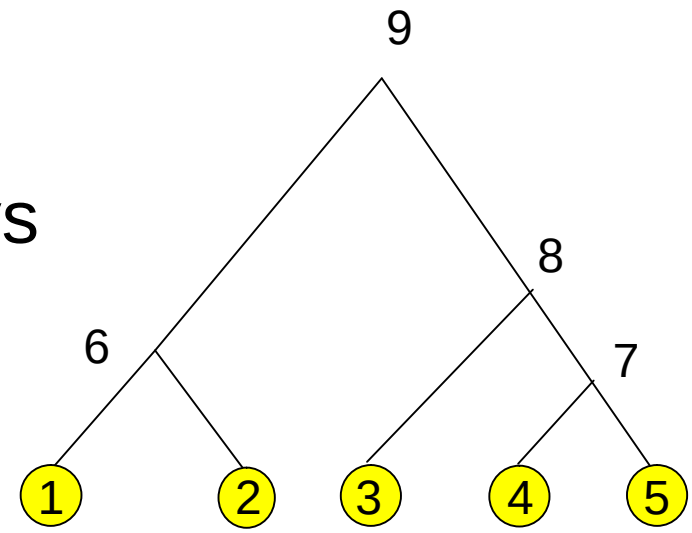
Computing parsimony

- Parsimony treats each site (position in a sequence) independently
- Total parsimony cost is the sum of parsimony costs of each site
- We can compute the minimal parsimony cost for a given tree by
  - First finding out possible assignments at each node, starting from leaves and proceeding towards the root
  - Then, starting from the root, assign a letter at each node, proceeding towards leaves

Labelling tree nodes

- An unrooted tree with *n* leaves contains 2n-1 nodes altogether
- Assign the following labels to nodes in a rooted tree
  - leaf nodes: 1, 2, …, n
  - internal nodes: n+1, n+2, …, 2n-1
  - root node: 2n-1
- The label of a child node is always smaller than the label of the parent node



Parsimony algorithm: first phase

- Find out possible assignments at every node for each site u independently. Denote site u in sequence i by $s_{i,u}$.

For i := 1, …, n do

  $F_i := \{s_{i,u}\}$ % possible assignments at node i

  $L_i := 0$ % number of substitutions up to node i

For i := n+1, …, 2n-1 do

  Let j and k be the children of node i

  If $F_j \cap F_k = \varnothing$ then $L_i := L_j + L_k + 1$, $F_i := F_j \cup F_k$

    else $L_i := L_j + L_k$, $F_i := F_j \cap F_k$

Parsimony algorithm: first phase

Choose u = 3 (for example, in general we do this for all u)

