An Algorithm for Reconstructing Protein and RNA Sequences

MARVIN B. SHAPIRO

National Institutes of Health, Bethesda, Maryland*

ABSTRACT. An algorithm for deriving the primary sequence of a protein or RNA is presented. The data is in the form of short sequences of letters which must be fitted together to form the unknown complete sequence. A computer program for carrying out the steps is described, with an example. It is shown that the algorithm cannot make an error and empirical results are given which illustrate the successful use of the algorithm in reconstructing complete sequences known to be solvable.

Introduction

Proteins and ribonucleic acids (RNA) are complex molecules of primary importance in the living process. The determination of their structure is a main goal of present-day molecular biology [1]. Both proteins and RNA are made up of elementary subunits linked in a chain and coiled in a very complicated three-dimensional structure. While the ultimate goal of the researcher is the elucidation of this spatial structure, a first and major step along the way to this goal is the determination of the linear structure of the chain of subunits. This is called the primary structure of a protein or RNA and its solution requires knowledge of what particular type of subunit is present at each position along the entire chain.

For RNA the basic types of subunits are the four nucleotides, adenylic acid (A), cytidylic acid (C), guanylic acid (G), and uridylic acid (U). Thus, an example of an RNA sequence containing seven subunits would be

GCCAGUA.

The protein subunits are called amino acids, of which there are twenty. Since there are more amino acid than RNA subunits and since the amino acids are usually abbreviated with a more complicated notation, the simpler RNA notation is used in the remainder of this paper, although the logical problem of reconstructing either type of sequence is the same.

At present, there is no experiment that can be carried out to directly determine the primary structure. The basic chemical procedure used for determining a sequence involves a fragmentation stratagem. Chemical methods are used to break a sample of approximately 10^{17} identical molecules into parts or fragments which can overlap. These fragments are themselves short sequences coming from different parts of the long sequence to be determined. Thus, each of these short sequences represents approximately 10^{17} identical RNA subsequences. Identical fragments which occur at more than one place, say m, in the molecule are recognized by $m \times 10^{17}$ identical fragments being recovered and analyzed.

The reconstruction problem is to link together enough of these smaller pieces or

* Division of Computer Research and Technology

fragments of the puzzle until the entire linear sequence of the molecule has been uniquely determined. The laboratory procedures employed involve the use of enzymes (chemical catalysts) which split or break the molecule either at certain prescribed places only, or in a random pattern. For example, one such enzyme will cleave an RNA sequence selectively after G only, and if allowed to interact with the RNA molecule for a long enough time, will cleave after every G linkage. Thus, the sample sequence given above would, upon reaction with this enzyme for a long enough time, be split into the following three fragments:

G
CCAG
UA.

Another enzyme cleaves selectively after both C and U. Thus use of two or more enzymes will then produce fragments which overlap or represent the same part of the original sequence. Other enzymes cleave an RNA at random places. Three possible fragments which could be recovered using such an enzyme on the above seven letter sequence are

CC
A
CAGU.

A random splitting enzyme does not split all molecules in the same place, so the fragments produced may overlap. Thus the second and third C's in the above three fragments are the same C.

In general the three basic pieces of laboratory information available for solving the logical problem of sequence reconstruction are:

1. The composition of the sequence or number of occurrences of each type of subunit (which gives the overall length).
2. The characteristics of the enzymes used to obtain the fragment data.
3. The fragment data itself.

Suppose the set of letters after which a particular enzyme cleaves is called a break set, Λ. For the sequence just illustrated, with the two sets of fragments given, we have one break set (consisting of G) since the random-splitting enzyme does not define a break set. The basic information available is then

1. The sequence contains seven letters consisting of 2 A's, 2 C's, 2 G's, and 1 U.
2. There is one break set, Λ_1, which cleaves after G. (Λ_1 = G.)
3. The six fragments obtained by both methods were:

1. 1 G
2. 1 CCAG
3. 1 UA
4. 0 CC
5. 0 A
6. 0 CAGU.

Here the (nonzero) number preceding the fragment indicates the number of the break set. 0 indicates that a random method of cleavage was used. Note that the 1 gives the information that a G is to the left of the fragment.

