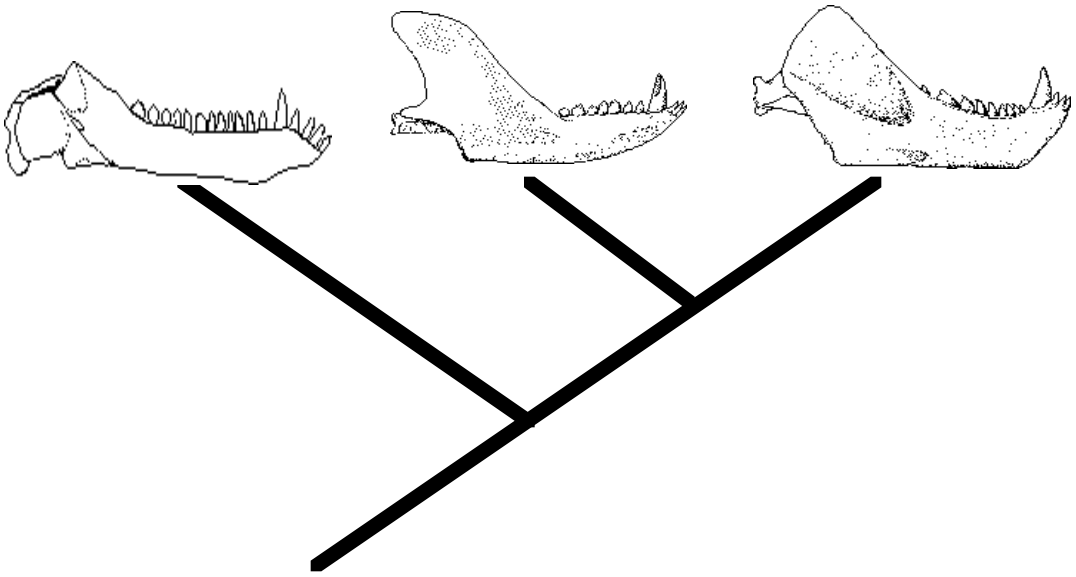
Cladistic Analysis Using Hennig86[1]



Diana Lipscomb

George Washington University

[1] Hennig86 version 1.5, Copyright (c)James S. Farris 1988 All rights reserved

Preface

Hennig86 is an interactive, phylogenetic analysis program for MS-DOS machines written by James Farris. It provides facilities for coding and weighting characters, for readir writing, diagramming and diagnosing phylogenetic trees, and for calculating most parsimonious trees. This guide is designed to acquaint you with the basic features and options of the Hennig86 computer package. The first part briefly reviews basic cladistic the and terminology. The remaining chapters describe how to calculate trees, diagnose cladograms, carry out character analysis, and deal with multiple trees. Each of these chapte has worked examples and describes how to carry out the calculations using Hennig86.

I hope this guide makes using phylogenetic methods simpler for you. Please report any errors or omissions you find with this guide to me.

Diana Lipsco

Department of Biological Scien
George Washington Univers
Washington D.C. 20052 U
e-mail: BIODL@GWU.E

Table Of Contents

Constructing Cladograms 1
- Introduction To Cladistics 1
- Hennig Argumentation And Wagner Trees 3
- Using Hennig86 To Construct A Cladogram 13
  - Creating A Data File 13
  - Starting Hennig86 And Commands For Constructing Basic Cladograms 15
  - Viewing And Interpreting Trees 17
  - Printing Trees 19
  - Saving Output On Disk 19
  - Contolling What Is Seen On The Monitor 22
  - Inserting Messages From The Keyboard 22
  - Viewing Files 23
  - Other Commands For Constructing Trees 25

Diagnosing A Cladogram 36
- Tree Lengths, Consistency And Retention Indices 36
- Optimization 44
- The Tree Editor - Dos Equis 49
- Analysing Trees Not Produced With Hennig86 54

Character Analysis 62

Outgroup Designations And Rerooting The Trees 69

Multiple Trees 73
- Consensus Trees 74
- Successive Weighting 75

Multistate Characters 84

References 109

Appendices
- Summary Of Hennig86 Commands 113
- Hennig86 Error Messages 119

Cladistics or Phylogenetic Systematics

The logic behind cladistics and arguments for why it is the preferred method of phylogenetic analysis is beyond the scope of this guide. There have been numerous books and papers on the subject, however, I recommend reading: Eldredge and Cracraft 1980, Farr 1970, 1979a, 1979b, 1980, 1982a, 1983, 1986, Farris et al., 1970, Lipscomb 1983, Nelson and Platnick 1981, Schoch 1986. In brief, cladistic methods group organisms that share derived characters. Taxa that share many derived characters are grouped more closely together tha those that do not. The relationships are shown in a branching hierarchical tree called a cladogram. The cladogram is constructed such that the number of changes from one charac state to the next are minimized. The principle behind this is the rule of parsimony - any hypothesis that requires fewer assumptions is a more defensible hypothesis.

Determining Primitive (Plesiomorphic) and Derived (Apomorphic) Characters

The first step in basic cladistic analysis is to determine which character states are primitive and which are derived. Many methods for doing this were proposed by Hennig and others, but one method can be justified on logical grounds. This is **outgroup comparis** In this method, if a taxon that is not a member of the group of organisms being classified h a character state that is the same as some of the organisms in the group, then that character state can be considered to be plesiomorphic. The outside taxon is called the **outgroup** and t organisms being classified are the **ingroup**. How can we justify using this method? Consid the following example in which a character has states ***a*** and ***b***. There are only two possibilities:

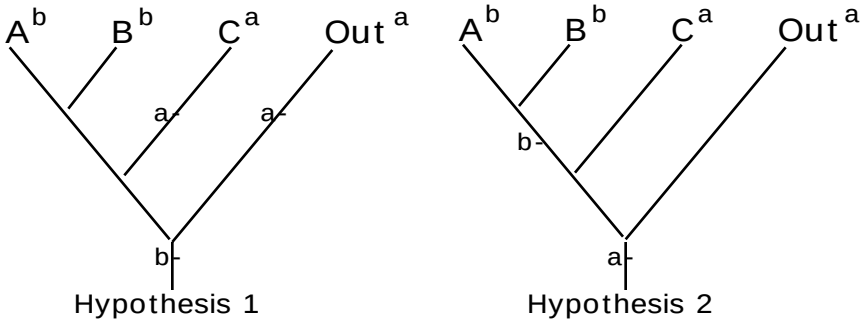
1. ***b*** is plesiomorphic and ***a*** is apomorphic

b --> ***a***

2. ***a*** is plesiomorphic and ***b*** is apomorphic

a --> ***b***

If state ***a*** is also found in a taxon outside the group being studied, the first hypothesi will force us to make more assumptions than the second. That is it is less parsimonious:



In hypothesis 1, state ***b*** is the plesiomorphic state and is placed at the base of the tree. If stat only evolved once, it must be assumed that character state ***a*** evolved two separate times - once in taxon C and once in the Outgroup. In hypothesis 2, state ***a*** is the plesiomorphic stat and is placed at the base of the tree. In this hypothesis, both character states ***a*** and ***b*** each

evolve only once. Therefore, hypothesis 2 is more parsimonious and is a more defensible hypothesis. This example illustrates why outgroup analysis gives the most parsimonious, and therefore logical, hypothesis of which state is plesiomorphic.

If the character has only two states, then the task of distinguishing primitive and derived character states is fairly simple: The state which is in the outgroup is primitive an the one found only in the ingroup is derived. (see section VII for analysis of multistate characters).

It is common practice to designate the primitive states as 0 (zero) and the derived states as 1 (one). Below is the character chart of some sample data with the information for the outgroup:

	Characters									
	1	2	3	4	5	6	7	8	9	10
Outgroup	a	a	a	a	b	a	a	a	a	a
Alpha	a	a	a	b	a	b	b	a	b	a
Beta	a	a	a	b	b	b	b	a	b	a
Gamma	a	a	a	a	b	a	a	a	b	a
Delta	b	b	b	a	b	a	b	b	b	a
Epsilon	b	b	b	a	b	a	b	a	b	b
Zeta	b	b	a	a	b	a	b	a	b	b
Theta	b	a	a	a	b	a	b	a	b	b

The rescored character chart is shown below the derived character states (apomorphies) cod as 1 and primitive states coded as zero (0):

	Characters									
	1	2	3	4	5	6	7	8	9	10
Outgroup	0	0	0	0	0	0	0	0	0	0
Alpha	0	0	0	1	1	1	1	0	1	0
Beta	0	0	0	1	0	1	1	0	1	0
Gamma	0	0	0	0	0	0	0	0	1	0
Delta	1	1	1	0	0	0	1	1	1	0
Epsilon	1	1	1	0	0	0	1	0	1	1
Zeta	1	1	0	0	0	0	1	0	1	1
Theta	1	0	0	0	0	0	1	0	1	1

Constructing a Cladogram

There are two ways to use the character information and group taxa together based o shared apomorphies.

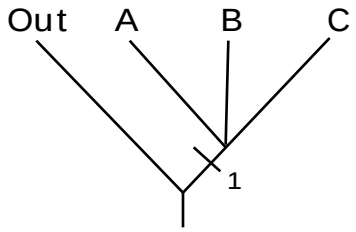
Hennig Argumentation

The first method was described by Hennig and is called Hennig argumentation. It works by considering the information provided by each character one at a time.

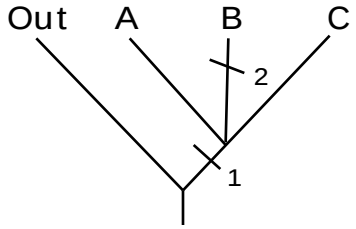
Consider the following small data set:

	Characters				
	1	**2**	**3**	**4**	**5**
Outgroup	**0**	**0**	**0**	**0**	**0**
A	**1**	**0**	**0**	**0**	**1**
B	**1**	**1**	**0**	**1**	**0**
C	**1**	**0**	**1**	**1**	**0**

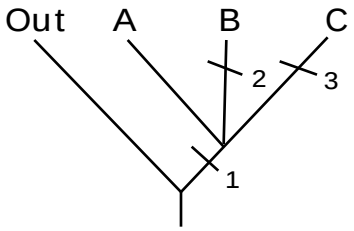
1. Look at the information in character 1 and unite all of the taxa that share the apomorph state. This character unites taxa A, B, and C:



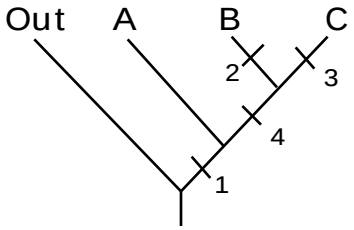
2. Character 2 - the derived state is found only in taxon B. It is an autapomorphy of that taxon and provides no information about the relationships among the taxa:



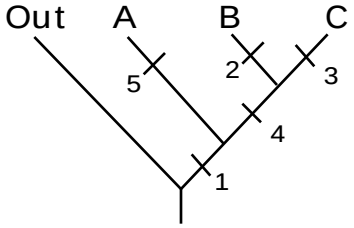
3. Character 3 - the derived state is an autapomorphy for taxon C:



4. Character 4 - the derived state is a synapomorphy that unites taxa B and C:



5. Character 5 - the derived state is an autapomorphy for taxon A:



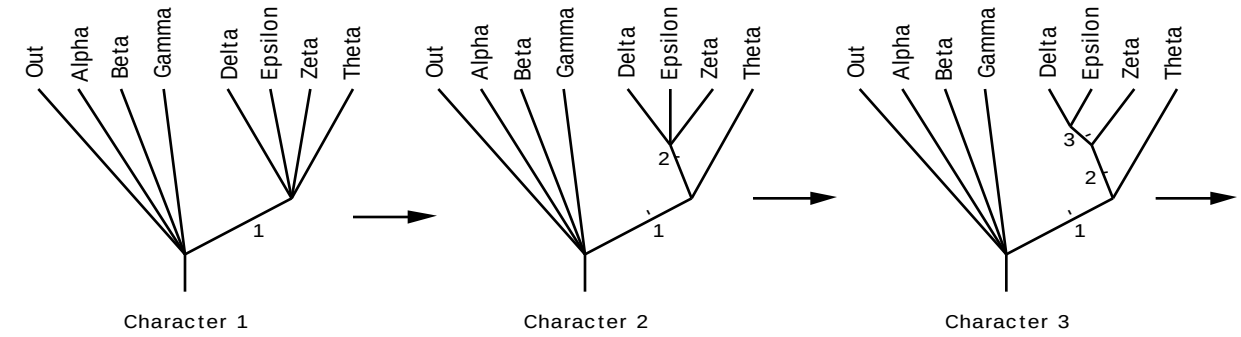
The relationships of the taxa are resolved.

Real sets of character data are rarely so simple. A more realistic situation would be one where the characters conflicts about the relationships among the taxa. When it does, choose the tree that requires the fewest assumptions about character states changing.

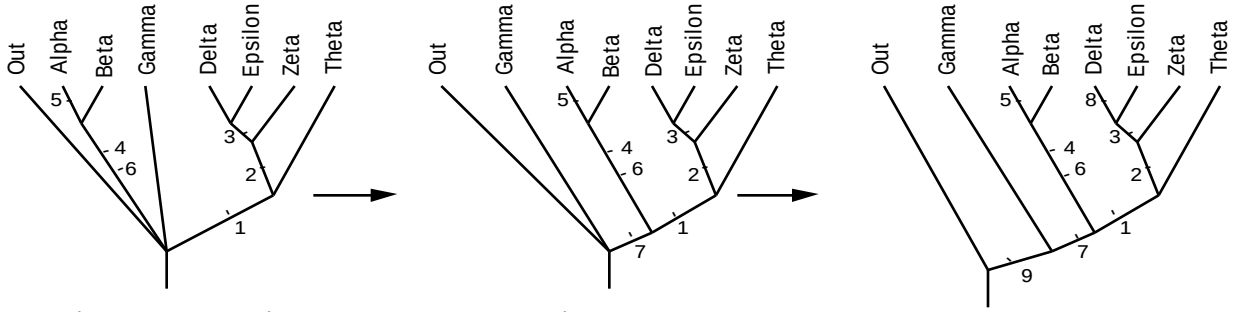
To examine this, construct a cladogram using Hennig argumentation from the mor complicated data set:

	Characters									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Outgroup	0	0	0	0	0	0	0	0	0	0
Alpha	0	0	0	1	1	1	1	0	1	0
Beta	0	0	0	1	0	1	1	0	1	0
Gamma	0	0	0	0	0	0	0	0	1	0
Delta	1	1	1	0	0	0	1	1	1	0
Epsilon	1	1	1	0	0	0	1	0	1	1
Zeta	1	1	0	0	0	0	1	0	1	1
Theta	1	0	0	0	0	0	1	0	1	1

1. Grouping taxa is straightforward when characters 1-9 are used:

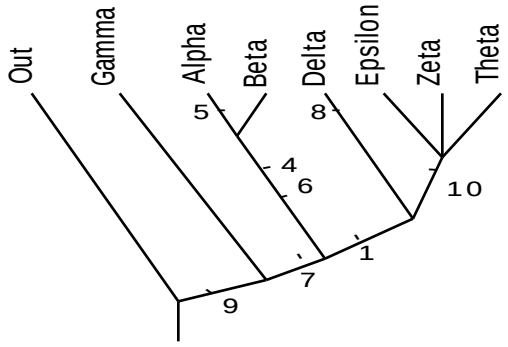


Character 1 Character 2 Character 3



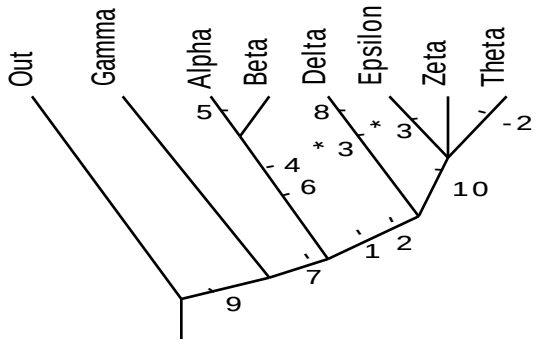
Characters 4, 5 and 6 Character 7 Characters 8 and 9

2. Character 10 conflicts with characters 2 and 3 - it suggests the following relationships of t taxa:



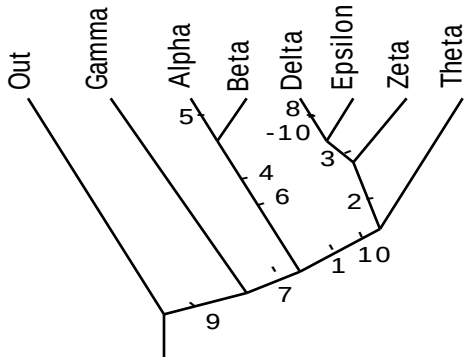
Character 10

If this tree is accepted, we must assume that character 2 secondarily reverses to state 0 in tax Theta and that character 3 is convergent in Delta and Epsilon:



Character 10
Assumes 1 convergence and 1 reversal

If the tree that characters 2 and 3 support is accepted, we must assume that character 10 reverses to state 0 in taxon Delta:



Assumes 1 reversal in Character 10

Since this tree requires us to make the fewest assumptions, it is the most parsimonious and therefore is the preferred cladogram.

References: Hennig 1966, Schoch, 1986.

Wagner Trees

A second way to construct a cladogram is to connect taxa together one at a time unti all the taxa have been added. When added, each taxon is joined to the tree to minimize th number of character state changes.

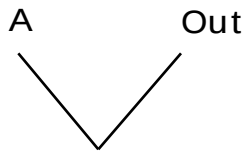
Consider again the following small data set:

	Characters				
	1	**2**	**3**	**4**	**5**
Outgroup	**0**	**0**	**0**	**0**	**0**
A	**1**	**0**	**0**	**0**	**0**
B	**1**	**1**	**0**	**1**	**0**
C	**1**	**0**	**1**	**1**	**1**

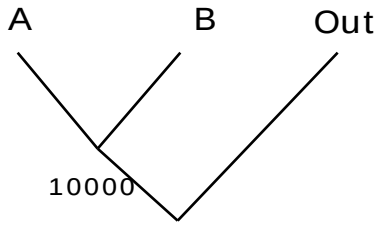
1. Find the organism with the lowest number of derived character states and connect it to t outgroup**.**

	Characters					#advanced steps
	1	**2**	**3**	**4**	**5**	
Outgroup	**0**	**0**	**0**	**0**	**0**	**0**
A	**1**	**0**	**0**	**0**	**0**	**1**
B	**1**	**1**	**0**	**1**	**0**	**3**
C	**1**	**0**	**1**	**1**	**1**	**4**

Organism A has the lowest number of advanced steps:

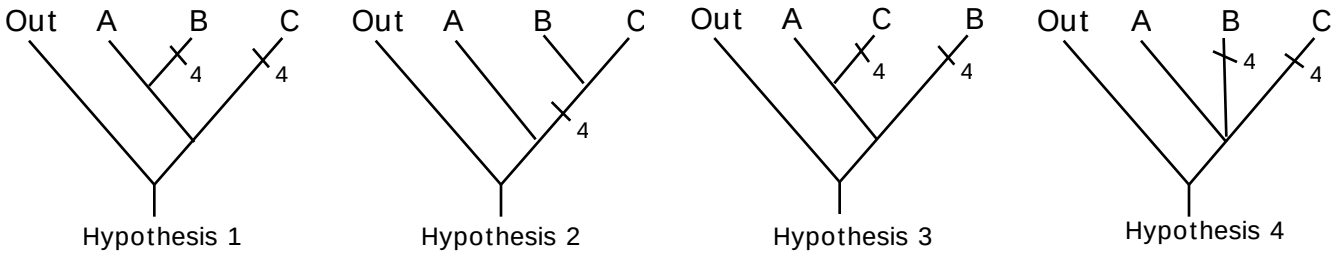


2. Now find the organism with the next lowest number of derived character states. Write name beside the first organisms' name and connect it to the line that joins the outgroup ar the first organism. At the point where the two lines intersect, list the most advanced state present in both of the two organisms. For the above character set, the second organism is B



Organisms A and B both have the derived state for character 1 but one or both of them hav state 0 for the remaining characters. At the point where the two lines intersect, 1000 is give

3. Find the organism with the next lowest number of derived character states and connect
to the point which requires the fewest number of evolutionary steps (character state change
to derive the organism. In this example, organism C is added next. There are several
different places it can be attached:



Hypotheses 1, 3, and 4 all require us to assume that character 4 state 1 evolved two times.
Hypothesis 2 does not require us to make that assumption. Therefore the most parsimonio
placement of organism C is as in Hypothesis 2. The analysis is complete.

Wagner Formula

The number of steps required to attach a taxon to any other can be summarized by the
formula:

$$\mathbf{d(A, B) = \sum / X(A}i\mathbf{) - X(B}i\mathbf{) /}$$

Where,

d = the number of character state changes between taxa A and B

$\sum$ = sum of all differences between

X(A,*i*) = the state of a character for taxon A
and
X(B,*i*) = the state of a character for taxon B

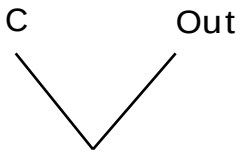
A simple way to see how this can be calculated is by working through an example using the
data set:

	Characters										**# advanced steps**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Out	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	
A	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**4**
B	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**3**
C	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**2**
D	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**5**
E	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**5**

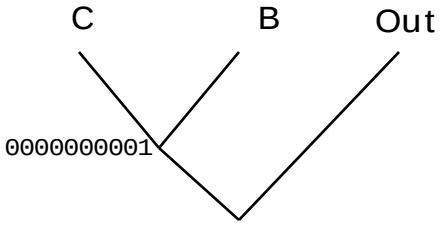
1. **Find the organism with the lowest number of derived character states and connect it to t**

outgroup.

C has the fewest number of derived steps. It is linked to the outgroup first:



2. **B** has the next fewest number of advanced states. It is joined to **C**. At the point whe the two lines intersect, the advanced states present in both are listed. For example, characte is at state 0 in **C** but in state 1 for **B**. 0 is the most advanced state that they share so it is listec next to the point where the lines leading to **C** and **B** intersect. On the other hand, both **B** ar **C** have state 1 for character 10. Therefore, a 1 is given for character 10 at the point where th two lines intersect. Listing these states is hypothesizing the characteristics present in the hypothetical ancestor of **B** and **C**.



3. A has the next lowest number of advanced character states. It must be connected such that the fewest number of steps are required to reach it (fewest number of character sta changes).

If **A** were attached to the line leading to **C**:

```
          A    1000110001
          C    0000000011
differences:   1   11  1        (4 steps would have to be made)
```

To attach **A** to the line leading to **C** would require the assumptions that characters 1, 6, and 9 all evolved advanced states.

If **A** were attached to the line leading to **B**:

```
          A    1000110001
          B    1000100001
differences:         1          (1 step would have to be made)
```

To attach **A** to the line leading to **B** would require the assumptions that character 6 evolved an advanced state.

If **A** were attached to the line leading to the point where the lines leading to **B** and **C** intersect:

```
          A     1000110001
          BC    0000000001
differences:    1    11          (3 steps would have to be made)
```

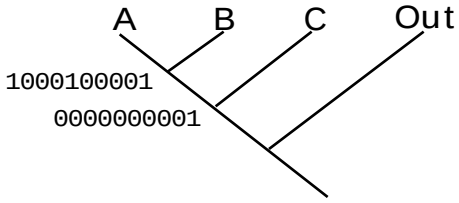
To attach **A** to the line leading to **BC** would require the assumptions that characters 5, and 6 all evolved advanced states.

If **A** were attached to the line leading to the Outgroup:

```
          A     1000110001
          O     0000000000
differences:    1    11    1(4 steps would have to be made)
```

To attach **A** to the line leading to the outgroup would require the assumptions that characters 1, 5, 6, and 10 all evolved advanced states.

Since it would take the fewest evolutionary steps to attach A to B, that is what is done:



4. Either **D** or **E** can be attached next. For this example, **D** will be attached first, then **E**.

If **D** were attached to the line leading to **A**:

```
          D     0110001101
          A     1000110001
differences:    111 1111  (7 steps would have to be made)
```

To attach **D** to the line leading to **A** would require the assumptions that characters 1,2,3,5,6,7, and 8 evolved advanced states.

If **D** were attached to the line leading to **B**:

```
          D     0110001101
          B     1000100001
differences:    111 1 11          (6 steps would have to be made)
```

To attach **D** to the line leading to **B** would require the assumptions that characters 1,2,3,5,7, and 8 evolved advanced states.

If **D** were attached to the line leading to **C**:

```
          D    0110001101
          C    0000000011
differences:    11   111 (5 steps would have to be made)
```

To attach **D** to the line leading to **C** would require the assumptions that characters 2, 7, 8, and 9 evolved advanced states.

If **D** were attached to the line leading to the point where the lines leading to **B** and **A** interse

```
          D    0110001101
          AB   1000100001
differences:    111 1 11       (6 steps would have to be made)
```

To attach **D** to the line leading to **AB** would require the assumptions that characters 2, 3, 5, 7 and 8 all evolved advanced states.

If **D** were attached to the line leading to the point where the lines leading to **B, A** and **C** intersect:

```
          D    0110001101
          ABC  0000000001
differences:     11   11       (4 steps would have to be made)
```

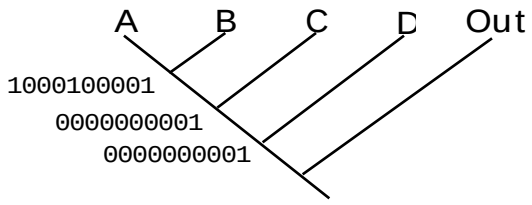
To attach **D** to the line leading to **ABC** would require the assumptions that character 3, 7 and 8 all evolved advanced states.

If **D** were attached to the line leading to the Outgroup:

```
          D    0110001101
          O    0000000000
differences:     11   11 1(5 steps would have to be made)
```

To attach **D** to the line leading to the outgroup would require the assumptions that characters 1, 5, 6, and 10 all evolved advanced states.

