Ladders

A Research Paper by David Senft

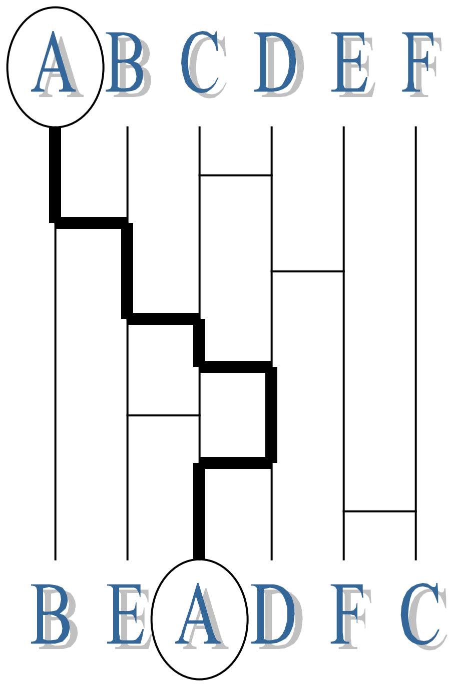


Table of Contents:

Paper Objectives:

You have already been given a basic concept of the function of a ladder, though you do not, as of yet, really know anything about the way ladders work. This paper will attempt to:

Explain ladders and how to read them
Explain why ladders can be used to define any arbitrary permutation
Examine underlying group theoretical concepts working behind the scenes
Produce Algorithms for generating minimally efficient solutions to permutations
Prove the efficiency of the proposed algorithms
Demonstrate and prove important properties of ladders and rung structures
Discuss what will be addressed in future papers

Section One: What is a Ladder?

A ladder can be expressed in two ways: a simple way, and a complex way. The simple definition should give you a good enough impression of the idea to get through this paper, while the second definition will provide a more rigorous connection to the underlying group theoretical concepts.

The Simple Definition of a Ladder:

A ladder is a group of vertical columns and horizontal bars ("rungs") between adjacent columns that collectively represent a permutation from an ordered arrangement above the ladder (the "top row") to an ordered arrangement below the ladder (the "bottom row" or "sequence").

Some Background Information from Group Theory:

A *permutation* is a rule or set of rules that rearrange a set of objects, such as the permutation that changes ABC to BCA. In ladder notation, the rungs that make up the ladder are the rules for the permutation of the top row to the bottom row. Permutations can be defined with "cycles" and "transpositions".

A *cycle* can be represented as a subset of objects in an arrangement $e_1, e_2, \ldots, e_n$ such that, after the cycle is performed, e_1 moves to e_2, e_2 moves to e_3, $\ldots e_{n-1}$ moves to e_n, and e_n moves to e_1. This cycle is said to have period n.

A *transposition* is a cycle of period 2; i.e., a cycle that switches the places of two objects.

Elementary group theory gives the following:

The permutations of a set of objects form a symmetric group.
A permutation can be expressed as a product of cycles.
A cycle can be expressed as a product of transpositions.
Therefore, a permutation can be expressed as a product of transpositions.

Adjacent Transpositions:

It will be shown in this paper that ladders are a way of representing arbitrary permutations. We will show that every rung in a ladder represents a transposition of two adjacent objects in an ordered sequence. Therefore, any permutation can be expressed as the product of "adjacent transpositions", or *AT*s. This gives our second definition of Ladders:

The Group Theoretical Definition of Ladders:

A ladder is a graphical means of depicting the adjacent transpositions whose product is a given permutation.

Section Two: Introduction to Ladders, Reading a Rung Structure

Terminology:

In each ladder diagram, there is a top row of letters, a bottom row of letters, vertical columns, and horizontal bars, as shown:

```
A  B  C  D  E          ← top row  (objects: A,B,C,D,E)
|---|   |   |   |
|   |   |---|   |
|   |---|   |   |      ← columns and bars
|   |   |   |---|
|   |   |---|   |
B  D  E  A  C          ← sequence
```