In the algorithm discussed in this paper it is assumed that the sequence of letters in the individual fragments is known. While this is normally true for RNA fragments up to about length eight, it is not generally true for longer RNA fragments and for most protein fragments. An extension of the algorithm to handle these unsequenced or "composition only" type fragments is now being developed.

The Algorithm

The reconstruction involves finding letters, in different fragments, which must be the same and using this information to combine or overlap the two fragments into one longer fragment. Thus, in the simplest case, where a particular type of letter occurs only once in the molecule, and where it is present in two fragments, the two fragments must overlap. For example, given the two fragments

0 CAGU
0 GUA

from the sequence above, since there is only one U in the overall composition the two U's here must in fact be the same and the two fragments must overlap to form

0 CAGUA.

Analysis of the G's would not have determined this overlap since there are two G's present in the overall composition and these two fragments alone do not provide sufficient information to determine that their G's are the same.

This basic overlapping logic, which is easily seen in the case of singly occurring letters, has been used in most sequencing work [2]. Earlier [3] a set of reconstruction rules was presented. The algorithm given here represents an extension of that logic to handle all sequenced data and at the same time a simplification in that it reduces the procedure to a series of basic steps. Also, it is now possible to show that the reconstruction procedure does not make any errors. This is defined and described later.

The algorithm proceeds by a simple procedure of attaching unique indexes to letters which, by analyzing their neighboring letters, must be different from any other previously indexed letter of the same type. That is, letters (of the same type) which have been assigned different indexes must occur in different places in the sequence. For the two fragments above, indexes would be assigned as follows:

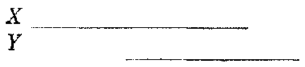
```
0 C(1)  A(1)  G(1)  U(1)
0 G     U(1)  A(2)
```

producing the single fragment

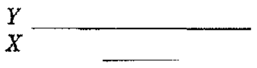
0 C(1) A(1) G(1) U(1) A(2).

If as many different indexes are established as the number of occurrences of that letter type in the overall sequence (this is called maximum indexing), then any subsequent fragment in which a letter of that type occurs must either (1) overlap a previous fragment; (2) incorporate a previous smaller fragment; (3) be incorporated into a previous larger fragment; or (4) represent an error in the composition data for that letter or in the fragment data. Overlapping of two fragments X and Y

is illustrated as



and incorporation of X into Y is



If the overlap or incorporation can be unambiguously established, then it is carried out.

If maximum indexing for a letter has not occurred, then incorporation or overlapping of a fragment containing a letter of that type subsequently encountered is only possible if there are fragments "equivalent" to it in the data. Two fragments are called equivalent if they are the same length, if they have the same sequence of letters and break set, if indexes occur at the same letter positions and if they are known to be from different places in the molecule. For example, given the indexed fragment

1 G(1) C(1)

(where the 1 at the left indicates break set 1, $\Lambda_1 = \text{C}$), if two more fragments

2 CG
2 CG

are encountered ($\Lambda_2 = \text{G}$) and the composition information indicates that there are 2 C's in all, then these two equivalent unindexed C's can be indexed as

2 C(1) G
2 C(2) G.

The reasoning here is that (1) since the two unindexed C's occur in fragments from the same break set they cannot be from the same place in the molecule (i.e., must have different indexes assigned) and (2) since the C's occur in identical looking fragments, one of the C's (it does not matter which) can overlap C(1) and the other can be assigned the missing index. If there had been three C's in all, the overlap with C(1) could not be made.

Analogously, an unindexed letter which matches two indexed letters occurring in equivalent fragments can be arbitrarily assigned the index of either one. For example, given two indexed fragments

2 C(1) G(1)
2 C(2) G(2)

and the information that the molecule contains exactly two C's, the C in the fragment

0 GC

could be indexed either 1 or 2 and would arbitrarily be indexed C(1).

