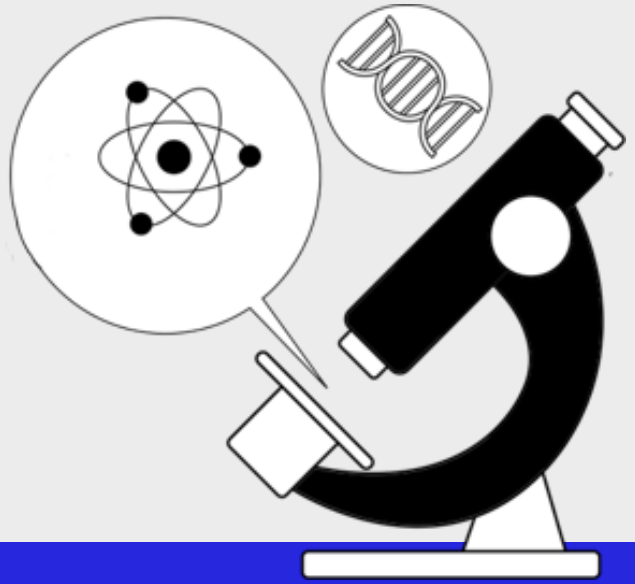
# Molecular Evolution and Reconstruction of Evolutionary Trees

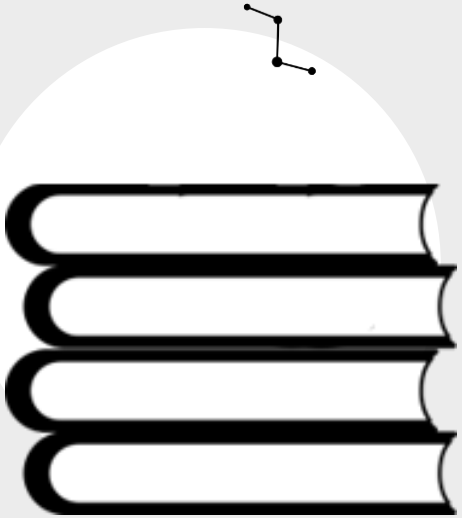
**Saeedeh Akbari**
Department of Computer Engineering
Sharif University of Technology
Fall 2023

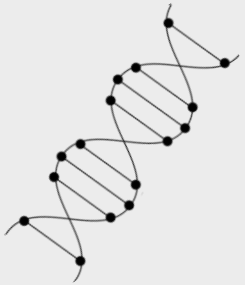


Adapted with modifications from lecture notes prepared by Phillip Compeau
Bioinformatics Algorithms: An Active Learning Approach

# Table of contents

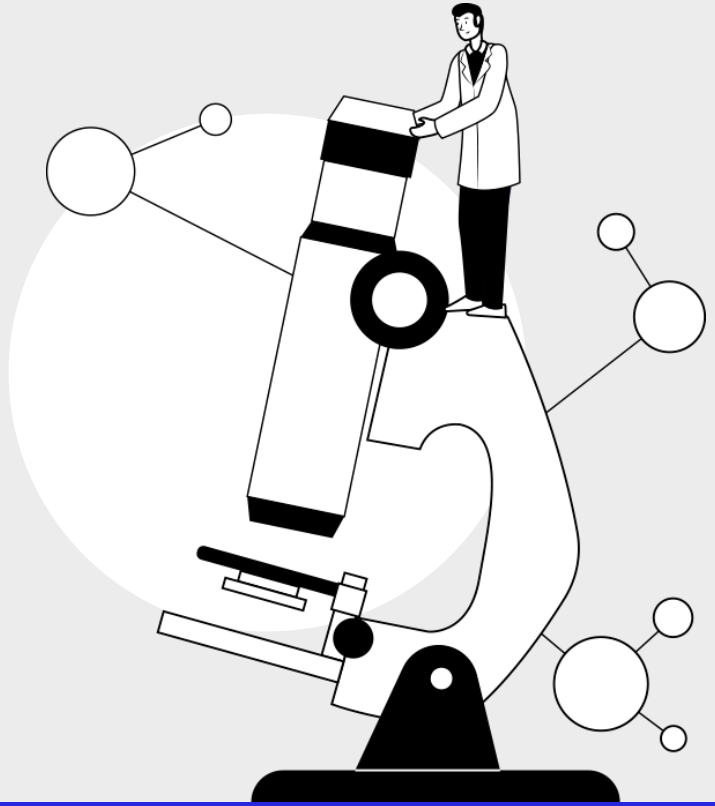






01

# Introduction



# (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.

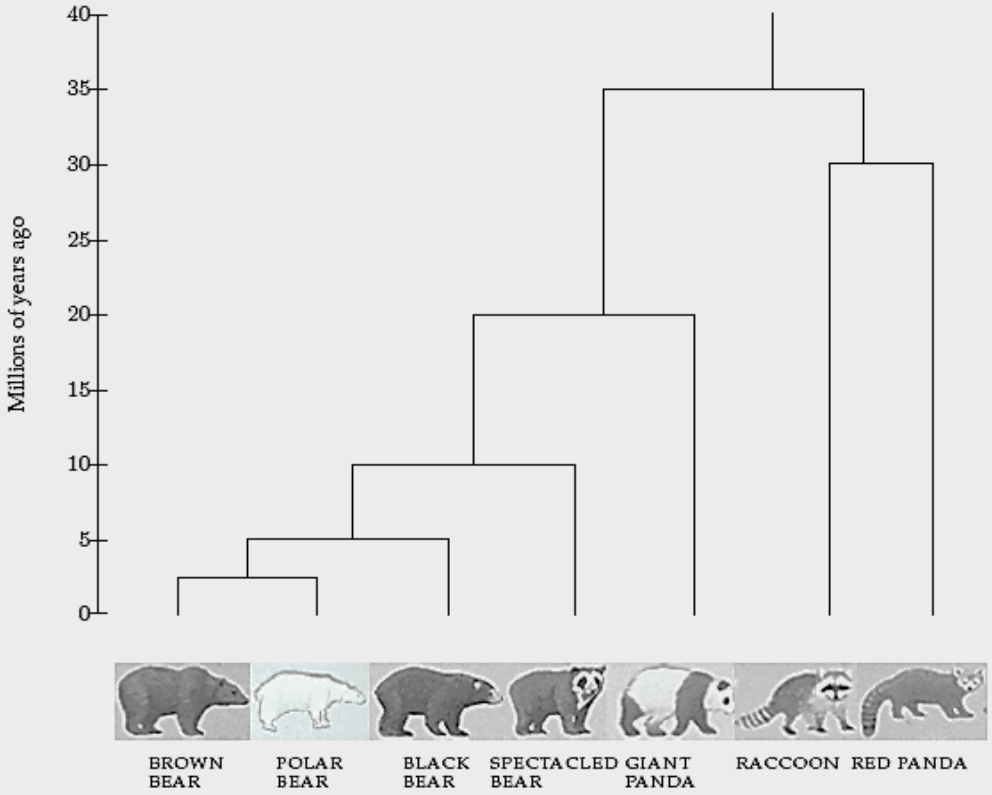
47 million year old primate fossil found 5/17/09

# Evolution and DNA Sequence 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

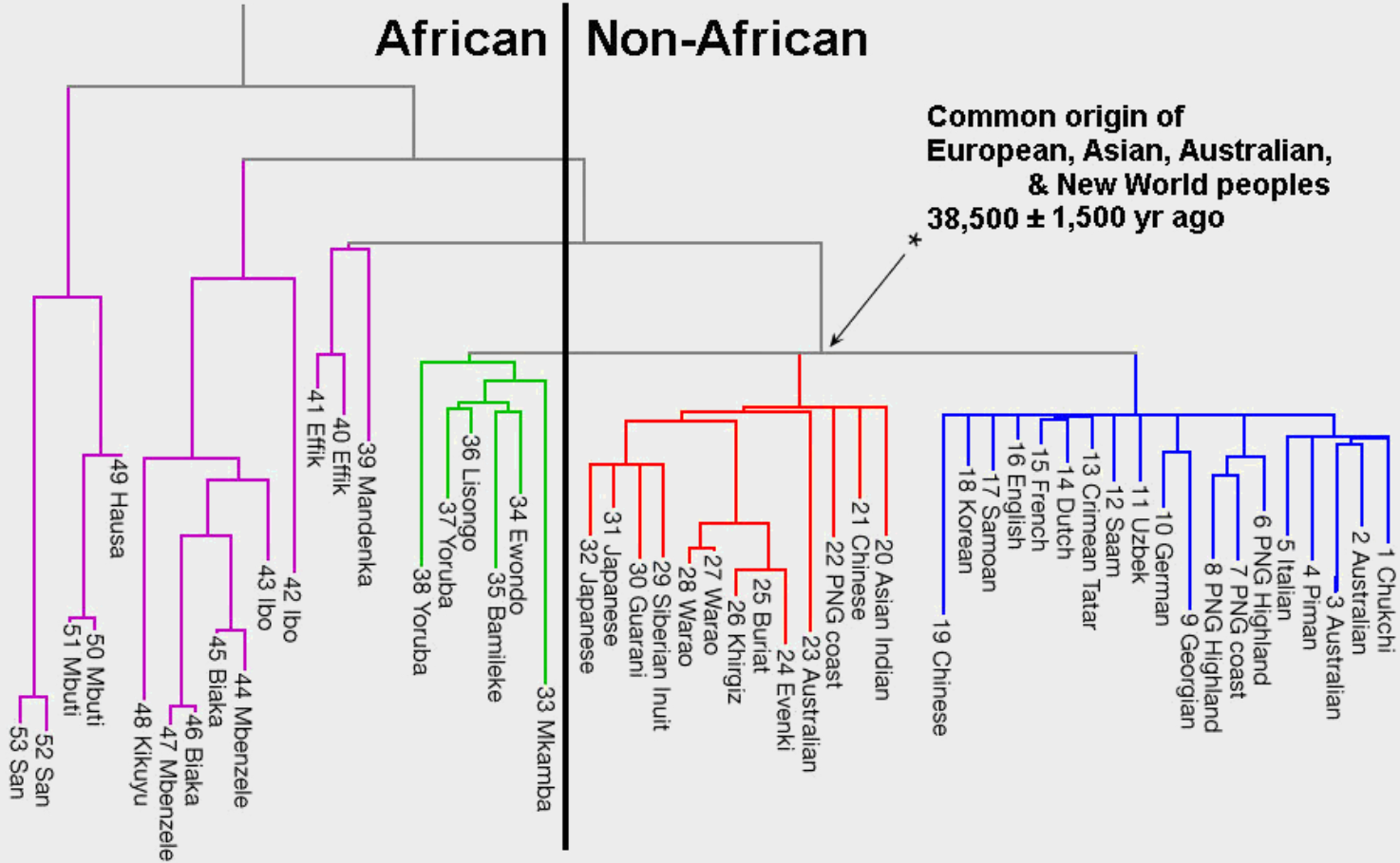


# Evolutionary Trees: DNA-based Approach

- 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.

# 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 150,000 years ago




https://www.mun.ca/biology/scarr/Out_of_Africa.html

# The Origin of Humans: 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 descendence from Neanderthals

**Multiregional:**

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

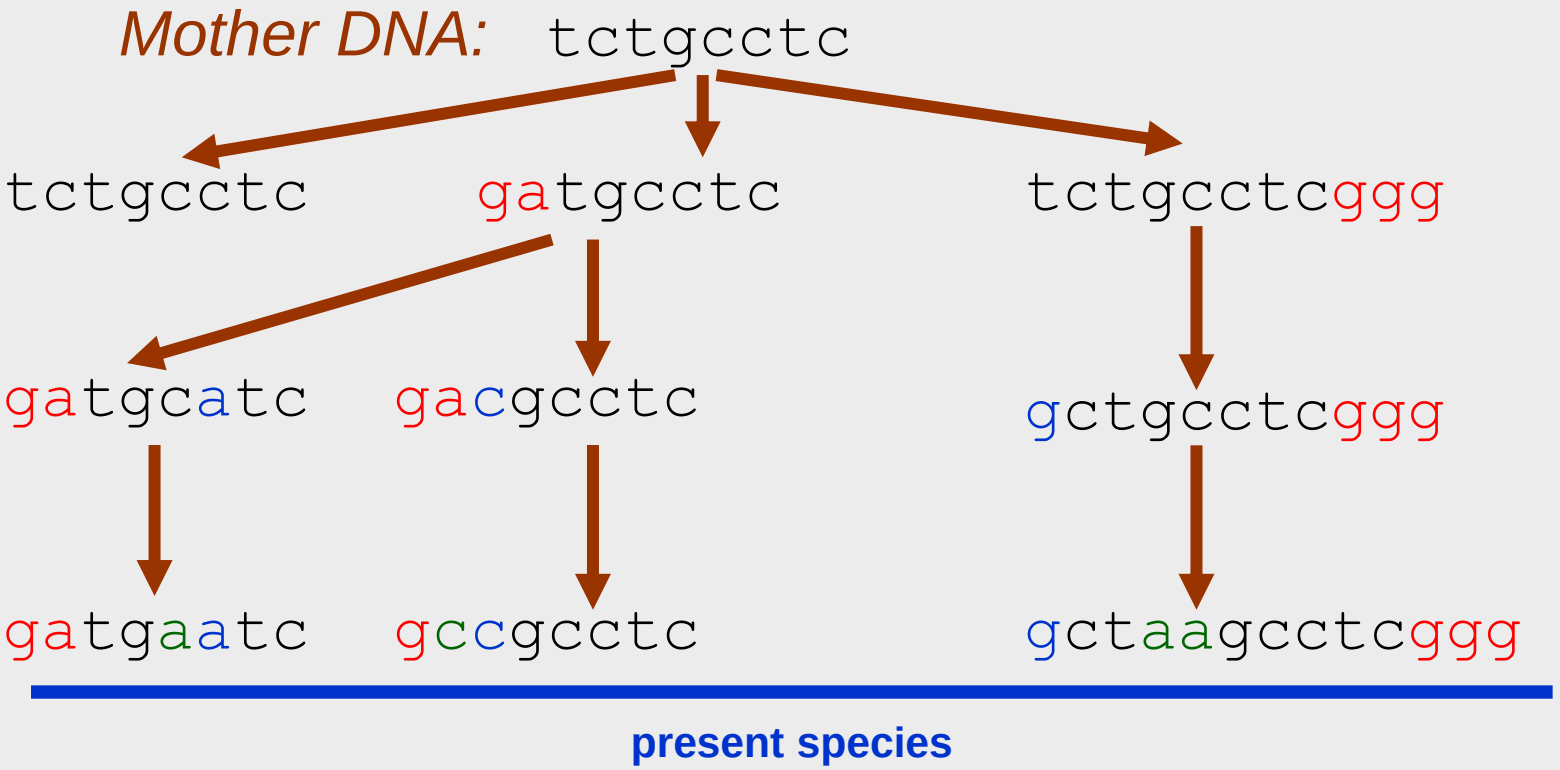
# Evolutionary Trees (or Phylogenetic Trees or Phylogenies)