Figure 1.1

Each letter, representing a starting point at the top and an ending point at the bottom, is an ***object***. The top row of letters, almost always* in alphabetical order, is simply referred to as the ***top row***, and represents the starting points of each object. The bottom row of letters, which changes from one ladder to the next, is called the ladder's ***sequence***, and represents the ending point for each objects path from the top row. (When not using ladder notation, the terms *sequence* and *arrangement* will be used interchangeably as meaning any ordered group of objects.) There can be any number of objects in a sequence, as long as there is the same number of objects in the top row. The ***size*** of a ladder is the number of objects it has, and thus the number of columns. The horizontal lines will be called ***rungs***, for the sake of maintaining a comparison to ladders, and tell how to convert the top row of objects into the sequence below. The ***rung structure*** of a ladder is the positioning of all the rungs as a whole. The area between columns is a ***gap***, and so a ladder of size n has $n - 1$ gaps. The ladder shown above has a size of 5, and thus has 5 objects, 5 columns, and 4 gaps. When two objects *cross paths*, their paths overlap with each other. Because paths can only overlap on rungs, crossing paths can also be termed *sharing a rung*. When two objects *change sides* their order reverses from top row to sequence. For example, if object A starts to the left of object C in the top row, and moves to the right of object C in the sequence, no matter what specific positions they might be in, A and C are said to change sides. Note that if two objects cross paths twice, they have not changed sides.

*Whenever ladders are being compared to other ladders or to their equivalent permutations, the top row will always be alphabetical. Only if a ladder is being considered in isolation should its top row be allowed to be a randomized sequence, and even that you will not find in this paper. For all intents and purposes, where this paper is concerned, the top row of a ladder will always be in alphabetical order.

How to Read a Rung Structure:

Reading a ladder's rung structure is easy. Let's look at the following ladder (size 5) as an example.

```
A  B  C  D  E
|---|   |   |   |
|   |---|   |   |
|---|   |   |   |
|   |   |   |---|
|   |   |---|   |
```

Figure 1.2

We will determine the sequence of the above ladder by reading its rung structure. Starting with object A, trace down each object's column until you get to a rung. At a rung, trace across the rung to the other column connected to the rung. Continue tracing downward until you find another rung or the bottom of the column. In the above case, tracing gives the following paths:

```
A  B  C  D  E       A  B  C  D  E       A  B  C  D  E       A  B  C  D  E       A  B  C  D  E
|---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |
|   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |
|---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |
|   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|
|   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |   |   |   |---|   |
         A              B     A         C  B     A         C  B     A  D      C  B  E  A  D
```

Figure 1.3 **Figure 1.4** **Figure 1.5** **Figure 1.6** **Figure 1.7**

This gives a final sequence of CBEAD

Theorem #1 – The Object Location Theorem:

Every position on every column (with the exception of those places where columns and rungs join) in any ladder structure is only used by one and only one object, while every rung is used by exactly two objects.

Proof:

Consider all the objects of a ladder falling such that they are all at the same horizontal level at any given time. (Thus, as objects go across rungs, the other objects wait for them before continuing to fall.) Note that whenever a rung is reached, only the two objects connected to it will switch places, all the while maintaining, after the switch, the same number of objects in each column (1 in each). Because all the objects are at the same level, only one pair of objects can use any one rung, because only two objects can come in contact with a rung at any one given horizontal level. Therefore, there is always just one object in each column at every horizontal level, and every rung is used by two, and only two objects that happen to be in the two columns that that rung connects.

Section Three: Defining Sequences, Why Ladders Work

Fact:

When two objects change sides, they cross paths an odd number of times.

Explanation:

It is easy to see that two objects that cross paths only once will change sides. If, however, they were to cross paths a second time, they would naturally end up on the same side of each other that they were originally on, and would not have changed sides. If they cross paths a third time (odd) they will have changed sides, and if they cross paths four times (even), they won't change sides.

New Notation:

Moving away from the ladder notation for just a moment, the following notation will be necessary in this section:

R(sequence) = {set of all ordered 2-element relations for sequence}

For example:

R(ABC) = {AB, AC, BC}

This means that in the sequence ABC, A comes before B, A comes before C, and B comes before C. The *relation* of A before B is written AB, and so forth. Note that in a sequence of size *n*, there will be ${}_nC_2$ defining relations.

Theorem #2 – The Sequence-Relation Theorem:

Every sequence has one and only one set of defining relations. In other words, a set of relations uniquely defines a sequence, and a sequence uniquely defines a set of relations.