The algorithm is used successively on each unindexed letter within each fragment. When no unindexed letters remain or when a complete cycle through the unindexed letters has occurred with no indexing or overlapping taking place, the procedure is completed and all deducible overlapping has been performed. Figure 1 represents the indexing procedure followed in the reconstruction algorithm. Figure 2 is a flowchart of the "nearest neighbor analysis" of two letters (of the same type) which is used in step 3 of the algorithm to determine whether they can be the same letter or whether they must be different.

An Example

The algorithm can be most easily understood by following through an example.

(*Please turn to page 726*)

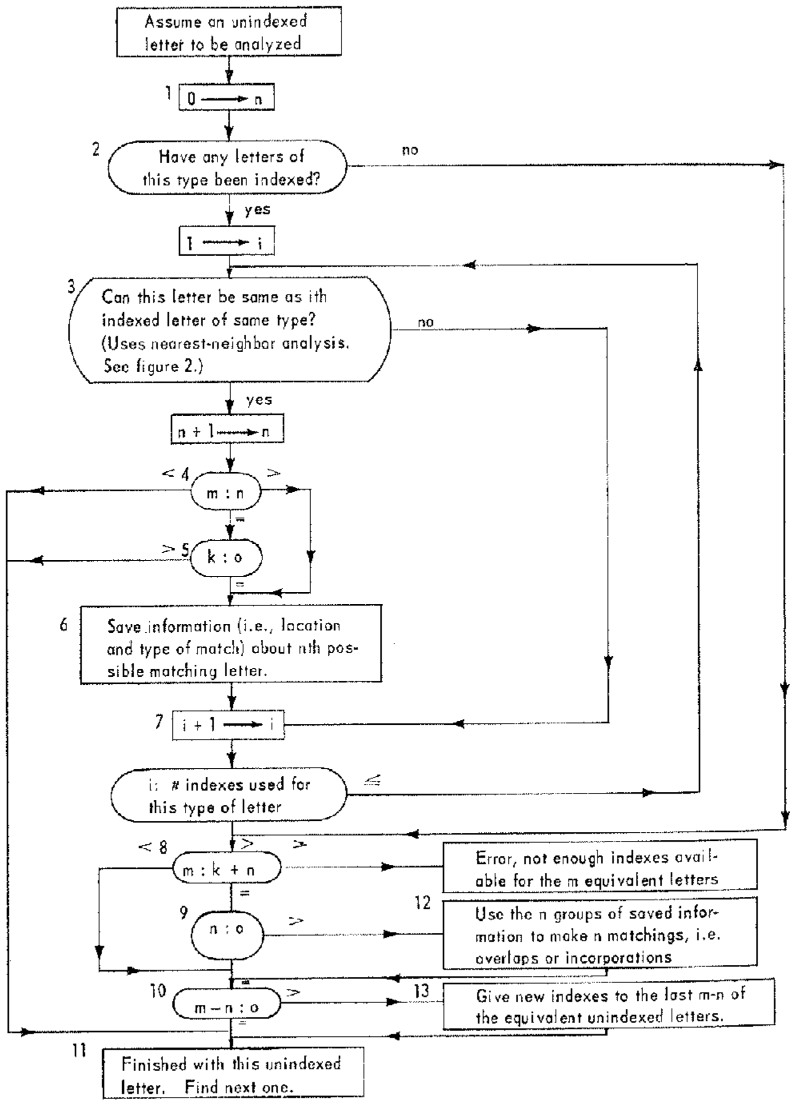

Assume an unindexed letter to be analyzed
1
0 → n
2
Have any letters of this type been indexed?
no
yes
1 → i
3
Can this letter be same as ith indexed letter of same type? (Uses nearest-neighbor analysis. See figure 2.)
no
yes
n + 1 → n
< 4
m : n
>
=
> 5
k : o
=
6
Save information (i.e., location and type of match) about nth possible matching letter.
7
i + 1 → i
i: # indexes used for this type of letter
≤
>
< 8
m : k + n
>
Error, not enough indexes available for the m equivalent letters
=
9
n : o
>
12
Use the n groups of saved information to make n matchings, i.e. overlaps or incorporations
=
10
m − n : o
>
13
Give new indexes to the last m-n of the equivalent unindexed letters.
=
11
Finished with this unindexed letter. Find next one.