## How are these trees built from DNA sequences?

In an evolutionary tree,

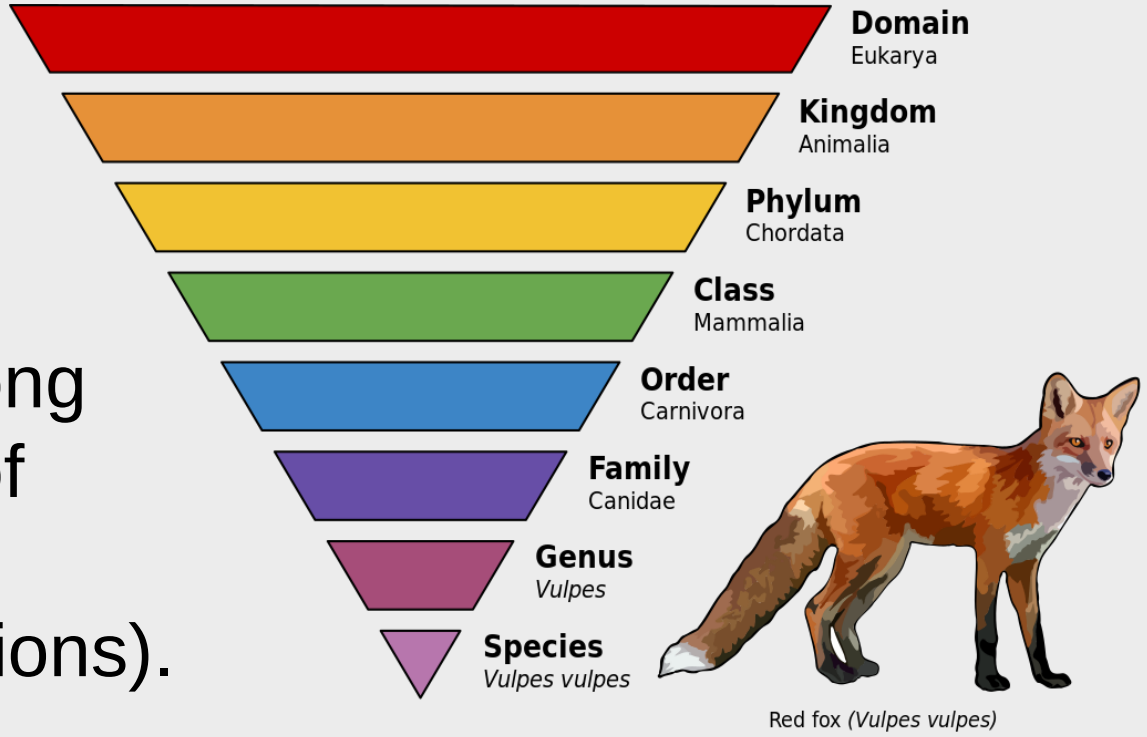
- leaves represent existing **species**
- internal vertices represent ancestors (and **speciation** events)
- root represents the oldest evolutionary ancestor
- edges (and subtrees under them) are sometimes called **lineages**

# The Biological Basis of Evolution



# Phylogenetics

**Phylogenetics** is the study of evolutionary relatedness among various groups of organisms (e.g., species, populations).



# Phylogenetic Trees

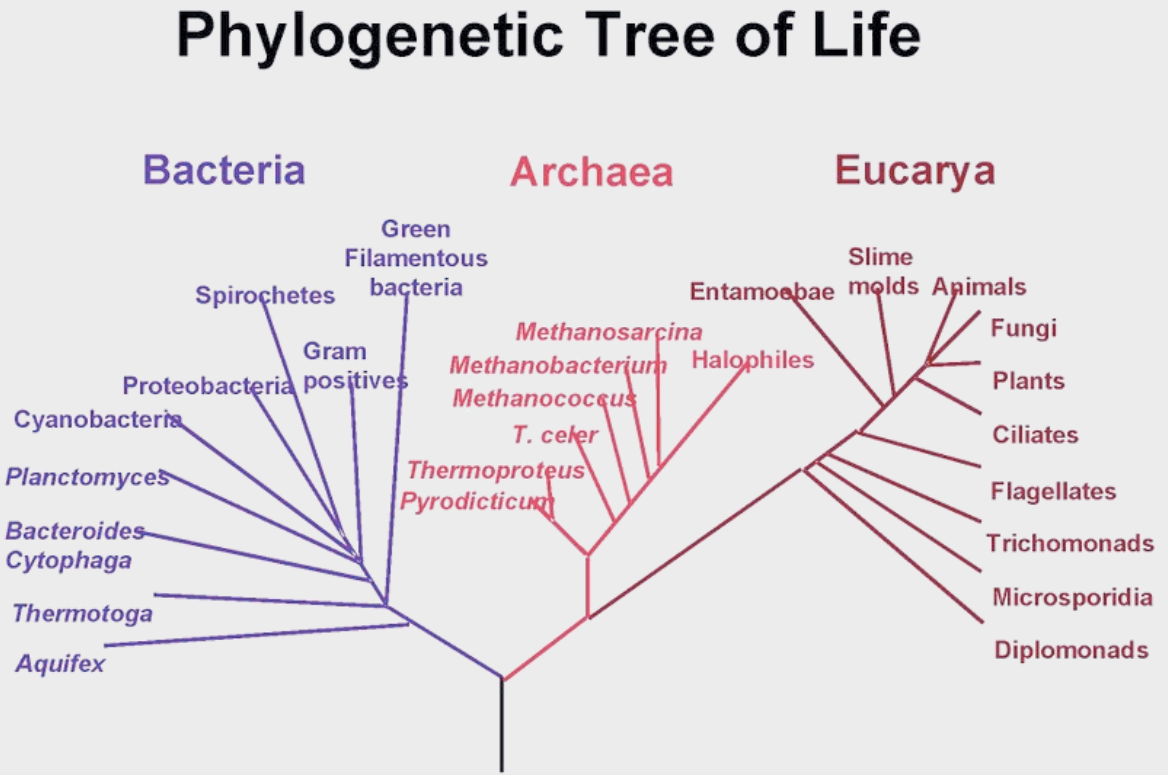
- A tree showing the evolutionary inter-relationships among various species or other entities that are believed to have a common ancestor.
- Each node with descendants represents the most recent common ancestor of the descendants, and edge lengths correspond to time estimates.
- Each node in a phylogenetic tree is called a taxonomic unit. Internal nodes are generally referred to as Hypothetical Taxonomic Units (HTUs) as they cannot be directly observed.

# Phylogenetic trees vs. Cladograms

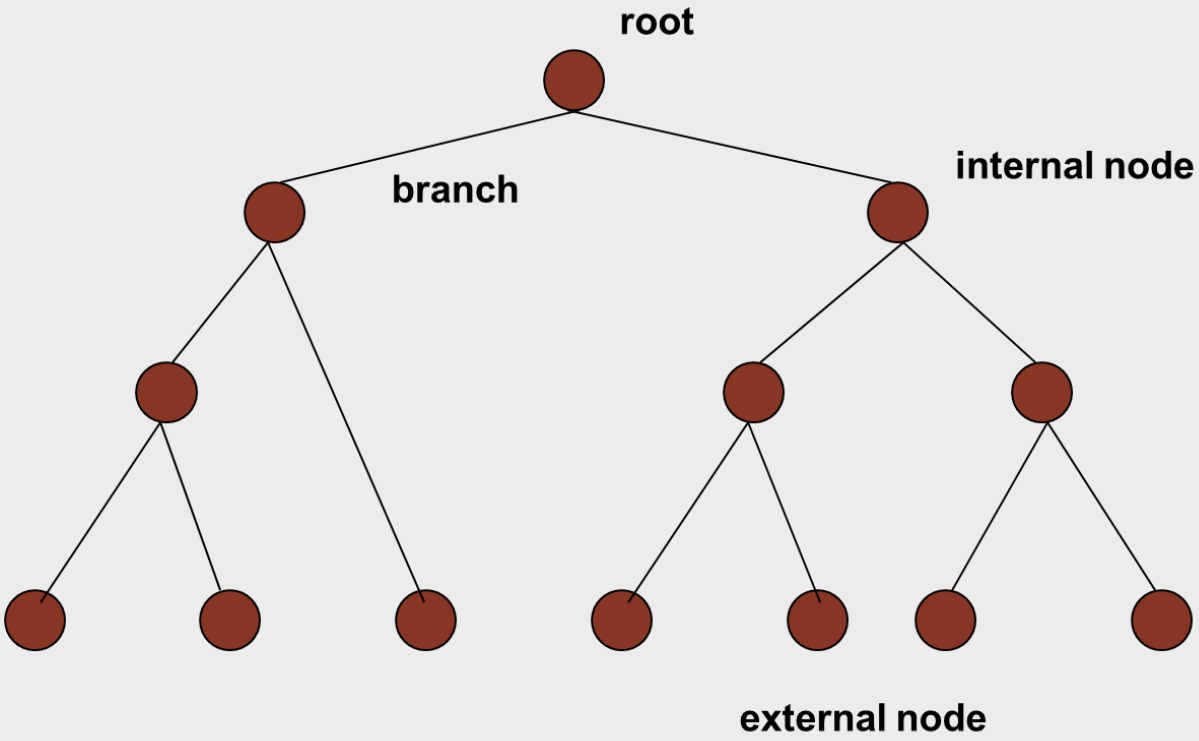
- Both show the relationships between different organisms, but only phylogenetic trees have branches that represent evolutionary time and amount of change.
  - A cladogram is a type of phylogenetic tree that ***only*** shows tree topology—the shape indicating relatedness.
    - It shows that, say, humans are more closely related to chimpanzees than to gorillas, but not the time or genetic distance between the species.
  - A phylogram, on the other hand, has branch distance proportional to evolutionary distance, whether based on genetics or characteristics.

# Rooted Phylogenetic Trees

A **rooted phylogenetic tree** is a directed tree with a unique node corresponding to the (usually imputed) most recent common ancestor of all the entities at the leaves of the tree.



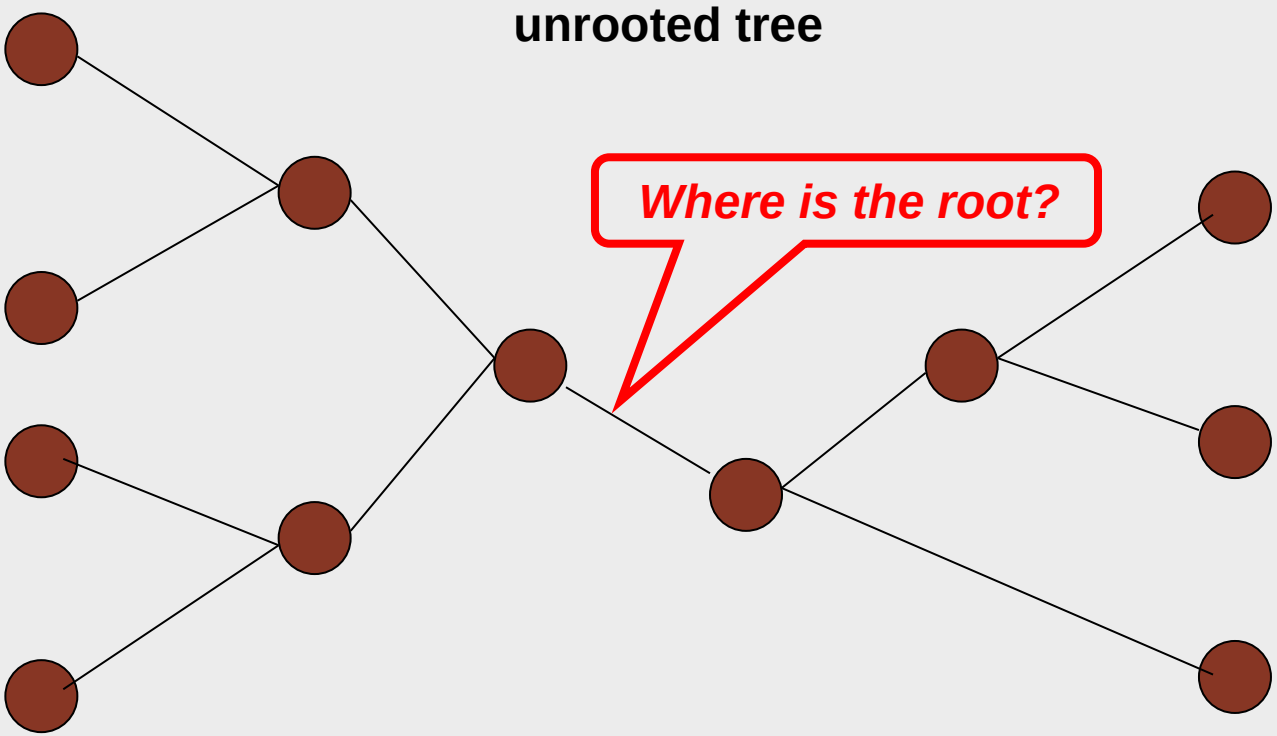
# Rooted Phylogenetic Trees



# Unrooted Phylogenetic Trees

- Unrooted phylogenetic trees can be generated from rooted trees by omitting the root from a rooted tree
- Root cannot be inferred on an unrooted tree without either an outgroup or additional assumptions (for instance, about relative rates of divergence).