Proof:

For arrangement A with *n* objects, there are ${}_nC_2$ defining relations. Each object in the sequence appears in *n*-1 relations (one relation with each other object in the sequence). In a real* relation set with ${}_nC_2$ relations, one object will appear first in all *n*-1 relations in which it appears (the first object in the sequence will be first in all its relations with other objects). Another object will appear first in *n*-2 relations, and appear second in its relation with the first object only. A third object will appear first in n-3 relations, until finally the last object in the sequence appears in n-n relations. The k^{th} object in the sequence will appear first in *n-k* relations. Because the sum of $(n-1)+(n-2)+\ldots+(n-n) = n^2 - (1+2+3+\ldots+n) = n^2 - n(n+1)/2 = n(n-1)/2 = {}_nC_2$ by definition, the named relations can be the only relations, and distinctly must define a single, unique arrangement of objects.

*Some relation sets do not define real sequences. For example {AB, BC, CA} cannot be expressed as a real sequence. As a result, this set is not a real set.

Fact – Relations Groups:

Given two sequences of the same size, there exists a set of relations that differ between the two sequences and a set of relations that are the same between the two sequences. All relations in either sequence belong to one and only one of these two sets.

Explanation:

A relation can clearly not belong to both the set of relations that differ and the set of relations that are the same, and because every pair of objects must be expressed in each set of relations, each relation can be found in both sets, and must be either different, or the same.

Example:

Taking two sequences of the same size, n, there can be found a set of those relations which the two sequences share in common. Call this the NCR function of the two sequences (NCR standing for Number of Common Relations) such that $0 \leq \text{NCR} \leq {}_nC_2$. Let the NDR function of the two sequences (Number of Different Relations) be equal to ${}_nC_2 - \text{NCR}$. For instance:

S_1 = (ABCD), R(S_1) = {AB, AC, AD, BC, BD, CD}
S_2 = (BDAC), R(S_2) = {BA, AC, DA, BC, BD, DC}
NCR(S_1,S_2) = 3, (Common relations = {AC, BC, BD})
NDR(S_1,S_2) = 3, (Different relations = {AB, AD, CD}*)

*Note that the order of the objects in the set of different relations doesn't matter, because they are different in S_1 and S_2, so they are simply expressed alphabetically (A before B, A before D, C before D).

Theorem #3 – The S.S.S. (Sequence-Set-Sequence) Theorem:

Consider two sequences of size *n* named sequence1 and sequence2, and the set S of all object pairs whose relations in sequence1 differ from their relations in sequence2:

Sequence1 and set S uniquely define sequence2.
Sequence2 and set S uniquely define sequence1.

Proof:

As shown in Theorem 2, a sequence is merely a set of relations. Therefore, by taking a set of n relations (such as those for sequence1), and changing a certain number of those relations (such as the ones whose pairs are found in set S), you will end up with a new set of n relations (defining sequence2), which still define a new sequence of the same size. By taking sequence2's relations and changing them back according to the same set S, you can only end with the same set of relations that were originally derived from, and define sequence1.

Example of application of Theorem #3:
You are given the following sequence (sequence1) and set S of differing relations, and have been asked to find the sequence that they define (sequence2):

Sequence1 = ABCDE
S = {AB, AD, AE, CD, CE, DE}*

*Note that the elements of set S are not relations, just combinations. The order of the objects in the pairs in set S is unimportant, so the pairs are given alphabetically.

First, you find the relations that define Sequence1:

R(Sequence1) = {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}

Then, you determine the relations in R(Sequence1) that are in set S (shown in bold):

R(Sequence1) = {**AB**, AC, **AD**, **AE**, BC, BD, BE, **CD**, **CE**, **DE**}

Then, reverse the order of the relations found in set S:

R(Sequence2) = {**BA**, AC, **AD**, **EA**, BC, BD, BE, **DC**, **EC**, **ED**}

Now, you determine the sequence that these new relations define (the object first in *n*-*k* relations, where ${}_{n}C_{2}$ is the number of relations, is the *k*th object in the sequence).