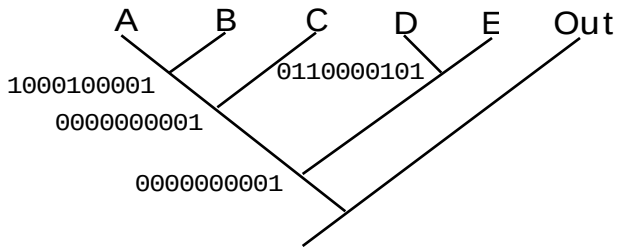
Since it would take the fewest evolutionary steps to attach D to the line leading to ABC, tha what is done:



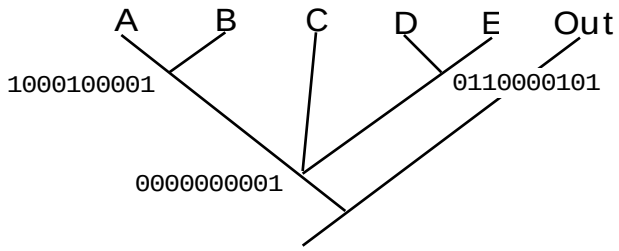
5. **E** is attached last:

E --> A=	7 steps	
E --> B =	6 steps	
E --> C =	5 steps	
E --> D =	2 steps	
E --> AB	=	6 steps
E --> ABC	=	4 steps
E --> ABCD	=	4 steps
E --> Out	=	5 steps

E is connected to D and the cladogram is complete:



Since there are no changes between the point where **DE** branch off and **C** branches off the tr
the tree could also be drawn as:



Hennig86 Data Files

Creating a Data File on a Disk

The first step in using Hennig86, is to create a file with the distributions of the character states in the taxa. This data can be stored on a separate disk or in a separate file or directory and accessed by Hennig86. The data must be in a text (or ASCII) file. Use a text editor or refer to your word processing manual for how to create text files (its usually simpl

The input data file is called a **procedure** file. It must be constructed in a specific framework that consists of seven parts:

1. The command **xread** should be typed on the first line.

2. A title, enclosed in single quotes. This line is optional - if you don't want a title, leave this line out.
The title may contain any character except a single quote ('). This includes contractions (don't, can't) and the possessive case. Do not do this:

```
'This is a reanalysis of Mike's data on Owls'
```

The program will read the title as:

```
'This is a reanalysis of Mike'
```

But it won't be able to interpret:

```
s data on Owls'
```

As a result, the rest of the file will not be read properly.

3. The number of characters.
The number must be in the range 1-500.

4. The number of taxa.
The number of taxa must be in the range 4-180. This number includes all of the ingroup and outgroup taxa.

5. The list of the taxa and their character data in several lines using the following form:

a) taxa name

The name of a taxa may include alphabetical letters, underscore and integers but the first character must be an alphabetical letter. The name can be any length, but only the first ten characters will be retained and printed on the final cladogram. Alphabetical letters are stored in lower case and printed on the final cladogram in lower case. Therefore, you must take care that all of your taxon names can be distinguished by the first ten characters in lower case.

You cannot use periods or blank spaces in the taxa name. This means that if you are working on several species within a genus, you cannot type in *B. teres, B. vorox, B. aurelia,* etc. Either replace the period and space with an underscore

(*B_teres*, *B_vorox*, *B_aurelia*) or use the species name only.

By default, the first taxa typed in is the outgroup. See section x to learn how to designate another taxon or multiple taxa as outgroups.

b) character data

Characters must be separated from the taxon name by a space. They must be numbers ranging from 0 to 9. An unknown or missing state may be represented by either **?** or **-** (dash). The character states may be separated by blank spaces but this is not necessary.

6. A semicolon

7. End the input file with **`proc/;`**

The information (1-7 above) can be typed on separate lines, or on one line with the different information separated by blank spaces, or with some information on separate line and some on one line. For example, the following data files for four taxa and three charact will produce the same results:

Data File 1:

```
xread 'title' 3 4 alpha 000 beta 100 gamma 110 delta 111; proc/;
```

Data File 2:

```
xread
‘title’
3 4
alpha 0 0 0
beta 1 0 0
gamma 1 1 0
delta 1 1 1
;
proc/;
```

Data File 3:

```
xread
 ‘title’
3 4
alpha  000
beta   100
gamma  110
delta  111
;
proc/;
```

Data File 4:

```
xread
'title'
3    4
alpha
0 0 0
beta
1 0 0
gamma
1 1 0
delta
1 1 1
;
proc/;
```

An input data file (or procedure file) for the data set:

	Characters 1 2 3 4 5 6 7 8 9 10
Outgroup	a a a a b a a a a a
Alpha	a a a b a b b a b a
Beta	a a a b b b b a b a
Gamma	a a a a b a a a b a

```
Delta      b b b a b a b b b a
Epsilon    b b b a b a b a b b
Zeta       b b a a b a b a b b
Theta      b a a a b a b a b b
```

Could look something like this:

```
xread
'data file for basic cladistic procedures'
10   8
Outgroup 0 0 0 0 1 0 0 0 0 0
Alpha 0 0 0 1 0 1 1 0 1 0
Beta 0 0 0 1 1 1 1 0 1 0
Gamma 0 0 0 0 1 0 0 0 1 0
Delta 1 1 1 0 1 0 1 1 1 0
Epsilon 1 1 1 0 1 0 1 0 1 1
Zeta 1 1 0 0 1 0 1 0 1 1
Theta 1 0 0 0 1 0 1 0 1 1
;
proc/;
```

The next section of this guide will call this file `sample.dat` and use it to illustrate how to r Hennig86. You may want to write this file on a disk and work along with the directions in this manual.

Using Hennig86 to analyze the data in SAMPLE.DAT

The procedure described below assumes that the Hennig86 program is on the C: dri and the data disk with the file **`sample.dat`** is in the a: drive. You may adjust the commands for your personal computer set-up. To start Hennig86, simply type **ss** and press enter:

```
C:\>ss
```

The monitor will display the message:

```
     Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                        All rights reserved
              This copy produced for the exclusive use of
                             Jane Doe

  *>
```

The ***>** symbol is the Hennig86 prompt for a command from the keyboard. At any point, y

may exit Hennig86 by typing **yama** and pressing the enter key.

First, tell the Hennig86 program where the input data is by typing the word procedure followed by the pathway:

```
*>procedure <pathname>;
```

The pathname consists of the drive where the data is stored and the name of the file. For example, if the data is on a disk in the A: drive in a file called sample.dat, then the pathnar would be **a:sample.dat** .

```
*>procedure a:sample.dat;
```

NOTE: This line must end with a semicolon.

Most Hennig86 commands can be abbreviated. For example, the command **procedure;** can abbreviated simply as **p**:

```
*>p a:sample.dat;
```

On the monitor, the following message appears:

```
procedure a:sample.dat *
xread
title (e.g.,data file for basic cladistic
                procedures)
procedure --
*>
```

The meaning of these messages is straightforward. The first line is the program tell you what its going to do (i.e., get the sample.dat file) followed by the first to lines of the **sample.dat** file. Finally, the prompt (*>) lets you know that the program is waiting for a command to tell it what to do with the data.

Constructing a simple cladogram

There are many commands in Hennig86 for calculating trees. The command **hennig;** (can abbreviated **h;**) constructs trees by a single pass through the data. This is quite rapid, and is good command for simple data sets[1]. After the prompt type the word hennig followed by a semicolon and press enter:

```
*>hennig;
```

[1]This command might not find the shortest tree if the data set is large, contains mucl homoplasy, or if more than one equally parsimonious tree exists. See the later sections of th guide for other tree building commands.

The message that then appears on the monitor is:

```
hennig length 11 ci 90 ri 90
```

What does this mean?

```
hennig - the method of analysis used
length 11 - eleven steps in the tree
ci 90 - the consistency index is 100
ri 90 - the retention index is 100
```

(The meaning of the length, consistency index and retention index are explained in the section II).

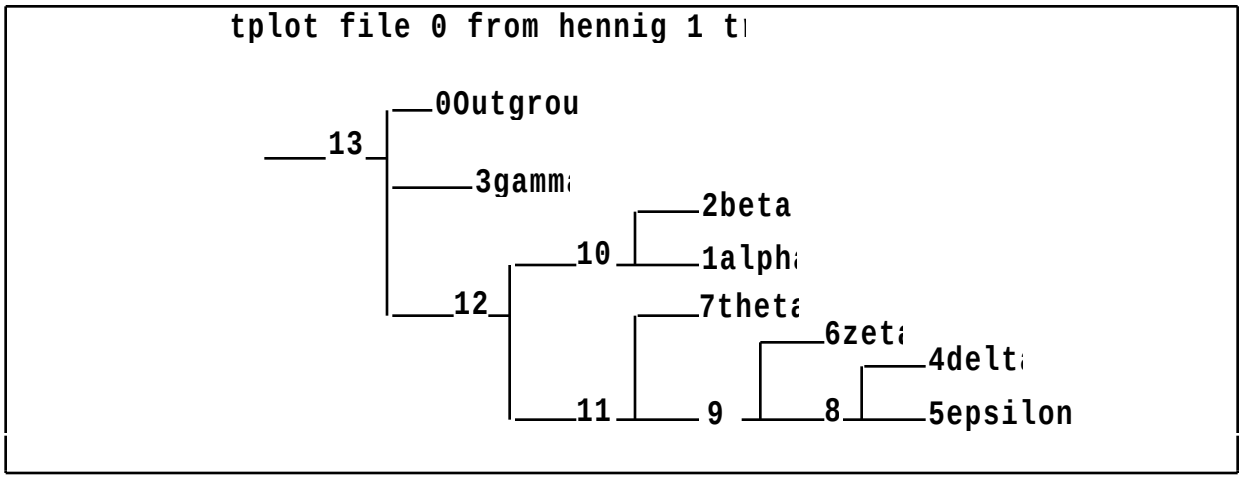
If the message on the monitor is **hennig>** instead of the line above, then you simply forgot type a semicolon at the end of the command. Type a semicolon and press ENTER and the correct message should appear.

Viewing and Interpreting the Tree

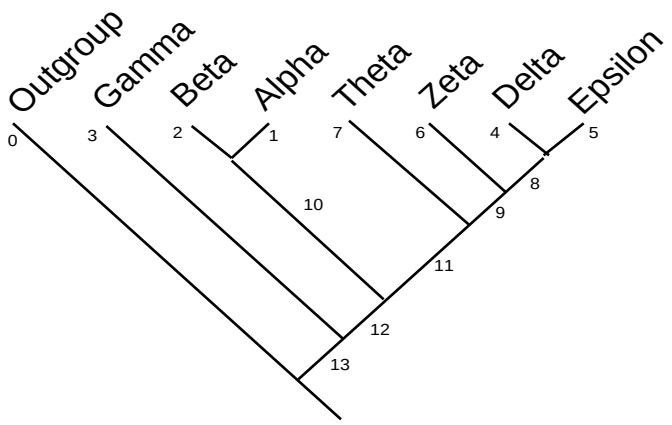
What does the cladogram look like? The command to show the tree is **tplot** (or it can be abbreviated **tp;**). At the prompt type **tplot** and press enter:

```
*>tplot;
```

The output that then appears on the monitor is:



A more conventional representation of this tree is:



By comparing this conventional cladogram to the tree produced by Hennig86, the notation used in the tplot output becomes clear:

1. The **taxa names** are written in lower case at the ends of the terminal branches. Only the first ten letters of each taxa name are printed.

2. Before each taxon name is **a number at the end of each terminal branch**. These numbers correspond to the order of the taxa in the data matrix. The outgroup is the first taxon and is numbered 0 (zero), the first taxon of the ingroup is numbered 1 (in this case it is alpha), and so forth.

3. The **internal branches of the cladogram** are also numbered. For example, taxa alpha and beta are sister taxa and share a common ancestor. A single branch leads to the node where the two groups diverge. This branch is labeled 10. These numbers make it easy to refer to specific parts of the cladogram.

Printing the Tree

To get a printout of your cladistic analysis as it appears on the monitor, turn on your printe and press the control key and P key (Ctrl-P) at the same time. This will direct all output tha appears on the monitor to the printer as well. After the printer has been activated with the Ctrl-P command, type **tp;** (or **tplot;**) and the tree will be printed as well as appearing on the monitor.

Exiting to DOS

To exit the Hennig86 program, at the command prompt type **yama** and press enter:

```
*>yama
```

Saving Output on Disk - Log Files

Having a printed copy of your analysis is good, but often you will want to save outp in a file on a disk. Saved files are called log files and they are created with the command:

```
*>log <pathname>; [Press Enter]
```

After the log file has been opened with this command, all subsequent output is sent to that file. Analyses and trees that were produced before the log file was opened are not saved.

Note: The pathname is the disk and the filename you want the output saved in. I find it useful to give my log files the same name as my data files but append them with the file typ .OUT.

Example:

Previously, a data file called **sample.dat** was written and saved in a file on a disk. To sa the analysis of this data in a file on a disk, use the following steps.

First, enter Hennig86 and load the data file:

```
C:\>ss
```

The monitor will display the message:

```
    Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                    All rights reserved
          This copy produced for the exclusive use of
                          Jane Doe
```

After the prompt, load the data file using the procedure command (here it is abbreviate as **p**):

```
*>p a:sample.dat;
```

The monitor will display the message:

```
procedure sample.dat *
xread
data file for basic cladistic procedures
procedure --
*>
```

To save all of the output to a file on a disk, create a log file:

```
*>log a:sample.out;
```

At this point, you should get the message on the monitor:

```
log a:sample.out*
```

(If the computer instead gives the message **log>**, then you forgot to type a semicolon at the end of the log command. Just type a semicolon now and press enter. The log file will be properly created and you can proceed.)

Construct a basic cladogram and print it:

```
*>h;tp;[Press Enter]
```

The tree is described and you are told a tree plot has been created for your data:

```
hennig length 11 ci 90 ri 90
tplot file 0 from hennig 1 tree
```

Note -- the trees are not displayed on the monitor. The trees are saved in the computer to l
written to the output file. To complete the writing of the file and to close the log file type:

```
>log/;[Press Enter]
```

the monitor shows the message:

```
log ---
```

The disk drive whirs and the output is written to the file **sample.out**. The output on the di
can be examined using the command:

***>view a:sample.out;**

or you can exit Hennig86 and view the file with a word processing program.

If the output is not saved and the computer displays the message:

```
log>
```

then you forgot to type the semicolon at the end of the command **log/;** Do so now and the file should be written.

Don't Worry!

If you exit the Hennig86 program and forget to close the log file (you type **yama** but forget to give the **log/;** command first), the output will be automatically written to the file.

Controlling What is Saved

The following commands will help control how output is sent to log files:

log <pathname>; creates a new log file**.**
log-; temporarily deactivates the log file and output is no longer saved.
log*; reactivates the file and output is saved in that file again. Old data are not overwritten - new data are appended to the end.
log/; closes the file and it is written to disk.

Creating a new log file using the **log <pathname>** command will also close the existing log file. The log commands can all be abbreviated with the letter **l** instead of the word **log**.

WARNING - If a file is closed with the **log/;** command, it cannot be reactivated with the **lo** command. Giving the command **log <old pathname>** will erase the old file and create a ne one with the old name.

If after typing any of the log commands, you get the message:

```
log>
```

then you forgot to type a semicolon at the end of the command. Simply type one after the prompt and press the ENTER key.

Controlling What Is Seen On The Monitor

When there is no open or active log file, all of the results of the analysis are displayed on th monitor. But when a log file is activated, only brief messages are copied to the monitor. These are just the default settings of the program. You can control what is displayed on the monitor.

The following commands control what is sent to the monitor when a log file is open:

display -; turns off the monitor and all output is sent to the log file.
display *; turns on the monitor so that all output is sent to it as well as the log file.
display; resets the default: brief messages are copied to the monitor and longer messages are sent to the log file.

The display commands can all be abbreviated with **d** instead of the word **display**.

Inserting Messages From the Keyboard Into the Output Data File

Often while doing a cladistic analysis, I make notes about what options I am using or about the data set being analyzed. The **quote** command allows me to insert these messages directl into the relevant part of the saved log file from the keyboard by first typing the word **quote** and pressing enter:

```
*> quote (NO SEMICOLON) [Press Enter]
```

The prompt on the monitor changes:

```
quote>
```

Messages up to several lines long can be typed in. If the printer is on, they will be printed. I the log file is open, they will be inserted into the log file. When the the message is finished type a semicolon to signal the end of the message and press enter.

For example:

```
*>quote
quote>
quote>The following is a tree for data [ENTER]
quote> the in the file sample.dat [ENTER]
quote>; [press ENTER]
*>
```

The quote command can be abbreviated with **q** instead of the word **quote**.

NOTE: For this command, the semicolon signals the end of the message - so do not use a semicolon as part of your message.

Viewing Files

While in Hennig86 you can look at any data or output file using the command:

*>**view <pathname>** (Note-**no** semicolon) [Press Enter]

The view command can be abbreviated with **v** instead of the word **view**.

Short files

The file viewer displays a file one page (or screen) at a time. If the viewed file is less than c screen long, the file will be shown followed by the Hennig86 command prompt (*>). The data set sample.out is small and less than one screen long. View that file from within Hennig86:

C:\->**ss**

The monitor will display the message:

```
Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                All rights reserved
     This copy produced for the exclusive use of
                  Jane Doe
```

Then type:

***>p a:sample.dat;**

The monitor will display the message:

```
procedure sample.dat *
xread
data file for basic cladistic procedures
procedure --
*>
```

Now type:

```
*>view a:sample.dat
```

The file will be displayed on the monitor followed by the (*>) prompt:

```
xread
'data file for basic cladistic procedures'
10
8
Out 0000000000
A 0001101001
B 0001001001
C 0000010000
D 1110001110
E 1110001010
F 1100001010
G 1000001010
;
proc/;
*>
```

Long files

If the file fills more than one screen, the first screen will appear on the screen followed by a line of control characters. These control characters allow you to move around within the f and view each page separately. The control line reads:

```
.dn-up/pgdn\pgup*end,ret 0 view>
```

Break this line up into its parts and its meaning becomes clear:

/ moves up one screen
\ moves down one screen
. (period) to go down one line at a time
- (dash) to go up one line at a time
***** to go to the end of the file
0 (zero) to go to the beginning of the file
, to exit view and return to Hennig86 command prompt

On the Hennig86 disk is a file called **PEG** (this is data from Peggy Bolick's 1985 paper in Cladistics 1:114-125). This file is longer than one screen.
Type:

```
*>view peg
```

The first page of the file should appear with the control line as the last line on the screen.

If the first page appears, but with a Hennig86 command prompt (*>) instead of the control line, you typed a semicolon at the end of the view command. Try again without the semicolon.

Move through this file using all of the control characters until you are familiar with how they all work.

NOTE: Viewing a file can be useful for looking at files on a disk, but it will not allow you to edit these files. For that you must exit the Hennig86 program and use a text editor or word processor.

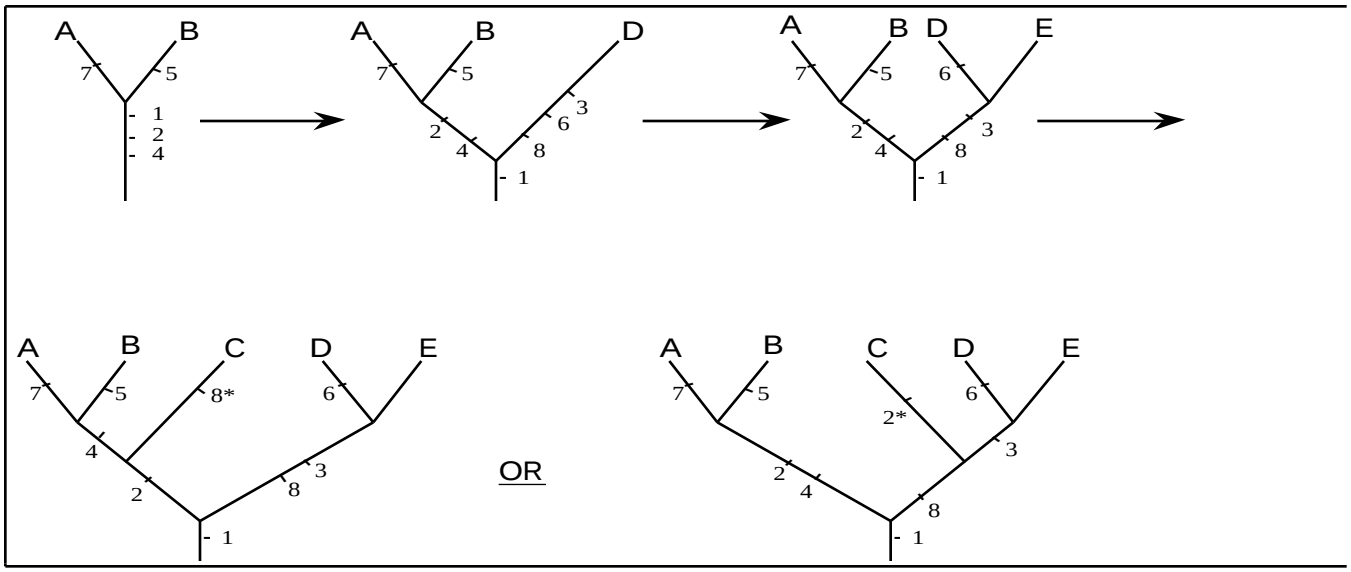
Additional Options for Calculating Trees

Multiple Trees for One Data Set

There is not always just one most parsimonious cladogram for a given data set. For examp consider the following data set:

```
xread
'Demonstrating Multiple Equal Trees'
8  6
Outgroup  00000000
A         11010010
B         11011000
C         11000001
D         10100101
E         10100001
;
proc/;
```

If the cladogram is constructed without using a computer, it becomes clear that taxon C can united with taxa A and B because they share the synapomorphy for character 2, or taxon C can be united with taxa D and E because they share the synapomorphy for character 8:



Both trees have 9 steps, both require just one homoplasy (when C is attached to the A-B cla character 8 is homoplasious; when C is attached to the D-E clade, character 2 is homoplasious), both trees are equally parsimonious.

Using Hennig86 To Find Multiple Equally Parsimonious Trees

The command **hennig;** is executed rapidly, but makes only one pass through the d and will retrieve only one tree. To find additional trees for simple data sets the command **mhennig;** (can be abbreviated **mh;**) is used. The **mhennig;** command constructs seve trees, each by a single pass but adding the taxa in a different sequence each time.

Try this by typing in the above data file and analyzing it in Hennig86 with both the **hennig;** command and the **mhennig;** command. In the directions below, the data file is on disk in the a drive and is called **multiple.dat**, you can adjust this to suit your own set-up:

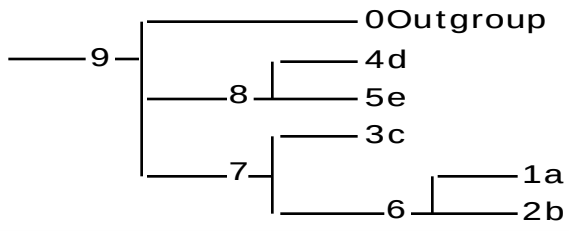
```
     C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                        All rights reserved
            This copy produced for the exclusive use of
                           Jane Doe

     *>procedure A:multiple.dat;

          procedure a:multiple.dat *
          xread
          Demonstrating Multiple Equal Trees
          procedure --
     *>hennig;
```

The command **hennig;** finds only one tree:

```
          hennig length 9 ci 88 ri 83
```

This tree can be seen on the monitor when the **tplot;** command is given:

```
     *>tplot;
          tplot file 0 from hennig 1 tree
```



Now try the command for multiple trees:

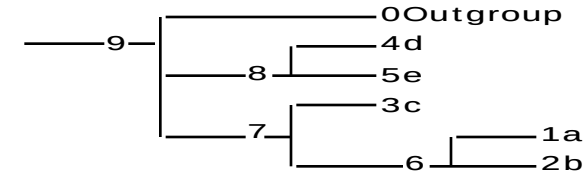
```
     *>mhennig;
```

The command **mhennig;** finds both of the trees found when the cladograms were constructed by hand:

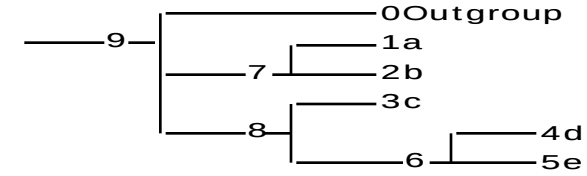
```
          mhennig length 9 ci 88 ri 83 trees 2
```

When the command for plotting the trees is given, both scroll past on the screen:

```
          *>tplot;

          tplot file 0 from mhennig 2 trees


          tree 0
```




```
          tree 1
```



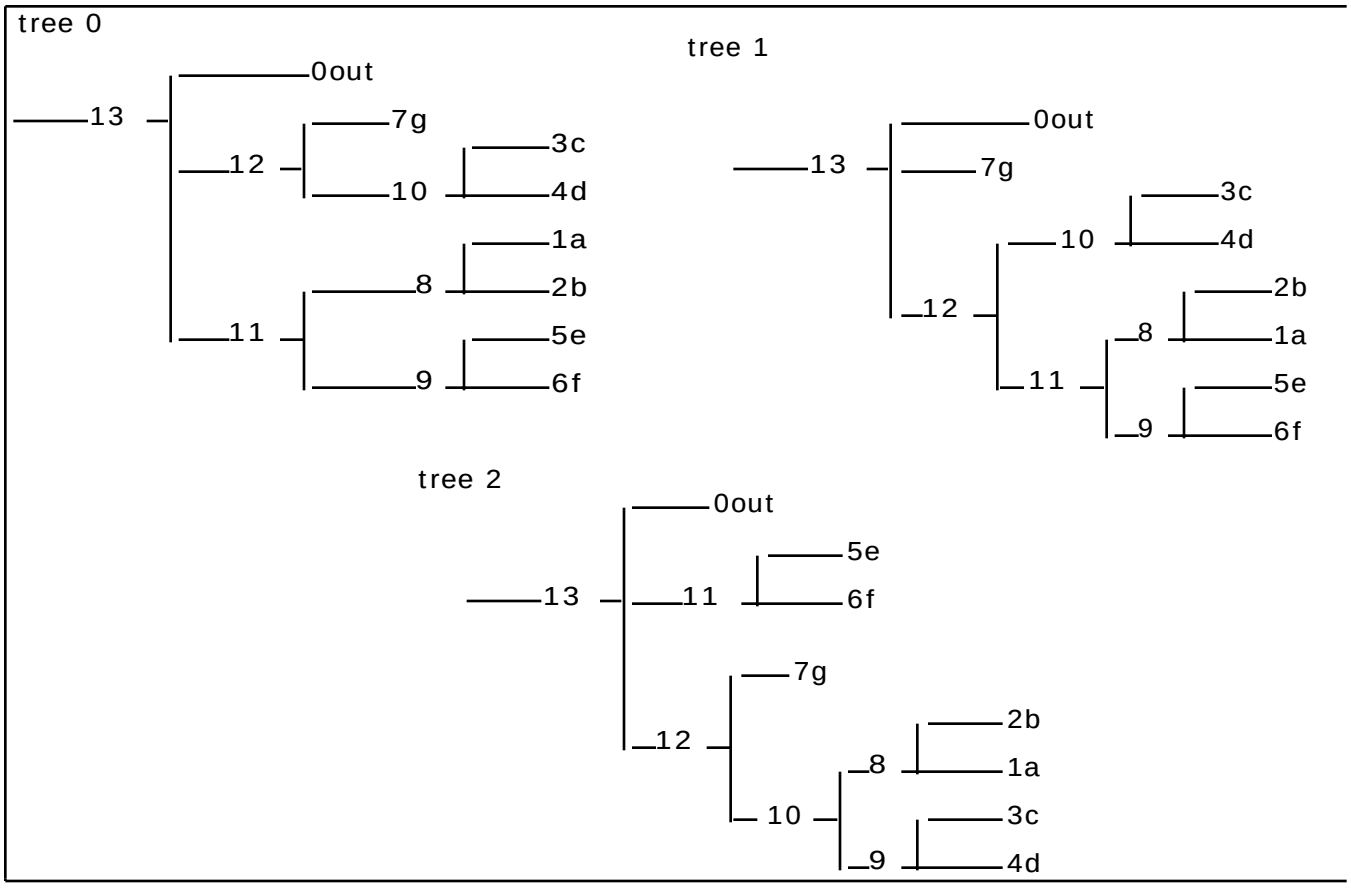
Note - Here is a convention you will need to get used to: the first tree is tree 0 (zero) in Hennig86 and the second tree is tree 1.

