

# **Molecular Evolution**

# Outline

- Evolutionary Tree Reconstruction
- “Out of Africa” hypothesis
- Did we evolve from Neanderthals?
- Distance Based Phylogeny
- Neighbor Joining Algorithm
- Additive Phylogeny
- Least Squares Distance Phylogeny
- UPGMA
- Character Based Phylogeny
- Small Parsimony Problem
- Fitch and Sankoff Algorithms
- Large Parsimony Problem

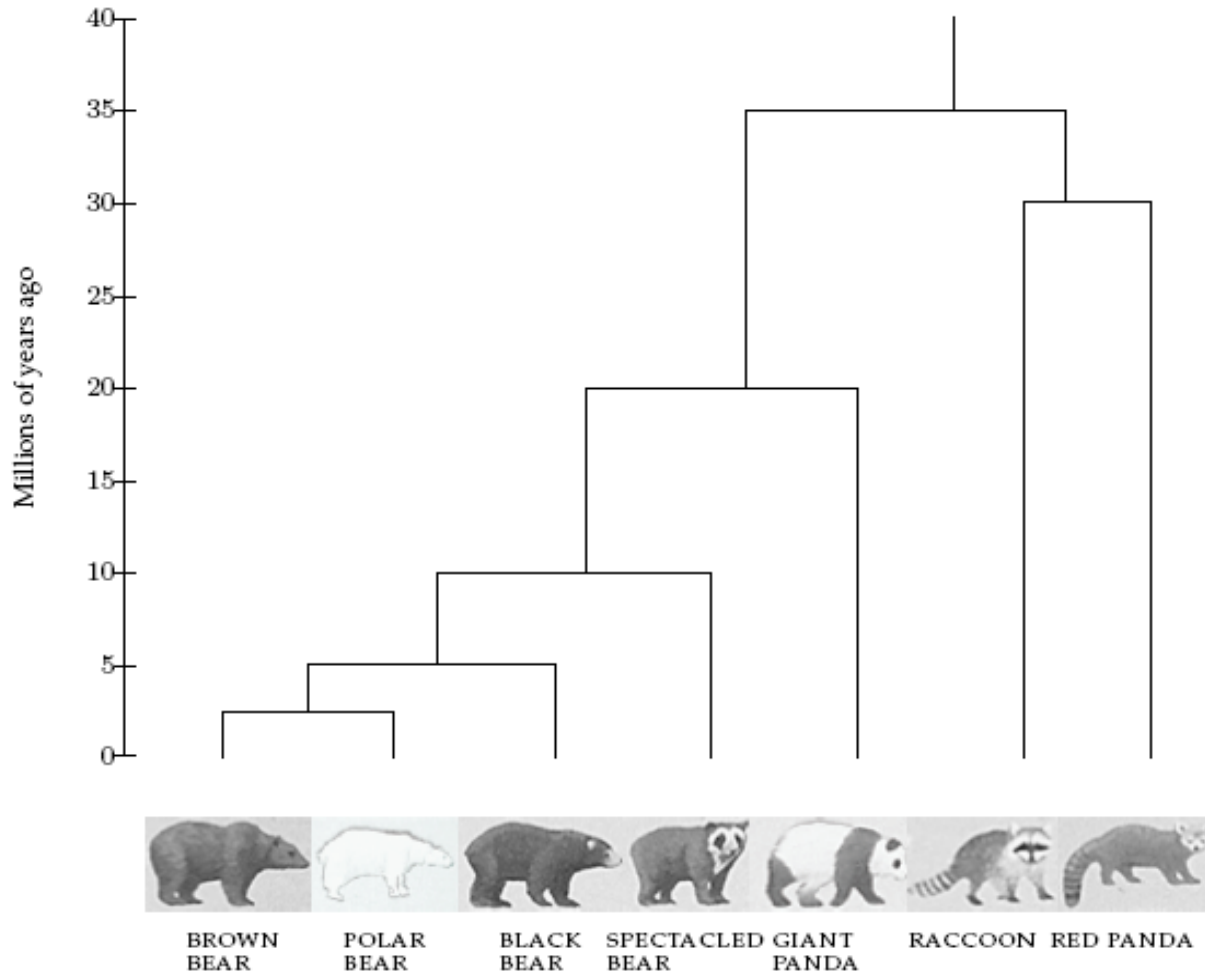
# Early Evolutionary Studies

- Anatomical features were the dominant criteria used to derive evolutionary relationships between species since Darwin till early 1960s
- The evolutionary relationships derived from these relatively subjective observations were often inconclusive. Some of them were later proved incorrect

# Evolution and DNA Analysis: the Giant Panda Riddle

- For roughly 100 years scientists were unable to figure out which family the giant panda belongs to
- Giant pandas look like bears but have features that are unusual for bears and typical for raccoons, e.g., they do not hibernate
- In 1985, Steven O'Brien and colleagues solved the giant panda classification problem using DNA sequences and algorithms

# Evolutionary Tree of Bears and Raccoons



# Evolutionary Trees: DNA-based Approach

- 40 years ago: Emile Zuckerkandl and Linus Pauling brought reconstructing evolutionary relationships with DNA into the spotlight
- In the first few years after Zuckerkandl and Pauling proposed using DNA for evolutionary studies, the possibility of reconstructing evolutionary trees by DNA analysis was hotly debated
- Now it is a dominant approach to study evolution.

# Emile Zuckerkandl on human-gorilla evolutionary relationships:



From the point of hemoglobin structure, it appears that gorilla is just an abnormal human, or man an abnormal gorilla, and the two species form actually one continuous population.

*Emile Zuckerkandl,*  
Classification and Human Evolution, 1963

# Gaylord Simpson vs. Emile Zuckerkandl:



From the point of hemoglobin structure, it appears that gorilla is just an abnormal human, or man an abnormal gorilla, and the two species form actually one continuous population.

*Emile Zuckerkandl,*  
Classification and Human Evolution, 1963



From any point of view other than that properly specified, that is of course nonsense. What the comparison really indicate is that hemoglobin is a bad choice and has nothing to tell us about attributes, or indeed tells us a lie.

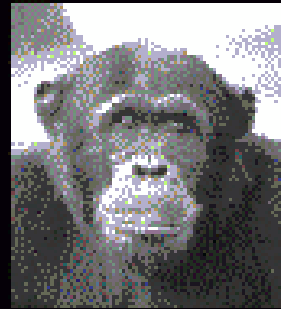
*Gaylord Simpson,*  
1964

Science,

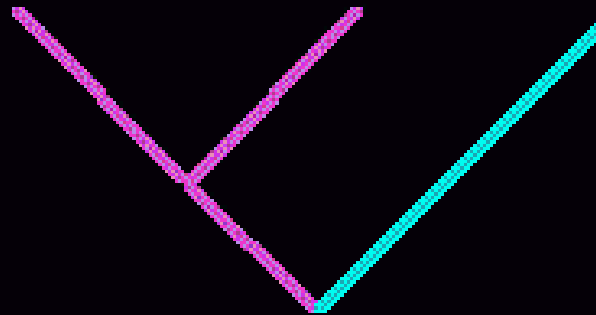


# Who are closer?

Does genetics show that humans and chimps are each other's closest relative?



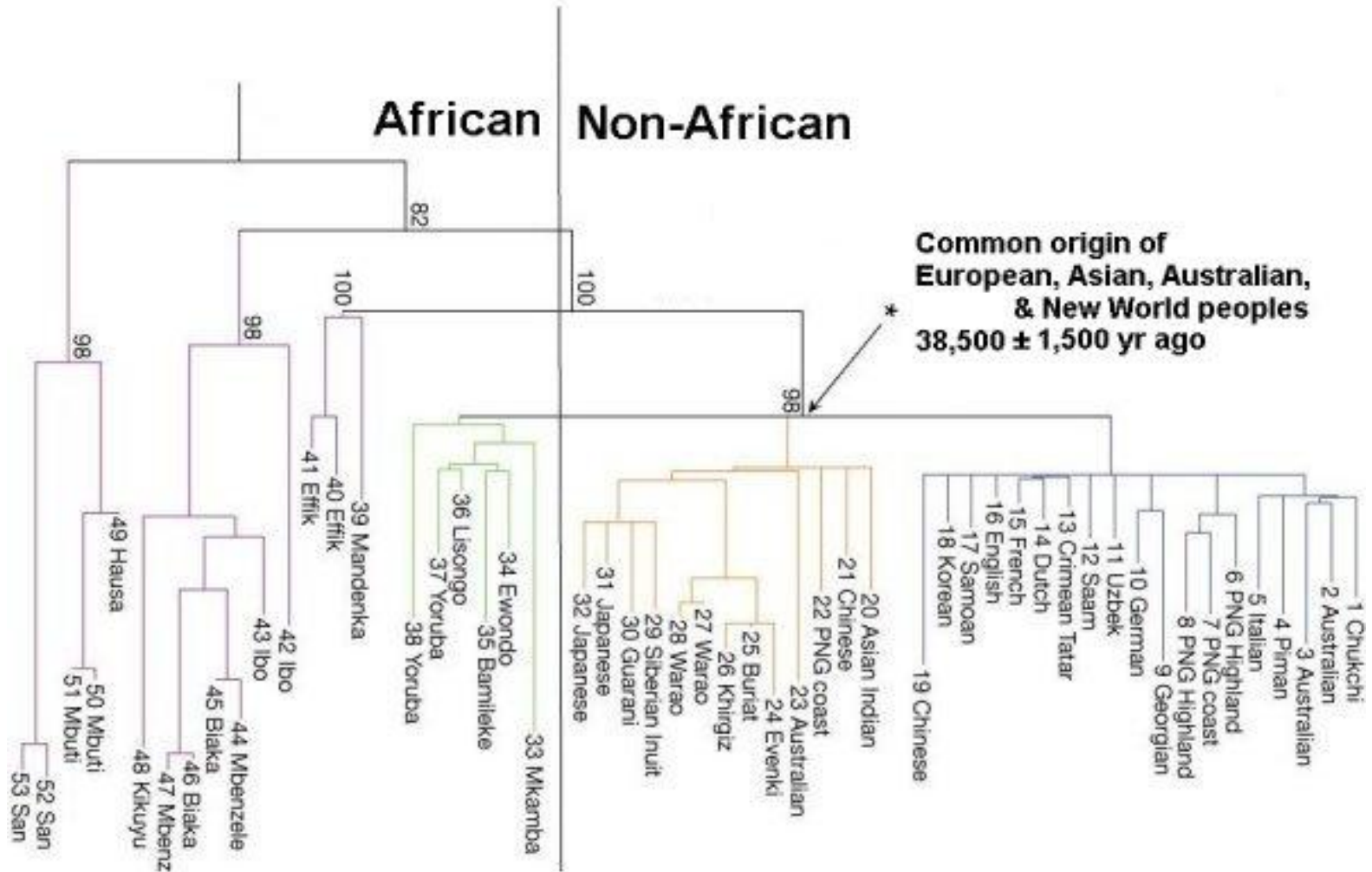
Gorillas



# Out of Africa Hypothesis

- Around the time the giant panda riddle was solved, a DNA-based reconstruction of the human evolutionary tree led to the **Out of Africa Hypothesis** that claims our most ancient ancestor lived in Africa roughly 200,000 years ago

# Human Evolutionary Tree (cont'd)



# The Origin of Humans: "Out of Africa" vs Multiregional Hypothesis

## Out of Africa:

- **Humans evolved in Africa ~150,000 years ago**
- **Humans migrated out of Africa, replacing other humanoids around the globe**
- **There is no direct descendents from Neanderthals**

## Multiregional:

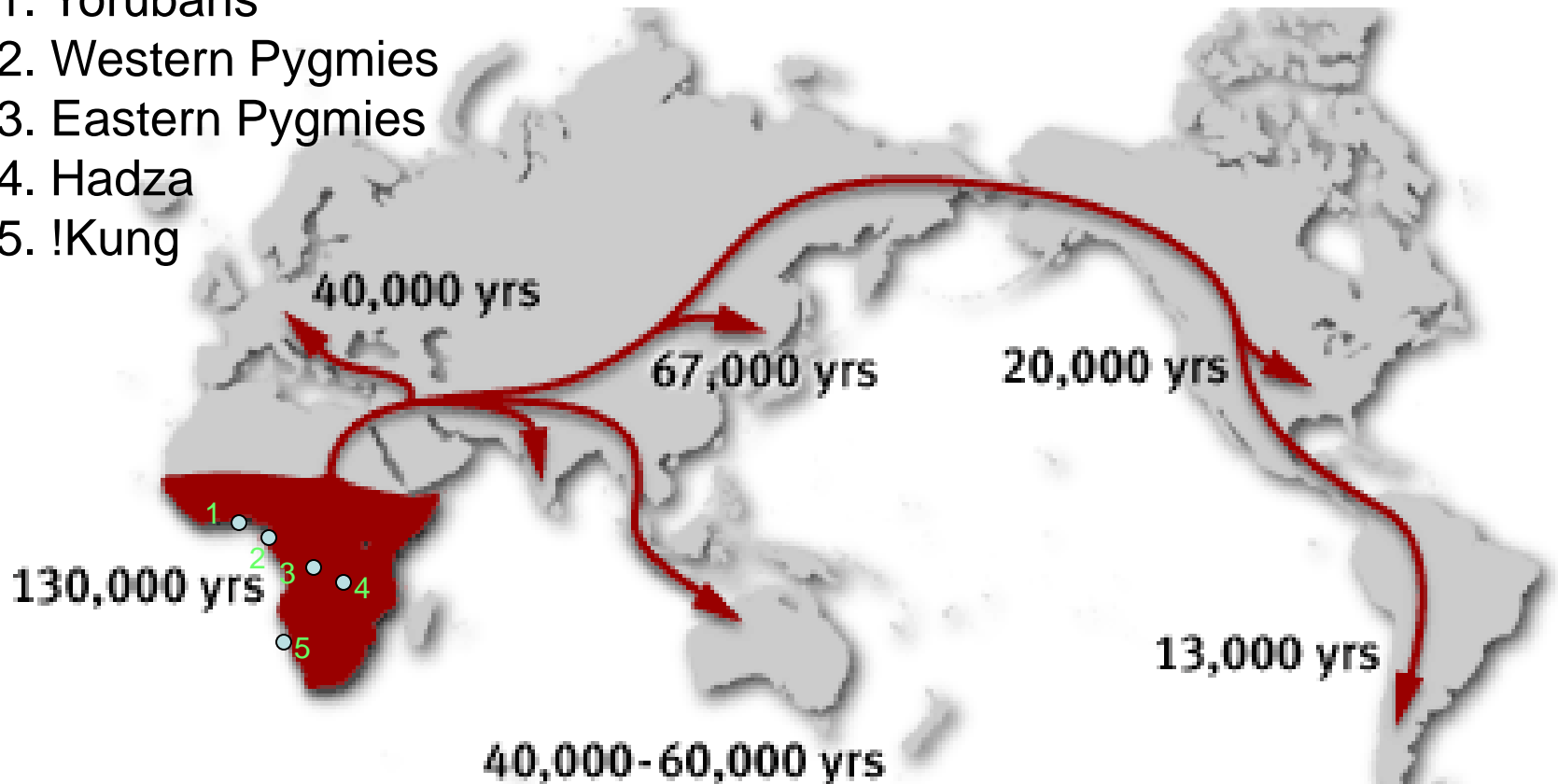
- Humans evolved in the last two million years as a single species.
- Independent appearance of modern traits in different areas
- Humans migrated out of Africa mixing with other humanoids on the way
- There is a genetic continuity from Neanderthals to humans

# mtDNA analysis supports “Out of Africa” Hypothesis

- African origin of humans inferred from:
  - African population was the most diverse (sub-populations had more time to diverge)
  - The evolutionary tree separated one group of Africans from a group containing all five populations.
  - Tree was rooted on branch between groups of greatest difference.