# Unrooted Phylogenetic Trees



# Distance and Character

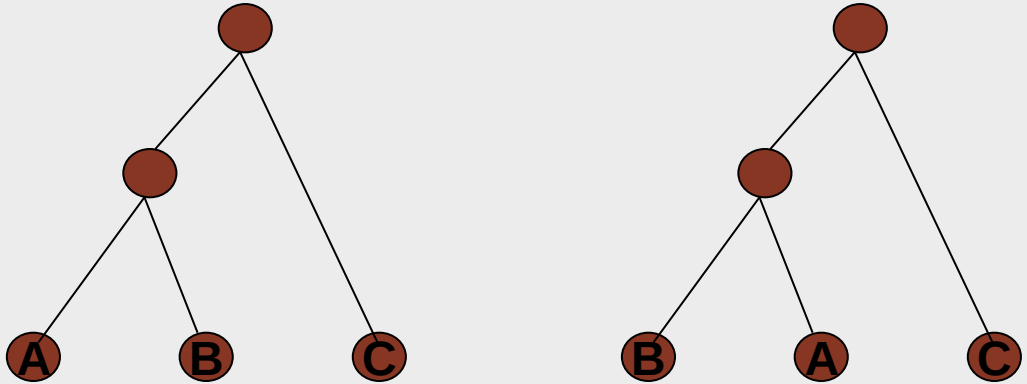
A tree can be based on:

1. **quantitative measures** like the **distance** or **similarity** between species, or

2. based on **qualitative aspects** like **common characters**.

# Trees and Branch Length

- A tree can be a branching tree-graph where branches indicate close phylogenetic relations.
- Alternatively, branches can have length that indicate the phylogenic closeness.
- Any rotation of the internal branches of a tree keeps the phylogenetic relations intact.

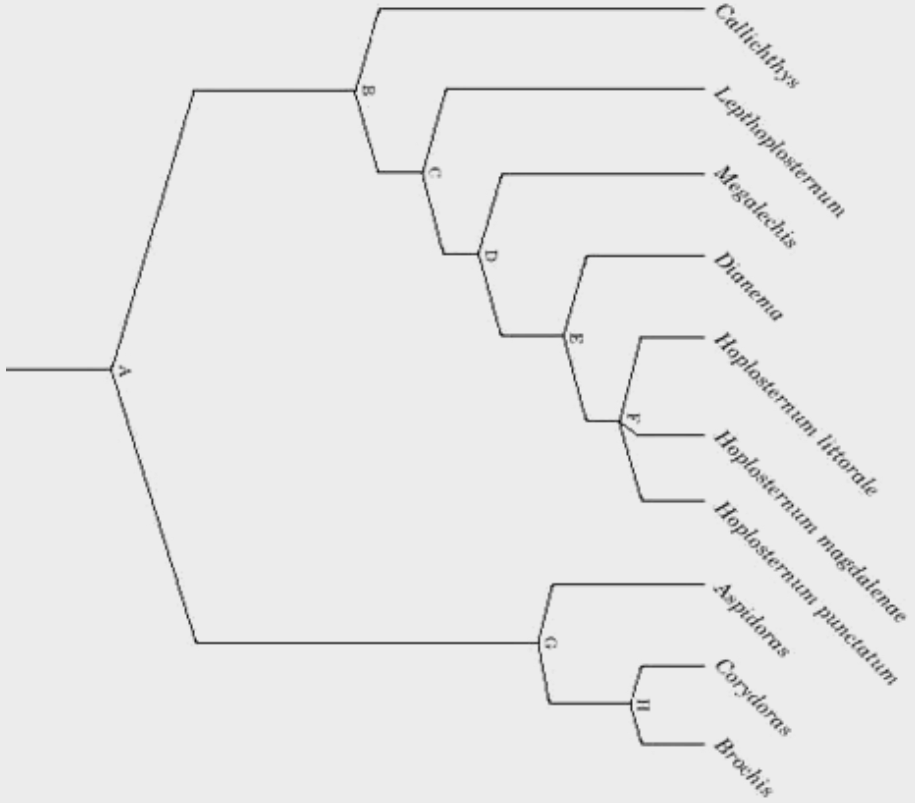


# Constructing Phylogenetic Trees

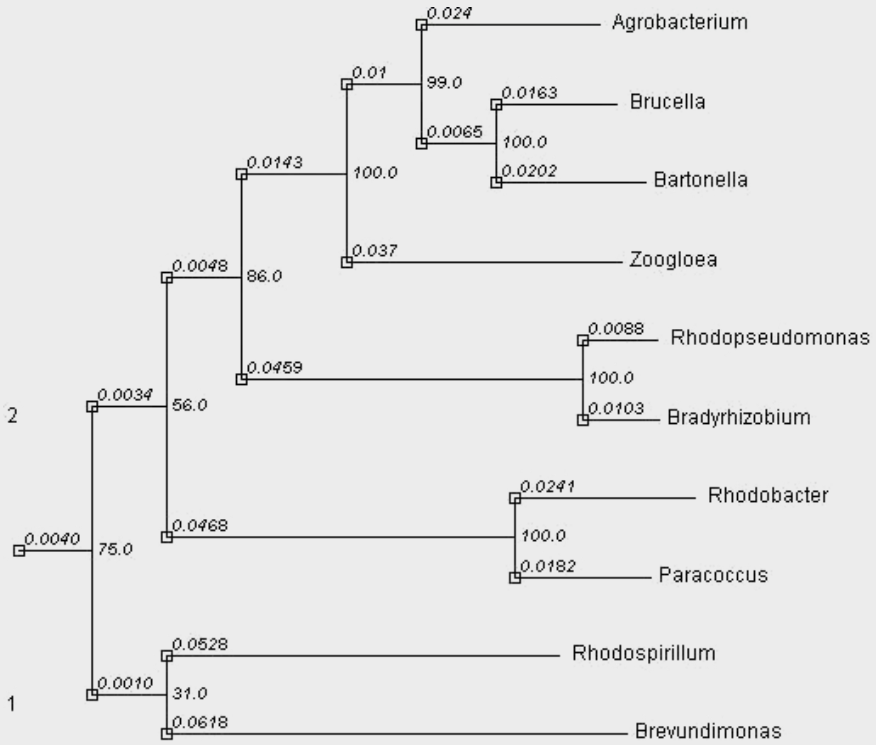
There are three main methods of constructing phylogenetic trees:

1. distance-based methods such as **UPGMA and neighbour-joining,**
2. parsimony-based methods such as **minimize parsimony scores**,
3. character-based methods such as **maximum likelihood or Bayesian inference.**

# Tree without Branch Length



# Tree with Branch Length



# Number of possible trees

- n is number of taxa
- # unrooted trees for n > 2: (2n – 5)!/(2n! – 3(n-3)!)
- # rooted trees for n > 1: (2n – 3)!/(2n ! – 2(n-2)!)
- n  = 5:  #rooted trees = 105
- n = 10 : #rooted trees = 34,459,425

# Representing trees

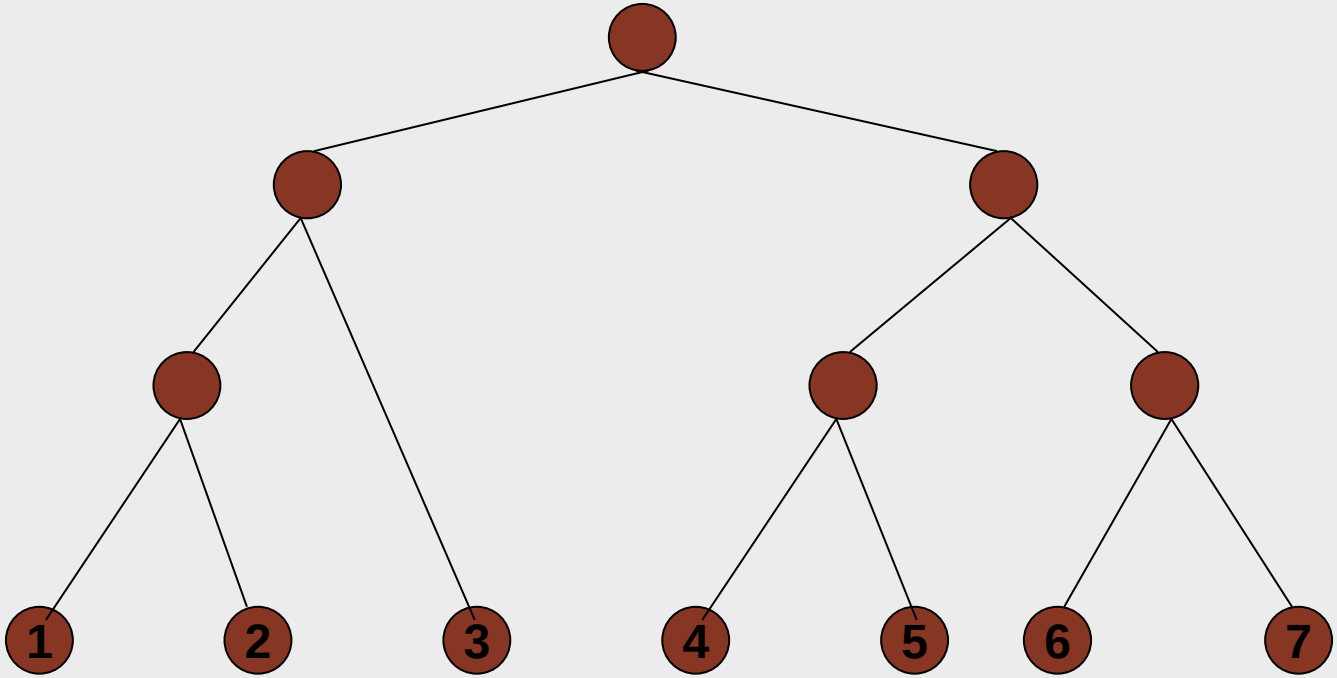
- Various representations:
- *n* taxa = *n* external nodes: (*n* -1) internal nodes

1. internal nodes with children: (*n* – 1)x3 matrix

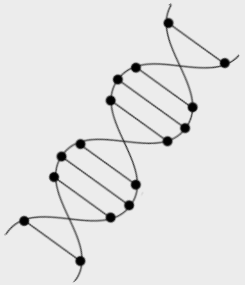
   ( internal node, daughter_1, daughter_2)

2. Newick format: see next slide for example

# Newick format

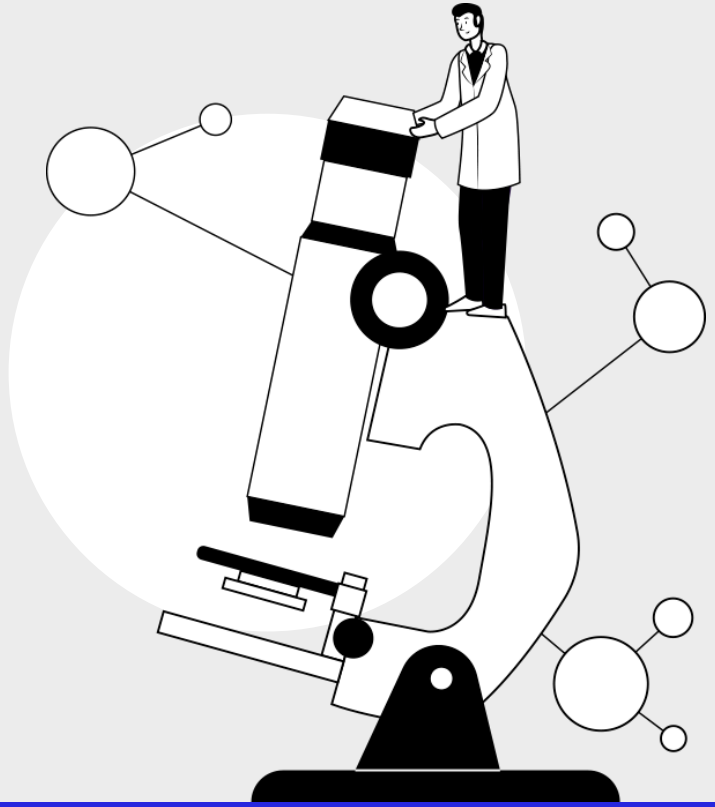



**Newick format: (((1,2),3),((4,5),(6,7)))**



# 02

# Distance Based Phylogeny



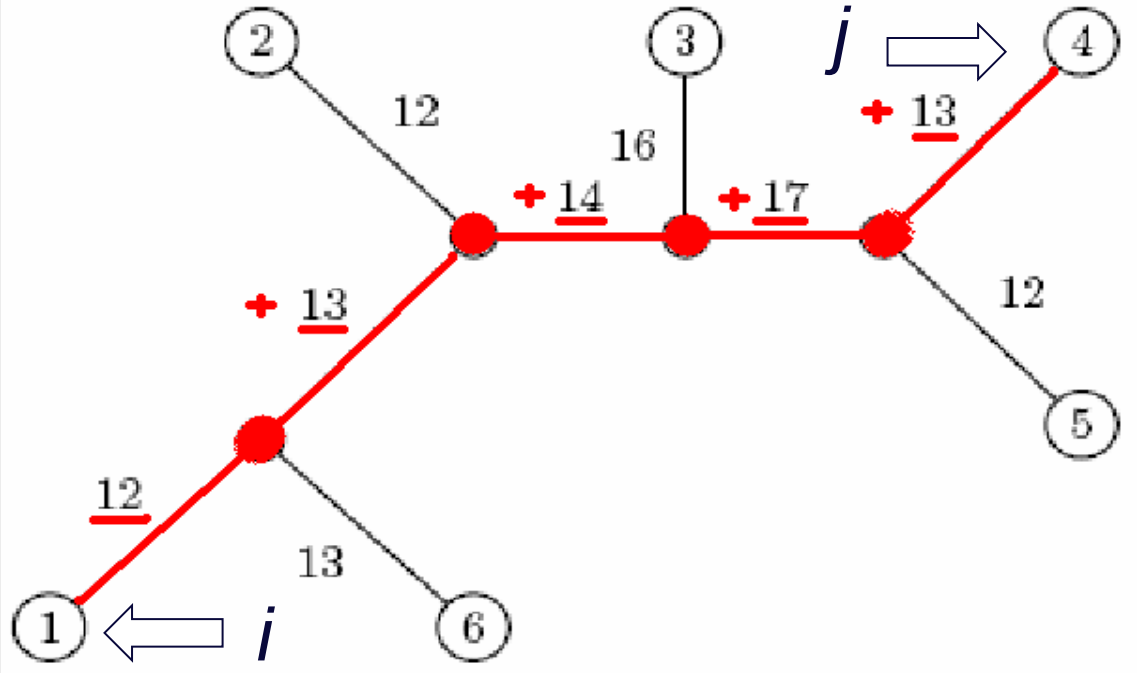
# Distances in Trees

- Edges may have weights reflecting:
  - **Number of mutations** in the evolutionary process from one species to another
  - **Time estimate** for evolution from one species into another
- In a tree T, we often compute
  - $d_{ij}(T)$ - the length of the path between leaves i and j
  - $d_{ij}(T)$ is called the tree distance between i and j

# Distances in Trees

- n taxa $\{t_1,\ldots,t_n\}$
- D matrix of pairwise genetic distances
- Additive distances: distance over path from i → j is: d(i,j)
- (total) length of a tree: sum of all branch lengths.

