Quick plug

Dr. DeBlasio is giving a talk at the Biological Sciences Seminar

- Friday 11 October 2019, 12:30-1:30
- Biological Sciences Research Building, Room 2.168

Toward building an automated bioinformatician: parameter advising for improved scientific discovery

Modern scientific software has a large number of tunable parameters that need to be adjusted to ensure computational performance and accuracy of the results. When these parameter choices are made incorrectly we may overlook significant results or falsely report insignificant ones. Optimizing the parameter choices for one input may not provide an assignment that's good for another, so this parameter optimization process typically needs to be repeated for each new piece of data. Standard machine learning methods for solving this problem need to repeatedly run the software which may not be suitable in practice. Because of the time consumption required to optimize parameters and the possible loss of accuracy that can result when chosen incorrectly, the default parameter vector that are provided by the tool developer is often used. These defaults are designed to work well on average, but most interesting cases are rarely “average”.

In this talk, I will describe my first steps in automatically learning the correct program configuration for biological applications using a framework we call “Parameter Advising”. To apply this framework to the problem of multiple sequence alignment we developed an accuracy estimator, called Facet, to help choose alignments since no ground truth is available in practice. When we use Facet for advising on the Opal aligner we boost accuracy by 14.6% on the hardest-to-align benchmarks. For the reference-based transcript assembly problem, when applying parameter advising to the Scallop assembler we see an increase in accuracy of 28.9%. The framework is general and can be extended to other problems in computational biology and beyond. I will discuss possible areas where parameter advising could be used to automatically learn to run complex analysis software

Office Hours

Do my current office hours work for everyone?

Would a different time be better?

- move Wednesday to Friday?
- move to 4?

Phylogenetics

CS 4390/5390

Plan

Background

Models

- Ultrametric
- Additive-distance
- Parsimony

Algorithms

- Neighbor Joining
- Maximum-parsimony

Why?

Evolution theory says all existing organisms are derived from the same common ancestor, and new species arise by splitting one population into to (or more) pieces that don't cross-breed.

As computer scientists, this means that we (should be able to) represent evolution as a rooted tree with all exigent species as leaves.

"... the great Tree of Life fills with its dead and broken branches the crust of the earth, and covers the surface with its ever-branching and beautiful ramifications" -Darwin, The Origin of Species

This view can be seen at many scales:

From Kishony Lab at Harvard Med School

https://www.youtube.com/watch?v=plVk4NVIUh8

Why?

Evolution theory says all existing organisms are derived from the same common ancestor, and new species arise by splitting one population into to (or more) pieces that don't cross-breed.

As computer scientists, this means that we (should be able to) represent evolution as a rooted tree with all exigent species as leaves.

"... the great Tree of Life fills with its dead and broken branches the crust of the earth, and covers the surface with its ever-branching and beautiful ramifications" -Darwin, The Origin of Species

This view can be seen at many scales:

From Kishony Lab at Harvard Med School

https://www.youtube.com/watch?v=plVk4NVIUh8

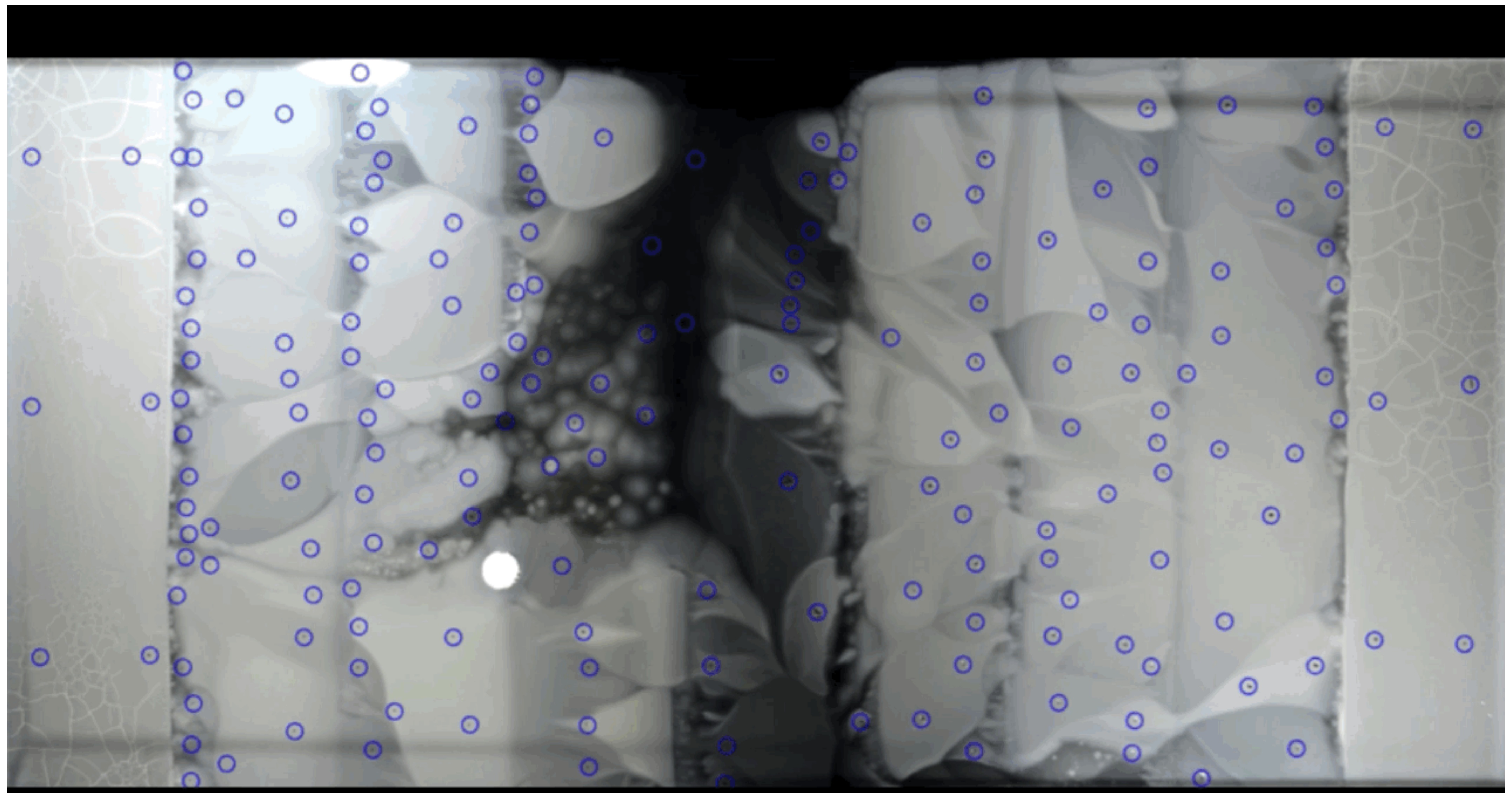
Why?

Evolution theory says all existing organisms are derived from the same common ancestor, and new species arise by splitting one population into to (or more) pieces that don't cross-breed.

As computer scientists, this means that we (should be able to) represent evolution as a rooted tree with all exigent species as leaves.

"... the great Tree of Life fills with its dead and broken branches the crust of the earth, and covers the surface with its ever-branching and beautiful ramifications" -Darwin, The Origin of Species

This view can be seen at many scales:



From Kishony Lab at Harvard Med School

https://www.youtube.com/watch?v=plVk4NVIUh8

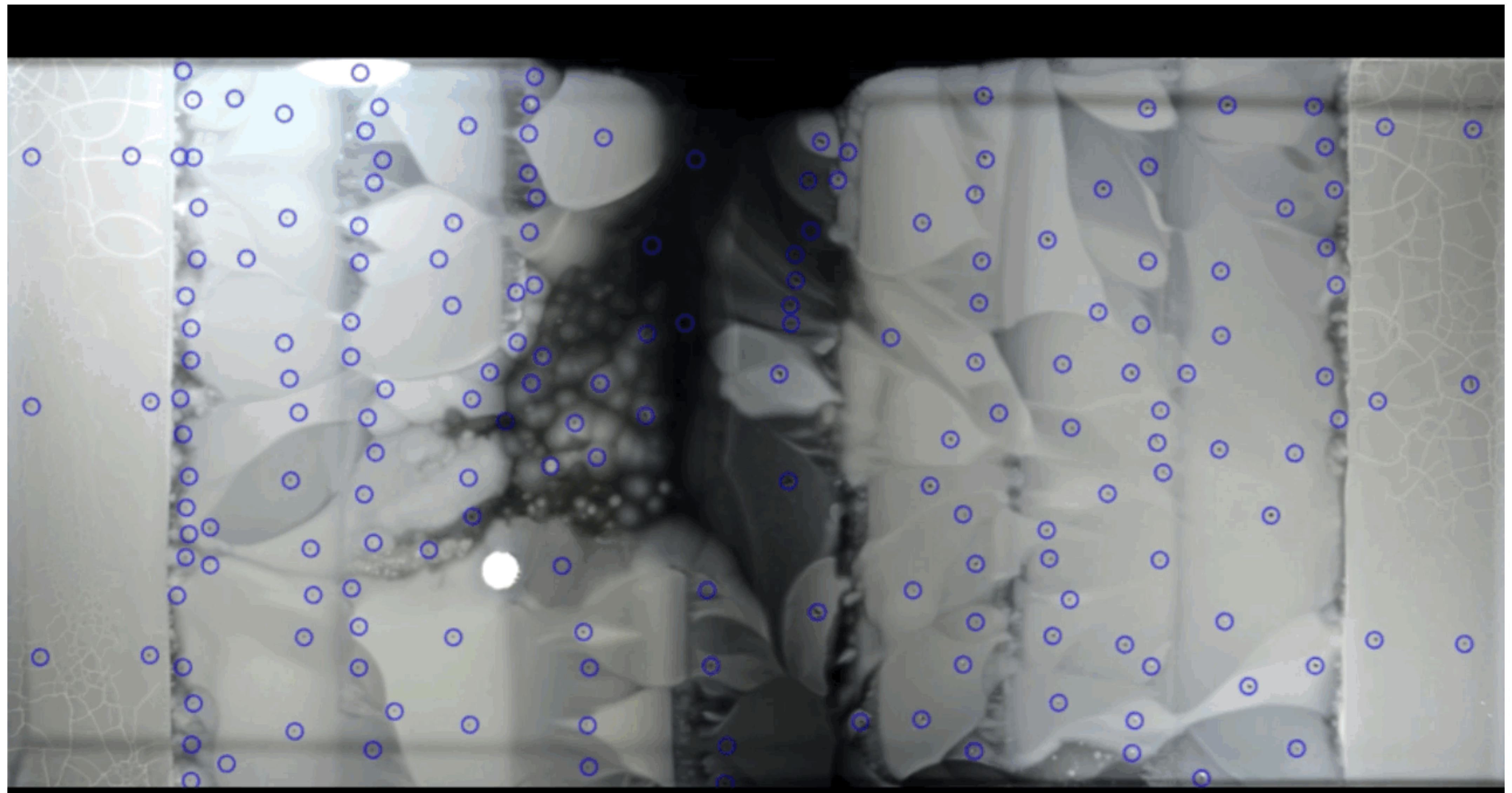
Why?

Evolution theory says all existing organisms are derived from the same common ancestor, and new species arise by splitting one population into to (or more) pieces that don't cross-breed.

As computer scientists, this means that we (should be able to) represent evolution as a rooted tree with all exigent species as leaves.

"... the great Tree of Life fills with its dead and broken branches the crust of the earth, and covers the surface with its ever-branching and beautiful ramifications" -Darwin, The Origin of Species

This view can be seen at many scales:

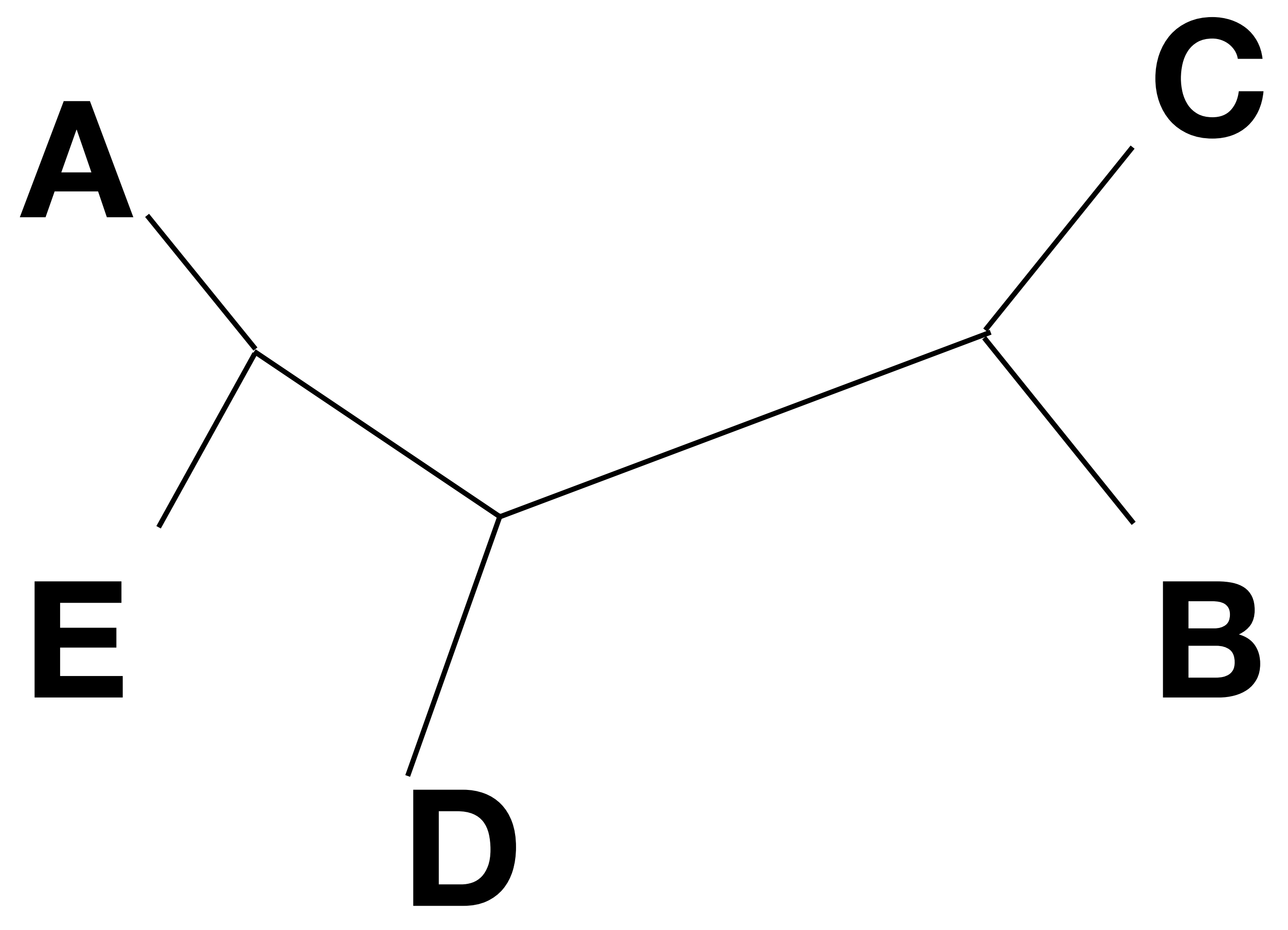


From Kishony Lab at Harvard Med School

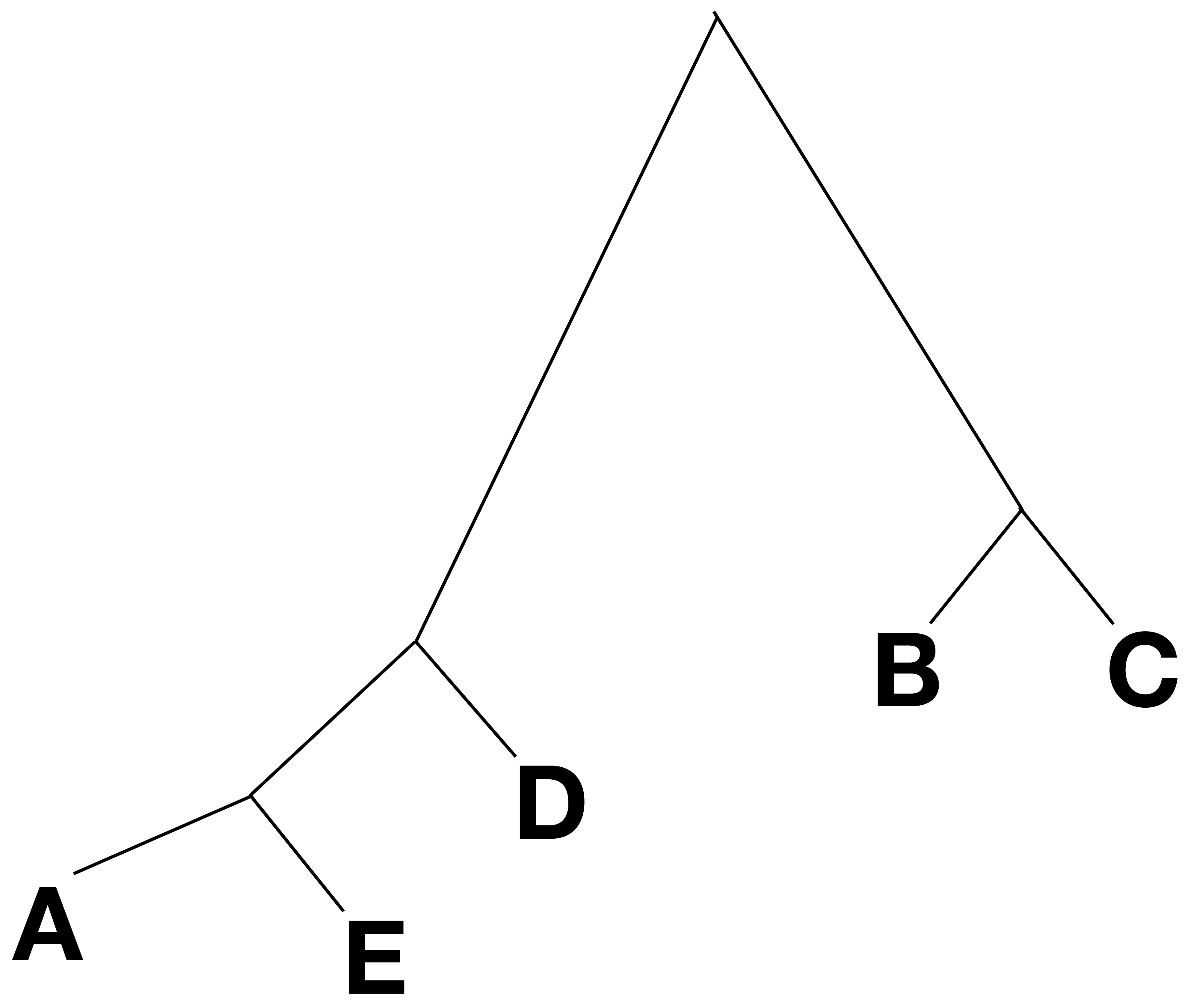
https://www.youtube.com/watch?v=plVk4NVIUh8

Some terminology

Unrooted



Rooted



Biological Methods/Controversy

Three major methods used "historically":

- evolutionary taxonomy
- phenetics (numerical taxonomy)
- cladistics.

Argument over which one is best.

This would determine the "ground truth" trees, or how to compare computed trees with each other.

Tree Building *Algorithms*

Two major classes:

- **Distance-based methods**
  - for each pair of items, get some evolutionary distance (edit distance, melting temp for DNA hybridization, strength of antibody cross reactions)
  - find a tree that "agrees" with the distances either ultametric or additive
  - most cases in real life don't match this so you have to find a good approx.
- **Maximum-Parsimony methods**
  - character-based data only (not necessarily DNA/RNA/Protein data)
  - infer sequences at the internal nodes and maximize parsimony (minimize the mutations) along branches

How does this relate?

The distance-based methods typically use distances derived from some sort of sequence alignment method.

- This is embedded in the algorithms we will present
- In most cases the choice of such a distance is arbitrary, so it won't be specified

What will be presented is an idealized combinatorial optimization solution, rather than being realistic and practical, but the ideas are the same with some modification.

Ultrametric Trees

Let *D* be a symmetric *n*x*n* matrix of real numbers. An *ultrametric* tree for *D* is a rooted tree *T* such that:

Ultrametric Trees

Let *D* be a symmetric *n*x*n* matrix of real numbers. An *ultrametric* tree for *D* is a rooted tree *T* such that:

- *T* contains *n* leaves labeled by a unique row of *D*.

Ultrametric Trees

Let *D* be a symmetric *n*x*n* matrix of real numbers. An *ultrametric* tree for *D* is a rooted tree *T* such that:

- *T* contains *n* leaves labeled by a unique row of *D*.
- Each internal node of *T* is leveled by one **entry** from *D* and has at least 2 children.

Ultrametric Trees

Let *D* be a symmetric *n*x*n* matrix of real numbers. An *ultrametric* tree for *D* is a rooted tree *T* such that:

- *T* contains *n* leaves labeled by a unique row of *D*.
- Each internal node of *T* is leveled by one **entry** from *D* and has at least 2 children.
- Along any path from the root to a leaf, the numbers labeling the internal nodes are **strictly decreasing**.

Ultrametric Trees

Let *D* be a symmetric *n*x*n* matrix of real numbers. An *ultrametric* tree for *D* is a rooted tree *T* such that:

- *T* contains *n* leaves labeled by a unique row of *D*.
- Each internal node of *T* is leveled by one **entry** from *D* and has at least 2 children.
- Along any path from the root to a leaf, the numbers labeling the internal nodes are **strictly decreasing**.
- For any two leaves *i,j* of *T*, *D(i,j)* is the leavel of the least common ancestor of *i* and *j* in *T.*

Ultrametric Trees

Let *D* be a symmetric *n*x*n* matrix of real numbers. An *ultrametric* tree for *D* is a rooted tree *T* such that:

- *T* contains *n* leaves labeled by a unique row of *D*.
- Each internal node of *T* is leveled by one **entry** from *D* and has at least 2 children.
- Along any path from the root to a leaf, the numbers labeling the internal nodes are **strictly decreasing**.
- For any two leaves *i,j* of *T*, *D(i,j)* is the leavel of the least common ancestor of *i* and *j* in *T.*

Therefore, *T* (if it exists) is a compact representation of *D*

Ultrametric Trees

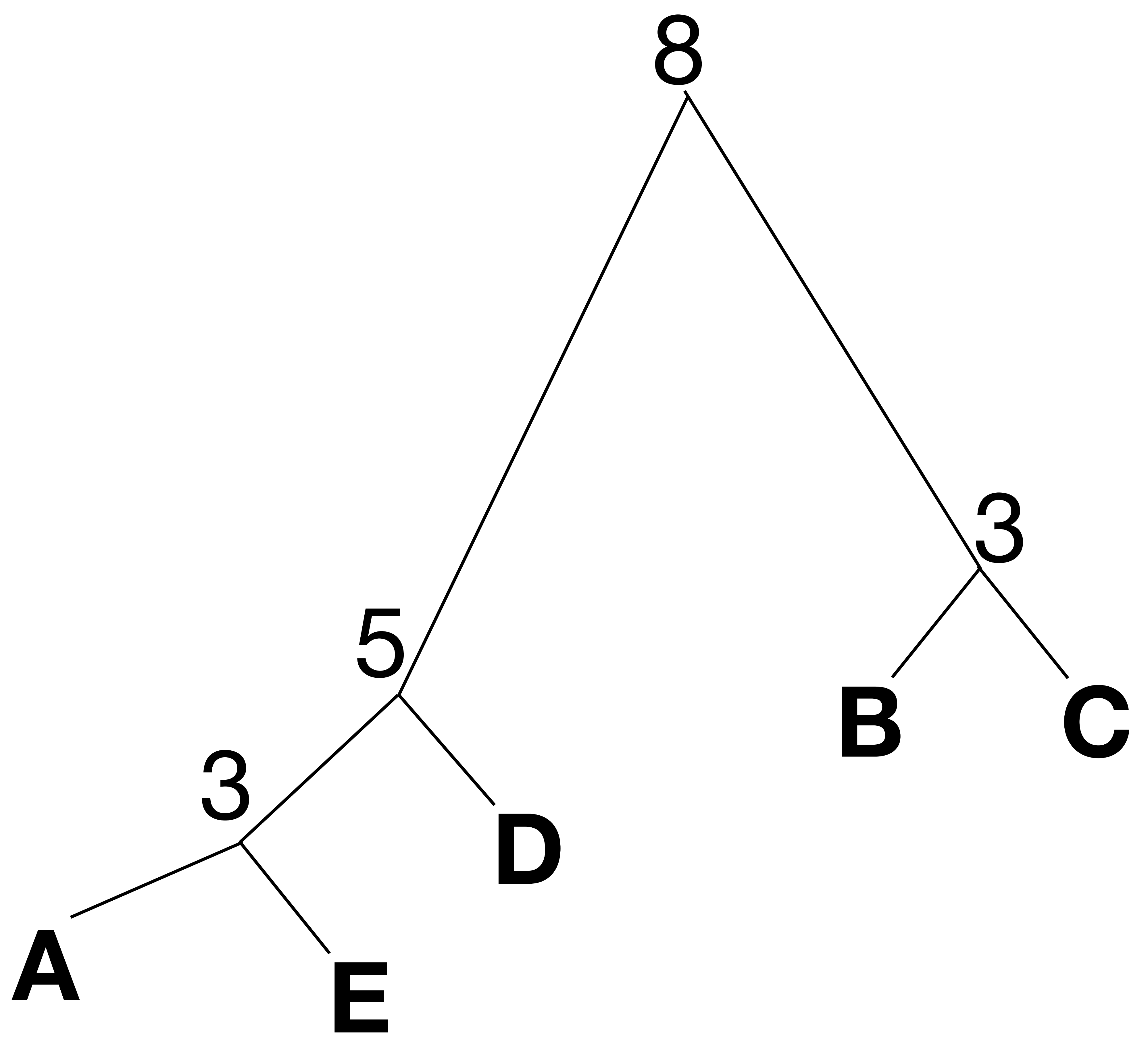
Let *D* be a symmetric *nxn* matrix of real numbers. A **min-***ultrametric* tree for *D* is a rooted tree *T* such that:

- *T* contains *n* leaves labeled by a unique row of *D*.
- Each internal node of *T* is leveled by one **entry** from *D* and has at least 2 children.
- Along any path from the root to a leaf, the numbers labeling the internal nodes are **strictly increasing**.
- For any two leaves *i,j* of *T*, *D(i,j)* is the leavel of the least common ancestor of *i* and *j* in *T.*

Therefore, *T* (if it exists) is a compact representation of *D*

Ultrametric Trees

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



Ultrametric Trees

Do ultrametric trees exist for all *D*?

Ultrametric Trees

Do ultrametric trees exist for all *D*?

- no, since values need to be shared, there can only be so many of them

Ultrametric Trees

Do ultrametric trees exist for all *D*?

- no, since values need to be shared, there can only be so many of them

Whats an easy test to see if an Ultrametric tree *might* exits?

Ultrametric Trees

Do ultrametric trees exist for all *D*?

- no, since values need to be shared, there can only be so many of them

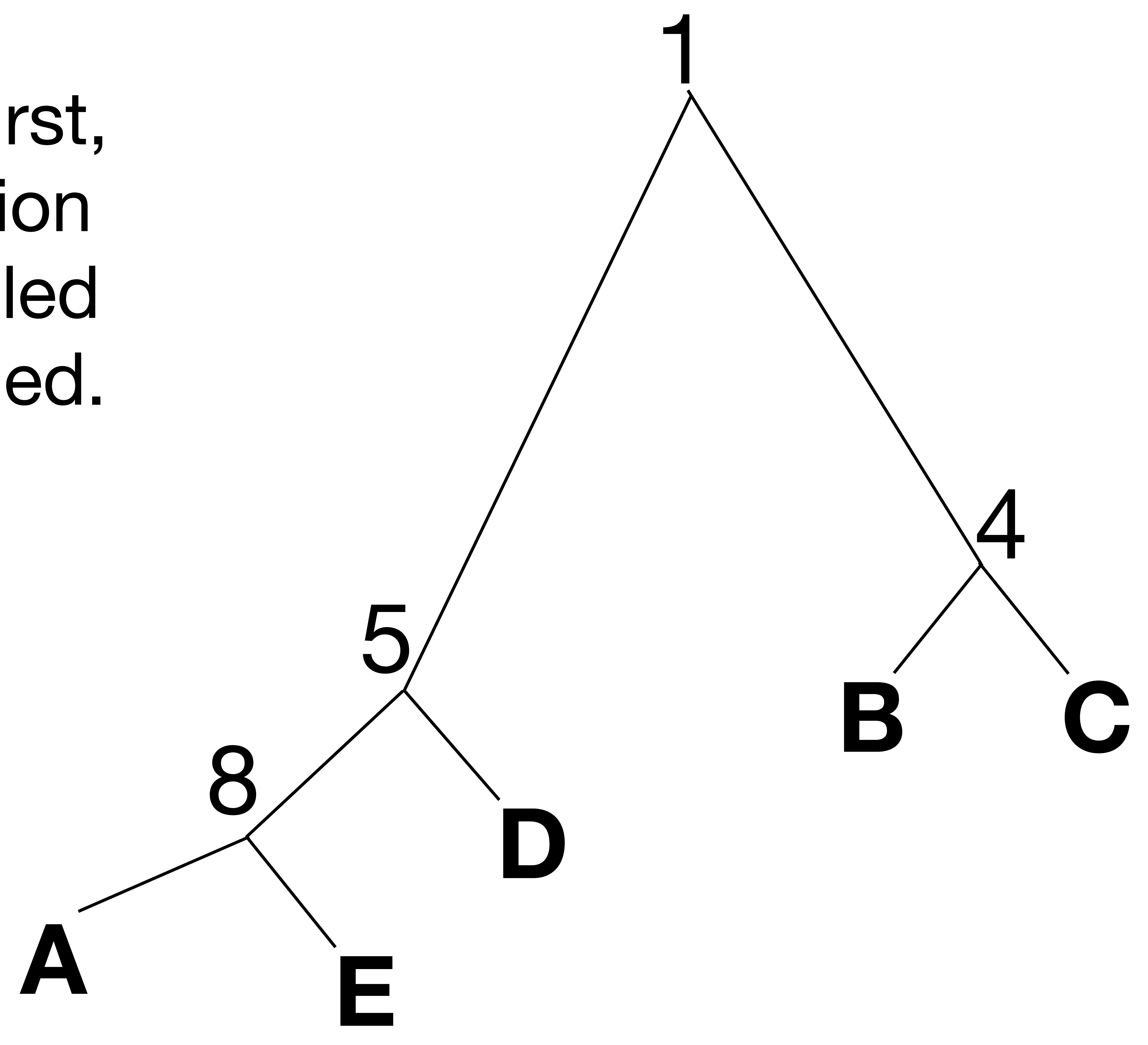
Whats an easy test to see if an Ultrametric tree *might* exits?

- check if the number of distinct values in *D* is less than *n-1* (the maximum number of internal nodes)

Ultrametric Trees

What does it even mean?