Sequence2 = BEADC

Now, going backwards:

Sequence2 = BEADC
S = {AB, AD, AE, CD, CE, DE}

R(Sequence2) = {**BA**, AC, **AD**, **EA**, BC, BD, BE, **DC**, **EC**, **ED**}

R(Sequence1) = {**AB**, AC, **AD**, **AE**, BC, BD, BE, **CD**, **CE**, **DE**}

Sequence1 = ABCDE

We arrive back at the same original sequence.

What Theorem 3 tells us About Ladders:

If the relation between two objects changes from one arrangement of objects to another, this is a fancy way of saying that the two involved objects have changed sides. Because a ladder is a representation of objects changing sides with one another, a ladder can also be considered the description of the set of differing relations from the top row to the sequence. Therefore, when the ladder is coupled with the top row, a bottom sequence is uniquely defined.

What the Relations Groups Fact tells us About Ladders:

The "Relations Groups" fact before Theorem 3 states that two different sequences have one and only one set of relations that differ between the two sequences. Therefore, when given a top row and a bottom sequence for a ladder, there is only one possible set of objects that must changes sides with one another in the ladder. Therefore, any rung structure that solves a ladder sequence permutation must include all the side changes that correspond to the differing relations between the two ordered arrangements of objects, and no more.

Section Four: Algorithms for Creating Rung Structure from Sequence

Overview:

This section contains two algorithms for generating a minimal solution to a ladder problem given any sequence. The algorithms are most different from each other in that the first may be considered a “recursive” algorithm, while the second may be considered “explicit.” The recursive algorithm is a piece-by-piece procedure, where each step generates a small part of the solution, and that which has already been generated affects that which is yet to be generated. The explicit algorithm, on the other hand, generates a formula to draw the entire rung structure at once. The recursive algorithm is far simpler, while the explicit algorithm has the obvious advantage of being explicit in the sense explained above. After the algorithms are introduced, it will be proved that they generate minimal solutions. A minimal solution is one that requires the least possible amount of rungs.

Algorithm 1: “the recursive algorithm”

Consider the following unsolved ladder sequence:

```
A B C D E
| | | | |
| | | | |
| | | | |
E B A D C
```

Figure 5.1

The algorithm is designed to secure the paths of each object in the sequence one at a time. In the above sequence, the path of object E will be secured first, then the path of object B, then A, and finally D. There is no reason to say that object C needs to be secured, for securing all the other objects leaves no option but for the last object to be secured.

Securing the first object:

To “secure” an object, it must be given a path to its destination in the sequence and no further rungs can be added which interfere with its path. In our example, object E comes first in the sequence and last in the top row. Rungs must therefore form a path from the last column down to the first, as shown in **Figure 5.2**. **Figure 5.3** shows object E visually separated from the other objects, representing its being secured:

```
A  B  C  D  E
|  |  |  |---|
|  |  |---|  |
|  |---|  |  |
|---|  |  |  |
E  B  A  D  C
```

Figure 5.2

```
             A          B  C  D  E
             |          |  |  |---|
             |          |  |---|  |
             |          |---|  |  |
             |----------|  |  |  |
secured →    E          B  A  D  C
```