Branch-swapping

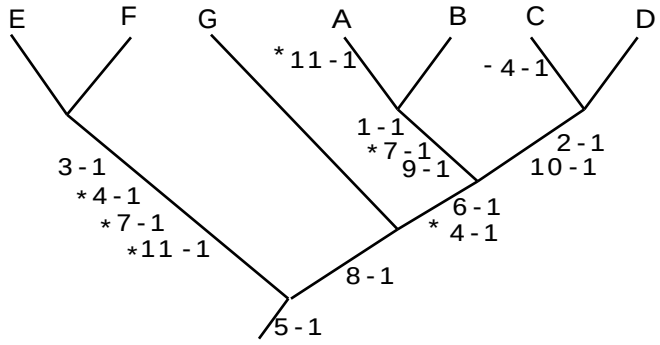
When cladograms are constructed, taxa are added one at a time to gradually build u the tree. When each taxon is added, it is checked against each branch of the existing tree to see where it is most parsimonious to put it. However, a clade of two or more taxa may be placed as a unit as parsimoniously (or even more parsimoniously) in another part of the tr For example, look at the following data set:

```
xread
'Demonstrating Branch Swapping'
11
8
Out   0  0  0  0  0  0  0  0  0  0  0
A     1  0  0  1  1  1  1  1  1  0  1
B     1  0  0  1  1  1  1  1  1  0  0
C     0  1  0  0  1  1  0  1  0  1  0
D     0  1  0  1  1  1  0  1  0  1  0
E     0  0  1  1  1  0  1  0  0  0  1
F     0  0  1  1  1  0  1  0  0  0  1
G     0  0  0  0  1  0  0  1  0  0  0
;
proc/;
```

If the cladogram is constructed by adding one taxon at a time using the **mhennig;** comman three trees are found:

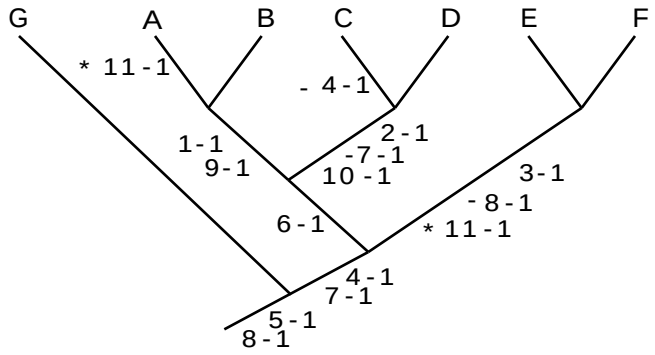


For the time being, just look at the last tree (tree 2). This cladogram is redrawn here in a more conventional format with the character state changes on the branches (reversals are designated with a dash [-] and convergent features with an asterisk [*]):



Counting the number of steps in the tree, we find 15 character state changes (length=15) with 1 character that shows reversal and 3 characters that show convergence.

Using this tree as a guide, draw another cladogram in which the clade with taxa E ar F is moved to be the sister group of clade 10 with taxa A, B, C and D. Place the characters or the branches they define and calculate the length of the tree:



The new cladogram is equally parsimonious to the three trees found by the **mhennig;** command (15 steps).

Moving the branches of a cladogram to check for equally parsimonious or more parsimonious trees is called **branch-swapping**.

Using Hennig86 to branch-swap

The command **hennig;** is executed rapidly, but makes only one pass through the dat and will retrieve only one tree. Additional trees for simple data sets can be found with the command **mhennig;**. To perform branch-swapping on the trees constructed with the **mhennig;** command, use it in conjunction with the command **bb;**. (The **bb;** command does not construct trees. It branch-swaps trees produced by other commands.)

Try this by typing in the above data file and analyzing it in Hennig86 with the **mhennig;** command followed by the **bb;** command. In the directions below, the data file is a disk in the A drive and is called swap.dat:

```
C:\->ss
 Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                All rights reserved
     This copy produced for the exclusive use of
                    Jane Doe

*>procedure A:swap.dat;

procedure a:swap.dat *
xread
Demonstrating Branch Swapping
procedure --

*>mhennig;
```

The command **mhennig;** finds only three trees:

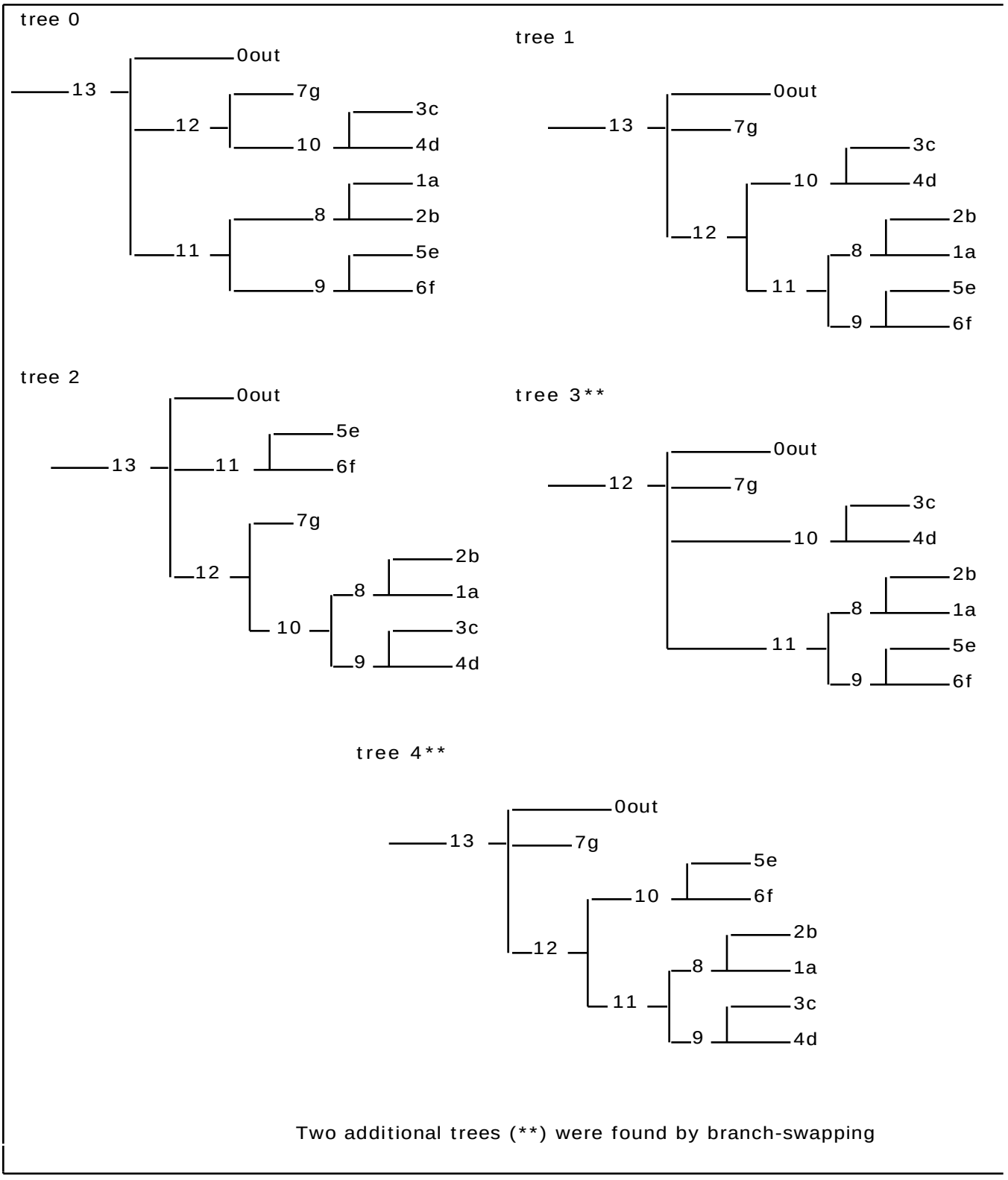
```
mhennig length 15 ci 73 ri 76 trees 3
```

Now try the command for branch-swapping:

```
*>bb;
```

The command **bb;** finds 2 more parsimonious tree:

```
bb length 15 ci 73 ri 76 trees 5

*>tplot;

tplot file 0 from bb 5 trees
```



hennig*;, mhennig*; and bb*; --

In addition to the **bb;** command, Hennig86 will do branch-swapping with three othe commands: **hennig*;**, **mhennig*;** and **bb*;**.

The **hennig*;** (can be abbreviated **h*;**) command constructs a single tree by a single pass through the data and then applies branch-swapping to that tree. After branch-swappin however, it will retaining just one tree.

*When would you want to use the **hennig*;** command?*

The main advantage of both the **hennig;** and **hennig*;** commands is their speed. Fo larger and more complicated data sets, the **hennig*;** command may quickly find a tree by branch-swapping that is more parsimonious than the one produced by the **hennig;** command.

The **mhennig*;** (can be abbreviated **mh*;**) constructs several trees, each by a single p but adding the taxa in a different sequence each time and then applies branch-swapping to each of the trees, retaining just one tree for each initial one.

*When would you want to use the **mhennig*;** command?*

The **mhennig*;** command is only slightly slower than the **hennig*;** commands, and has the advantage of finding more of the equally parsimonious trees.

As we have already seen, the **bb;** command applies branch- swapping to trees constructed by another command (e.g.,**mh; h; mh*;** or **h*;**) and stores these trees in a new t file. Unlike **h;** and **mh;**, which retain only one tree for each initial one found, with the **bb;** command up to 100 of the shortest trees are retained.

The **bb*;** is the same as the **bb;** command except all trees will be retained up to the limits of the space allocated in the computer for tree storage. In other words, if the ***** optior not used, no more than 100 trees will be retained. If the ***** option is used, several hundred trees may be found and stored in a tree file.

*The **bb;** vs. the **bb*;** command*

The **bb*;** command is slower than **bb;**. Even with a short data set such as **swap.dat** (t one we used above), there is a difference in the amount of time needed to execute the commands. If the **hennig;** command is used with **bb;**, five trees are found in 2 seconds. If t **hennig;** command is used with **bb*;**, five trees are found in 3 seconds. If a data set is large a contains many homoplasies, the **bb;** command may significantly reduce the amount of computing time spent finding trees by branch-swapping.

However, if the **bb;** command is used and the maximum 100 trees are found:

```
bb length 49 ci 58 ri 53 trees 100
```

Then it is possible that the other equally parsimonious trees can be found. In these cases, u
the **bb*;** command.

Implicit Enumeration

Three other commands in Hennig86 can be used to construct cladograms:**ie; ie*;** and
ie-;. These commands are guaranteed to find the most parsimonious tree.

ie; generates trees by an exact algorithm (implicit enumeration) and retains up to 10
of the trees. The results are certain to be the most parsimonious trees, but it may be time
consuming if the data set is large or contains much homoplasy.

The **ie*;** command is the same as the **ie;** command except all trees will be retained u
to the limits of the space allocated in the computer for tree storage. In other words, if the *****
option is not used, no more than 100 trees will be retained.

ie-; will identify one tree, certain to be of minimum length. For large data sets, this
may be much faster than **ie;** or **ie*;**

How long these commands take to analyze the data is highly data dependent. Platn
(Table 2, *Cladistics*, 1989, 5:145-161) found that the analysis of the data set from Collette and
Russo's work on mackerels (*Cladistics*, 1985, 1:141-158) took 3 seconds to find four equally
parsimonious trees with the commands **mhennig*;** and **bb;** used together. The **ie*;** analysi
of the same data took 3 minutes and 10 seconds to find the same four trees.

BUT: the implicit enumeration commands are guaranteed to find all the most parsimonious trees

Platnick (Table 2, 1989) also found that the analysis of the data set from Presch's wor
on scincomorphans (Presch, 1988, In: Estes Pregill eds, Phylogenetic Relationships of Lizar
Families. Stanford Univ. Press) found only one tree using the commands **mhennig*;** and
together. The **ie*;** command took longer, but found four equally parsimonious trees.

Timing a Computation

Determining how long a Hennig86 command takes to execute is done with the
command **watch** (can be abbreviated **w**). The time required to carry out each command wil
be displayed on the monitor and in the log file. To turn the watch off, give the command
watch- (or **w-**). No semicolon is necessary for the watch commands.

Command Summary - Calculating trees

hennig; Constructs a single tree by a single pass through the data. This is quite rapid but may not find the shortest tree if there is much homoplasy in the data.

hennig*; Constructs a single tree by a single pass through the data and then applies branch-swapping to the initial tree, retaining just one tree.

mhennig; Constructs several trees, each by a single pass but adding the taxa in a different sequence each time.

mhennig*; Constructs several trees, each by a single pass but adding the taxa in a different sequence each time and then applies branch-swapping to each of the trees, retaining just one tree for each initial one.

bb; Applies branch-swapping to trees constructed by another command (e.g., **mh;** or **h;**) and stores these trees in a new tree file. Unlike **h;** and **mh;**, whic retain only one tree for each initial one found, with the **bb;** command up to 100 of the shortest trees retained.

bb*; Same as the **bb;** command except all trees will be retained up to the limits of the space allocated in the computer for tree storage. In other words, if t ***** option is not used, no more than 100 trees will be retained.

ie; Generates trees by implicit enumeration and retains up to 100 of the trees. The results are certain to be the most parsimonious trees, but it may be time consuming if the data set is large or contains much homoplasy.

ie*; Same as the **ie;** command except all trees will be retained up to the limits of the space allocated in the computer for tree storage. In other words, if t ***** option is not used, no more than 100 trees will be retained.

ie-; Identifies one tree, certain to be of minimum length. For large data sets, this may be much faster than **ie;** or **ie*;**

The best command to use is **ie*;** It will find and retain all the most parsimonious trees. If the data set is large, with a lot of homoplasy the **ie*;** command may be prohibitively slow. A combination of the commands **mh*;bb;** ie-**;bb;** usually find all the trees and will run significantly faster. If you just want a quick preliminary analysis, the **mh*; h;** or **mh;** commands.

Tree Diagnosis

When a cladogram is constructed for a given set of characters, it is built so that the fewest assumptions of homoplasy are made because it will be the most parsimonious for th data. Parsimony is used because it produces the most informative classification for the character data (Farris, 1983). It follows that the fewer the homoplastic steps on a tree, the shorter the length and the greater the descriptive power of the tree for a set of characters.

Therefore, after a cladogram has been constructed, it is useful to know how many character show convergences and parallelisms (i.e., homoplasy). When a command that constructs a cladogram is executed, a line with this information is given:

```
C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                    All rights reserved
          This copy produced for the exclusive use of
                         Jane Doe

                        *>p a:sample.dat
procedure a:sample.dat *
xread
title (e.g., data file for basic cladistic procedures)
procedure --
*> ie;

ie length 11 ci 90 ri 90 trees 1
```

Length

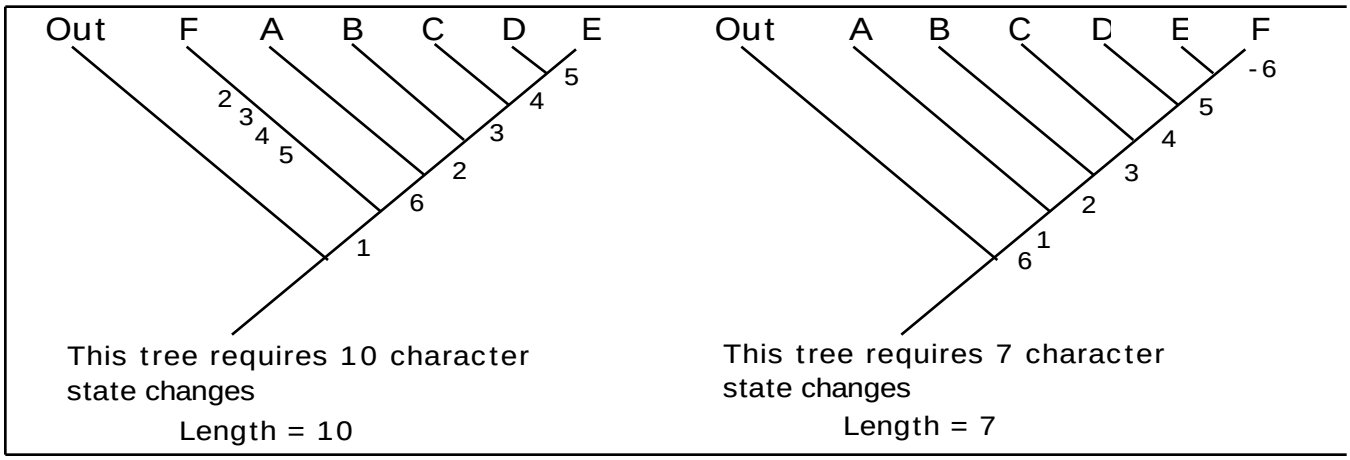
The **length**, or number of **steps**, is the total number of character state changes necess to support the relationship of the taxa in a tree. The better a tree fits the data, the fewer homoplasies will be required and the fewer number of character state changes will be required. Therefore, a tree with a lower length fits the data better than a tree with a higher length. The tree with the lowest length compels us to assume fewer homoplasies and so is more parsimonious - it will be the hypothesis of taxa relationship that is selected.

Consider the following data set (Diagnosis Data Set 1) of 6 taxa, 1 outgroup, and 6 characters:

	Characters					
	1	2	3	4	5	6
Outgroup	0	0	0	0	0	0
Taxa A	1	0	0	0	0	1
Taxa B	1	1	0	0	0	1

```
Taxa C    1 1 1 0 0 1
Taxa D    1 1 1 1 0 1
Taxa E    1 1 1 1 1 1
Taxa F    1 1 1 1 1 0
```

Character 6 in this data set suggests that taxa A, B, C, D, and E are in a group that excludes taxon F. The other characters suggest that taxa F and E are most closely related and that the are as a group related to taxon D; the clade with taxa F, E and D are related to C, then this group is related to B; finally, taxon A is joined to the clade containing B, C, D, E, and F:



The second tree has a lower length because it requires fewer homoplasious steps. In other words, it is the most parsimonious, and therefore is a better hypothesis of the relationships the taxa.

Create a Hennig86 data file for the data set. Calculate a tree using the command **ie;** :

```
C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                  All rights reserved
      This copy produced for the exclusive use of
                        Jane Doe

                      *>p a:diagnosis.dat
procedure a:diagnose.dat *
xread
title
procedure --
*> ie;

ie length 7 ci 85 ri 85 trees 1
```

The length (in this case 7 steps) is given when the command **ie;** has calculated the tree.

The length of the tree can also be determined with the command:

`xsteps l;` *(note this is a lower case L)*

This command can be abbreviated:

`xs l;`

NOTE: If you type an uppercase L instead of lowercase, the status of any log files will be displayed. This does not appear to have any damaging affect on the log file.

References: Camin and Sokal (1965), Farris (1970), Swofford and Maddison (1987), Maddison (1989).

Consistency Index

The relative amount of homoplasy can be measured using the **consistency index** (often abbreviated **CI**). It is calculated as the number of steps expected given the number of character states in the data, divided by the actual number of steps multiplied by 100. The formula for the CI is:

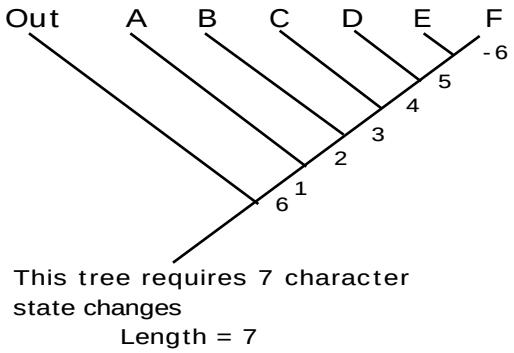
$$\mathbf{CI = \frac{\text{total character state changes expected given the data set}}{\text{actual number of steps on the tree}} \times 100}$$

For example, in the data set above:

Diagnosis Data Set 1

```
           1 2 3 4 5 6      Character  State Changes in
                                                  Data
Outgroup   0 0 0 0 0 0                1            0 --> 1
 Taxa A    1 0 0 0 0 1                2            0 --> 1
 Taxa B    1 1 0 0 0 1                3            0 --> 1
 Taxa C    1 1 1 0 0 1                4            0 --> 1
 Taxa D    1 1 1 1 0 1                5            0 --> 1
 Taxa E    1 1 1 1 1 1                6            0 --> 1
 Taxa F    1 1 1 1 1 0
                              Total character state changes
                                  expected in the data set = 6
```

The cladogram of these data is:



The character state changes in the tree total 7 because character 6 reverses to state 0 in taxon

```
Character   State Changes on Tree:
     1              0 --> 1
     2              0 --> 1
     3              0 --> 1
     4              0 --> 1
     5              0 --> 1
     6              0 --> 1 --> 0
```

The **Consistency Index** = 6 / 7 X 100 = 85

References: Farris (1970, 1991), Kluge and Farris (1969), Goloboff (1991)

Retention Index

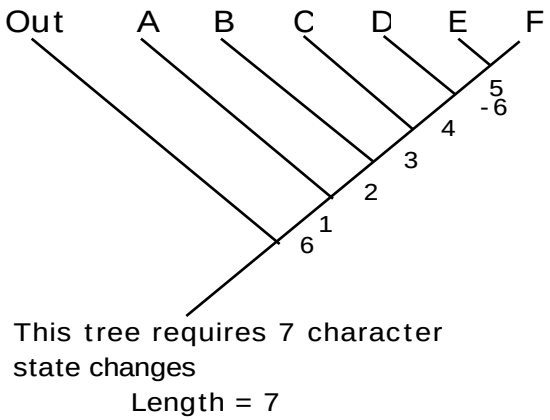
Another measure of the relative amount of homoplasy required by a tree is the **retention index (RI)**. The retention index measures the amount of synapomorphy expected from a data set that is retained as synapomorphy on a cladogram.

In order to describe the Retention Index and distinguish it from the CI, lets look at another example:

Diagnosis Data Set 2

```
           1 2 3 4 5 6        Character   State
                                        Changes in Data
Outgroup   0 0 0 0 0 0              1          0 --> 1
 Taxa A    1 0 0 0 0 1              2          0 --> 1
 Taxa B    1 1 0 0 0 1              3          0 --> 1
 Taxa C    1 1 1 0 0 1              4          0 --> 1
 Taxa D    1 1 1 1 0 1              5          0 --> 1
 Taxa E    1 1 1 1 1 0              6          0 --> 1
 Taxa F    1 1 1 1 1 0
                        Total character state changes
                             expected in the data set = 6
```

The cladogram of these data is:



The character state changes in the tree total 7 because character 6 reverses to state 0 in the li leading to taxa E and F:

Character	State Changes on Tree
1	0 --> 1
2	0 --> 1
3	0 --> 1
4	0 --> 1
5	0 --> 1
6	0 --> 1 --> 0

The **Consistency Index** = 6 / 7 X 100 = 85.7

The consistency index is the same for this data set and the one used above, but the data sets are not identical. In the first data set, the homoplasious character (#2) reverses in a termin branch. In data set 2, the homoplasy defines a clade including 2 terminal branches. Unlike the first data set, in Data Set 2 the homoplasy is informative about the branching pattern of the taxa. This additional information is measured by the retention index but is ignored by consistency index.

To calculate the retention index:

$$\textbf{RI} = \frac{\textbf{maximum number of steps on tree - number of state changes on the tree}}{\textbf{maximum number of steps on tree - number of state changes in the data}} \times 100$$

The maximum number of steps on the tree is the total number of taxa with state 1 or state ((whichever is smaller) , summed over all the characters.

The **RI** of the Data Set 1:

```
maximum number of steps =
            1 2 3 4 5 6      Character         Max steps
Outgroup    0 0 0 0 0 0                1             1
 Taxa A     1 0 0 0 0 1                2             2
 Taxa B     1 1 0 0 0 1                3             3
 Taxa C     1 1 1 0 0 1                4             3
 Taxa D     1 1 1 1 0 1                5             2
 Taxa E     1 1 1 1 1 1                6             2
 Taxa F     1 1 1 1 1 0
                                Total max steps in data set = 13

            RI =  13 - 7  X 100 = 85.7
                  13 - 6
```

But, The RI of the second data set is higher than the consistency index:

```
maximum number of steps =

            1 2 3 4 5 6      Character         Max steps
Outgroup    0 0 0 0 0 0                1             1
 Taxa A     1 0 0 0 0 1                2             2
 Taxa B     1 1 0 0 0 1                3             3
 Taxa C     1 1 1 0 0 1                4             3
 Taxa D     1 1 1 1 0 1                5             2
 Taxa E     1 1 1 1 1 0                6             3
 Taxa F     1 1 1 1 1 0
                                Total max steps in data set = 14

            RI =  14 - 7  X 100 = 87.5
                  14 - 6
```

References: Farris (1991)

Fit of Individual Characters

It is often of interest to examine how well an individual character is fit by a tree. W can measure the length, consistency index, and retention index of individual characters.