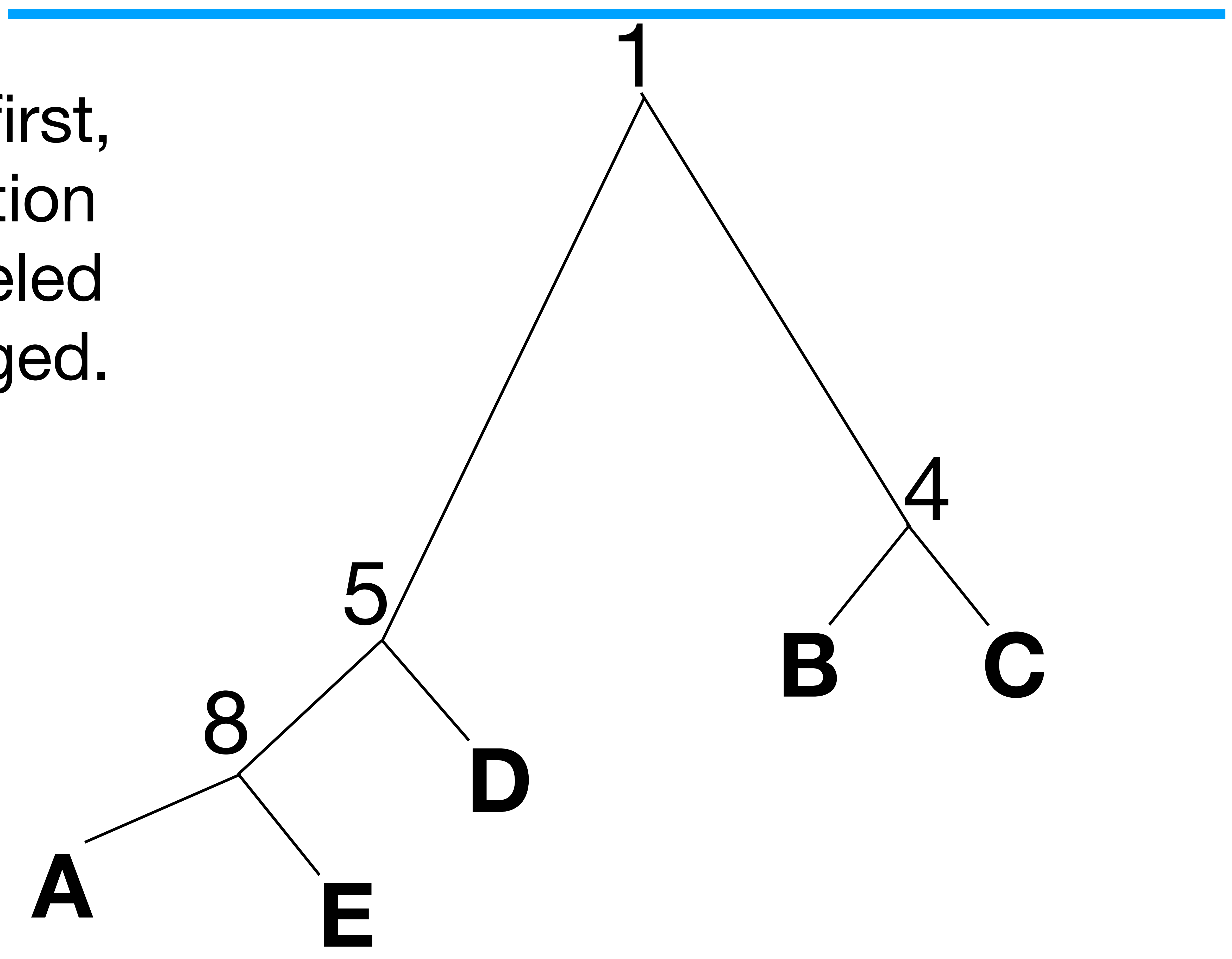
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.



Ultrametric Trees

What does it even mean?

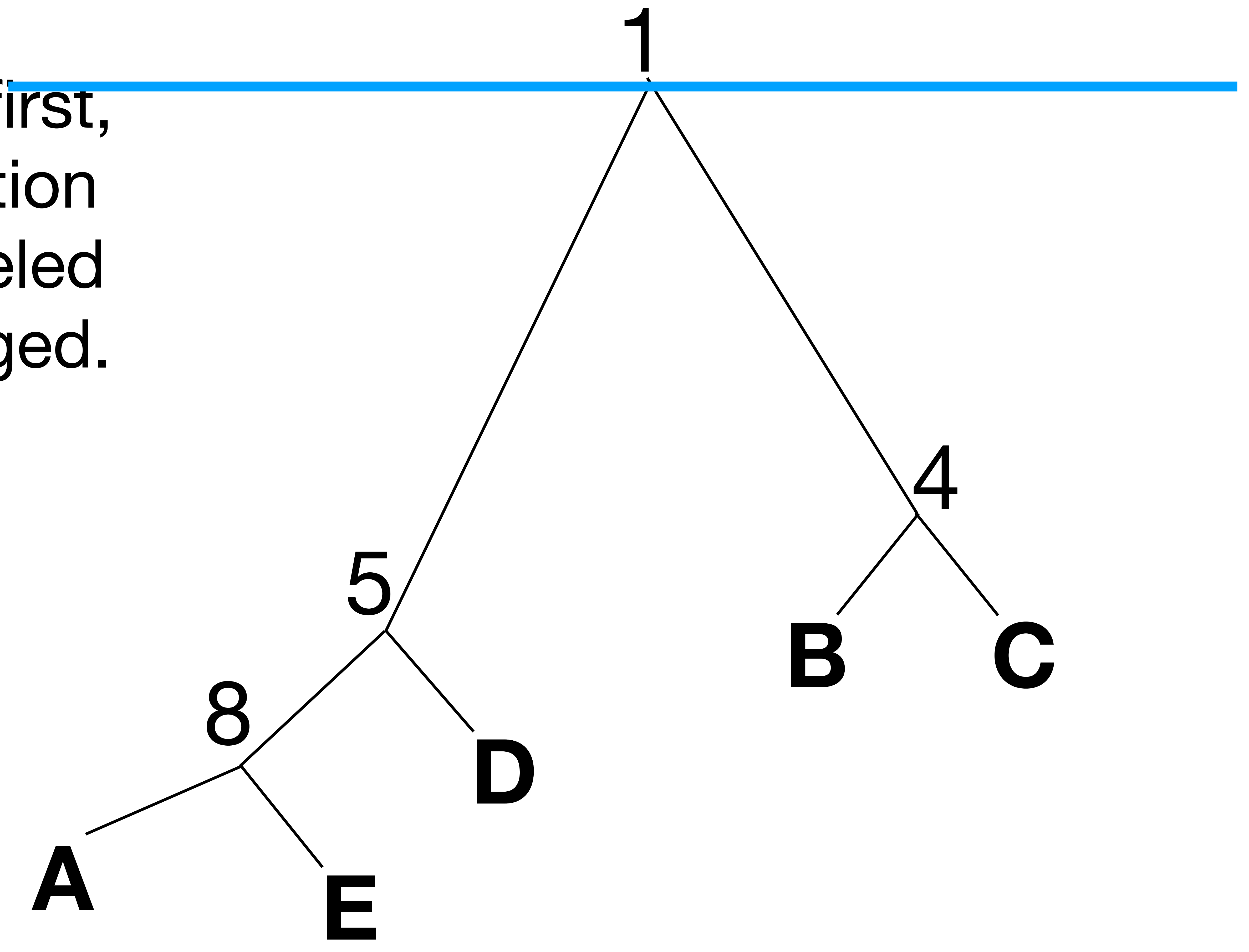
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.



Ultrametric Trees

What does it even mean?

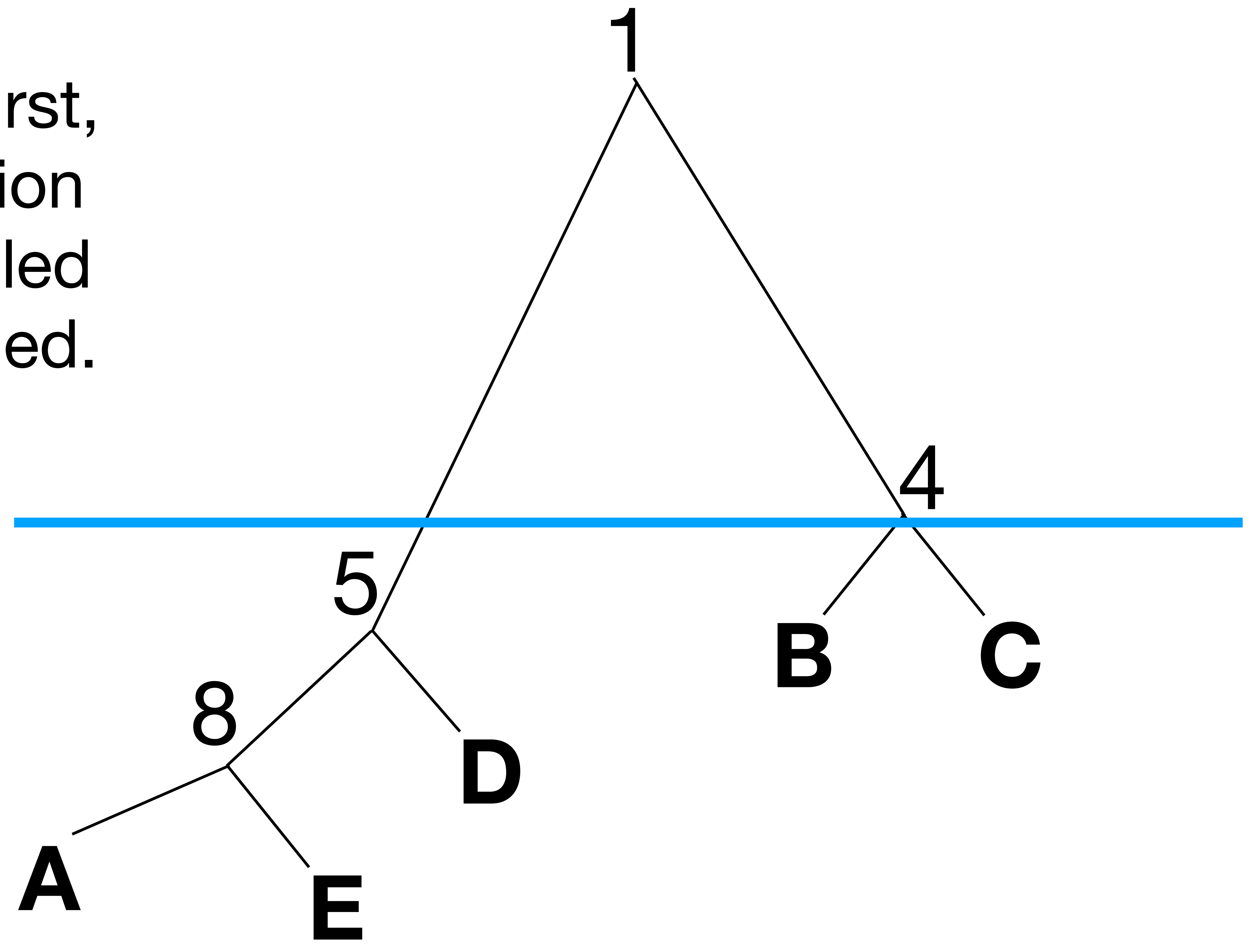
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.



Ultrametric Trees

What does it even mean?

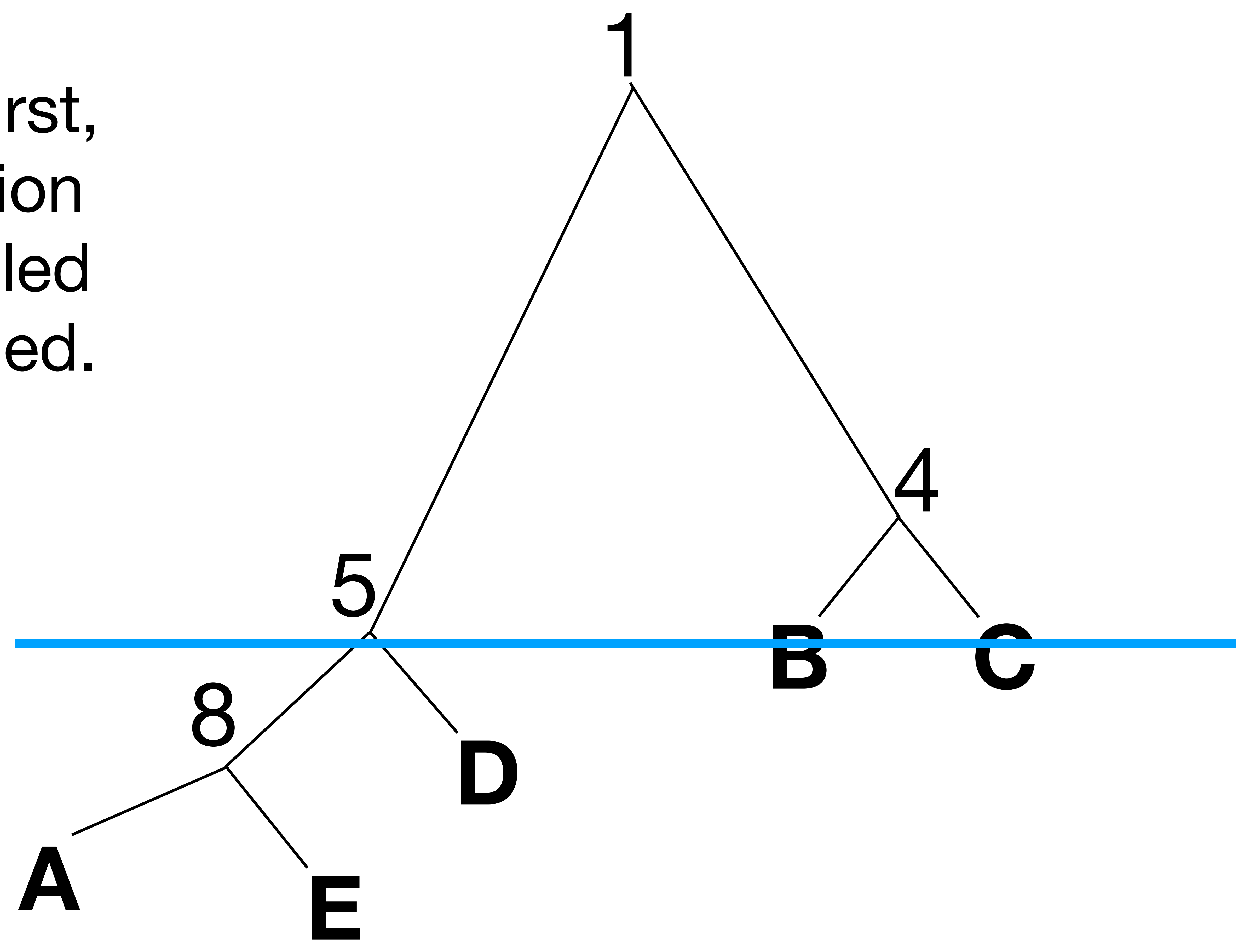
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.



Ultrametric Trees

What does it even mean?

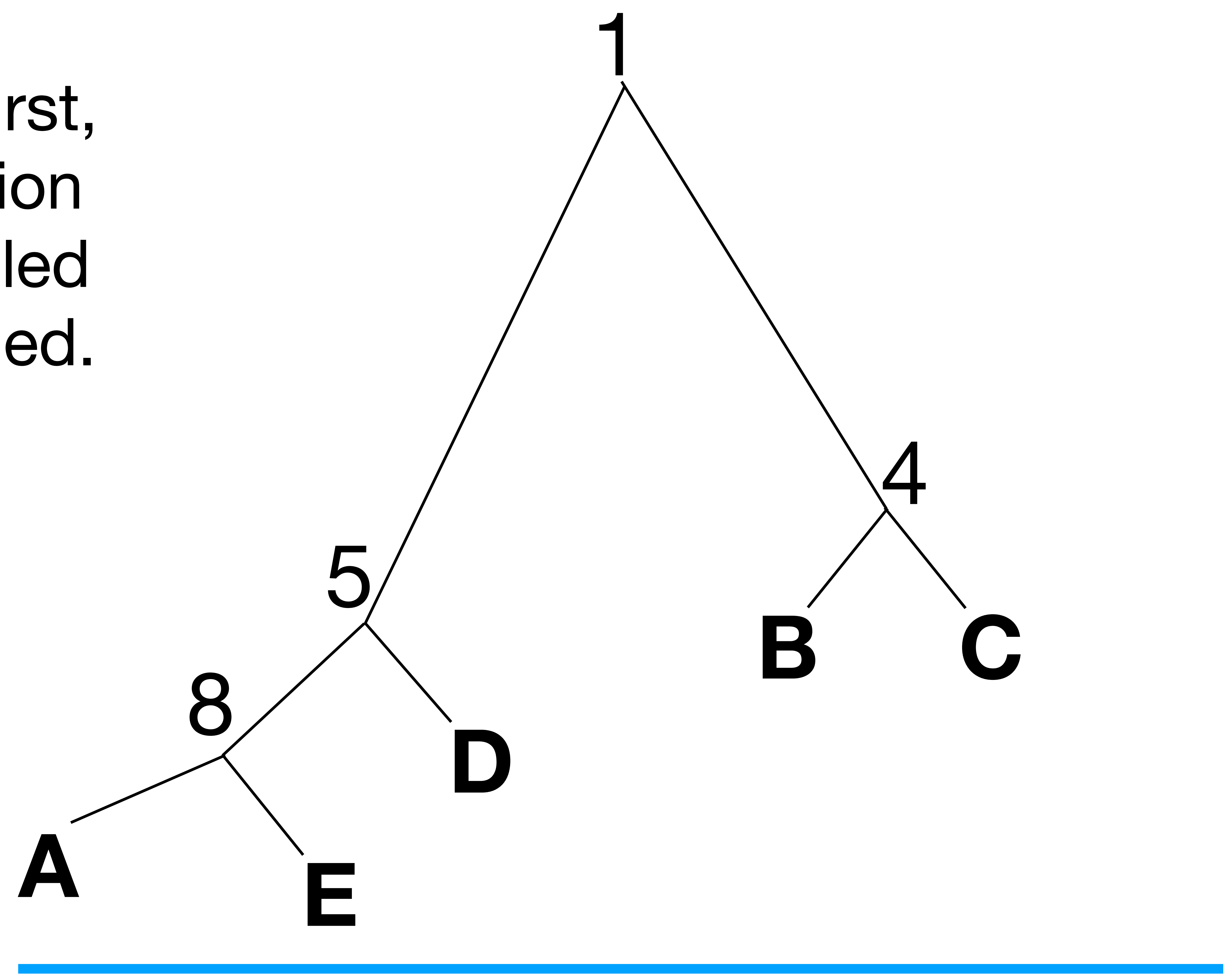
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.



Ultrametric Trees

What does it even mean?

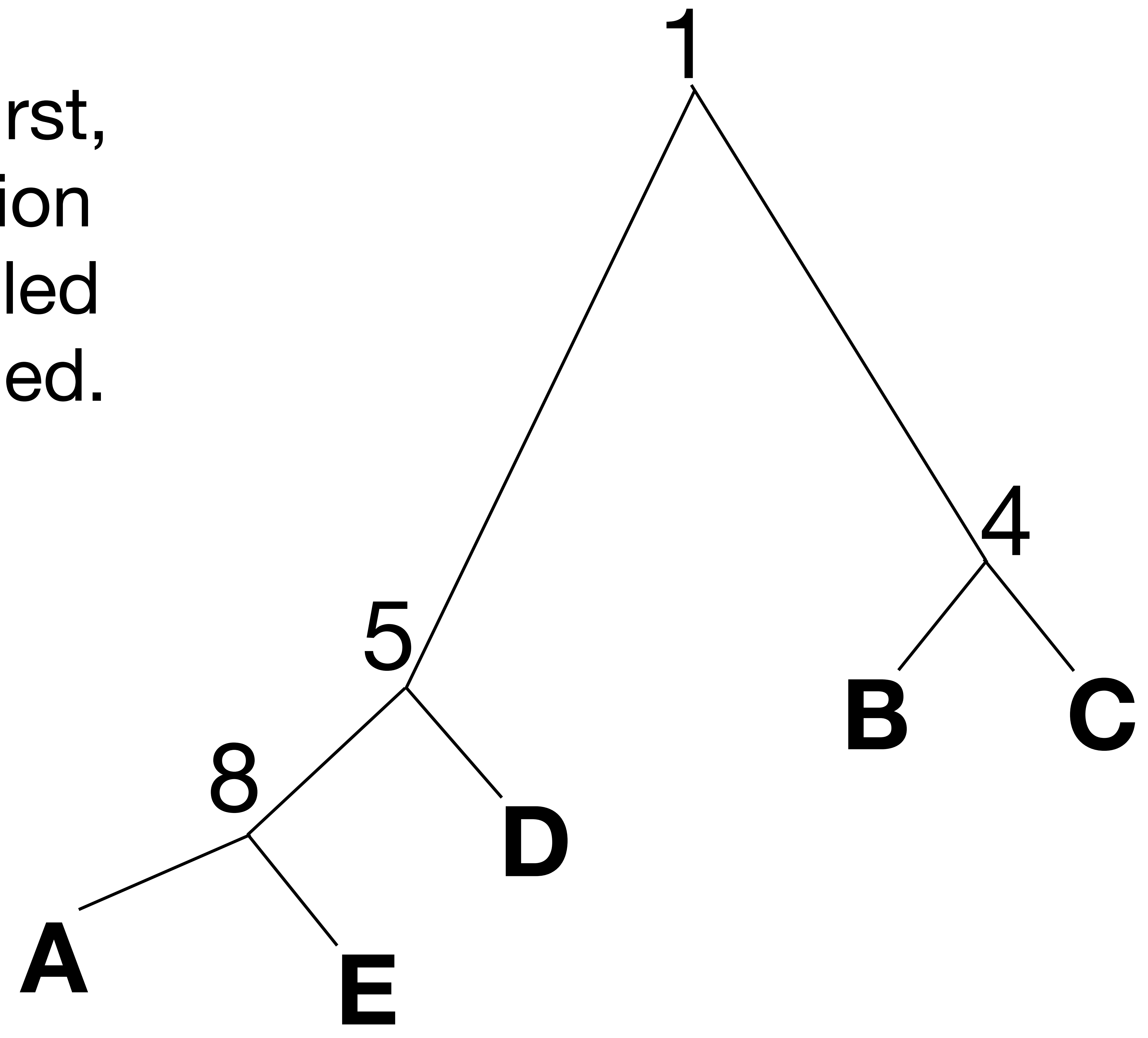
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.



Ultrametric Trees

What does it even mean?

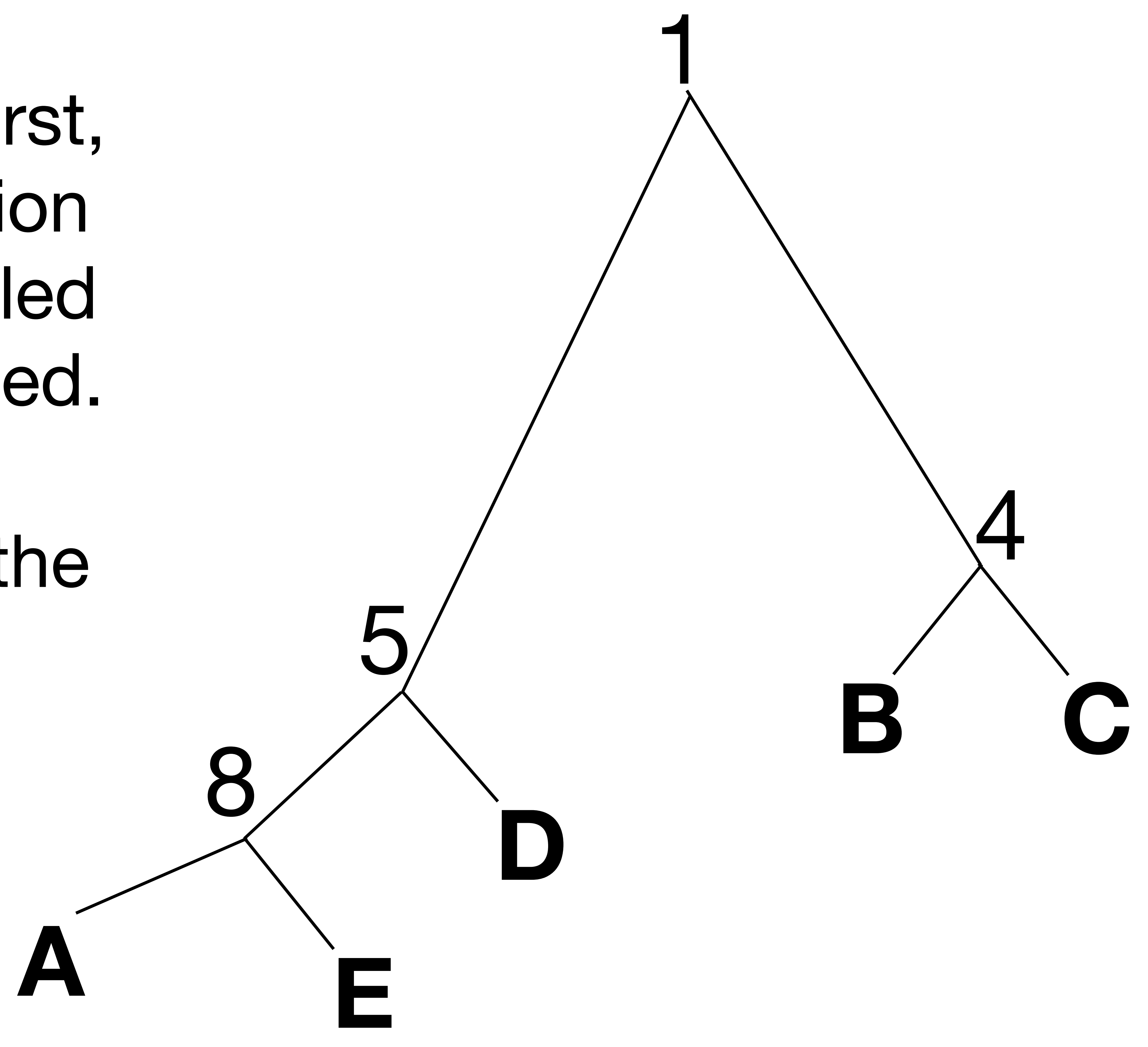
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.



Ultrametric Trees

What does it even mean?

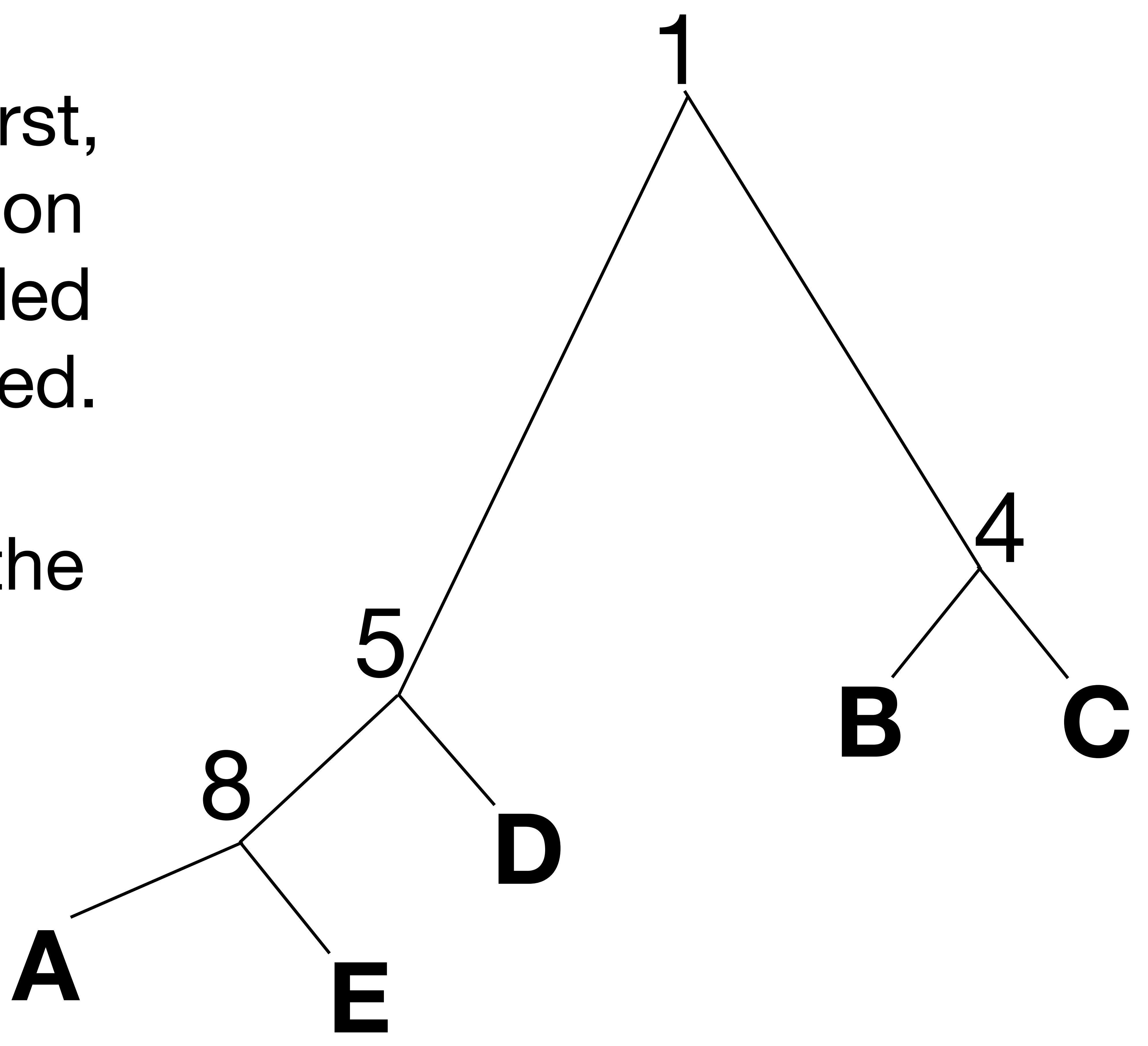
- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.
- For an ultrametric tree, the time is the length of the edges from the node to the leaves (in time)



Ultrametric Trees

What does it even mean?

- Think about the min-ultrametric tree first, then imagine the top to bottom direction being time. Each internal node is labeled by the absolute time the things diverged.
- For an ultrametric tree, the time is the length of the edges from the node to the leaves (in time)
- The difference is that is time ***since*** divergence or time ***of*** divergence



Ultrametric Trees

Is there a way to easily test if a set of distances is ultrametric?

Ultrametric Trees

Is there a way to easily test if a set of distances is ultrametric?

- We know from before that for a matrix *D* to be ultrametric, the number of distinct values as to be fewer than *n-1*.

Ultrametric Trees

Is there a way to easily test if a set of distances is ultrametric?

- We know from before that for a matrix *D* to be ultrametric, the number of distinct values as to be fewer than *n-1*.
- **Definition** A symmetric matrix *D* defines an (min-)*ultametric distance* iff for every three indices *i,j,k*, there is a tie for the maximum (minimum) of *D(i,j), D(j,k)* and *D(i,k).*

Ultrametric Trees

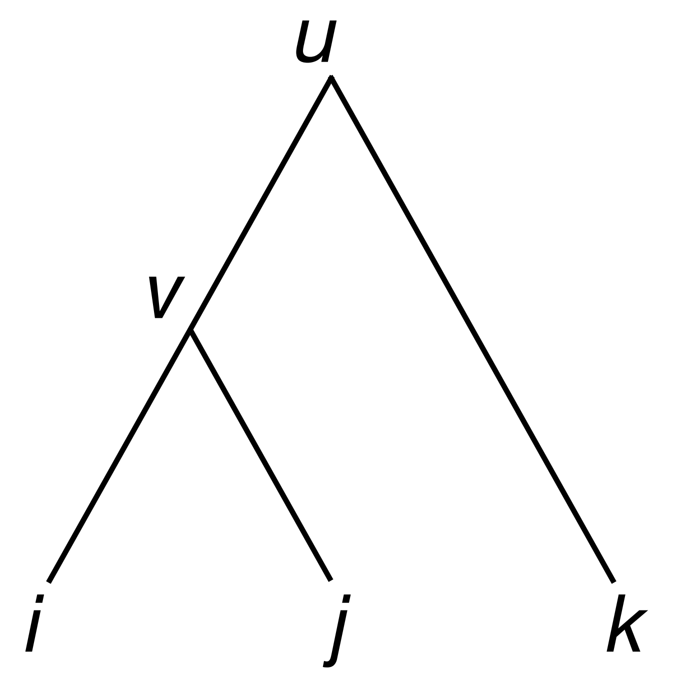
Is there a way to easily test if a set of distances is ultrametric?