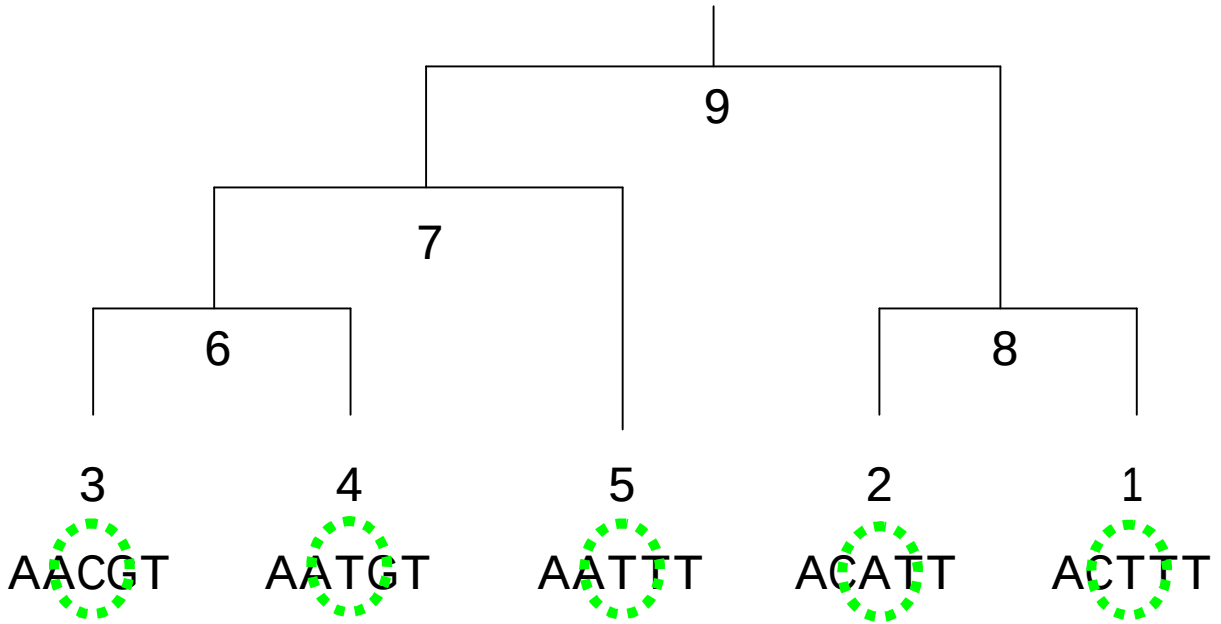
$F_1 := \{T\}$
$L_1 := 0$

$F_2 := \{A\}$
$L_2 := 0$

$F_3 := \{C\}$, $L_3 := 0$

$F_4 := \{T\}$, $L_4 := 0$

$F_5 := \{T\}$, $L_5 := 0$



Parsimony algorithm: first phase

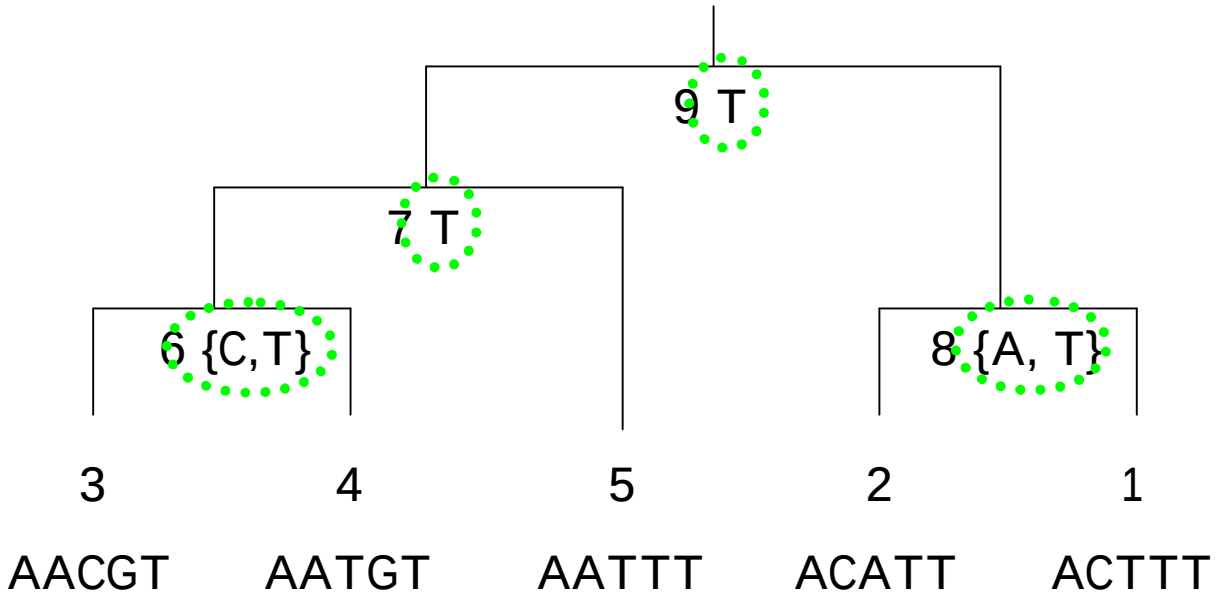
$F_6 := F_3 \cup F_4 = \{C, T\}$

$L_6 := L_3 + L_4 + 1 = 1$

$F_7 := F_5 \cap F_6 = \{T\}$
$L_7 := L_5 + L_6 = 1$

$F_8 := F_1 \cup F_2 = \{A, T\}$
$L_8 := L_1 + L_2 + 1 = 1$

$F_9 := F_7 \cap F_8 = \{T\}$
$L_9 := L_7 + L_8 = 2$



⇨ Parsimony cost for site 3 is 2

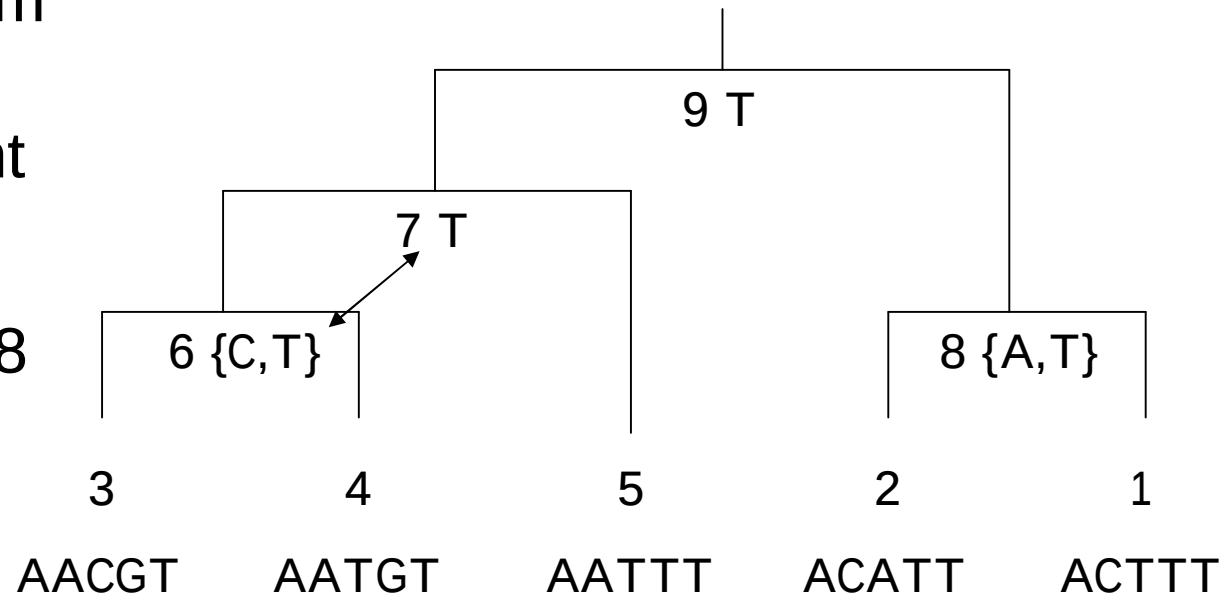
Parsimony algorithm: second phase

- Backtrack from the root and assign $x \in F_i$ at each node
- If we assigned y at parent of node i and $y \in F_i$, then assign y
- Else assign $x \in F_i$ by random

Parsimony algorithm: second phase

At node 6, the algorithm assigns T because T was assigned to parent node 7 and T $\in F_6$.

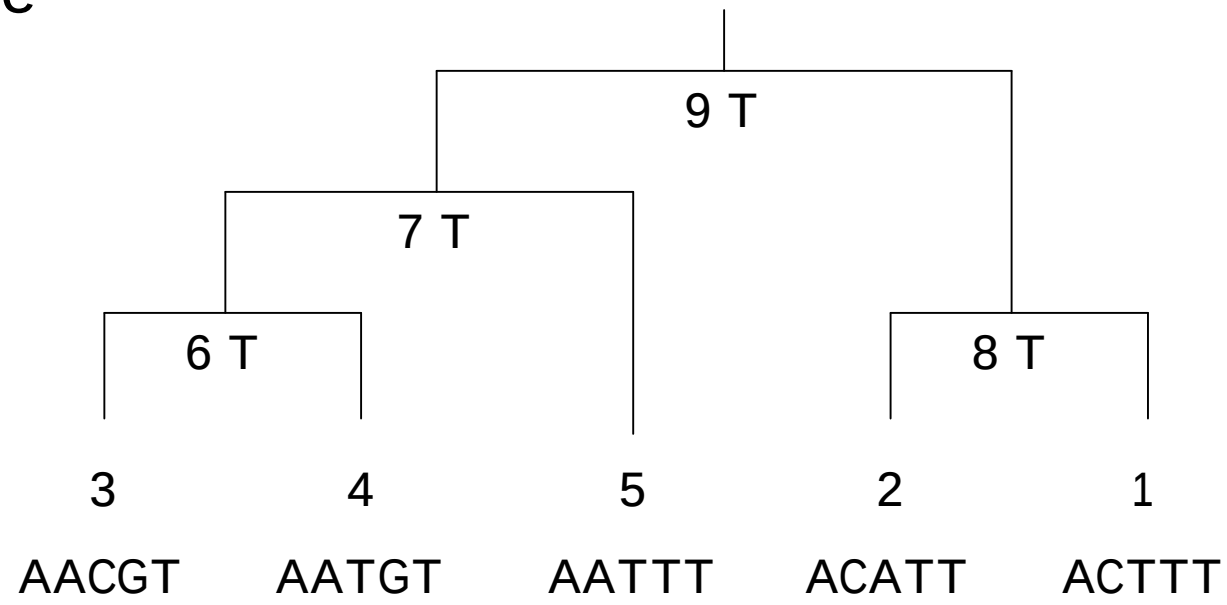
T is assigned to node 8 for the same reason.