Length

The length of an individual character is the number of character state changes or ste required to fit it to the tree. In the first diagnosis data set, character 6 has a length of 2. It changes from state 0 to 1 in the branch leading to the ingroup taxa (step 1) and changes agai from state 1 to 0 in the line leading to taxon F.

Consistency Index

The consistency index for an individual character is the number of advanced states has in the data set (i.e., the number of state changes we expect) divided by the number of ste in the tree multiplied by 100. In the first diagnosis data set, character 6 has only one advanc state (state 1), but it actually requires 2 steps to fit the cladogram. The CI of this character is:

$$CI = \frac{1}{2} \times 100 = 50.0$$

Retention Index

The retention index for an individual character is the number of taxa with states 1 0 (whichever is lower) (this is the maximum steps for the character) take away the number steps the character makes in the tree divided by the maximum steps for the character take away the number of state changes we expect multiplied by 100. In the first diagnosis data se character 6 has two taxa with state 0 - therefore 2 is the maximum steps. The RI of this character is:

$$RI = \frac{2 - 2}{2 - 1} \times 100 = 0.0$$

The length, consistency index, and retention index of any character on a tree can als be determined with the command:

xsteps c; *(note this is a lower case C)*

This command can be abbreviated:

xs c;

NOTE: If you type an uppercase **C** instead of lowercase, the length of the tree will be display instead.

For example, calculate a tree for this data set using the command **ie;** :

```
C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                  All rights reserved
         This copy produced for the exclusive use of
                       Jane Doe

*>p a:diagnosis.dat
procedure a:diagnose.dat *
xread
title
procedure --
*> ie;

ie length 7 ci 85 ri 85 trees 1

*> xs c;
```

The result is:

```
xsteps file 0 from ie 1 tree
tree 0 length 7 ci 85 ri 85
character/steps/ci/ri
  0   1   2   3   4   5
  1   1   1   1   1   2
100 100 100 100 100  50
100 100 100 100 100   0
```

The first line (**xsteps file 0 from ie 1 tree**) simply tells you what the program is doing - executing the xsteps command on the single tree obtained by the ie command.

The second line (**tree 0 length 7 ci 85 ri 85**) repeats the diagnostics for the entire tree.

The third line (**character/steps/ci/ri**) is the legend explaining the final four lines:

```
  0   1   2   3   4   5  <--- The character
  1   1   1   1   1   2  <--- The number of steps
100 100 100 100 100  50  <--- The consistency index
100 100 100 100 100   0  <--- The retention index
```

NOTE: The first character in the data set is called character 0. If you find that confusing, a simple remedy is to add a dummy character as the first character in the data set. This dumm character should have state 0 for all taxa and the outgroup so that it will not affect the cladogram. In the results, simply ignore character 0 (the dummy character) and character 1 will correspond to the first real character. Try this with the diagnose data set now, the resu of the **xs c;** command should then be:

```
xsteps file 0 from ie 1 tree
tree 0 length 7 ci 85 ri 85
character/steps/ci/ri
  0   1   2   3   4   5   6
  1   1   1   1   1   1   2
100 100 100 100 100 100  50
100 100 100 100 100 100   0
  ↑
   Ignore character 0  -it is the dummy character
```

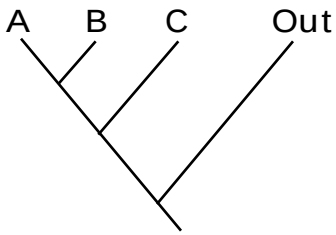
Optimization - Determining Where Character State Changes Occur

Mapping character state changes onto a tree so that the fewest number of steps are hypothesized is called optimization.

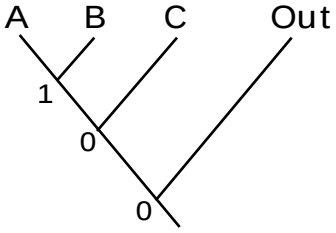
Consider the following simple example for 4 taxa and 2 characters:

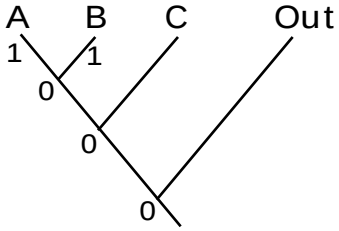
```
Characters
      1 2
Out   0 0
A     1 1
B     1 1
C     0 1
```

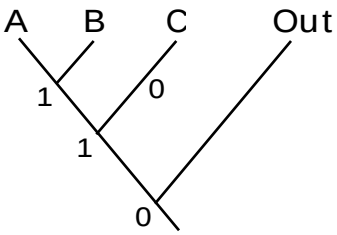
The cladogram for this data set is:

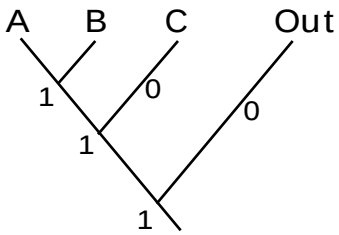


Several different hypotheses for where the changes in character 1 occurred:









The first hypothesis is the most parsimonious and is the preferred hypothesis of where the character state changes occurred.

Using Hennig86 to Optimize Character State Changes

The character state changes that occur on the internal branches of a cladogram can b determined with Hennig86 by giving the command

```
xsteps h;
```

Lets return to the file called **diagnosis.dat:**

```
xread
'Data to demonstrate diagnosis options'
6 7
Out 000000
A   100001
B   110001
C   111001
D   111101
E   111111
F   111110
;
proc/;
```

To use Hennig86 to construct a tree of this data and to optimize the changes of the characte states, type:

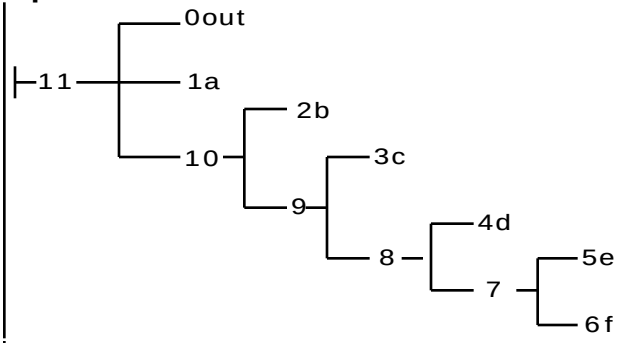
```
*>ss
*>p b:diag1.dat;
*>log b:diag1.out;
*>ie;tp;
*>xsteps h;
*>log/;
```

It isn't necessary to save the output in a log file, but it is easier to examine the results.

NOTE: If you type an uppercase **H** instead of a lower case **h**, then the length of the tree will displayed followed by the execution of the **hennig;** command.

Exit Hennig86 and examine the log file with a text editor or word processor:

```
ie length 7 ci 85 re 85 trees 1
tplot file 0 from ie 1 tree
```




```
xsteps file 0 from ie 1 tree
tree 0

character 0
 7  8  9  10  11
```

```
 1  1  1   1   1
|
|character 1
 7  8  9  10  11
 1  1  1   1   0
|
|character 2
 7  8  9  10  11
 1  1  1   0   0
|
|character 3
 7  8  9  10  11
 1  1  0   0   0
|
|character 4
 7  8  9  10  11
 1  0  0   0   0
|
|character 5
 7  8  9  10  11
 1  1  1   1   1
```

What does this mean?:

```
  character 0
 7  8  9  10  11
 1  1  1   1   1
```

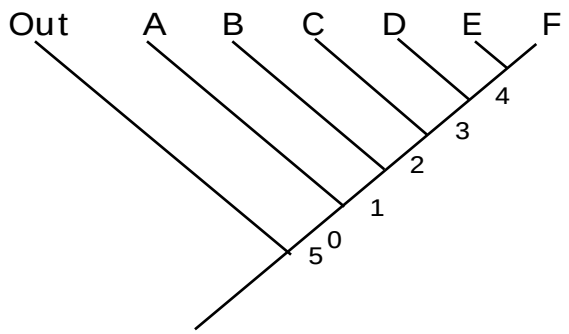
- For the first character of the data set, it is most parsimonious conclude state 1 is at nodes 7 through 11.

Here again, the first character is referred to as **character 0**. The solution is simple if you don't like it: Insert a dummy character as the first character. For this dummy characte make the states 0 for all taxa and the outgroup. Such a character will not affect the length, grouping of taxa, consistency index, etc. of the cladogram. Simply ignore character 0 in the output and character 1 now refers to the first character in your data chart.

- For the second character of the data set, at nodes 7-10, it is state 1, but at node 11 it is 0.

```
character 1
 7  8  9  10  11
 1  1  1   1   0
```

This information corresponds to mapping the advanced state of the characters on the tree i the following way:



Notice that the changes that occur in the terminal branches are not revealed with the command **xs h;**. You must return to the original data set and map these on by hand.

All **xstep** commands can be executed at the same time - the options do not need to b listed separately**.** They can be invoked in various combinations**:**

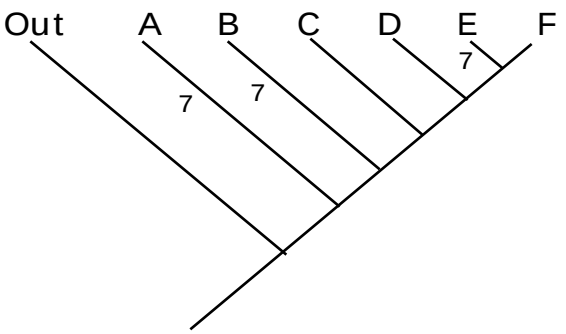
```
xsteps hl;
xsteps hc;
xsteps cl;
xsteps hcl;
```

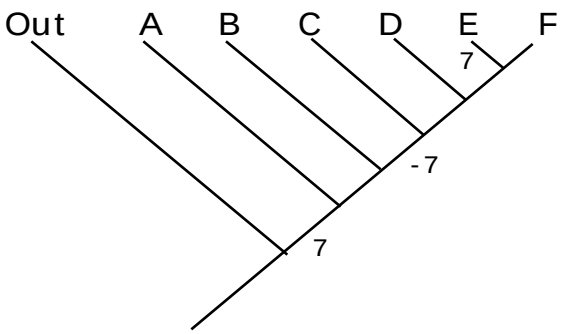
NOTE **-** they must be typed in lowercase letters. If you type **xs HCL;** by mistake you will get the length of the tree followed by the message **hcl ?**.

Sometimes, more than one state can be optimized on a branch:

```
Out 0000000
A   1000001
B   1100011
C   1110010
D   1111010
E   1111111
F   1111110
```

When optimizing character 7 onto the resulting cladogram, two different but equally parsimonious possibilities are found:





If this data set is analyzed with Hennig86, the output from the **xs h;** command reflects all of these possibilities for character 7:

```
xsteps file 0 from ie 1 tree
tree 0

character 0
 7  8  9  10  11
 1  1  1   1   1

character 1
 7  8  9  10  11
 1  1  1   1   0

character 2
 7  8  9  10  11
 1  1  1   0   0

character 3
 7  8  9  10  11
 1  1  0   0   0

character 4
 7  8  9  10  11
 1  0  0   0   0

character 5
 7  8  9  10  11
 1  1  1   1   1

character 6
 7  8  9   10   11
 1  1  1   0.1  0.1
```

- this means that this character is either 0 or 1 at the internal branch labeled 10, and at the internal branch labeled 11. The period separates the two possibilities - it is not intended to be a decimal point.

DOS EQUIS

The distribution of characters on a tree can be examined using the tree editor, DOS EQUIS. This tree editor also allows characters and trees to be interactively modified.

To see how Dos Equis works, create the cladogram for the file **sample.dat**:

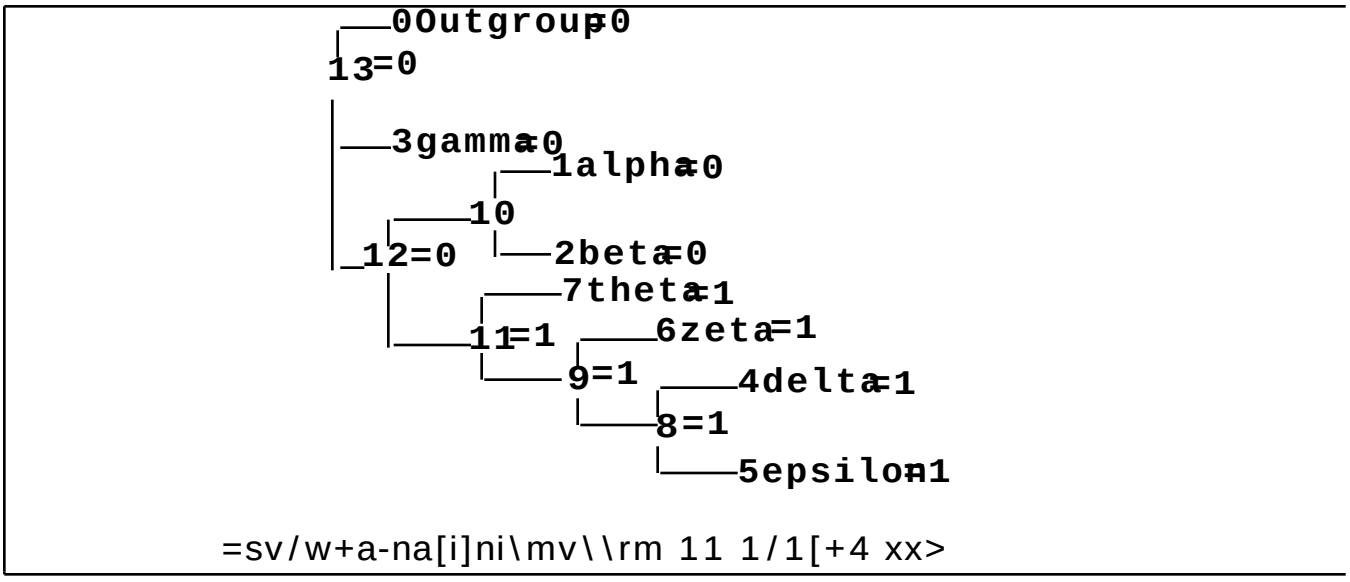
```
C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                  All rights reserved
      This copy produced for the exclusive use of
                       Jane Doe

                      *>p a:sample.dat
procedure a:sample.dat *
xread
title (e.g., data file for basic cladistic procedures)
procedure --
*> ie;

ie length 11 ci 90 ri 90 trees 1

     tplot file 0 from hennig 1 t

             ___0Outgrou
   ____13___|
            |______3gamm
            |                   _____2beta
            |          _____10 |_____1alpha
            |_____12__|         _____7theta
                      |        |           ____6zeta
                      |        |          |          ____4delta
                      |_____11 |_____ 9   |_____8___|_____5epsilon

          *>
```

At the prompt type the command **xx**:

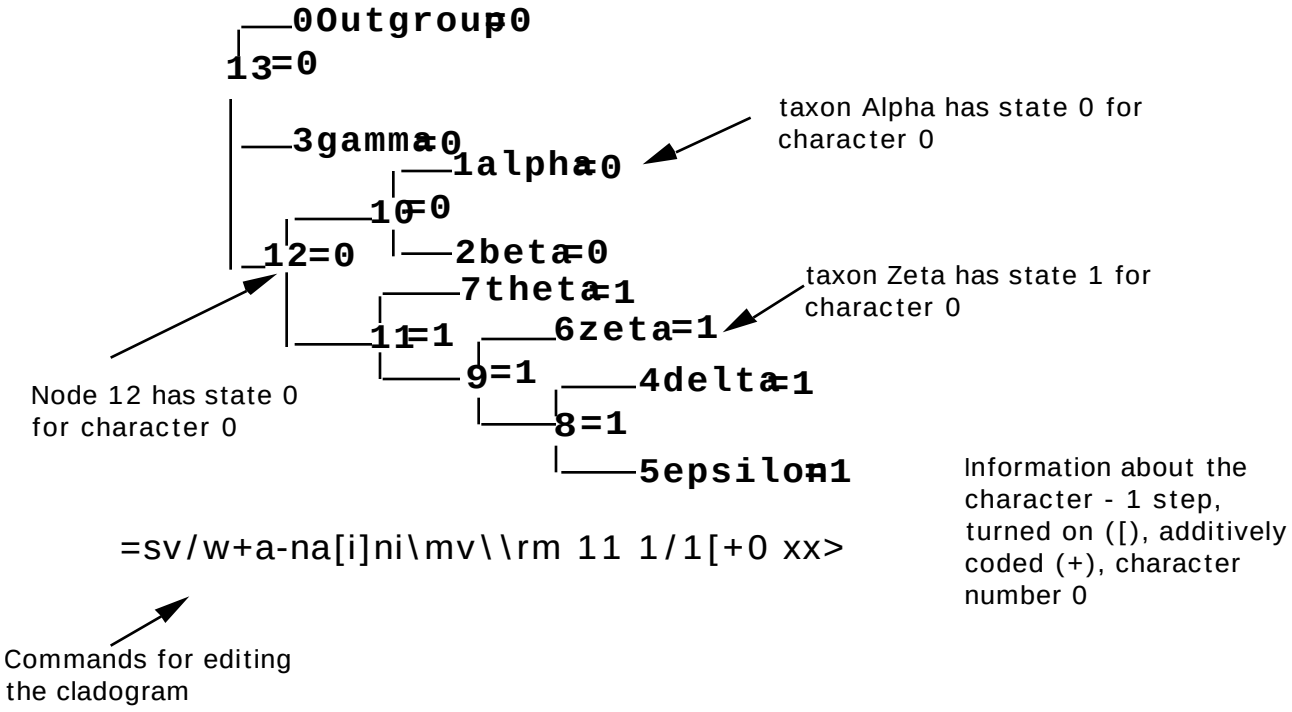
```
*>xx
```

Note: DO NOT end this command with a semicolon.

On the monitor you will see:



The last line shows the control characters that can be used to control the tree editor. What other information is in the tree shown on the screen?



Using The Control Characters- When The Tree Is Not Longer Than The Screen

The commands that can be entered at the end of the prompt line are:

1. **Character Number** typing the number for a character will cause

	the information for that character to be displayed
2. **/(followed by number between 1 & 99)**	set the weight of the current character to the number following the slash.
3. **+** (plus sign)	makes the character additive (only affects multistate characters)
4. - (dash)	makes the character nonadditive (only affects multistate characters)
5. **[** (bracket)	turns a character on
6. **]**	turns a character off; when a character is turned off, a new tree is not calculated unless Dos Equis is exited and a cladogram generating command is given. However, the number of steps in the tree will be changed.
7. **\\ (followed by a branch number)**	deletes a branch
8. **\ (branch #1) (branch #2)**	moves branch number 1 to branch number 2.
9. **=** (equal sign)	save the modified tree and new character codings in a file and exit to Hennig86 prompt.
10. **;** (semicolon)	exit to Hennig86 prompt without saving modified tree.

If when any of these commands are given, the command is repeated and you are immediately returned to Hennig86, then you typed a semicolon at the end of the comman ***>xx**. Reenter Dos Equis without typing a semicolon.

Multiple commands can be given on a single line:

```
...xx>/5 \9 10 \\8
```

This will cause the current character to be weighted 5 times, unite branches 9 and 10, and delete branch 8.

Using The Control Characters- When The Tree Is Longer Than The Screen

The commands that can be entered are the same as given above except:

1. **Character Number**

Multiple Trees

When there are many equally parsimonious trees, the Dos Equis command will onl display the first tree that was generated. To see other trees they can be saved in a tree file ar brought in one at a time. The following procedure will do this:

```
C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                  All rights reserved
      This copy produced for the exclusive use of
                        Jane Doe

*>p a:multiple.dat
procedure a:multiple.dat *
xread
title
procedure --
*> ie;


     ie length 9 ci 88 ri 83 trees 2

*>tplot;
```

The output that then appears on the monitor is:

```
     tplot file 0 from ie 2 trees


     tree 0
                ________________0Outgroup
      ____9__|            ______4d
             |_____8__|______5e
             |             ______3c
             |_____7__|            _____1a
                      |________6__|_____2b
     tree 1
                ________________0Outgroup
      ____9__|            ______1a
             |_____7__|______2b
             |             ______3c
             |_____8__|            _____4d
                      |________6__|_____5e
```

These two trees can be saved in a file (called here tree.file) in parenthetical notation with th

following command:

```
*>tsave tree.fil;
```

The first tree can be examined with dos equis immediately by typing **xx;**.

When ready to examine the second tree, choose that tree with the command:

```
*>tchoose 1;
```

Remember - the first tree is called tree 0, the second tree 1, and so forth.

Now when you give the **xx;** command, the second tree can be examined.

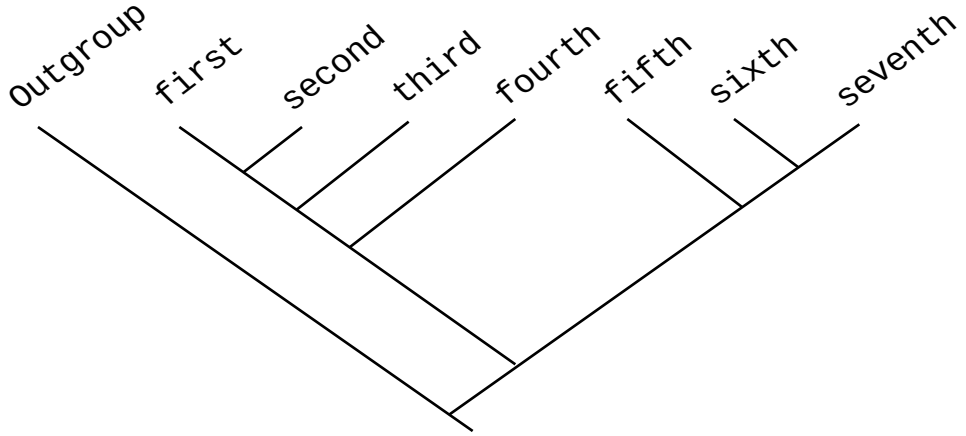
To return to the first tree, bring all of the trees back into memory by calling up the saved tre file:

```
*>p tree.fil
```

Inputting Trees Not Generated with Hennig86

It is sometimes useful to be able to diagnose a tree not obtained with Hennig86. Thi might be a tree published in the literature, one obtained with another program such as PAU or PHYLIP, or you may just want to scrutinize your own preconceived notions.

The **tread** option of Hennig86 allows you to input a tree and diagnose it in relationship to a character data file. Let us start with a simple example: Suppose a phylogenetic tree of seven taxa was published in 1972 by another systematist using traditio synthetic methods. This tree has become generally accepted and appears in many basic textbooks --



Since the publication of this phylogenetic tree, systematists have been reexamining these ta
and have gathered new character data on them:

	Characters									
Taxa	1	2	3	4	5	6	7	8	9	10
Outgp	0	0	0	0	0	0	0	0	0	0
First	1	1	1	1	0	0	1	0	0	0
Second	1	1	1	1	0	0	0	0	1	0
Third	1	0	0	1	0	0	0	0	0	0
Fourth	1	0	1	1	0	0	0	1	0	0
Fifth	1	0	0	0	1	0	0	0	0	0
Sixth	1	0	0	0	1	1	0	0	0	1
Seventh	1	0	0	0	1	1	0	0	0	0

To determine how well this tree matches the new data, create two files. First, create
data file with the characters. In this example, this file is referred to later as **new.dat**.

```
xread
'New data on the genus x'
10
8
Outgroup  0  0  0  0  0  0  0  0  0  0
First     1  1  1  1  0  0  1  0  0  0
Second    1  1  1  1  0  0  0  0  1  0
Third     1  0  0  1  0  0  0  0  0  0
Fourth    1  0  1  1  0  0  0  1  0  0
Fifth     1  0  0  0  1  0  0  0  0  0
Sixth     1  0  0  0  1  1  0  0  0  1
Seventh   1  0  0  0  1  1  0  0  0  0
;
proc/;
```

Second, make a tree file (in this example called **intree.dat**):

The first line of the tree file is simply 'tread', which tells the program that this is a file that
contains a tree.

The second line is an optional title:

```
 tread
'Tree of genus x published in 1972'
```

The third line is the tree.
Trees can be represented in many different ways (see Options section); for this example we
will use a standard parenthetical notation:

1. Terminal taxa are represented by numbers. The number is
determined by their order in the data file. For this example --

```
Outgroup -   0
First -     1
Second -    2
Third -     3
Fourth -    4
Fifth -     5
Sixth -     6
Seventh -   7
```

2. Parentheses enclose each monophyletic group. In the tree, taxa First and Second form a monophyletic group -

(1 2)

This group with the taxon Third forms a higher monophyletic group-

((1 2)3)

And so forth. The entire notation for the tree is -

(0((((1 2)3)4)((5 6)7)))

3. The line describing the tree ends in a semicolon:

(0((((1 2)3)4)((5 6)7)));

NOTE: If the tree is large, the parenthetical notation may be entered on several lines. The semicolon signals the end of the input.

The final tree file should read:

```
tread
'Tree of genus x published in 1972'
(0((((1 2)3)4)((5 6)7)));
proc/;
```

Analysis - Enter the Hennig86 program and call up the character data file with the procedu command:

```
C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                    All rights reserved
          This copy produced for the exclusive use of
                        Jane Doe
          *>

          *>p a:new.dat;

          procedure a:tree.dat *
          xread
          My data on the genus x
          procedure --
          *>
```

The data have now been read by the program. Now call up the tree file:

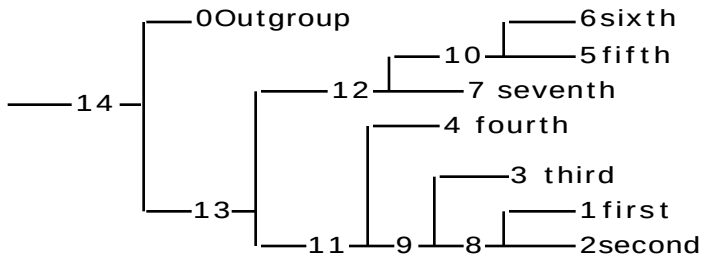
```
*>p a:intree.dat;

procedure a:intree.dat *
tread
tree of genus x published in 1972
procedure --
*>
```

At this time, I always look at the tree to make sure I made no mistakes with the parenthetic notation:

```
*> tplot;
```

If everything was done correctly, the monitor should display:

tplot file 0 from tread 1 tree



The tree can now be evaluated using the diagnosis commands:

```
*>xs hcl;

xsteps file 0 from tread 1 tree
tree 0
character 0
   8   9  10  11  12  13  14
   1   1   1   1   1   1 0.1
character 1
  8  9 10 11 12 13 14
  1  0  0  0  0  0  0
character 2
   8   9  10  11  12  13  14
   1 0.1   0 0.1   0   0   0
character 3
  8  9 10 11 12 13 14
  1  1  0  1  0  0  0
character 4
  8  9 10 11 12 13 14
```

```
      0  0  1  0  1  0  0
    character 5
       8   9  10  11  12  13  14
       0   0 0.1   0 0.1   0   0
    character 6
      8  9 10 11 12 13 14
      0  0  0  0  0  0  0
    character 7
      8  9 10 11 12 13 14
      0  0  0  0  0  0  0
    character 8
      8  9 10 11 12 13 14
      0  0  0  0  0  0  0
    character 9
      8  9 10 11 12 13 14
      0  0  0  0  0  0  0
    tree 0 length 12 ci 83 ri 77
    character/steps/ci/ri
       0   1   2   3   4   5   6   7   8   9
       1    1   2   1   1   2   1   1   1   1
    100 100  50 100 100  50 100 100 100 100
    100 100  50 100 100   0 100 100 100 100

    tree/length
        0
        12
```

In other words, this tree requires a convergence in character 2 and in character 5.