- We know from before that for a matrix *D* to be ultrametric, the number of distinct values as to be fewer than *n-1*.
- **Definition** A symmetric matrix *D* defines an (min-)*ultametric distance* iff for every three indices *i,j,k*, there is a tie for the maximum (minimum) of *D(i,j), D(j,k)* and *D(i,k).*
- If a *D* has an ultra metric tree, it is an ultrametric distance.

Ultrametric Trees

Is there a way to easily test if a set of distances is ultrametric?

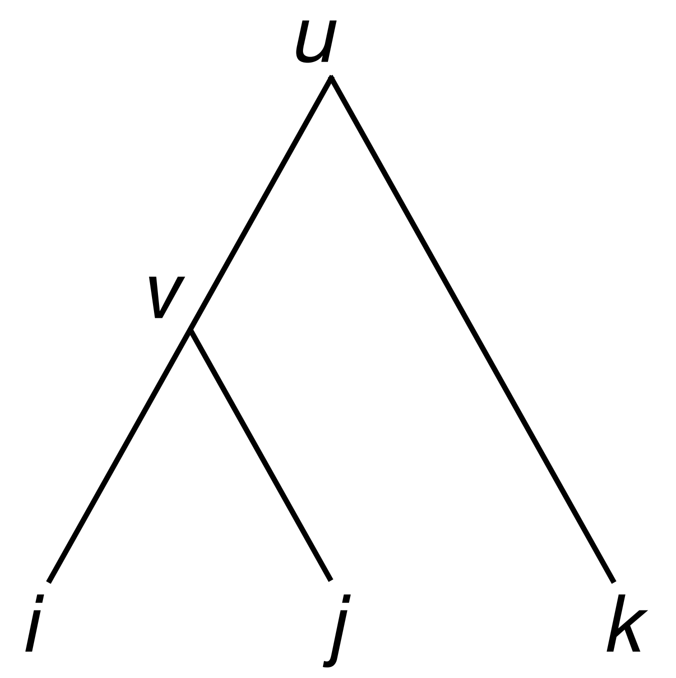
- We know from before that for a matrix *D* to be ultrametric, the number of distinct values as to be fewer than *n-1*.
- **Definition** A symmetric matrix *D* defines an (min-)*ultametric distance* iff for every three indices *i,j,k*, there is a tie for the maximum (minimum) of *D(i,j), D(j,k)* and *D(i,k).*
- If a *D* has an ultra metric tree, it is an ultrametric distance.



Ultrametric Trees

Is there a way to easily test if a set of distances is ultrametric?

- We know from before that for a matrix *D* to be ultrametric, the number of distinct values as to be fewer than *n-1*.
- **Definition** A symmetric matrix *D* defines an (min-)*ultametric distance* iff for every three indices *i,j,k*, there is a tie for the maximum (minimum) of *D(i,j), D(j,k)* and *D(i,k).*
- If a *D* has an ultra metric tree, it is an ultrametric distance.
- What about the converse?



Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Proof The "only-if" part is observed in the figure on the last slide. Prove the "if" by construction:

Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Proof The "only-if" part is observed in the figure on the last slide. Prove the "if" by construction:

- let *i* be an index such that $D(i,i) \neq D(i,j)$ for all $i \neq j$.

Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Proof The "only-if" part is observed in the figure on the last slide. Prove the "if" by construction:

- let *i* be an index such that $D(i,i) \neq D(i,j)$ for all $i \neq j$.
- assume there are *d* distinct values in row *i* of *D,* then the path from the root to the leaf labeled by *i* must pass though exactly *d* nodes, in decreasing order.

Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Proof The "only-if" part is observed in the figure on the last slide. Prove the "if" by construction:

- let *i* be an index such that *D(i,i)* ≠ *D(i,j)* for all *i* ≠ *j.*
- assume there are *d* distinct values in row *i* of *D,* then the path from the root to the leaf labeled by *i* must pass though exactly *d* nodes, in decreasing order.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a	0	4	3	4	5	4	3	4
b								
c								
d								
e								
f								
g								
h								

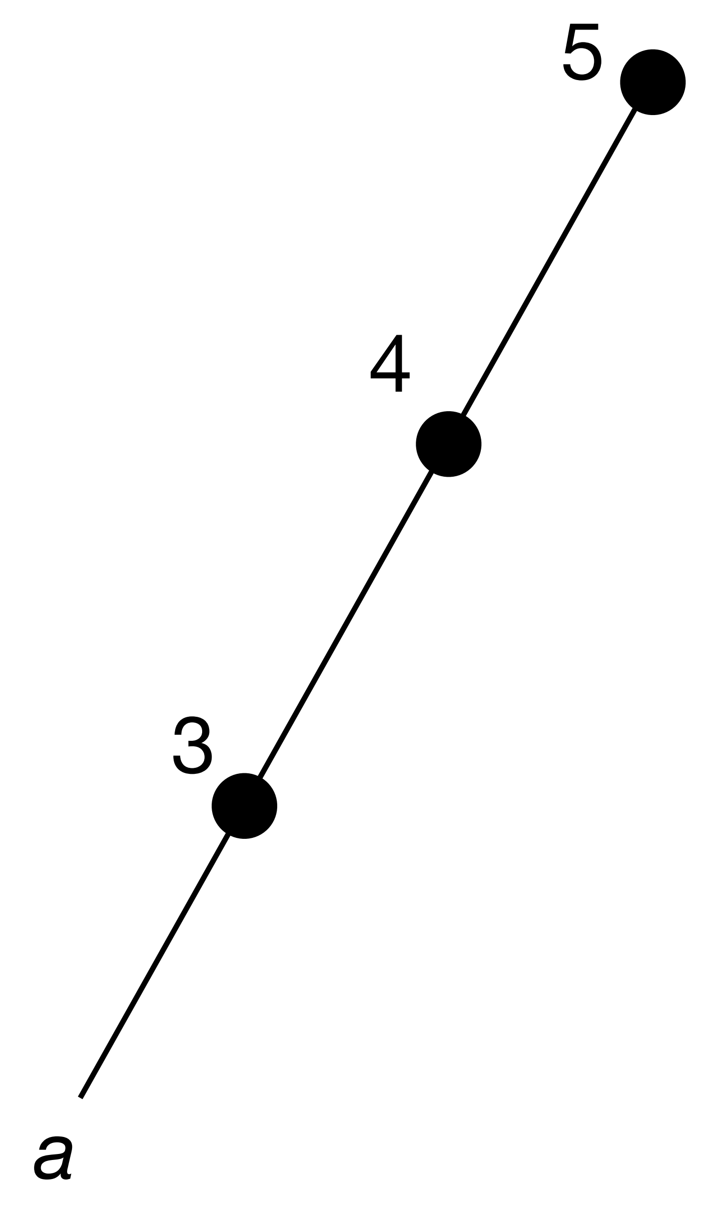
Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Proof The "only-if" part is observed in the figure on the last slide. Prove the "if" by construction:

- let *i* be an index such that *D(i,i)* ≠ *D(i,j)* for all *i* ≠ *j*.
- assume there are *d* distinct values in row *i* of *D*, then the path from the root to the leaf labeled by *i* must pass though exactly *d* nodes, in decreasing order.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a	0	4	3	4	5	4	3	4
b								
c								
d								
e								
f								
g								
h								



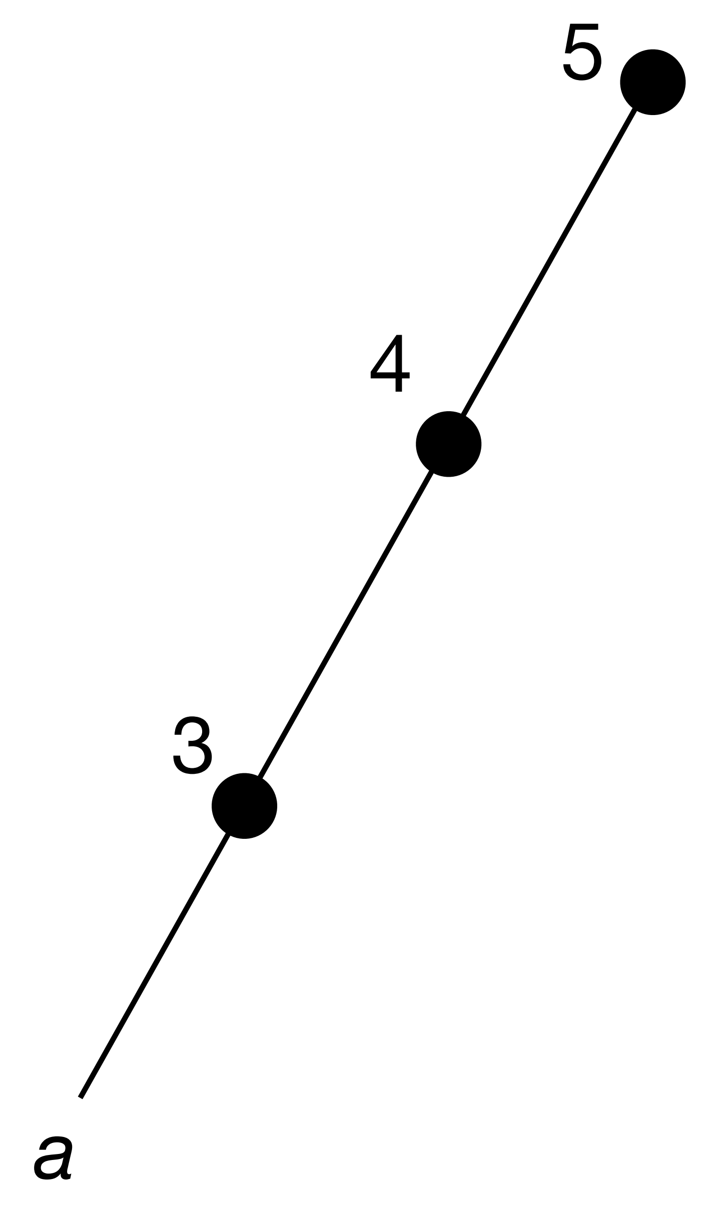
Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Proof The "only-if" part is observed in the figure on the last slide. Prove the "if" by construction:

- let *i* be an index such that *D(i,i)* ≠ *D(i,j)* for all *i* ≠ *j.*
- assume there are *d* distinct values in row *i* of *D,* then the path from the root to the leaf labeled by *i* must pass though exactly *d* nodes, in decreasing order.
- the other children of those internal nodes, have subtrees with leaf labels of the other indices with that value in row *i,* call these groups of leaves (s.t. *D(i,j)* = *D(i,k)*) **classes**, and the set of *d-1* classes a **partitioning**.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a	0	4	3	4	5	4	3	4
b								
c								
d								
e								
f								
g								
h								



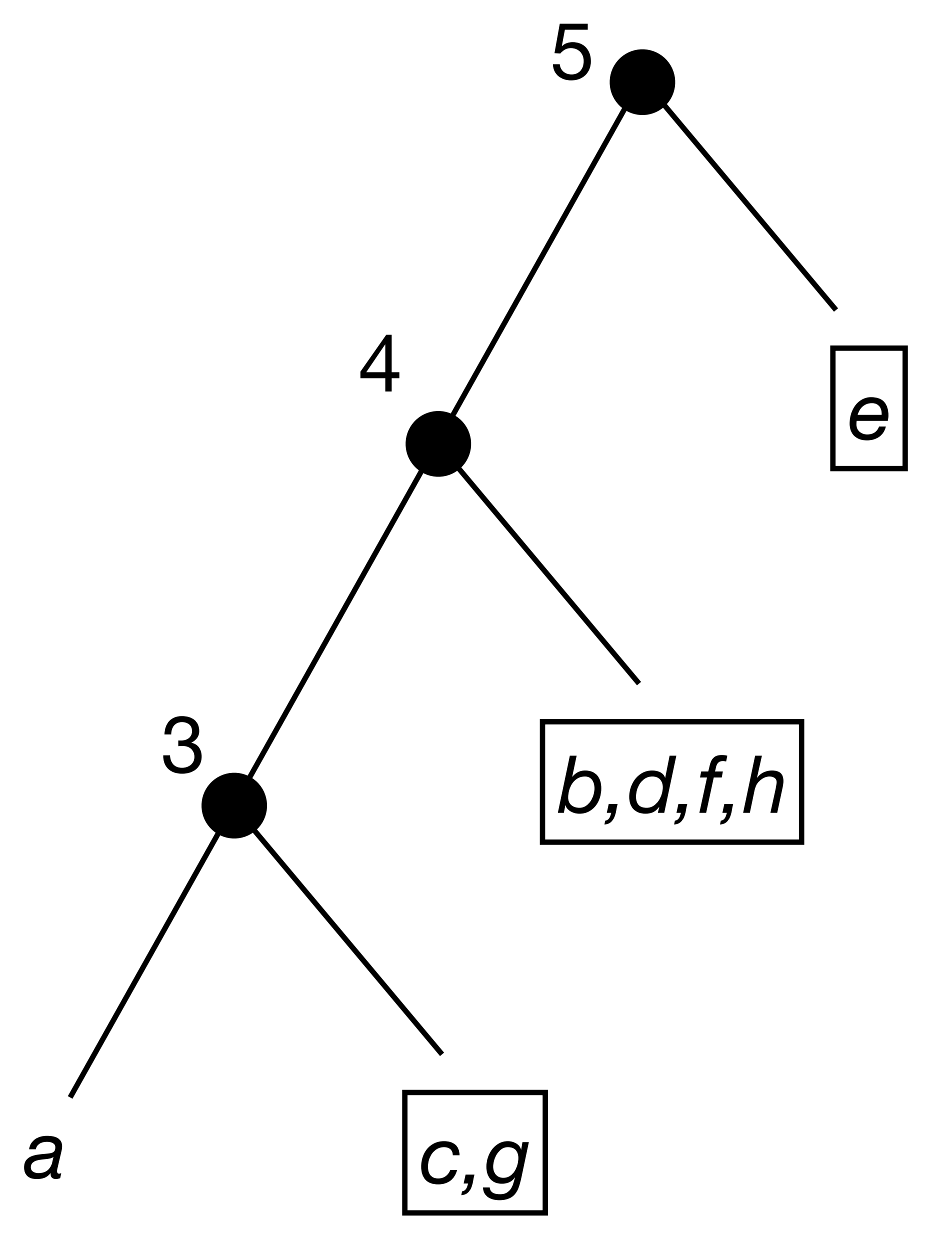
Ultrametric Trees

Theorem A symmetric matrix *D* has an (min-)ultramtric tree iff *D* is an (min-)ultrametric distance.

Proof The "only-if" part is observed in the figure on the last slide. Prove the "if" by construction:

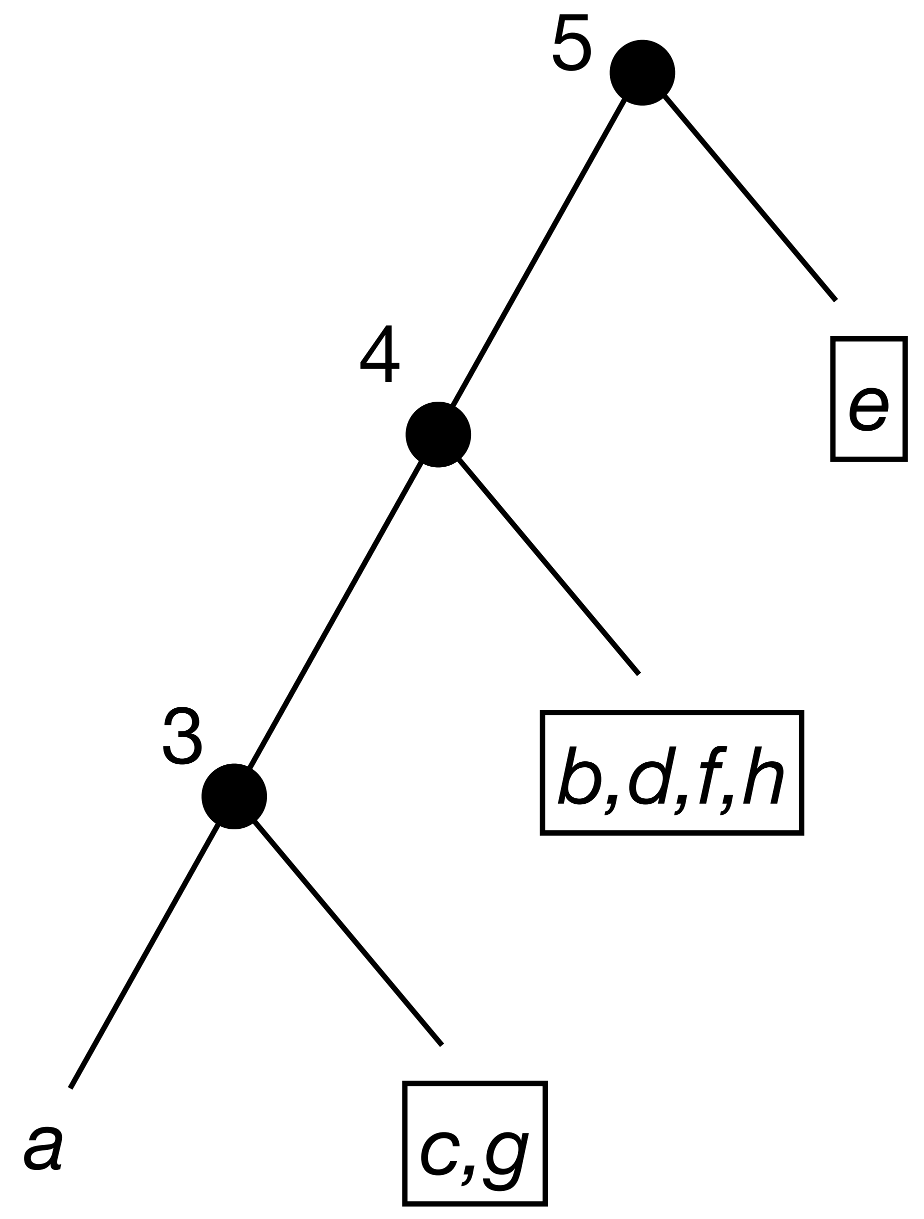
- let *i* be an index such that *D(i,i)* ≠ *D(i,j)* for all *i* ≠ *j.*
- assume there are *d* distinct values in row *i* of *D,* then the path from the root to the leaf labeled by *i* must pass though exactly *d* nodes, in decreasing order.
- the other children of those internal nodes, have subtrees with leaf labels of the other indices with that value in row *i,* call these groups of leaves (s.t. *D(i,j)* = *D(i,k)*) **classes**, and the set of *d-1* classes a **partitioning**.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a	0	4	3	4	5	4	3	4
b								
c								
d								
e								
f								
g								
h								



Ultrametric Trees

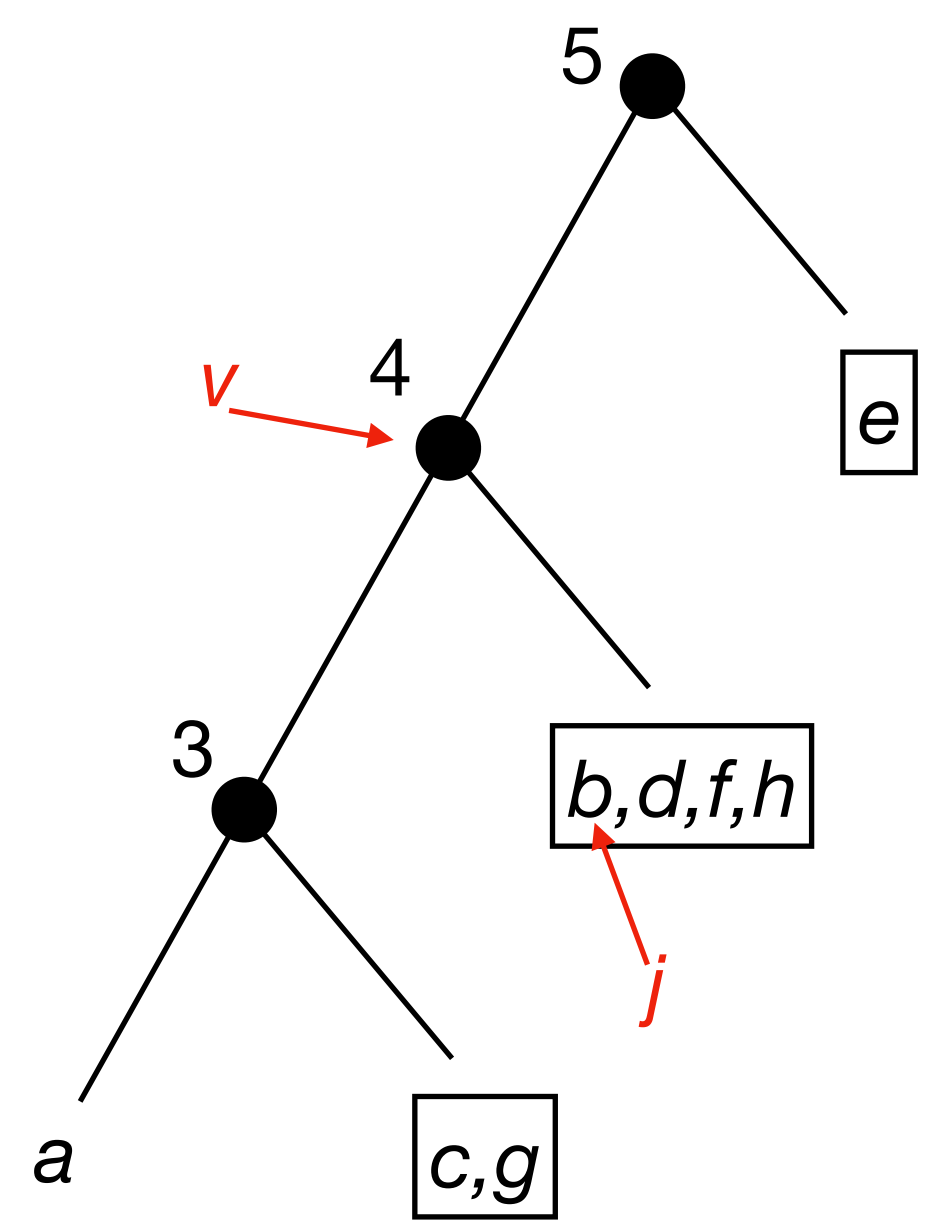
Proof (continued)



Ultrametric Trees

Proof (continued)

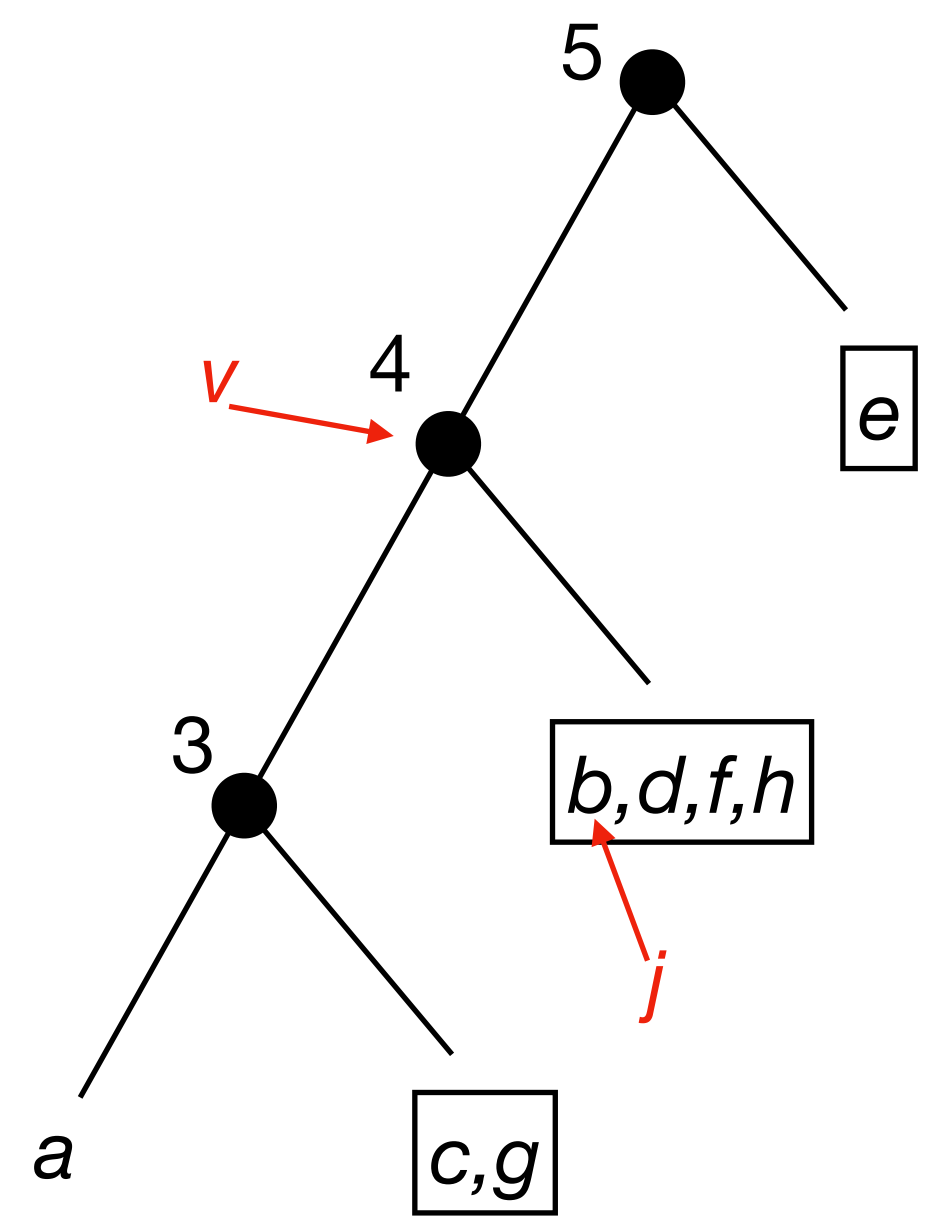
- for each internal node *v*, let *j* be a leaf contained in the class at that node.