# Distance in Trees: an Example




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

# Distance Matrix

- Given n species ➔ compute an n x n distance matrix $D_{ij}$

- $D_{ij}$ may be defined as the (weighted) edit distance between the same gene in species i and j, where the gene of interest is sequenced for all n species

$D_{ij}$ – edit distance between i and j

(min number of edit operations to transform one sequence into the other)

**ACTGGA T**
**AGT GACT**

# Edit Distance vs. Tree Distance

- Note the difference between

**$d_{ij}(T)$ – tree distance between i and j**

(the length of the path between leaves i and j )

**$D_{ij}$ – edit distance between i and j**

(min number of edit operations to transform one sequence into the other)

# Fitting Distance Matrix

- Given n species, we can compute the n x n distance matrix $D_{ij}$
- Evolution of these genes is described by a tree that we don't know
- We need an algorithm to construct a tree that best fits the distance matrix $D_{ij}$

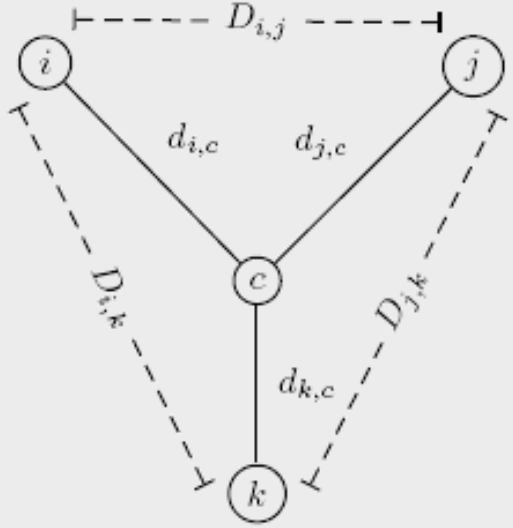
Lengths of path in an (***unknown***) tree *T*

**Fitting means** $D_{ij} = d_{ij}(T)$

Edit distance between species (***known***)

# Reconstructing a 3 Leaved Tree

- Tree reconstruction for any 3x3 matrix is straightforward
- We have 3 leaves i, j, k and a center vertex c



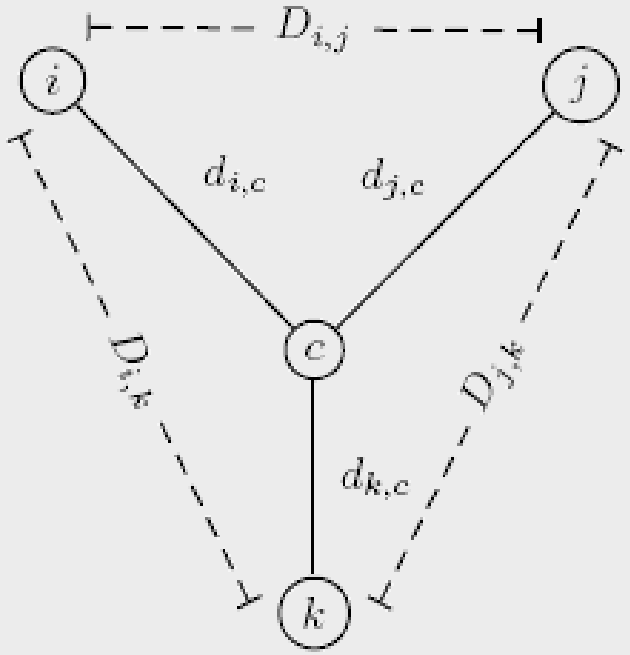

Observe:

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

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

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

# Reconstructing a 3 Leaved Tree (cont'd)




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

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

$$2d_{ic} + d_{jc} + d_{kc} = D_{ij} + D_{ik}$$

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

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

Similarly,

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

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

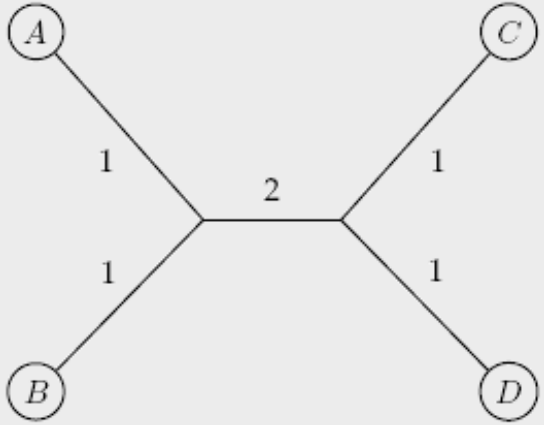
# Trees with > 3 Leaves

- A degree-3 tree with n leaves has 2n-3 edges
  - n-1 internal nodes (including root) ➔ n-2 internal edge + n leave edge
- This means fitting a given tree to a distance matrix D requires solving a system of “n choose 2” equations with 2n-3 variables
- This is not always possible to solve for n > 3

# 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 Phylogenetic Reconstruction

- **Goal:** Reconstruct an evolutionary tree from a distance matrix
- **Input**: n x n distance matrix $D_{ij}$
- **Output:** weighted tree T with n leaves fitting D

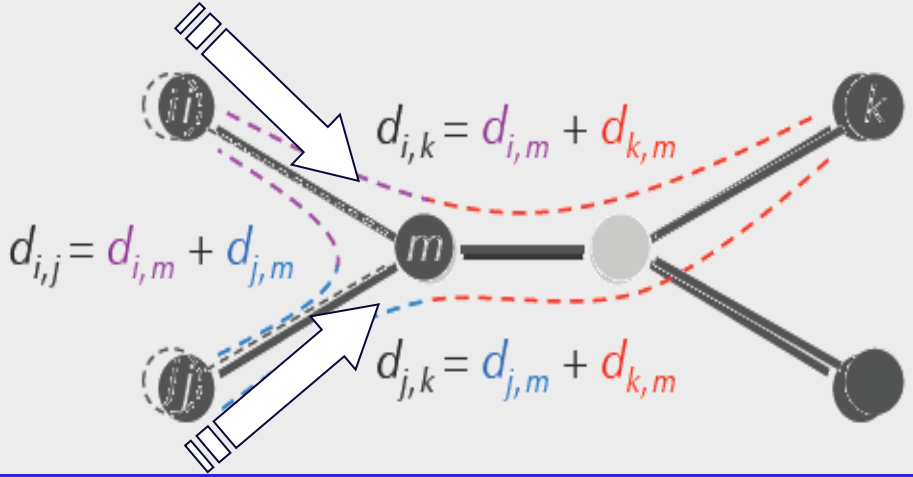
- If D is additive, this problem has a solution and there is an algorithm to solve it

# Using Neighboring Leaves to Construct the Tree

- Find neighboring leaves i and j with parent m
- Remove the rows and columns of i and j
- Add a new row and column corresponding to m, where the distance from m to any other leaf k can be computed as:

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

Compress *i* and *j* into *k*, and iterate algorithm for rest of tree



# Distance Based Phylogenetic Reconstruction

Recursive algorithm :

- find a pair of neighboring leaves i and j by selecting the minimum element $D_{i,j}$ in the distance matrix;
- replace i and j with their parent, and recompute the distances from this parent to all other leaves as described;
- solve the Distance-Based Phylogeny problem for the smaller tree;
- add the previously removed leaves i and j back to the tree.

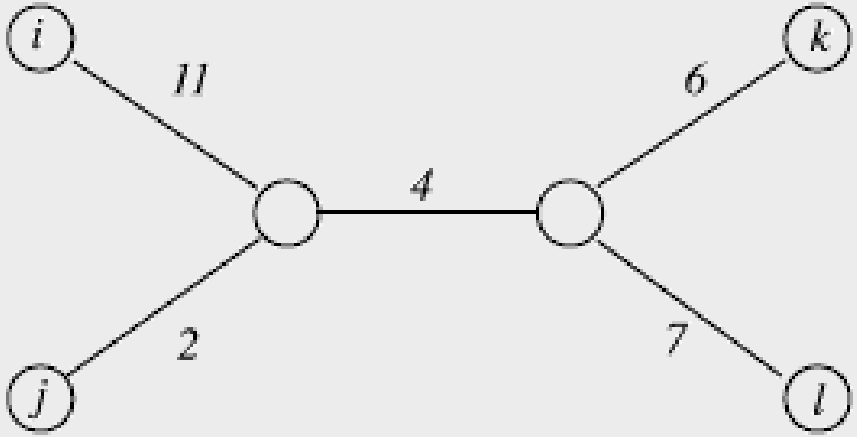
# Finding Neighboring Leaves

To find neighboring leaves we could simply 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!

# Neighbor Joining Algorithm

- In 1987, Naruya Saitou and Masatoshi Nei developed a neighbor joining algorithm (NJ) for phylogenetic tree reconstruction
  - Finds a pair of leaves that are close to each other but far from the other leaves: implicitly finds a pair of neighboring leaves
- Advantage: works well for additive and other non-additive matrices

# Degenerate Triples

- A degenerate triple is a set of three distinct species/taxa $1 \leq i,j,k \leq n$ where $D_{ij} + D_{jk} = D_{ik}$

- Species j in a degenerate triple i,j,k lies on the evolutionary path from species i to species k (or is attached to this path by an edge of length 0)

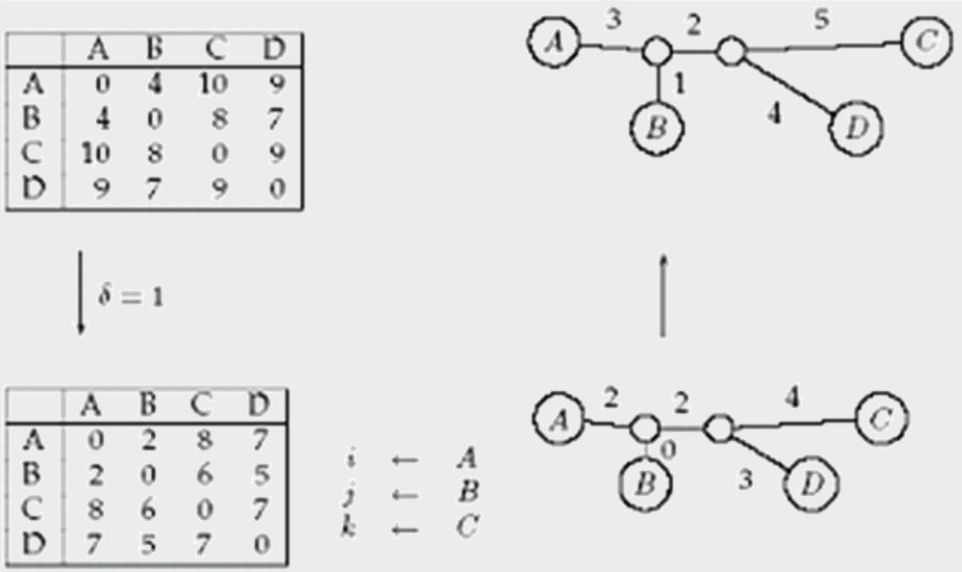
# Looking for Degenerate Triples

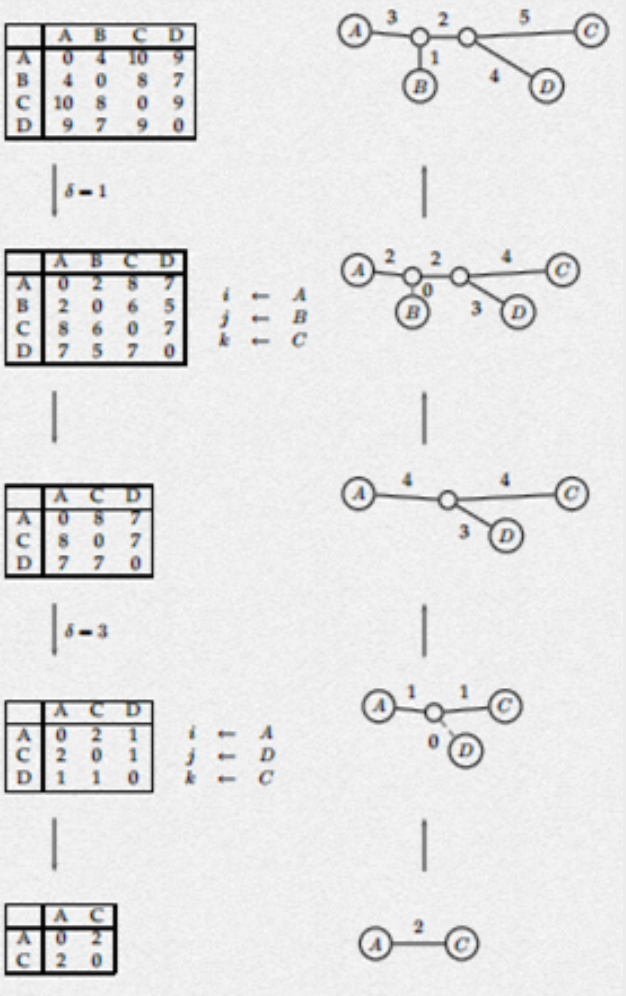
- If distance matrix D has a degenerate triple i,j,k then j can be “removed” from D thus reducing the size of the problem

- If distance matrix D does not have a degenerate triple i,j,k, one can “create” a degenerative triple in D by shortening all hanging edges (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
δ = 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 ← A
j ← B
k ← C
A C D
A 0 8 7
C 8 0 7
D 7 7 0
δ = 3
A C D
A 0 2 1
C 2 0 1
D 1 1 0
i ← A
j ← D
k ← C
A C
A 0 2
C 2 0
A B C D
3 2 5 1 4
A 2 2 4 0 3
A 4 4 3
A 1 1 0
A 2

# Finding Degenerate Triples

- If there is no degenerate triple, all hanging edges are reduced by the same amount δ, so that all pair-wise distances in the matrix are reduced by 2δ.
- Eventually this process collapses one of the leaves (when δ = 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.