The other nodes have only one possible letter to assign

Parsimony algorithm

First and second phase are repeated for each site in the sequences,
summing the parsimony costs at each site



Properties of parsimony algorithm

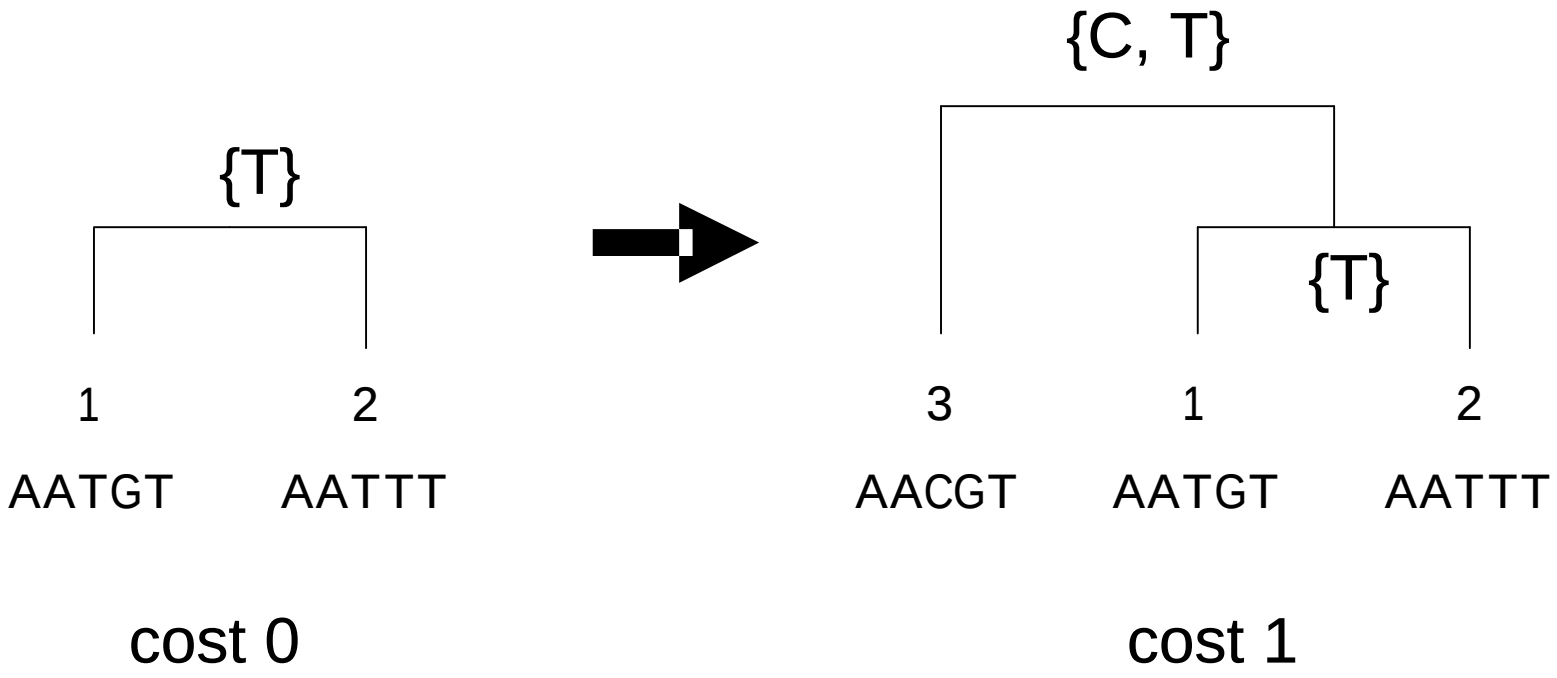
- Parsimony algorithm requires that the sequences are of same length
  - First align the sequences against each other and remove indels
  - Then compute parsimony for the resulting sequences
- Is the most parsimonious tree the correct tree?
  - Not necessarily but it explains the sequences with least number of substitutions
  - We can assume that the probability of having fewer mutations is higher than having many mutations

Finding the most parsimonious tree

- Parsimony algorithm calculates the parsimony cost for a given tree…
- …but we still have the problem of finding the tree with the lowest cost
- Exhaustive search (enumerating all trees) is in general impossible
- More efficient methods exist, for example
  - Probabilistic search
  - Branch and bound

Branch and bound in parsimony

- We can exploit the fact that adding edges to a tree can only increase the parsimony cost



Branch and bound in parsimony

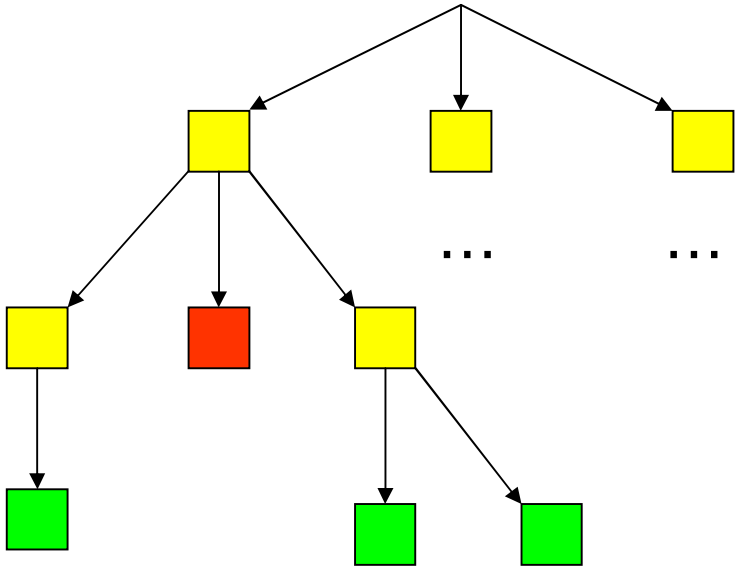
Branch and bound is a general search strategy where