# Human Migration Out of Africa

1. Yorubans
2. Western Pygmies
3. Eastern Pygmies
4. Hadza
5. !Kung



# Two Neanderthal Discoveries

Feldhofer,  
Germany

Mezmaiskaya,  
Caucasus

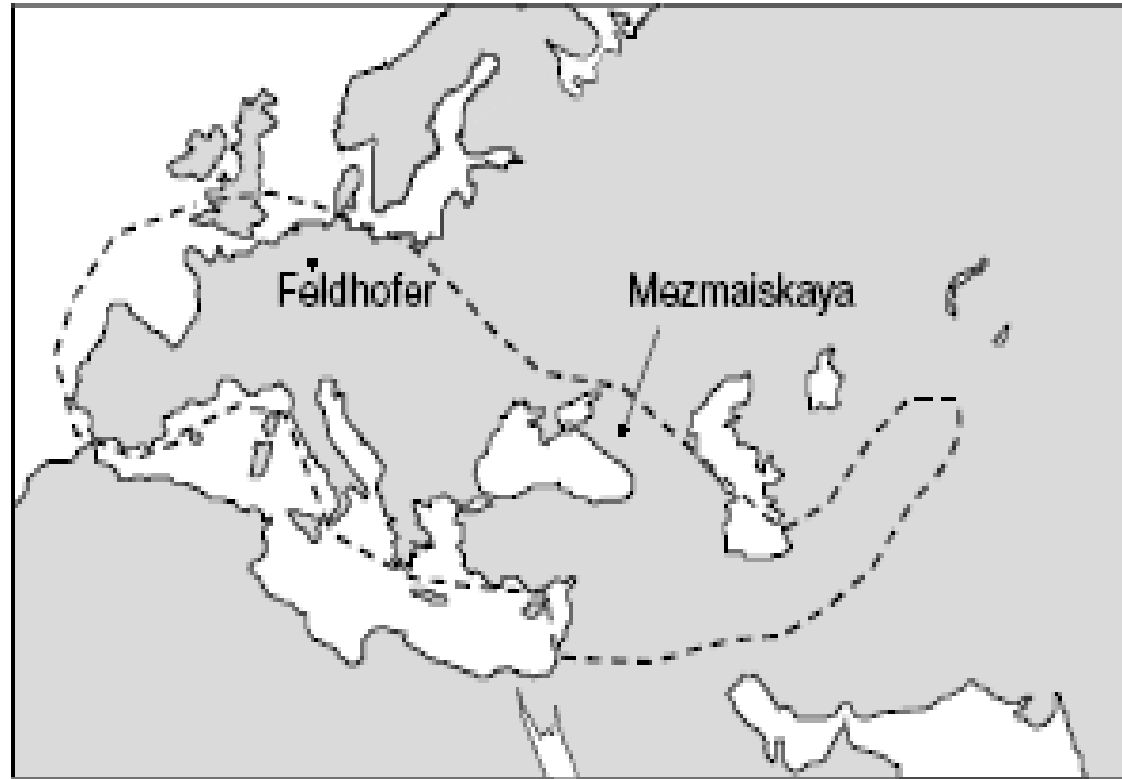
Distance:  
25,000km



# Two Neanderthal Discoveries



Figure 1. Illustration of Neanderthal Man  
(Reprinted by permission from John Gurche/National Geographic.)



- *Is there a connection between Neanderthals and today's Europeans?*
- *If humans did not evolve from Neanderthals, whom did we evolve from?*



# Multiregional Hypothesis?

- Many predict some genetic continuity from the Neanderthals through to the Cro-Magnons up to today's Europeans
- Can explain the occurrence of varying regional characteristics

# Evolutionary trees

- A tree with leaves = species, and edge lengths representing evolutionary time
- Internal nodes also species: the ancestral species
- Also called “phylogenetic tree”
- How to construct such trees from data?

# Tree Construction Algorithms

The evolutionary tree construction algorithms can be broadly divided into two basic categories:

1. **Distance based methods**: input consists of evolutionary distance data and output is a weighted tree whose pair-wise distances “agree” with the given evolutionary data. If the data is *ultrametric*, there is an elegant solution. If the data is not ultrametric but *additive*, there is also an efficient algorithm. Otherwise, “approximate” algorithms have to be developed.

**2. Maximum Parsimony methods:** These are “character” ( an observable trait or characteristic, which sometime may be linked to a DNA or amino acid sequence, but need not always one).

The goal is to build a tree, whose leaves represent the *taxa* or species and the internal nodes represent ancestral taxa such that the total number of “mutations “ implied by the tree is **minimized** . This is also referred to as “**maximizing parsimony**” in the literature. If the input taxa are represented by molecular sequences, the tree corresponds to a *Phylogenetic alignment* with minimum cost over all trees.

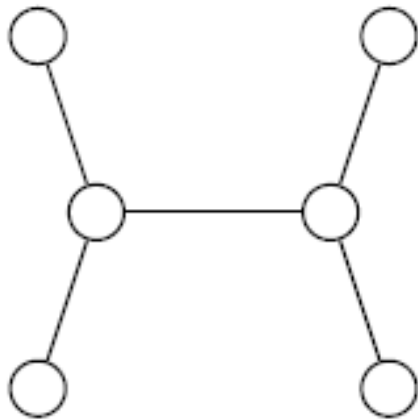
# Evolutionary Trees

*How are these trees built from DNA sequences?*

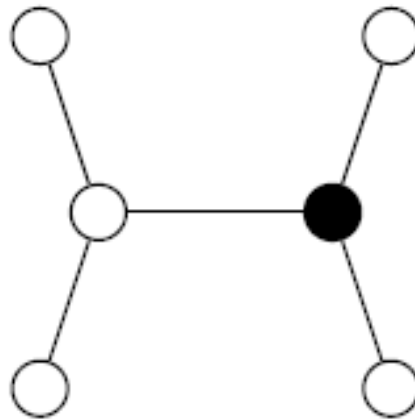
- leaves represent existing species
- internal vertices represent ancestors
- If the tree is “rooted”, then the root represents the oldest evolutionary ancestor

# Rooted and Unrooted Trees

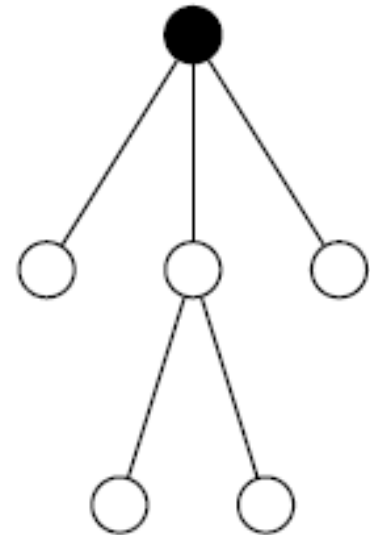
In the unrooted tree the position of the root (“oldest ancestor”) is unknown. Otherwise, they are like rooted trees



(a) Unrooted tree



(b) Rooted tree



(c) The same rooted tree

# Weighted Binary Trees

- Internal vertices have degree 3
- Edges  $(v, w)$  have weights reflecting:
  - # of mutations on the evolutionary path from  $v$  to  $w$  ( one species to another )
  - or
  - Time estimate for evolution from  $v$  to  $w$  ( one species to another).

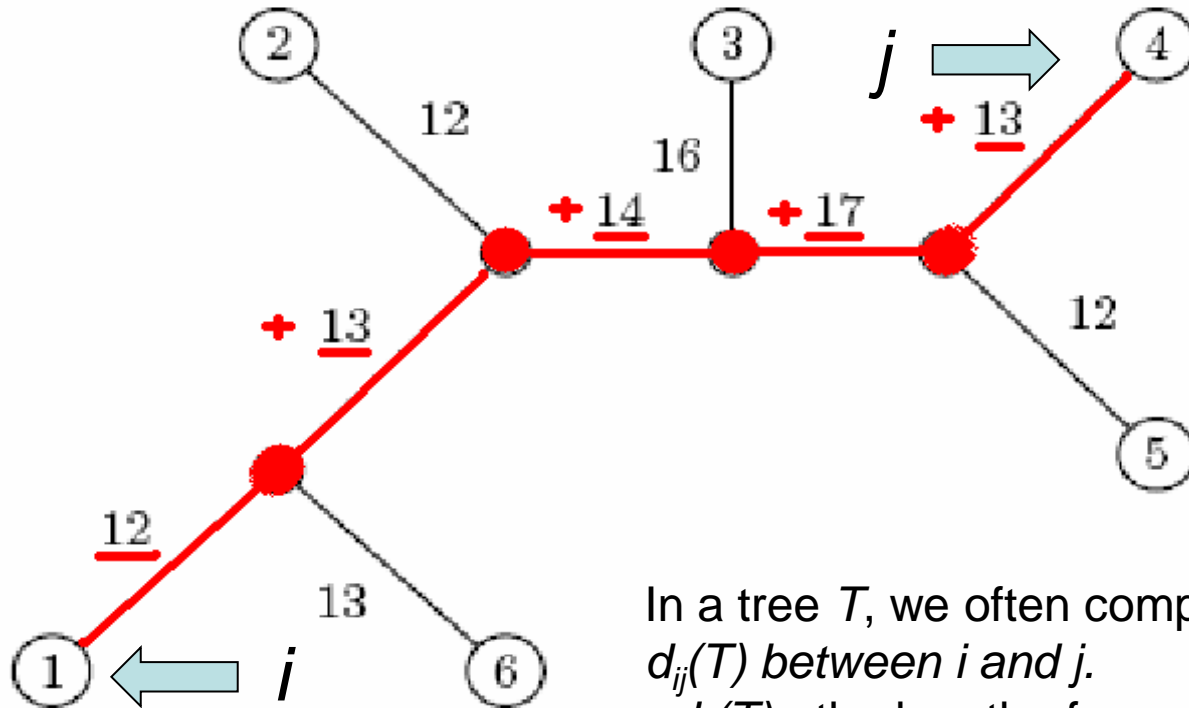
# Molecular Clock

- $t(v)$  = time corresponding to the “moment” when the species  $v$  produced its descendents
- $t(w)$  = time corresponding to the “moment” when the species  $w$  produced its descendents
- $t(w) - t(v)$  = weight of the edge  $(v, w)$
- Every leaf has time  $t=0$
- Every internal node has “past” or negative time.



# **Distance Based Methods**

# Distance in Trees: an Example



In a tree  $T$ , we often compute *tree distance*  $d_{ij}(T)$  between  $i$  and  $j$ .

$d_{ij}(T)$  - the length of a path between leaves  $i$  and  $j$

$$d_{1,4} = 12 + 13 + 14 + 17 + 13 = 69$$

# Distance Matrix

- Given  $n$  species, we can compute the  $n \times n$  ***distance matrix***  $D_{ij}$
- $D_{ij}$  may be defined as the **edit distance** between a gene in species  $i$  and a gene in species  $j$ . We assume that the genes of interest have been sequenced for all  $n$  species.

Or

- It could be some other experimental data viz. gene expression level difference in a microarray

# Fitting Distance Matrix

- Given  $n$  species, we can compute the  $n \times n$  **distance matrix**  $D_{ij}$
- Evolution of these genes is described by a tree that **we don't know**. Let  $T$  be such a tree and
- $d_{ij}(T)$  – **tree distance between  $i$  and  $j$**
- We need an algorithm to construct a tree that best **fits** the distance matrix  $D_{ij}$

# Fitting Distance Matrix

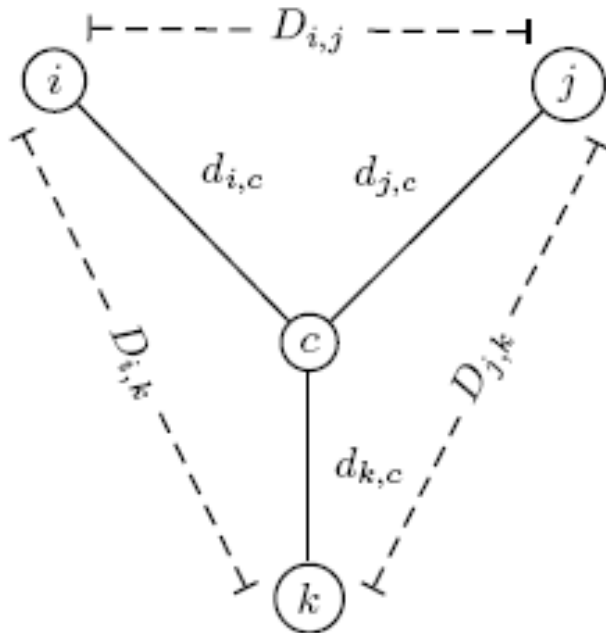
Lengths of path in an (*unknown*) tree  $T$

• Fitting means  $\underbrace{D}_{ij} = \underbrace{d}_{ij}(T)$

Edit distance between species (*known*)

# Reconstructing a Tree with 3 Leaf nodes

- Tree reconstruction for any 3x3 matrix is straightforward
- We have 3 leaf nodes  $i, j, k$  and an internal center node  $c$



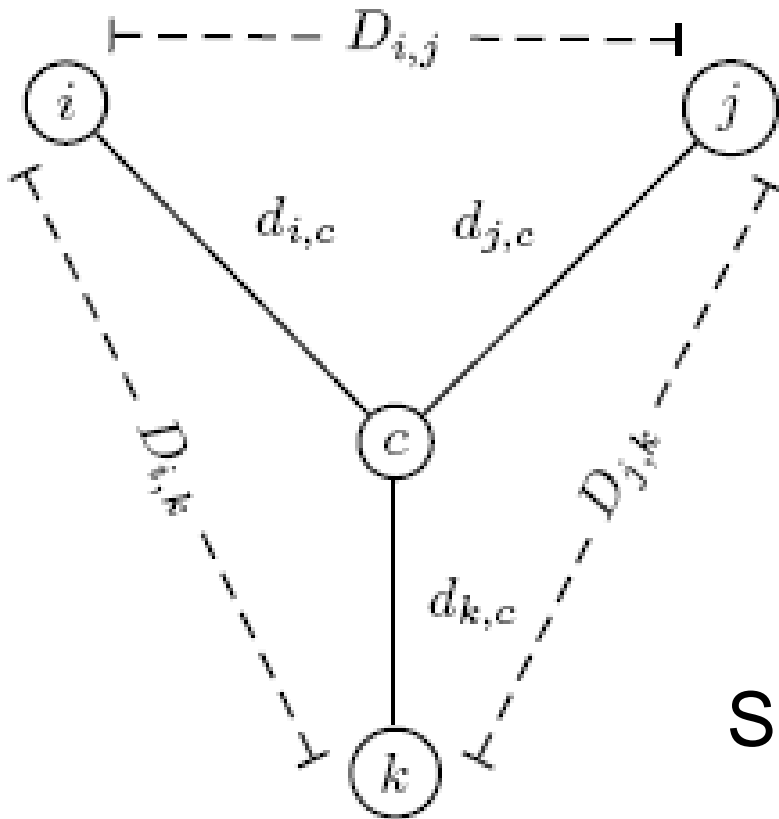
Observe:

$$d_{ic} + d_{jc} = D_{ij}$$

$$d_{ic} + d_{kc} = D_{ik}$$

$$d_{jc} + d_{kc} = D_{jk}$$

# Reconstructing a 3 Leaved Tree (cont'd)



$$d_{ic} + d_{jc} = D_{ij}$$

$$+ \underline{d_{ic}} + \underline{d_{kc}} = \underline{D_{ik}}$$

$$2d_{ic} + \underbrace{d_{jc} + d_{kc}}_{D_{jk}} = D_{ij} + D_{ik}$$

$$2d_{ic} + D_{jk} = D_{ij} + D_{ik}$$

$$d_{ic} = (D_{ij} + D_{ik} - D_{jk})/2$$

Similarly,

$$d_{jc} = (D_{ij} + D_{jk} - D_{ik})/2$$


$$d_{kc} = (D_{ki} + D_{kj} - D_{ij})/2$$

# Trees with $> 3$ Leaf nodes

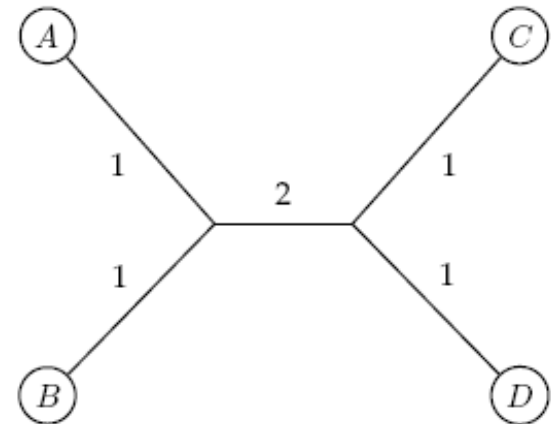
- A tree with  $n$  leaves has  $2n-3$  edges
- This means fitting a given tree to a distance matrix  $D$  requires solving a system of “ $n$  choose 2” equations with  $2n-3$  variables, which is not always possible to solve for  $n > 3$ .
- For example, if  $n=4$ , we have a system of 6 equations with  $2n-3=5$  variables
- But, there is a special case when these equations can be solved: it is the case for the **additive tree**.