Ultrametric Trees

Proof (continued)

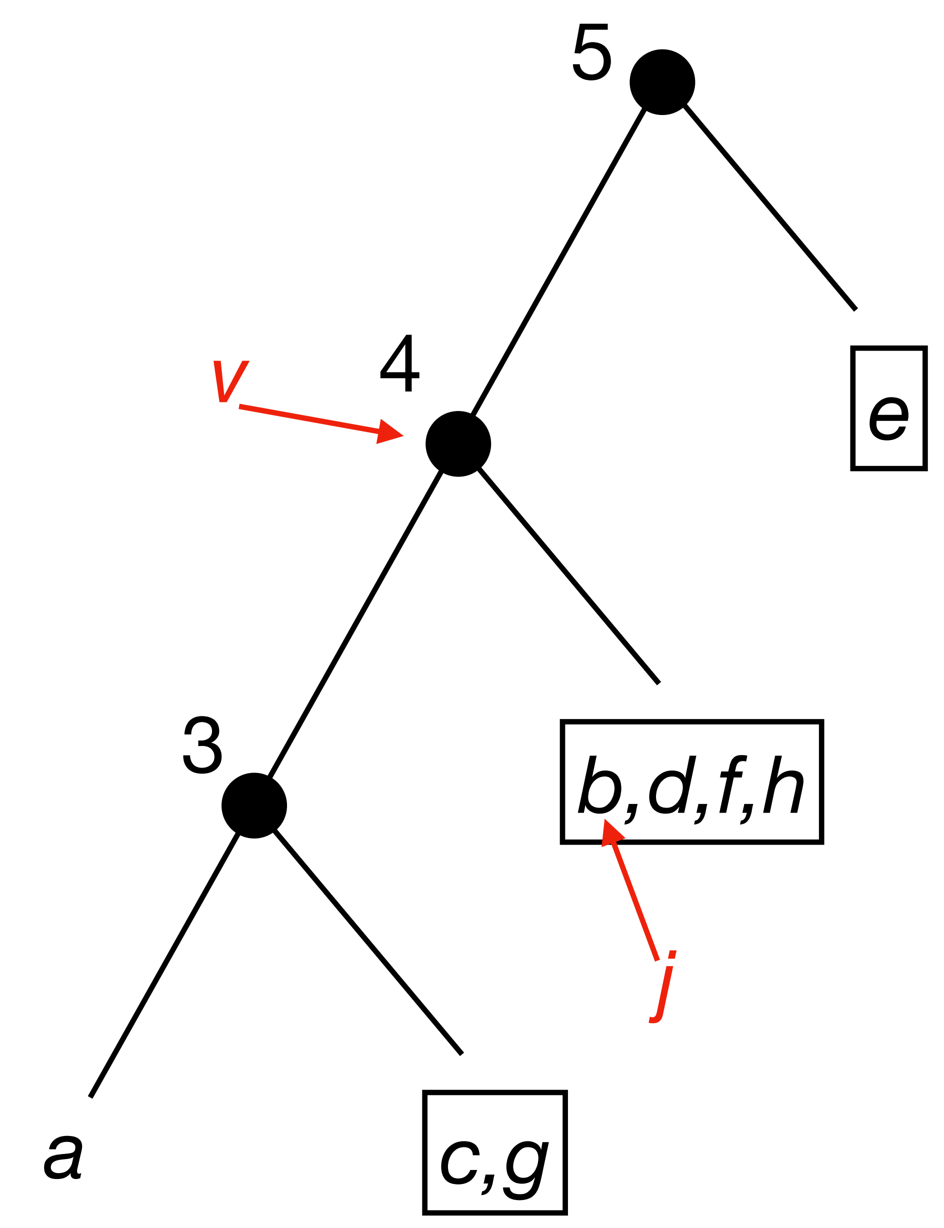
- for each internal node *v,* let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:



Ultrametric Trees

Proof (continued)

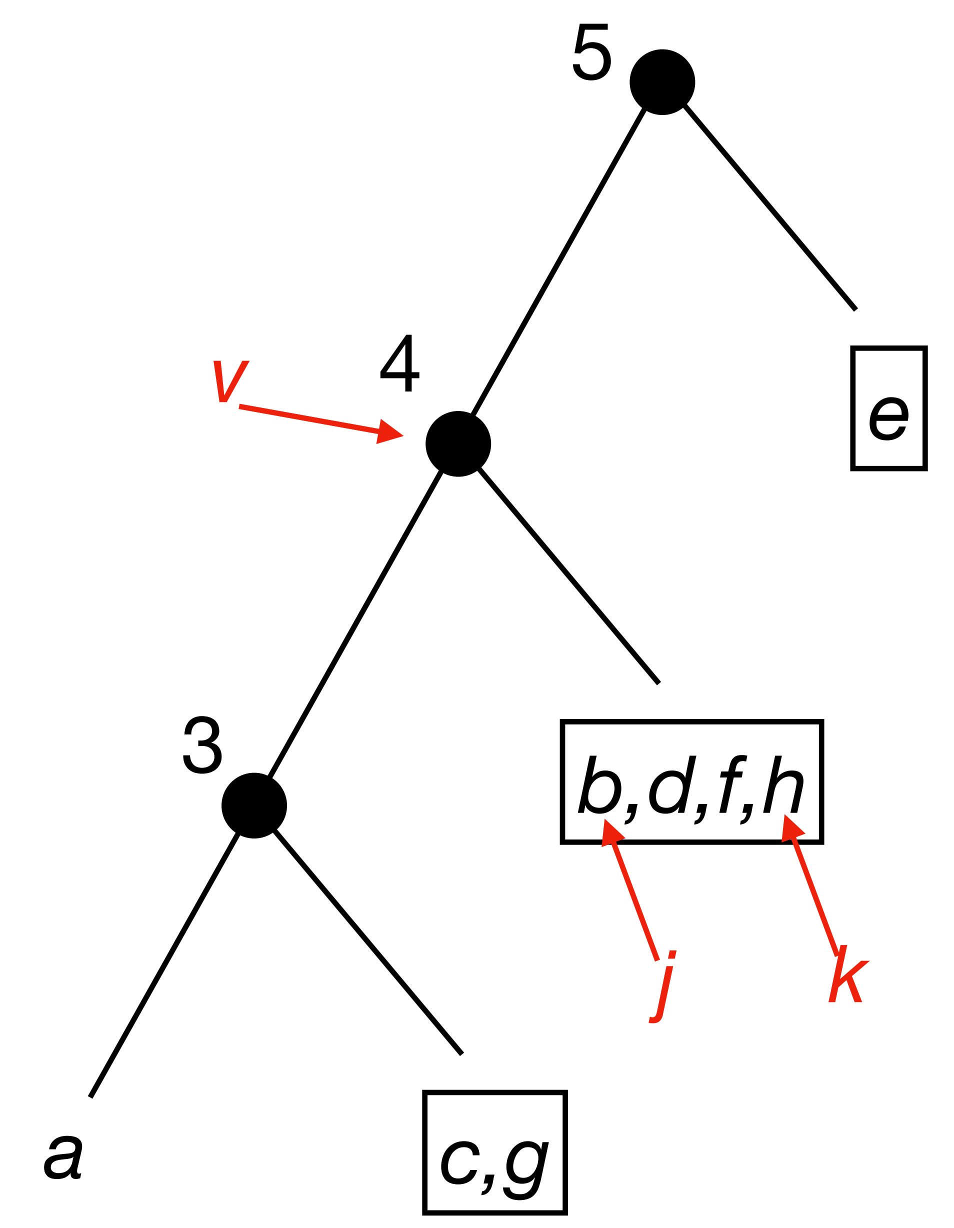
- for each internal node *v*, let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
    - *j* and *k* are in the same class,



Ultrametric Trees

Proof (continued)

- for each internal node *v*, let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
    - *j* and *k* are in the same class,

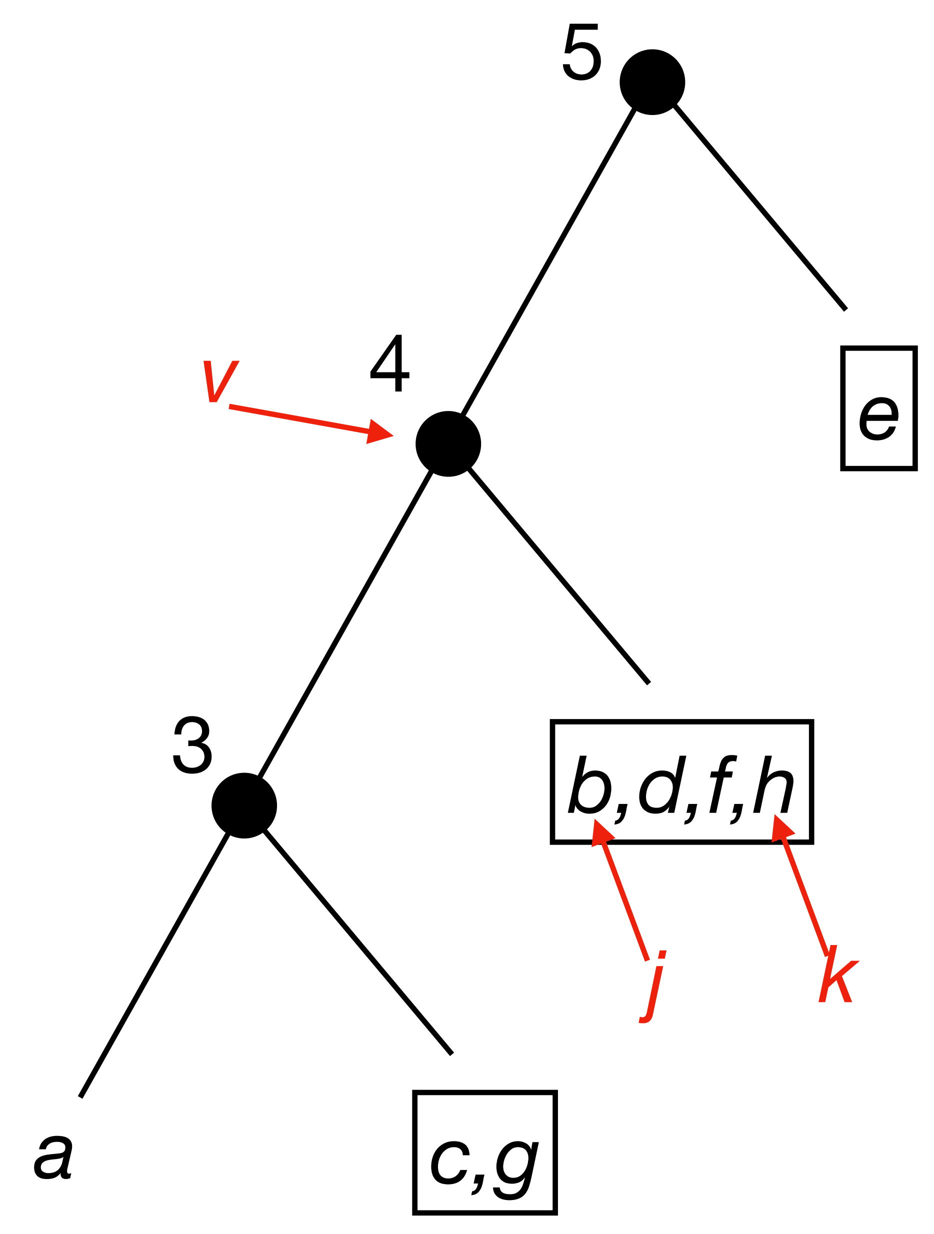


Ultrametric Trees

Proof (continued)

- for each internal node *v,* let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
  - *j* and *k* are in the same class,

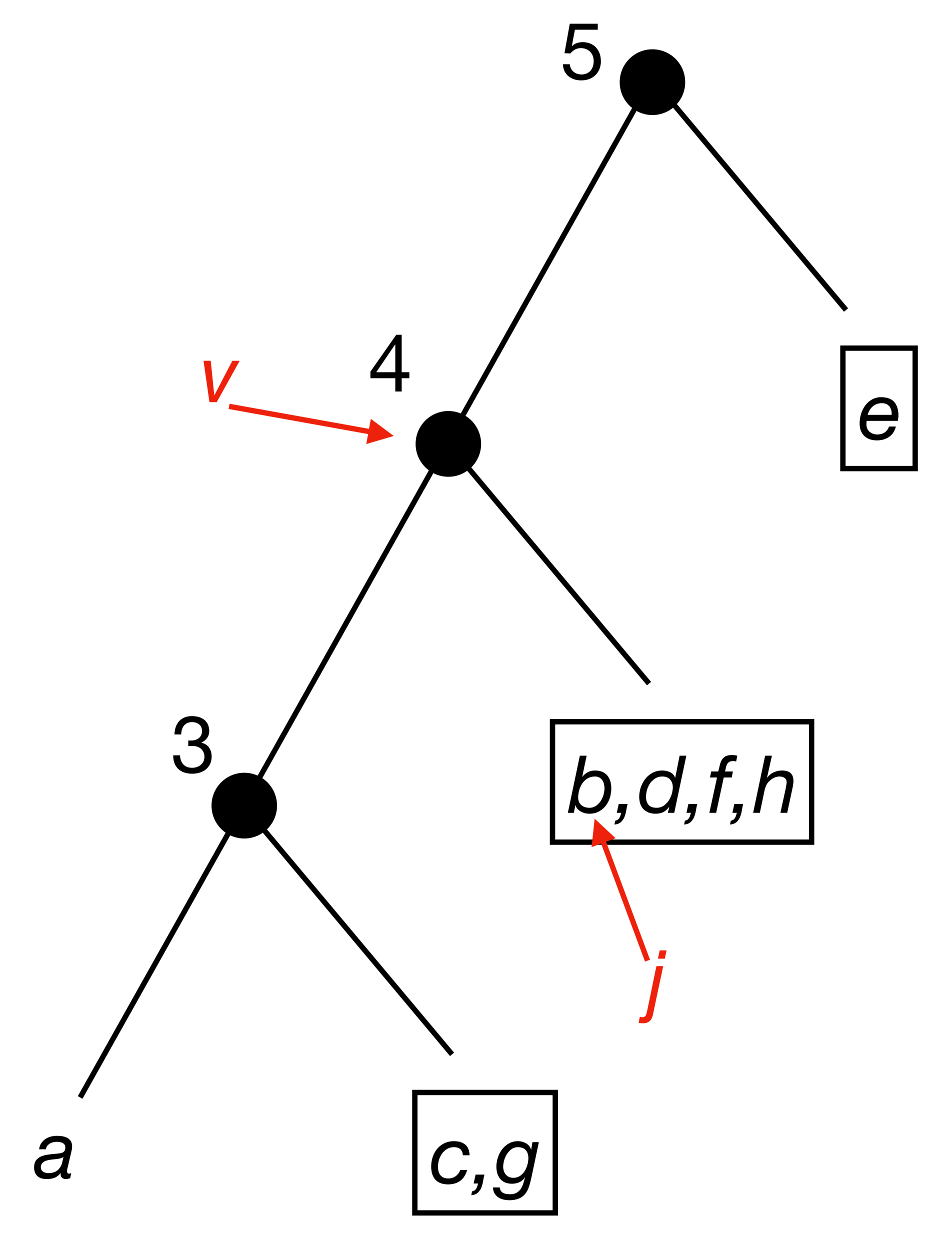
Since *D* is an ultrametric matrix, and *D(i,j)=D(i,k)* we know *D(j,k)* < *D(i,j),* so *D(j,k)* can be correctly represented once we build the subtree



Ultrametric Trees

Proof (continued)

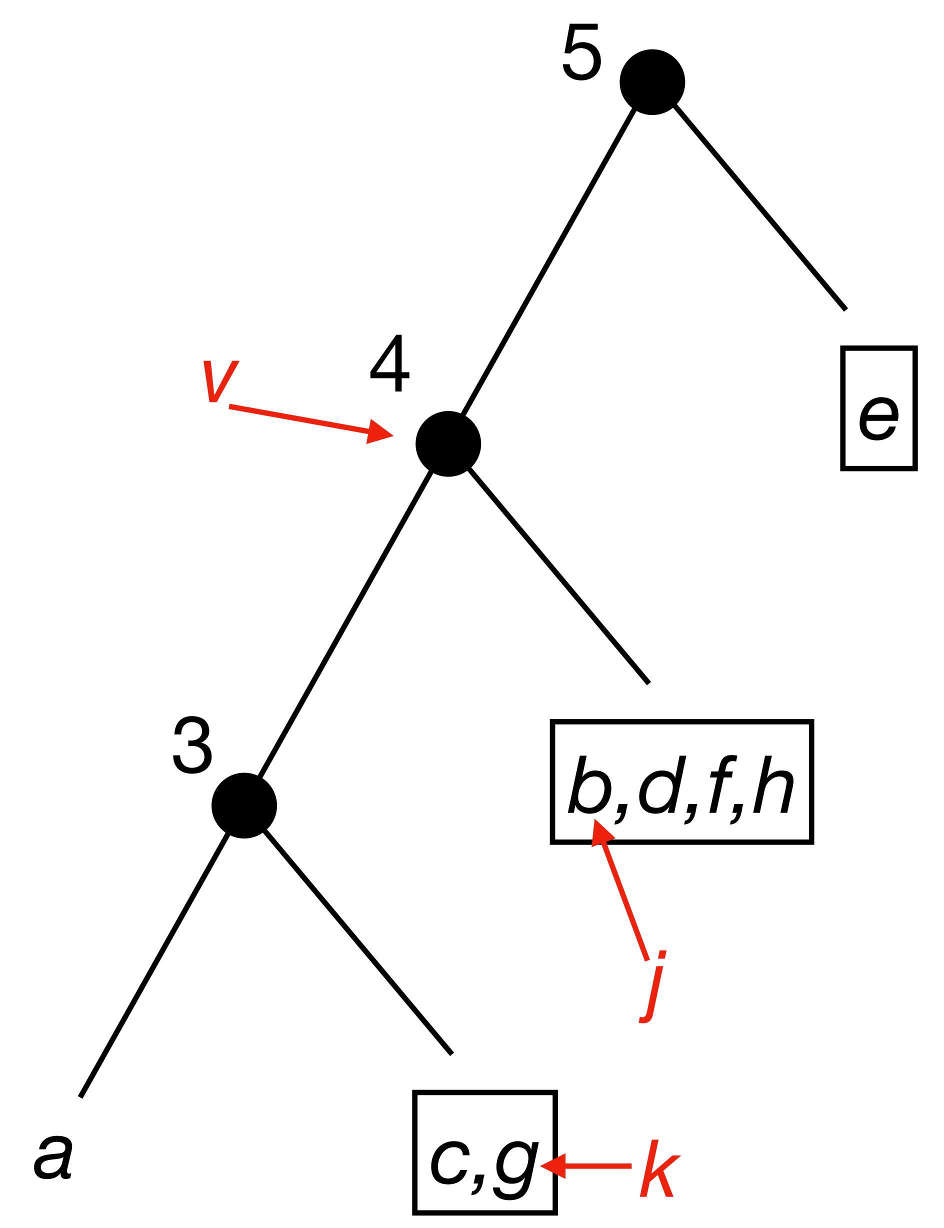
- for each internal node *v*, let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
  - *j* and *k* are in the same class,
  - *k* is located between leaf *i* and node *v*



Ultrametric Trees

Proof (continued)

- for each internal node *v,* let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
  - *j* and *k* are in the same class,
  - *k* is located between leaf *i* and node *v*

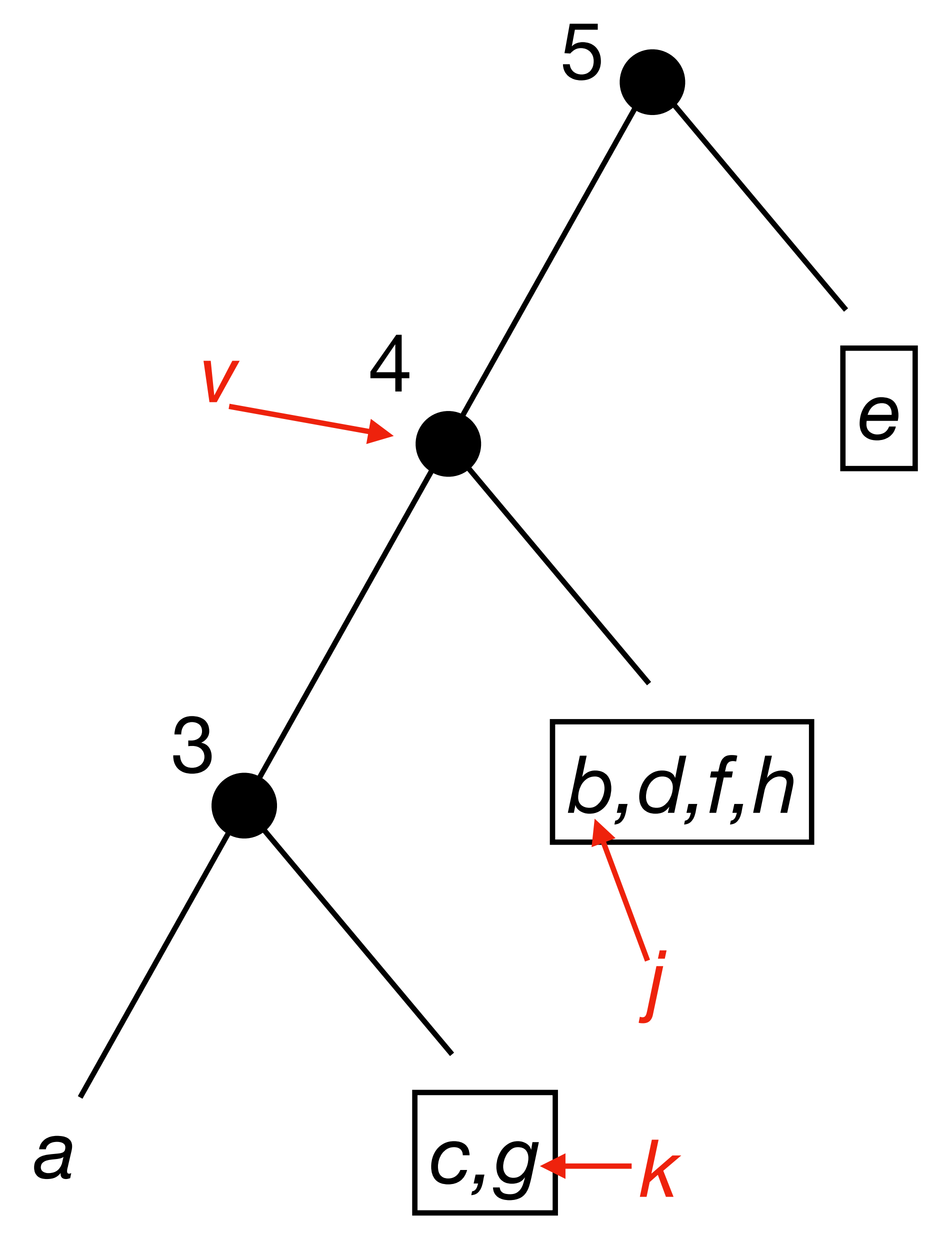


Ultrametric Trees

Proof (continued)

- for each internal node *v*, let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
  - *j* and *k* are in the same class,
  - *k* is located between leaf *i* and node *v*

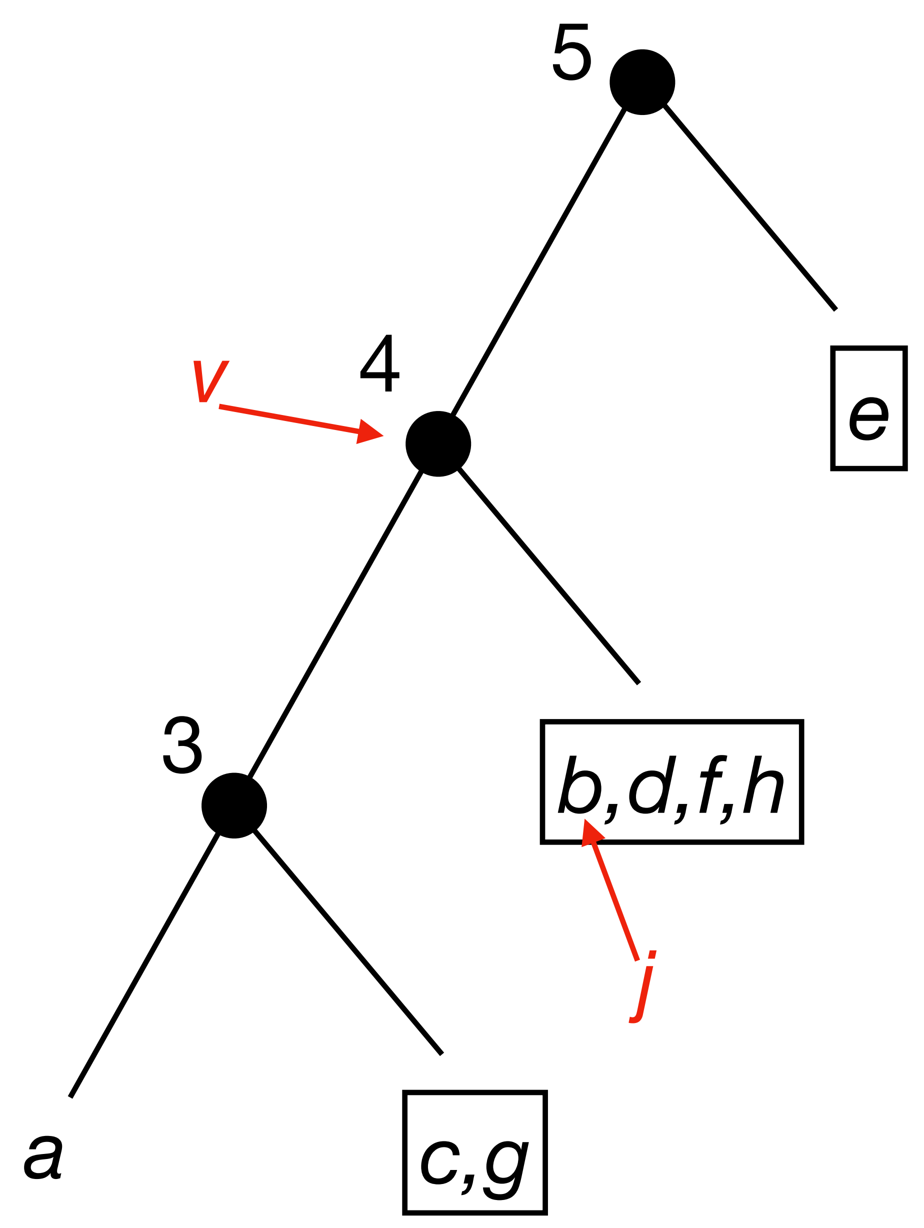
D(i,j) < *D(i,k)* so *D(j,k)* = *D(i,k)*,
and *D(j,k)* is correctly represented



Ultrametric Trees

Proof (continued)

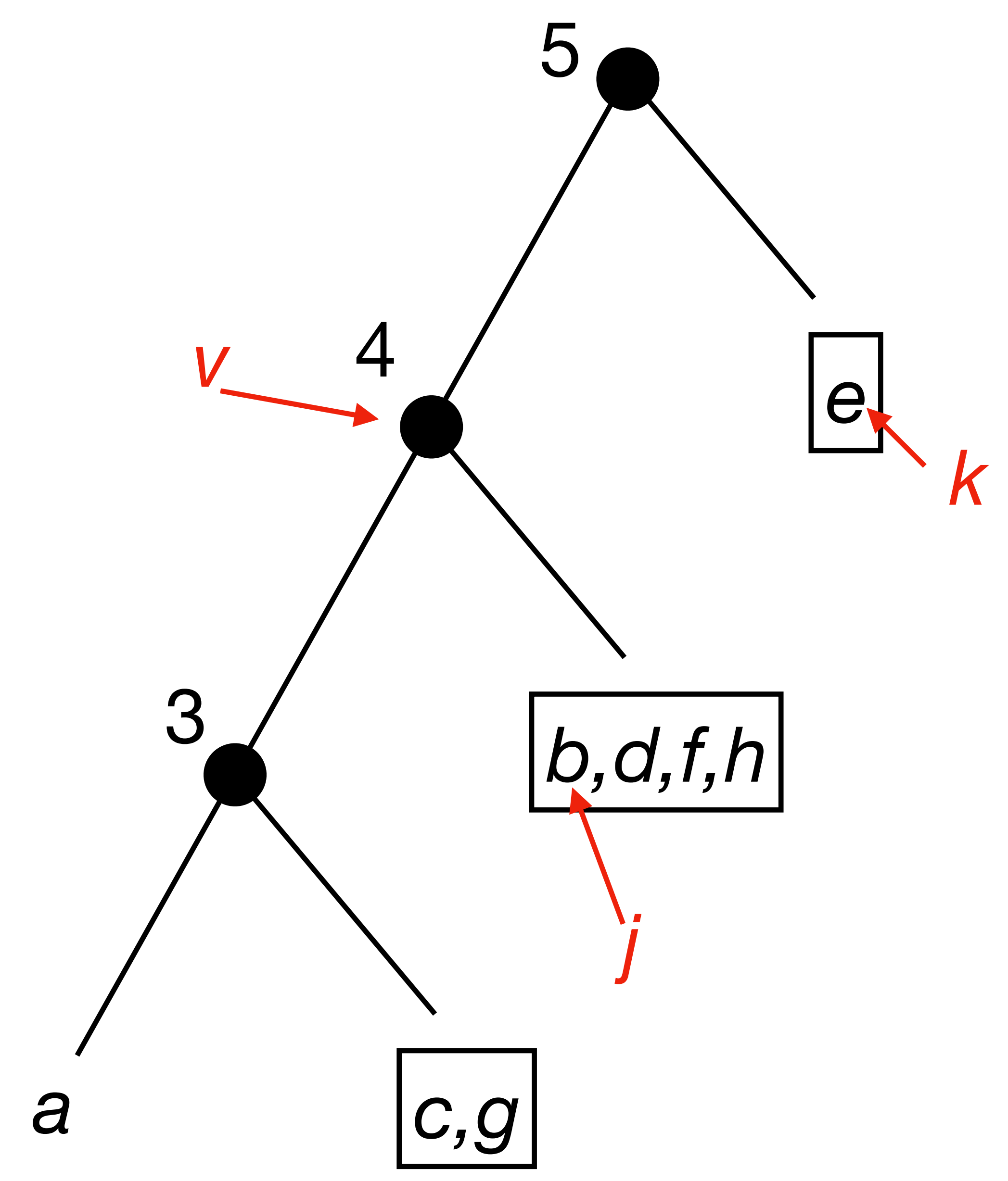
- for each internal node *v,* let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
    - *j* and *k* are in the same class,
    - *k* is located between leaf *i* and node *v*
    - *k* is located between node *v* and the root



Ultrametric Trees

Proof (continued)

- for each internal node *v,* let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
  - *j* and *k* are in the same class,
  - *k* is located between leaf *i* and node *v*
  - *k* is located between node *v* and the root

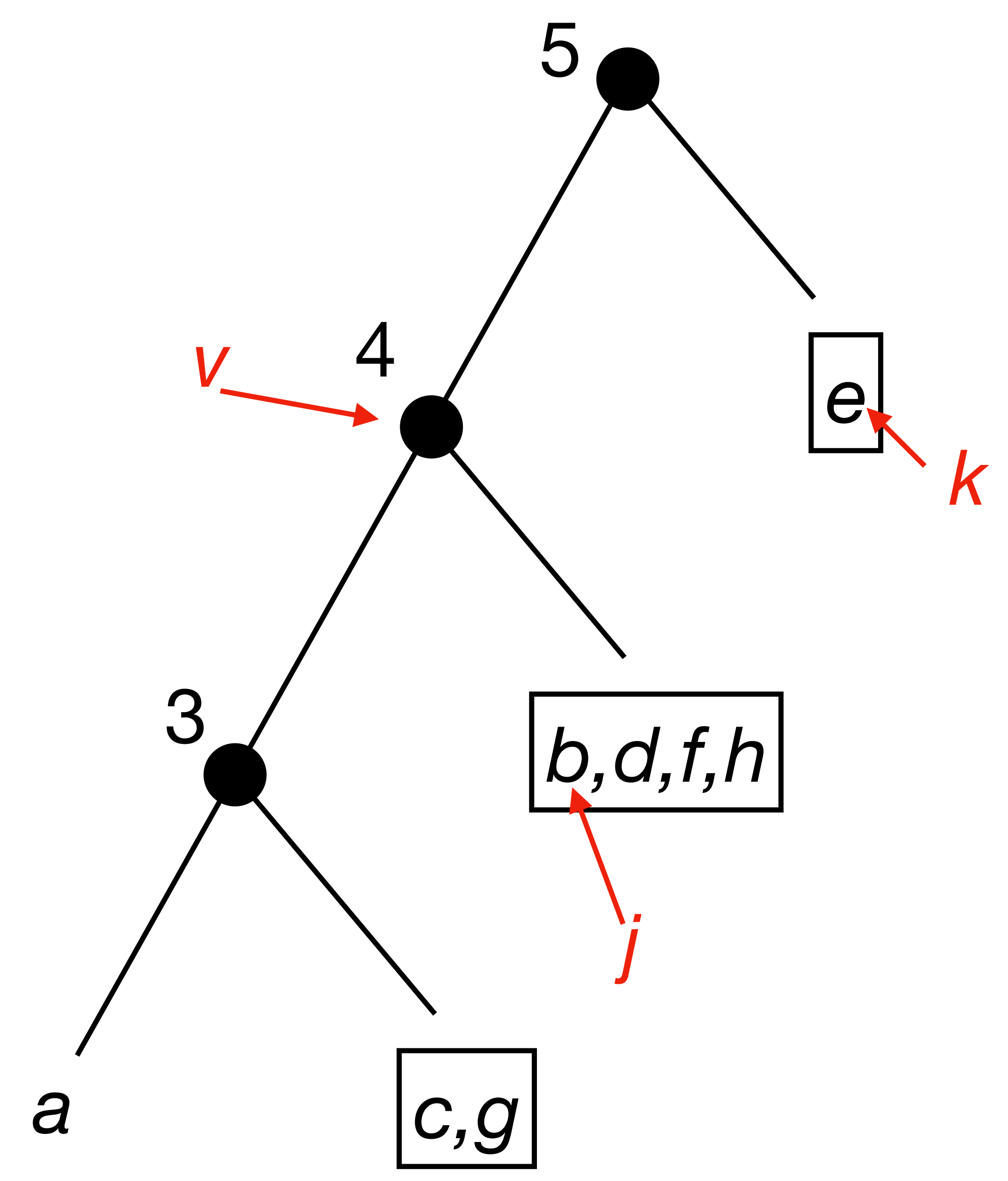


Ultrametric Trees

Proof (continued)

- for each internal node *v,* let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
  - *j* and *k* are in the same class,
  - *k* is located between leaf *i* and node *v*
  - *k* is located between node *v* and the root

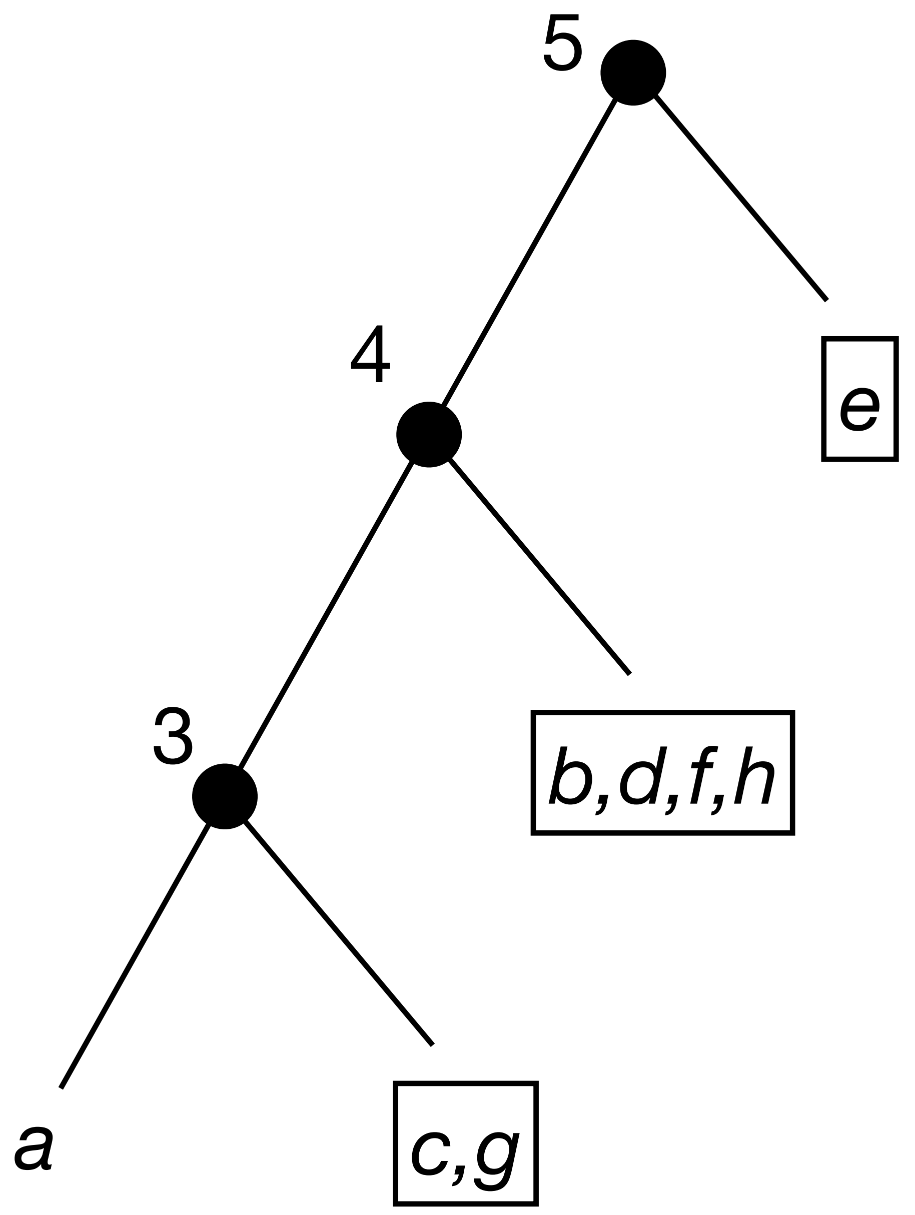
D(i,j) > *D(i,k)* so *D(j,k)* = *D(i,k),*
and *D(j,k)* is correctly represented



Ultrametric Trees

Proof (continued)

- for each internal node *v,* let *j* be a leaf contained in the class at that node.
- for each other node *k* there are 3 cases:
  - *j* and *k* are in the same class,
  - *k* is located between leaf *i* and node *v*
  - *k* is located between node *v* and the root
- When *D* is an ultrametric distance, the recursive approach constructs an ultrametric tree.



