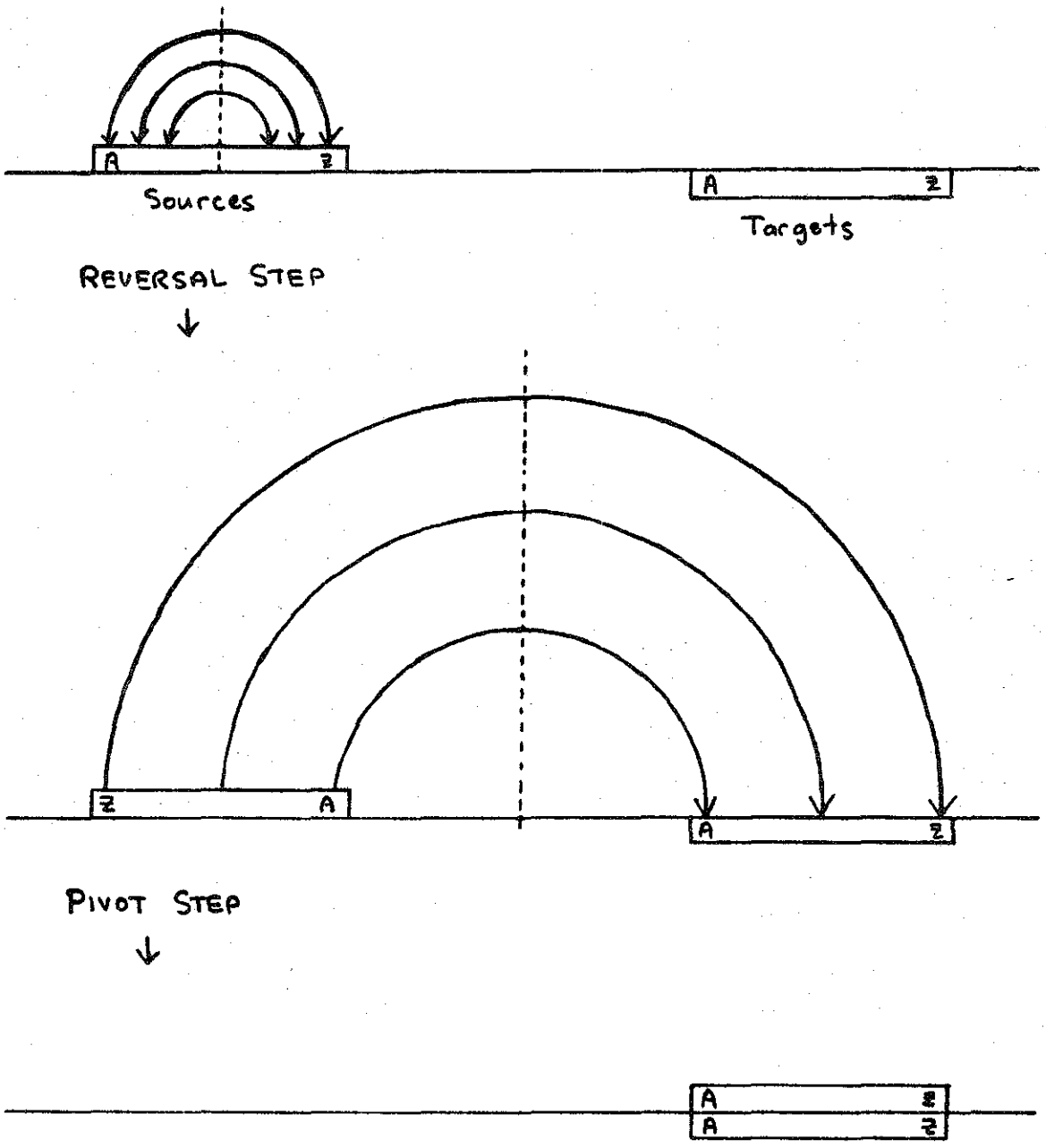


Figure 6.17

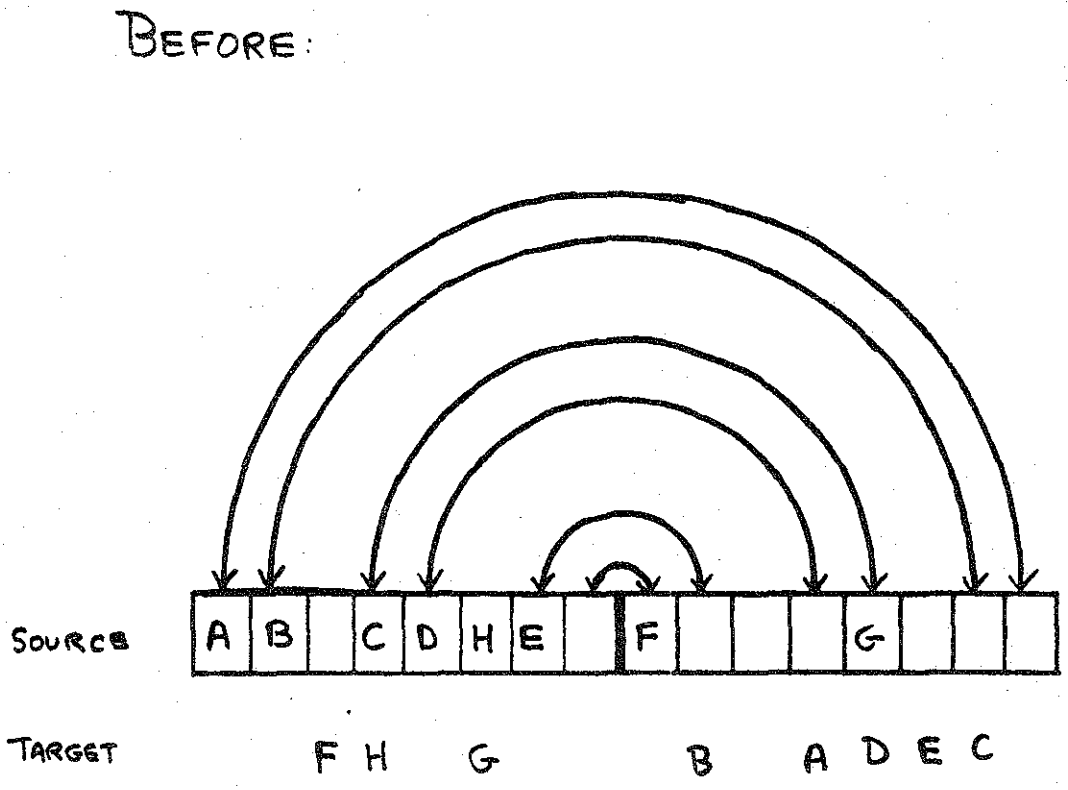
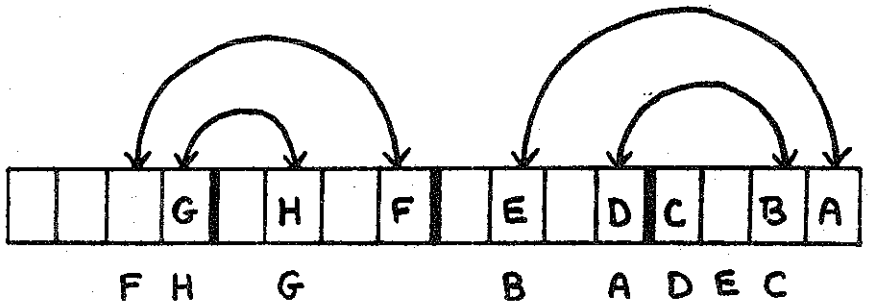

BEFORE:
SOURCE
A B C D H E F G
TARGET
F H G B A D E C
AFTER:
G H F E D C B A
F H G B A D E C

Figure 6.18

BEFORE:



AFTER:

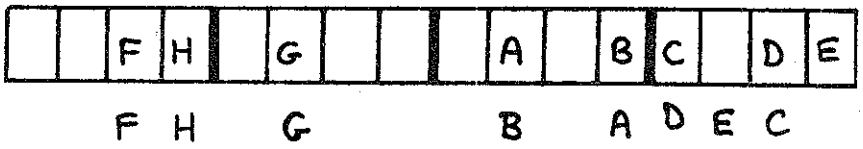


Figure 6.19

7. SUMMARY AND CONCLUSIONS

7.1 Summary

In this dissertation, we have considered the problem of data movement in a network of cells arranged to form a complete binary tree with connections between horizontally adjacent cells. Specifically, we have studied how the horizontal connections might be used in routing data among the leaf cells. This problem arises in the design of a machine to implement reduction languages, as described in [1].

The first set of algorithms developed (SS1-SS4, Chapter 3) show how to broadcast copies of a single data item from one of the leaf cells to the rest of the leaf cells. Using the theorems of Chapter 2, these algorithms are shown to use a shortest path or a path which is no more than two steps longer than a shortest path to route an item from one cell in L to another. When these algorithms are extended to handle several items simultaneously (Algorithms MS1-MS4), many unnecessary copies of the items are created.