# Additive Distance Matrices

Matrix  $D$  is  ADDITIVE if there exists a tree  $T$  with  $d_{ij}(T) = D_{ij}$

	A	B	C	D
A	0	2	4	4
B	2	0	4	4
C	4	4	0	2
D	4	4	2	0



NON-ADDITIVE otherwise 

	A	B	C	D
A	0	2	2	2
B	2	0	3	2
C	2	3	0	2
D	2	2	2	0

?

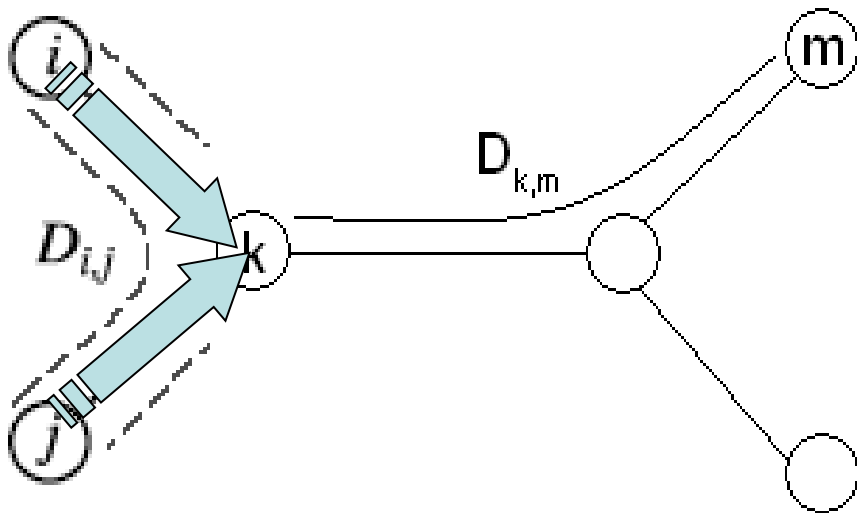
# Distance Based Tree Problem

- Goal: Reconstruct an evolutionary tree from a distance matrix
- Input:  $n \times n$  distance matrix  $D_{ij}$
- Output: weighted tree  $T$  with  $n$  leaves fitting  $D$
- If  $D$  is additive, this problem has a solution and there is a simple algorithm to solve it

# Using Neighboring Leaves to Construct the Tree

- Find **neighboring leaves**  $i$  and  $j$  with parent  $k$
- Remove the rows and columns of  $i$  and  $j$
- Add a new row and column corresponding to  $k$ , where the distance from  $k$  to any other leaf  $m$  can be computed as:  $2D_{km} = D_{im} + D_{jm} - D_{ij}$

$$D_{km} = (D_{im} + D_{jm} - D_{ij})/2$$



Compress  $i$  and  $j$  into  $k$ . Iterate for rest of tree

# Finding Neighboring Leaves

- But, how to find the neighboring leaves of an unknown tree?
- Select a pair of closest leaves?

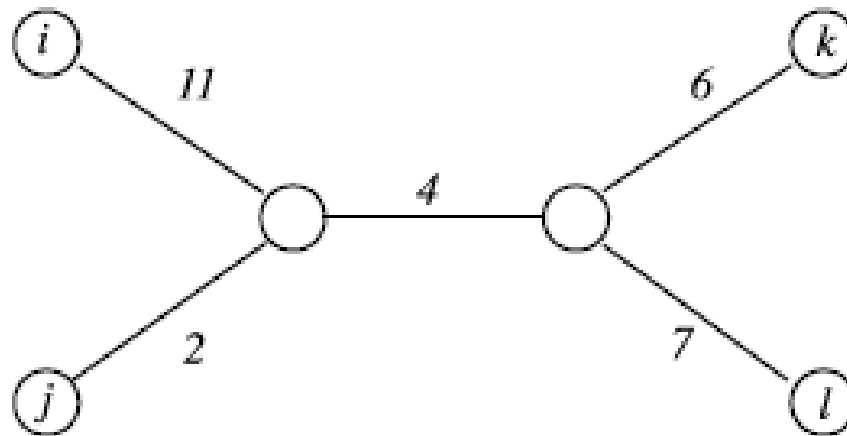
# Finding Neighboring Leaves

- But, how to find the neighboring leaves of an unknown tree?
- Select a pair of closest leaves?

WRONG

# Finding Neighboring Leaves

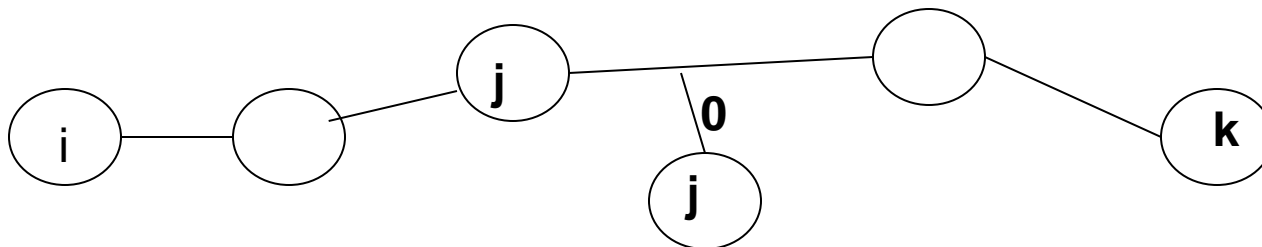
- Closest leaves aren't necessarily neighbors
- $i$  and  $j$  are neighbors, but  $(d_{ij} = 13) > (d_{jk} = 12)$



- Finding a pair of neighboring leaves is a nontrivial problem! For the moment, we postpone this approach.

# Degenerate Triples

- A degenerate triple is a set of three distinct elements  $1 \leq i, j, k \leq n$  where  $D_{ij} + D_{jk} = D_{ik}$
- Element  $j$  in a degenerate triple  $i, j, k$  lies on the evolutionary path from  $i$  to  $k$  (or is attached to this path by an edge of length 0).



# Looking for Degenerate Triples

- If distance matrix  $D$  **has** a degenerate triple  $i,j,k$  then  $j$  can be “removed” from  $D$  thus reducing the size of the problem.
- If distance matrix  $D$  **does not have** a degenerate triple  $i,j,k$ , one can “create” a degenerate triple in  $D$  by shortening all hanging edges (edge leading to a leaf) in the tree.



# Shortening Hanging Edges to Produce Degenerate Triples

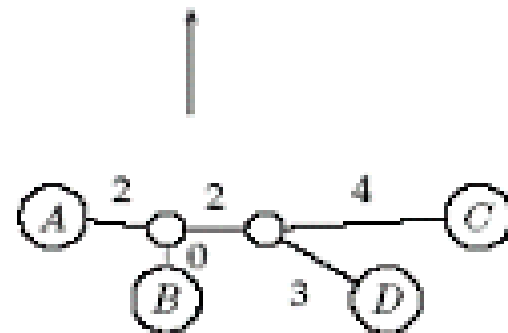
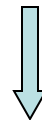
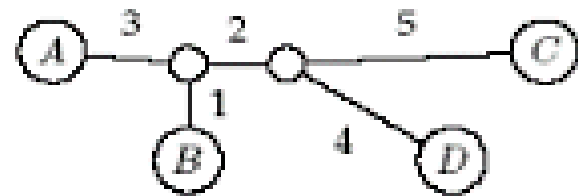
- Shorten all “hanging” edges (edges that connect leaves) until a degenerate triple is found

	A	B	C	D
A	0	4	10	9
B	4	0	8	7
C	10	8	0	9
D	9	7	9	0

↓  $\delta = 1$

	A	B	C	D
A	0	2	8	7
B	2	0	6	5
C	8	6	0	7
D	7	5	7	0

$i \leftarrow A$   
 $j \leftarrow B$   
 $k \leftarrow C$



# Finding Degenerate Triples

- If there is no degenerate triple, all hanging edges are reduced by the same amount  $\delta$ , so that all pairwise distances in the matrix are reduced by  $2\delta$ .
- Eventually this process collapses one of the leaves (when  $\delta$  equals the length of shortest hanging edge), forming a degenerate triple  $i,j,k$  and reducing the size of the distance matrix  $D$ .
- The attachment point for  $j$  can be recovered in the reverse transformations by saving  $D_{ij}$  for each collapsed leaf.

# Reconstructing Trees for Additive Distance Matrices.

	A	B	C	D
A	0	4	10	9
B	4	0	8	7
C	10	8	0	9
D	9	7	9	0

$\delta = 1$

	A	B	C	D
A	0	2	8	7
B	2	0	6	5
C	8	6	0	7
D	7	5	7	0

$\downarrow$

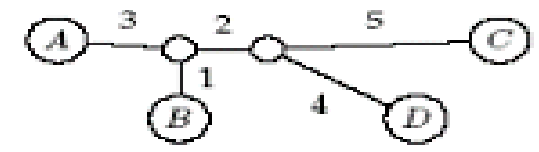
	A	C	D
A	0	8	7
C	8	0	7
D	7	7	0

$\delta = 3$

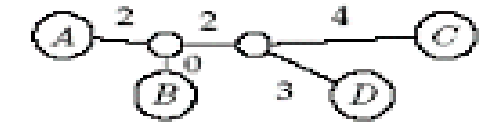
	A	C	D
A	0	2	1
C	2	0	1
D	1	1	0

$\downarrow$

	A	C
A	0	2
C	2	0

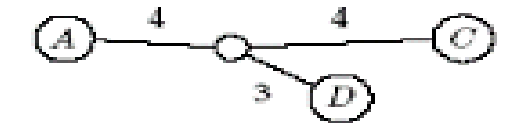


$\downarrow$

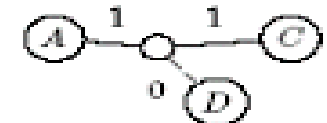


i ↑ A  
j ↑ B  
k ↑ C

$\downarrow$



$\downarrow$



i ↑ A  
j ↑ D  
k ↑ C

$\downarrow$



Actually, during construction, you cannot see the graph. The graph is obtained during the reverse construction starting from  $T_2$  to  $T_3$  until  $T_n$ . During forward phase, you only change the distance matrix.

# Shortening of the Hanging Edge

If we reduce length of every *hanging edge* (an edge leading to a leaf) by an amount  $\delta$ , then the distance matrix of the resulting tree becomes  $(d_{ij}-2\delta)$  since the distance between any two leaves is reduced by  $2\delta$ . If we continue this process, it will eventually lead to the “collapse” of one of the leaves when  $\delta$  is equal to the length of the shortest hanging edge. At this point, the tree  $T=T_n$  with  $n$  leaves will be transformed into a tree with  $n-1$  leaves,  $T_{n-1}$ . Thus the transformations  $T_n \rightarrow T_{n-1} \rightarrow \dots \rightarrow T_3 \rightarrow T_2$  is possible. Since  $T_2$  is a tree with only one edge, we can easily construct it and then reverse the transformation as  $T_2 \rightarrow T_3 \rightarrow \dots \rightarrow T_{n-1} \rightarrow T_n$  recovering the information about collapsed edges at every step. We now make a more formal description of the algorithm.

# Additive Tree Algorithm

1. **Additive**( $D$ )
2.   **if**  $D$  is a  $2 \times 2$  matrix
3.      $T =$  tree of a single edge of length  $D_{1,2}$
4.     **return**  $T$
5.   **if**  $D$  is non-degenerate
6.      $\delta =$  trimming parameter of matrix  $D$
7.     **for all**  $1 \leq i \neq j \leq n$
8.          $D_{ij} = D_{ij} - 2\delta$
9.   **else**
10.      $\delta = 0$

# Additive Algorithm (cont'd)

11. Find a triple  $i, j, k$  in  $D$  such that  $D_{ij} + D_{jk} = D_{ik}$
12.  $x = D_{ij}$
13. Remove  $j^{\text{th}}$  row and  $j^{\text{th}}$  column from  $D$
14.  $T = \text{Additive}(D)$
15. Add a new vertex  $v$  to  $T$  at distance  $x$  from  $i$  to  $k$
16. Add  $j$  back to  $T$  by creating an edge  $(v, j)$  of length 0
17. **for** every leaf  $l$  in  $T$
18.     **if** distance from  $l$  to  $v$  in the tree  $\neq D_{l,j}$
19.         output “matrix is not additive”
20.     **return**
21. Extend all “hanging” edges by length  $\delta$
22. **return**  $T$

# Additive Algorithm(Cont'd)

- This algorithm checks if the matrix  $D$  is additive, and if so, returns the tree  $T$ .
- How to compute the trimming parameter  $\delta$  ?
- The algorithm has  $O(n^4)$  time complexity because the matrix has to be tested for degeneracy for all triplets taking  $O(n^3)$  time
- And the recursive call takes place  $O(n)$  times.
- As we will see later we can construct an additive tree via ultrametric tree construction.