Figure 5.3

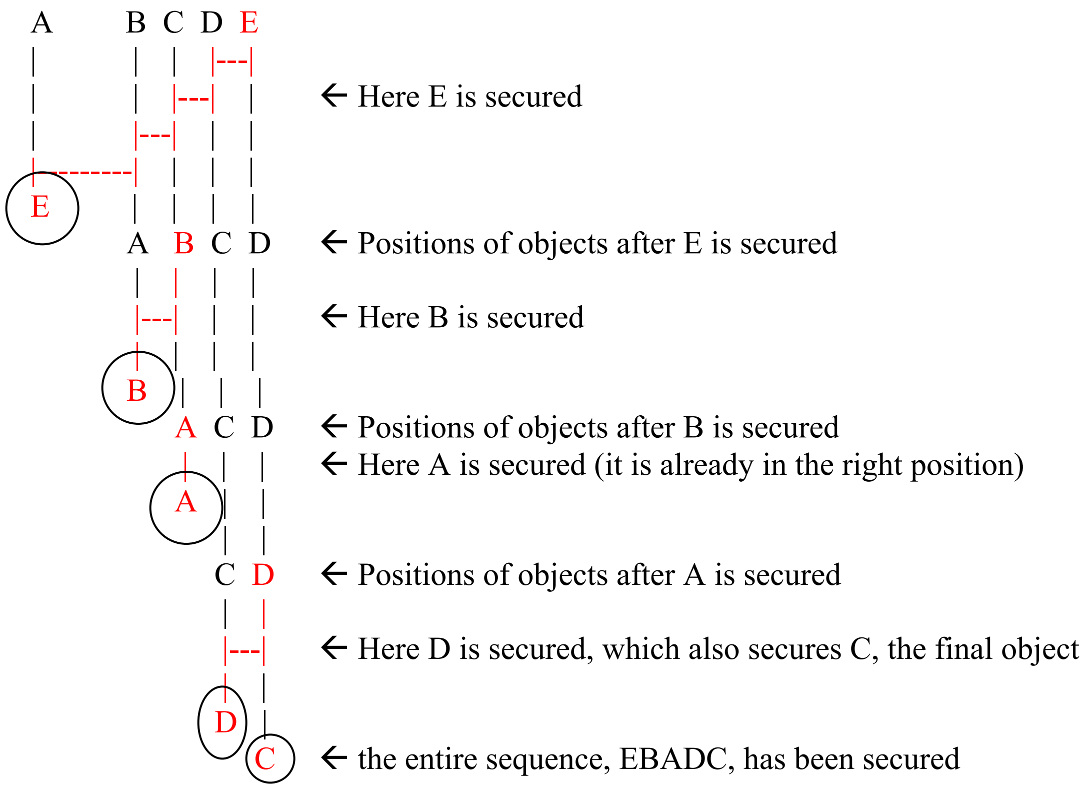
Securing the rest of the objects:
You can see by observing the effects of E's path, that object A now slides into the second column, B into the third, C into the fourth, and D to the fifth. Because object E has been secured, it is no longer a factor in the development of the rest of the solution. We thus essentially have a four columned ladder to work with, the top row and the sequence being the same as in **Figure 5.2**, only without the secured object E. We will, for the meantime, forget object E and show only the other four columns and objects, making B the first object in the unsecured sequence. To continue, we "secure" this first object B (second in the actual sequence). Here is the imaginary size 4 leftover ladder and the securing of its first object:

```
A  B  C  D
|  |  |  |
|---|  |  |
|  |  |  |
B  A  D  C
```

Figure 5.4

This process of securing continues until each object has been secured. The entire process is drawn out in a step-by-step format in **Figure 5.5**, with all secured objects circled (together representing the sequence):

(Keep in mind that the original sequence is EBADC)



A B C D E
← Here E is secured
E
A B C D
← Positions of objects after E is secured
← Here B is secured
B
A C D
← Positions of objects after B is secured
← Here A is secured (it is already in the right position)
A
C D
← Positions of objects after A is secured
← Here D is secured, which also secures C, the final object
D
C
← the entire sequence, EBADC, has been secured

Figure 5.5

The final result:
The final minimal rung structure, generated above, is as follows:

```
A  B  C  D  E
|  |  |  |---|
|  |  |---|  |
|  |---|  |  |
|---|  |  |  |
|  |---|  |  |
|  |  |  |---|
E  B  A  D  C
```

Figure 5.6

Note that the lowest rung above (in gap 4) could be moved up without affecting the solution. The above is designed to indicate each rung drawn below all previous rungs to indicate when in the algorithm it was generated.

Though the above may seem very complex, rest assured that after a little practice, generating the rung structure in this manner becomes very simple. The "securing" of objects in such a lengthy way was done only to clarify the process. In actual practice, the algorithm only requires only a few brief glances to see where your next rungs need to be placed. This second algorithm, however, requires more time to execute.

Left-right or right-left:
Keep in mind as you try this algorithm that the process of securing objects in the sequence works going both left to right or right to left. Securing the last object, then the second to last, and so on will generate a minimal rung solution that is often different from the solution when performed left to right, though it can also be the exact same solution.

Algorithm 2: "the explicit algorithm"

Overview of algorithm:
This algorithm, which you will observe is more complicated than the previous algorithm, has one clear advantage. This algorithm, in contrast to the recursive algorithm, uses numbers that can all be generated independently, and thus provides a rung solution that can be drawn all at once rather than step by step.