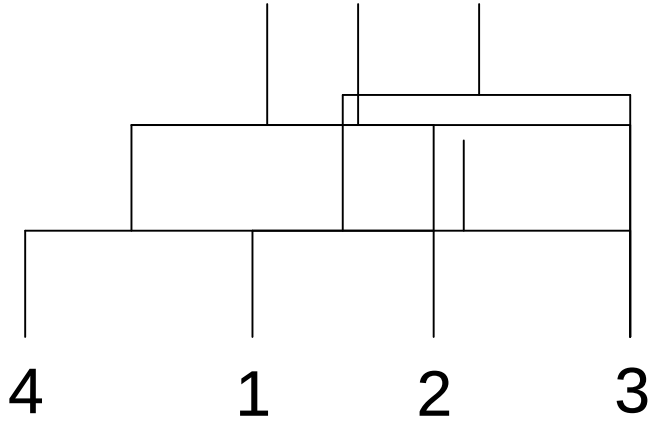
- Each solution is potentially generated
- Track is kept of the best solution found
- If a partial solution cannot achieve better score, we abandon the current search path

In parsimony…

- Start from a tree with 1 sequence
- Add a sequence to the tree and calculate parsimony cost
- If the tree is complete, check if found the best tree so far
- If tree is not complete and cost exceeds best tree cost, do not continue adding edges to this tree

Branch and bound graphically





Partial tree, no best complete tree constructed yet

Complete tree: calculate parsimony cost and store

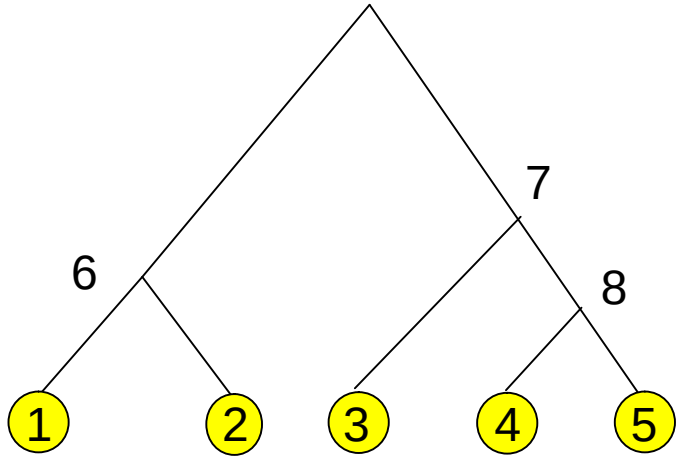
Partial tree, cost exceeds the cost of the best tree this far

Distance methods

- The parsimony method works on sequence (character string) data
- We can also build phylogenetic trees in a more general setting
- *Distance methods* work on a set of pairwise distances d_{ij} for the data
- Distances can be obtained from phenotypes as well as from genotypes (sequences)

Distances in a phylogenetic tree

- Distance matrix D = (d_{ij}) gives pairwise distances for *leaves* of the phylogenetic tree
- In addition, the phylogenetic tree will now specify distances between leaves and internal nodes
  - Denote these with d_{ij} as well



Distance d_{ij} states how far apart species i and j are evolutionary (e.g., number of mismatches in aligned sequences)

Distances in evolutionary context

- Distances d_{ij} in evolutionary context satisfy the following conditions
  - Symmetry: $d_{ij} = d_{ji}$ for each i, j
  - Distinguishability: $d_{ij} \neq 0$ if and only if $i \neq j$
  - Triangle inequality: $d_{ij} \leq d_{ik} + d_{kj}$ for each i, j, k
- Distances satisfying these conditions are called metric
- In addition, evolutionary mechanisms may impose additional constraints on the distances
  - ▷ *additive* and *ultrametric* distances

Additive trees

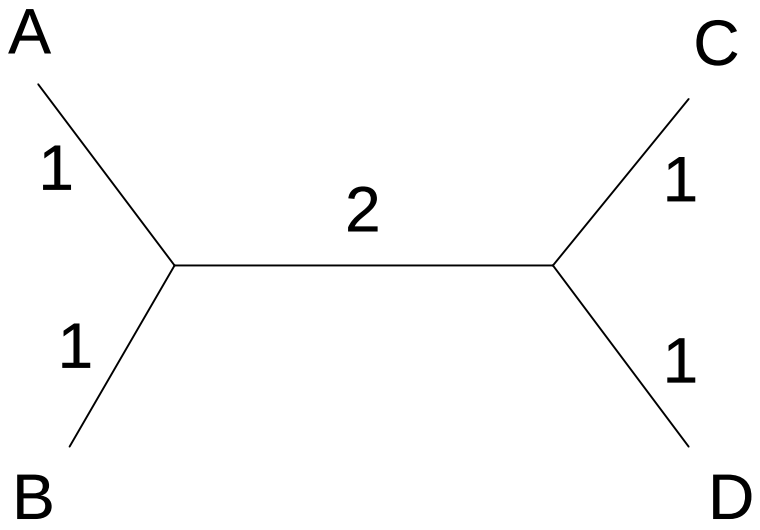
- A tree is called *additive*, if the distance between any pair of leaves (i, j) is the sum of the distances between the leaves and the first node k that they share in the tree

$$d_{ij} = d_{ik} + d_{jk}$$

- ”Follow the path from the leaf i to the leaf j to find the exact distance d_{ij} between the leaves.”

Additive trees: example

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



Ultrametric trees

- A rooted additive tree is called a *ultrametric tree*, if the distances between any two leaves i and j, and their common ancestor k are equal

  $d_{ik} = d_{jk}$

- Edge length d_{ij} corresponds to the time elapsed since divergence of i and j from the common parent
- In other words, edge lengths are measured by a *molecular clock* with a constant rate

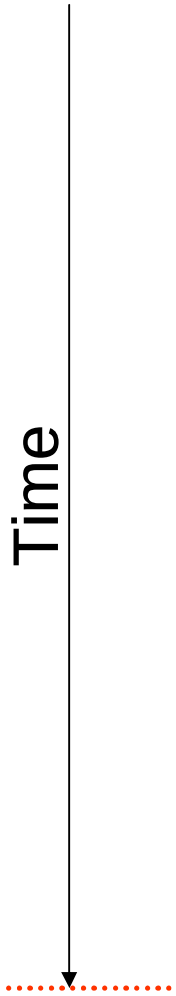
Identifying ultrametric data

- We can identify distances to be ultrametric by the three-point condition:

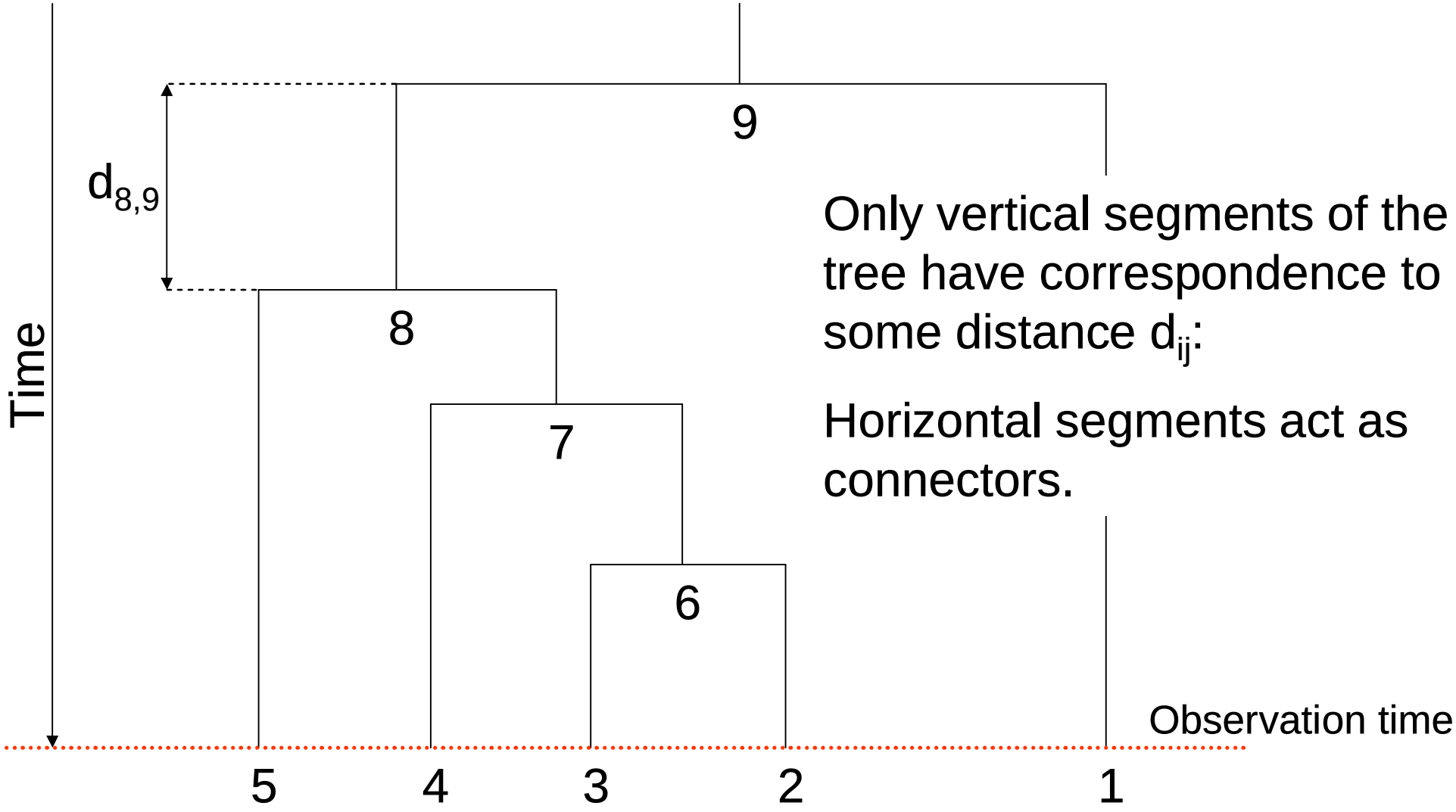
  D corresponds to an ultrametric tree if and only if for any three **species** i, j and k, the distances satisfy $d_{ij} \leq \max(d_{ik}, d_{kj})$

- If we find out that the data is ultrametric, we can utilise a simple algorithm to find the corresponding tree

Ultrametric trees



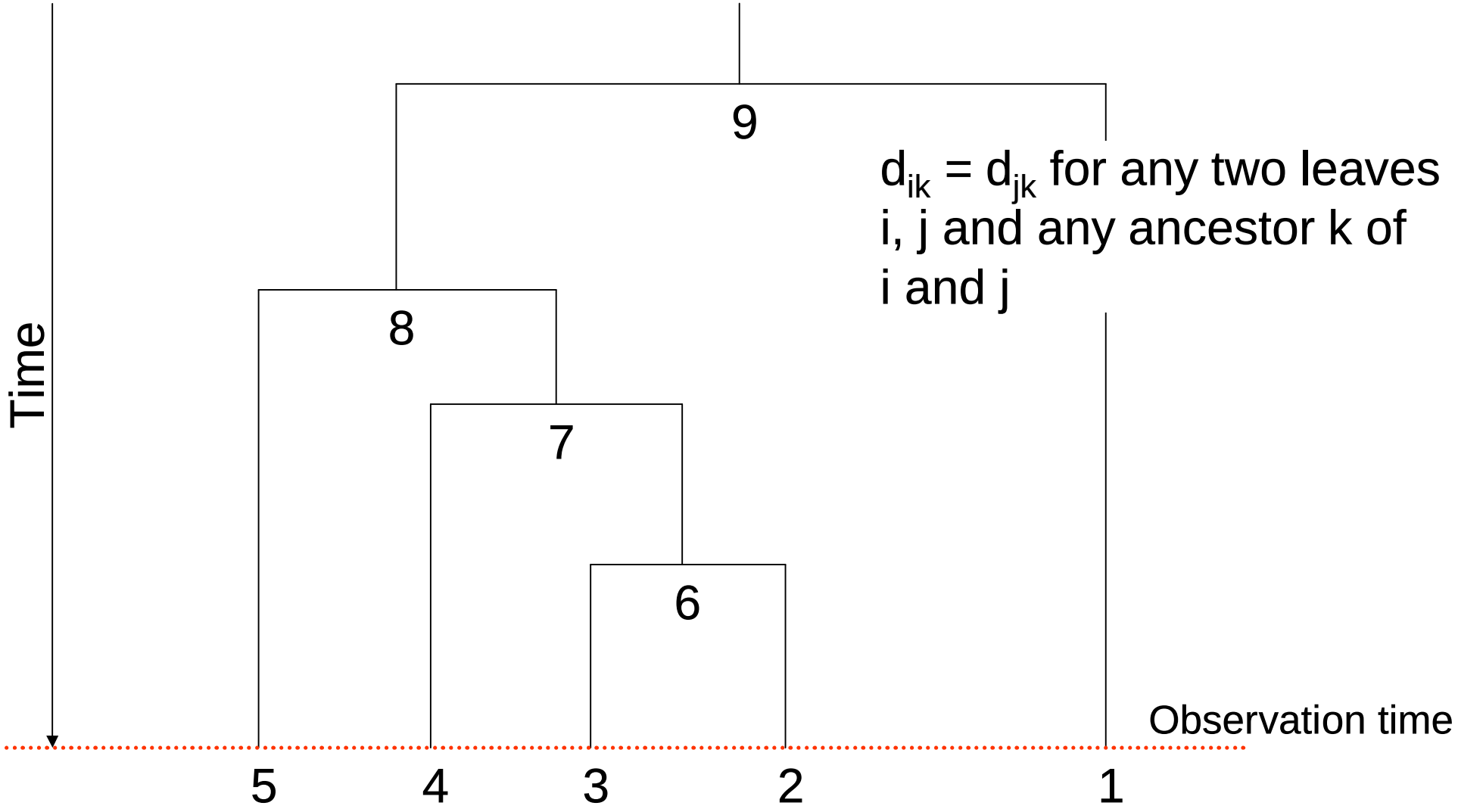
Ultrametric trees



Only vertical segments of the tree have correspondence to some distance d_{ij}:

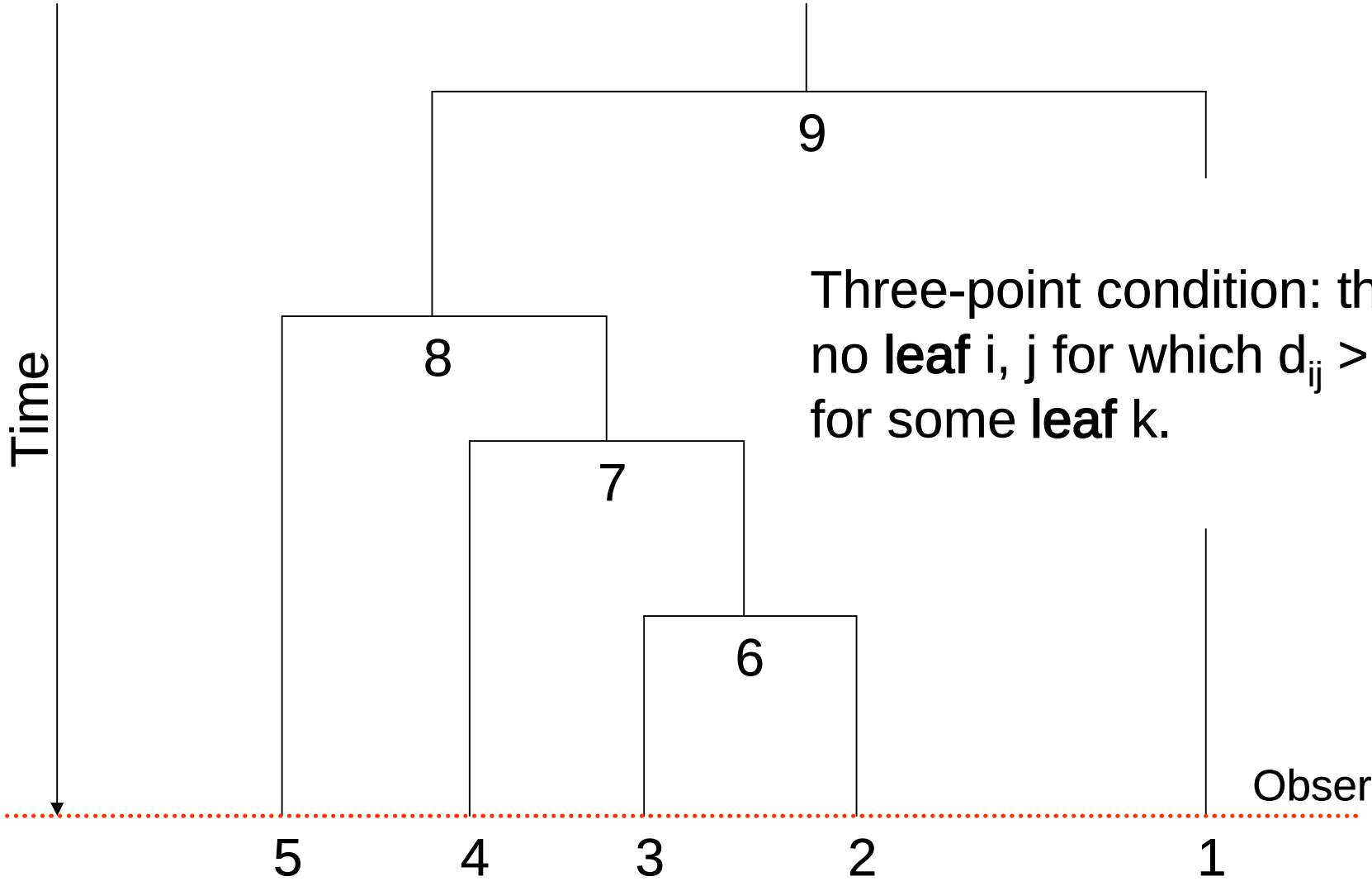
Horizontal segments act as connectors.

Ultrametric trees



$d_{ik} = d_{jk}$ for any two leaves i, j and any ancestor k of i and j

Ultrametric trees



Three-point condition: there exists no **leaf** i, j for which $d_{ij} > \max(d_{ik}, d_{jk})$ for some **leaf** k.

UPGMA algorithm

- UPGMA (unweighted pair group method using arithmetic averages) constructs a phylogenetic tree via clustering
- The algorithm works by at the same time
  - Merging two clusters
  - Creating a new node on the tree
- The tree is built from leaves towards the root
- UPGMA produces a ultrametric tree

Cluster distances

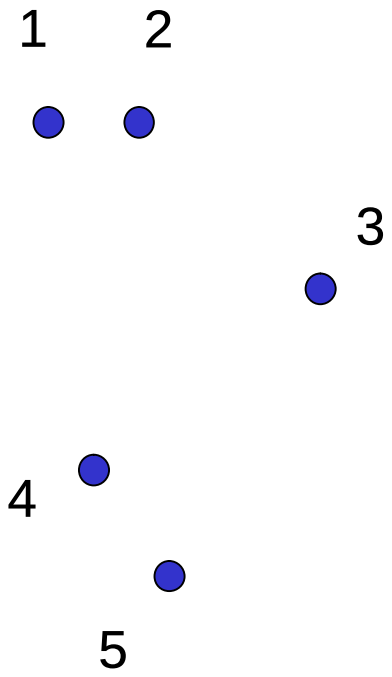
- Let distance d_{ij} between clusters C_i and C_j be

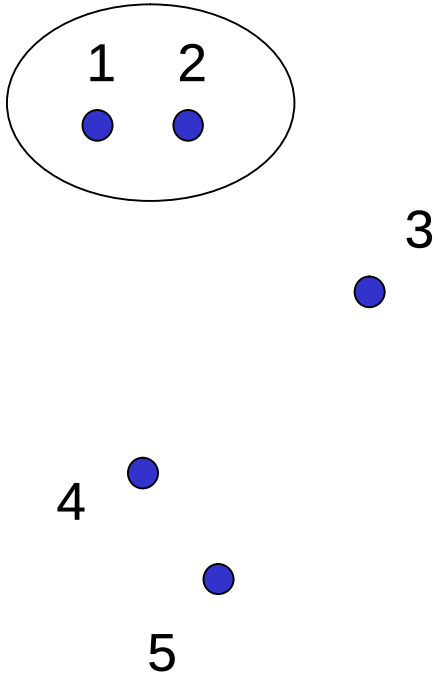
$$d_{ij} = \frac{1}{|C_i||C_j|} \sum_{p \in C_i, q \in C_j} d_{pq}$$