FIG. 1. The reconstruction algorithm. n = count of indexed letters which can possibly be the same as the letter to be indexed; k = number of indexes that have not yet been found for this type of letter; m = number of equivalent letters. Two letters are equivalent if they are the same type and occur in the same position in equivalent fragments.

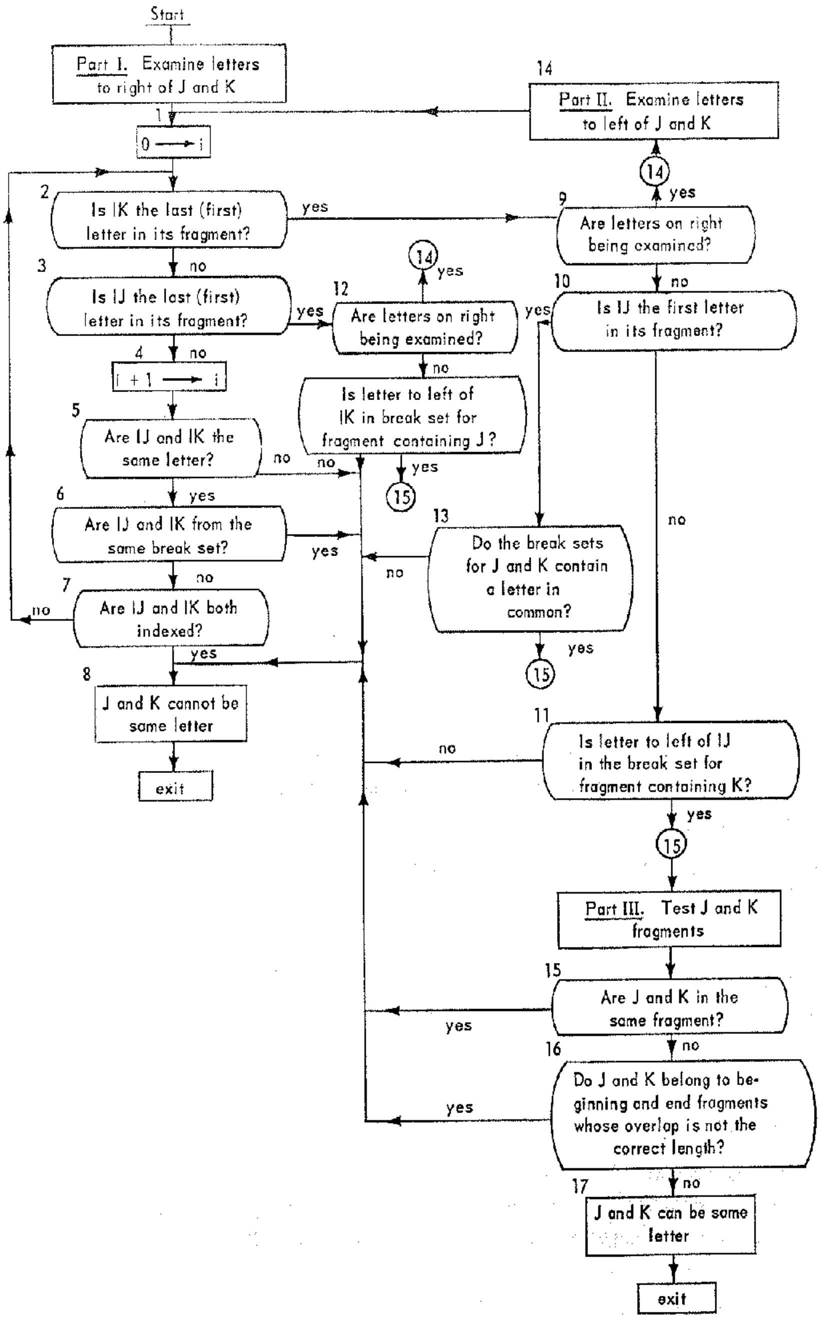

Start
Part I. Examine letters to right of J and K
1
0 → i
2
Is IK the last (first) letter in its fragment?
yes
no
3
Is IJ the last (first) letter in its fragment?
yes
no
4
i + 1 → i
5
Are IJ and IK the same letter?
no
yes
6
Are IJ and IK from the same break set?
yes
no
7
Are IJ and IK both indexed?
no
yes
8
J and K cannot be same letter
exit
14
Part II. Examine letters to left of J and K
9
Are letters on right being examined?
yes
no
10
Is IJ the first letter in its fragment?
yes
no
12
Are letters on right being examined?
yes
no
Is letter to left of IK in break set for fragment containing J?
no
yes
13
Do the break sets for J and K contain a letter in common?
no
yes
11
Is letter to left of IJ in the break set for fragment containing K?
no
yes
15
Part III. Test J and K fragments
15
Are J and K in the same fragment?
yes
no
16
Do J and K belong to beginning and end fragments whose overlap is not the correct length?
yes
no
17
J and K can be same letter
exit

Fig. 2. Nearest neighbor analysis to determine whether unindexed letter J can be the same as a previously indexed letter, K, of the same type. i = number of positions to right (left) of the J and K positions. IJ and IK are the letters in the ith position to the right (left) of letters J and K, respectively.

Let us assume the following input:

1. The sequence to be reconstructed has a composition of 2 A's, 3 C's, 2 G's, and 1 U.
2. The break sets are A_1 = C, U and A_2 = G.
3. The seven fragments obtained were:

   1. 1 BAGC
   2. 1 AGU
   3. 2 CCAG
   4. 0 CA
   5. 1 C
   6. 1 C
   7. 0 UC

NOTE. The letter B in the first fragment does not represent a nucleotide but is a marker which denotes the beginning fragment from a sequence. In the computer implementation of the algorithm the B is treated like an extra letter type but it is never indexed.