The numbers needed for this algorithm are the following numbers, each of which is generated independently from the others (they are shown along with the letters that will represent them algebraically in the following algorithm):

Notation:
- Column number (for each object) = c = the column the object is found in, the leftmost column being column one
- Essential number (for each object) = e = the number of other objects that come before this object in the sequence but after it in the top row
- Rung number (for the entire ladder sequence) = Sum(e) = the sum of the values of e for all objects in the ladder's sequence

The algorithm:
For the last object in the sequence, list all integral values in ascending order from c – e to c – 1. (In other words, if c = 4 and e = 3, list the numbers from 4 – 3 (= 1) to 4 – 1 (= 3), or 1,2,3) If c – e is greater than c – 1 (when e = 0), do not list any numbers (You may want to think of the process as listing the numbers greater than or equal to c – e and less than or equal to c – 1 in ascending order).

Continue this process for each object in the sequence, moving from the right to the left, appending each new list of numbers to the end of the previous object's list. The final result is called the *gap list*. The gap list that the problem in **Figure 5.8** (page 16) produces is 3,4,5,6,5,4,1,2

Note that, because you are listing all the numbers from (c – e) to (c – 1), and because the number of integers from integer x to smaller integer y is equal to $x – y$ + 1**,** the number of integers listed for an object will always be equal to (c – 1) – (c – e) + 1, or e. Therefore, the total quantity of listed numbers will equal the sum of the values of e for all the objects, or Sum(e).

Reading the completely generated list from left to right, place a rung in the gap whose number corresponds to each number in the list. For example, if the next number in the list is a 3, create a rung in the third gap from the left. Make sure that each new rung you create is below all previous rungs. *Note:* because each number in the list represents a rung, and because the number of numbers in the list is equal to Sum(e), you can see that Sum(e) = the number of rungs needed for the solution to the problem.

Keep in mind that this algorithm does not produce the only solution to a problem, nor the only minimal solution. It simply produces one minimal solution to the problem. The proof that the solution is minimal can be found in the next section.

Comparing the NDR Function to Sum(*e*):

The NDR function, defined in the Relations Groups fact in Section Three, gives the number of relations, or location-relationships between objects, that differ from one sequence to another of the same size. This means that if any object starts to the left of another object at the top of a ladder, and ends to the right of it on the bottom, the relation between those two objects has changed, and is counted by the NDR function of the two sequences. The number e for any object gives the number of other objects that started to the left of it and ended to the right of it, such that adding up all the e values for all the objects in a ladder, or Sum(e), will give the result of the NDR function of that ladder. Note that, because e gives only the objects that go from the left to the right, and not the right to the left, object switches are not counted twice in Sum(e). Therefore, NDR(ladder) = Sum(e).

For Algorithm 2, an alternate means of finding e:

If you find the definition of the value of e in Algorithm 2 a bit plain, here is a somewhat more interesting way to find the e values for each object that may put into perspective the source of the numbers. In the sequence, starting with the rightmost object, moving left, count the position of the alphabetically earliest letter, beginning the count with position zero. For example, if the sequence is BDAGEFC, first find the position of a, counting from the right, beginning with zero:

```
B  D  A  G  E  F  C     (A, e = 2, c = 3)
0  1  2
```

Figure 5.7

Cross off A, recording that it is in position 2 from the left. Thus, A’s essential number is 2. Now count the same position for B, C, and all other objects, crossing them off as you count them. However, when counting the position of an object, skip over any crossed off objects:

B D ~~A~~ G E F C (B, *e* = 0, c = 1) ← object A has been crossed off
0

~~B~~ D ~~A~~ G E F C (C, *e* = 4, c = 7) ← object B has been crossed off
- 0 - 1 2 3 4

~~B~~ D ~~A~~ G E F ~~C~~ (D, *e* = 0, c = 2)
- 0

~~B~~ ~~D~~ ~~A~~ G E F ~~C~~ (E, *e* = 1, c = 5)
- - - 0 1

~~B~~ ~~D~~ ~~A~~ G ~~E~~ F ~~C~~ (F, *e* = 1, c = 6)
- - - 0 - 1

~~B~~ ~~D~~ ~~A~~ G ~~E~~ ~~F~~ ~~C~~ (G, *e* = 0, c = 4) (the last object will always have *e* = 0)
- - - 0

Figure 5.8

Note that this finds *e* for each letter in alphabetical order. This is done because it is assumed that the top row of the ladder is made up of the letters in alphabetical order. If the top row is not in alphabetical order, do this process for each letter in the order they appear in the top row.