Options

1. The data file and the tree file can be combined. Analysis of the following file gives the same result as reading in the two files separately.

```
xread
'My data on the genus x'
10
8
Out       0  0  0  0  0  0  0  0  0  0
First     1  1  1  1  0  0  1  0  0  0
Second    1  1  1  1  0  0  0  0  1  0
Third     1  0  0  1  0  0  0  0  0  0
Fourth    1  0  1  1  0  0  0  1  0  0
Fifth     1  0  0  0  1  0  0  0  0  0
Sixth     1  0  0  0  1  1  0  0  0  1
Seventh   1  0  0  0  1  1  0  0  0  0
;
tread
(0((((1 2)3)4)((5 6)7)));
proc/;
```

2. The parenthetical notation can be entered from the keyboard while in Hennig86:

```
                    C:\->ss

Hennig86 Version 1.5 Copyright (c) James S. Farris 1988
                       All rights reserved
         This copy produced for the exclusive use of
                           Jane Doe
         *>

         *>p a:tree.dat;

         procedure a:tree.dat *
         xread
         My data on the genus x
         procedure --
         *>
```

The data have now been read by the program. Now input the tree:

```
*>tread (0((((1 2)3)4)((5 6)7)));
```

If the tree is large, the parenthetical notation may be entered on several lines. The semicol signals the end of the input.

3. Alternative Ways of Describing Trees

A. The trees can be represented in parenthetical notation but with the actual taxa names instead of numbers:

```
(0((((1 2)3)4)((5 6)7)));
```

is the same as:

```
(Outgroup((((First Second)Third)Fourth)((Fifth Sixth)Seventh)))
```

i. The taxa names can be abbreviated as desired - they simply need to remain recognizable:

```
(Outgroup((((First Second)Third)Fourth)((Fifth Sixth)Seventh)))
```

is the same as:

```
(Out((((Fir Sec)Th)Fo)((Fif Six)Sev)));
```

But, if abbreviated too much, some of the taxa names become indistinguishable:

```
(O((((F S)T)F)((F S)S)));
```

NOTE: YOU WILL GET NO ERROR MESSAGE IF YOU MAKE THIS MISTAKE.

ii. Taxa names and numbers can be used together:

```
(0((((1 2)3)4)((Fifth Sixth)Seventh)));
```

B. Instead of ( ) to enclose monophyletic groups, many sorts of delimiters can be use
{ } [ ] or \ /

This allows you to chose a delimiter that is convenient for your keyboard.
All of the following represent the same tree -

```
{0{{{{1 2}3}4}{{5 6}7}}};
 [0[[[[1 2]3]4][[5 6]7]]];
 \0\\\\1 2/3/4/\\5 6/7///;
```

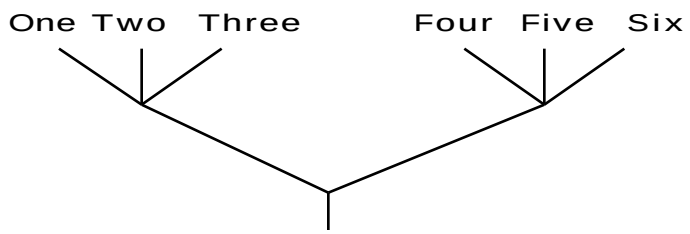
i. Types of delimiters can be mixed, but they vary in priority

	Priority
1	( )
2	\ /
3	[ ]
4	{ }

```
{0{[\(1 2)3/4]\(5 6)7/}};
```
-- You may find that the monophyletic groups are easier to pick out in this mixed notation.

4. Unresolved Trees

Unresolved trees are accepted. Simply don't separate the taxa by delimitors. For example, the tree:

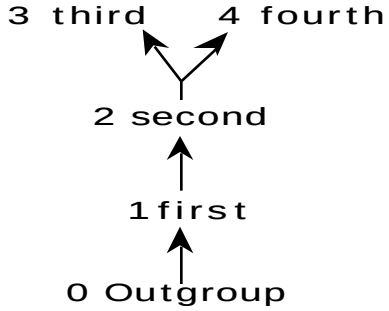


Would be noted as:

```
(Out((1 2 3)(4 5 6)));
```

5. Ancestors

Some traditional trees relate the taxa as sets of ancestors and descendants:



Such trees can be put into Hennig86. By placing a period (.) before the taxon name c number, it is designated as an ancestor. Descendant taxa are preceded by a dash and are separated from each other with slashes \. Commas divide the tree into manageable parts. The following lines describe the tree:

```
*>tread .0 -1\2\3 4,
 .1 -2\3 4,
 .2 -3\4;
```

Character Analysis

Controlling Characters

Turning Characters On and Off

In analysis of real data, systematists often wonder how much a specific character affe a cladogram or wish to compare two different sets of characters. Of course, you could write several different data files each with a different combination of characters. But with Hennig86 there is an easier way.

The command **cc;** reveals the current settings of the characters in a data set and allo us to turn characters on and off, weight characters, and to nonadditively order multistate characters.

Try this for the file **sample.dat**:

```
*>ss
*>p a:sample.dat;
*>cc;
```

The computer responds with:

```
ccode
   0    1    2    3    4    5    6    7    8    9
 1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[
```

What does this mean? Consider just the first entry in the line:

```
   0      <-- character number[2]
 1+[      <-- it has a weight of 1, is ordered
           additively (+), and on ([).
```

The top line indicates the character

The number 1 in the bottom line gives the weight of the character. In the sample.da file all characters are weighted equally.

The + (plus sign) indicates that the character is being additively ordered. Since only

---

[2]The first character is referred to as character 0. If you think of it as character 1 and find this "renumbering" irritating, insert a dummy character as the first character. For this dummy character, make th states 0 for all taxa and the outgroup. Such a character will not affect the length, grouping of taxa, consistenc index, etc. of the cladogram. Simply ignore character 0 in the output and character 1 now refers to the first character in your data chart.

multistate characters can be ordered, this is discussed in the section on multistate character

The [(left bracket) indicates that the character is on and will be used to construct the tree.

To turn off a character so that it will not be used in constructing the cladogram, type

```
*>cc ] <the number of the character>;
```

The character numbers can each be listed separately and separated by a space:

```
*>cc ] 2 4 5 6;
```

or a comma:

```
*>cc ] 2,4,5,6;
```

or sequential states can be indicated by a range in which the lowest number and highest number are separated by a period (.) (Do not use a dash - [1]):

```
*>cc ] 2 4.6;
```

The space between the bracket and the number is not necessary:

```
*>cc ]2 4.6; and *>cc ] 2 4.6;
```

both turn off characters 2, 4, 5 & 6.

NOTE: If you type a character number that is greater than the number of characters in the d set, the command will not be executed for any of the characters and you will receive the err message:

```
*>cc ] 34;
ccode 34 ?
*>
```

Try now turning off characters 2, 8 and 9 in the data set **sample.dat**:

```
*>ss
*>p a:sample.dat;
*>cc ] 2 8 9;
*>cc;
```

The monitor shows:

[1]If a dash is used only the lowest and highest numbers in the range will be turned off and the highest number will be ordered nonadditively.

```
ccode
   0    1    2    3    4    5    6    7    8    9
 1+[  1+[  1+]  1+[  1+[  1+[  1+[  1+[  1+]  1+]
```

Notice that under characters 2, 8 and 9, the bracket is turned around - this indicates that the characters are turned off. A cladogram can now be constructed with the data set using only characters 0-1 & 3-7 by giving a command such as **mh; h;** or **ie;**:

```
*>ie;tp;
ie length 7 ci 100 ri 100 trees 1
tplot file 0 from ie 1 tree
          ┌──0Outgrou
     ┌──12──┤
     │    └──3gamma
     │          ┌──2beta
     │       ┌──9──┤
     │       │  └──1alpha
     │    ┌──11──┐
     │    │      │       ┌──6zeta
     │    │      │    ┌──8──┤
     │    │      └──10──┘  └──4delt
     │    │             │
     │    │             └──5epsilon
```

To turn a character back on so that it will not be used in constructing the cladogram, reverse the direction of the bracket:

***>cc [<*the number of the character*>;**

To turn all the characters on or off use a **.**(period) in place of the character numbers:

```
*>cc [ .;
*cc;
ccode
   0    1    2    3    4    5    6    7    8    9
 1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[


*>cc ] .;
*cc;
ccode
   0    1    2    3    4    5    6    7    8    9
 1+]  1+]  1+]  1+]  1+]  1+]  1+]  1+]  1+]  1+]
```

Weighting Characters

There are many arguments both for and against weighting characters. Some maint

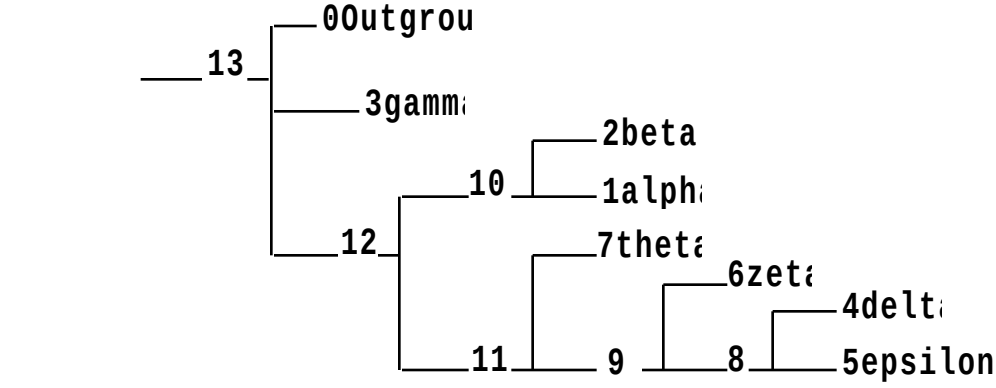
that important characters deserve more weight, while their critics argue that their is no objective way to distinguish important from unimportant features. Some systematists wis to weight data from one source to prevent its information from being overwhelmed by a more numerous form of characters (e.g., when combining morphological with molecular information). While others argue such data sets should be analyzed separately.

If weighting can be justified, then characters can be given weights with the Hennig8 program using the command:

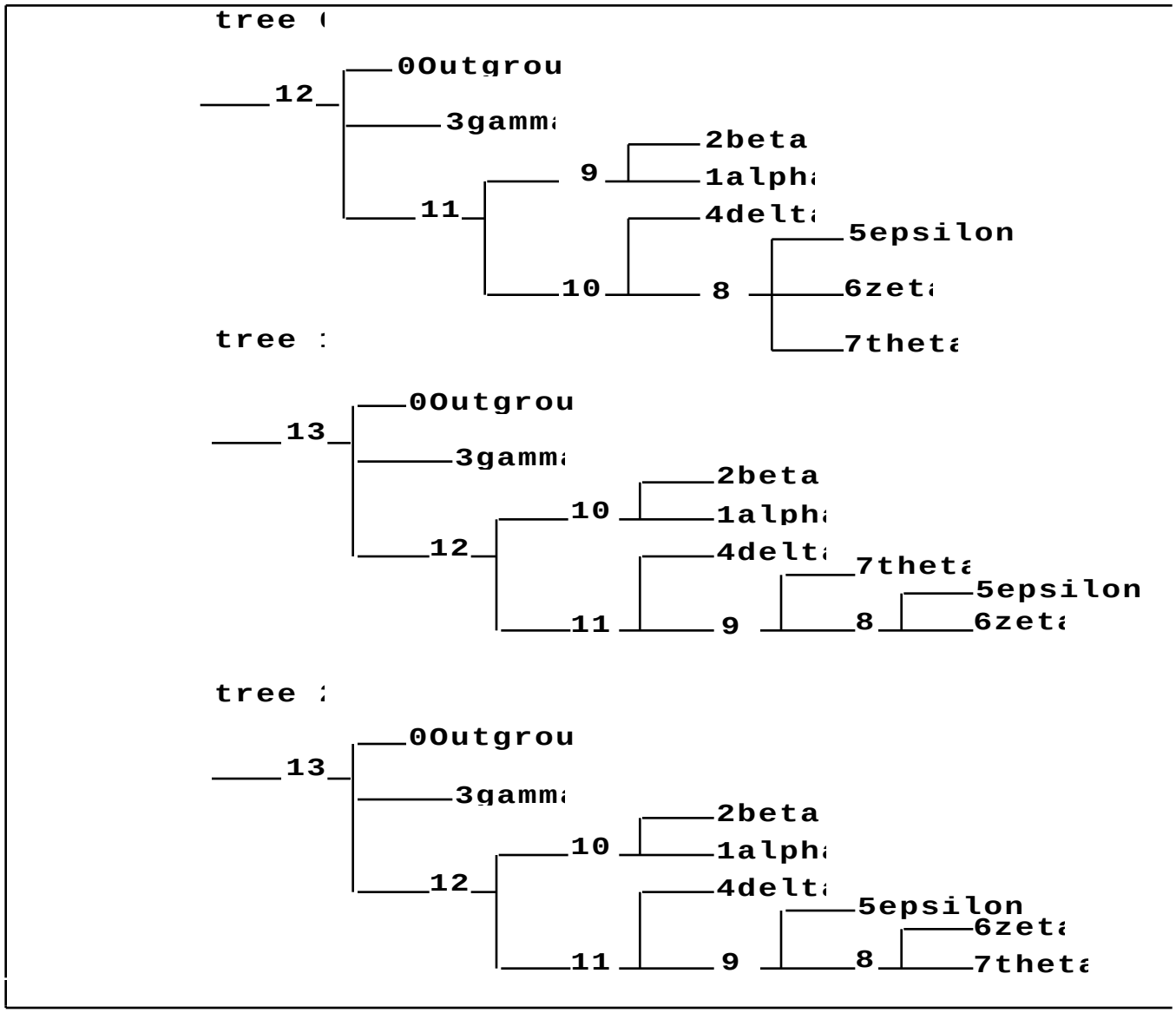
```
*>cc /<new weight> <character number>;
```

The weights can range from 1 to 100. If in the data set **sample.dat**, character number 10 is given a weight of 99, transforming from state 0 to state 1 is not counted as one step but as 9 steps. This will outweigh characters 2 and 3 and will give new trees

```
*>p a:sample.dat;
procedure a:sample.dat *
xread
title
procedure --

*>cc;
ccode
   0    1    2    3    4    5    6    7    8    9
 1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[
*>ie;tp;
ie length 11 ci 90 ri 90 trees 1
tplot file 0 from ie 1 tree
```




```
*>cc /99 9;
*cc;
ccode
   0    1    2    3    4    5    6    7    8    9
 1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[ 99+[

*>ie;tp;
ie length 110 ci 98 ri 99 trees 3
tplot file 0 from ie 3 tree
```



To reset the weight of character 9 back to 1 type either:

```
>*cc /1 9;
```

or

```
*>cc /. 9;
```

Turning Characters Off and On, and Weighting with a Single cc Command

Several cc options can be given on a line if they are separated by an asterisk (*****). For example:

```
*>cc ] 1 2 * /5 9;
```

Will turn off characters 1 and 2 and will give character 9 a weight of 5:

```
          *cc;
          ccode
        0     1     2     3     4     5     6     7     8     9
      1+[   1+]   1+]   1+[   1+[   1+[   1+[   1+[   1+[   5+[
```

Saving **cc** Settings

Options used with the **cc;** command can be saved in **ckeep** files. The ckeep files are n named but given numbers between 1 and 9. The files are recalled with the command **cget** followed by the number of the file.

For example:

```
                *>p a:sample.dat;
                procedure a:sample.dat *
                xread
                title
                procedure --
                *>cc;

      ccode
         0     1     2     3     4     5     6     7     8     9
       1+[   1+[   1+[   1+[   1+[   1+[   1+[   1+[   1+[   1+[

                *>ckeep 1;
                *>cc ]1 3;
                *>cc;

      ccode
         0     1     2     3     4     5     6     7     8     9
       1+[   1+]   1+[   1+]   1+[   1+[   1+[   1+[   1+[   1+[

                *>ckeep 2;
                *>cc /5 7;
                *>cc;

      ccode
         0     1     2     3     4     5     6     7     8     9
       1+[   1+]   1+[   1+]   1+[   1+[   1+[   5+[   1+[   1+[

           *>ckeep 3;
                *>cget 1;
                *>cc;

      ccode
```

```
   0    1    2    3    4    5    6    7    8    9
 1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[  1+[

            *>cget 2;
            *>cc;

ccode
   0    1    2    3    4    5    6    7    8    9
 1+[  1+]  1+[  1+]  1+[  1+[  1+[  1+[  1+[  1+[

            *>cget 3;
            *>cc;

ccode
   0    1    2    3    4    5    6    7    8    9
 1+[  1+]  1+[  1+]  1+[  1+[  1+[  5+[  1+[  1+[
```

NOTE - **ckeep** files are not permanently written to the disk and so are lost when you exit the Hennig86 program with the command **yama**.

Outgroups

The first taxon in a data file is used as the outgroup in Hennig86 by default. There is no need to rescore the states as all 0 (zero) in the outgroup, the program automatically sets state in the outgroup as the pleisiomorphic state. In other words, if a designated outgroup state 1, then state 0 in the ingroup is treated as the apomorphic state and 1 as the plesiomorphic.

For the following data set:

```
xread
10  8
Alpha0 0 0 1 0 1 1 0 1 0
Outgroup  0 0 0 0 1 0 0 0 0 0
Beta      0 0 0 1 1 1 1 0 1 0
Gamma0 0 0 0 1 0 0 0 1 0
Delta1 1 1 0 1 0 1 1 1 0
Epsilon   1 1 1 0 1 0 1 0 1 1
Zeta      1 1 0 0 1 0 1 0 1 1
Theta1 0 0 0 1 0 1 0 1 1
;
```

Alpha is treated as the outgroup and any state it has is considered plesiomorphic.

Another taxon can be designated as the outgroup with the command:

outgroup=<*taxon number*>;

(Remember - The first taxon in the data set is 0, the next is number 1, etc.)
or

outgroup=<*taxon name*>;

If the name is used, it can be abbreviated but must be distinguishable from the other taxa names.
Gamma can be designated as the outgroup by typing:

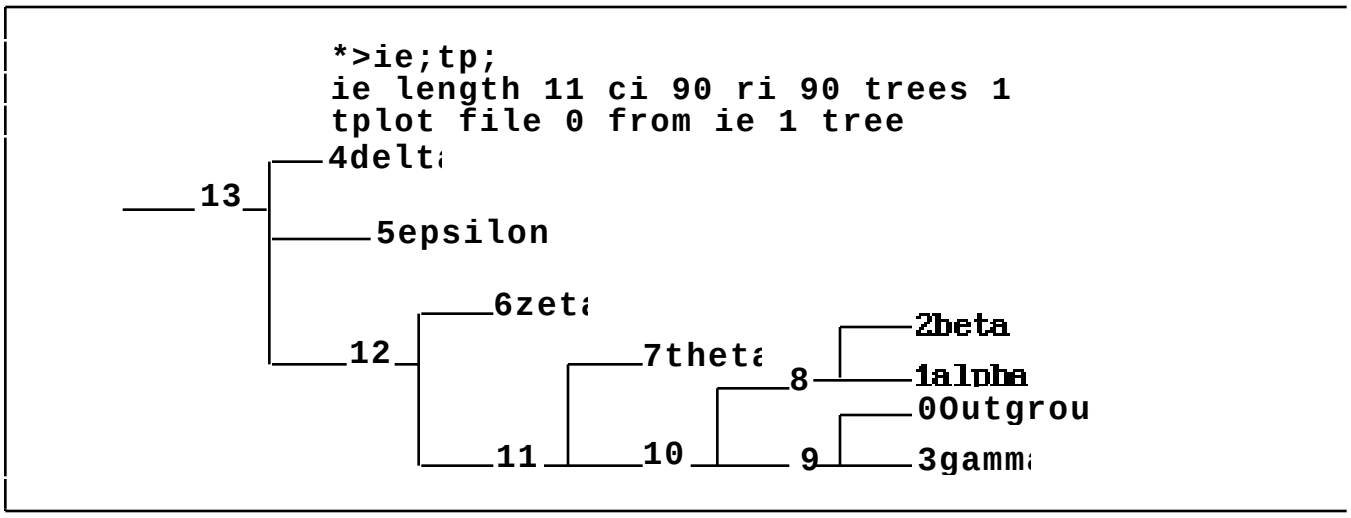
```
*>ss
*>p a:sample.dat;
procedure a:sample.dat *
xread
title
procedure --
*>outgroup=4;
```

or

```
*>outgroup=gamma;
```

or

```
*>outgroup=g;
```



Designating More Than One Taxon as Outgroup with Hennig86

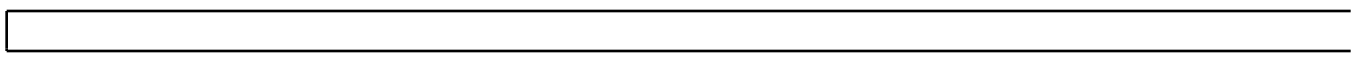
Under ideal conditions, the choice of an outgroup is obvious and, if there is little convergence in the data, usually different outgroups can be used to give the same result.

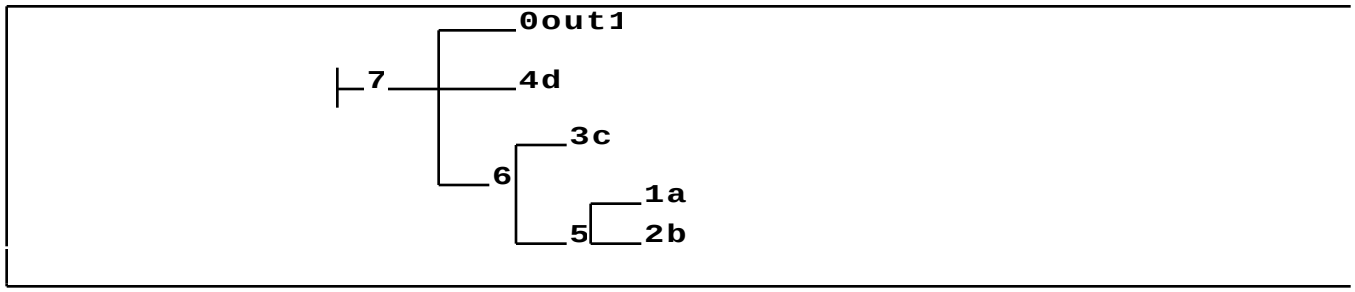
When convergence is suspected between the outgroup and the ingroup or when the monophyly of the ingroup is in doubt, it is advisable to use more than one outgroup.

Consider the following example:

	Characters									
	1	2	3	4	5	6	7	8	9	10
out1	0	0	0	0	0	0	0	0	0	0
A	1	0	0	1	1	1	1	1	1	1
B	1	0	0	1	1	1	1	1	1	1
C	1	1	1	0	1	1	1	1	1	1
D	1	1	1	0	1	1	0	0	0	0

The cladogram for this data is:





However, because other outgroups also have state 1 for characters 7, 8, 9, and 10, there is reason to suspect that these characters are unreliable.

When these outgroups are added to the data set:

```
          Characters
     1 2 3 4 5 6 7 8 9 10
out1 0 0 0 0 0 0 0 0 0 0
out2 0 0 0 0 0 0 1 1 1 1
out3 0 0 0 0 0 1 1 1 1 1
A    1 0 0 1 1 1 1 1 1 1
B    1 0 0 1 1 1 1 1 1 1
C    1 1 1 0 1 1 1 1 1 1
D    1 1 1 0 1 1 0 0 0 0
```

When using multiple outgroups, use the outgroup command and identify all the outgroups by their number or name in the data set. For example, the command:

```
*>outgroup=1 3 4;
```

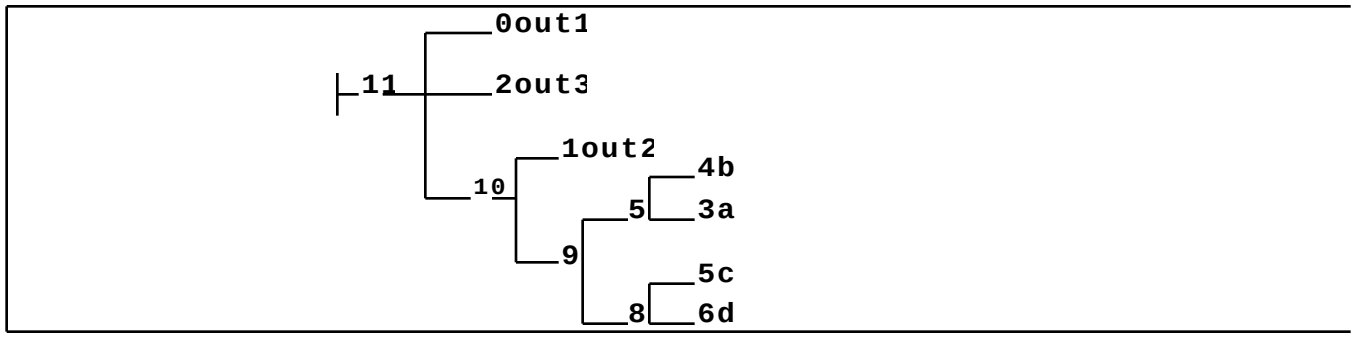
sets taxa 1, 3, and 4 as the outgroups.

When multiple outgroups are used, the root of the tree is placed so that the ingroup monophyletic and the outgroup is paraphyletic[2], provided that is possible.