Ultrametric Trees

Theorem If *D* is an ultrametric matrix, then the ultrametric tree for *D* is unique.

- when constructing the tree, the partition is *forced* by the labels in *D*.
- that path from the root to *i* has to exist in *every* tree.
- the positioning of the classes in the tree is also forces.
- uniqueness is implied by these facts.

Theorem If *D* is an ultrametric matrix, then the ultrametric tree can be constructed in $O(n^2)$-time.

Ultrametric Trees

Data acquisition

Ultrametric Trees

Data acquisition

- **molecular clock theory** -- "accepted" mutations occur in proteins at a constant rate, therefore the time of the split (value in *D*) between proteins is the number of changes over 2. Measured physically or chemically.

Ultrametric Trees

Data acquisition

- **molecular clock theory** -- "accepted" mutations occur in proteins at a constant rate, therefore the time of the split (value in *D*) between proteins is the number of changes over 2. Measured physically or chemically.
- **lab-based methods** -- example is hybridization experiments. Heat DNA till the double strand breaks, put two sets of (now single strand) DNA in the same solution and allow them to hybridize and measure at what temperature they separate again. The higher the temperature, the stronger the bond and thus smaller the value in *D*.

Ultrametric Trees

Data acquisition

- **molecular clock theory** -- "accepted" mutations occur in proteins at a constant rate, therefore the time of the split (value in *D*) between proteins is the number of changes over 2. Measured physically or chemically.
- **lab-based methods** -- example is hybridization experiments. Heat DNA till the double strand breaks, put two sets of (now single strand) DNA in the same solution and allow them to hybridize and measure at what temperature they separate again. The higher the temperature, the stronger the bond and thus smaller the value in *D*.
- **sequence-based methods** -- use the edit distance or some other similarity measure to find the values in *D.*

Ultrametric Trees

Data acquisition

- **molecular clock theory** -- "accepted" mutations occur in proteins at a constant rate, therefore the time of the split (value in *D*) between proteins is the number of changes over 2. Measured physically or chemically.
- **lab-based methods** -- example is hybridization experiments. Heat DNA till the double strand breaks, put two sets of (now single strand) DNA in the same solution and allow them to hybridize and measure at what temperature they separate again. The higher the temperature, the stronger the bond and thus smaller the value in *D*.
- **sequence-based methods** -- use the edit distance or some other similarity measure to find the values in *D.*

Most "real" data is not ultrametric, and ultrametric data does not necessarily reflect reality.

Ultrametric Trees

Data acquisition

- **molecular clock theory** -- "accepted" mutations occur in proteins at a constant rate, therefore the time of the split (value in *D*) between proteins is the number of changes over 2. Measured physically or chemically.
- **lab-based methods** -- example is hybridization experiments. Heat DNA till the double strand breaks, put two sets of (now single strand) DNA in the same solution and allow them to hybridize and measure at what temperature they separate again. The higher the temperature, the stronger the bond and thus smaller the value in *D*.
- **sequence-based methods** -- use the edit distance or some other similarity measure to find the values in *D.*

Most "real" data is not ultrametric, and ultrametric data does not necessarily reflect reality.

- when it does happen (or close to), its strong evidence that what's being measured is close to capturing the true evolutionary history

Ultrametric Trees

Data acquisition

- **molecular clock theory** -- "accepted" mutations occur in proteins at a constant rate, therefore the time of the split (value in *D*) between proteins is the number of changes over 2. Measured physically or chemically.
- **lab-based methods** -- example is hybridization experiments. Heat DNA till the double strand breaks, put two sets of (now single strand) DNA in the same solution and allow them to hybridize and measure at what temperature they separate again. The higher the temperature, the stronger the bond and thus smaller the value in *D*.
- **sequence-based methods** -- use the edit distance or some other similarity measure to find the values in *D.*

Most "real" data is not ultrametric, and ultrametric data does not necessarily reflect reality.

- when it does happen (or close to), its strong evidence that what's being measured is close to capturing the true evolutionary history
- *related question:* what is the smallest amount of perterbation needed to make the data ultrametric?

Additive-distance trees

Ultrametric is the "holy grail", but when its not able to be obtained, we can use a less stringent model.

Additive-distance trees

Ultrametric is the "holy grail", but when its not able to be obtained, we can use a less stringent model.

Definition

Additive-distance trees

Ultrametric is the "holy grail", but when its not able to be obtained, we can use a less stringent model.

Definition

- Let D be a symmetric n by n matrix where the numbers on the diagonal are all 0, and the off-diagonal numbers are all strictly positive.

Additive-distance trees

Ultrametric is the "holy grail", but when its not able to be obtained, we can use a less stringent model.

Definition

- Let *D* be a symmetric *n* by *n* matrix where the numbers on the diagonal are all 0, and the off-diagonal numbers are all strictly positive.
- Let *T* be an edge-weighted tree with at least *n* nodes, where *n* distinct nodes are labeled with rows of *D.*

Additive-distance trees

Ultrametric is the "holy grail", but when its not able to be obtained, we can use a less stringent model.

Definition

- Let *D* be a symmetric *n* by *n* matrix where the numbers on the diagonal are all 0, and the off-diagonal numbers are all strictly positive.
- Let *T* be an edge-weighted tree with at least *n* nodes, where *n* distinct nodes are labeled with rows of *D.*
- Tree *T* is called an *additive tree* if for every pair of *labeled* nodes (*i, j*), the path from node *i* to node *j* has total weight (or distance) exactly *D(i,j)*.

Additive-distance trees

Ultrametric is the "holy grail", but when its not able to be obtained, we can use a less stringent model.

Definition

- Let *D* be a symmetric *n* by *n* matrix where the numbers on the diagonal are all 0, and the off-diagonal numbers are all strictly positive.
- Let *T* be an edge-weighted tree with at least *n* nodes, where *n* distinct nodes are labeled with rows of *D.*
- Tree *T* is called an *additive tree* if for every pair of *labeled* nodes (*i, j*), the path from node *i* to node *j* has total weight (or distance) exactly *D(i,j)*.

Problem

Additive-distance trees

Ultrametric is the "holy grail", but when its not able to be obtained, we can use a less stringent model.

Definition

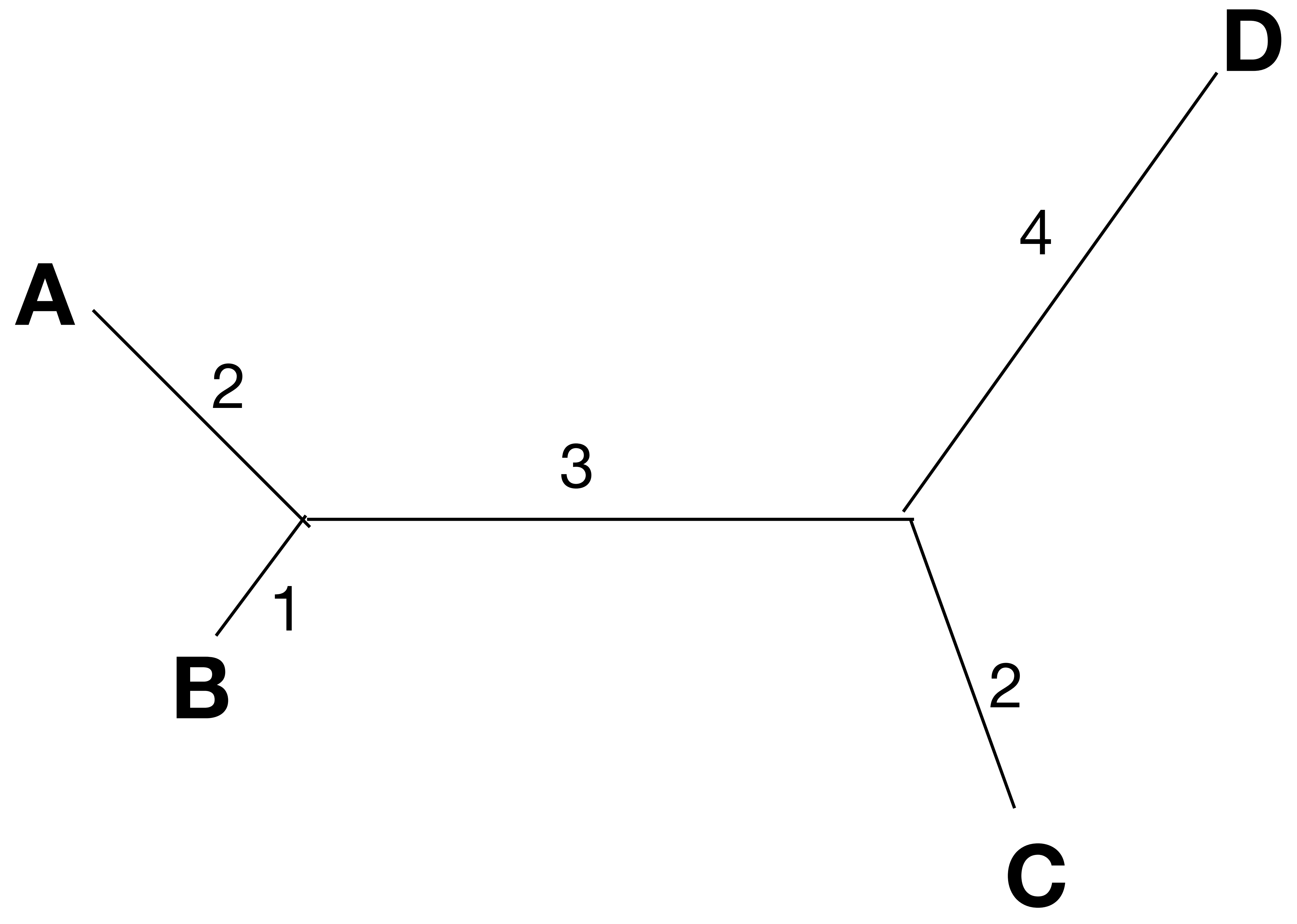
- Let *D* be a symmetric *n* by *n* matrix where the numbers on the diagonal are all 0, and the off-diagonal numbers are all strictly positive.
- Let *T* be an edge-weighted tree with at least *n* nodes, where *n* distinct nodes are labeled with rows of *D.*
- Tree *T* is called an *additive tree* if for every pair of *labeled* nodes (*i, j*), the path from node *i* to node *j* has total weight (or distance) exactly *D(i,j)*.

Problem

- Given a matrix *D* with 0s on the diagonals, and positive numbers in all other locations, find the additive tree *T* or determine that one does not exist.

Additive-distance trees

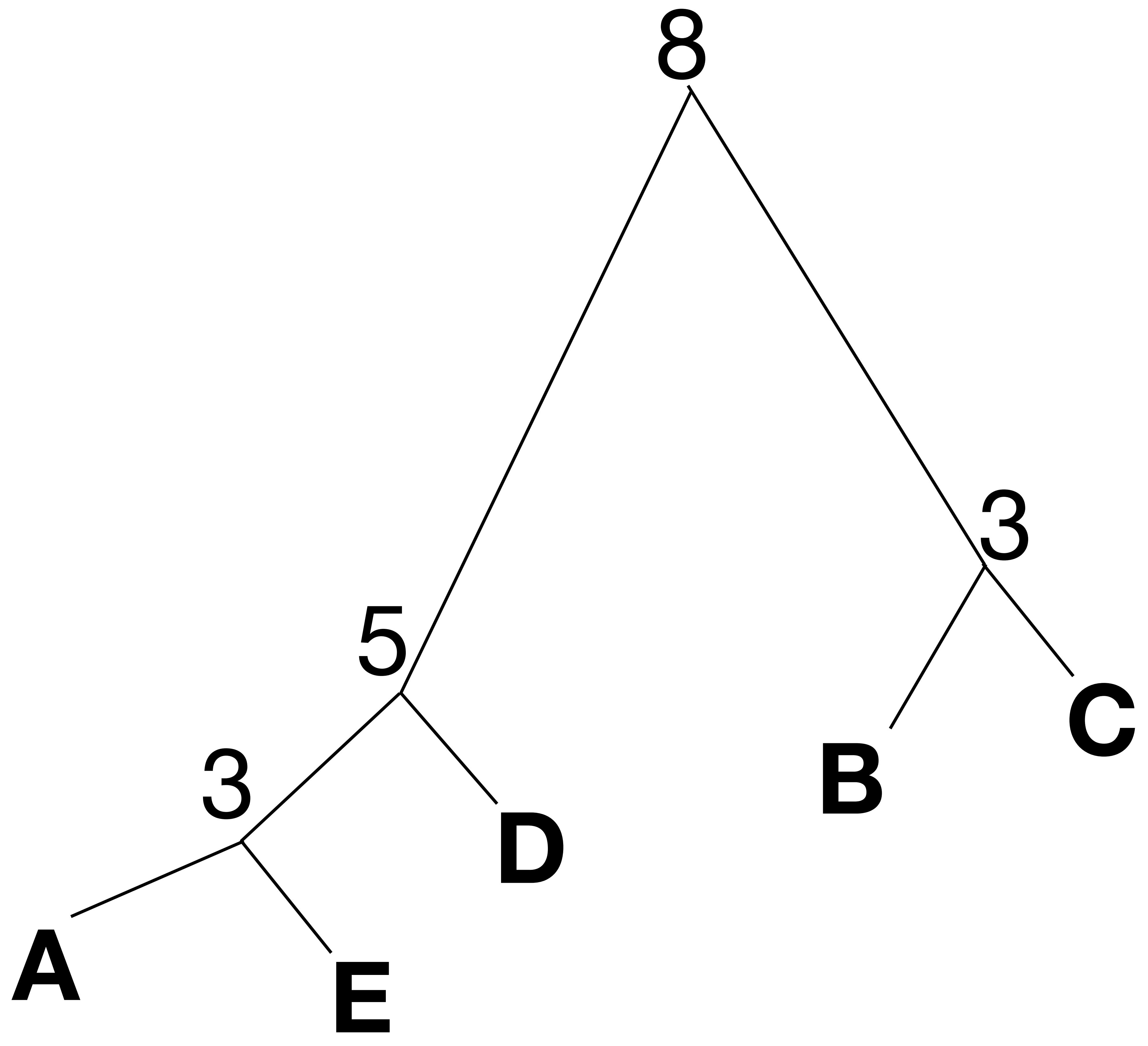
	A	B	C	D
A	0	3	7	9
B		0	6	8
C			0	6
D				0



Additive-distance trees

D is Ultrametric $\xrightarrow{\checkmark}$ **D is Additive**

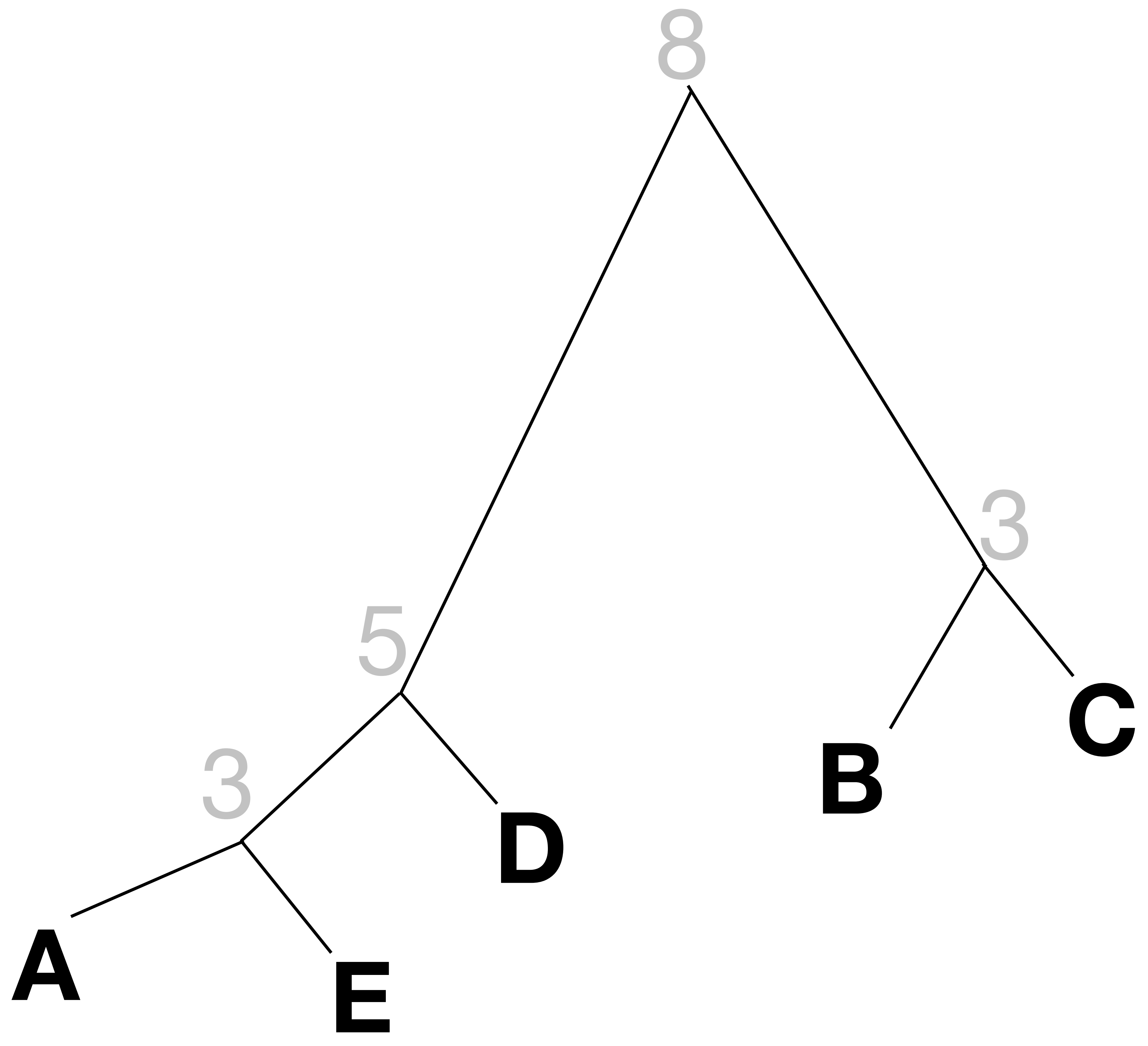
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



Additive-distance trees

$$\text{D is Ultrametric} \xrightarrow{\checkmark} \text{D is Additive}$$

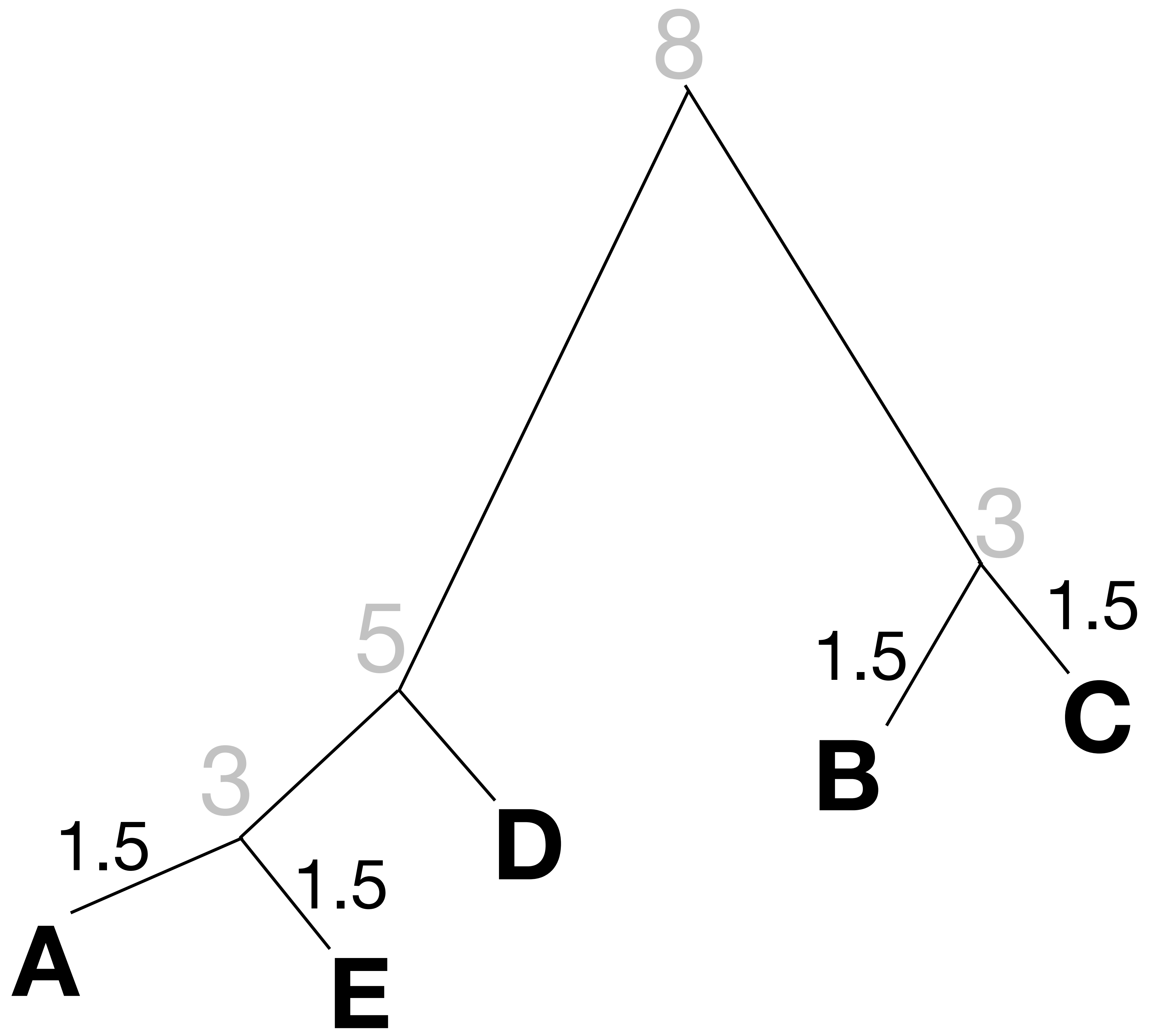
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



Additive-distance trees

D is Ultrametric $\xrightarrow{\checkmark}$ D is Additive

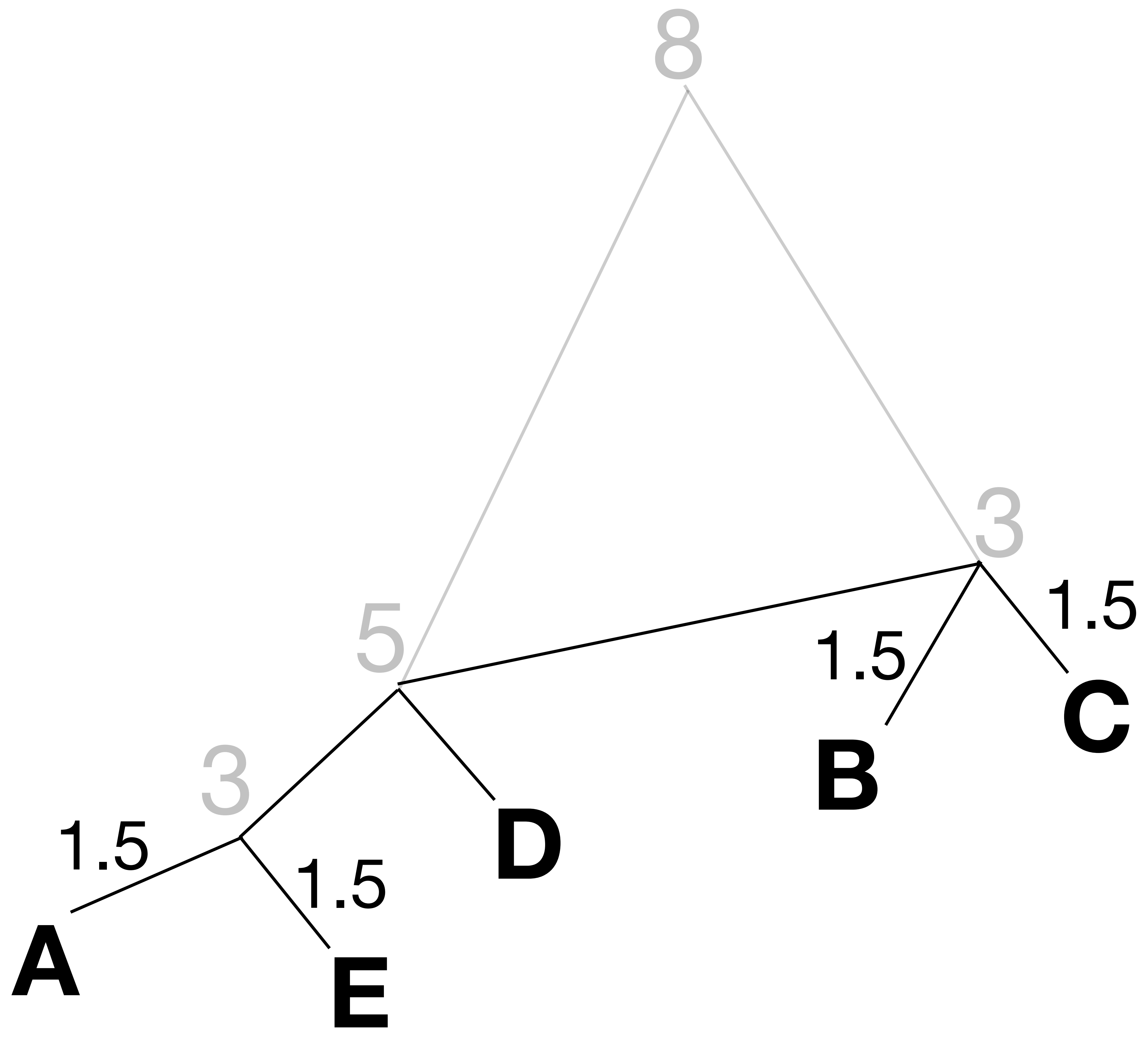
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



Additive-distance trees

$$\text{D is Ultrametric} \xrightarrow{\checkmark} \text{D is Additive}$$

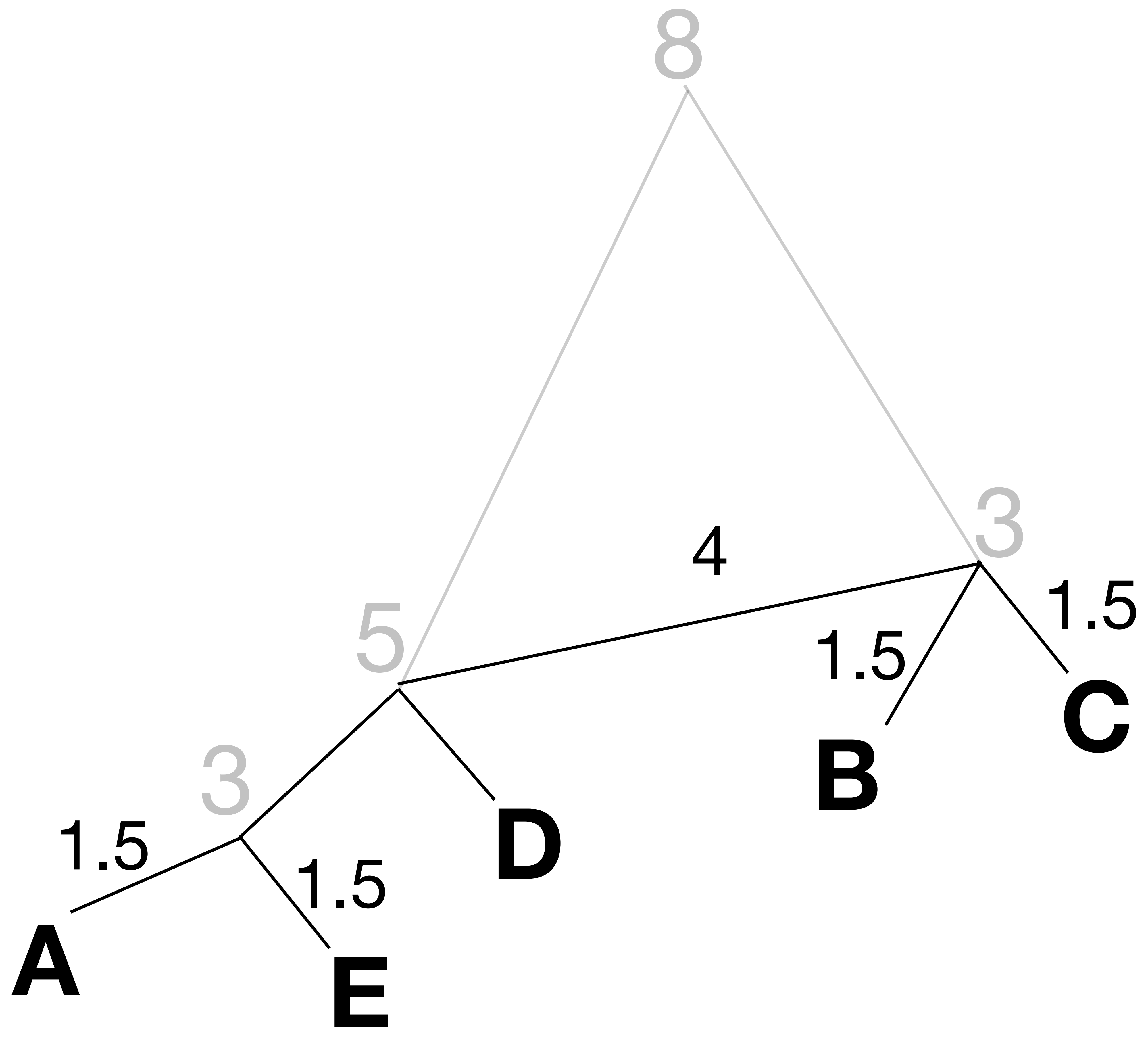
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