Comparing Algorithm 2 to Algorithm 1:

Undoubtedly Algorithm 2 may seem daunting and a bit overwhelming when looked at without a concept of what is going on behind the scenes. If you study the process carefully, you will see that Algorithm 2 generates the rung structure equivalent to what you would get if you performed Algorithm 1 from the right to the left rather than from left to right, (both of which are acceptable methods). The essential number *e* (Algorithm 2) represents the number of positions each object has to move to get to where it has to be once it is that object's turn to be secured (Algorithm 1). Crossing off an object after is has been counted in generating *e* (so that it no longer affects the essential numbers of other objects) represents the state of being secured in Algorithm 1. Finding the value of $c - e$ in Algorithm 2 is finding the difference between the object's ending place (*c*) and the number of horizontal spaces to the right it must travel (*e*), producing the number of the gap in which the first rung must be made. Finding $c - 1$ is finding the number of the gap to the left of the object in the sequence, which is the last gap in which a rung must be made. By listing the numbers from $c - e$ to $c - 1$, you list the gaps in which rungs must be made to bring the object from its starting place to its ending place. Lastly, by finding the list of numbers for the rightmost object first and moving left, you secure the rightmost object first and move left securing objects. Because all new rungs in the securing process must be made after all previous rungs, the gap lists for each object must be appended to the end of all previous gap lists.

Section Five: Proving the Optimal Efficiency of the Algorithms

Notation:

If two objects "share" a rung, they both use that rung.

Fact:

If two object cross paths, they must share a rung.

Explanation:

The only place that objects can cross paths is on rungs, and if two objects cross paths on a rung, they must both use that rung.

Fact:

If two objects change sides, they share an odd number of rungs.

Explanation:

If two objects change sides, they cross paths an odd number of times. If two objects cross paths, they share a rung.

New Notation:

Because objects that change sides must share a rung, and because each rung is shared by two and only two objects, each rung can from now on be referred to by the two objects that share it. Thus, a rung by which object B changes sides with object E shall be called "rung BE," or "the BE rung," with the order of objects B and E not important, but for convenience named here in alphabetical order.

Fact:

If there exist two rungs in the same ladder with the same name (i.e. two "BE" rungs), they are considered "duplicate rungs" and can be eliminated in pairs without affecting the solution.

Explanation:

Eliminating one "XY" rung will make the position of objects X and Y switch places anywhere below that rung, and keep X and Y in the same place anywhere above it. No other objects are affected. Thus, any other "XY" rungs before removing the rung are still "XY" rungs. Removing a second "XY" rung brings X and Y back to where they were before removing the first "XY" rung, such that the entire sequence is unaffected. Thus, if there is an odd number of a "XY" rungs, you need only one, and if there is an even number, you don't need any.

Fact:

If a rung structure has two duplicate rungs, it is not a minimal rung structure.

Explanation:

If rungs can be removed without changing the sequence (as duplicate rungs can), the solution isn't minimal.

Theorem #4 – Rung Essentiality Theorem:

All non-duplicate rungs are essential to the permutation.

Proof:

Going back to Theorem 3, A ladder problem can be interpreted as simply a given sequence (the top row) accompanied by a description of all the objects relations that must change, or the objects that must change sides (the rung structure). The proof of Theorem 3 demonstrates that a unique sequence can be defined by an initial arrangement and the set of *all* relations that must change. Therefore, each changing relation is necessary for the definition of the new sequence, so any rung that is essential for two objects to change sides is essential to the definition of the permutation. A rung would only be unnecessary if it was not needed to define Therefore, the only way for a rung structure to not be minimal is if it has one or more sets of duplicate rungs, for all non-duplicate rungs are essential.

New Terminology:

For non-minimal rung structures, rungs will be distinguished as either *duplicate rungs*, defined above, or *essential rungs*, being necessary for the permutation of sequences.