# The Four Point Condition

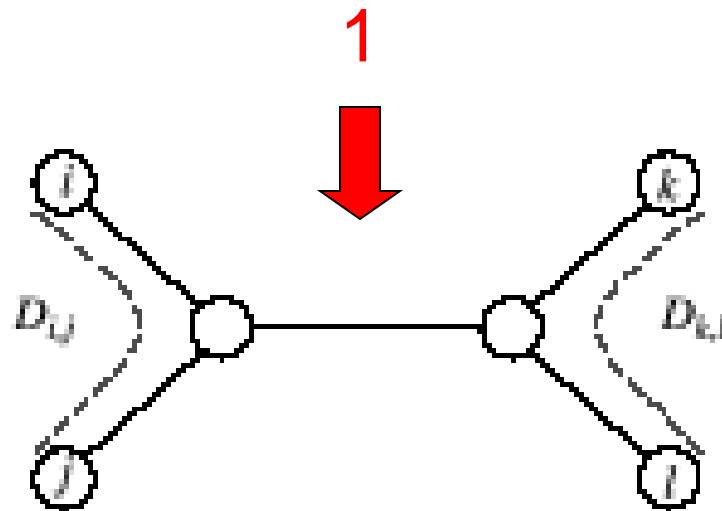
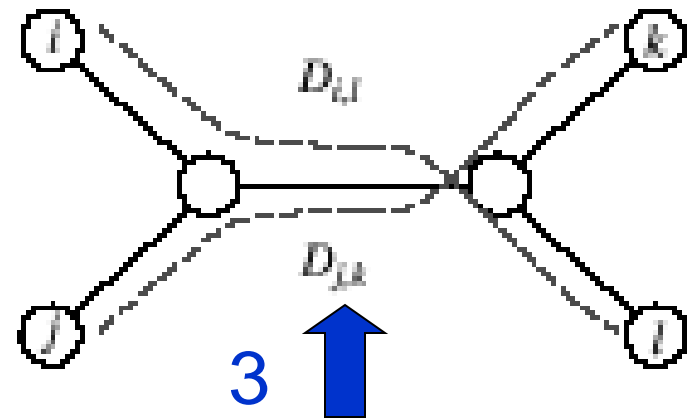
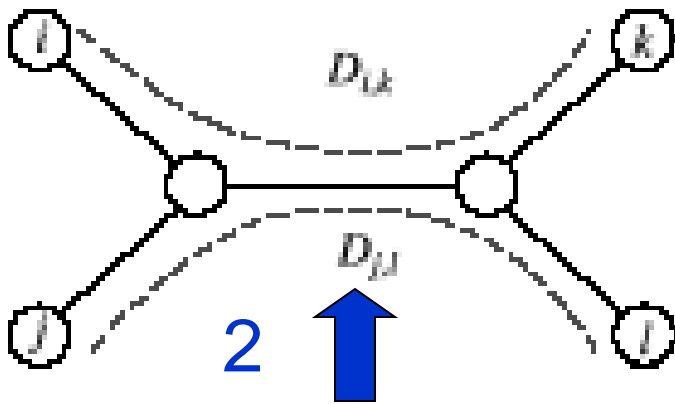
- Additive Algorithm provides a way to check if distance matrix  $D$  is additive
- **An even more efficient additivity check is the “four-point condition”**
- Let  $1 \leq i, j, k, l \leq n$  be four distinct leaves in a tree



# The Four Point Condition (cont'd)

Let  $1 \leq i, j, k, l \leq n$  be four distinct indices. Compute the 3 sums

Compute: 1.  $D_{ij} + D_{kl}$ , 2.  $D_{ik} + D_{jl}$ , 3.  $D_{il} + D_{jk}$



2 and 3 represent the same number: the length of all edges + the middle edges (they are counted twice)

1 represents a smaller number: the length of all edges skipping the middle edge

# The Four Point Condition

- If  $D$  is an additive matrix then these three sums can be represented by a tree with four leaves as shown in the previous slide.
- The elements  $1 \leq i, j, k, l \leq n$  satisfy the four-point condition if two of the sums  $D_{ij} + D_{kl}$ ,  $D_{il} + D_{jk}$  are same and the third sum  $D_{ik} + D_{jl}$  is smaller than these two.

# The Four Point Condition: Theorem

- The four point condition for the quartet  $i,j,k,l$  is satisfied if two of these sums are the same, with the third sum smaller than these first two
- **Theorem** : An  $n \times n$  matrix  $D$  is additive if and only if the four point condition holds for **every** quartet  $1 \leq i,j,k,l \leq n$
- An algorithm to construct an additive tree based on four point theorem has complexity  $O(n^4)$  and is very inefficient.
- (We will return to Additive Tree construction after we discuss Ultrametric Trees)

# Least Squared Distance Problem

- If the distance matrix  $D$  is NOT additive, then we look for a tree  $T$  that approximates  $D$  the best:

$$\textbf{Squared Error} : \sum_{i,j} (d_{ij}(T) - D_{ij})^2$$

Squared Error is a measure of the quality of the fitness between distance matrix and the tree: we want to minimize it.

- **Least Squares Distance Problem:** Given the input  $n \times n$  distance matrix  $(D_{i,j})$ , find a weighted tree  $T$  with  $n$  leaves minimizing **Squared Error**

$$\sum_{i,j} (d_{ij}(T) - D_{ij})^2$$

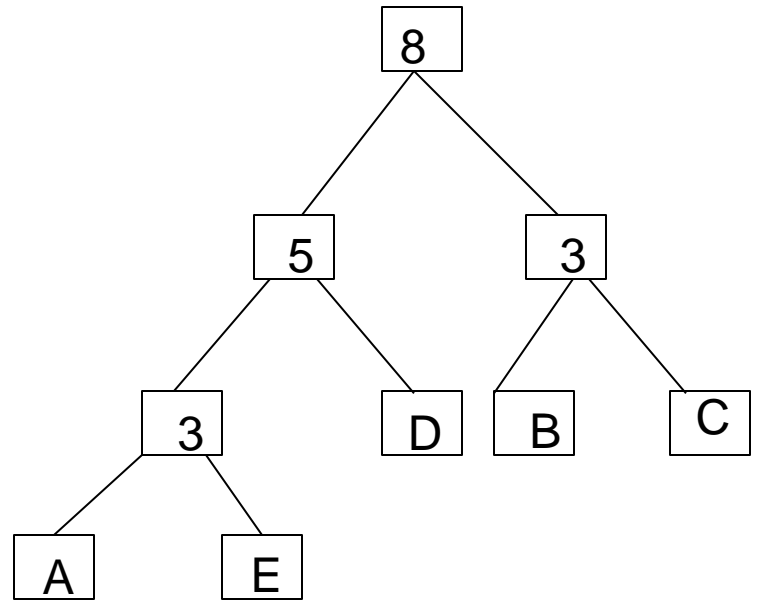
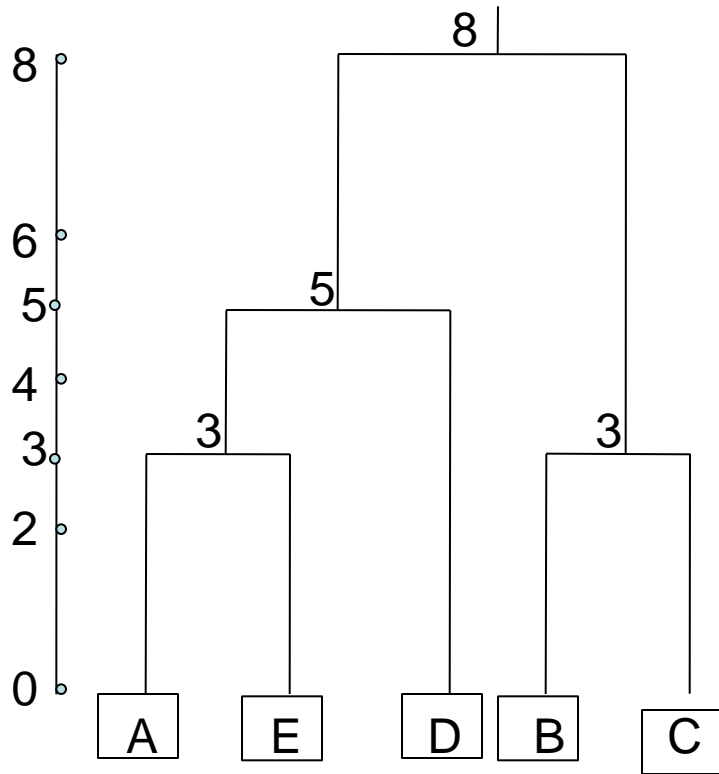
- finding the best approximation tree  $T$  for a non-additive matrix  $D$  is NP-hard.

# Ultrametric Tree and Distances

Let  $D$  be a  $n \times n$  symmetric matrix with real numbers. An **ultrametric tree** for  $D$  is a rooted tree  $T$  with the following properties:

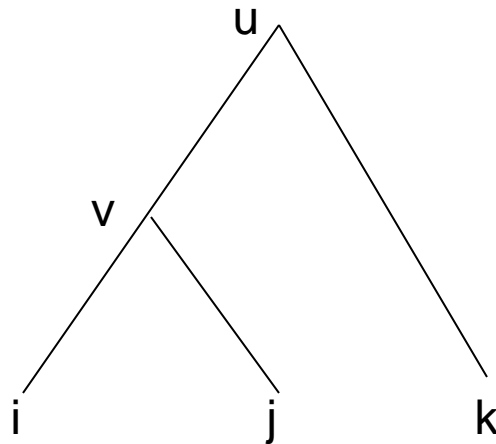
1.  $T$  has  **$n$  leaves**, each labeled by a **unique row of  $D$**
  2. Each internal node of  $T$  is labeled by one real number from  $D$  and has **at least two children**.
  3. Along any path in the tree from root to a leaf, the numbers are **strictly decreasing**.
  4. For any two leaves  $i$  and  $j$ ,  **$D(i,j)$  is the label of the least common ancestor of  $i$  and  $j$  in  $T$ .**
- ( For simplicity, assume  $D(i,i) \neq D(i,j)$  for all  $j \neq i$ , and

	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



# “Three Point Condition” – Ultrametric Matrix

**Definition:** A symmetric matrix  $D$  of real numbers defines an **ultrametric distance** if for any three indices  $i, j$  and  $k$ , there is a tie for the maximum of  $D(i,j)$ ,  $D(i,k)$  and  $D(j,k)$ . That is, two of the distances are equal and the third is less than the maximum.



# The Basic Theorem

**Theorem:** *A symmetric matrix  $D$  has an ultrametric tree if and only if  $D$  is an ultrametric matrix.*

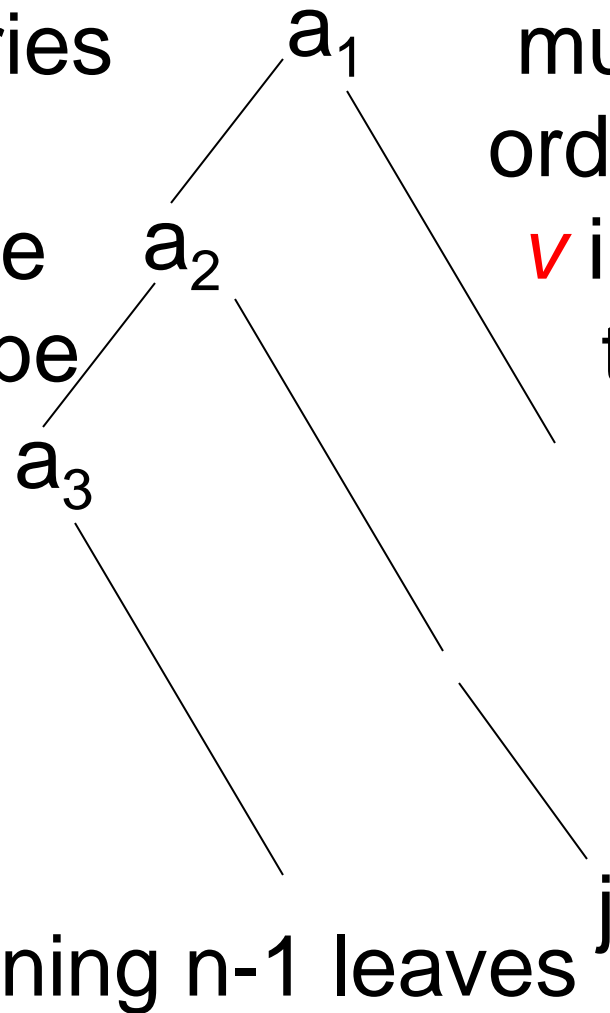
It is easy to see that if  $D$  has an ultrametric tree then  $D$  is an ultrametric matrix. The converse is proved by giving a constructive algorithm to obtain an ultrametric tree  $T$  for  $D$ .



If there are  $p$  distinct entries in row  $i$  of  $D$  then any ultrametric tree  $T$  for  $D$  must have a path from the root to the leaf  $i$  with exactly  $p$  nodes, and the entries

decreasing  
Internal node  
 $D(i, j)$  must be  
ancestor

This fixes  
appear  
path  
path



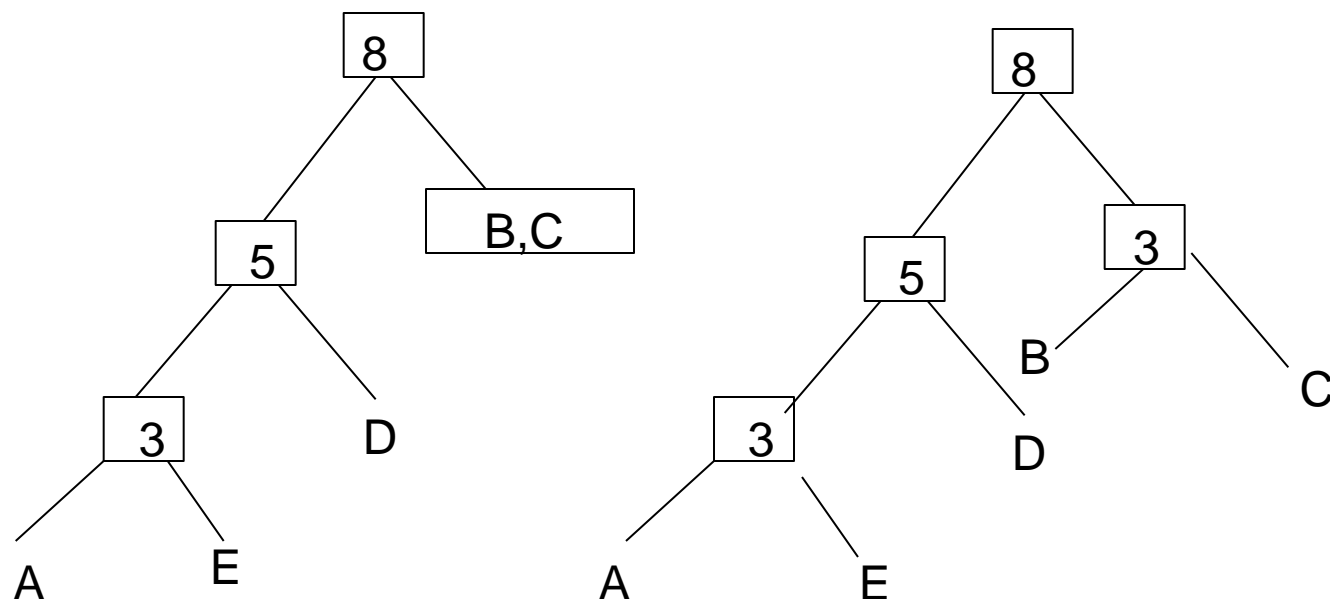
must appear in strict order. Furthermore, any  $v$  in this path labeled the least common of leaf  $i$  and leaf  $j$ , where leaf  $j$  must in  $T$  relative to the of leaf  $i$ . Thus, the  $i$  partitions the in  $p-1$  classes.

Call this partition  $P$ . Leaf nodes  $j$  and  $k$  belong to the same partition of  $P$  if and only if  $D(i, j) = D(i, k)$ . Thus, each node in the path to  $i$  gives rise to a distinct partition in  $P$ .