The cladogram for this data can be generated with the following commands:

```
*>p outgroup.dat;
*>outgroup=0 1 2;
*>ie;tp;
ie length 12 ci 83 ri 71 trees 1
tplot file 0 from ie 1 tree
```

[2]Because the outgroup is considered paraphyletic, the relationships among the outgroup taxa can not be relied on. If you are interested in these relationships, do a separate analysis where these taxa are treated an a ingroup.



Addition of these extra outgroups reveals that taxon D should be united with taxon C.

The command to designate multiple outgroups can be written other ways:

1. If several taxa in sequence are to be designated as outgroups, give the first and last name separated by a period (.). In the example above, the first three taxa in the data set are outgroups:

```
*> outgroup=0.2;   (taxa 0, 1, & 2)
*> outgroup=out1.out3;
```

2. Use the control character [to indicate which taxa are in the ingroup, and the] to indicate which taxa are in the outgroup:

```
*> outgroup=[3 4 5 6 ]0 1 2;
```

(taxa 0, 1, & 2 are outgroups, taxa 3, 4, 5, and 6 are ingroups)

or

```
*> outgroup=[3.6 ]0.2;
```

or

```
*> outgroup=[a.d ]out1.out3;
```

or

```
*> outgroup= ]0.2 [3.6;
```

etc.

3. In order to see which taxa are being used as outgroups, give the command

```
*>outgroup;
```

and they will be listed:

```
outgroup out1 out2 out3
```

Rerooting a Cladogram

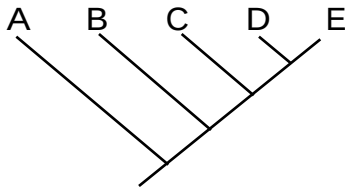
Once a cladogram has been generated, it can be rerooted by using any of the taxa by

designating it with the **out=<*taxonname*>** command and the **reroot** command.

Consider the following data set rooted initially with taxon A:

```
xread
5  5
A 10000
B 11000
C 11100
D 11110
E 11111
;
```

gives the following tree:



To reroot this tree with E as the outgroup:

```
*> out=e;
*> reroot
reroot file 0 from hennig 1 tree
*> tplot

 ┌─── e
 │
 ├── d
 │
 │  ┌─ c
 │  │
 │  │  ┌─ b
 │  └──┤
 │     └─ a
```

Multiple Trees

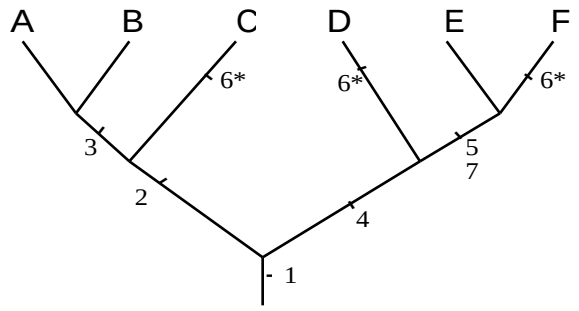
Optimizing Characters

When there are multiple equally parsimonious trees, the states of a character may b optimized onto the different trees differently.

Consider the following example:

```
xread
'Demonstrating More About Multiple Equal Trees'
7  7
Outgroup  0000000
A         1110000
B         1110000
C         1100010
D         1001010
E         1001101
F         1001111
;
proc/;
```

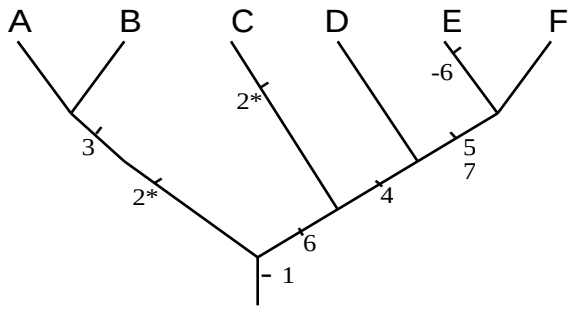
There are 2 equally parsimonious trees for this data set:



Length =9

Character 6 occurs 3 times

OR



Length =9

Character 6 reverses 1 time
Character 2 occurs 2 times

Obviously character 2 fits the first tree very well but has a convergence in the second tree. Therefore, its consistency and retention index will be higher in the first tree. On the other hand, character 6 fits the second tree better than the first

When there is more than one equally parsimonious tree for the data set, it may be useful to know what is the best possible length, consistency index, and retention index for a character. Hennig86 will supply this information with the command:

***>xsteps m;** (or **xs m;**)

For the above data set, the following would be revealed:

```
xsteps file 0 from ie 2 trees
best fits
character/steps/ci/ri
   0   1   2   3   4   5   6
   1   2   1   1   1   2   1
 100 100 100 100 100  50 100
 100 100 100 100 100  50 100

worst fits
character/steps/ci/ri
   0   1   2   3   4   5   6
   1   2   1   1   1   3   1
 100  50 100 100 100  33 100
 100  50 100 100 100   0 100
```

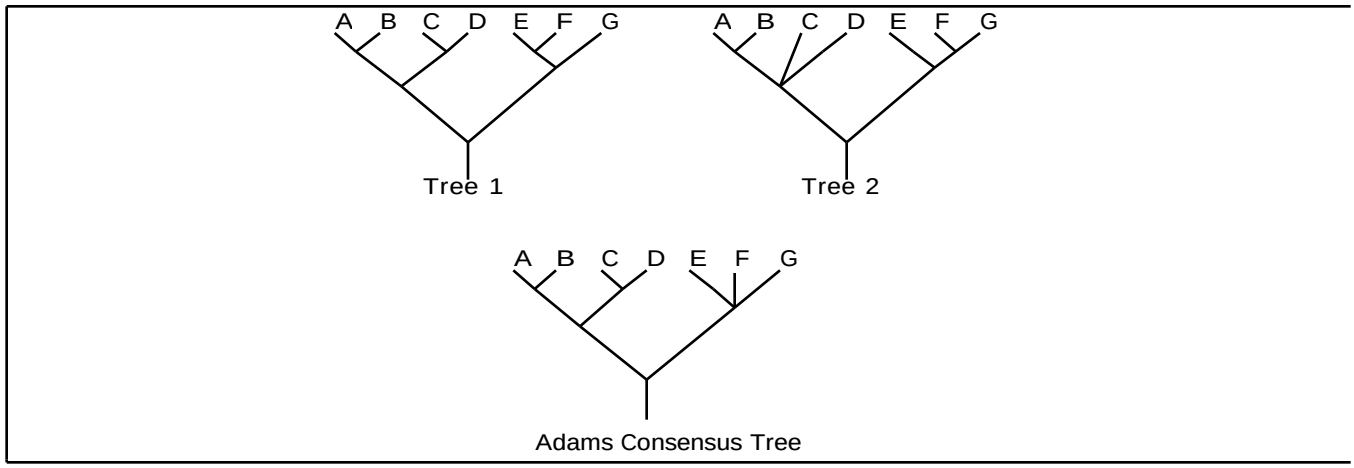
For this data, the best that can be obtained is that character 2 will have no homoplas (ci and ri = 100) or character 6 will have one homoplasy (2 steps instead of 1). In the worst cases character 2 has one homoplasy and character 6 has two homoplasies. All other characters are consistent with both cladograms.

Consensus Trees

In cladistic analysis, multiple equally parsimonious cladograms are often obtained from one data set. How do we handle multiple phylogenetic hypotheses, and at the same time meet the demand for one taxonomic classification? Some workers have argued that a classification should be based only on the components which all cladograms have in comm (e.g., Anderberg and Tehler, 1990). Consensus trees are used to show the information abou taxa relationships that all the equally parsimonious cladograms have in common. There a several ways to form consensus cladograms: Adams, strict, Nelson, and majority rules consensus trees.

Adams consensus cladograms:

The Adams consensus cladogram shows all of the information not in conflict in the multiple tree. Adams consensus trees essentially work by collasping branch points with different possitions in the alternative cladograms to the first branch point (or node) summarizing the different positions:

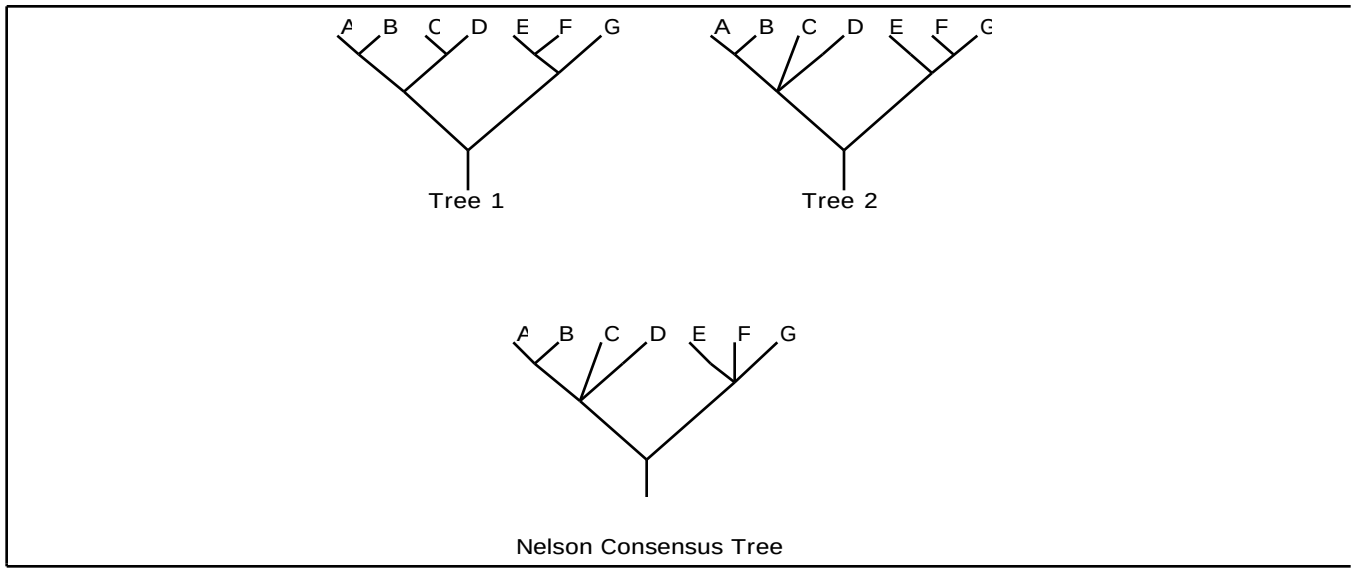


Tree 1 placed taxon C with taxon D, but Tree 2 was not resolved about the placemen taxon C. In forming the Adams consensus tree, taxon C is united with D because this information is not contradicted by Tree 2. On the other hand, Tree 1 places taxon F with ta E, but Tree 2 places taxon F with taxon G. The Adams consensus shows the only nonconflicting information - that taxa E, F, and G arise together but the exact relationship o the three taxa is not in agreement.

Nelson and Strict Consensus Trees

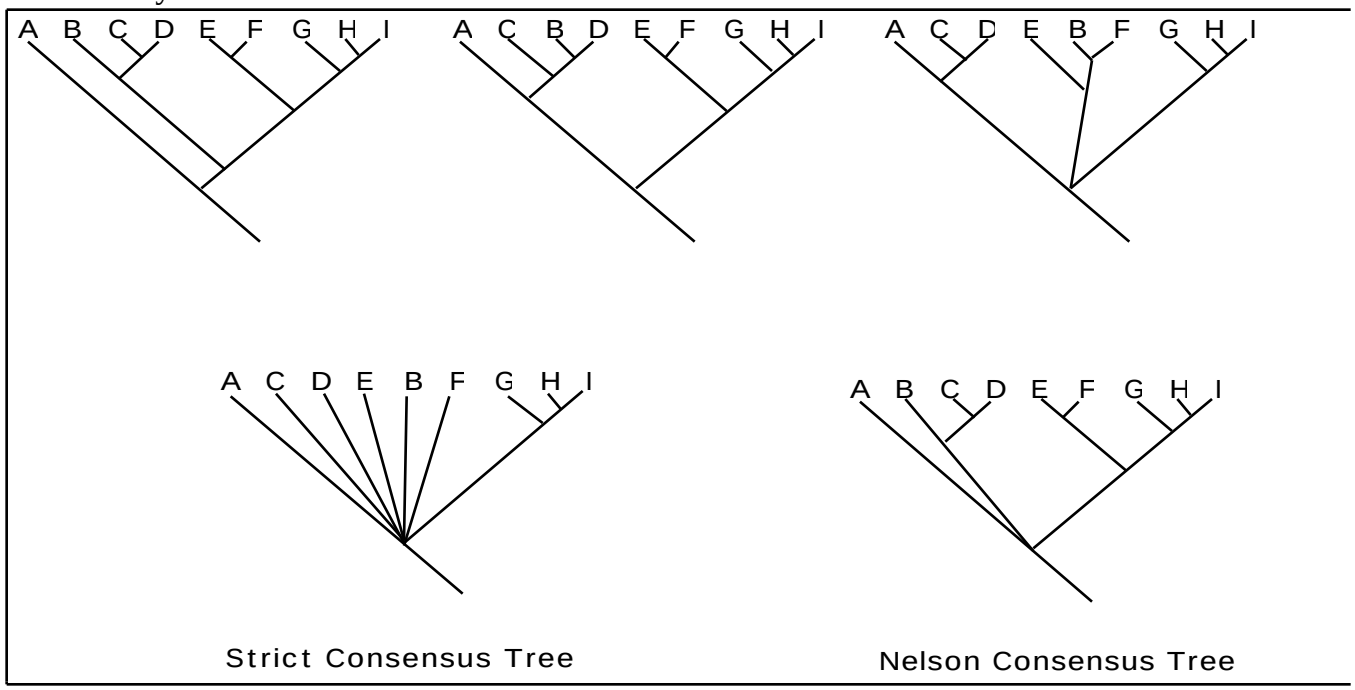
Nelson and strict consensus trees are often mistakenly thought to be the same type consensus. The former were originally described by Nelson (1979) and named strict consen trees by Sokal and Rohlf (1981). Page (1989) showed that, when there are more than two equally parsimonious cladograms, the technique described by Nelson may give different results than that described by Sokal and Rohlf.

When their are just two cladograms, Nelson and strict consensus trees only show t relationships agreed on by all multiple trees:



The only agreement between the two trees is that A and B are sister taxa, Taxa A, B, C and D are sister taxa, and that E, F, and G are sister taxa.

When there are more than two trees, Nelson and strict consensus can give different results. The strict consensus tree contains only those branches occurring in <u>all</u> of the cladograms. Nelson trees combine replicated branches that are presented in at least 2, but not necessarily all of the trees:



Hennig86 can be used to generate strict consensus trees from multiple trees with the command:

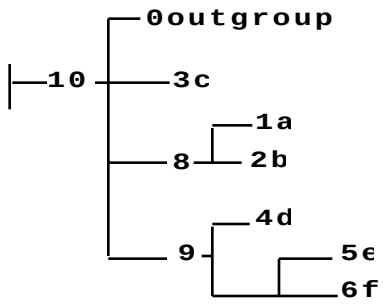
```
*>nelsen;
```

or

```
*>n;
```

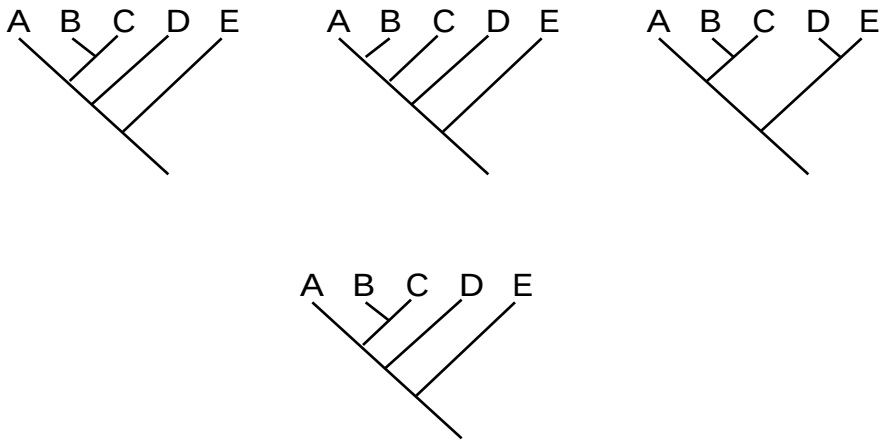
Try this now with the data set shown on page 74. You should get these results:

```
*>nelsen;
nelsen file 0 from ie 2 trees
*>tp;
tplot file 0 from nelsen 1 tree
```



Majority Rule Consensus

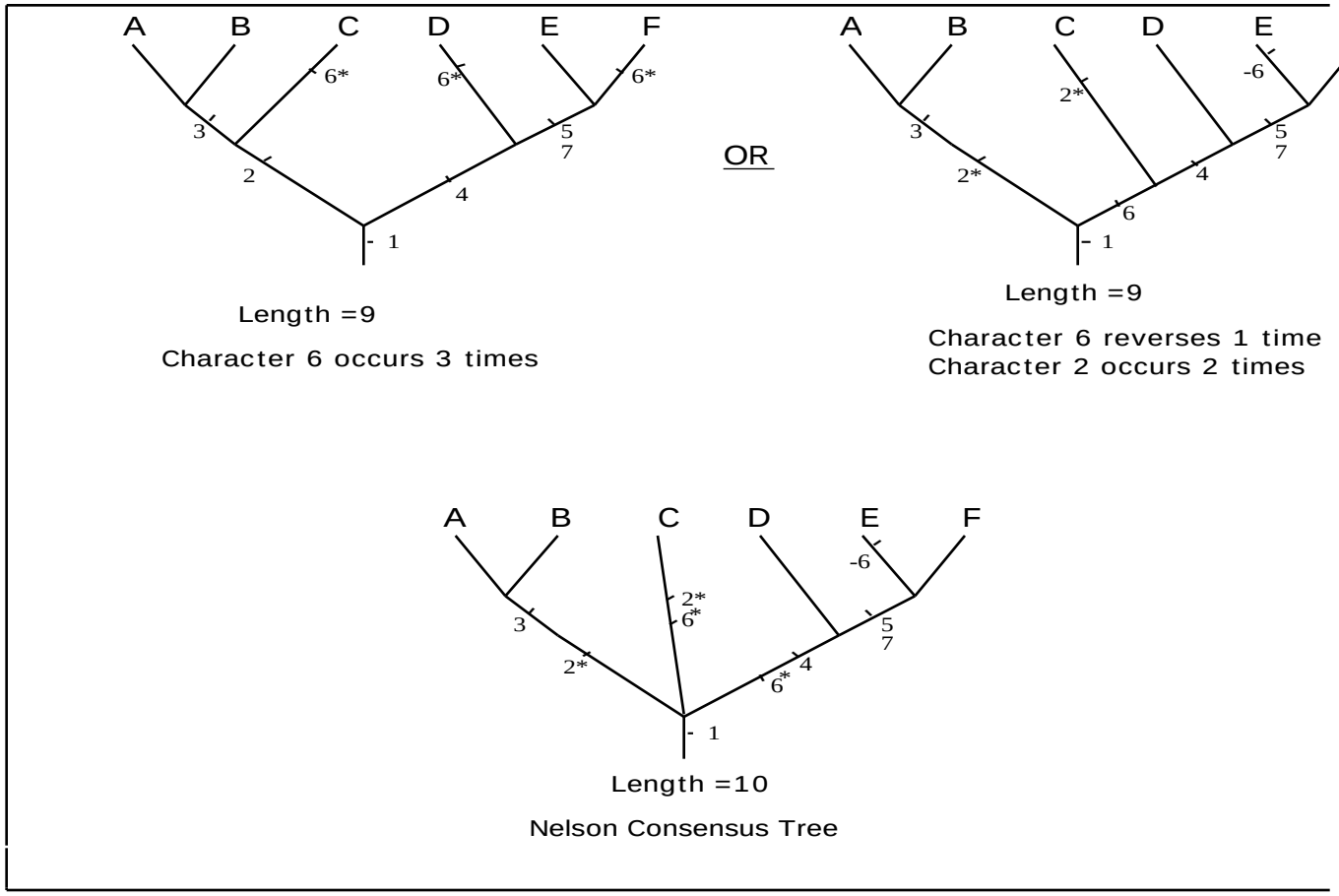
Majority rules consensus trees contain only those branches that occur in more than half of the cladograms (Margush and McMorris, 1981). Therefore, this method can only be used when there are three or more equally parsimonious cladograms.



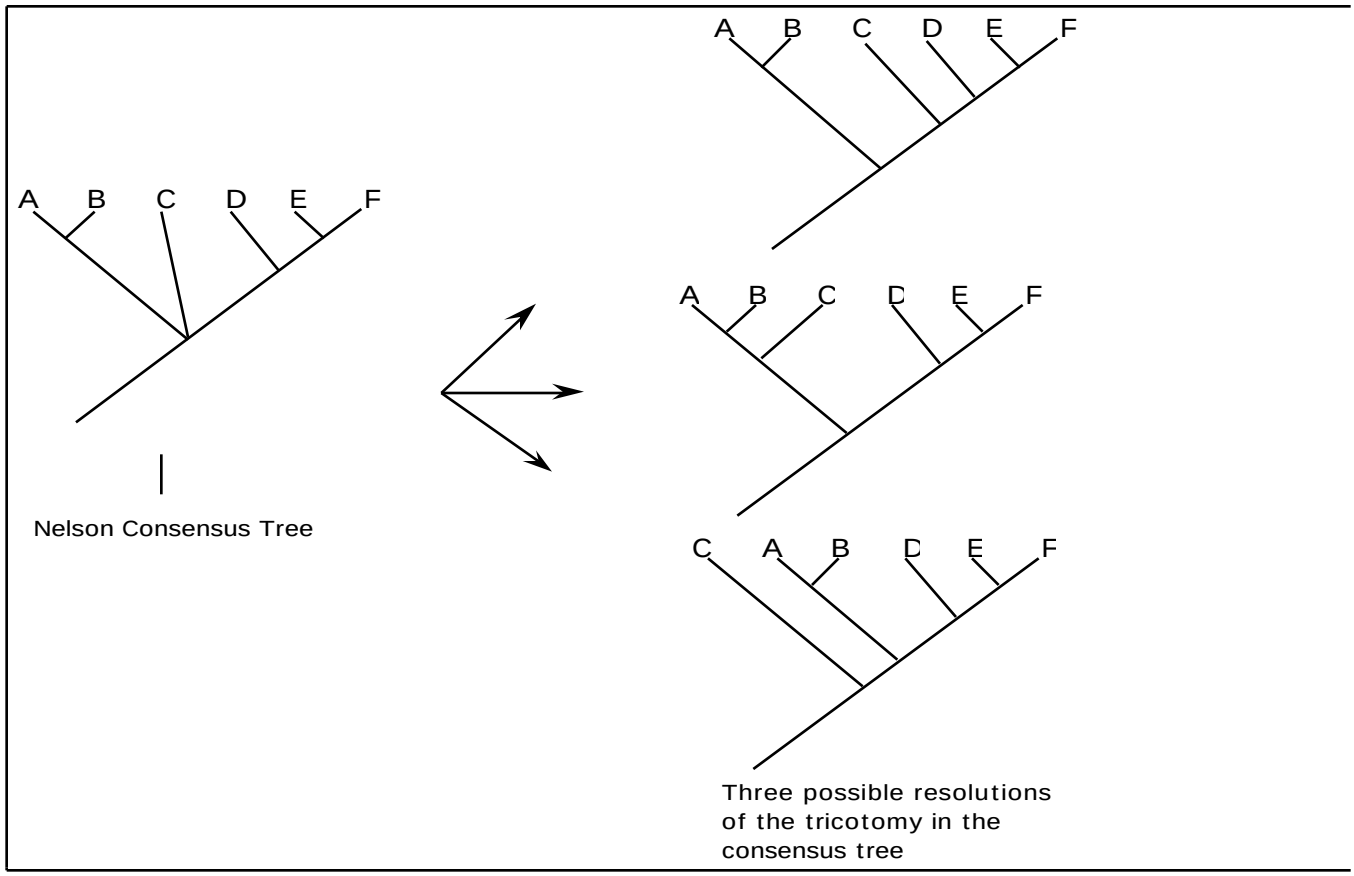
Majority Rule Consensus

Problems With Consensus Trees:

1. A consensus tree will always require more steps (have a longer length and lower consistency index) and will, therefore, be less parsimonious than any of the trees from whi it was formed (Miyamoto 1985):



2. Consensus trees result in unresolved branching points. Unresolved nodes imply many relationships of the taxa, but not all of these relationships may be possible given the origina data set and cladograms. For example, the unresolved tricotomy in the example data, impl the following three resolved relationships are possible:

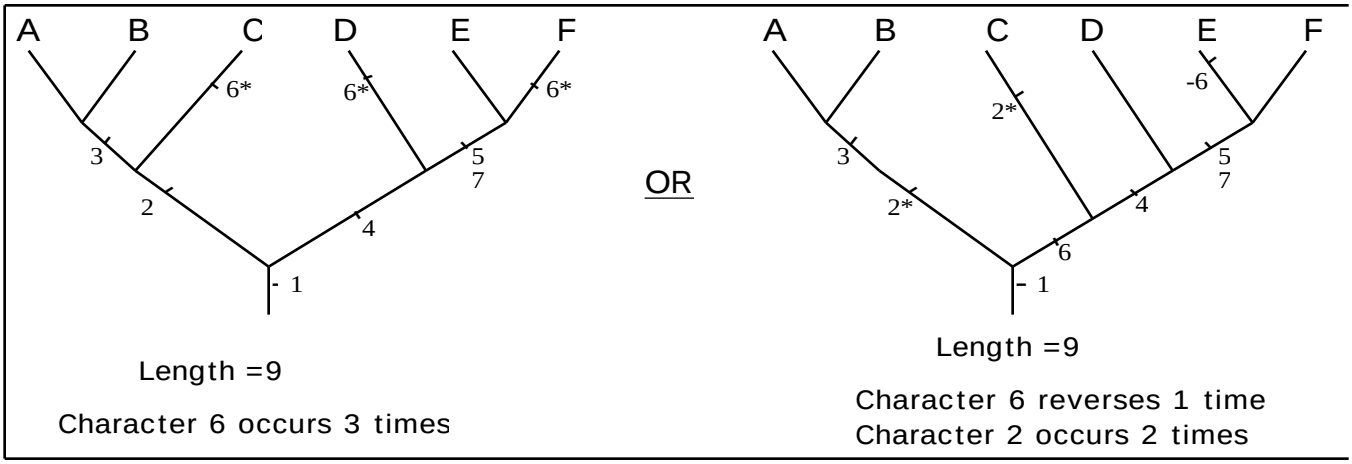


Resolved tree 3 is not possible given the original cladograms used to form the consensus tr

3. Nelson, Adams, and majority rules consensus trees may be in conflict with the original cladograms.

Successive Weighting

When many equally parsimonious trees are produced, criteria other than strict parsimony may be used to choose among them. One such criterion is to choose the cladogr which requires the fewest number of characters to have homoplasies. For example, consid the following two cladograms:



Although both trees have the same length, the first tree requires only one character to be homoplasious. Therefore, the first tree may be chosen over the second because it has both shortest length and the fewest homoplasies .