The algorithm proceeds to look at each of the letters in the order of input. Each fragment below is shown after the algorithm is used on each of its letters.

Fragment 1. 1 B A(1) G(1) C(1)

This is the first occurrence of an A, G, or C, so that a branch to step 8 is made in step 2 of the algorithm. $n = 0$, $m = 1$, $k > 0$.

Fragment 2. 1 A(2) G(2) U(1)

The A and G were indexed because, using "nearest neighbor analysis," they are not the same as A(1) and G(1), respectively. In step 8 of the algorithm $m = 1$, $n = 0$, and $k = 1$ for both the A and G. The U is indexed because it is the first U encountered.

Fragment 3. 2 C C(2) A(2) G(2)

The first C is unindexed because it can be C(1), in step 4 $m = 1$, $n = 1$ and, with one C index missing, $k = 1$. C(2) was indexed because of a different left-hand neighbor from C(1). The AG overlap with fragment 2 was detected because $m = 1$, $n = 1$, and $k = 0$ (all A indexes have been assigned) in step 8. Thus, at the point where A was assigned the index 2, fragments 2 and 3 were combined to give

2 C C(2) A(2) G(2) U(1).

Fragment 4. 0 C A(2)

The C was unindexed for the same reason that the first C in fragment 3 was not. Then the A was matched uniquely with A(2) ($m = 1$, $n = 1$, $k = 0$ in step 8) and fragment 4 was incorporated into the new overlapped fragment and thrown away.

Fragments 5 and 6. 1 C(2)
1 C(3)

Since there are two equivalent fragments, $m = 2$. There is one possible matching indexed letter, C(2), so that $n = 1$, $k = 1$ (one missing C index). In step 8, $m = k + n$. In step 9, $n = 1$, resulting in an incorporation of fragment 5 into

C(2) in the overlapped fragment, and in step 10, $m - n = 1$, resulting in assignment of a new index to give C(3).

Fragment 7. 0 U(1) C(3)

The U overlaps U(1) ($m = 1$, $n = 1$, $k = 0$) and C(3) is incorporated into the C.

At the completion of pass 1, the original set of fragments is represented as

1. 1 B A(1) G(1) C(1)
2. 2 C C(2) A(2) G(2) U(1) C(3).