We could now solve the ultrametric tree problem recursively on each class and connect these trees to form the ultrametric tree for the full matrix  $D$

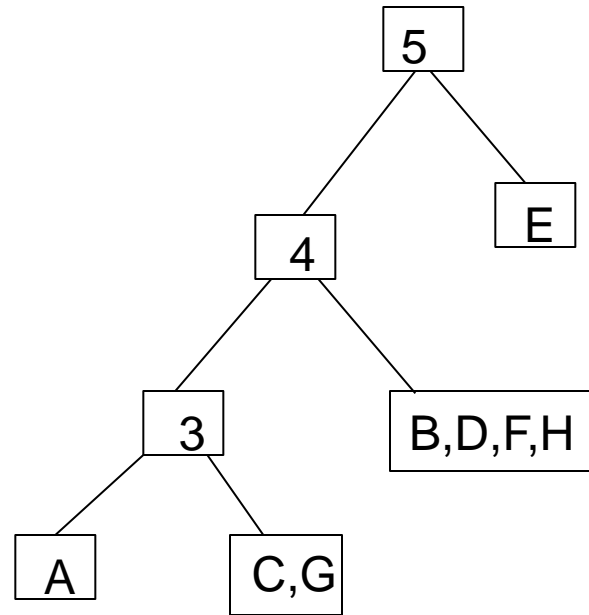
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0

A Symmetric ultrametric Matrix



An ultrametric tree for the matrix.

	A	B	C	D	E	F	G	H
A	0	4	3	4	5	4	3	4
B								
C								
D								
E								
F								
G								
H								



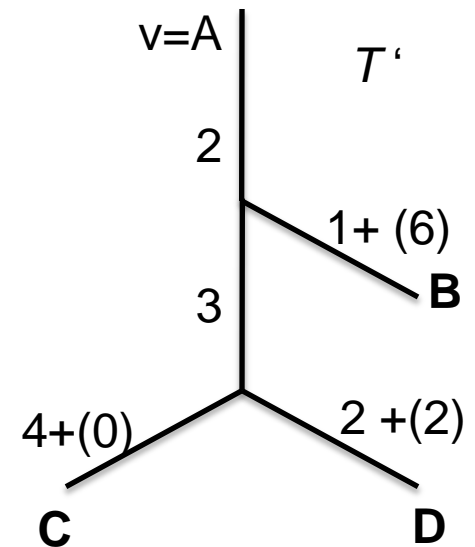
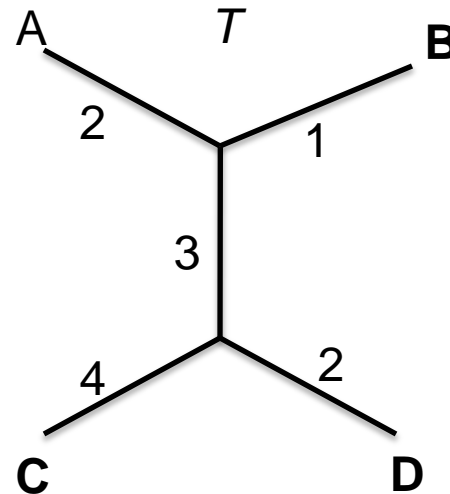
Without loss of generality, we can make the first row of the matrix to be one of the rows having the maximum value of all the entries in the matrix. Label this row to be A. Its distinct set of non-zero real numbers are (3,4,5). Form a path from root of the tree with strictly decreasing numbers (5,4,3) ending with the leaf node labeled A. The set of columns associated with numbers (5,4,3) viz. [(E), (B,D,F,H), (C,G)] are attached as right child of the internal nodes 5, 4 and 3, respectively. The resulting tree is ultrametric if and only if the submatrices formed by the right child sets are in their turn also ultrametric. Repeat the construction step recursively for each of the subsets for the right children. The tree is unique – if  $D$  is an ultrametric matrix then the ultrametric tree for  $D$  is unique. **The construction time for building the tree is  $O(n^2)$ . Why?**

## The Transformations $D \rightarrow T \rightarrow T' \rightarrow D'$ : Reduction of Additive Tree problem to Ultrametric Tree problem in time $O(n^2)$

Assume  $D$  is additive and  $T$  is the corresponding additive tree<sup>\*\*</sup>. Let  $v$  be the row in  $D$  with max entry (viz. row A, Max value  $m_v=9$ ). Construct a tree  $T'$  which is a sort of a copy of  $T$  with node rooted at  $v$  marked as the root and all leafs are equidistant from the root by adding the difference  $m_v - D(v,i)$  to each edge leading to leaf  $i$ . These differences for nodes B,C and D are 6,0 and 2, respectively. Also, note that each internal node is equidistant to any leaf node in its subtree.

	A	B	C	D
A	0	3	9	7
B		0	8	6
C			0	6
D				0

$D$

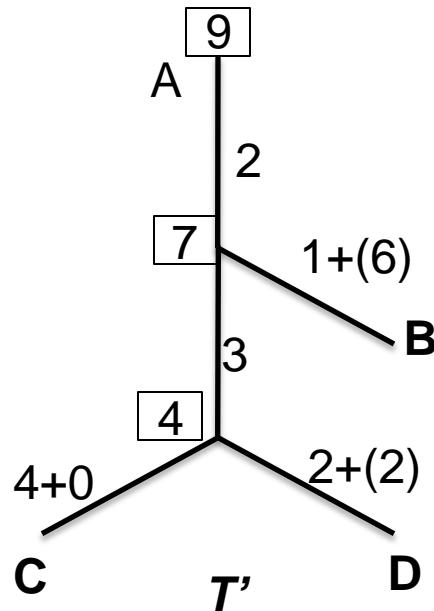


<sup>\*\*</sup> We can use the **AdditivePhylogeny( $D$ )** algorithm for this purpose now. But, we are going to develop a more efficient algorithm in the following slides.

# The Transformations $D \rightarrow T \rightarrow T' \rightarrow D'$ (contd.)

Now, label each internal node in  $T'$  with the unique distance from it to any of the leaves in the Tree. These are 9, 7 and 4 for our example, which are non-increasing.

We can now define an ultrametric matrix  $D'$  where  $D'(i,j)$  is the label at the least common ancestor of leaves  $i$  and  $j$  in  $T'$ .



	A	B	C	D
A	0	9	9	9
B		0	7	7
C			0	4
D				0

$D'$

We will show soon how to build the original additive matrix  $D$  and the tree  $T$  from  $D'$

But, in the above transformations, we needed both  $T$  and  $T'$ . The question is : without knowing  $T$  and  $T'$ , can we derive the ultrametric matrix  $D'(i,j)$  directly? Yes. This is how:

Consider two leaves  $i$  and  $j$  of  $T$ , and let  $w$  be their least common ancestor. We can deduce their equal distance to  $w$  without the knowledge of  $T'$ .

With reference to Figure , we have:

$$D(v,i) = x+y,$$

$$D(v,j) = x+z,$$

$$D(i,j) = y+z$$

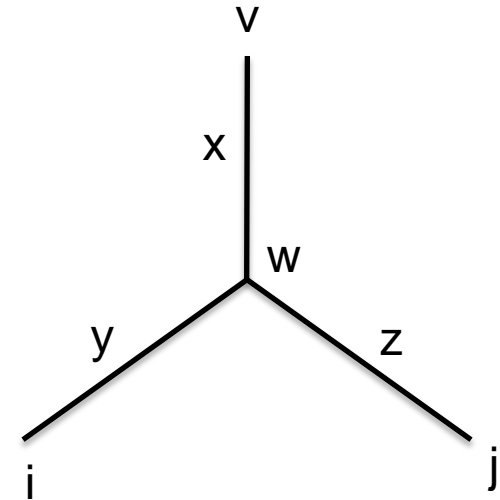
which gives

$$2x = (x+y) + (x+z) - (y+z) = D(v,i) + D(v,j) - D(i,j)$$

$$\text{or } x = [D(v,i) + D(v,j) - D(i,j)] / 2$$

$$y = D(v,i) - x$$

$$z = D(v,j) - x$$



Now, the distance from  $w$  to  $i$  is

$$y + m_v - D(v,i) = (D(v,i) - x) + m_v - D(v,i) \quad (\text{substituting } y = D(v,i) - x \text{ for } y)$$

$$= m_v - x$$

$$= m_v - [D(v,i) + D(v,j) - D(i,j)] / 2$$

$$= m_v + [D(i,j) - D(v,i) - D(v,j)] / 2$$

Similarly, the distance from  $w$  to  $j$  is  $m_v + [D(i,j) - D(v,i) - D(v,j)] / 2$

Thus, we have

**Lemma:** *Without knowing  $T$  and  $T'$ , we can deduce*  
$$D'(i,j) = m_v + [D(i,j) - D(v,i) - D(v,j)]/2$$

And we can state the theorem

**Theorem:** *If  $D$  is an additive matrix, then  $D'$  is an ultrametric matrix where*

$$D'(i,j) = m_v + [D(i,j) - D(v,i) - D(v,j)]/2$$

The converse of the theorem is

**Theorem:** *If Matrix  $D'$  is ultrametric, then matrix  $D$  is additive.*

**Proof:** Let  $T^*$  be the ultrametric tree for  $D'$  (we cannot take  $T'$  which is yet unknown). First, assign weights to edges of  $T^*$  such that the path from any leaf node  $i$  to its parent node  $w$  has a distance equal to the label of the node  $w$ . To do this, assign to each edge  $(p, q)$  the absolute difference between the two numbers written at nodes  $p$  and  $q$ . As a result, the path-distance between any pair of leaf nodes  $(i, j)$  is exactly twice the number appearing at their common ancestor. Now, since  $T^*$  is ultrametric tree for  $D'$ , this distance must be :  $2xD'(i, j) = 2m_v + D(i, j) - D(v, i) - D(v, j)$

Now, do the reverse arithmetic on the edge weights, that is, “shrink” leaf edge going to the leaf node  $i$  by  $m_v - D(v, i)$ . This makes the tree distance between leaf nodes  $i$  and  $j$  to be exactly  $D(i, j)$ . This final step created the additive tree for  $D$  from the ultrametric tree  $D'$ .



**Additive Tree Algorithm :**

1. Create matrix  $D'$  from  $D$  and construct the ultrametric tree  $T^*$  from  $D'$ .
2. Next, assign an edge label to each edge equal to the absolute difference between the node labels of its endpoints.
3. Then for each leaf  $i$ , subtract  $m_v - D(v,i)$  from the distance on the edge into leaf  $i$ .

The resulting tree is the additive tree for the matrix  $D$ .

**D**

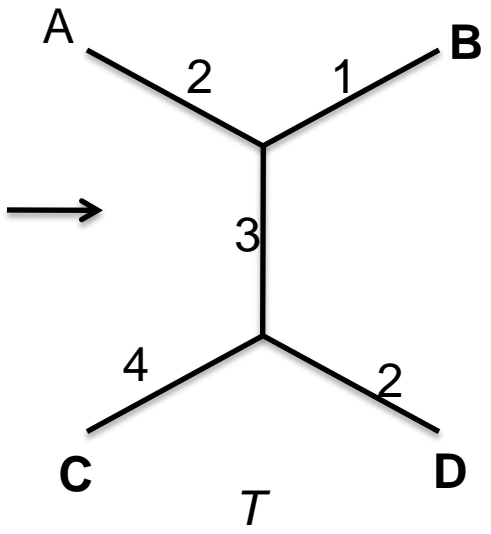
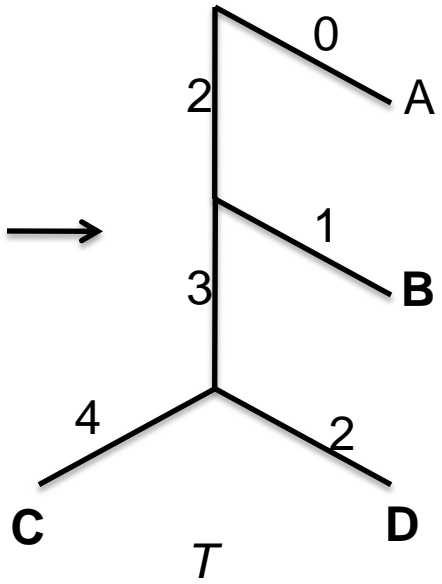
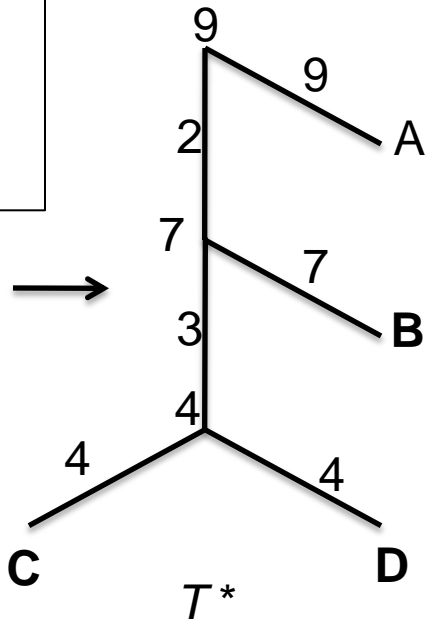
	A	B	C	D
A	0	3	9	7
B		0	8	6
C			0	6
D				0



	B	C	D
A	9	9	9
B		7	7
C			4

**D'**

**Theorem:** The additive tree algorithm takes  $O(n^2)$  time.



# Character Based Trees

# Why Character Based Tree?

For evolutionary tree construction, we typically start with an  $n \times m$  alignment matrix ( $n$  species,  $m$  nucleotides in each). This can be transformed to a distance matrix and then we can use the distance based tree construction algorithms as discussed earlier. But, some information gets lost during this transformation because after we obtain the distance based tree, reverse transformation of the distance matrix to an alignment matrix is impossible. So, we need better techniques to handle character or attributes based tree construction algorithms.

## Character-Based Tree Reconstruction

- **Character-based reconstruction** algorithms use the  $n \times m$  alignment matrix directly instead of using distance matrix.  
( $n = \#$  of species,  $m = \#$  of characters)
- **GOAL:** determine what character strings at internal nodes would best explain the character strings for the  $n$  observed species

# Character-Based Tree Reconstruction (cont'd)

- Characters may be nucleotides, where A, G, C, T are **states** of this character. Alternately, the characters may be the attributes of the species such as the # of eyes or legs or the shape of a beak or a fin.
- By setting the length of an edge in the tree to be the Hamming distance between the end vertices of the edge, we may define the **parsimony score** of the tree as the sum of the lengths (weights) of the edges.