# Additive Phylogeny Algorithm

1. **AdditivePhylogeny**($D$)
2. **if** $D$ is a $2$ x $2$ 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 Phylogeny Algorithm

11. Find a triple $i, j, k$ in $D$ such that $D_{ij} + D_{jk} = D_{ik}$
12. $x = D_{ij}$
13. Remove $j^{th}$ row and $j^{th}$ column from $D$
14. $T$ = AdditivePhylogeny($D$)
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 $j$ 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** $T$

# The Four Point Condition

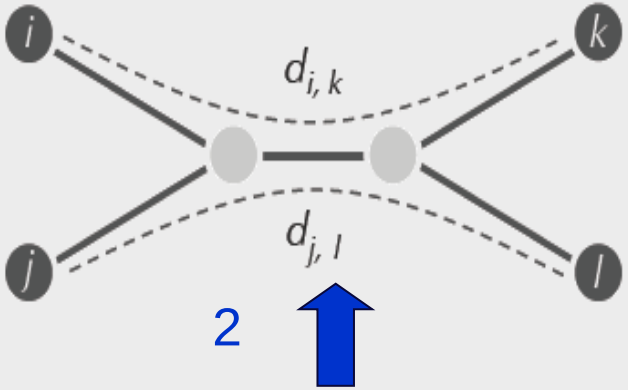
- AdditivePhylogeny provides a way to check if distance matrix D is additive

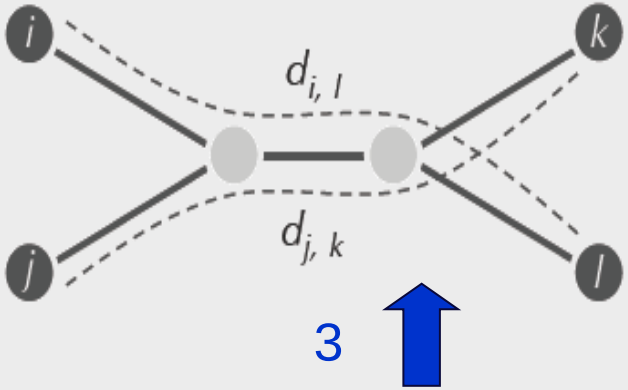
- An even more efficient additivity check is the “four-point condition”

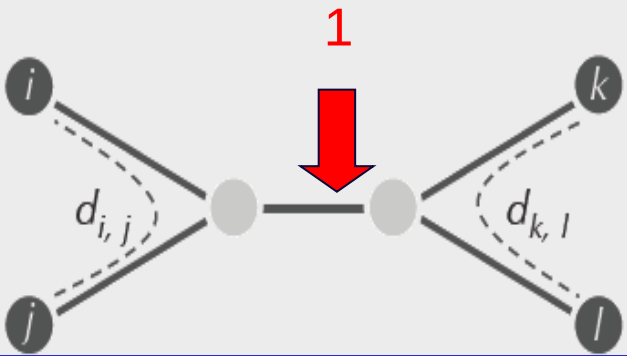
- Let $1 \leq i,j,k,l \leq n$ be four distinct leaves in some (unknown) tree. This quartet induces an (unknown) 4-leaf tree (or 4-tree) where the neighboring pairs are, e.g., i,j and k,l

# The Four Point Condition (cont'd)

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 with 2 X the middle edge**

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

# Four-point formula

when (1,2) and (i,j) are neighbor-couples !

$d(1,2) + d(i,j) < d(i,1) + d(2,j)$

Four-point condition

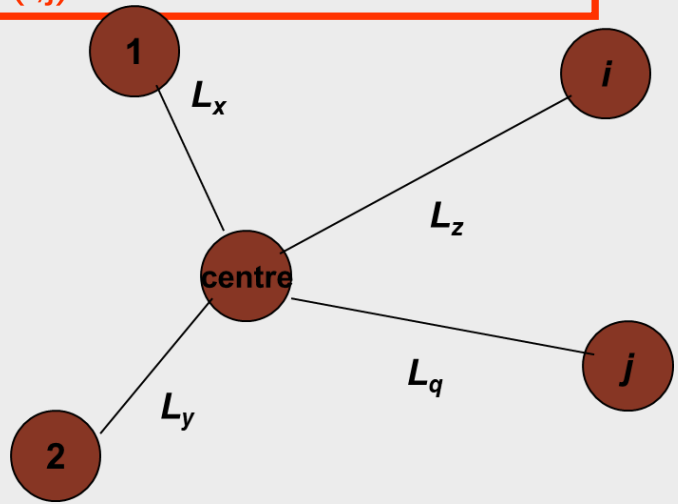
Minimize d(i,j) ***AND*** total distance in tree

$R_i = \sum_j d(t_i, t_j)$

$M(i,j) = (n\text{-}2)d(i,j) - R_i - R_j$

$M(i,j) < M(i,k)$ for all $k$ not equal to $j$

If *i* and *j* are neighbours!



$$D^*_{i,j} = (n - 2) \cdot D_{i,j} - TotalDistance_D(i) - TotalDistance_D(j).$$

**D**

| | *i* | *j* | *k* | *l* | $TotalDistance_D$ |
|---|---|---|---|---|---|
| ***i*** | 0 | 13 | 21 | 22 | 56 |
| ***j*** | 13 | 0 | 12 | 13 | 38 |
| ***k*** | 21 | 12 | 0 | 13 | 46 |
| ***l*** | 22 | 13 | 13 | 0 | 48 |

**D***

| | *i* | *j* | *k* | *l* |
|---|---|---|---|---|
| ***i*** | 0 | -68 | -60 | -60 |
| ***j*** | -68 | 0 | -60 | -60 |
| ***k*** | -60 | -60 | 0 | -68 |
| ***l*** | -60 | -60 | -68 | 0 |

**D**

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $TotalDistance_D$ |
|---|---|---|---|---|---|
| $v_1$ | 0 | 13 | 21 | 22 | 56 |
| $v_2$ | 13 | 0 | 12 | 13 | 38 |
| $v_3$ | 21 | 12 | 0 | 13 | 46 |
| $v_4$ | 22 | 13 | 13 | 0 | 48 |

**D***

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $v_1$ | 0 | -68 | -60 | -60 |
| $v_2$ | -68 | 0 | -60 | -60 |
| $v_3$ | -60 | -60 | 0 | -68 |
| $v_4$ | -60 | -60 | -68 | 0 |

**D**

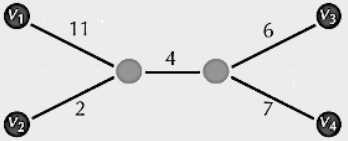
| | $v_5$ | $v_3$ | $v_4$ | $TotalDistance_D$ |
|---|---|---|---|---|
| $v_5$ | 0 | 10 | 11 | 21 |
| $v_3$ | 10 | 0 | 13 | 23 |
| $v_4$ | 11 | 13 | 0 | 24 |

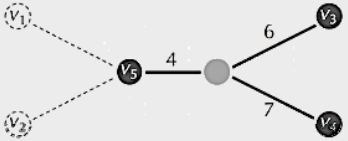
**D***

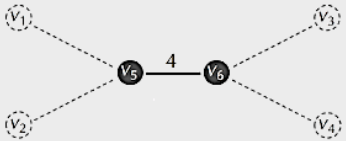
| | $v_5$ | $v_3$ | $v_4$ |
|---|---|---|---|
| $v_5$ | 0 | -34 | -34 |
| $v_3$ | -34 | 0 | -34 |
| $v_4$ | -34 | -34 | 0 |

**D**

| | $v_5$ | $v_6$ |
|---|---|---|
| $v_5$ | 0 | 4 |
| $v_6$ | 4 | 0 |

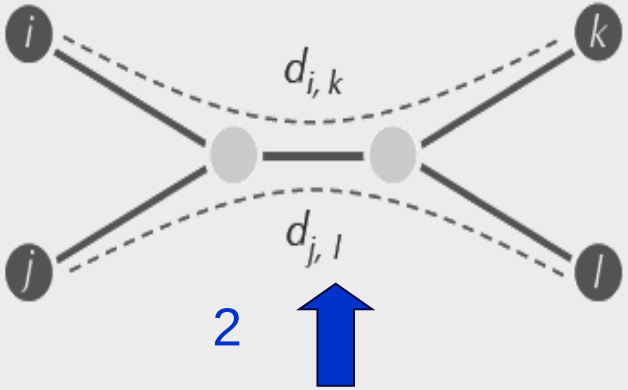


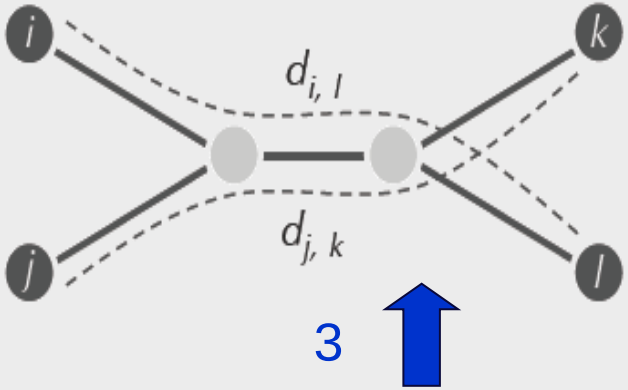



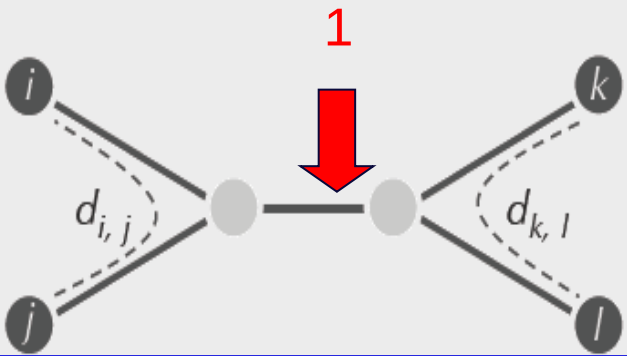



# The Four Point Condition (cont'd)

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 with 2 X the middle edge**

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

# The Four Point Condition: Theorem

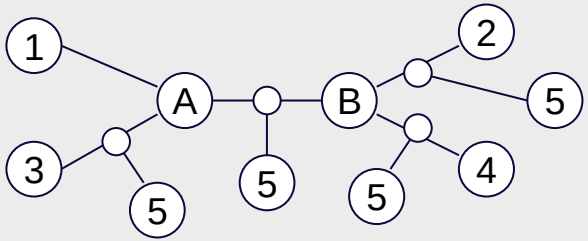
- The four point condition for the quartet i,j,k,l is satisfied if two of these sums are the same, with the third sum smaller than the first two

- Theorem (Buneman): An n x n matrix D is additive if and only if the four point condition holds for every quartet $1 \leq i,j,k,l \leq n$

# Tree reconstruction based on Buneman's Theorem

Consider leaves 1,2,3,4. If $D_{1,3} + D_{2,4} < D_{1,2} + D_{3,4} = D_{1,4} + D_{2,3}$

then their quartet tree is:

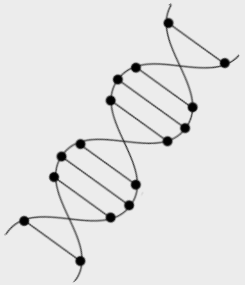



Now we insert leaf 5 into the tree, and consider quartet 1,2,3,5. If : $D_{1,2} + D_{3,5} < D_{1,3} + D_{2,5} = D_{1,5} + D_{2,3}$

then we insert leaf 5 between ancestor A and leaf 3. But If

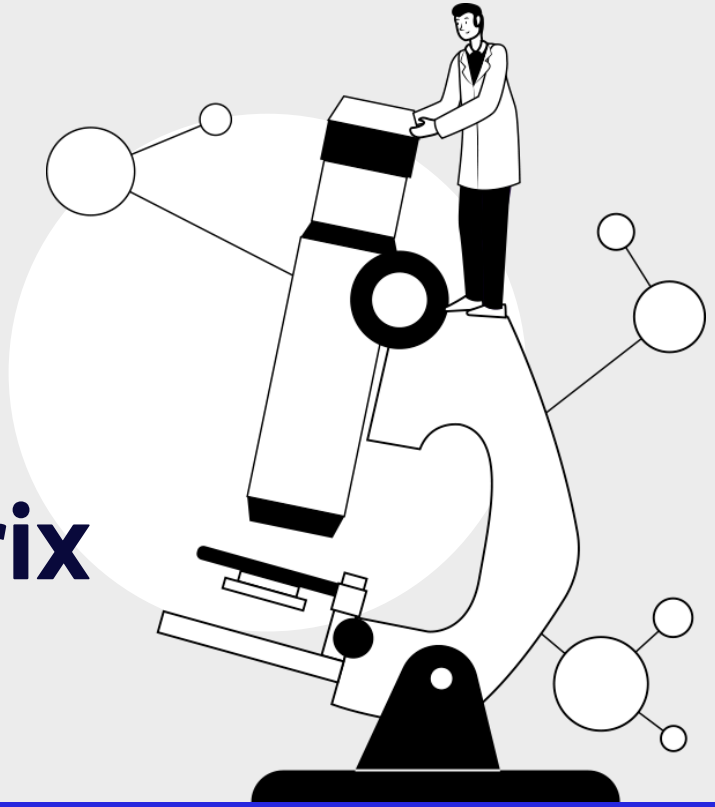
$$D_{1,3} + D_{2,5} < D_{1,2} + D_{3,5} = D_{1,5} + D_{2,3}$$

then leaf 5 could be inserted either between A and B or between B and 2 or between B and 4. So, we need to further consider a quartet 1,2,4,5 and decide the precise edge to insert 5.



# 03

# Tree for Non-additive Matrix



# Least Squares Distance Phylogeny Problem

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

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

- Squared Error is a measure of the quality of the fit between distance matrix and the tree: we want to minimize it.
- Least Squares Distance Phylogeny Problem: Finding the best approximation tree T for a non-additive matrix D (which is NP-hard).