The C in fragment 2 is the only unindexed letter left to analyze in pass 2. It can match C(1) ($n = 1$) and, since $m = 1$ and $k = 0$ in step 8, the overlap is necessary. The final result at the end of pass 2 is then

1. 1 B A(1) G(1) C(1) C(2) A(2) G(2) U(1) C(3).

The Computer Program

The reconstruction algorithm has been programmed for a digital computer.[1] The program accepts as input the composition, break set, and fragment information. It then passes through the fragment data as many times as necessary until a complete pass is made without any letter being indexed or without any overlapping of fragments, at which point a final set of fragments, with all possible overlaps made and all incorporated fragments having been eliminated, is output. Figure 3 is a flowchart of the main or control program.

Structurally there is one basic list of letter information in the program. It contains pointers to connect and reference fragment letters. Overlapping of fragments is done by changing forward and backward pointers for the two letters involved, so that letters or fragments are never moved. The running time on a 360/50 computer for a sequence of length 240 with input of 960 fragments is about two minutes. The time increases more than linearly with sequence length for the cases that have been tried.

The Reduced Set and Isotomers

The set of fragments remaining as the final output is called the reduced set. It should represent the smallest set of fragments containing the same information as the set of input fragments contained. If the reduced set consists of one fragment which is as long as the entire sequence (as in the example given) then this represents the solution to the reconstruction problem. When the reduced set contains two or more fragments, more than one solution exists (this is an empirical result and is discussed later); that is, more than one sequence exists which could have produced the set of input fragments, using the break sets defined. For example, the two sequences

BACGUCCGACG

and

BACGACGUCCG

[1] The program was written in the FORTRAN II language for the Honeywell 800 computer and has been converted to FORTRAN IV for the IBM 360.

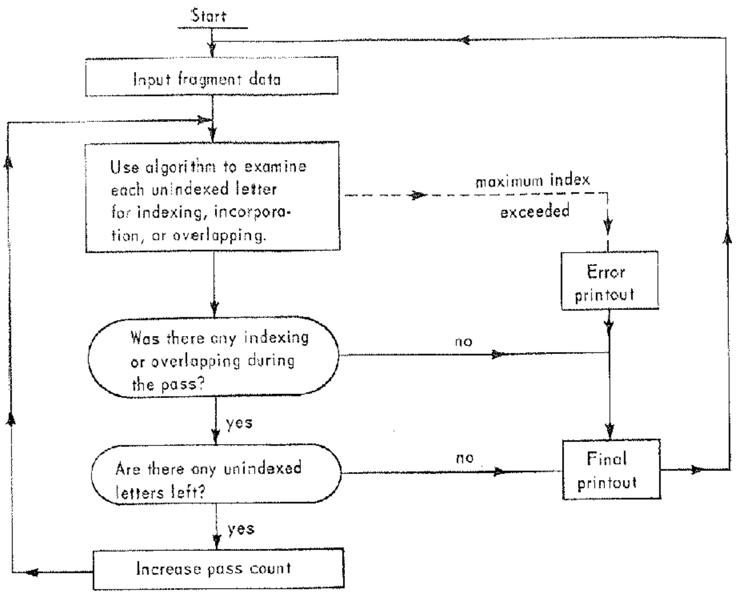

Start
Input fragment data
Use algorithm to examine each unindexed letter for indexing, incorporation, or overlapping.
maximum index exceeded
Error printout
Was there any indexing or overlapping during the pass?
no
yes
Are there any unindexed letters left?
no
Final printout
yes
Increase pass count

FIG. 3. Flowchart of the control program.

would, using break sets $A_1 = C$ and $A_2 = G$, produce the identical set of input fragments

1 BAC	2 BACG
1 GUC	2 UCG
1 GAC	2 ACG.
1 G,	

Given these fragments as input data the computer produced as the reduced set the following output, containing four fragments which can be put together in two ways:

```
Final Output. 2 Passes
                                      A  C  G  U
Number in polymer                     2  3  3  1
Number indexed                        2  3  3  1
                            1 G(1) U(1) C(2) G
                            1 G(2) A(2) C(3) G
                            1 G(3)
                            2 B    A(1) C(1) G
```

These alternative sequences which produce the same set of fragments under given break set conditions are called isotomers [4]. The theory underlying the existence of isotomers has been developed and with some concepts from graph theory, a procedure for computing the number of isotomers, given a sequence and a collection of break sets, has been devised and a computer program written. See [5, 6].