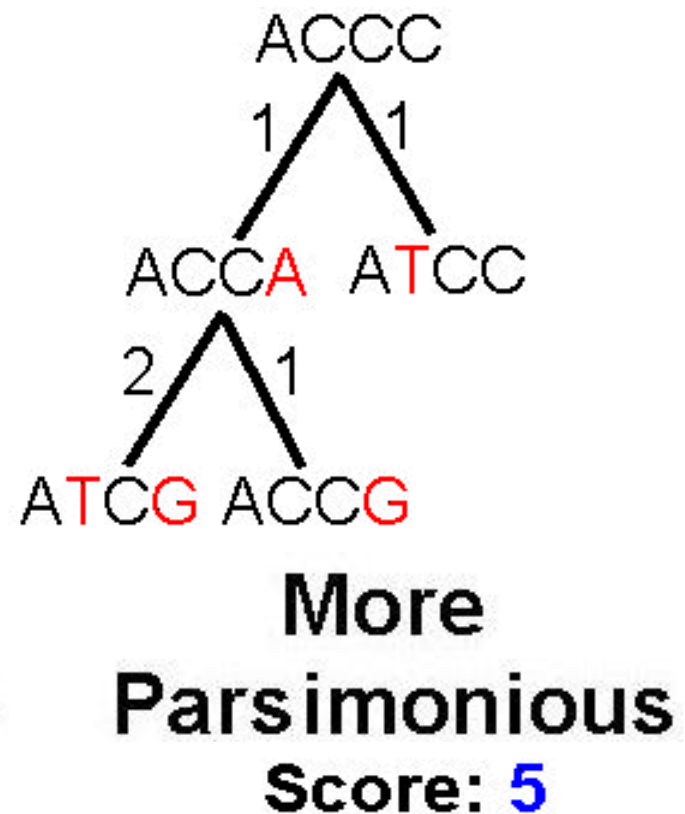
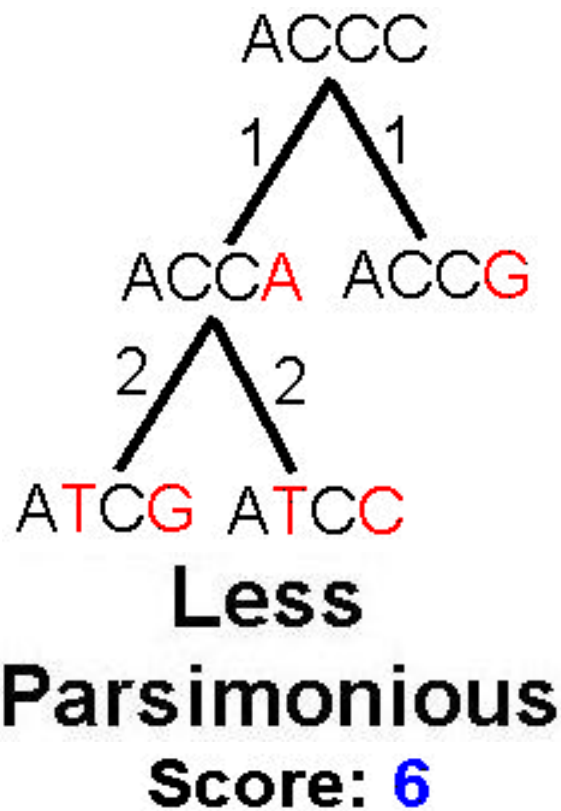
# Parsimony Approach to Evolutionary Tree Reconstruction

- Applies Occam's razor principle (“**keep it simple, stupid**”) to identify the simplest explanation for the data.
- Assumes observed character differences resulted from the fewest possible mutations.
- Seeks the tree that yields lowest possible **parsimony score** - sum of cost of all mutations found in the tree

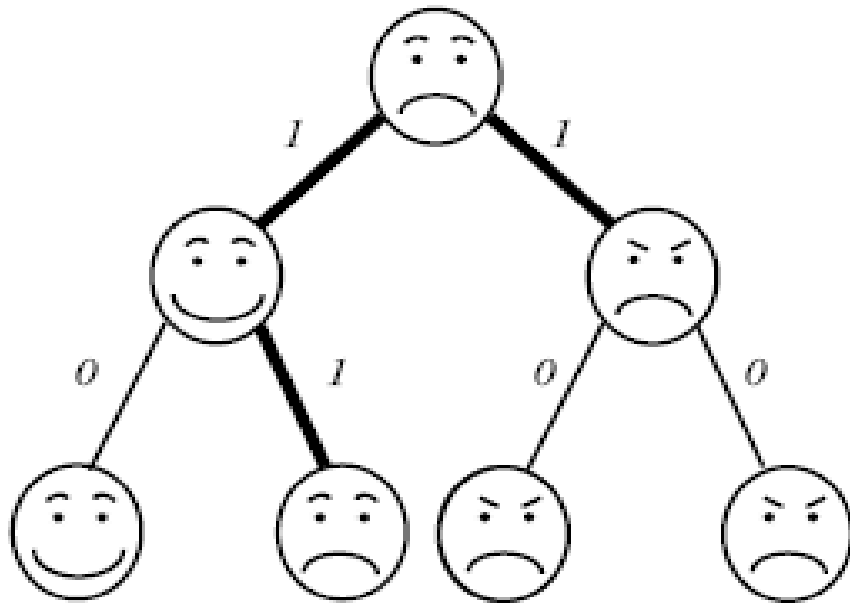
# Parsimony and Tree Reconstruction

Alignment matrix:  
n=3,m=4

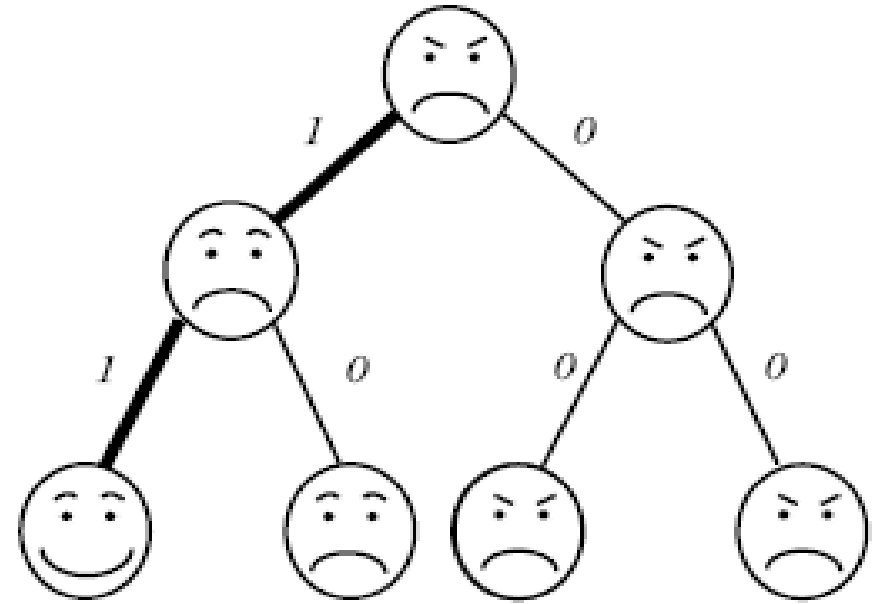
A	T	C	G
A	T	C	C
A	C	C	G



# Character-Based Tree Reconstruction (cont'd)



(a) *Parsimony Score=3*



(b) *Parsimony Score=2*

**Figure 10.16** If we label a tree's leaves with characters (in this case, eyebrows and mouth, each with two states), and choose labels for each internal vertex, we implicitly create a *parsimony* score for the tree. By changing the labels in (a) we are able to create a tree with a better parsimony score in (b).



# Score Definition

Given a tree with every vertex labeled by an  $m$ -long string of characters, *length* of an edge  $(v, w)$  is the Hamming distance  $d_H(v, w)$  between the strings  $v$  and  $w$ . The parsimony score of the tree is

$$\sum_{\text{all edges in the tree}} d_H(v, w)$$

The strings at the internal nodes are unknown. The problem is to find these strings that minimizes the parsimony score. For *Small Parsimony* problem the **tree is given** but the **internal node labels are unknown**. For *Large Parsimony* problem **both the tree and the strings** at the internal nodes are **unknown**.

# Small Parsimony Problem

- Input: Tree  $T$  with each leaves labeled by an  $m$ -character strings.
- Output: Labeling of internal vertices of the tree  $T$  minimizing the parsimony score.
- We can assume that every leaf is labeled by a single character, because the characters in the string are assumed to be independent. Thus the small parsimony problem can be solved independently for each character. The final tree will be the union of these trees.
- We will start by solving first **the weighted parsimony problem** as defined in the next slide.

# Weighted Small Parsimony Problem

- A more general version of Small Parsimony Problem
- Input includes a  $k \times k$  scoring matrix describing the cost of transformation of each of  $k$  states into another one.
- For “Un-weighted” Small Parsimony problem, the scoring matrix has 0 entry at all diagonal positions and has 1 in all other positions. This corresponds to the Hamming distance between the character strings associated with the nodes  $v$  and  $w$  of an edge. For a string of only one character
$$d_H(v, w) = 0 \text{ if the character for } v \text{ and } w \text{ are equal}$$
$$d_H(v, w) = 1 \text{ otherwise.}$$

# Scoring Matrices

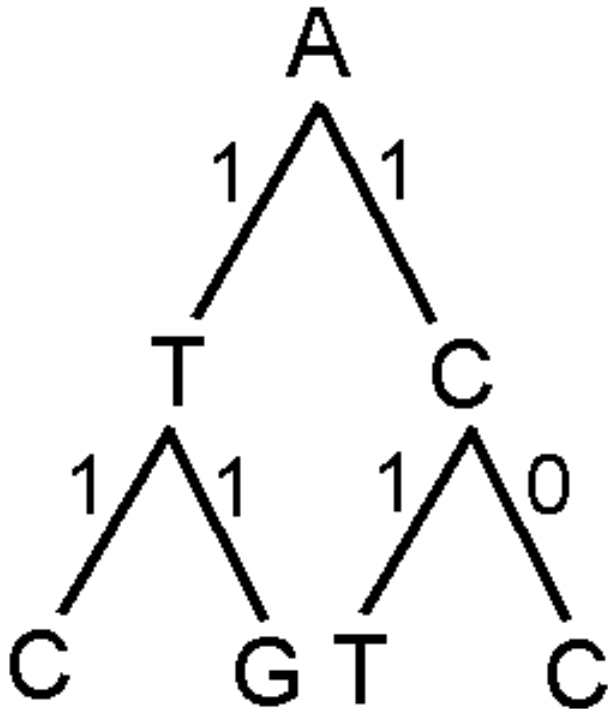
Small Parsimony Problem

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

Weighted Parsimony Problem

	A	T	G	C
A	0	3	4	9
T	3	0	2	4
G	4	2	0	4
C	9	4	4	0

# Unweighted vs. Weighted

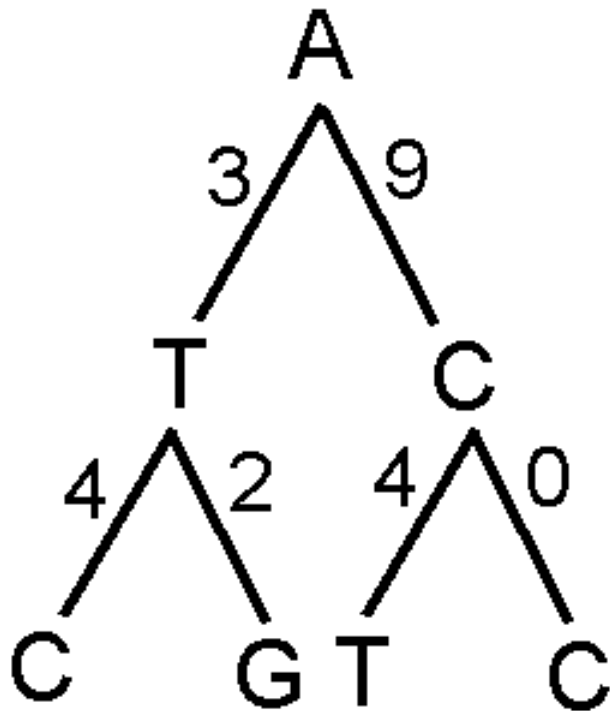


Small Parsimony Scoring Matrix:

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

Small Parsimony Score: 5

# Unweighted vs. Weighted



Weighted Parsimony Scoring Matrix:

	A	T	G	C
A	0	3	4	9
T	3	0	2	4
G	4	2	0	4
C	9	4	4	0

Weighted Parsimony Score: 22

# Weighted Small Parsimony Problem: Formulation

- Input: Tree  $T$  with each leaf labeled by elements of a  $k$ -letter alphabet and a  $k \times k$  scoring matrix  $(\delta_{ij})$
- Output: Labeling of internal vertices of the tree  $T$  minimizing the weighted parsimony score

# Sankoff Algorithm: Dynamic Programming (1975)

- The basic idea is to calculate and keep track of a score for every possible label at each vertex. Let  $s_t(v)$  = minimum parsimony score of the **subtree** rooted at vertex  $v$  if  $v$  has character  $t$
- The score at each vertex is based on scores of its children. If each parent node has two children then:

$$s_t(\mathbf{parent}) = \min_i \{s_i(\mathbf{left child}) + \delta_{i,t}\} + \min_j \{s_j(\mathbf{right child}) + \delta_{j,t}\}$$

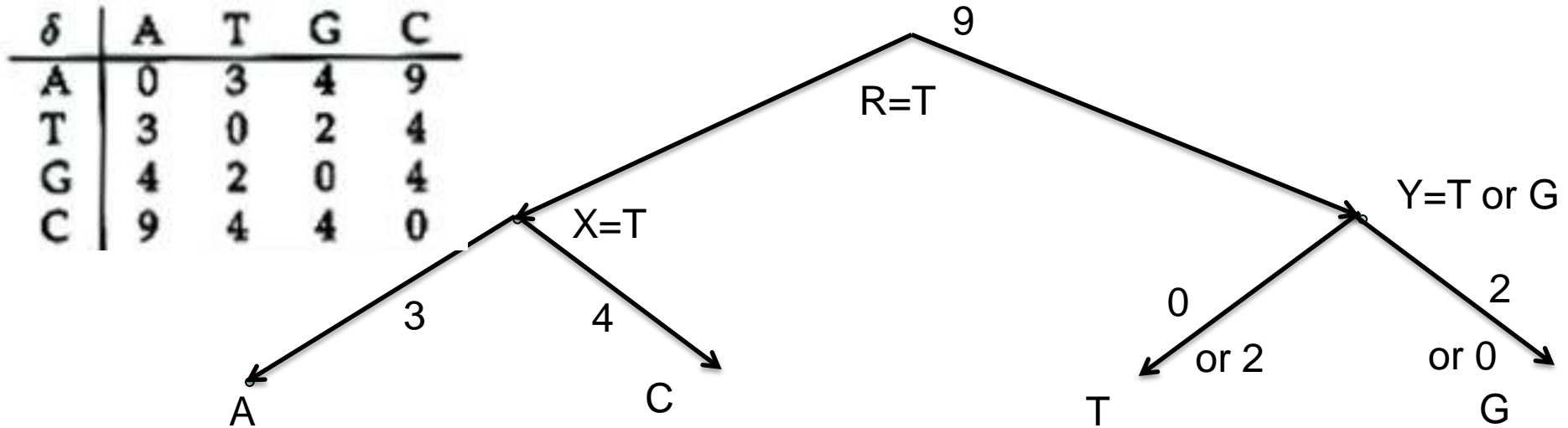


In other words, for an internal vertex  $v$  with children  $u$  and  $w$ , the score  $s_t(v)$  can be computed by first computing  $k$  scores  $s_i(u)$ , and  $k$  scores  $s_j(w)$  for  $1 \leq i, j \leq k$ , and adding the minimum scores for the children as (below  $i, j, t$  are characters):

$$s_t(v) = \min_i \{s_i(u) + \hat{\partial}_{i,t}\} + \min_j \{s_j(w) + \hat{\partial}_{j,t}\}$$

The initial conditions assign the scores  $s_t(v)$  at the leaves as:  $s_t(v)=0$  if  $v$  is labeled by the character  $t$  and otherwise  $s_t(v)$  is infinity ( a very large number  $L$ ). The minimum weighted parsimony score is given by the smallest score at the root.