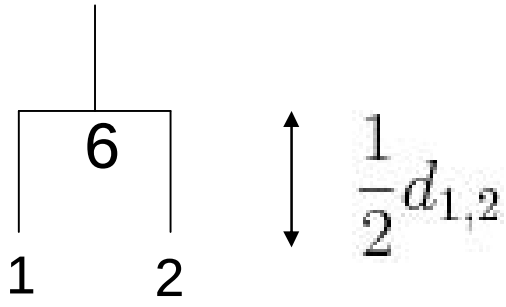
that is, the average distance between points (species) in the cluster.

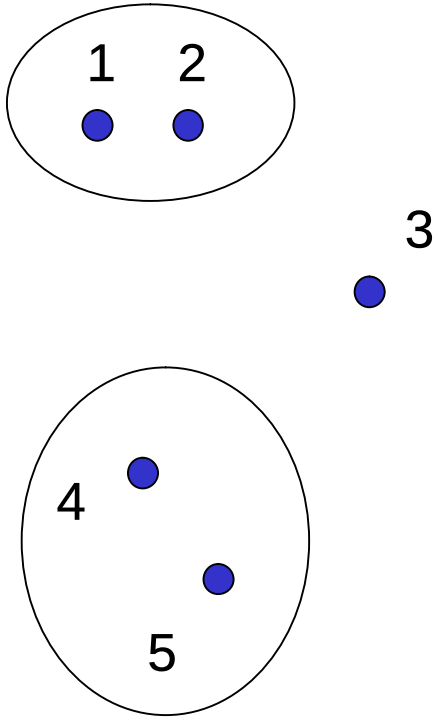
UPGMA algorithm

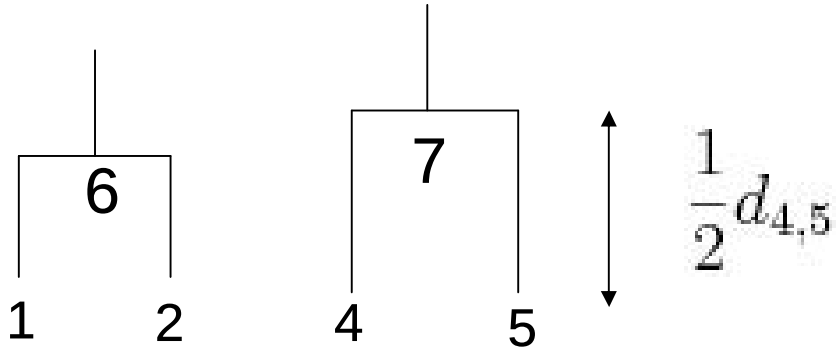
- Initialisation
  - Assign each point i to its own cluster C_i
  - Define one leaf for each sequence, and place it at height zero
- Iteration
  - Find clusters i and j for which d_{ij} is minimal
  - Define new cluster k by $C_k = C_i \cup C_j$, and define d_{kl} for all l
  - Define a node k with children i and j. Place k at height $d_{ij}/2$
  - Remove clusters i and j
- Termination:
  - When only two clusters i and j remain, place root at height $d_{ij}/2$