By making trial overlaps and recombining the fragments in the reduced set in all possible ways and by inserting missing letters in all possible ways when indexes are missing, all isotomers can be reconstructed. The number of isotomers becomes very large as the size of the reduced set and/or the number of missing indexes increase. A program has been written [7] which accepts as input the reduced set (plus other information) and produces all isotomers (within given time and output limitations).

Validity of the Algorithm

There are two fundamental questions to be asked about the algorithm. First, does it alter the set of isotomers associated with the original set of input data, and second, does it reconstruct a complete sequence whenever the set of isotomers consists of only one sequence? The first question is much easier to answer. It requires that it be shown that the algorithm does not alter the information content of the original set of data by creating new isotomers or eliminating some. To answer the second question one must show that whenever there is only one isotomer associated with the input data, the algorithm will find that isotomer.

If essential information is lost in any step (for example, by incorrect fragment incorporation), this would increase the number of ways of reconstructing the fragment data. Correspondingly, the isotomer set would be reduced (perhaps to zero) if, at any stage, new information was created by incorrect overlapping of fragments.

EQUIVALENT FRAGMENTS AND LETTERS. Since the notion of equivalent fragments is crucial to the discussion that follows it is necessary to have a precise description of it. When the fragments are collected in the laboratory all those with the same sequence and length are collected together and the actual number of fragments collected is found by measuring the total amount. If these fragments are produced by a nonrandom break set they are called equivalent fragments and the corresponding letters in them are equivalent letters. If an amount m of equivalent fragments is collected, this indicates that this sequence of letters occurred at m different places in the molecule. Assignment of a letter to a place in the molecule is accomplished by indexing, with different indexes indicating different places. Each of these m fragments can be assigned to any of m different places in the molecule; it doesn't matter which one, just so no two of the m are assigned the same place.

IMPOSSIBILITY OF ALTERING THE SET OF ISOTOMERS. First it is shown that the algorithm does not make indexing errors. There are two kinds of indexing errors which it is possible to make. It is necessary to show that (A) if two letters (of the same type) are not from the same place in the molecule the algorithm will not assign them the same index, and (B) if two letters are from the same place in the molecule they are not assigned different indexes. Referring to Figure 1, it is necessary to show that in case (A) box 12 cannot be reached and in case (B) box 13 cannot be reached. The quantities m, n, and k (defined previously) which are used in the flowchart are found by straightforward counting and matching operations and they are assumed to be correct. Nearest neighbor analysis, which is used in box 3, is discussed later.

To show that (A) is true assume that a letter G, which occurs in m equivalent fragments, is being analyzed and is from a different part of the molecule than a previously indexed letter G(i). Since G and G(i) are assumed to be different, nearest neighbor analysis can decide one of two things—that they are different letters (for one of the three reasons given later) or that they could be the same. If it decides that they are different, then, if box 12 is ever reached, G(i) would not be given the same index as G because it was never considered as a possible match. In the other case nearest neighbor analysis would decide that, because G(i) and G happen to have the same neighboring letters (and fulfill the other two conditions described later), they can be the same letter. Since G and G(i) are in fact different, there must be at least m other G's which could match the current one and these m are divided into two groups, containing n already indexed and k not yet indexed. Then the total number of possible matching letters to G would be at least $k + n + 1 > m$. Thus, in Figure 1, the left branch of box 8 would be taken, box 12 would be bypassed, and the current G would not be given the same index as G(i), which we assumed came from a different place in the molecule.

For case (B) assume that the letter G being analyzed came from the same place in the molecule as a previously indexed G(i). That is, one of the m equivalent G's must be G(i). Conversely, there must be $m - 1$ places other than G(i) that the current G's can be assigned to. It was shown in (A) that no two of the m current G's can be assigned the same one of these m locations. For (B) to be true either the current G must remain unindexed or it must be given the same index as G(i), that is, all m equivalent G's must not be given a different index than G(i). Consider three cases:

(a) $m = n$. The current G is left unindexed because if $k > 0$ the left branch of box 5 is taken and if $k = 0$, $m = n$ in box 10.