# An Example to illustrate the basic approach



Suppose, the leaves have been assigned initial values A,C,T,G in that order. If we assign the internal node X the values A,C,T,G, the parsimony scores will be

$$\delta(A,A) + \delta(A,C) = 0+9=9, \quad \delta(C,A) + \delta(C,C) = 9+0=9,$$

$$\delta(T,A) + \delta(T,C) = 3+4=7, \quad \delta(G,A) + \delta(G,C) = 4+4=8.$$

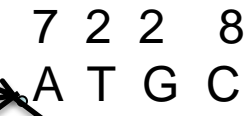
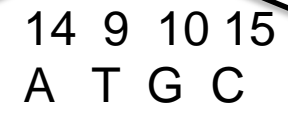
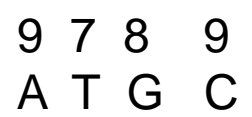
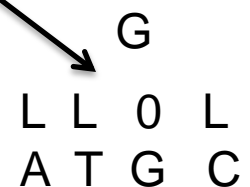
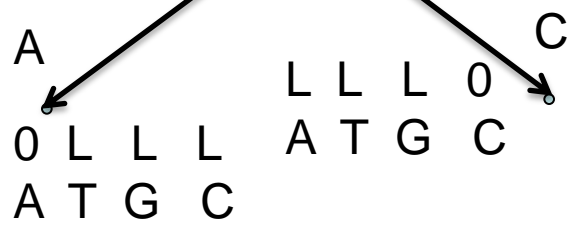
Thus minimum score 7 is achieved if  $X=T$ . By a similar analysis, we can conclude that Y should be assigned either T or G giving minimum parsimony score  $\delta(T,T) + \delta(T,G) = 0+2=2$  or  $\delta(G,T) + \delta(G,G) = 2+0=2$ , respectively. It is obvious that the root Node R must be assigned T to get minimum parsimony score 9.

# Sankoff Algorithm: Two Phase

- In the first phase, the scores at the root vertex are computed by going up the tree starting from the leaf nodes.
- The initial conditions assign the scores  $s_t(v)$  at the leaves according to the rule  $s_t(v)=0$  if  $v$  is labeled by letter  $t$  and  $s_t(v)$  is infinity (in practice a very large number  $L$ ) otherwise.
- In the second phase, after the scores at root vertex have been computed the Sankoff algorithm moves down the tree and assigns each vertex with optimal character.
- Note, in general, it is not possible to assign character while travelling up the tree because, as we noted, in our example the same minimum score may happen from different choices, We need to carry upwards all the possible scores for all possible values in an internal node. The next few slides illustrate the algorithm.
- The algorithm is repeated for each of the  $m$  columns of the alignment matrix.

L=A very large number  
 The leaves are labeled  
 by A, C, T, G in  
 that order

$\delta$	A	T	G	C
A	0	3	4	9
T	3	0	2	4
G	4	2	0	4
C	9	4	4	0

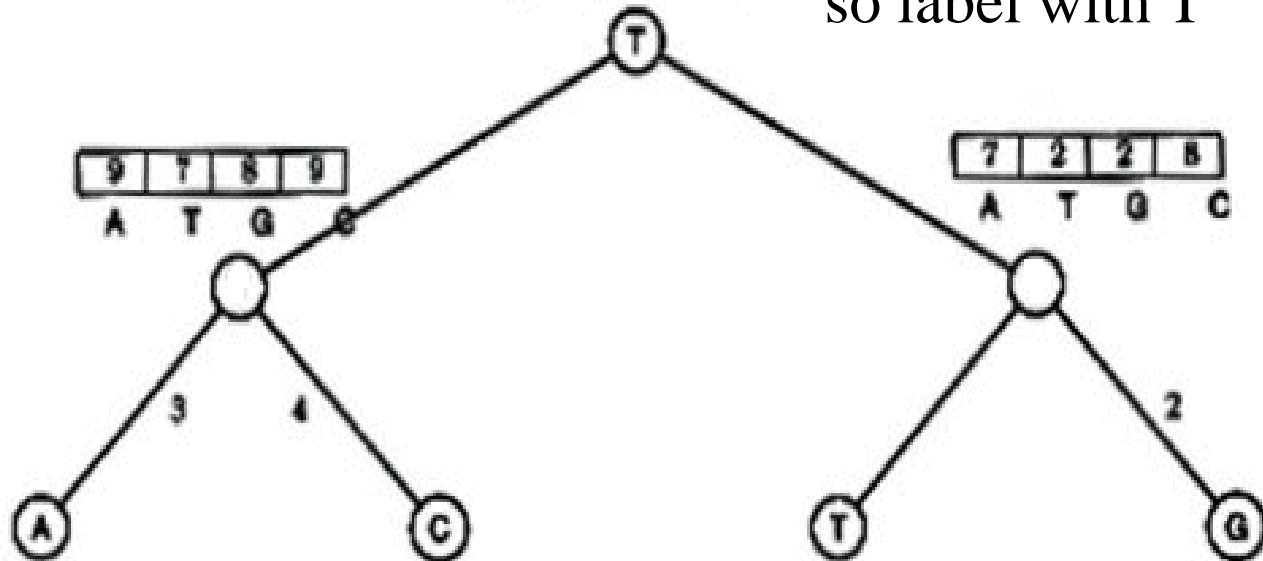


# Sankoff Algorithm (cont.)

Smallest score at root is minimum weighted parsimony score

14	9	10	15
A	T	G	C

In this case, 9 –  
so label with T



0	∞	∞	∞
A	T	G	C

∞	∞	∞	0
A	T	G	C

∞	0	∞	∞
A	T	G	C

∞	∞	0	∞
A	T	G	C

# Sankoff Algorithm (Trace Back step for the dynamic algorithm)

Minimum parsimony score 9 is derived from score  $7+2=9$

The running time for the algorithm is  $O(nk)$

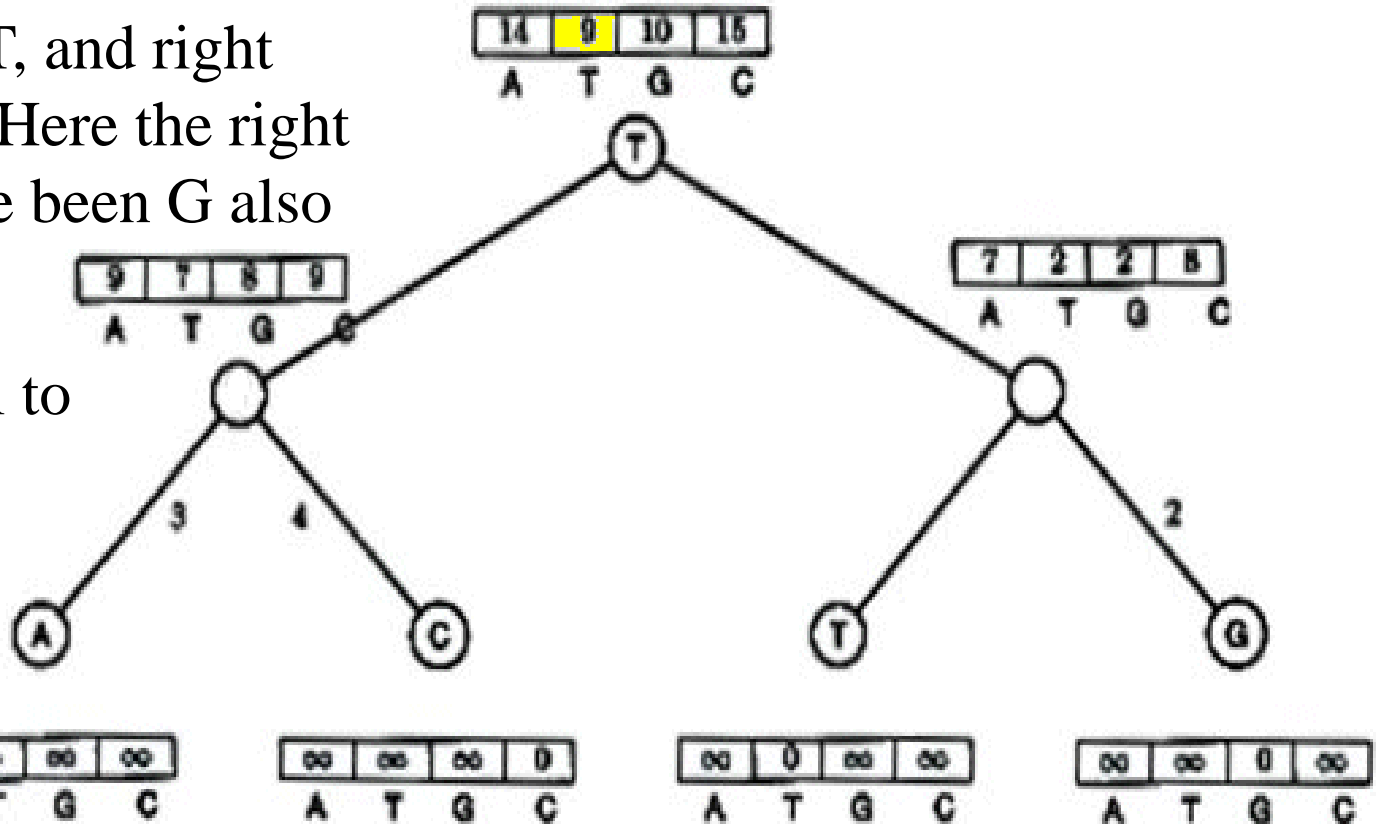
So left child is T, and right child is also T. Here the right child could have been G also but making the

root node equal to

T would have

Increased the

score by 2



# Fitch's Algorithm (1971)

- Solves Small Parsimony problem for unweighted tree.
- Dynamic programming in essence
- Assigns a set of letter to every vertex in the tree.
- If the two children's sets of character overlap, it's the common set of them
- If not, it's the combined set of them.

# Fitch Algorithm

- 1) Assign a **set of possible letters** to every vertex, traversing the tree from leaves to root in *postorder* (*Left, Right, Self*)
- Each node's set is the combination of its children's sets (leaves contain their label). Assume that the internal vertex  $v$  has two child nodes  $u$  (left) and  $w$  (right). The set of letters assigned to vertices  $v$ ,  $u$  and  $w$  are denoted as  $S_v$ ,  $S_u$ ,  $S_w$ , respectively. Then,

$$\begin{aligned} S_v &= S_u \cap S_w \quad \text{if } S_u \text{ and } S_w \text{ overlap} \\ &= S_u \cup S_w \quad \text{otherwise} \end{aligned}$$

E.g. if the node  $v$  has a left child labeled  $S_u = \{A, C\}$  and a right child labeled  $S_w = \{A, T\}$ , the node will be given the set  $S_v = \{A\}$ . If the right child had only  $S_w = \{T\}$ , then  $S_v = \{A, C, T\}$ .

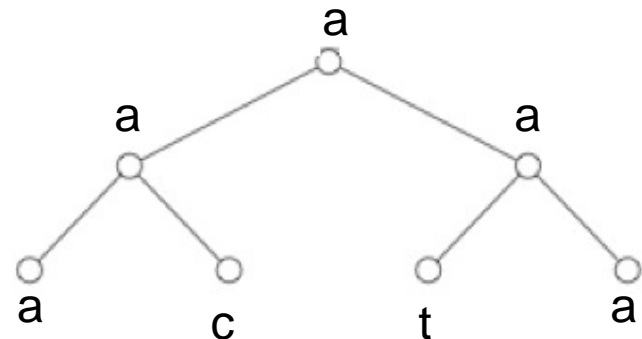
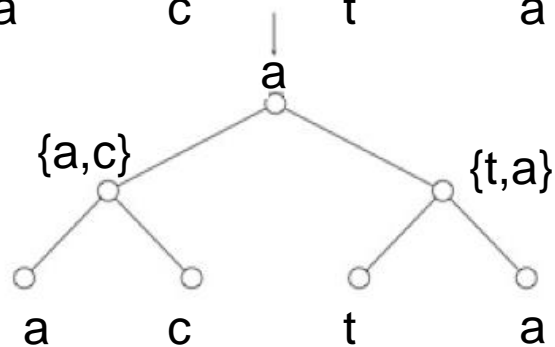
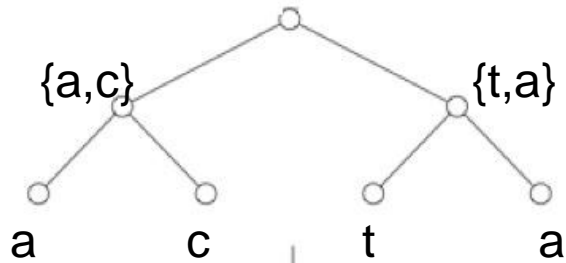
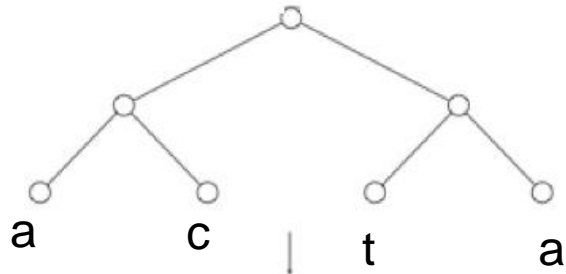


# Fitch Algorithm (cont.)

- 2) Traverse the tree from root to leaves in preorder (Self, Left, Right)
- Assign root arbitrarily from its set of letters
  - For all other vertices, if its parent's label is in its set of letters, assign it its parent's label
  - Else, choose an arbitrary letter from its set as its label

# Fitch's Algorithm

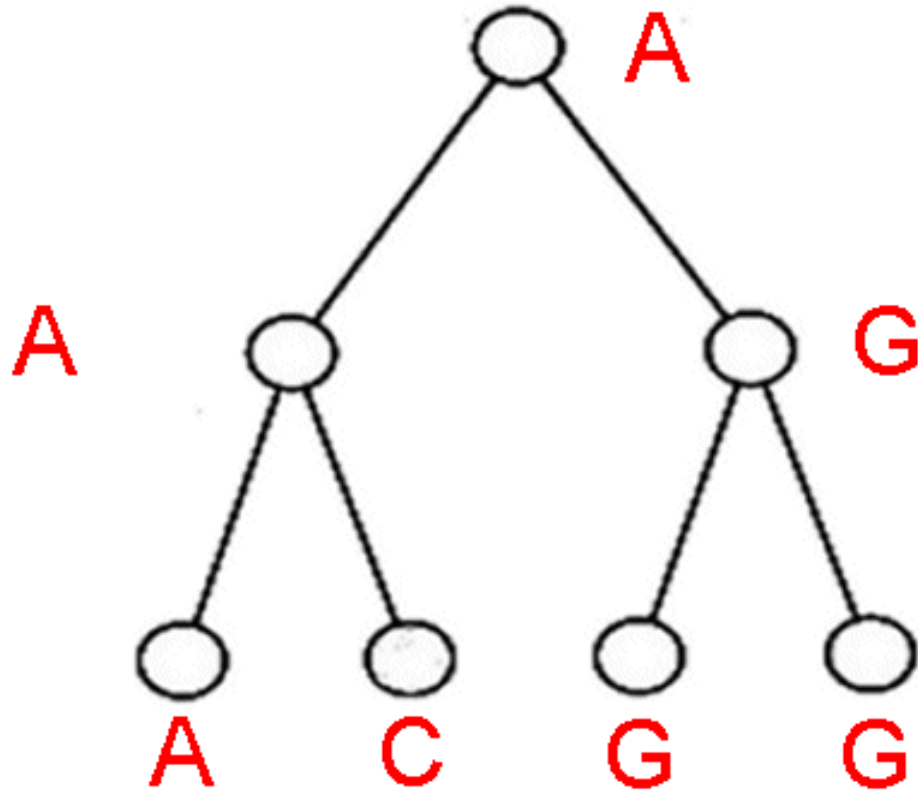
An example:



# Fitch vs. Sankoff

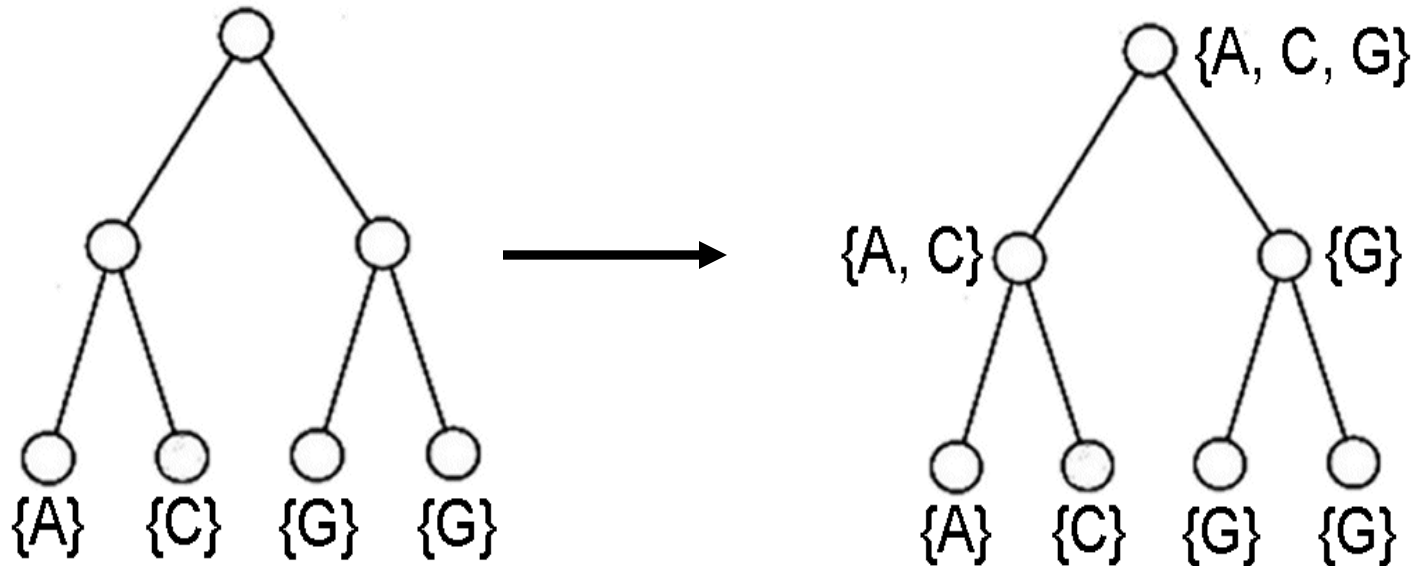
- Both have an  $O(nk)$  runtime
- Are they actually different?
- Let's compare ...

# Fitch Algorithm (Another Example)



# Fitch

As seen previously:



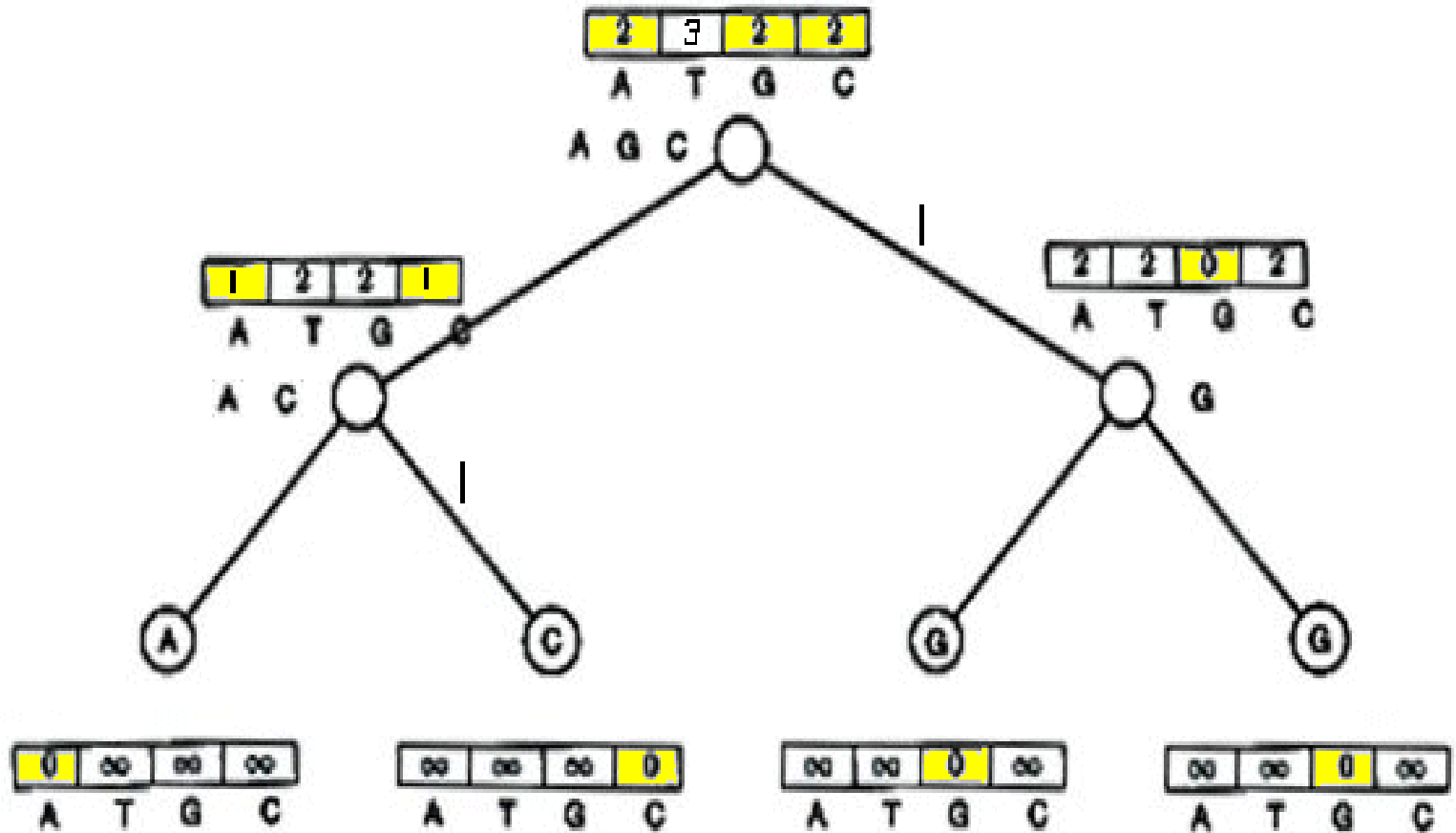
# Comparison of Fitch and Sankoff

- As seen earlier, the scoring matrix for the Fitch algorithm is merely:

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

- So let's do the same problem using Sankoff algorithm and this scoring matrix

# Sankoff



# Sankoff vs. Fitch

- The Sankoff algorithm gives the **same** set of **optimal** labels as the Fitch algorithm
- For Sankoff algorithm, character  $t$  is *optimal* for vertex  $v$  if  $s_t(v) = \min_{1 \leq i \leq k} s_i(v)$ 
  - Denote the set of optimal letters at vertex  $v$  as  $S(v)$ 
    - If  $S(\text{left child})$  and  $S(\text{right child})$  overlap,  $S(\text{parent})$  is the intersection
    - Else it's the union of  $S(\text{left child})$  and  $S(\text{right child})$
    - This is also the Fitch recurrence
- The two algorithms are **identical**

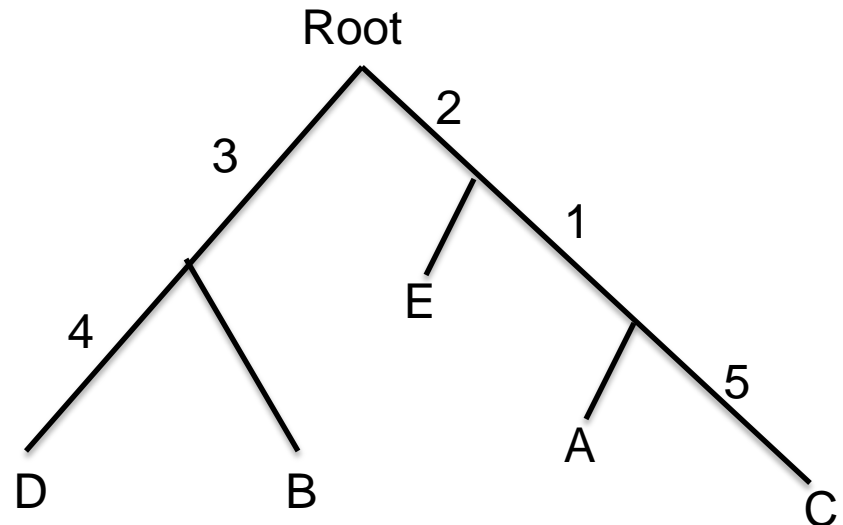


# Phylogenetic Tree –a special case of maximum parsimony problem

- Let  $M$  be a  $n \times m$  0-1 matrix describing  $n$  species or objects, and let  $m$  be the number of characters or attributes. Each character takes two values 0 or 1. The matrix element  $M(p,i)=1$  if *the* object  $p$  has the attribute  $i$ .

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

=M



Given  $M$ , a phylogenetic tree for  $M$  is a rooted tree  $T$  with  $n$  leaves that obey the following:

1. Each of the  $n$  objects labels one leaf of the tree  $T$ .
2. Each of the  $m$  characters or attributes labels exactly one edge of the tree  $T$ .
3. For any object  $p$ , the characters that label the edges along the unique path from the root to the leaf  $p$  specify all the characters of  $p$  whose state is 1.

We assume that the root represents an ancestral object that has none of the present characters corresponding to an object described by  $(0 \ 0 \ 0 \ \dots \ 0)$ .

We also assume that each character on the edge mutates the state 0 to 1 but there is no back mutation from 1 to 0.

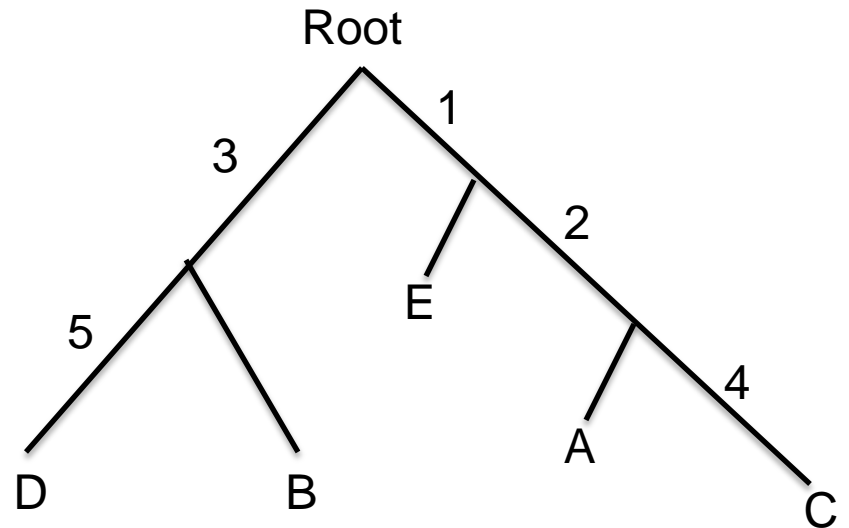
# Perfect Phylogeny Problem

- Given a  $n \times m$  0-1 matrix  $M$ , determine whether there is a phylogenetic tree for  $M$ , and if so, build one.

It will be convenient to reorder the matrix  $M$  to a “sorted” matrix  $M'$  as follows: Consider each column of  $M$  as a binary number with most significant bit in row 1 and sort the  $m$  numbers corresponding to the columns of  $M$  in decreasing order from left to right, placing the largest number in column 1. Obviously, if  $M'$  has a phylogenetic tree, so does  $M$ .

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

=M'

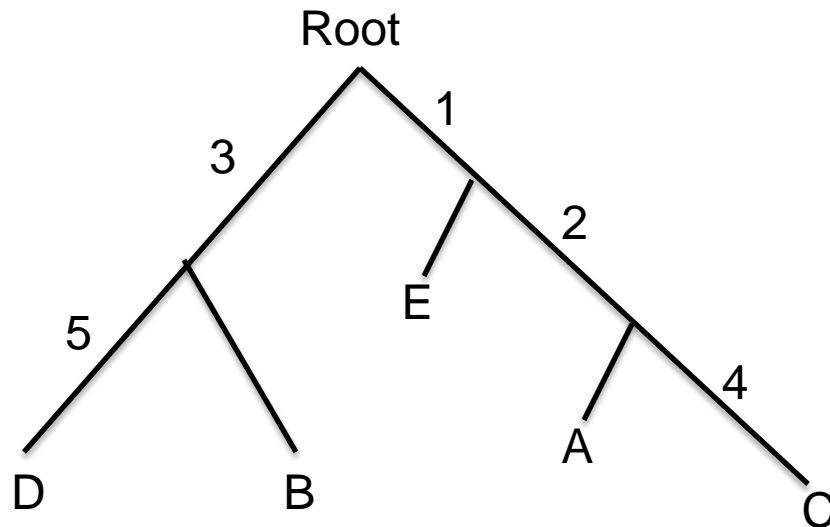


**Def:** For any column  $k$  of  $M'$ , let  $Q_k$  be the set of objects with a 1 in column  $k$ . For our example,  $Q_1=(A,C,E)$ ,  $Q_2=(A,C)$ ,  $Q_3=(B,D)$ ,  $Q_4=(C)$  and  $Q_5=(D)$

**Theorem:** Matrix  $M'$  (or  $M$ ) has a phylogenetic tree if and only if for every pair of columns  $(i, j)$ , either  $Q_i$  and  $Q_j$  are disjoint or one contains the other.

# Phylogenetic Tree Algorithm

1. Sort the numbers corresponding to the columns of  $M$  in decreasing order using  $O(nm)$  radix sort, creating the sorted matrix  $M'$ .
2. For each row  $p$  of  $M'$ , construct the string consisting of the characters in sorted (increasing) order, that  $p$  possesses. (viz.  $A \rightarrow 12$ ,  $B \rightarrow 3$ ,  $C \rightarrow 124$ ,  $D \rightarrow 35$ ,  $E \rightarrow 1$ ).
3. Build the keyword tree  $T$  for the  $n$  strings constructed in step 2.



# An Example: no phylogenetic tree possible

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

$Q_1=(A,C,E)$ ,  $Q_2=(A,C)$ ,  $Q_3=(B,C,E)$ ,  $Q_4=(BD)$  and  $Q_5=(D)$ .

The set  $Q_3=(B,C,E)$  is not disjoint nor contained in any other set. Thus, the matrix does not satisfy the condition of the theorem.

# Large Parsimony Problem

- Input: An  $n \times m$  matrix  $M$  describing  $n$  species, each represented by an  $m$ -character string
- Output: A tree  $T$  with  $n$  leaves labeled by the  $n$  rows of matrix  $M$ , and a labeling of the internal vertices such that the parsimony score is minimized over all possible trees and all possible labelings of internal vertices

# Large Parsimony Problem (cont.)

- Possible search space is huge, especially as  $n$  increases
  - $(2n - 3)!!$  possible rooted trees
  - $(2n - 5)!!$  possible unrooted trees
- Problem is NP-complete
  - Exhaustive search only possible w/ small  $n (< 10)$
- Hence, branch and bound or heuristics used