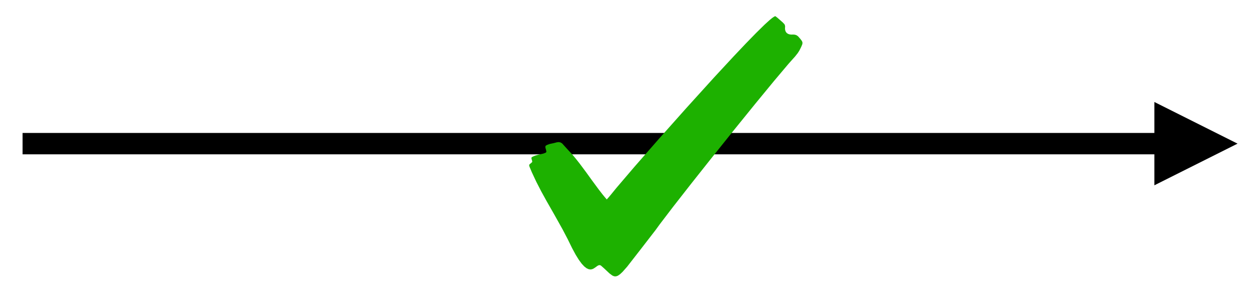
Additive-distance trees

D is Ultrametric $\xrightarrow{\checkmark}$ **D is Additive**

	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



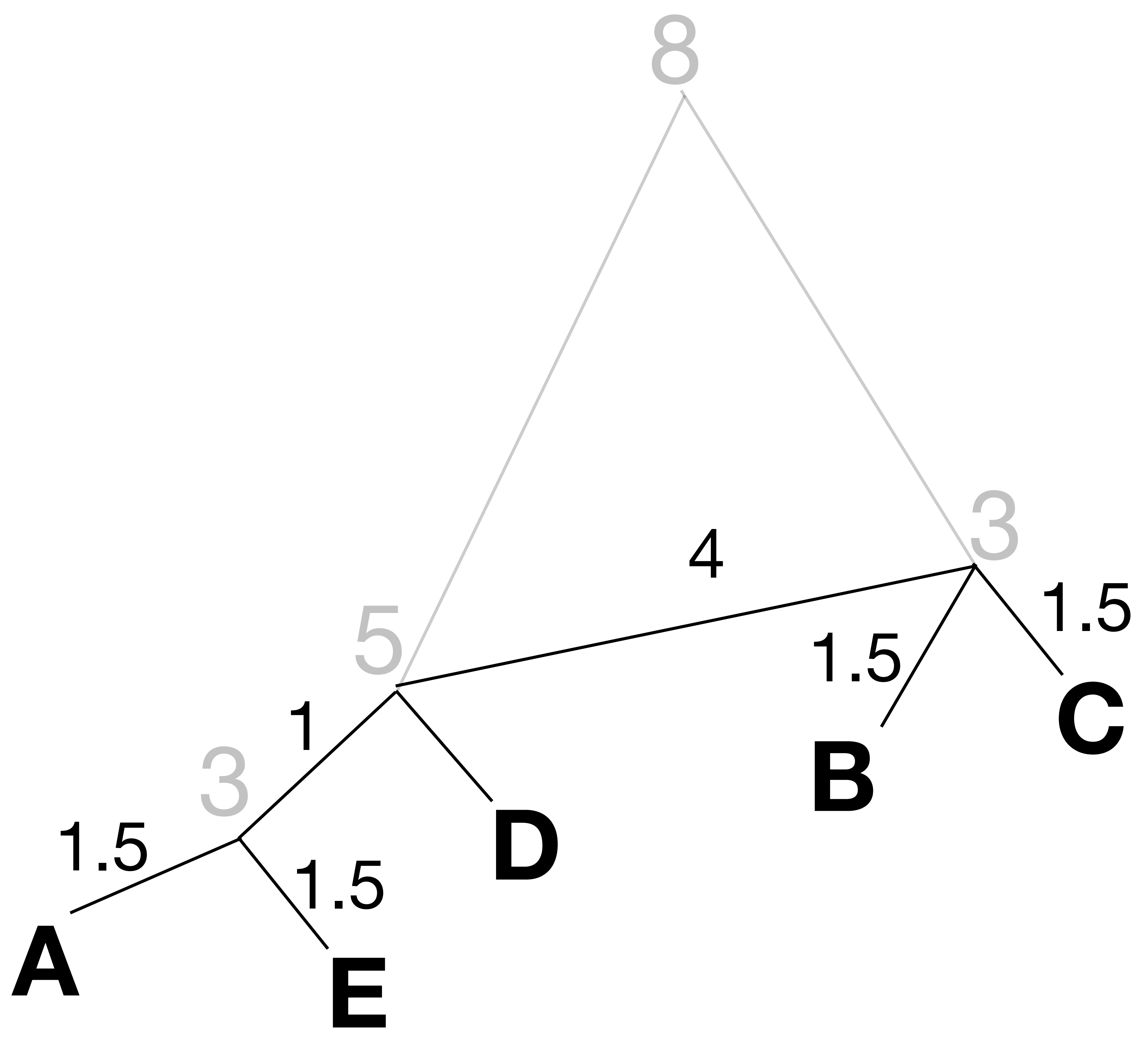
Additive-distance trees



D is Ultrametric

D is Additive

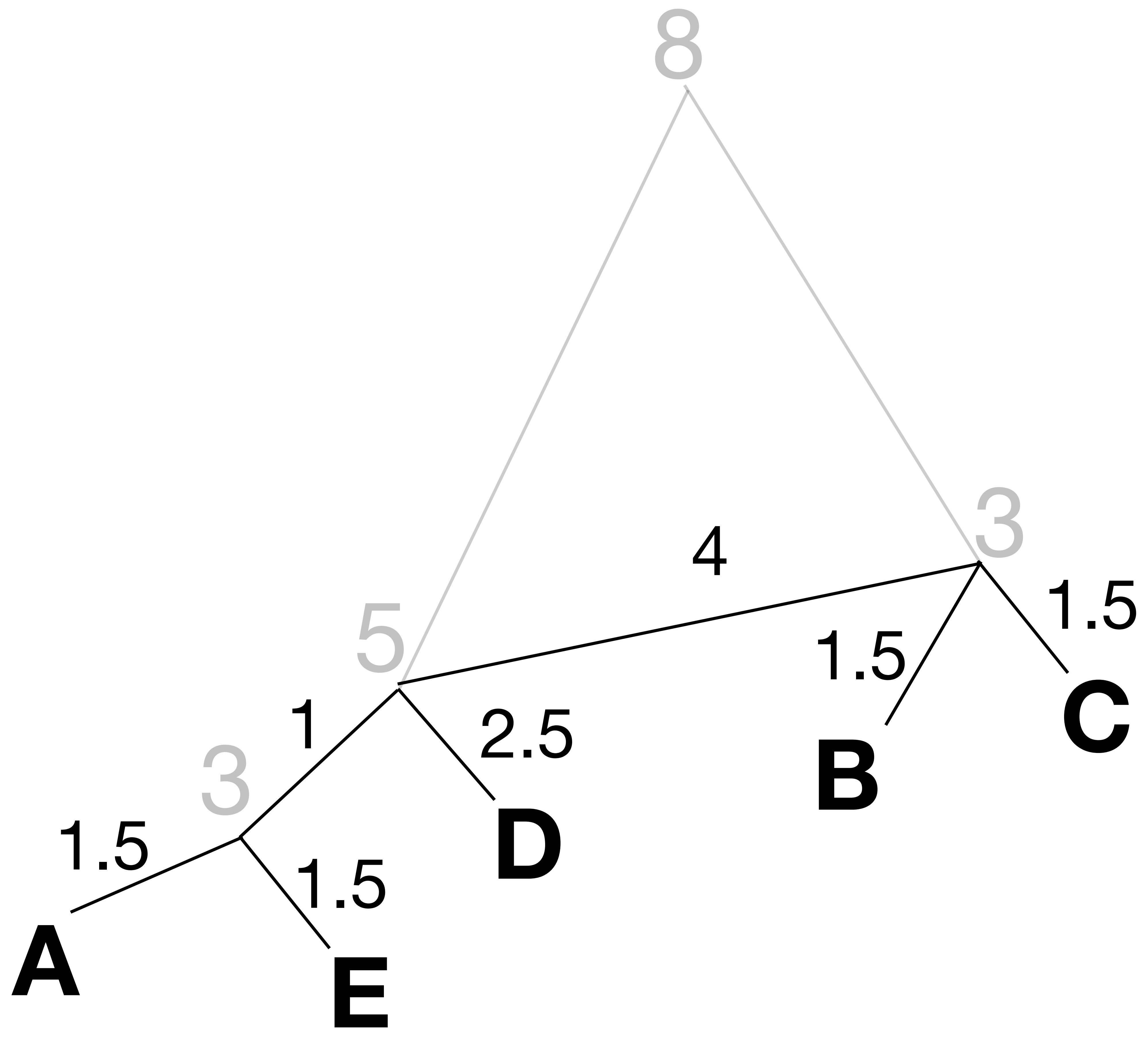
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



Additive-distance trees

$$\text{D is Ultrametric} \xrightarrow{\checkmark} \text{D is Additive}$$

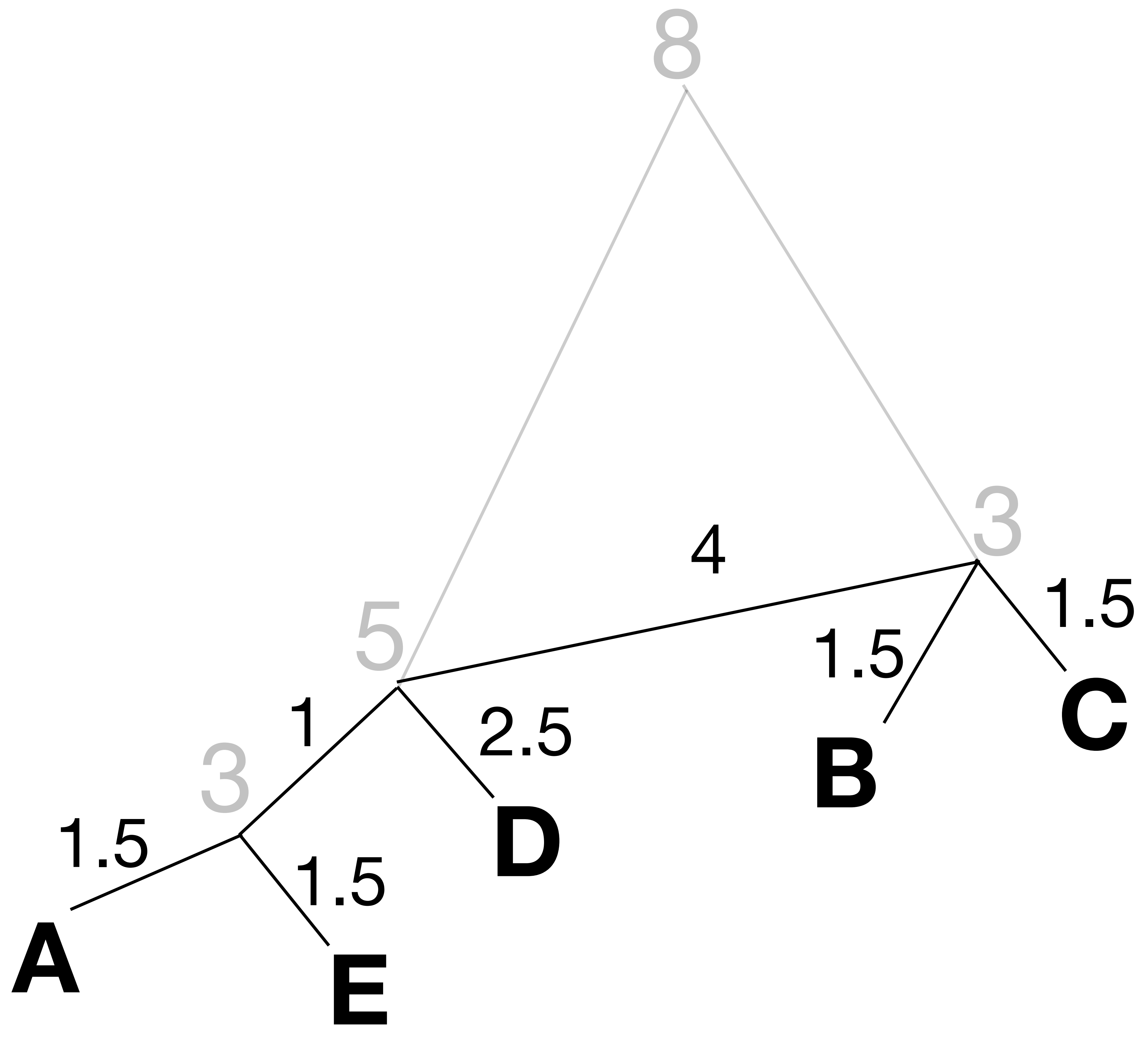
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



Additive-distance trees

$$\text{D is Ultrametric} \underset{\longleftarrow}{\xrightarrow{\checkmark}} \text{D is Additive}$$

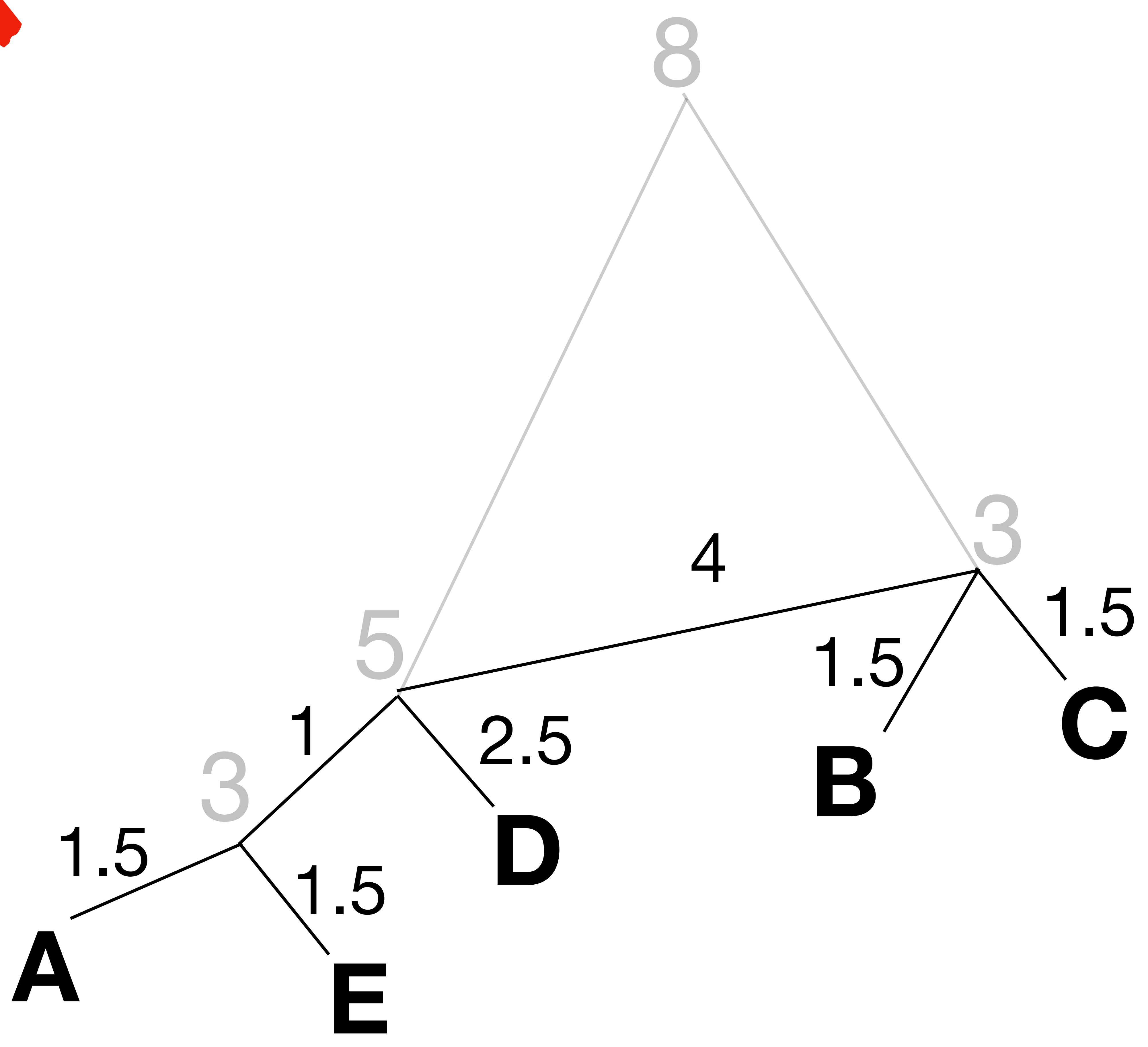
	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



Additive-distance trees

$$\begin{tikzcd} \textbf{D is Ultrametric} \arrow[r, shift left, "\checkmark"] & \textbf{D is Additive} \arrow[l, shift left, "\times"] \end{tikzcd}$$

	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



Additive-distance trees

The algorithms for solving this problem run in $O(n^2)$ and have been described in at least a dozen publications.

The problem can also be reduced to solving the ultrametric tree problem by constructing a special D' matrix in $O(n^2)$ time.

Details are in Gusfield Section 17.4.1.

Parsimony

Parsimony's main principle: "if there exists more than one possible answer to the question, the simpler answer is more likely to be correct" (when you hear hooves think horses not zebra).

Parsimony

Parsimony's main principle: "if there exists more than one possible answer to the question, the simpler answer is more likely to be correct" (when you hear hooves think horses not zebra).

In sequence evolution each character in a sequence will be modified at most one time (sometimes called the *infinite sites* model).

Parsimony

Parsimony's main principle: "if there exists more than one possible answer to the question, the simpler answer is more likely to be correct" (when you hear hooves think horses not zebra).

In sequence evolution each character in a sequence will be modified at most one time (sometimes called the *infinite sites* model).

Therefore, we can change the sequence data into a binary labeling

- 0 if the character is unchanged in this sequence
- 1 if it has already been modified

Parsimony

Parsimony's main principle: "if there exists more than one possible answer to the question, the simpler answer is more likely to be correct" (when you hear hooves think horses not zebra).

In sequence evolution each character in a sequence will be modified at most one time (sometimes called the *infinite sites* model).

Therefore, we can change the sequence data into a binary labeling

- 0 if the character is unchanged in this sequence
- 1 if it has already been modified

Definition Let M be an n by m binary (0-1) matrix representing n objects in terms of m characters or traits that describe the object. Each character takes one of two possible states, 0 or 1, and cell (p,i) of M has the value of 1 iff object p has character i.

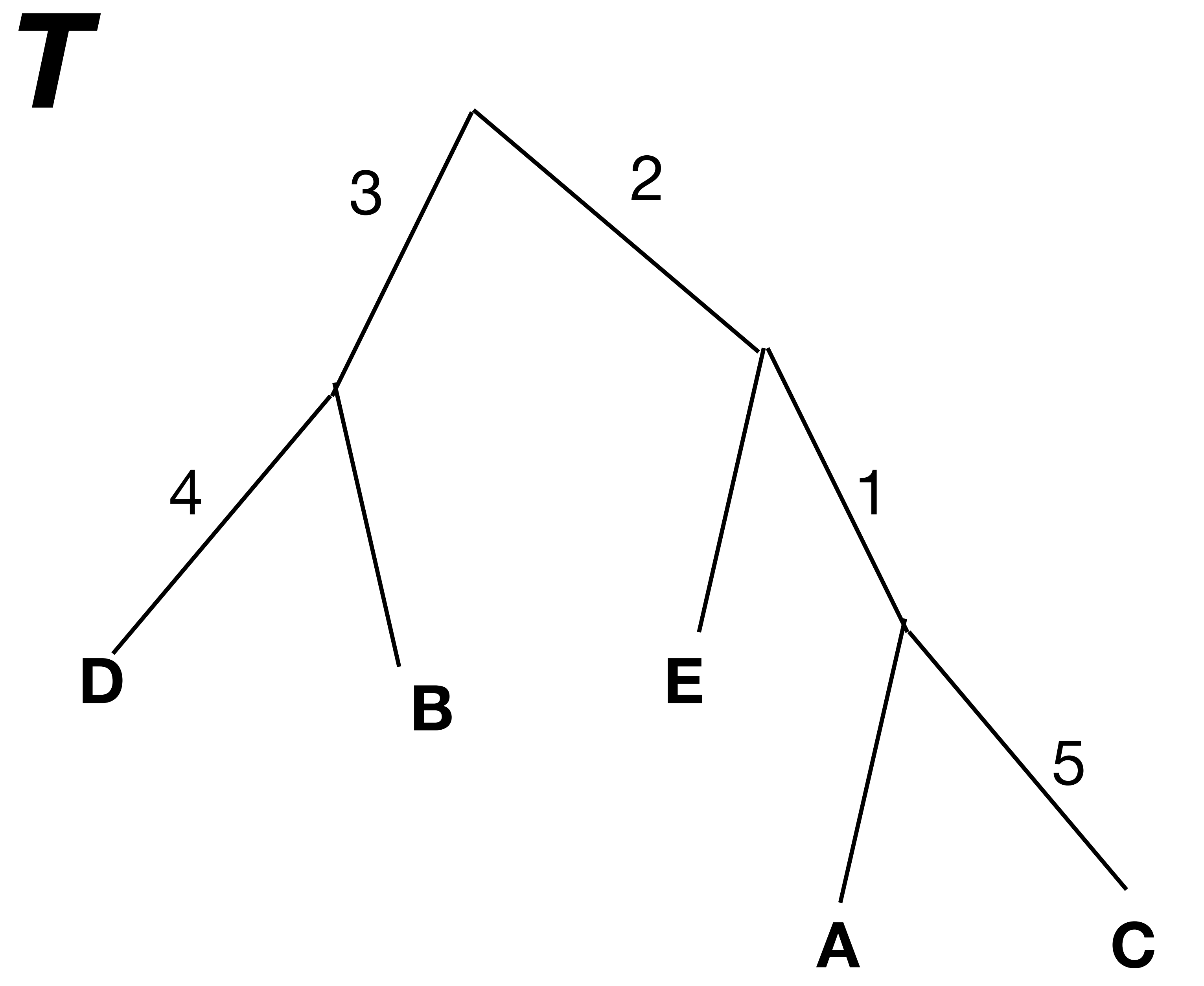
Parsimony

Definition Given an *n* by *m* binary character matrix *M*, a *phylogenetic tree* for *M* is a rooted tree *T* with exactly *n* leaves that obeys the following:

- each of the *n* objects labels exactly 1 leaf of *T*
- each of the *m* characters labels exactly 1 edge of *T*
- for any object *p,* the characters that label the edges along the unique path from the root to the leaf specify all of the characters of *p* whose state is *1.*

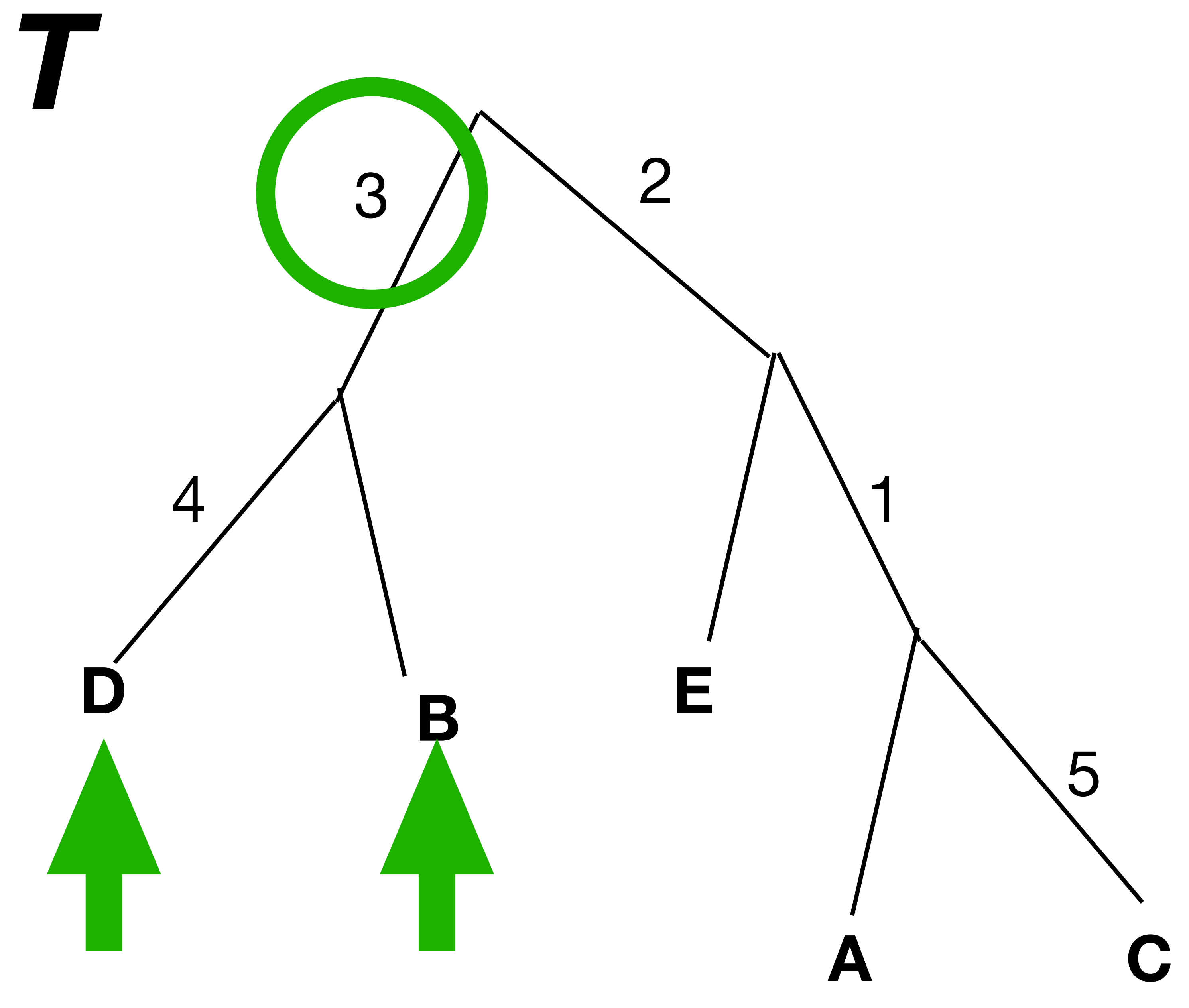
Parsimony

M	**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



Parsimony

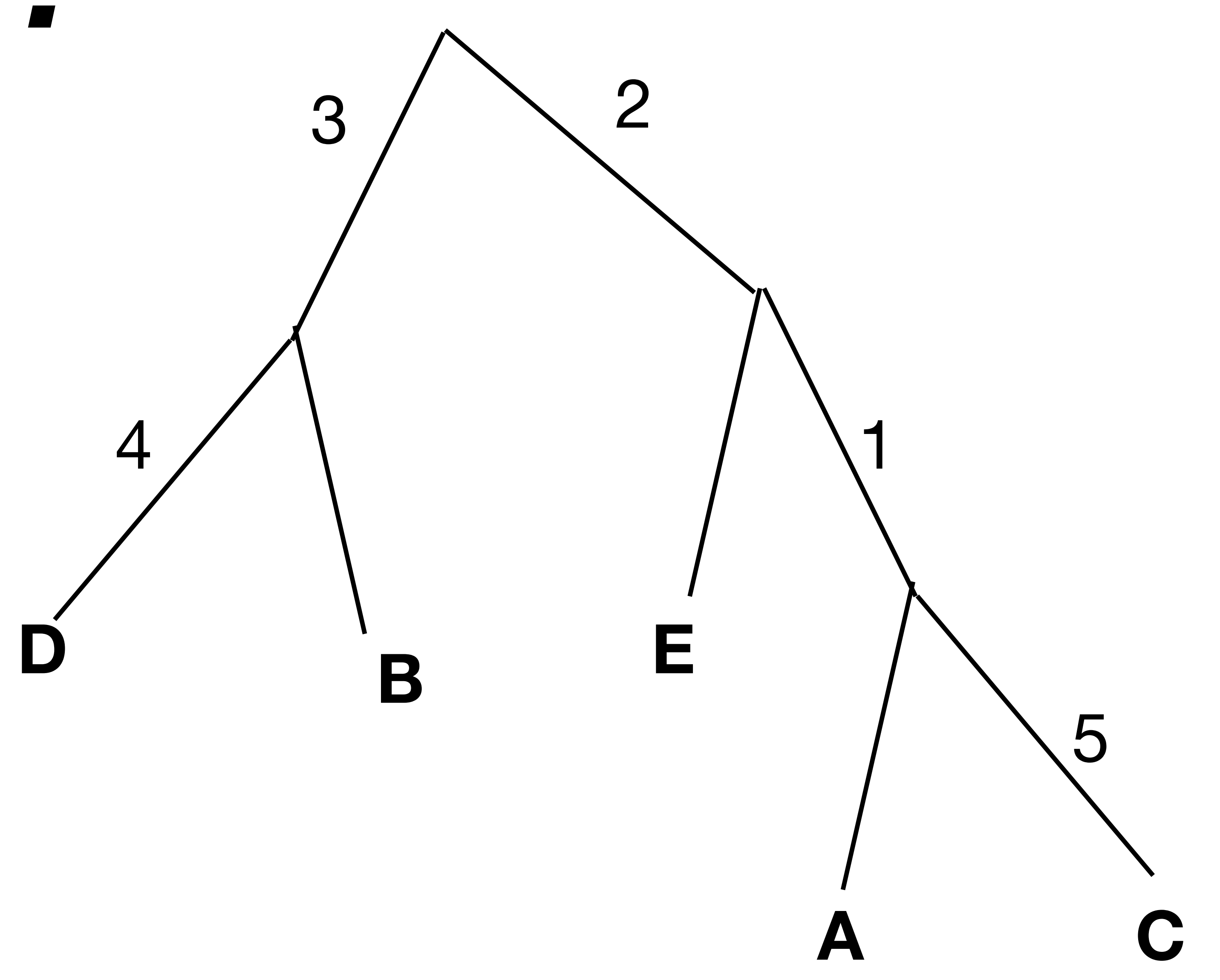
M	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	0
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	0



Parsimony

M'	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	**1**
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	0

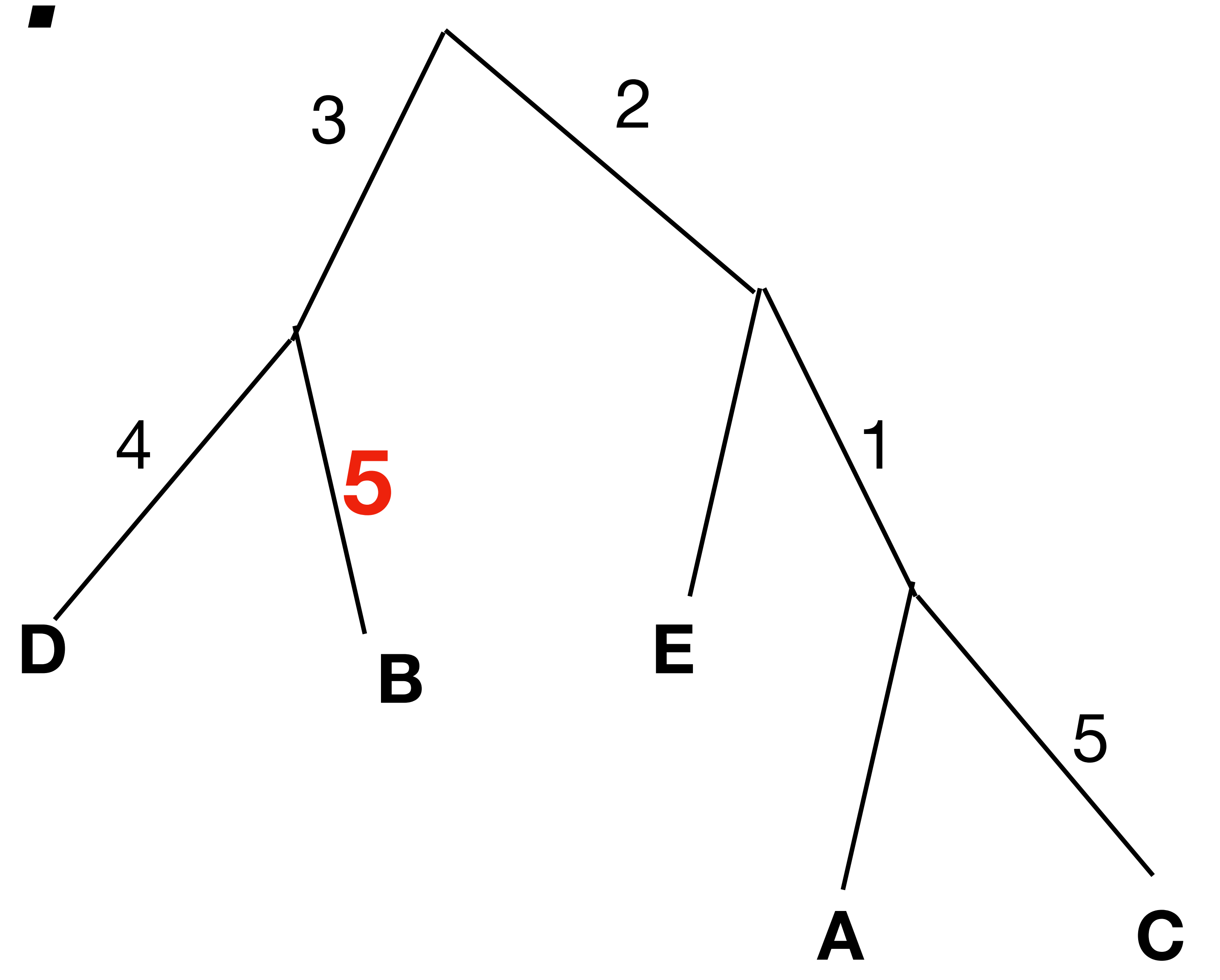
T'?