Fact:

Because all non-duplicate rungs are essential rungs, a rung structure without duplicate rungs (i.e. every rung has a different name) is a minimal rung structure.

Theorem #5 – Algorithm Essentiality Theorem:

The algorithms described will never produce duplicate rungs, and therefore always produce minimal rung structures.

Proof:

As described in the comparison of the two algorithms, both Algorithm 1 and Algorithm 2 "secure" objects one at a time, set them aside in a way, and continue their processes with the remaining objects. This alone is proof that the solutions generated by the algorithms will never generate algorithms with duplicate rungs, and that therefore the algorithms are both minimal. When an object is secured, all necessary rungs are created that are needed to secure the object, such that if you are securing object x, all the generated rungs will be rungs used by object x, and in each rung object x changes sides with a different object. After these rungs are created, object x is secured such that no future rungs will be used by object x. Because the securing process itself does not create duplicate rungs, and because being secured ensures no future involvement in duplication, the entire process creates only essential, and never duplicate rungs. Both algorithms, therefore, are minimally efficient.

Section Six: Important Ladder Properties, Examining Rung Structures

Theorem #6 - The Even-Odd Rung Theorem:

Multiple ladder solutions for a given sequence will either all have an odd number of rungs or all have an even number of rungs. In other words, a given permutation cannot have both an odd-'runged' solution and an even-'runged' solution.

Proof:

This theorem draws from the idea shown in Theorem 4. Because all non-duplicate rungs are essential to a solution, the only way to modify the number of rungs in a solution is to add or take away a pair of duplicate rungs that do not affect the outcome of the solution (essential rungs cannot be taken away, only moved to equivalent locations, which does not change the total number of rungs). Because there is a certain base number of essential rungs (equal to the NDR function of the permutation) to which duplicate pairs must be added, the number of total rungs in a solution will always be equal to that base number, or greater than that number by a multiple of two (for example, if $NDR(S_1,S_2) = 5$, the total number of rungs in the solution can only be a number in the set $\{5, 7, 9, \ldots\}$. Because all the numbers in such a set must be either all even or all odd, there can never be both odd and even numbers of rungs in the solutions to a single permutation.

Sidenote:

Those familiar with cyclic notation for permutations (which will be discussed further in later papers) will notice that an even permutation in cyclic notation (a permutation always written with an even number of cycles) is an odd permutation in ladder notation (has an odd number of rungs).

Fact:

There can exist multiple minimally efficient solutions to a single permutation (I consider two permutations "different" if they cannot be expressed with the same gap notation, a definition that may change in later papers). All solutions to a single permutation, however, have all the same rungs by name:

Example of multiple solutions:

The rung structures with gap notation 1234123121 and 4321432434 both produce the permutation from ABCDE to EDCBA.

Example of false multiple solutions:
The gap notation 2312 and 2132 are both possible gap notation for what I call a "false multiple solution." False multiple solutions occur when the gap solution has two consecutive numbers that differ in value by more than one. As a result, the rungs they represent do not affect each other and can be written with either rung above the other or, if one so chooses, horizontally across from each other. Though in future papers, when I examine the quantity of multiple solutions, I may consider these as essentially different solutions, for now I will treat them as equals.

Fact:

Reversal of objects can be achieved, on a ladder of size n, with the gap notation:
(n-1), (n-2),…,(1),(n-2),(n-3),…,(1),(n-3),(n-4),…,(1),…,(1)

Conjecture #1 – The Transmutation by Reversal Conjecture:

Different rung structures that produce the same solution can be changed into each other through a series of *triangular-reversals*. A triangular reversal is the act of taking a triangular group of rungs that together reverse the order of the objects on the involved columns, and flipping it horizontally (a group of rungs that reverses objects will work both forwards and backwards).

Section Seven: Future Papers

Some topics to be discussed in future papers:

- **Multiple Solutions**
  - **For arbitrary permutations**
  - **How many different unique solutions are there for reversing n objects?**
- **Addressing the Transmutation by Reversal Conjecture**
- **Composition of Ladders (Sticking Ladders on top of Each Other)**
  - **Examining the composition of ladders as a group**
  - **Comparing composition of ladders to traditional composition of permutations**