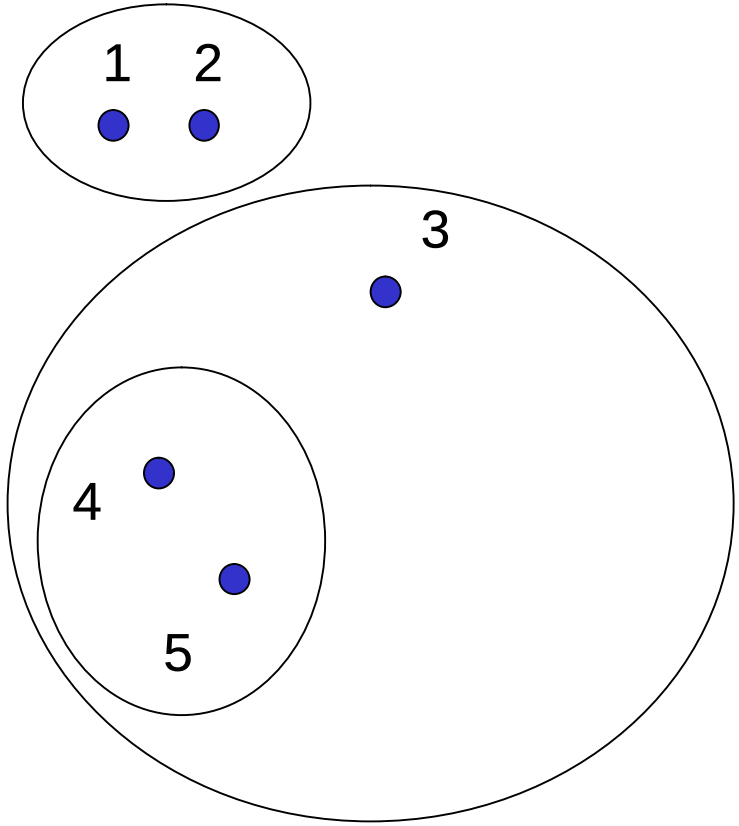

1
2
3
4
5

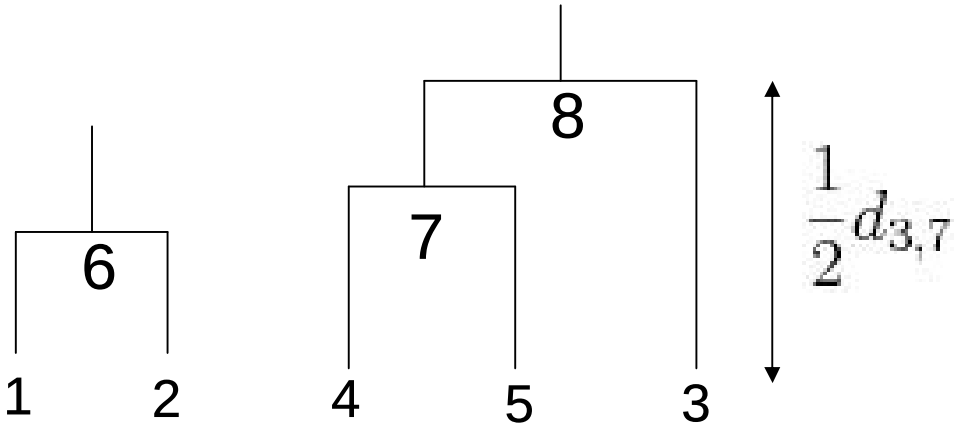

1
2
3
4
5

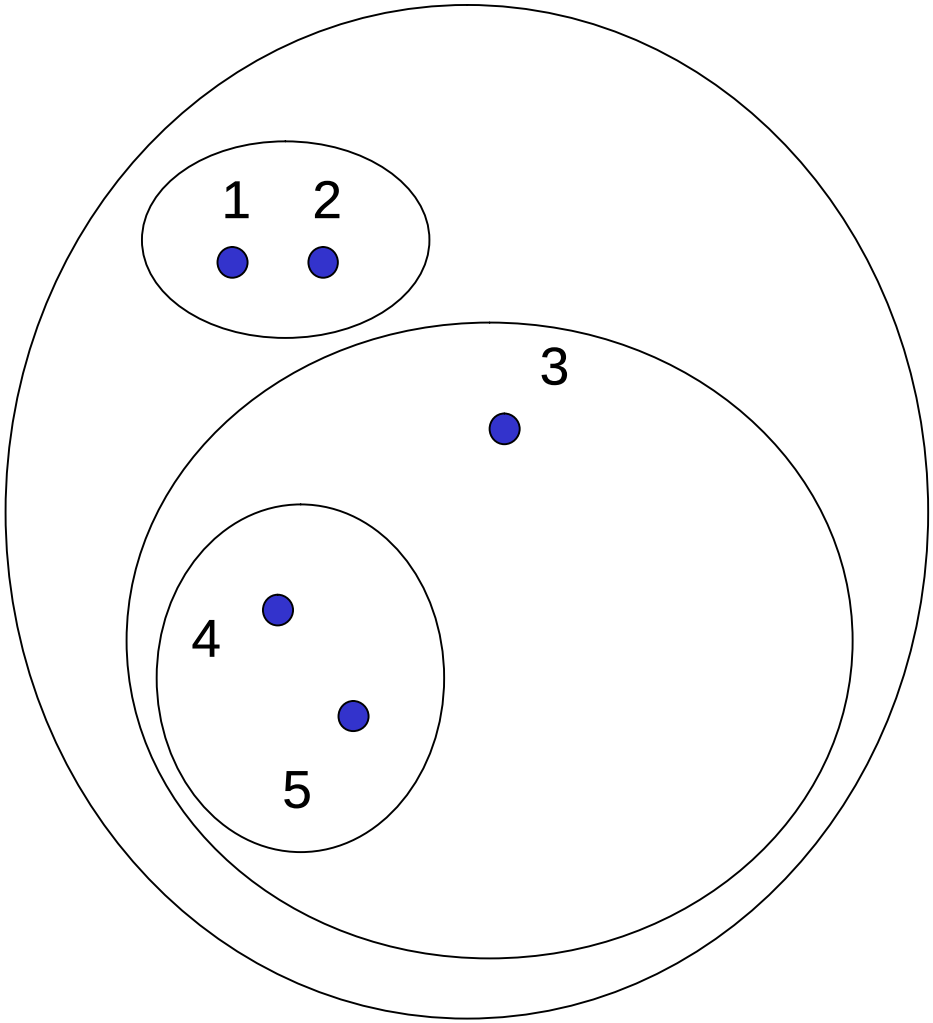

6
1
2
$\frac{1}{2}d_{1,2}$

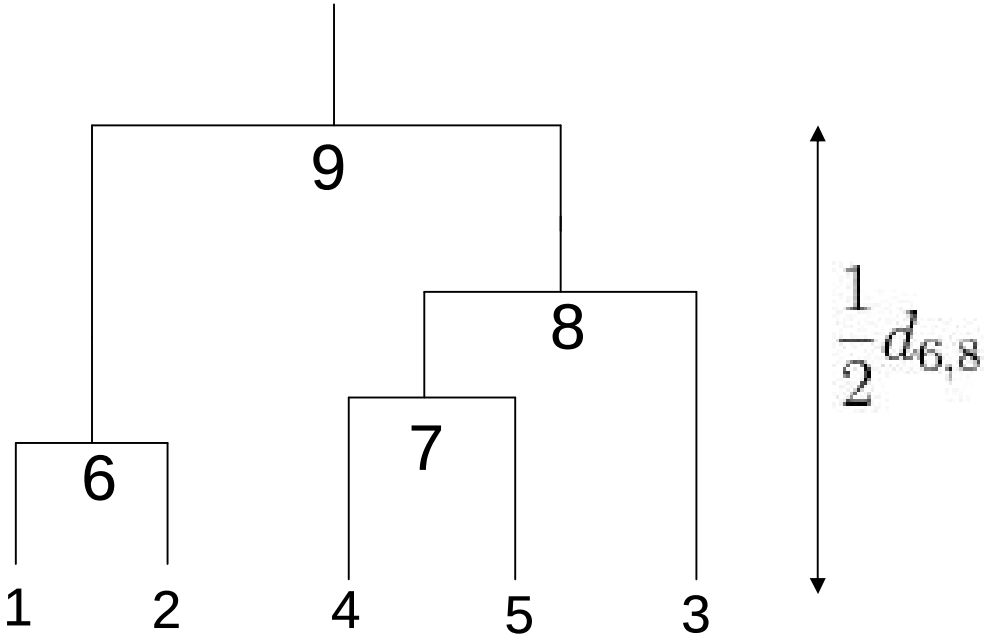

1
2
3
4
5


6
7
1
2
4
5
$\frac{1}{2}d_{4,5}$


1
2
3
4
5


6
1
2
8
7
4
5
3
$\frac{1}{2}d_{3,7}$


1
2
3
4
5


9
8
7
6
1
2
4
5
3
$\frac{1}{2}d_{6,8}$

UPGMA implementation

- In naive implementation, each iteration takes $O(n^2)$ time with n sequences => algorithm takes $O(n^3)$ time

- The algorithm can be implemented to take only $O(n^2)$ time (Gronau & Moran, 2006)

Problem solved?

- We now have a simple algorithm which finds a ultrametric tree
  - If the data is ultrametric, then there is exactly one ultrametric tree corresponding to the data (we skip the proof)
  - The tree found is then the ”correct” solution to the phylogeny problem, if the assumptions hold
- Unfortunately, the data is not ultrametric in practice
  - Measurement errors distort distances
  - *Basic assumption of a molecular clock does not hold usually very well*