Parsimony

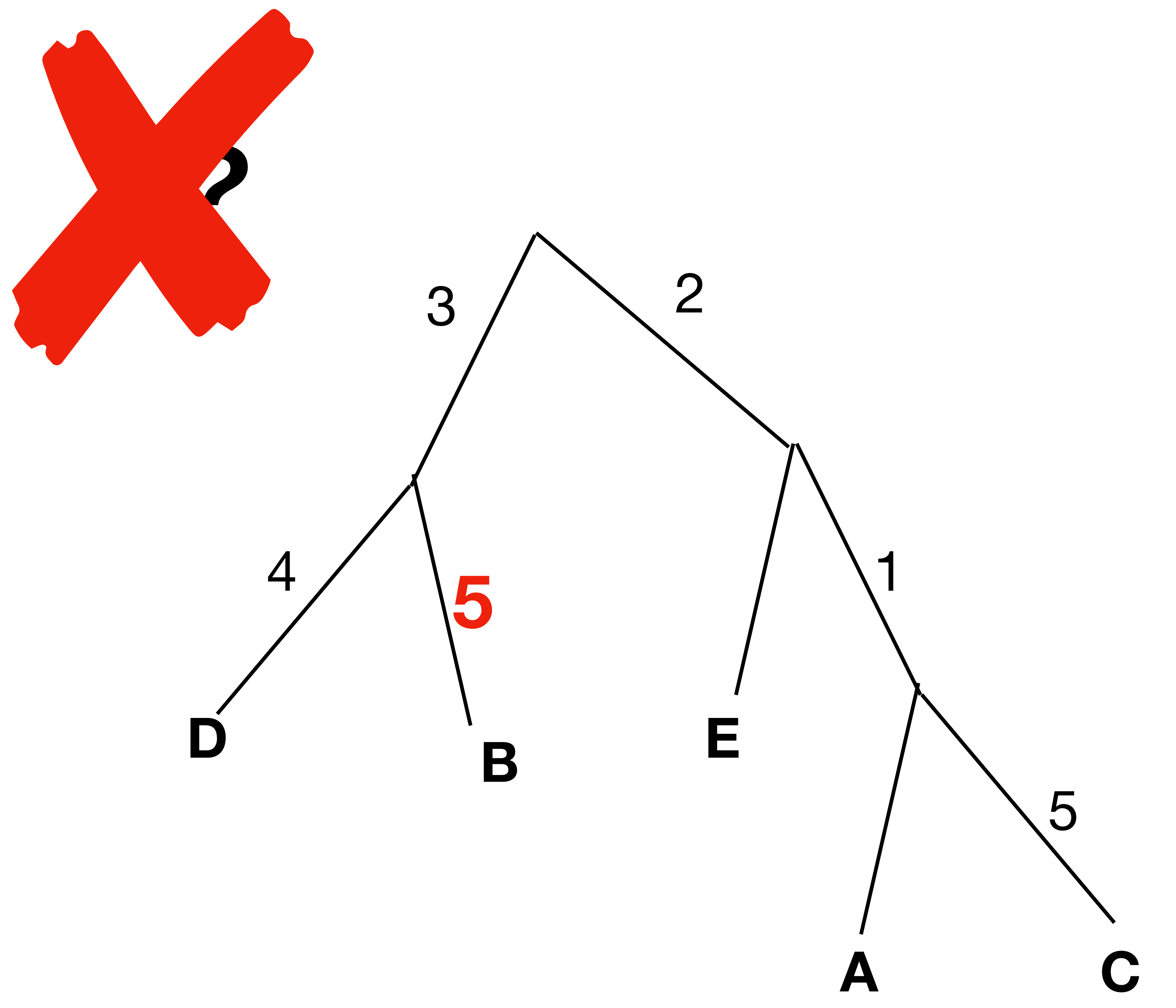
M'	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	0

T'?



Parsimony

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



This violates the definition
since column *5* now labels two edges!

Parsimony

The **perfect phylogeny** problem

- Given an *n* by *m* binary matrix *M*, determine if a phylogenetic tree exists if so, find it.

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or
- one is a subset of the other.

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* *(O_i)* are disjoint from those in *j* *(O_j),* or
- one is a subset of the other.

Proof (sketch)

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or
- one is a subset of the other.

Proof (sketch)

- (Tree→Columns) Assume there is a tree *T, f*or any two columns *i* and *j*, and let e_i be the edge where *i* changes states (similarly for e_j), either:

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or
- one is a subset of the other.

Proof (sketch)

- (Tree→Columns) Assume there is a tree *T, f*or any two columns *i* and *j*, and let e_i be the edge where *i* changes states (similarly for e_j), either:
    - $e_i=e_j$, in which case $O_i=O_j$,

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or
- one is a subset of the other.

Proof (sketch)

- (Tree→Columns) Assume there is a tree *T, f*or any two columns *i* and *j*, and let e_i be the edge where *i* changes states (similarly for e_j), either:
  - e_i=e_j, in which case O_i=O_j,
  - e_i is on the path from the root to e_j, in which case $O_i \subseteq O_j$

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or
- one is a subset of the other.

Proof (sketch)

- (Tree→Columns) Assume there is a tree *T, f*or any two columns *i* and *j*, and let e_i be the edge where *i* changes states (similarly for e_j), either:
  - e_i=e_j, in which case O_i=O_j,
  - e_i is on the path from the root to e_j, in which case $O_i \subseteq O_j$
  - e_j is on the path from the root to e_i, in which case $O_j \subseteq O_i$

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or
- one is a subset of the other.

Proof (sketch)

- (Tree→Columns) Assume there is a tree *T, f*or any two columns *i* and *j*, and let e_i be the edge where *i* changes states (similarly for e_j), either:
  - e_i=e_j, in which case O_i=O_j,
  - e_i is on the path from the root to e_j, in which case $O_i \subseteq O_j$
  - e_j is on the path from the root to e_i, in which case $O_j \subseteq O_i$
  - the paths from the root diverge before either e_i or e_j, in which case $O_i \cap O_j = \varnothing$

Parsimony

Theorem Matrix *M* has a phylogenetic tree iff for every pair of columns *i, j* either:

- the set of rows that contain 1's in *i* (O_i) are disjoint from those in *j* (O_j), or
- one is a subset of the other.

Proof (sketch)

- (Tree→Columns) Assume there is a tree *T, f*or any two columns *i* and *j*, and let e_i be the edge where *i* changes states (similarly for e_j), either:
  - $e_i = e_j$, in which case $O_i = O_j$,
  - e_i is on the path from the root to e_j, in which case $O_i \subseteq O_j$
  - e_j is on the path from the root to e_i, in which case $O_j \subseteq O_i$
  - the paths from the root diverge before either e_i or e_j, in which case $O_i \cap O_j = \varnothing$
- (Columns→Tree)
  - using similar arguments above you can construct a tree such that given that a pair of columns is disjoint or containing you can place them in the tree either ahead of or on a separate branch from the other.
  - this also leads to a method for tree construction.

Tree Compatibility

Definition A phylogenetic tree *T*' is a *refinement* of *T* if *T* can be obtained by a series of contractions of edges in *T*'.

- *T*' contains more information than *T*, but still agrees with the evolutionary history.

Definition Trees T_1 *and* T_2 are *compatible* if there exists some phylogenetic tree T_3 refining both.

Tree compatibility problem Given phylogenetic trees T_1 and T_2 determine if the two are compatible.

Tree Compatibility

Tree compatibility problem Given phylogenetic trees T_1 and T_2 determine if the two are compatible.

Assuming T_1 has *n* internal nodes and *m* leafs. Build M_1 with *m* rows and *n* columns, and let let $M_1(i,j)$ be *1* if leaf *i* is in the subtree rooted at node *j*. (similarly for M_2 from T_2).

Create matrix M_3 as the concatenation of the columns of M_1 and M_2.

Theorem T_1 *and* T_2 are compatible iff there is a phylogenetic tree for M_3.

Construction Algorithms

Up to now, what has been examined are idealized models in decreasing strictness.

Since the data we get from natural sources (be it biology, chemistry, engineering applications, etc.), we need heuristics of some sort.

Two major classes:

- Neighbor-joining methods
- Maximum parsimony

Both work on the parsimony principles.

Maximum Parsimony

The **Maximum Parsimony Problem** (sometimes called the Large Parsimony Problem) is stated as follows:

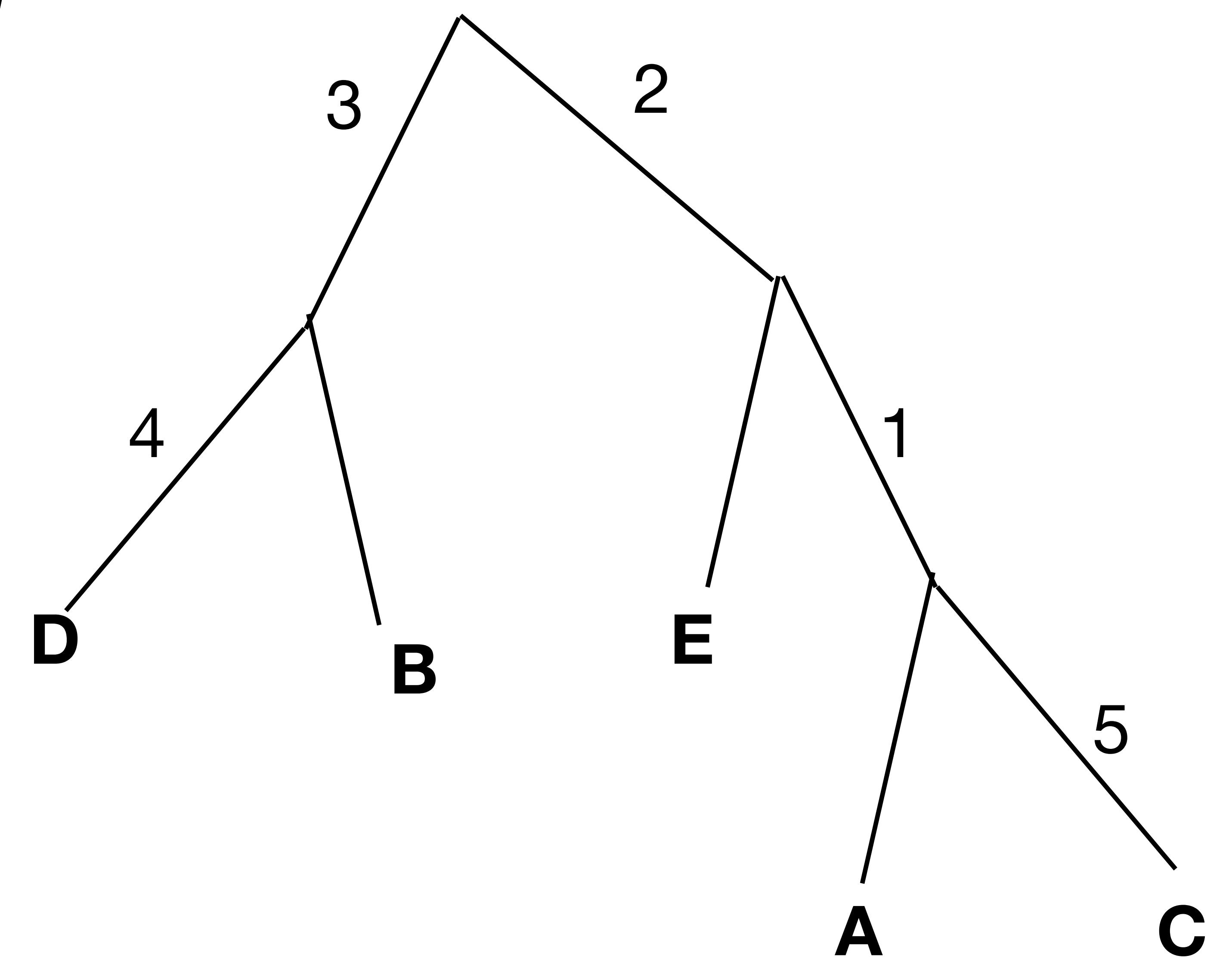
- Given a matrix *M* for a set *S* of *n* taxa
- find the tree *T* wihch is leaf labeled by *S* and minimizes the edges that are labeled by character position changes.

This problem is *NP-Hard*

- naïve solution is to enumerate all possible trees, but there are (2n-5)!!
- (here p!! = 1*3*5*p)

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

T



Maximum Parsimony

The **Maximum Parsimony Problem** (sometimes called the Large Parsimony Problem) is stated as follows:

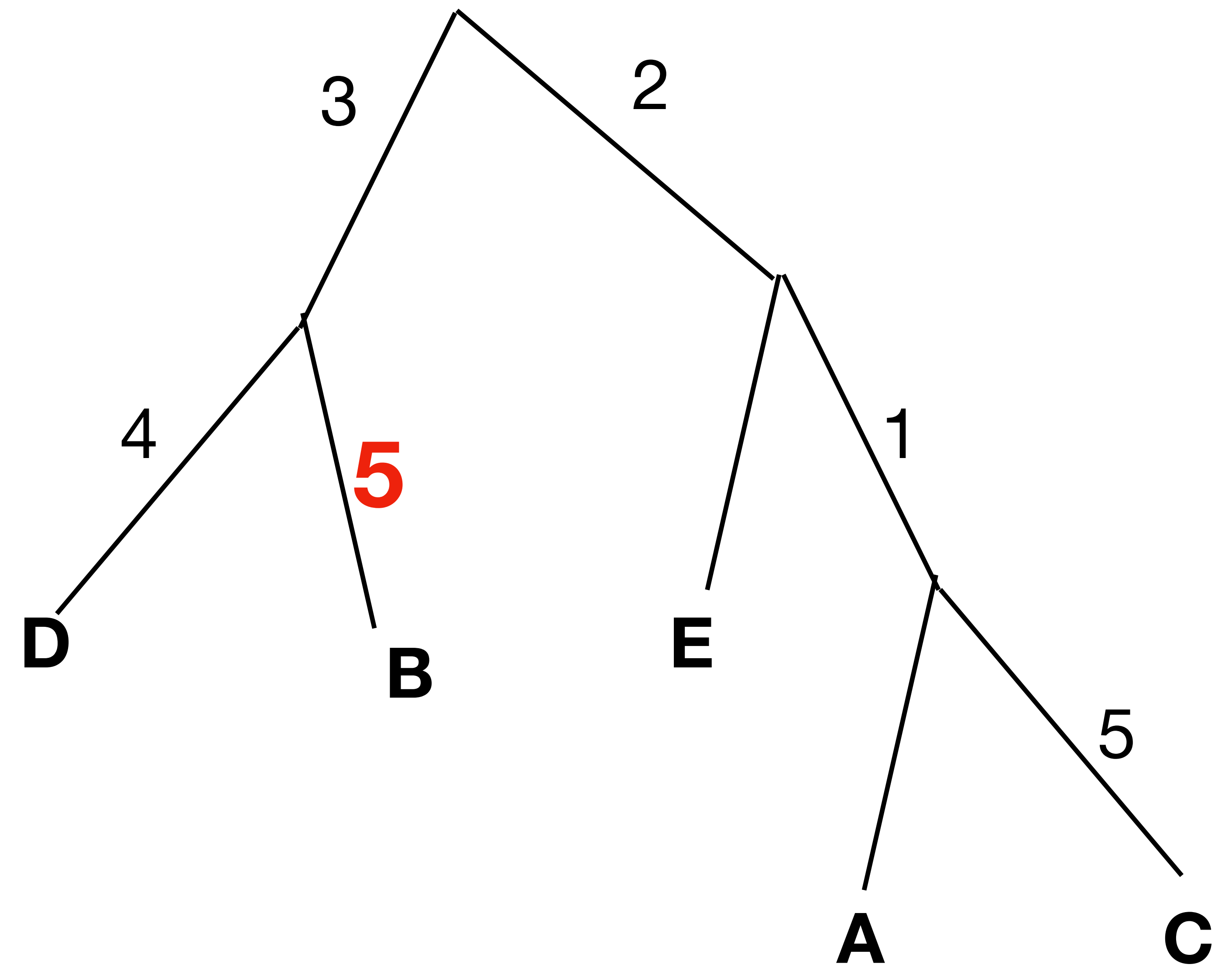
- Given a matrix *M* for a set *S* of *n* taxa
- find the tree *T* wihch is leaf labeled by *S* and minimizes the edges that are labeled by character position changes.

This problem is *NP-Hard*

- naïve solution is to enumerate all possible trees, but there are (2n-5)!!
- (here p!! = 1*3*5*p)

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

T



Maximum Parsimony

Branch and Bound -- Henry and Penny (1982)

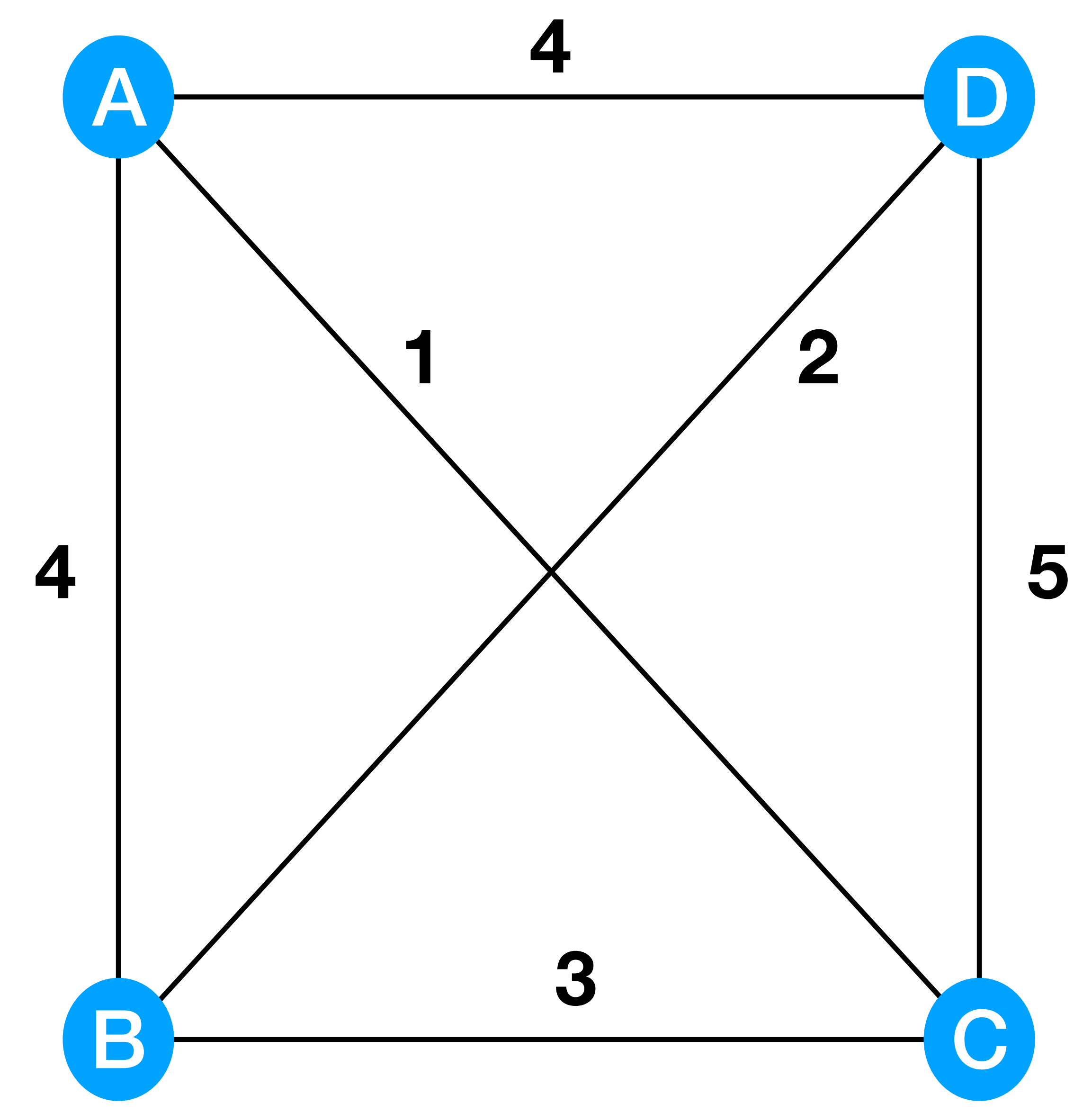
- Starting with a tree of 3 taxa (a star tree) add new taxa at each possible location and recurse.
- Since the number of mutations is monotonically increasing, stop any computational branch that cannot be optimal (based on the scores of the other computational branches).
- You can use some fast heuristic to get a starting lower bound.

Maximum Parsimony

2-Approximation Algorithm

- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.

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

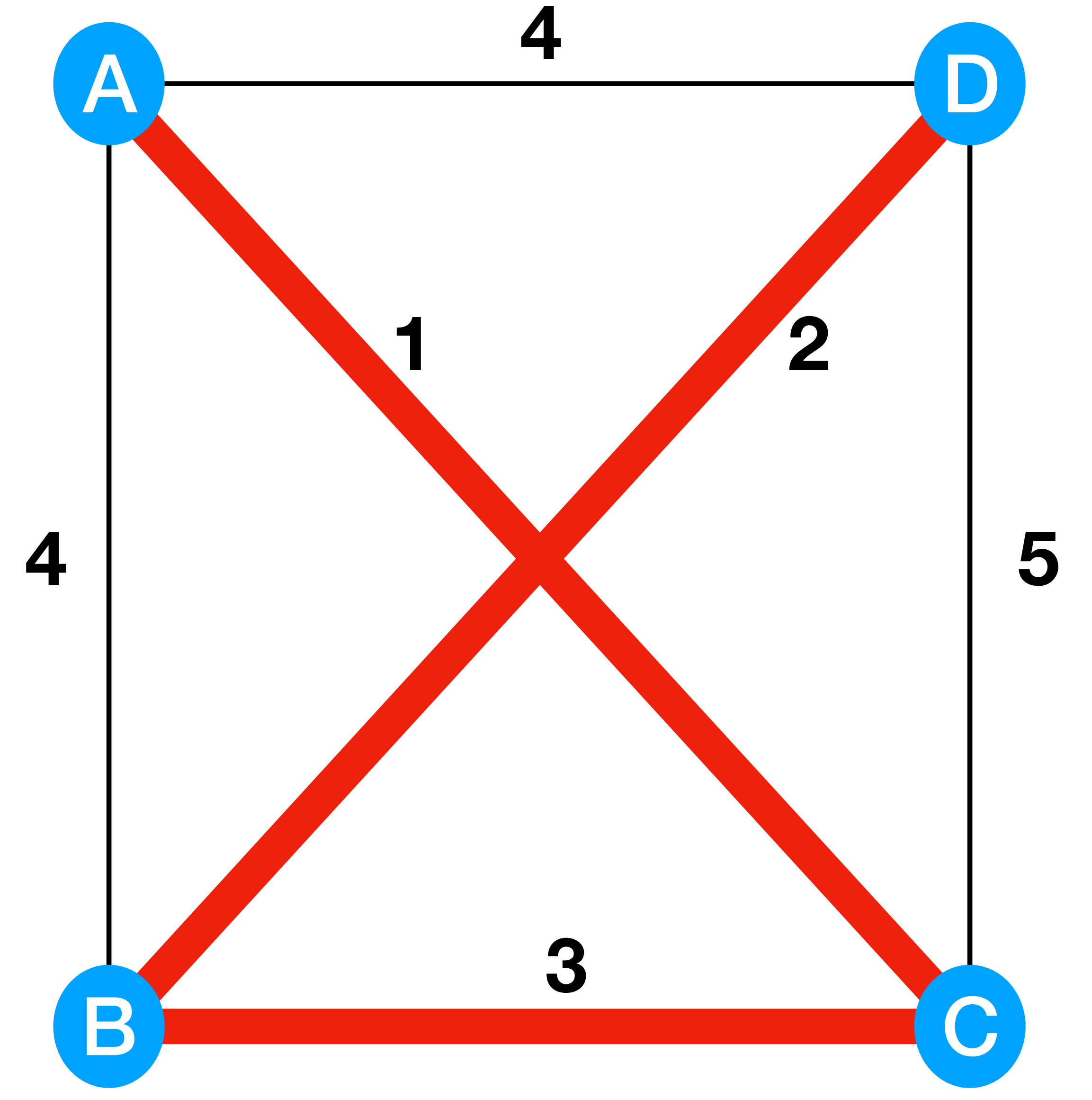


Maximum Parsimony

2-Approximation Algorithm

- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.
- Find the minimum spanning tree of the graph.

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

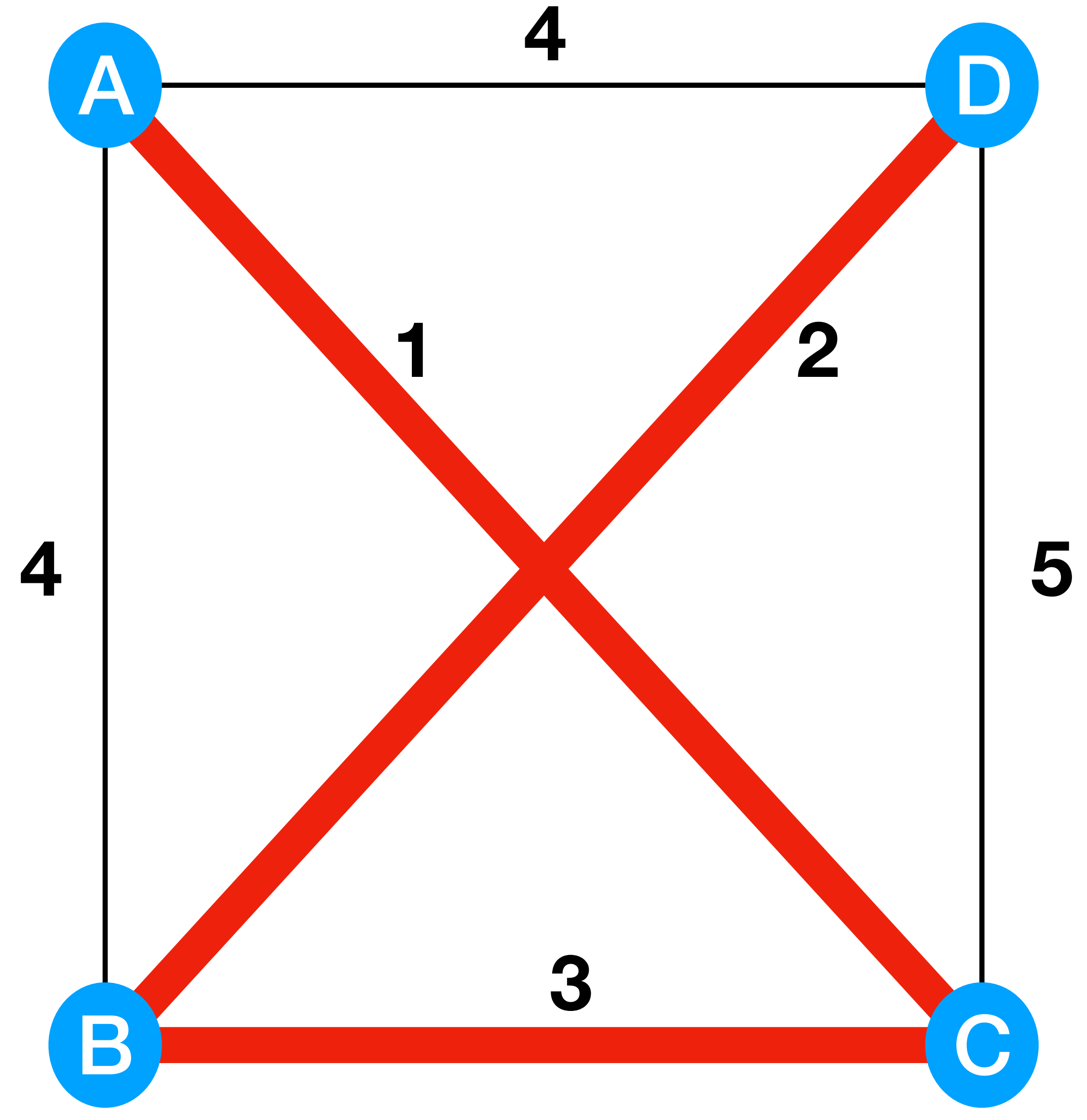


Maximum Parsimony

2-Approximation Algorithm

- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.
- Find the minimum spanning tree of the graph.
- Convert into a phylogenetic tree by adding extra edges with the taxa at the leaves.