# Ultrametric Evolutionary Trees

- Molecular clock:  measuring evolutionary time
- Assign an age to every node v in a rooted binary tree (denoted AGE(v)), where:
  - all of the leaves of the tree have age 0 because they correspond to present-day species.
- Define the weight of an edge (v,w) as AGE(v) − AGE(w).
- Path length between the root and any node would be equal to the difference between their ages.
  - distance from the root to any leaf is the same
  - ultrametric tree

# UPGMA: Unweighted Pair Group Method with Arithmetic Mean

- UPGMA is a hierarchical clustering algorithm that:
  - Computes the distance between clusters using average pairwise distance (avg link)
  - Assigns a height to every vertex in the tree, effectively assuming the presence of a molecular clock and thus dating every vertex

# Clustering in UPGMA

- Given two disjoint clusters $C_i$, $C_j$ of sequences,

$$d_{ij} = \frac{1}{|C_i| \times |C_j|} \sum_{\{p \in Ci,\, q \in Cj\}} d_{pq}$$

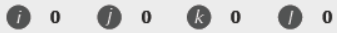
- Note that if $C_k = C_i \cup C_j$, then distance to another cluster $C_l$ is:

$$d_{kl} = \frac{d_{il} |C_i| + d_{jl} |C_j|}{|C_i| + |C_j|}$$

# UPGMA Algorithm

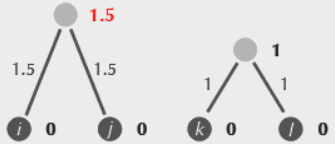
**UPGMA**$(D, n)$
- $Clusters \leftarrow n$ single-element clusters labeled $1, \ldots, n$
- construct a graph $T$ with $n$ isolated nodes labeled by single elements $1, \ldots, n$
- **for** every node $v$ in $T$
  - $\text{AGE}(v) \leftarrow 0$
- **while** there is more than one cluster
  - find the two closest clusters $C_i$ and $C_j$ (break ties arbitrarily)
  - merge $C_i$ and $C_j$ into a new cluster $C_{\text{new}}$ with $|C_i| + |C_j|$ elements
  - add a new node labeled by cluster $C_{\text{new}}$ to $T$
  - connect node $C_{\text{new}}$ to $C_i$ and $C_j$ by directed edges
  - $\text{AGE}(C) \leftarrow D_{C_i, C_j} / 2$
  - remove the rows and columns of $D$ corresponding to $C_i$ and $C_j$
  - remove $C_i$ and $C_j$ from *Clusters*
  - add a row/column to $D$ for $C_{\text{new}}$ by computing $D(C_{\text{new}}, C)$ for each $C$ in *Clusters*
  - add $C_{\text{new}}$ to *Clusters*
- *root* $\leftarrow$ the node in $T$ corresponding to the remaining cluster
- **for** each edge $(v, w)$ in $T$
  - length of $(v, w) \leftarrow \text{AGE}(v) - \text{AGE}(w)$
- **return** $T$

| | *i* | *j* | *k* | *l* |
|---|---|---|---|---|
| *i* | 0 | 3 | 4 | 3 |
| *j* | 3 | 0 | 4 | 5 |
| *k* | 4 | 4 | 0 | 2 |
| *l* | 3 | 5 | 2 | 0 |

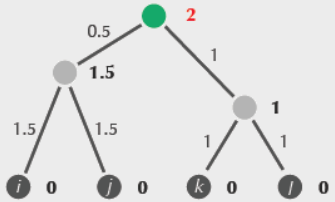



| | *i* | *j* | *k* | *l* |
|---|---|---|---|---|
| *i* | 0 | 3 | 4 | 3 |
| *j* | 3 | 0 | 4 | 5 |
| *k* | 4 | 4 | 0 | **2** |
| *l* | 3 | 5 | **2** | 0 |


| | *i* | *j* | {*k*, *l*} |
|---|---|---|---|
| *i* | 0 | **3** | 3.5 |
| *j* | **3** | 0 | 4.5 |
| {*k*, *l*} | 3.5 | 4.5 | 0 |




| | {*i*, *j*} | {*k*, *l*} |
|---|---|---|
| {*i*, *j*} | 0 | **4** |
| {*k*, *l*} | **4** | 0 |



# UPGMA’s Pros. & Cons.

⊕ UPGMA offers a step forward from ADDITIVE PHYLOGENY, since it can analyze non-additive distance matrices.

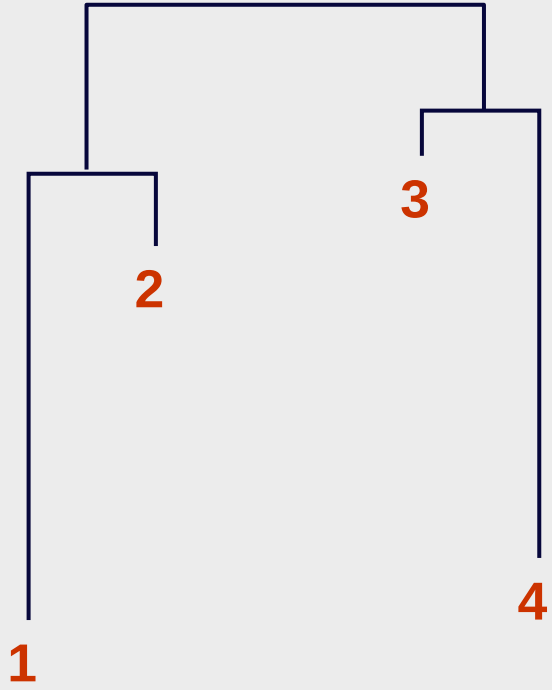
— The algorithm produces an ultrametric tree:

(the distance from the root to any leaf is the same)

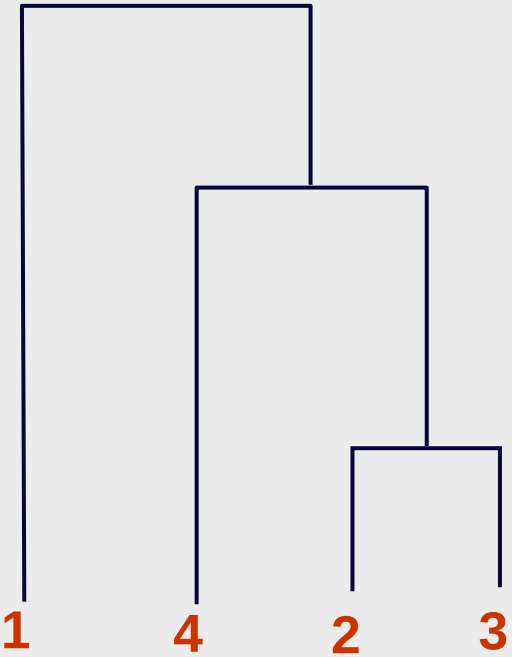
- UPGMA assumes a constant molecular clock: all species represented by the leaves in the tree are assumed to accumulate mutations (and thus evolve) at the same rate.

# UPGMA's Weakness: Example

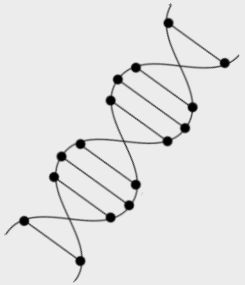
Correct tree




UPGMA

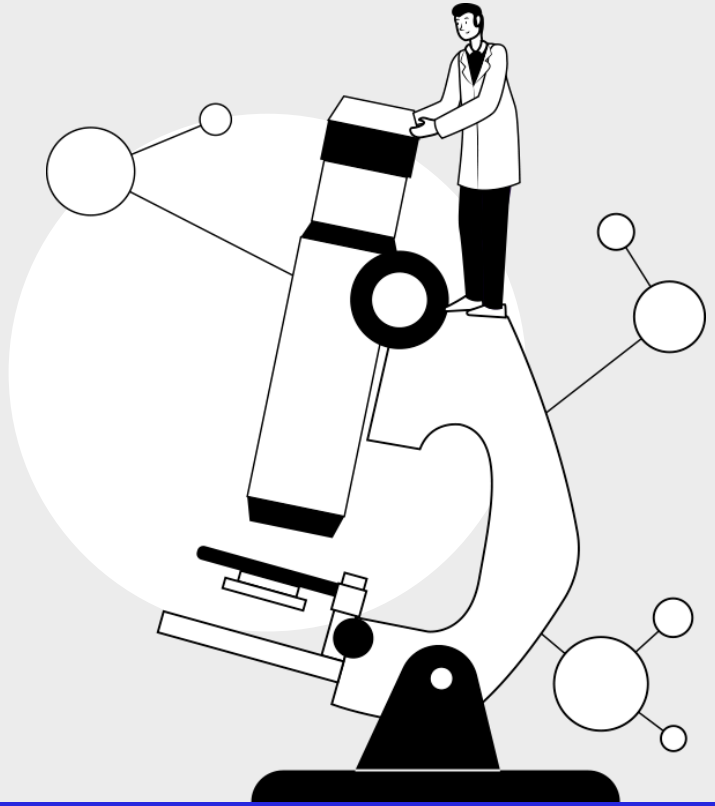



**This is called long branch attraction**



# 04

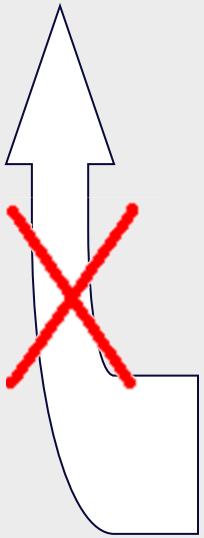
# Character Based Phylogeny



# Alignment Matrix vs. Distance Matrix

Sequence a gene of length l nucleotides (or amino acids) in n species to generate an n x m (multiple) alignment matrix

CANNOT be transformed back into alignment matrix because information was lost on the forward transformation



Transform into

*n* x *n* distance matrix

# Character-Based Phylogenetic Reconstruction

A potentially better technique:

- Character-based reconstruction algorithms use the n x m alignment matrix

  (n = # species, m = #characters)

  directly instead of using distance matrix
- GOAL: Determine a tree and the character strings at the internal nodes of the tree that would best explain the n observed character strings at the leaves (species)

# Character-Based Phylogenetic Reconstruction (cont'd)

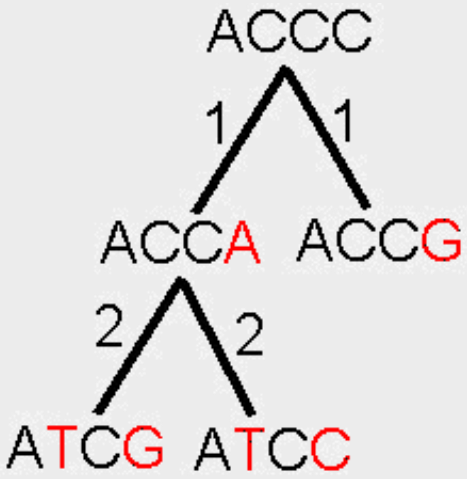
- Characters may be nucleotides, where A, G, C, T are the states of this character, or amino acids. Other characters may be the # of eyes or legs or the shape of a beak or a fin.

- By setting the length of an edge in the tree to the (weighted) Hamming distance, we may define the parsimony score (actually cost) of the tree as the sum of the lengths (i.e., weights) of the edges.

# Parsimony Approach to Evolutionary Tree Reconstruction

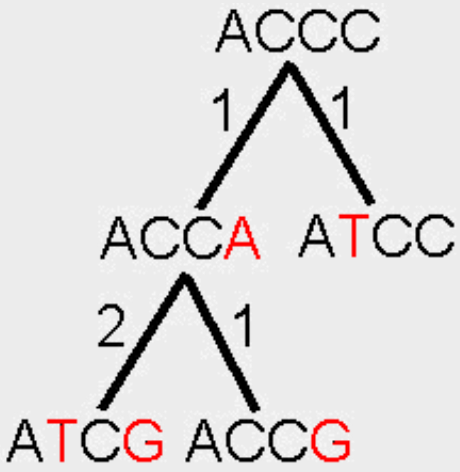
- Assumes the observed character differences at the leaves resulted from the fewest possible mutations in the tree
- Seeks the tree that yields lowest possible parsimony score - sum of cost of all mutations found in the tree
- Suffers from long branch attraction described on slide 69 and is statistically inconsistent, but still among the most popular methods in practice

# Parsimony and Tree Reconstruction




Parsimonious

Score: 6




Parsimonious

Score: 5

# Small Parsimony Problem

- **Input**: Tree T with each leaf labeled by an  m-character string.
- **Output**: Labeling of all internal nodes of the tree T minimizing the parsimony score (i.e., total Hamming distance or number of mutations).
- We can assume that every leaf is labeled by a single character, because the characters in the string are independent in this problem.

# Weighted Small Parsimony Problem

- A more general version of the Small Parsimony Problem
- Input includes a k*k scoring matrix describing the cost of transformation/mutation from each of the k states into another one
- Weighted Hamming distance is used instead
- For the Small Parsimony problem, the scoring matrix is based on Hamming distance

  $d_H(v, w) = 0$ if $v=w$

  $d_H(v, w) = 1$ otherwise
- Equivalent to calculating the Tree Score of a multiple sequence alignment under a given evolutionary tree

# Scoring Matrices

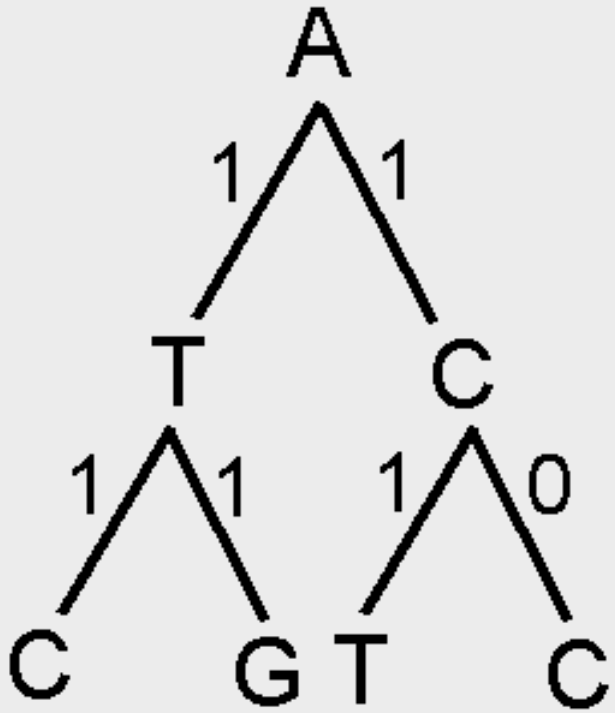
Small Parsimony Problem

| | 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

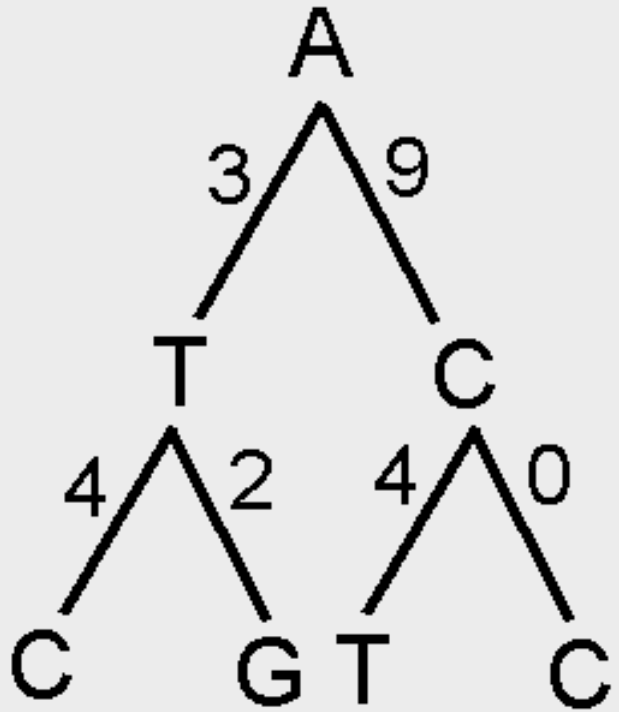



Small Parsimony Scoring Matrix

| | 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 a state and a k x k scoring matrix ($\delta_{ij}$), assuming the character has k states.