The tree with the fewest characters having homoplasies can be determined using a procedure called **successive weighting** (Farris 1969, Carpenter 1988). In this procedure, the b fits of the characters are used to calculate weights. If the best fit of a character is to have a consistency index of 100, then the character will get a high weight. If the best fit of a charact is a consistency index of 33, the character will get a low weight. The weighted characters are then used to create a new cladogram. If many equal cladograms are found with the weighte data, recalculate new weights and a new tree. Continue the process until either a minimur number of cladograms are obtained or the same answer is obtained two times ia row.

Weights for successive weighting can be calculated with Hennig86 with the comma

```
*>xsteps w; (or xs w;)
```

To obtain new trees with successive weighting use the following commands:

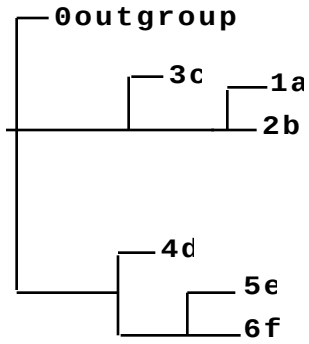
```
*>p many.dat;
*>ie;tp;
*>xs w;
*>cc;
*>ie;tp;
```

etc. until a minimum number of trees is found.

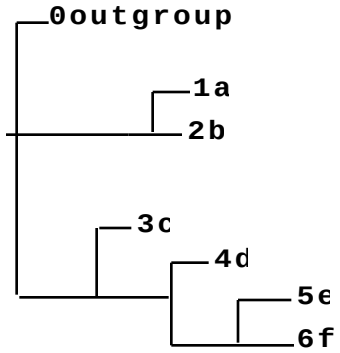
For example, applying successive weighting to the data set on page 74:

```
procedure many.dat *
xread
```

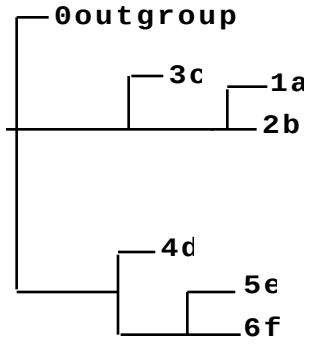
```
demonstrating more about multiple equal trees
procedure --

ie length 9 ci 77 ri 77 trees 2
tplot file 0 from ie 2 trees

tree 0
```




```
tree 1
```




```
xsteps file 0 from ie 2 trees
ccode
0    1    2    3    4    5    6
10+[ 10+[ 10+[ 10+[ 10+[  2+[ 10+[

ie length 66 ci 93 ri 94 trees 1

tplot file 0 from ie 1 tree
```



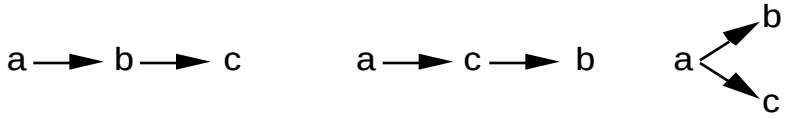
Multistate Characters

A character transformation series (Hennig, 1966) is a hypothesis that specifies which states of a character evolve directly into which other states (= character phylogeny of Henni 1966; character state tree of Farris, 1970). As has been explained by Mickevich (1982), to determine character transformation series, we must know both the character state polarity a character state order. Character polarity specifies which state is plesiomorphic. Order of character state change is the relative relationship of states of a character.

Character state polarity is solved using the outgroup comparison method which has been shown to be an extension of the parsimony criterion (Farris, 1982). There is only one possible character state order when there are two states:

a → b

But order is problematic when there are multiple states. In a character which has th states (**a**, **b**, and **c**), if state **a** is also found in the outgroup, it is considered to be plesiomorph But this does not tell us the order of the apomorphic states **b** and **c**. In fact, there are three possible transformations of the character:



The more states in a character, the more transformations are possible and the more puzzlin the order.

Systematists have employed a variety of methods for determining character state order. The major methods are:

1. Those that impose a rule of how the character evolved
2. Nonadditive character analysis
3. Unordered analysis or Fitch parsimony
4. Transformation Series Analysis
5. FIG/FOG analysis
6. Homology analysis

Rules of Evolution

Since Darwin, many biologists have proposed rules of evolution. Some of these ha been applied to the problem of determining character state transformation series:

a. Biogenetic Law - (Ontogeny recapitulates phylogeny) the ontogenetic sequence is the same as the character state phylogeny.

b. Stratigraphic occurrence - states can be traces from primitive to derived through their relative occurrence in the fossil record.

c. Cope's rule of size - primitive organisms are small, advanced organisms are large. Th character phylogeny can be deduced from the size of the organisms.

d. Cope's rule of the unspecialized - general features are primitive; advanced features ar specialized.
e. Morphocline analysis (gradualism) - assumes evolution proceeds in small adaptive steps (gradualism) and multistate characters should be ordered to minimize the amount morphological change between character states (Maslin, 1952).

Nonadditive character analysis

One solution, proposed by Sokal and Sneath, to the problem of ordering multistate characters is to change each state of a multistate character into a character. Each of these characters would have two states - present or absent.

For example, consider the character for type of insect wings:

Wings: (**a**) absent (e.g., fleas); (**b**) present with hard outer covering (e.g., beetles); (**c**) present and covered with scales (e.g., butterflies); (**d**) present and membranous (e.g, flies).

Can be changed to five binary characters:

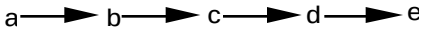
Wings: (a) absent; (b) present.
Wings with hard outer covering: (a) absent; (b) present.
Wings covered with scales: (a) absent; (b) present.
Wings membranous: (a) absent; (b) present.

Although this method appears to avoid the question of order, it incorrectly treats th states or forms of a character as distinct unrelated structures. Each state may be a modificat of one of the other states or it may be modified into one of the other states (or both!).

When the states are disconnected and converted into characters, the transformational information is lost. The loss of this information may result in less informative classifications:

	1	2	3	4	5
A	a	a	a	a	a
B	b	b	b	b	b
C	c	c	c	c	c
D	d	d	d	d	d
E	e	e	e	e	e
Out	a	a	a	a	a

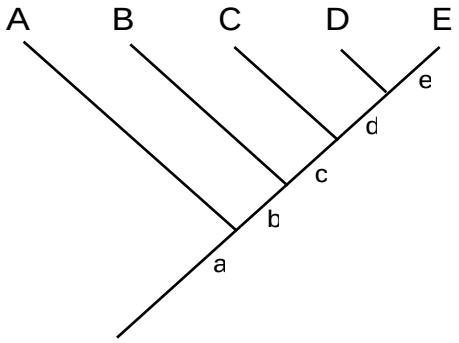
If



Then:

	1	2	3	4	5
A	0	0	0	0	0
B	1	1	1	1	1
C	2	2	2	2	2
D	3	3	3	3	3
E	4	4	4	4	4
Out	0	0	0	0	0

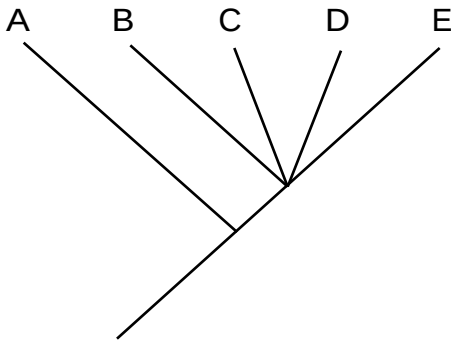
The cladogram would be:



But, if the states are nonadditively transformed into characters:

	1a	1b	1c	1d	1e	2a	2b	2c	2d	2e	3a	3b	3c	3d	3e	4a	4b	4c	4d	4e	5a	5b	5c	5d	5e
A	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-
B	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-
C	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-
D	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-
E	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+
Out	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-

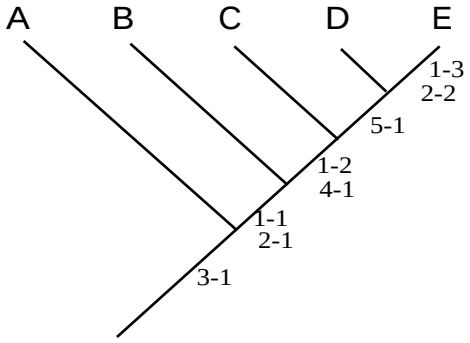
The cladogram of this transformed data set is:



The loss of the information may even result in a different tree. Consider the following data set where characters 1 and 2 have multiple states:

	1	2	3	4	5
A	0	0	1	0	0
B	1	1	1	0	0
C	2	1	1	1	0
D	2	1	1	1	1
E	3	2	1	1	1

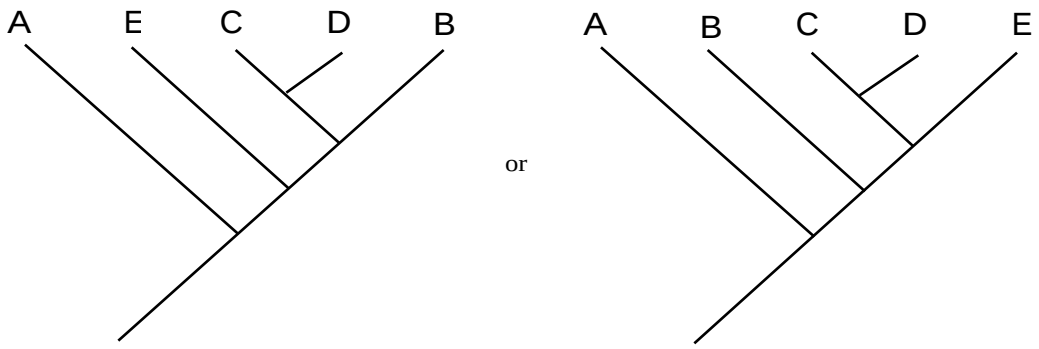
If the states transform in the order indicated, the cladogram will be:



But, if the states are nonadditively transformed into characters:

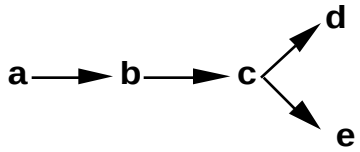
	1-1	1-2	1-3	2-1	2-2	3	4	5
A	0	0	0	0	0	1	0	0
B	1	0	0	1	0	1	0	0
C	0	1	0	1	0	1	1	0
D	0	1	0	1	0	1	1	1
E	0	0	1	0	1	1	1	1

The cladogram will be:

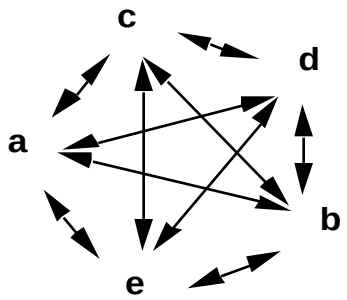


Unordered analysis or Fitch parsimony

This methods assumes that any state can transform into any other state with just or evolutionary step. In an ordered transformation, such as:



to transform from state **a** to state **c**, the state must pass through state **b**. When counting evolutionary steps, **a** to **c** will be two steps. In an unordered analysis, any state is allowed to transform into any other state with just one step. The transformation series used to constr the cladogram is:



The program Hennig86 can be used to analyze multistate characters using this unordered model by giving the **ccode;** command after the procedure file has been loaded.

If all multistate characters are to be unordered, it is simpliest to unorder all characters with the command:

***>cc -.;**

This will have no affect on the two-state characters.

If only some of the characters are to be unordered, give the **cc -** command followed by the characters:

```
*>cc - 2 4 5 7 9.22;
```

NOTE: The default setting of hennig86 is ordered character states.

Ordered Character Analysis

Using the MST Program

The MST program was written by J. S. Farris to translate data files written for the C-read option of the PHYSYS program into data files for the Hennig86 program. Even thou you may not need to translate PHYSYS files, the MST program can also be used to quickly additive binary code multistate character state trees or to write files in which characters and their states are denoted with words and symbols rather than numbers.

The MST disk contains 3 files:

MST.EXE	- the mst program
MIT2	- a sample mst data file
MIT2.OUT	- the file written by the MST program for the data in the file

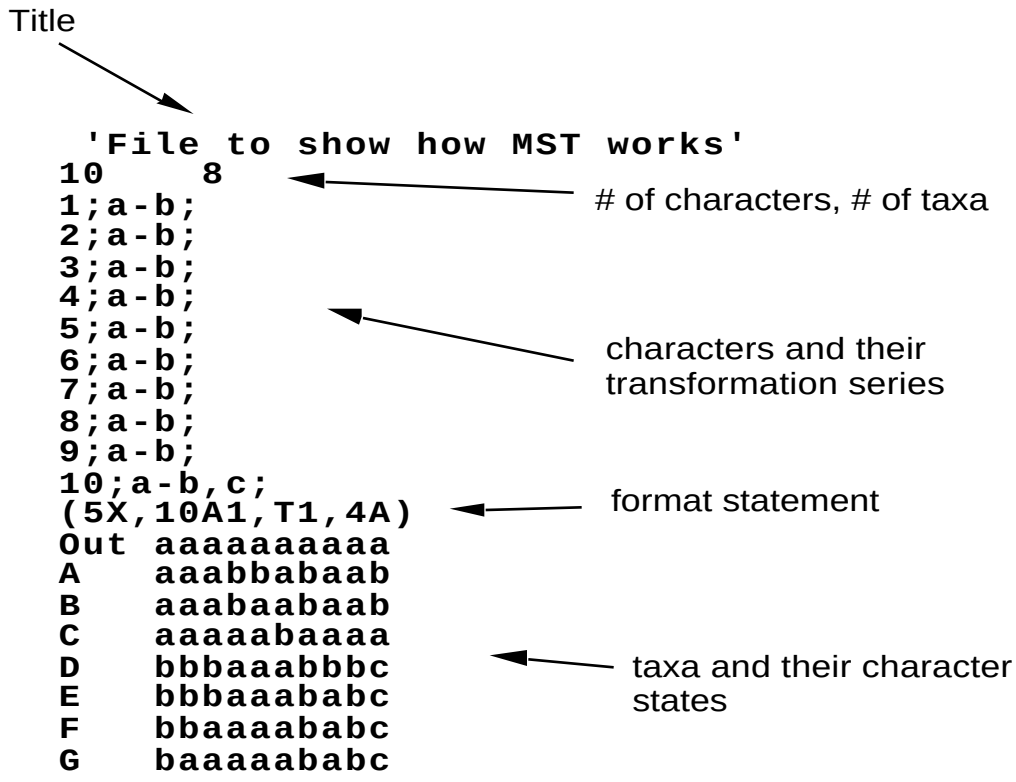
To use MST, all data are written into files (called here input files) in a specific format. The MST program rewrites these files into an output file compatible with the Hennig86 progra

MST Input Files

The files read into the MST program are identical to a C-read PHYSYS file. These fi consist of:

1. Title - (quotes are not necessary):
   Sample File to show how MST works
   or
   '**Sample File to show how MST works**'
2. The number of characters and taxa on one line:
   10 8
3. A line for each character showing its transformation series. See section I below.
4. A format statement to inform MST what form the data is in. See section II below
5. The taxa names and the character states. See section III below.

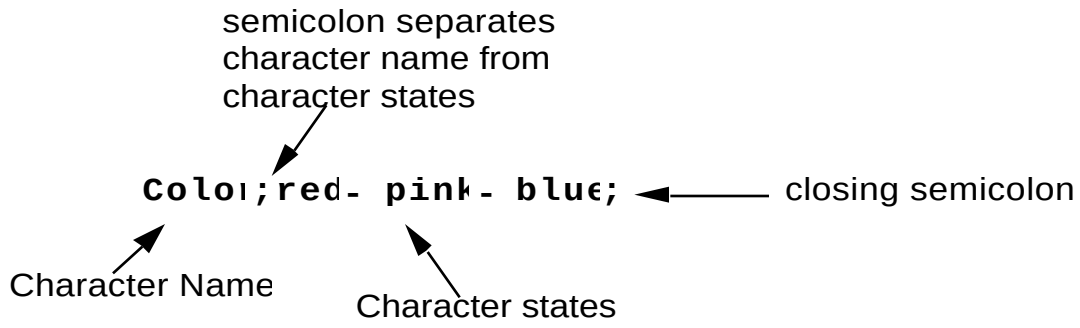
A final input file would look something like this:



I. Notation for Characters' Transformation Series

A. How to describe the character and its states

All of the characters in the data set, whether multistate or binary, must be represented by a line indicating each one's transformation series. These lines consist o the name of the character separated from the transformation by a semicolon, the transformation series itself, and a closing semicolon. For example, if we are using a character about color with the states red, pink and blue, the character could be written i the data file:



The character name or the character state name can be a number or a word up to 8 characters long (with no blank spaces).

B. Multistate Character Transformation Series

Multistate characters with nonbranching transformation series can be entered f analysis by computer programs as consecutive numbers (e.g., 0-9). For some programs, such as Hennig86, it is necessary to recode branching character state trees into linear variables. The most commonly used method of coding is additive binary coding (Farri et al., 1970). (Modifications to this method have been described by Pimentel and Riggi (1987), O'Grady and Deets (1987), O'Grady et al. (1989), and Mickevich and Weller (1990 The MST program translates a given transformation series into additive binary code a writes it into a Hennig86 xread file.

1. Linear Transformation Series

If the states of the character transform linearly:

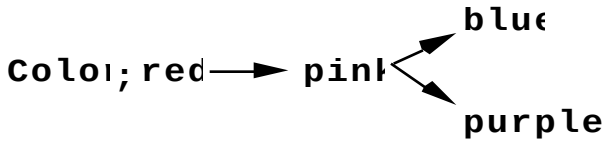
Color;red->blue->pink;

Transformation series are indicated with a dash:

Color;red-blue-pink;

2. Simple branching transformations

If more that one state is derived from another state, as in the following exampl where both colors blue and purple are derived from pink:

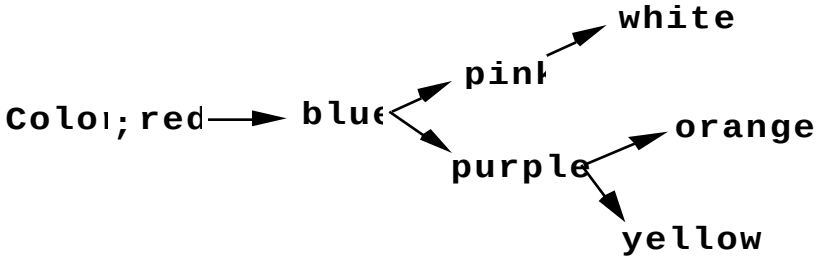


the multiple descendant states are separated with a comma:

Color;red-pink-blue,purple;

3. Complex branching transformations

If a transformation series has several branches



the subbranches are enclosed in parentheses or brackets

color;red-blue-(pink-white),(purple-orange,yellow);

or **color;red-blue-[pink-white],[purple-orange,yellow];**

II. <u>The Format Statement</u>

The format statement simply indicates the form the data is in. You may be familiar with this type of statement from other programs.

A. The format statement consists of the following fields separated by commas:

1. The number of spaces found before the characters are listed.
2. The number of characters, whether they are numbers or letters and how many spaces each occupies.
3. The number of spaces found before the character name is listed.

4. An alphanumeric field for reading the name of the taxon.

B. If the data has been written in the following way: **Name abbabbab**

The format statement is:

(5X,8A1,T1,4A)

This format statement includes:

1. The fact that the beginning of the list of character states is 5 spaces from the left margin (5X).
2. There are 8 characters (8), they are represented by letters (A), and they occupy one space with no blank spaces in between them (1). 3. The name of the taxon is found the first space of the left margin (T1).
4. The name of the taxon is 4 letters long(4A).

C. If the data has been written in the following way: **Name a b a b b a b**

The format statement must include:

1. The fact that the beginning of the list of character states is 5 spaces from the left margin.
2. There are 8 characters, they are represented by letters, and they occupy one space with a blank space in between them (so that each character + its blank space takes 2 spaces).
3. The name of the taxon is found at the first space of the left margin.
4. The name is 4 letters long.

The format statement is:

(5X,8A2,T1,4A)

D. If the data has been written in the following way: **abbabba Name**

The format statement is:

(8A1,T10,4A)

E. If the data has been written so that each state is a word:

Name red yes yes no big tip yes no

The format statement is:

(5X,8A4,T1,4A)

Note that each short word has had blank spaces added after it so that it takes up the sar amount of space as the longer words.

F. If the data occupies more than one line, indicate what is on each line and separate t lines with a slash (/):

a a a a b b b b b b b b b b b b
a a a a a a a a a a Name

The format statement is:

(17A2/10A2,T21,4A)

III. The taxa names and characters are entered so as to be compatible with the format statement. Unlike a Hennig86 file, each line for each taxon must be in the same format, wi the same number of letters in the taxa names, and the same number of blank spaces betwee characters.

IV. MST

A. Running MST

1. Access the drive or directory with the MST. EXE file.
2. At the prompt type:

C> **mst <*input file*> <*output file*>** (press return)

The drive will whir, and the file will be translated into a Hennig86 file. Warning: y should give the output file a different name from the input file otherwise the program wil overwrite the input file and replace the data in it with the output file.

B. Example Data file

1. Create this data file called MST. TRY on a disk:

'File to show how MST works'
10 8
1;a-b;
2;a-b;
3;a-b;
4;a-b;
5;a-b;
6;a-b;
7;a-b;
8;a-b;
9;a-b;
10;a-b,c;
(5X,10A1,T1,4A)
Out aaaaaaaaaa
A aaabbabaab
B aaabaabaab
C aaaaabaaaa
D bbbaaabbbc
E bbbaaababc
F bbaaaababc
G baaaaababc

In this file the last character (10) is multistate with states b and c derived separately from sta a.

2. Go to the drive or directory that has the MST. EXE program and type:

C> mst mst.try mst.out (press return)

The MST. OUT file will contain the translated file that can be read by Hennig86.

3. Using a text editor or word processor, look at the MST. OUT file: **xrea**

```
'file to show how mst works'
11 8
out
0 0 0 0 0 0 0 0 0 00
a
0 0 0 1 1 0 1 0 0 10
b
0 0 0 1 0 0 1 0 0 10
c
0 0 0 0 0 1 0 0 0 00
d
1 1 1 0 0 0 1 1 1 01
e
1 1 1 0 0 0 1 0 1 01
f
1 1 0 0 0 0 1 0 1 01
g
1 0 0 0 0 0 1 0 1 01
;
procedure / ;
```

V. Error messages

If the MST program is unable to translate a data file, it will write an error message in the output file. Some commonly encountered types of error messages are given below:

A. Characters and character states must be 8 letters or less. If a longer name is given, output file will indicate this with **NAMESIZE**.

For example, the name of the first character is too long in the data file below:

```
'File of cell characters'
5 4
microtubules;a-b;
nucleus;a-b;
plastid;a-b;
vacuole;a-b;
(5X,5A1,T1,4A)
Out aaaaaaaaaa
A  aaabbabaab
B  aaabaabaab
C  aaaaabaaaa
```

this resulted in the following error message in the output file:

xread
'file of cell characters'
9 namesize

B. If the number of characters or taxa is incorrect, the program may encounter data or format statement when it expects a character name. If this happens the **NAMESIZE** message may be given because the line of data or the format statement is too long for t taxon name.

C. If a taxon has a state that is not listed in the transformation series of the character, t MST program will stop when it encounters the unknown state.

For example, the following data file:

'File to show how MST works'
10 8
1;a-b;
2;a-b;
3;a-b;
4;a-b;
5;a-b;
6;a-b;
7;a-b;
8;a-b;
9;a-b;
10;a-b-c;
(5X,10A1,T1,4A)
Out aaaaaaaaaa
A aaabbabaab
B aaabaabaab
C aacaabaaaa
D bbbaaabbbc
E bbbaaababc
F bbaaaababc
G baaaaababc

Gives the following output file:

xread
'file to show how mst works'
10 8
out 0 0 0 0 0 0 0 0 0 00
a 0 0 0 1 1 0 1 0 0 10
b 0 0 0 1 0 0 1 0 0 10
curtax 3 after b char 3 field c unknown

A taxon (in this case taxon c) has a state (in this case state c for character 3) that is not in

the transformation series (in this case the transformation series was just 3;a-b). Fix the transformation series to reflect the position of state c and try again.

D. Many times the MST program aborts because the format statement is not compatibl with the way the data is written. This can be very frustrating but by reading the messag in the output file, you can often figure out where you went wrong. For example, if the format statement does not indicate a space between characters -

e.g., the format statement is:

(5X,8A1,T1,4A)

but the data has been written with spaces between the character states:

Name a b b a b b a b

The output file will read:

xread
'**file title**'
11 8
curtax 0 after none char 2 field unknown

The program encountered a blank where it expected character 2 and could not interpret it.

In another example, the format statement reads:

(1X,8A2,T1,4A)

But the data is written in the following form:

Equis a b b a b b a b

If you proofread carefully, you will see that the format statement mistakenly indicates that the first character is found on the second space of the line when in reality it begins one space after the taxon name, *Equis*. When the MST program tries to translate this l it expects the letter q in the taxon name to be a character state and can't interpret it. In output file this is indicated with the following message:

xread
'**horse data**'
8 8
curtax 0 after none char 1 field q unknown

E. The letters used in the format statement must be in uppercase. If a lowercase letter any unrecognized letter or symbol is used, the output file will indicate this by showing the symbol or letter it couldn't interpret and the message **FORMAT SYMBOL**.

Ordered Analysis 1: Transformation Series Analysis

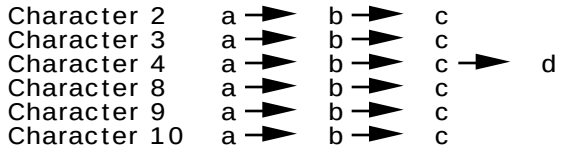
Transformation series analysis (TSA) seeks to maximize congruence among charact through an iterative procedure (e.g., Mickevich, 1982). TSA uses the hierarchy of the tree t determine the order of character states. The order derived in this way is referred to as a cladistic or cladogram character (Mickevich, 1982).