Chapter 4 presents techniques which reduce the number of unnecessary copies being generated. This involves storing information in the cells of T so that when a copy of a data item arrives at a particular cell, the item may use the stored information to decide where it should be routed. In some cases (as discussed in Section 4.5), these techniques guarantee that no extra copies of data items are generated. In such cases, we say that the cells contain Complete Path Information. Algorithms which are based on the movement patterns of SS1-SS4 and which use complete path information are referred to as CP1-CP4.

The CP algorithms are analyzed briefly in Chapter 5. The major result is that the CP algorithms require $\Theta(n)$ steps to complete many interesting movement patterns involving n data items. The data movement algorithm presented in [1], which does not use the horizontal conncetions, also requires $\Theta(n)$ steps for all movement patterns involving n data items. Therefore, the algorithms of Chapter 4 represent only a linear improvement over the algorithm which does not use horizontal connections.

In Chapter 6, we establish the fact that some patterns can be completed in $\Theta(n/\log(n))$ steps. The applicability of this result is limited since, except for the problem of reversing the contents of n adjacent cells, we are not able to show how to compute the information which must be stored

in the cells of T to guide data movement.

7.2 Suggestions for Further Work

Since the machine of [1] is still in the paper design stage, and since reduction languages and their derivatives are not in use for programming, this research has necessarily had a theoretical orientation.

If reduction language (or functional) programming becomes more widespread, it will be possible to ask such questions as what data movement problems appear most often in reduction language programs. These patterns should then be analyzed more precisely than those of Chapter 5, where only order-of-magnitude bounds were established.

Also, it may be possible to design specific algorithms for these common problems, as was done for the problem of reversal in Chapter 6. Incorporating such special-purpose algorithms into the machine of [1] may be difficult. However, the potential benefits are great since a large portion of the execution time for reduction language programs is spent in data movement.

REFERENCES

[1] G. Magó, "A Network of Microprocessors to Execute Reduction Languages", to be published in the International Journal of Computer and Information Sciences.

[2] J.W. Backus, "Programming Language Semantics and Closed Applicative Languages", in Conference Record of ACM Symposium on Principles of Programming Languages, Boston, Mass., 1973, pp. 71-86.

[3] Mark Pozefsky, "Programming in Reduction Languages", Ph.D. Dissertation, Department of Computer Science, University of North Carolina, Chapel Hill, N.C. 1977.

[4] Alexis Koster, "Execution Time and Storage Requirements of Reduction Language Programs on a Reduction Machine", Ph.D. Dissertation, Department of Computer Science, University of North Carolina, Chapel Hill, N.C. 1977.

[5] S.S. Patil and J.B. Dennis, "The Description and Realization of Digital Systems", Revue Francaise d'Automatique, d'Informatique et de Recherche Operationelle, February, 1973, pp. 56-69.

[6] K.E. Sahin, "Intermodular Communication without Addressing in Planar Arrays of Modules Connected with One-Way Channels", Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, Mass. 1969.

[7] Kenneth J. Thurber, "Interconnection Networks--A Survey and Assessment", AFIPS Conference Proceedings, vol. 43, 1974, pp. 909-919.

[8] C.J. Chen and A.A. Frank, "On Programmable Parallel Data Routing Networks via Crossbar Switches for Multiple Element Computer Architectures", in Proceedings of the 1974 Sagamore Computer Conference on Parallel Processing, Springer-Verlag, 1975.

[9] Tse-yan Feng, "A Versatile Data Manipulator", in Proceedings of the 1973 Sagamore Computer Conference on Parallel Processing, Department of Electrical and Computer Engineering, Syracuse University, 1973, p. 101.

[10] Samuel E. Orcutt, "Implementation of Permutation Functions in Illiac IV-Type Computers", IEEE Transactions on Computers, vol. C-25, no. 9, September, 1976, pp. 929-936.

[11] C.H. Sequin, A.M. Despain and D.A. Patterson, "Communication in X-tree, a Modular Multiprocessor System", in Proceedings of the 1978 Annual Conference, Association for Computing Machinery, 1978, vol. 1, pp. 194-203.

[12] Donald Knuth, "Big Omicron and Big Omega and Big Theta", in SIGACT News, vol. 8, no. 2, 1976, pp. 18-24.

[13] D. Tolle, private communication.