(b) $m > n$. New indexes are given to the last $m - n$ of the m equivalent G's (in box 13), but either one of the first n is given the same index as G(i) (if $k = m - n$), or all n are left unindexed (if $k > m - n$).

(c) $m < n$. The G remains unindexed because the left branch of box 4 is taken.

Thus, either the m equivalent current letters are left unindexed or if some of them ($m - n$) are given indexes different than G(i) then of the n remaining either all are left unindexed or one of them is assigned the same index as G(i). In no case are all m given different indexes than G(i), which was assumed to be from the same place in the molecule as one of the equivalent G's.

The nearest neighbor analysis flowchart in Figure 2 represents a straightforward test of the three conditions for deciding whether a given unindexed letter and a previously indexed letter can be the same. If the two letters (1) had at least one different neighboring letter, or (2) appeared in the same fragment, or if (3) the two letters or their neighbors occurred in fragments produced by the same break set, then the two letters could not be the same. Otherwise they could be the same letter and are considered as possibilities for the same index assignment (in Figure 1, n is increased by 1). The test for condition (3) requires that break set information for each letter be carried and updated, so that it can always be determined when examining a letter which break set(s) it has belonged to. Thus, for example, when letters in two different fragments are overlapped, the overlapping procedure, in addition to connecting the two fragments, updates the break set information of all overlapping letters to reflect that they occurred in two break sets.

The overlapping or incorporation of fragments which results from the correct assignment of matching indexes does not alter the original set of isotomers because the assignment is made only when the two letters involved logically must be the same (as opposed to possibly being the same). Since it is shown that either no indexing or only correct indexing can occur, the set of isotomers is not altered.

THE SECOND QUESTION. By generating random sequences and using the theoretical results [5] to determine whether isotomers exist, it is possible to know which sequences can and which cannot be solved, for given break sets. Using this approach the second fundamental question about the algorithm can be answered empirically. A great number of cases have been tested in this manner with a variety of sequence lengths, compositions, and break set conditions. For all of them the algorithm produced a reduced set of one fragment, the complete sequence, when there were in fact no other isotomers. Thus the second question is answered empirically.

The algorithm is conservative in its decision making in that no indexing step is taken unless it is logically necessary. Nevertheless, the computer results indicate that this logic is sufficient to reconstruct a complete sequence when there is a unique solution to the problem.

ACKNOWLEDGMENT. The author wishes to thank J. E. Vinton and Dr. R. J. Connor of the Division of Computer Research and Technology, National Institutes of Health, for many helpful discussions and suggestions in connection with this work.

REFERENCES

1. STEINER, R. F., AND EDELHOCH, H. *Molecules and Life.* Van Nostrand, Princeton, N. J., 1965.
2. HOLLEY, R. W., ET AL. Structure of a ribonucleic acid. *Science 147* (1965), 1462–1465.
3. BRADLEY, D. F., MERRIL, C. R., AND SHAPIRO, M. B. Reconstruction of protein and nucleic acid sequences. I. Systematics and computer program. *Biopolymers 2* (1964), 415–444.
4. MERRIL, C. R., MOSIMANN, J. E., BRADLEY, D. F., AND SHAPIRO, M. B. Reconstruction of protein and nucleic acid sequences. II. Isotomers. *Biochem. and Biophys. Res. Commun. 19* (1965), 255–260.
5. MOSIMANN, J. E., SHAPIRO, M. B., MERRIL, C. R., BRADLEY, D. F., AND VINTON, J. E. Reconstruction of protein and nucleic acid sequences. IV. The algebra of free monoids and the fragmentation stratagem. *Bull. Math. Biophys. 28* (1966), 235–260.
6. VINTON, J. E., AND MOSIMANN, J. E. Paper to be published.
7. ——. Computer program writeup. (Unpublished.)

RECEIVED AUGUST, 1966; REVISED DECEMBER, 1966