TSA is illustrated for the data:

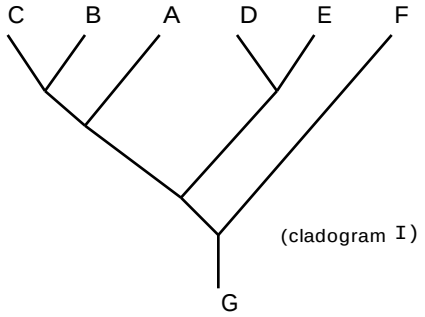
	Character										
	1	2	3	4	5	6	7	8	9	10	11
Taxon											
A	b	b	b	b	a	b	b	b	b	b	b
B	b	b	b	d	a	b	b	b	c	b	b
C	b	b	c	d	a	b	b	c	c	b	b
D	b	c	c	c	a	b	a	c	c	c	a
E	b	c	a	c	a	b	a	c	a	c	a
F	b	a	a	a	b	a	a	a	a	a	a
G out	a	a	a	a	a	a	a	a	a	a	a

This example is complicated to give the details of how the method works. The following steps are modified from Lipscomb (1990):

1. **With an initial set of transformation series for each character, construct a cladogram**. The initial transformation series for the multistate characters are:



The order is contrived for this example, but any hypothesis may be used to initially order the characters. The corresponding cladogram is:



2. **From the cladogram, a set of cladogram characters is constructed**. Cladogram characters are character state trees that are derived by mapping the states on the hierarchy of the cladogram:

   a. **Construct a cladistic distance matrix giving the number of nodes or cladogenetic events between each taxon in the initial cladogram**. The

taxa closest to one another have the fewest intervening nodes and are said to be nearest neighbors (for details see Mickevich, 1981). The cladistic distance matrix for the initial cladogram is:

	A	B	C	D	E	F
A	-					
B	2	-				
C	2	1	-			
D	3	4	4	-		
E	3	4	4	1	-	
F	3	4	4	3	3	-
G	3	4	4	3	3	1

b. **For each multistate character, a nearest neighbor diagram is constructed**. There are two ways to do this. The first, which forms a nearest neighbor matrix, is described by Mickevich (1982): (i) a list of nearest taxa neighbors is made from the cladistic distance matrix. In the example, the nearest neighbor of taxon C is B and vice versa; on the other hand, taxon A is nearest neighbor to taxa B and C. The nearest neighbors of just the taxa are:

Taxon	Nearest Neighbor
A	B + C
B	C
C	B
D	E
E	D
D + E	A & F
F	G
G	F

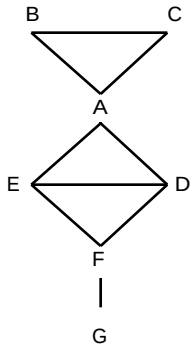
(ii) From the list of nearest taxa neighbors, the nearest neighbors of *characters states* for these taxa are determined by the rule that a state is nearest neighbor to another state if the taxa they are in are nearest neighbors. For character 2, state **a** is nearest neighbor to state **c**. This is because state **a** is found in taxon F and it is a nearest neighbor of taxa E and D which have state **c**. In the nearest neighbor matrix (below), a 1 is placed where **a** and **c** intersect. The nearest neighbor of state **b** is also state **c** since taxa E and D are adjacent to taxon A which has state **b**. Nearest neighbor matrices for all the multistate characters are:

	2				3				4			
	a	b	c		a	b	c		a	b	c	d
a			1	a		1	1	a			1	
b			1	b	1		2	b			1	1
c	1	1		c	1	2		c	1	1		
								d		1		

	8				9				10		
	a	b	c		a	b	c		a	b	c
a			1	a		1	1	a			1
b			2	b	1		2	b			1
c	1	2		c	1	2		c	1	1	

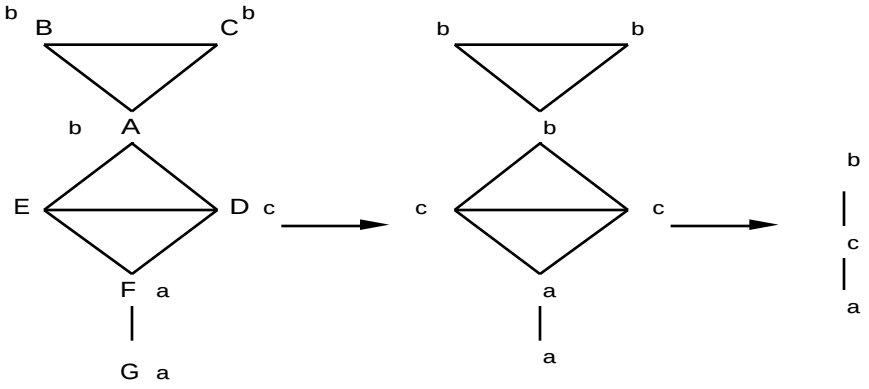
States which are nearest neighbors more than once on a cladogram are linked by numbers higher than one on the matrix. For example, states **b** and **c** in character 3 are found in nearest taxa neighbors C and B and again in taxa D and A. Since these states are nearest neighbors twice, a 2 is placed where **b** and **c** intersect on the matrix.

A second method involves making a network of the states showing them linked to their nearest neighbor rather than in a matrix (Lipscomb, 1990): (i) nearest taxa neighbors are linked to each other in a diagram:



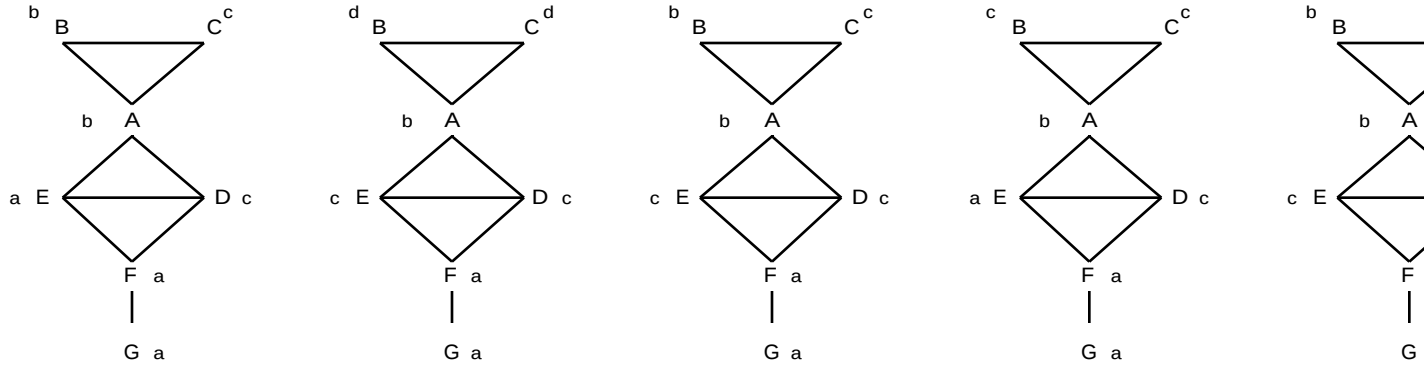
Nearest Taxa Neighbors

The character states for each taxa are substituted for the taxa names and the branches with the same character state are collapsed to give a network:



Character 2 Character 2

The network then shows the nearest character neighbors. Nearest neighbor networks for all the multistate characters are given below. States that are nearest neighbors more than once are linked by more than one line.



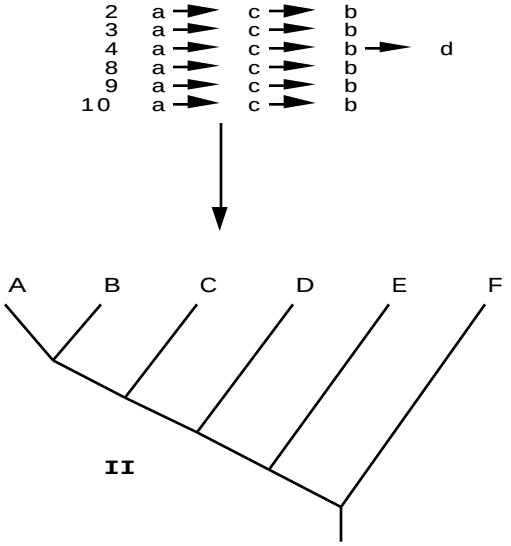
Character 3 Character 4 Character 8 Character 9 Charact

c. **Construct the cladogram characters**. The transformation series that exactly reflects the hierarchy of the cladogram is called the cladogram character (Mickevich, 1982). The nearest neighbor matrix or diagram indicates the order of the states. For character 2, this order is **a - c - b** because **a** is nearest neighbor to **c** and **b** is nearest neighbor to **c**. This order indicates that to go from state **a** to state **b** or from state **b** to state **a,** you must go through state **c**. The outgroup has state **a** which gives the plesiomorphic pole. A combination of the order and the polarity gives us the transformation series **a -> c-> b**. This transformation series (see above) is the cladogram character for character 2. The cladogram characters for all the multistate characters are given in the figures above. The cladogram characters for characters 3 and 9 require extra explanation. Looking at either the nearest neighbor matrix or the network, it is clear that the order is equivocal. In character 3, for example, state **b** is nearest neighbor to **a** once and **c** twice. The order that

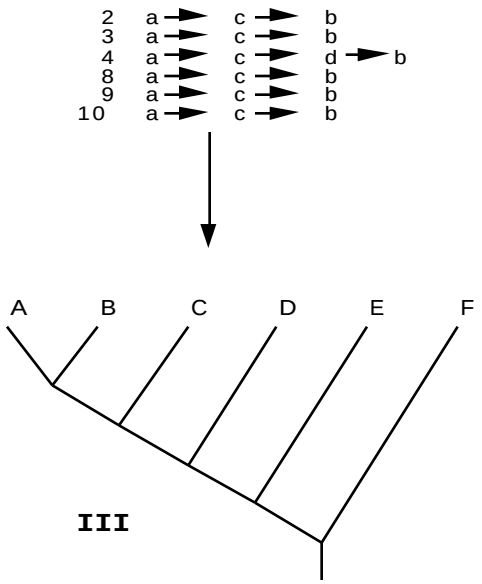
is used is the one for which there is the most evidence - that state **c** is nearest neighbor of state **b**.

3. **Compare the cladogram characters to the initial transformation series.** In the example, you can see that for characters 2, 3, 4, 8, 9 and 10 the cladogram character is different from the initial transformation.

4. **Replace the initial transformation series with that of the cladogram characters and construct a new cladogram**. In this example, a new cladogram (II) is found:



5. **Construct a new set of cladogram characters as in step 2 above for the new cladogram**. **Compare the new set of cladogram characters with those used to construct the second cladogram. If they are the same then stop. If they are different, replace the transformation series with the new cladogram characters, construct a new cladogram and set of cladogram characters. Continue this iterative procedure until the transformation series and the cladogram characters are the same**:



In this example, the transformation of the cladogram character for 4 is different from that used to construct this cladogram. This transformation of character 4 together with the previously determined transformations for characters 2, 3, 8, 9 and 10 are used to construct a new cladogram (III). The cladogram characters for cladogram II the same as those used to form the cladogram III, so we stop. The transformation series of the multistate characters determined from the cladogram characters of this tree are, then, the accepted transformations. These transformation series can, of course, be tested with additional data such as more taxa and / or characters.

Homology and Transformation Series

(This method is described in detail in Lipscomb 1992.)

When states are considered to be alternative forms of a character, ordering the distinct states does seem enigmatic. However, this concept of character states is incorrect; states are modifications of either the original attribute or other states so that a character is an internested set of synapomorphies. For example, given a character consisting of three states with the transformation series **A -> B -> C**, states **A**, **B**, and **C** are not alternative and distinct character states, but rather state **B** is an addition or modification of state **A**, and state **C** is a modification of **B**. In other words, **A** is a more general, more inclusive, and more plesiomorphic character state relative to **B** and **C** (and likewise **B** relative to **C**) with the result that an organism that has state **B** or **C** also has state **A** .

Therefore, the transformation series hypothesizes a hierarchy of homologies such t
a state is either a modification of an adjacent state, or is modified in an adjacent state, or bo
From this it follows that ascertaining state order is a problem of determining the degree of
homology among the states. The methods phylogeneticists use to postulate and test
homology should be used to postulate and test the order of states in a transformation series
a multistate character. Most phylogeneticists agree that this method is a two-step process th
applies first a similarity criterion and then a criterion of congruence with other characters.

Thus, determining a transformation series is a two-step process in which (1) an orde
of the states is postulated so that states that are most similar are adjacent to each other,
followed by (2) testing the hypothesis of order using congruence between the groups define
by the hypothesized transformation series and those suggested by other characters.

STEP 1 - APPLYING THE SIMILARITY CRITERION

Homologous structures are usually similar, and this similarity may be of any form:
grossly morphological, ontogenetic, molecular and so on. The observation that any two sta
are similar relative to other states provides a basis for hypothesizing that the two states sha
a level of homology and should be adjacent to each other in a transformation series.
Therefore, the first step in determining a character state tree is meticulous examination of
details (morphological, biochemical, ontogenetic, physiological, etc.) available for the states
The states are then arranged in an order so that those most similar are adjacent to each oth
with intermediate states linking extreme forms of the character. It is not necessary to make
evolutionary assumptions about character state changes to assess the relative similarity of t
states as some authors have claimed (Hauser and Presch, 1991). That two states are similar
simply asserts that a description fitting one state also applies to the other. The more simila
two states are, the more specific and detailed the description can be and still apply to both.

Some systematists consider the similarity criterion to be one test of homology (e.g.,
Patterson, 1982) such that, given all possible transformation series, those that postulate
homology among nonsimilar forms are rejected. Other systematists (e.g., Cracraft, 1981: 26)
prefer to view the similarity criterion, not as a test of homology *per se*, but as a factor that
compels us to postulate homology. The alternative viewpoints do not appear to affect the
final hypotheses of character state transformation that are selected.

Similarity of states alone is not sufficient for hypothesizing a complete transformati
series, because the observation of similarity does not differentiate between apomorphy and
plesiomorphy. In other words, similarity implies state adjacency but not polarity.
Consequently, once state order has been established with observed similarity, it must be
polarized using the outgroup comparison method in order to achieve a complete
transformation series.

For the data set (below), the states of the multistate character (character 1), can be ordered sc
that those most similar in morphological detail are adjacent to each other. In this case two
different transformation series are hypothesized:

Once transformation series have been formulated using the similarity criterion, cladograms can be constructed. But this is not the final step. The similarity criterion, by its does not distinguish between homology and homoplasy. It only provides initial hypothese of homology which can be further tested by congruence with other characters

STEP 2 - APPLYING THE CONGRUENCE CRITERION

The congruence of a transformation series with the other characters can be used bot to test its support and as a means for choosing among several alternative transformation series that have been hypothesized for the same character. Returning to the two transformation series hypothesized for the multistate character in the sample data, we can their congruence with other proposed homologies by constructing cladograms with both (both give the same cladogram), comparing the groups in the cladogram with those propos by the transformation series, and noting the conflicts between the transformation series an the groups on the tree. In this example, the first transformation series and the cladogram a congruent, and this provides support for the first hypothesis of state transformation. On th other hand, the second transformation series and the cladogram conflict as to the existence several groups, and this suggests that the second hypothesis of character transformation should be rejected.

A Summary Of The Commands Used In Hennig86

Alphabetical Order

assist Lists commands that can be used

batch Turn on batch switch

batch - Turn off batch switch

bb; Applies branch-swapping to trees constructed by another command (e.g., **mh;** or **h;**) and stores these trees in a new tree file. Unlike **h;** and **mh;**, which retain only one tree for each initial one found, with the **bb;** command up to 100 of the shortest trees are retained.

bb*; Same as the **bb;** command except all trees will be retained up to the limits of the space allocated in the computer for tree storage. In other words, if the * option is not used, no more than 100 trees will be retained.

bytes Display bytes of free ram

ccode Control character coding

ccode /	set weight
ccode]<character number>	deactivate character
ccode [<character number>	activate character
ccode +<character number>	additively code a multistate character
ccode +.	additively code all multistate characters
ccode -<character number>	nonadditively code a multistate character
ccode -.	nonadditively code all multistate characters

ccode ; Display current character codings

cget x Set coding from code file x

ckeep x Save current coding in code file x

display Short listings of results to monitor

display - No listing of results to monitor

display* All listing of results to monitor

erase s Delete tree files in scope s

files Display directory of treefiles

get x Make tree file x current

hennig; Constructs a single tree by a single pass throughthe data. This is very

rapid but may not find the shortest tree if there is much homoplasy in the data.

hennig*; Constructs a single tree by a single pass through the data and then applies branch-swapping to the initial tree, retaining just one tree.

ie; Generates trees by implicit enumeration and retains up to 100 of the trees. The results are certain to be the most parsimonious trees, but it may be time consuming if the data set is large or contains much homoplasy.

ie*; Same as the **ie;** command except all trees will be retained up to the limits of the space allocated in the computer for tree storage. In other words, if the ***** option is not used, no more than 100 trees will be retained.

ie-; Identifies one tree, certain to be of minimum length. For large data sets, this may be much faster than **ie;** or **ie*;**

keep x Save current tree file as tree file x

log <*filename*> Create a dos file as new log file

log -	deactivate log file
log *	activate log file
log /	close log file

mhennig; Constructs several trees, each by a single pass but adding the taxa in a different sequence each time.

mhennig*; Constructs several trees, each by a single pass but adding the taxa in a different sequence each time and then applies branch-swapping to each of the trees, retaining just one tree for each initial one.

nelsen Calculate strict consensus tree

outgroup Control outgroup

outgroup = <*taxon name*>	set outgroup
outgroup ;	list outgroup(s) alphabetically

procedure <*filename*> Open dos file as procedure file

procedure - Deactivate procedure file

procedure* Activate procedure file

procedure / Close procedure file

quote	Insert a message from the keyboard
reroot	Current treefile according to current outgroup
steps	Display max/min steps per character
tchoose s	Select trees in scope s from current tree file
tlist	Display trees in parenthetical notation
tplot	Produce tree diagrams
tread	Read trees
tsave n	Save current tree file on dos file n
txascii	Use extended ascii characters in tree plots
txascii -	Don't use extended ascii characters in tree plots
view n	Inspect dos file n
view *	Close and inspect current log file
watch	Turn on stopwatch
watch -	Turn off stopwatch
xread	Read character data
xsteps	Diagnose trees in current tree file
xsteps h	list possible states for hypothetical ancestors
xsteps c	list character fits
xsteps m	list best/worst fits
xsteps l	list tree lengths
xsteps u	produce file of distinct trees
xsteps w	set character weights according to fits
xx	Display and modify diagnosed tree
yama	Return to dos

A Summary Of The Commands Used In Hennig86

By Function

CONSTRUCTING TREES

hennig; Constructs a single tree by a single pass throughthe data. This is very rapid but may not find the shortest tree if there is much homoplasy in the data.

hennig*; Constructs a single tree by a single pass through the data and then applies branch-swapping to the initial tree, retaining just one tree.

mhennig; Constructs several trees, each by a single pass but adding the taxa in a different sequence each time.

mhennig*; Constructs several trees, each by a single pass but adding the taxa in a different sequence each time and then applies branch-swapping to each of the trees, retaining just one tree for each initial one.

bb; Applies branch-swapping to trees constructed by another command (e.g., **mh;** or **h;**) and stores these trees in a new tree file. Unlike **h;** and **mh;**, which retain only one tree for each initial one found, with the **bb;** command up to 100 of the shortest trees are retained.

bb*; Same as the **bb;** command except all trees will be retained up to the limits of the space allocated in the computer for tree storage. In other words, if the ***** option is not used, no more than 100 trees will be retained.

ie; Generates trees by implicit enumeration and retains up to 100 of the trees. The results are certain to be the most parsimonious trees, but it may be time consuming if the data set is large or contains much homoplasy.

ie*; Same as the **ie;** command except all trees will be retained up to the limits of the space allocated in the computer for tree storage. In other words, if the ***** option is not used, no more than 100 trees will be retained.

ie-; Identifies one tree, certain to be of minimum length. For large data sets, this may be much faster than **ie;** or **ie*;**

nelsen Calculate strict consensus tree

CONTROLING CHARACTERS

ccode Control character coding

ccode / Set weight

ccode]<character number> Deactivate character

ccode [<character number> Activate character

ccode +<character number> Additively code a multistate character

ccode +. Additively code all multistate characters

ccode -<character number> Nonadditively code a multistate character

ccode -. Nonadditively code all multistate characters

ccode ; Display current character codings

cget x Set coding from code file x
ckeep x Save current coding in code file x

DIAGNOSING TREES

xsteps Diagnose trees in current tree file
xsteps h List possible states for hypothetical ancestors
xsteps c List character fits
xsteps m List best / worst fits
xsteps l List tree lengths
xsteps u Produce file of distinct trees

ENTERING DATA FROM THE KEYBOARD

batch Turn on batch switch
batch - Turn it off

EXITING HENNIG86 AND RETURNING TO DOS

yama Return to dos

HELP

assist Lists commands that can be used

MEMORY

bytes Display bytes of free ram

MESSAGES

quote Enter message from keyboard to be inserted into output file

OUTGROUPS

outgroup = <*taxon name*> Set outgroup to that taxon
outgroup ; List outgroup alphabetically
reroot Change root according to current outgroup

READING INPUT

procedure n open dos file n as procedure file
procedure - deactivate procedure file
procedure* activate it
procedure / close it

SAVING OUTPUT

log <*filename*> create a dos file as new log file
log - deactivate log file
log * activate log file
log / close log file

SUCCESSIVE WEIGHTING

xsteps w	set character weights according to fits

TIME CALCULATIONS

watch	Turn on stopwatch
watch -	Turn-off stopwatch

TREES

erase s	Delete tree files in scope s
files	Display directory of treefiles
get x	Make tree file x current
keep x	Save current tree file as tree file x
reroot	Current treefile according to current outgroup
tlist	Display trees in parenthetical notation
tchoose s	Select trees in scope s from current tree file
tplot	Produce tree diagrams
tread	Read trees
tsave n	Save current tree file on dos file n
txascii	Use extended ascii characters in tree plots
txascii -	Don't use extended ascii characters in tree plots

VIEWING OUTPUT

display	Short listings to monitor
display -	No listing to monitor
display*	All listing to monitor
tlist	Display trees in parenthetical notation
tplot	Produce tree diagrams
txascii	Use extended ascii characters in tree plots
txascii -	Don't use extended ascii characters in tree plots
view n	Inspect dos file n
view *	Close and inspect current log file
xx	(Dos Equis) display and modify diagnosed tree

APPENDIX B

Common Error Messages in Hennig86

Error messages in Hennig86 are easy to figure out because the symbols or words the program does not understand are printed followed by a question mark. This is followed by prompt (*****>) so that you can reenter commands or exit to edit the data file. Common error messages are given below:

1. ***<Name of A Taxon> ?***

The number of taxa in line 4 of the data file may be incorrect. The program has encountered a taxa name when it expected a character or the end of the file. The numb of taxa should include all of the taxa of both the ingroup and the outgroup. Exit Hennig86 (type yama) and edit the data file.

2. **' ?**

The command **xread** as the first line of the data file is missing. The first symbo found when reading the data file was the quotes of the title line. Exit Hennig86 (type yama) and edit the data file.

3. When there is a title in the data set but the following message appears:

```
xread no title
xread characters
```

The quotes or at least the first quote of the title line are missing. The program could not read the characters and is prompting you to enter them from the keyboard. Exit Hennig86 (type yama) and edit the data file.

4. When a procedure file is read, the title is displayed on the monitor. If the title is followed by **xread characters** (e.g.):

```
'title read correctly'
 xread characters
```

Check the number of characters in the data file. If you have more that 500, the prograr

will not read the data set. Edit the dataset to remove uninformative characters.

5. Prints out the entire file. When the command ***>H;** is entered, you get the message

```
hennig no data
*>
```

The closing quote of the title line is missing and the program treated the whole file as the title. Exit Hennig86 (type yama) and edit the data file.

6. **procedure> _**

You did not type a semicolon after the command **p <*pathname*>**. Type a semicolon and press return to go on.

7. **xread <*The First Letter of a Taxon Name*> ?**

This could be due to one of two errors. a1) Perhaps the number of characters in line three of the data file si incorrect and the program has encountered a taxa name where it expected a character state.

(b) Or you did not have a space between a taxon name and the beginning of its character states. The computer counted the first character as part of the name and thus could not find all of the character states before it encountered another name. Exit Hennig86 (type yama) and edit the data file.

8. **Printer Error** - instead of the tree that appears on the monitor printing, a string of numbers and letters connect the taxa:

```
   IM0outgroup
FM9NM1first
   :    IM2second
   HM89      IM3third
```

Your printer is not in IBM mode. Look in your printer manual to see how to change it IBM mode or run the program from within Microsoft windows.

9. **; ?**

But there is no semicolon in your data file. If your input file is large, the last comman of the file should be **procedure/;**

10. **Command Repeat** - If , when you enter a command such as **h** to create a cladogram the computer repeats the command followed by a prompt but does not execute the command:

```
hennig>
```

You simply forgot to type a semicolon at the end of the command. All you have to do enter a semicolon, press return (or enter) and the command will be executed.

11. **cc comand does not work**

If you type a character number that is greater than the number of characters in t data set, the command will not be executed for any of the characters. For example, if characters 1, 2, 3 and 4 are to be turned off for a data set of 20 characters, but the followi was accidently typed in:

```
*>cc ] 1 2 34;
ccode 34 ?
*>
```

Reenter the characters correctly and press return to execute the command.

12. **cc command affects the wrong characters**

Remember, Hennig86 calls the first character 0, the next character 1, and so forth. If a d set has 20 characters and you want to use the cc-command to turn off character 20, the correct command would be:

```
*> cc ] 19;
```

13. **cc Comand does not work on a Range of Characters**

If a dash is used to separate the range of characters, the command will not work (the dash is the symbol for making multistate characters additive or nonadditive).

Instead only the lowest and highest numbers in the range will be turned off and the highest number will be ordered nonadditively. Use a **.**(period) to indicate a range of numbers:

```
*>cc ] 6-9;
*>cc;
ccode
 0    1    2    3    4    5    6    7    8    9
1+[  1+[  1+[  1+[  1+[  1+[  1+]  1+[  1+[  1-]
```

Instead type

```
*>cc ] 6.9;
*>cc;
ccode
 0    1    2    3    4    5    6    7    8    9
1+[  1+[  1+[  1+[  1+[  1+[  1+]  1+]  1+]  1+]
```