- **Output**: Labeling of each internal vertex of the tree T with a state minimizing the total weighted parsimony score

# Sankoff’s Algorithm

- Consider each state at a node and check every state of each child to determine the minimum score under the given state
- An example

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

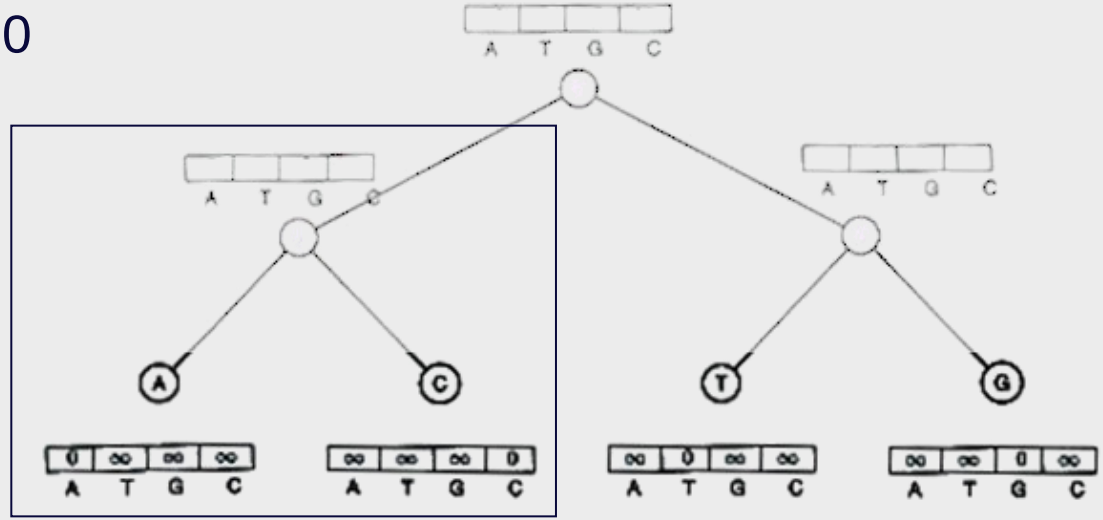
# Sankoff Algorithm: Dynamic Programming

- Calculate and keep track of the score for every possible label/state at each vertex
- $s_t(v)$ = minimum parsimony score of the sub-tree rooted at vertex v if v has state t
- The score at each vertex is based on scores of its children:

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

# Sankoff Algorithm (cont.)

- Begin at leaves:
  - If leaf has the state in question, score is 0
  - Else, score is ∞



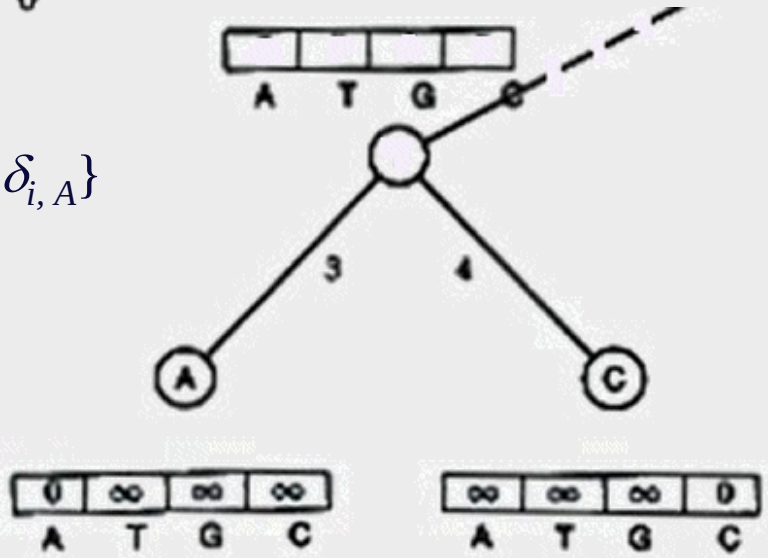
# Sankoff Algorithm (cont.)

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

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

$$s_A(v) = \min_i\{s_i(u) + \delta_{i,A}\} + \min_j\{s_j(w) + \delta_{j,A}\}$$

$$s_A(v) = 0$$




| | $s_i(u)$ | $\delta_{i,A}$ | sum |
|---|---|---|---|
| A | 0 | 0 | **0** |
| T | ∞ | 3 | ∞ |
| G | ∞ | 4 | ∞ |
| C | ∞ | 9 | ∞ |

# Sankoff Algorithm (cont.)

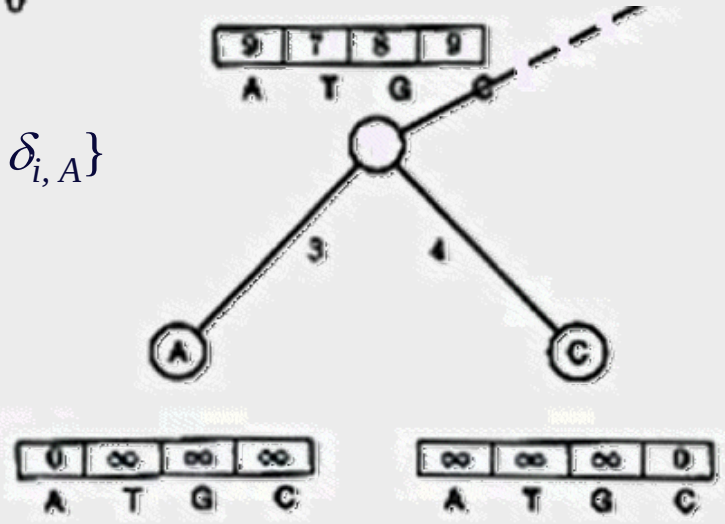
| δ | A | T | G | C |
|---|---|---|---|---|
| A | 0 | 3 | 4 | 9 |
| T | 3 | 0 | 2 | 4 |
| G | 4 | 2 | 0 | 4 |
| C | 9 | 4 | 4 | 0 |

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

$$s_A(v) = \min_i\{s_i(u) + \delta_{i,A}\} + \min_j\{s_j(w) + \delta_{j,A}\}$$

$$s_A(v) = 0$$

$$+ 9 = \mathbf{9}$$

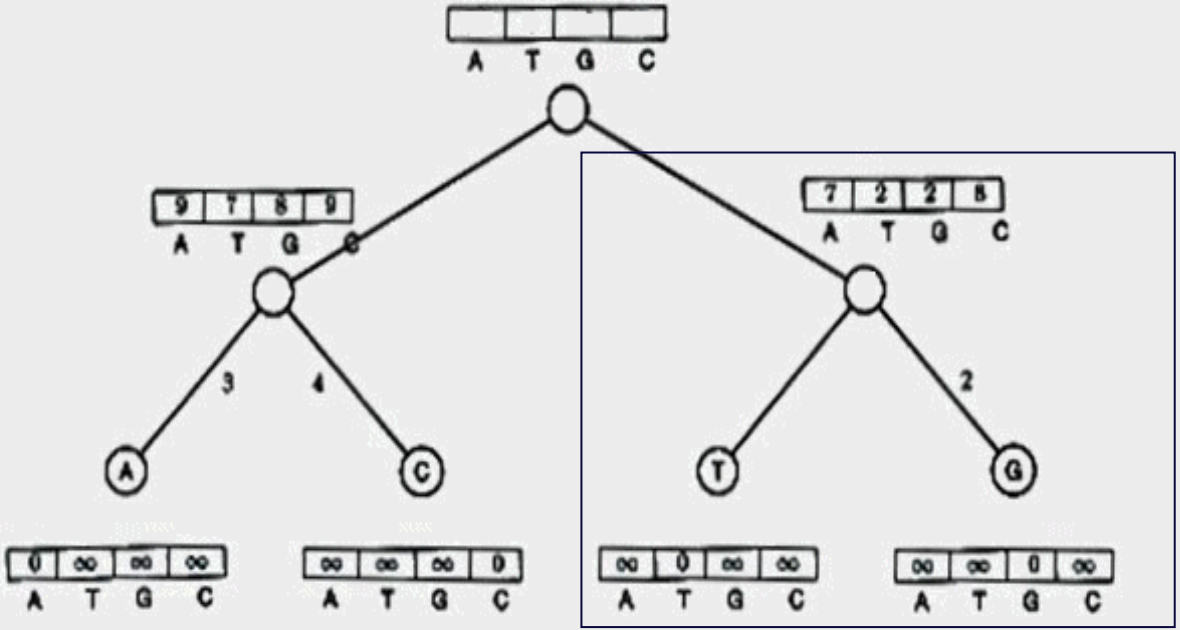



| | $s_j(w)$ | $\delta_{j,A}$ | sum |
|---|---|---|---|
| A | ∞ | 0 | ∞ |
| T | ∞ | 3 | ∞ |
| G | ∞ | 4 | ∞ |
| C | 0 | 9 | **9** |

Repeat for T, G, and C

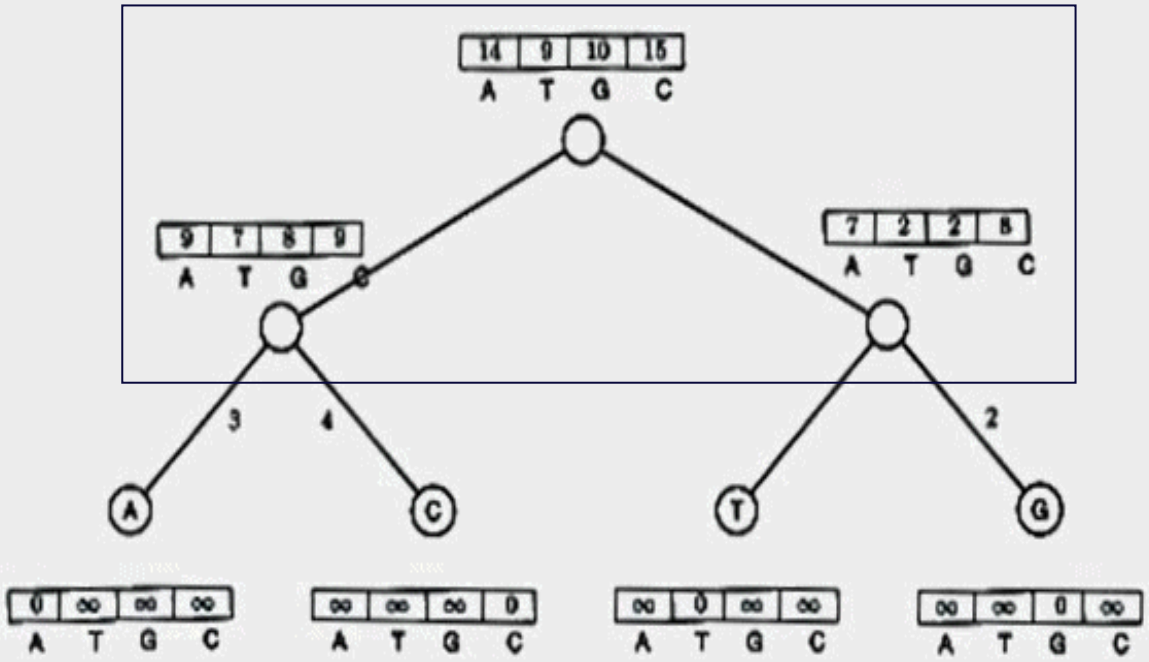
# Sankoff Algorithm (cont.)

Repeat for right subtree



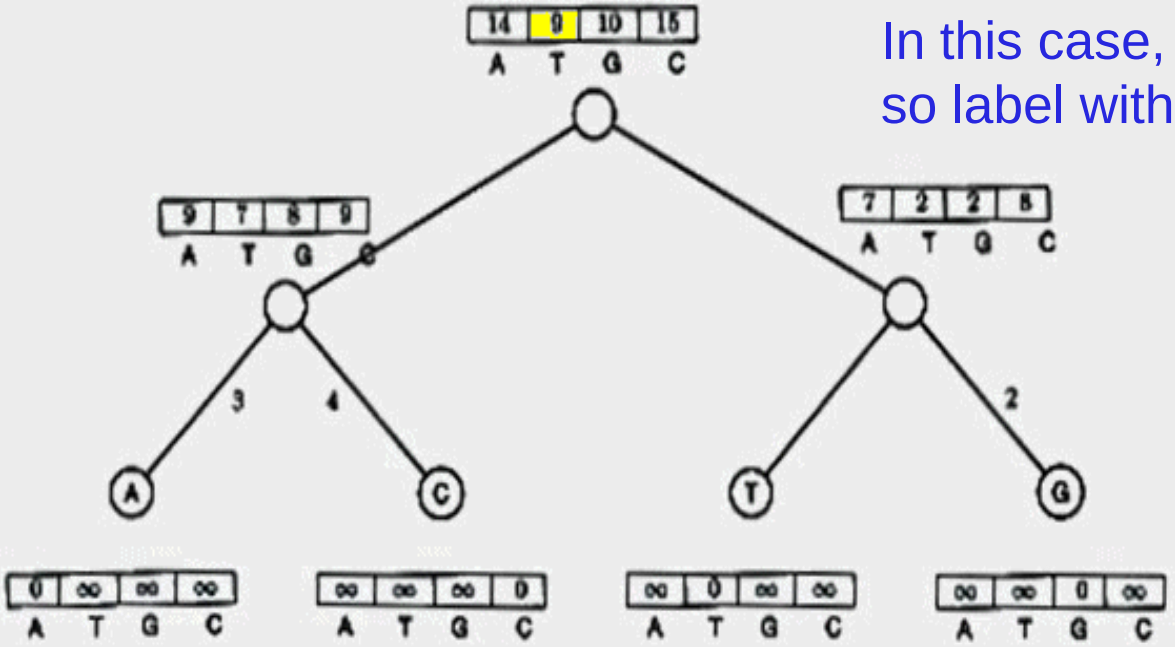
# Sankoff Algorithm (cont.)

Repeat for root



# Sankoff Algorithm (cont.)

Smallest score at root is minimum weighted parsimony score




In this case, 9 – so label with T

# Sankoff Algorithm: Traveling down the Tree

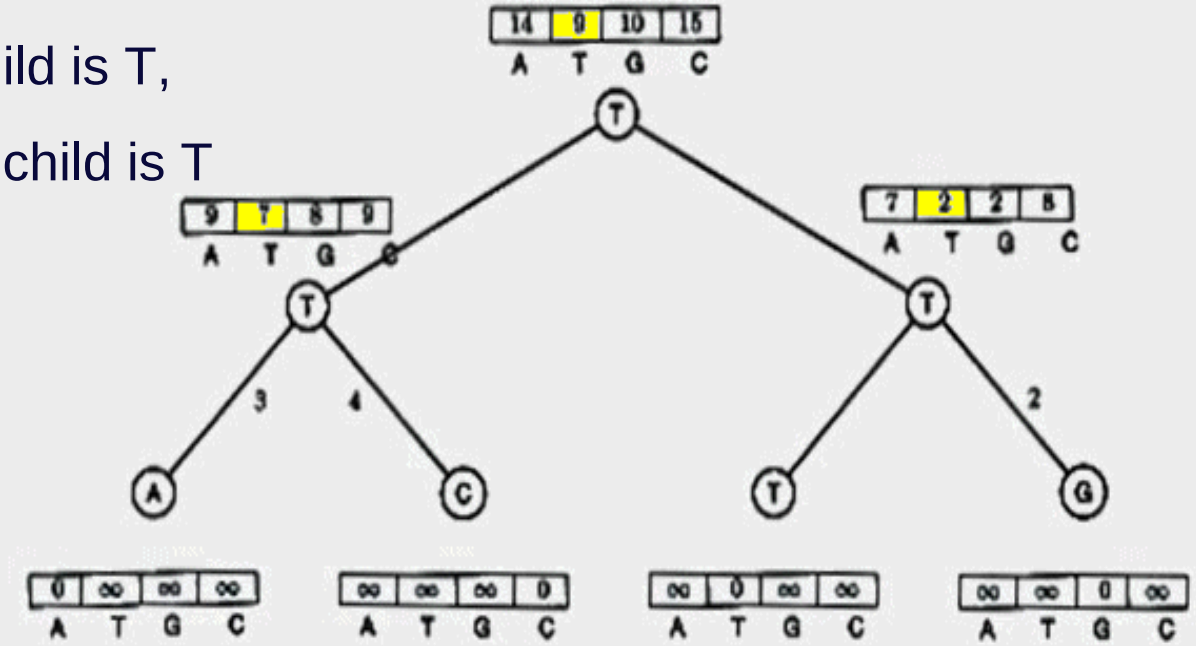
- The scores at the root vertex have been computed by going up the tree
- After the scores at root vertex are computed the Sankoff algorithm moves down the tree and assign each vertex with the optimal state (backtracing).

# Sankoff Algorithm (cont.)

9 is derived from 7 + 2

So left child is T,

and right child is T



# Fitch’s Algorithm

- Solves the Small Parsimony problem
- Dynamic programming in essence
- Assigns a set of states to every vertex in the tree
- Each leaf has an input state
- If the two children’s sets of states overlap, it’s the common set of them
- If not, it’s the combined set (union) of them

# Fitch Algorithm

1) Assign a set of possible states to every vertex, traversing the tree from leaves to root

- Each node’s set is either the intersection or union of its children’s sets (a leaf’s set consists of its label)
- E.g., if the node we are looking at has a left child labeled {A, C} and a right child labeled {A, T} (or {T}), the node will be given the set {A} (or {A,C,T}).

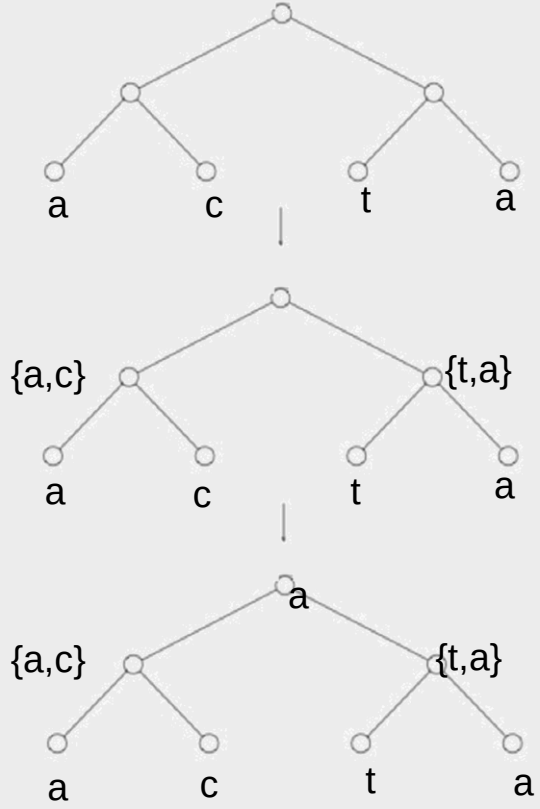
# Fitch Algorithm (cont.)

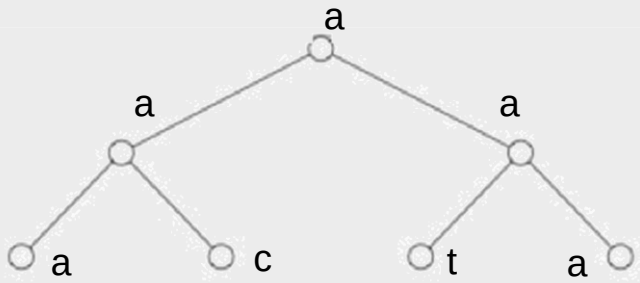
2) Assign labels to each vertex, traversing the tree from root to leaves

- Assign the root an arbitrary state from its set of states
- For all other vertices, if its parent's label is in its set of states, assign it its parent's label
- Else, choose an arbitrary state from its set as its label

# Fitch Algorithm (cont.)

An example:



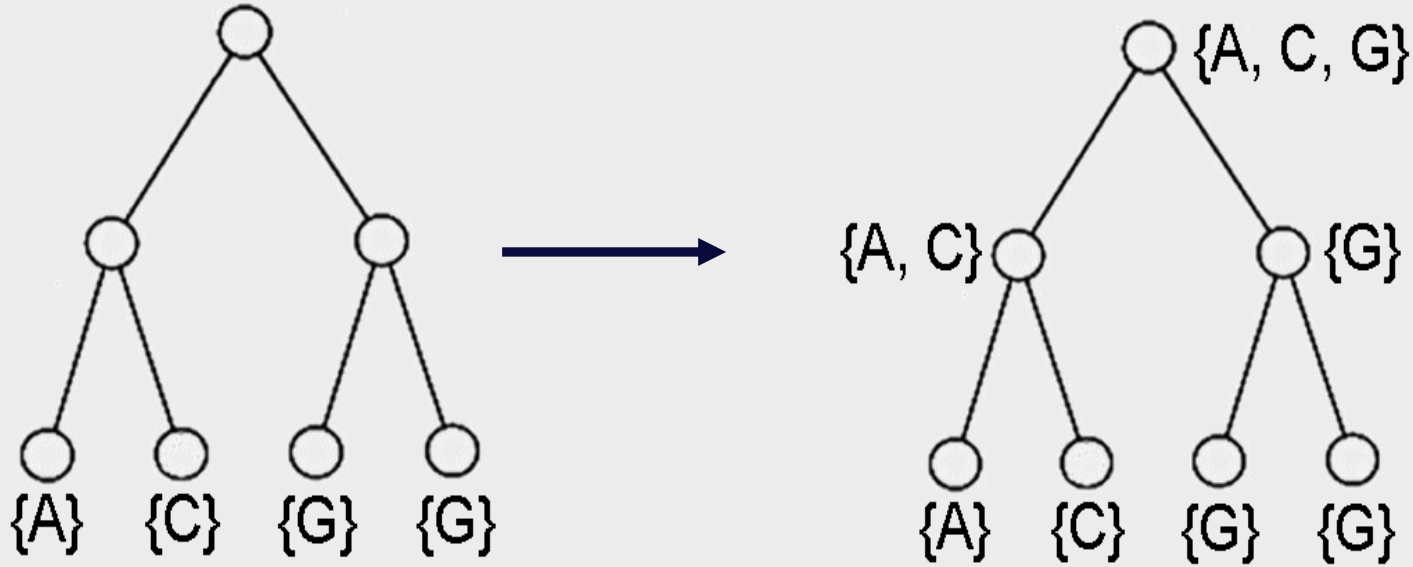



# Fitch vs. Sankoff

- Both have an O(nk) running time
- Are they actually different?
- Let’s compare …

# Fitch

As seen previously:



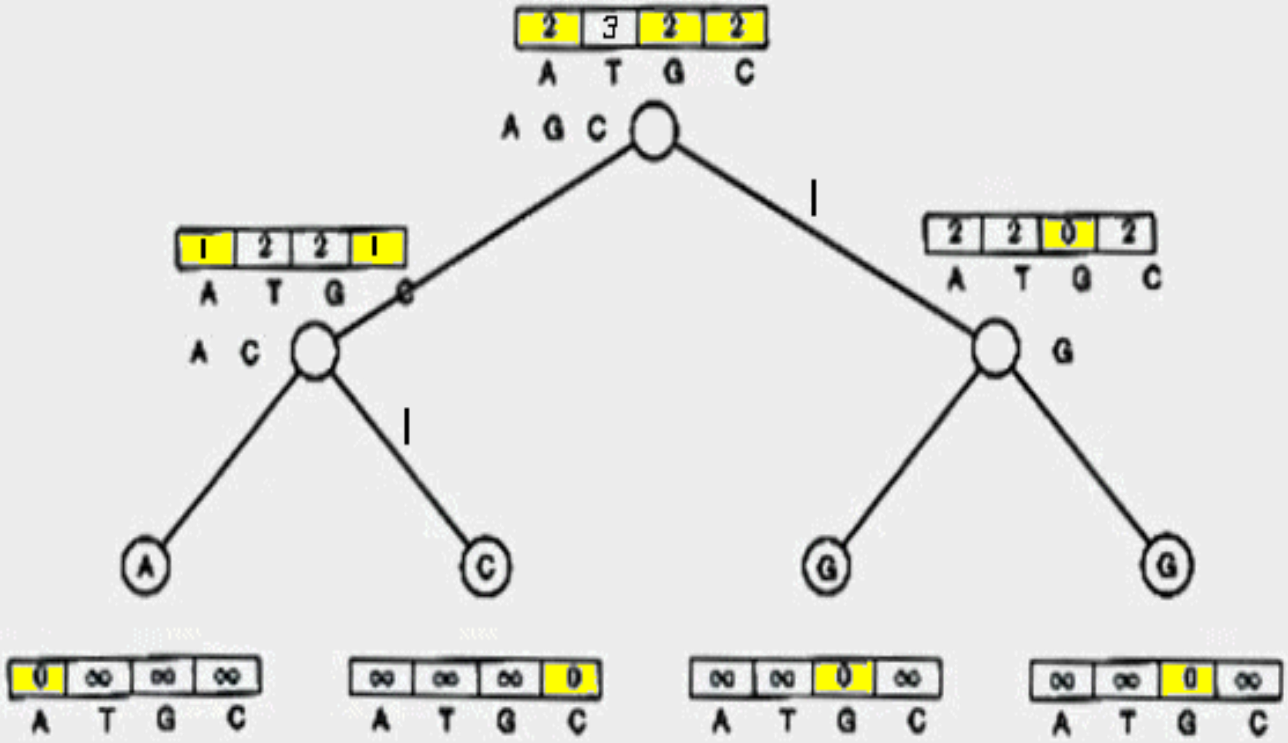
# Comparison of Fitch and Sankoff

- 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


2 3 2 2
A T G C
A G C
1 2 2 1
A T G C
A C
2 2 0 2
A T G C
G
1
1
A
C
G
G
0 ∞ ∞ ∞
A T G C
∞ ∞ ∞ 0
A T G C
∞ ∞ 0 ∞
A T G C
∞ ∞ 0 ∞
A T G C

# Sankoff vs. Fitch

- The Sankoff algorithm gives the same set of optimal labels as the Fitch algorithm
- For Sankoff algorithm, letter t is optimal for vertex v if $v$ if

$$s_t(v) = \min_{1 \le i \le k} s_i(v)$$

  - Denote the set of optimal letters at vertex v as S(v)
    - If S(left child) and S(right child) overlap, S(parent) is the intersection
    - Else it's the union of S(left child) and S(right child)
    - This is also the Fitch recurrence
- The two algorithms are **identical**

# Sources


- http://www.math.tau.ac.il/~rshamir/ge/02/scribes/lec01.pdf
- http://bioinformatics.oupjournals.org/cgi/screenpdf/20/3/340.pdf
- http://www.absoluteastronomy.com/encyclopedia/M/Mi/Minimum_spanning_tree.htm
- Serafim Batzoglou (UPGMA slides) http://www.stanford.edu/class/cs262/Slides
- Watkins, W.S., Rogers A.R., Ostler C.T., Wooding, S., Bamshad M. J., Brassington A.E., Carroll M.L., Nguyen S.V., Walker J.A., Prasas, R., Reddy P.G., Das P.K., Batzer M.A., Jorde, L.B.: Genetic Variation Among World Populations: Inferences From 100 Alu Insertion Polymorphisms