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

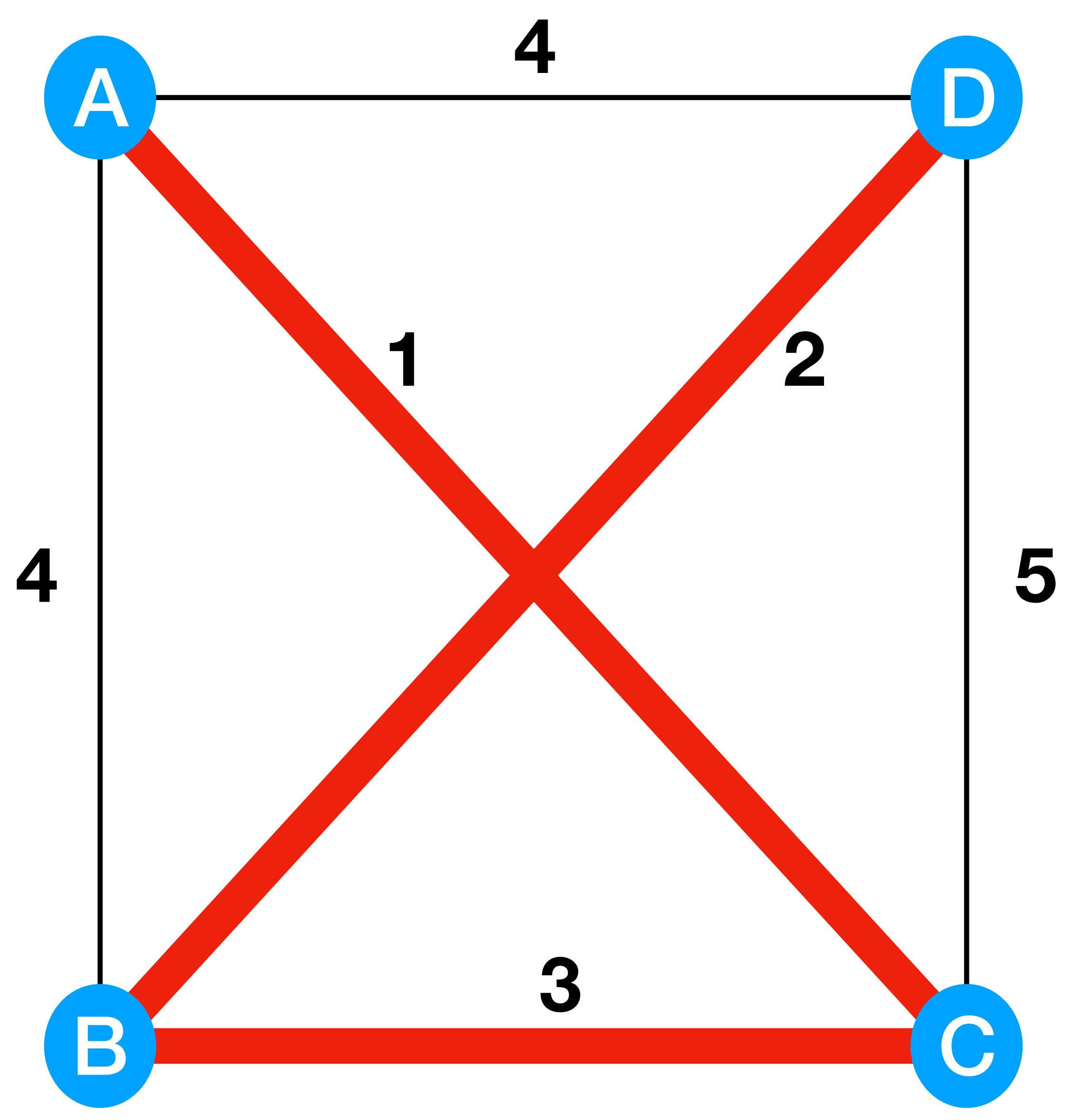


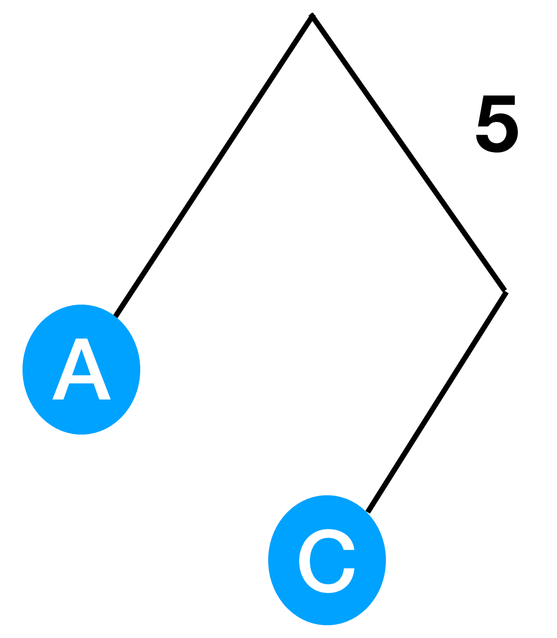
Maximum Parsimony

2-Approximation Algorithm

- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.
- Find the minimum spanning tree of the graph.
- Convert into a phylogenetic tree by adding extra edges with the taxa at the leaves.

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



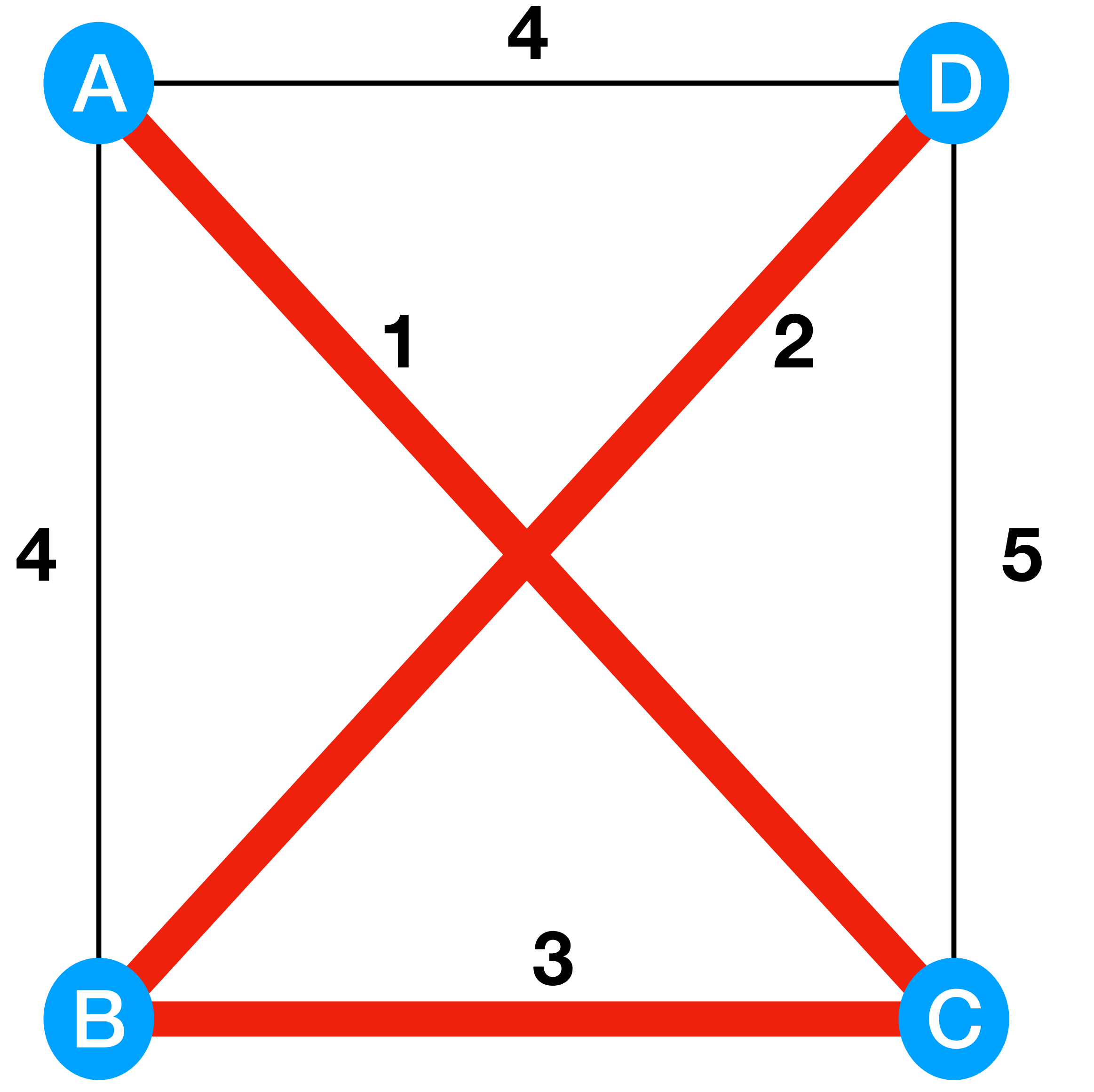


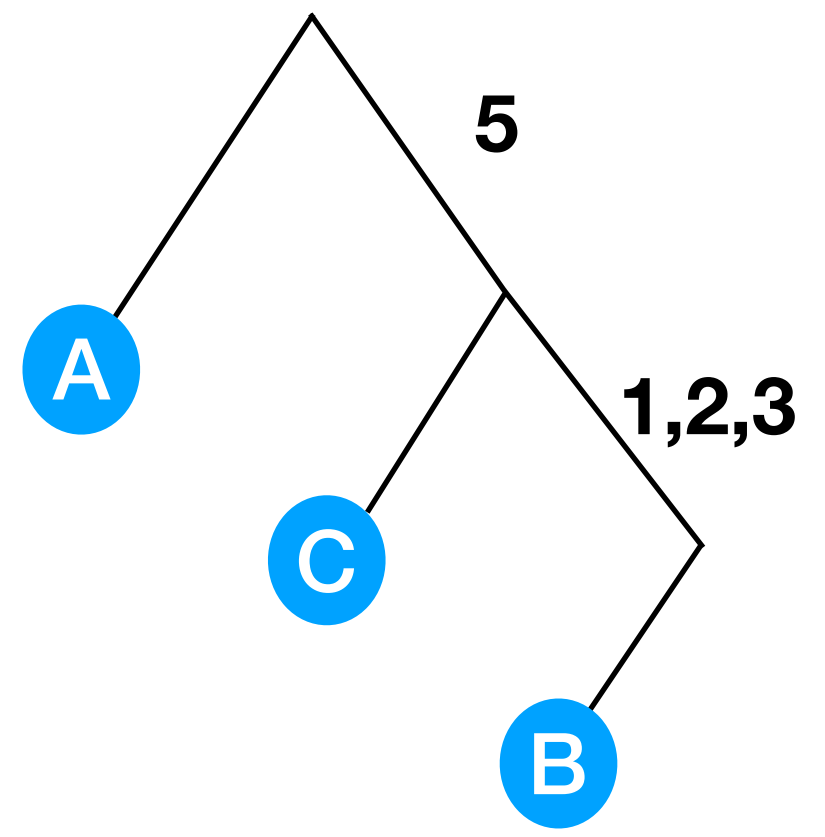
Maximum Parsimony

2-Approximation Algorithm

- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.
- Find the minimum spanning tree of the graph.
- Convert into a phylogenetic tree by adding extra edges with the taxa at the leaves.

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



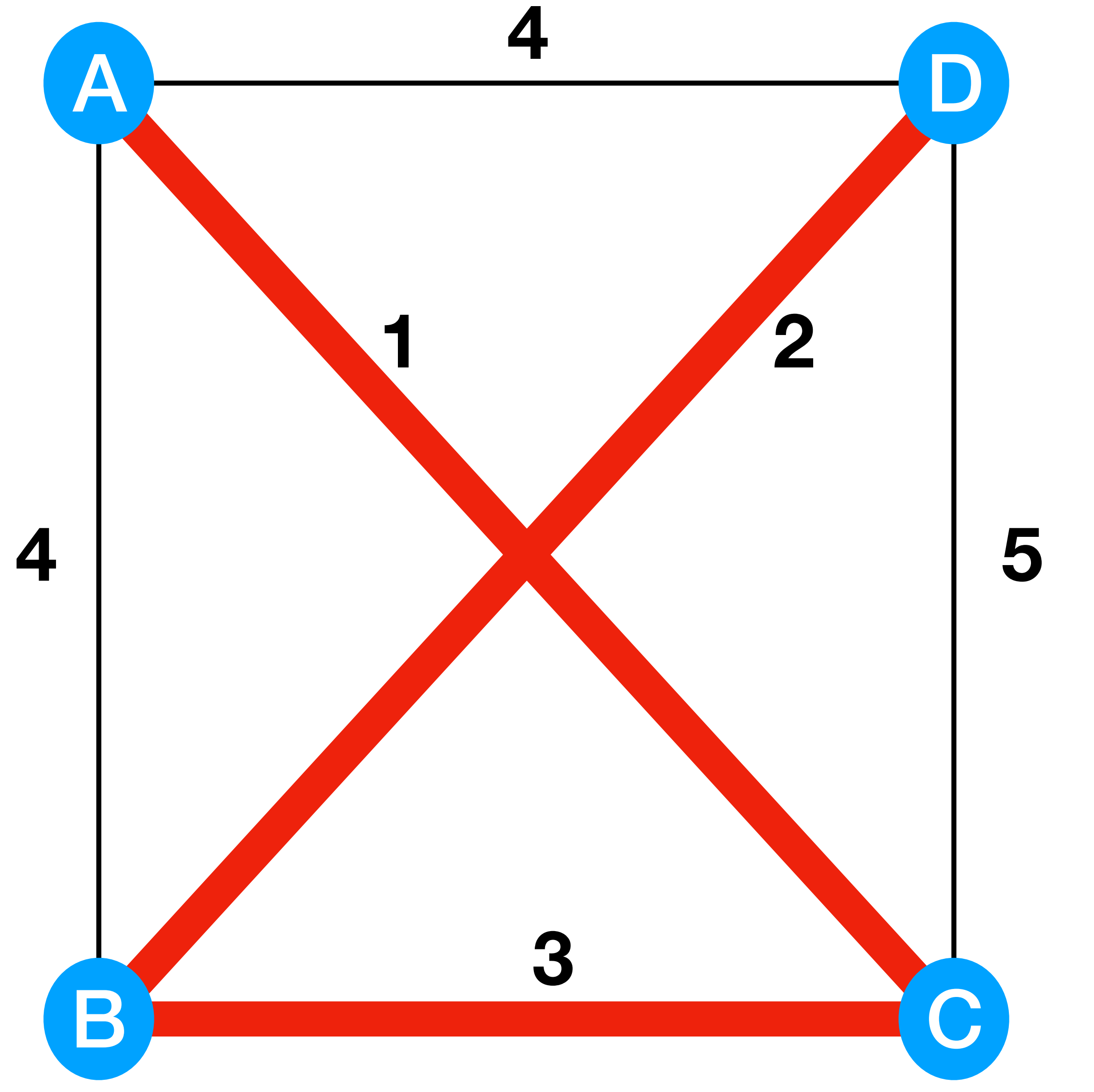


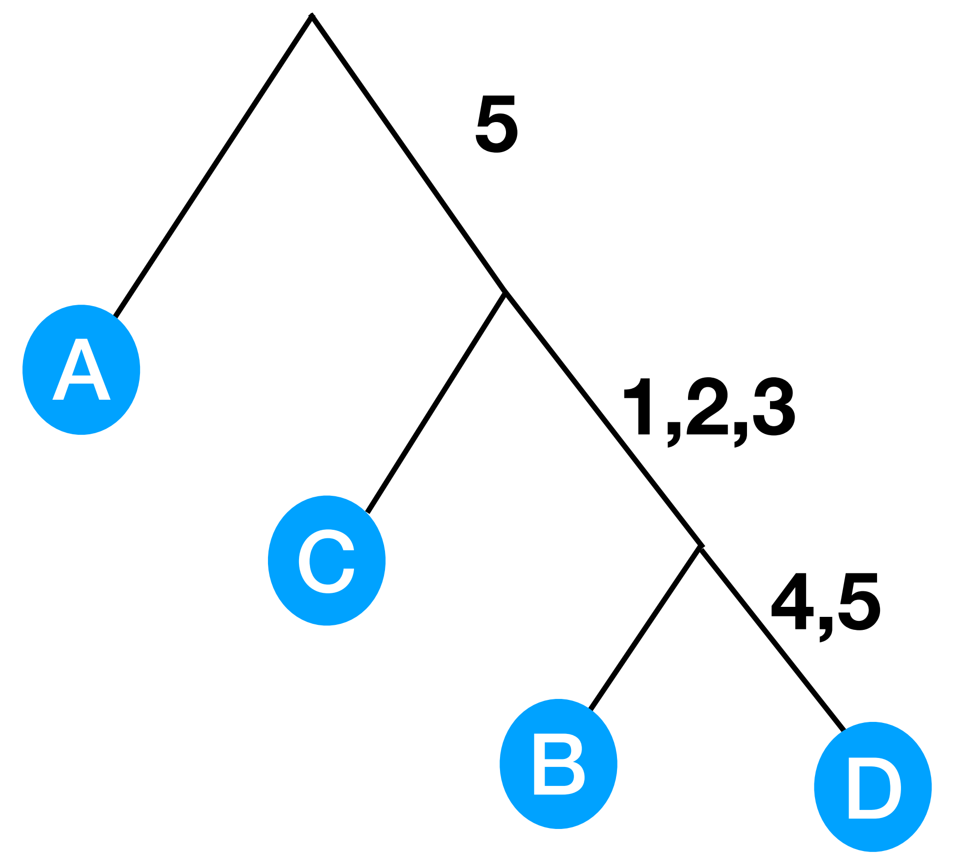
Maximum Parsimony

2-Approximation Algorithm

- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.
- Find the minimum spanning tree of the graph.
- Convert into a phylogenetic tree by adding extra edges with the taxa at the leaves.

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





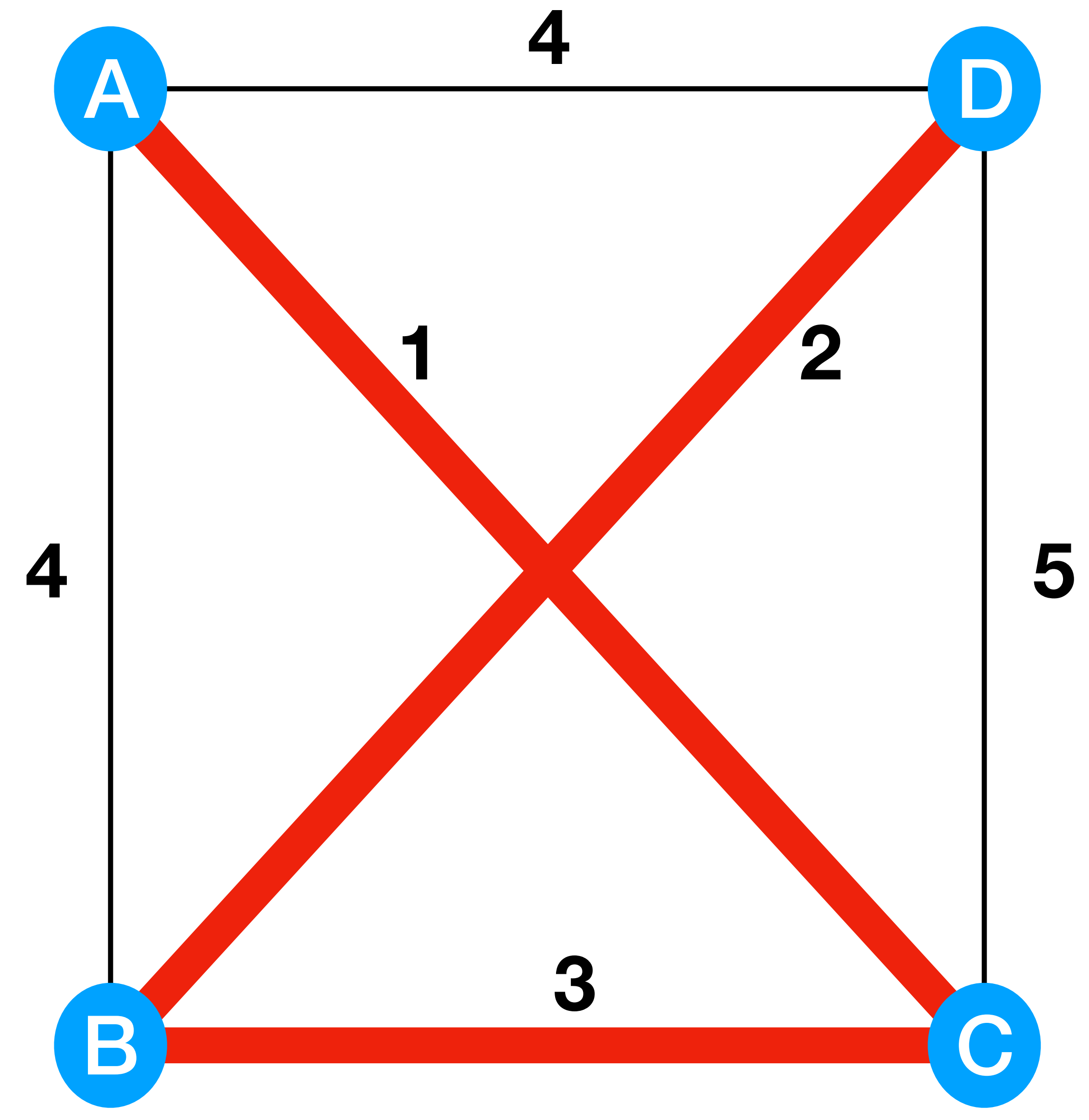
Maximum Parsimony

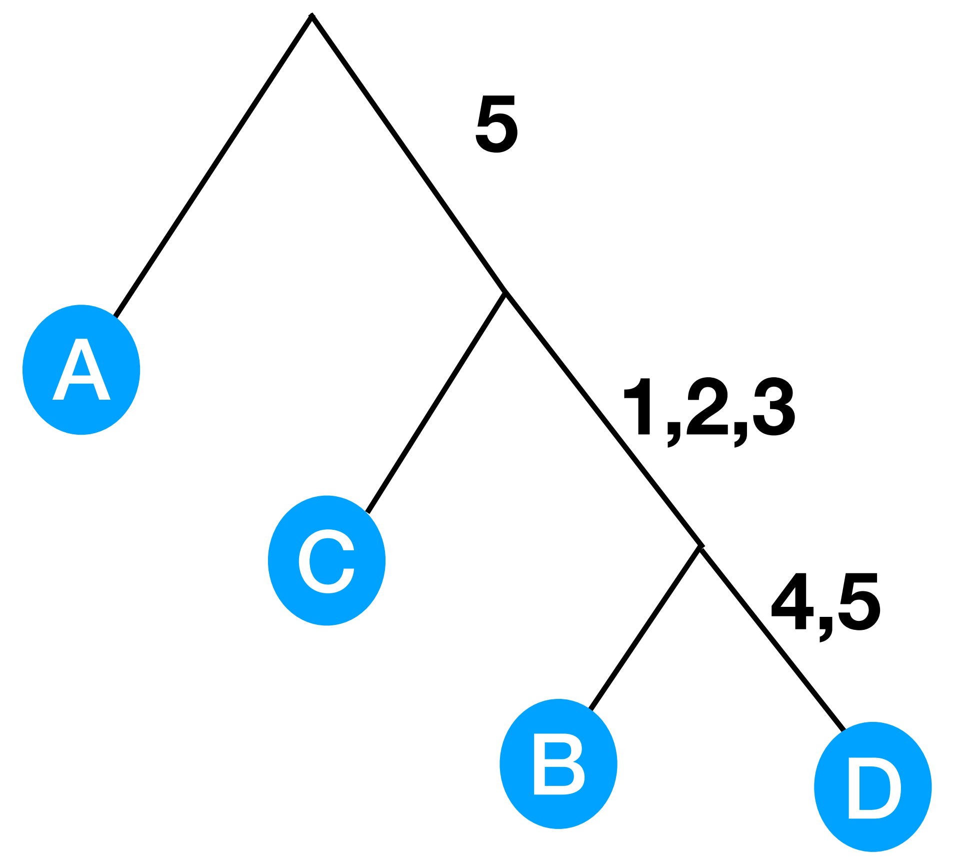
2-Approximation Algorithm

- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.
- Find the minimum spanning tree of the graph.
- Convert into a phylogenetic tree by adding extra edges with the taxa at the leaves.

Running time:

M	**1**	**2**	**3**	**4**	**5**
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	0	1	1	0





Maximum Parsimony

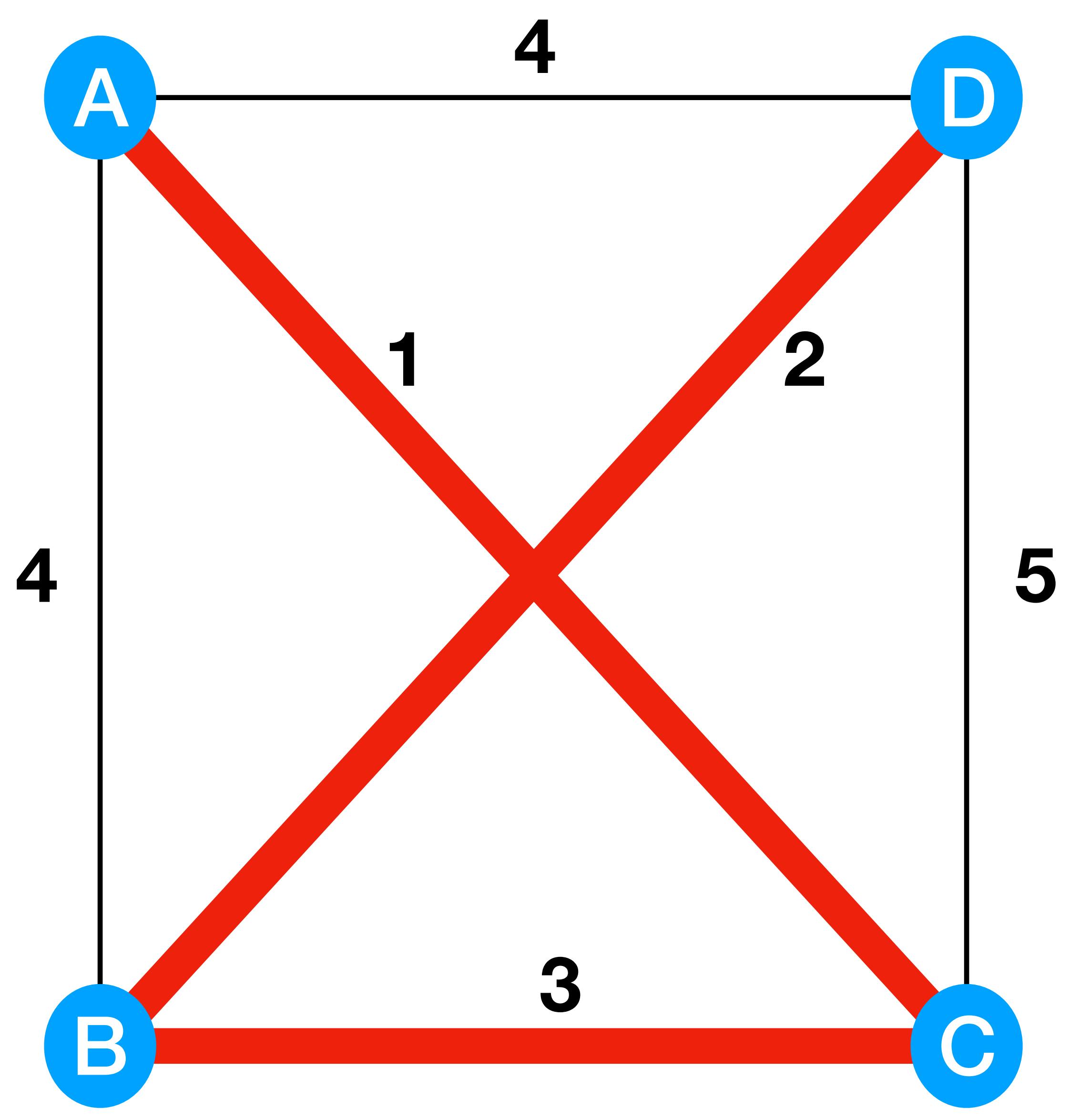
2-Approximation Algorithm

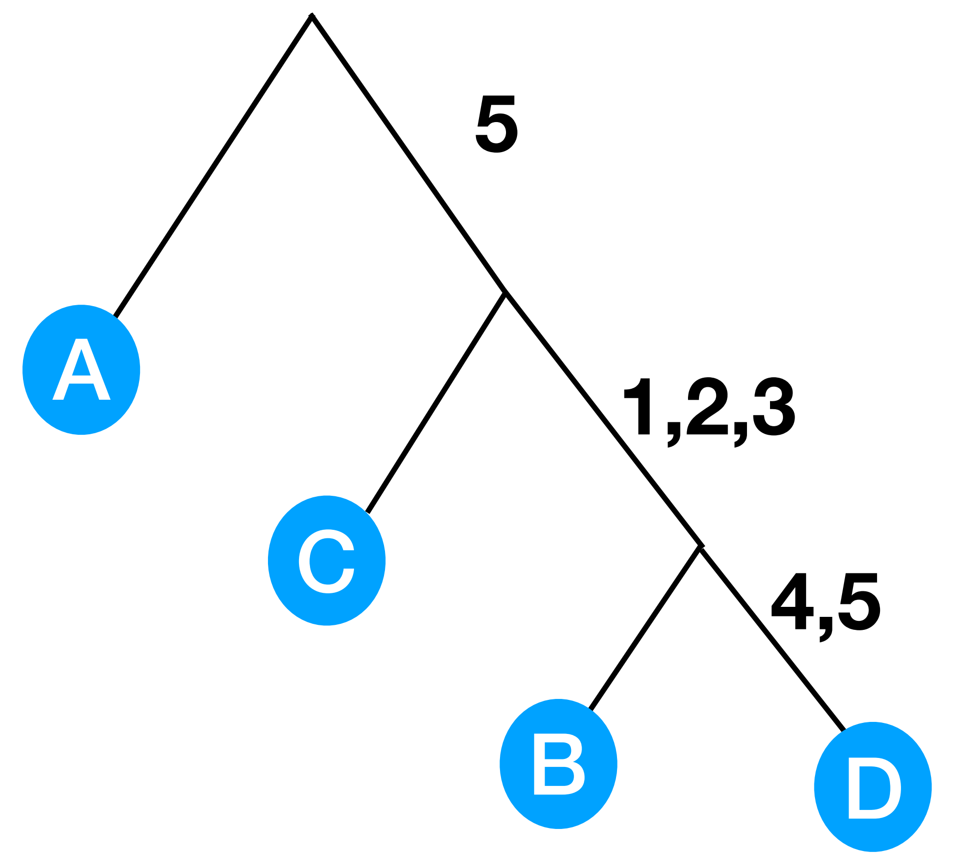
- From *M* create an undirected fully connected graph where nodes are the labels *S*, and edge weights are determined by the hamming distance between the sequences.
- Find the minimum spanning tree of the graph.
- Convert into a phylogenetic tree by adding extra edges with the taxa at the leaves.

Running time:

- O(n^2m) time, dominated by the graph construction

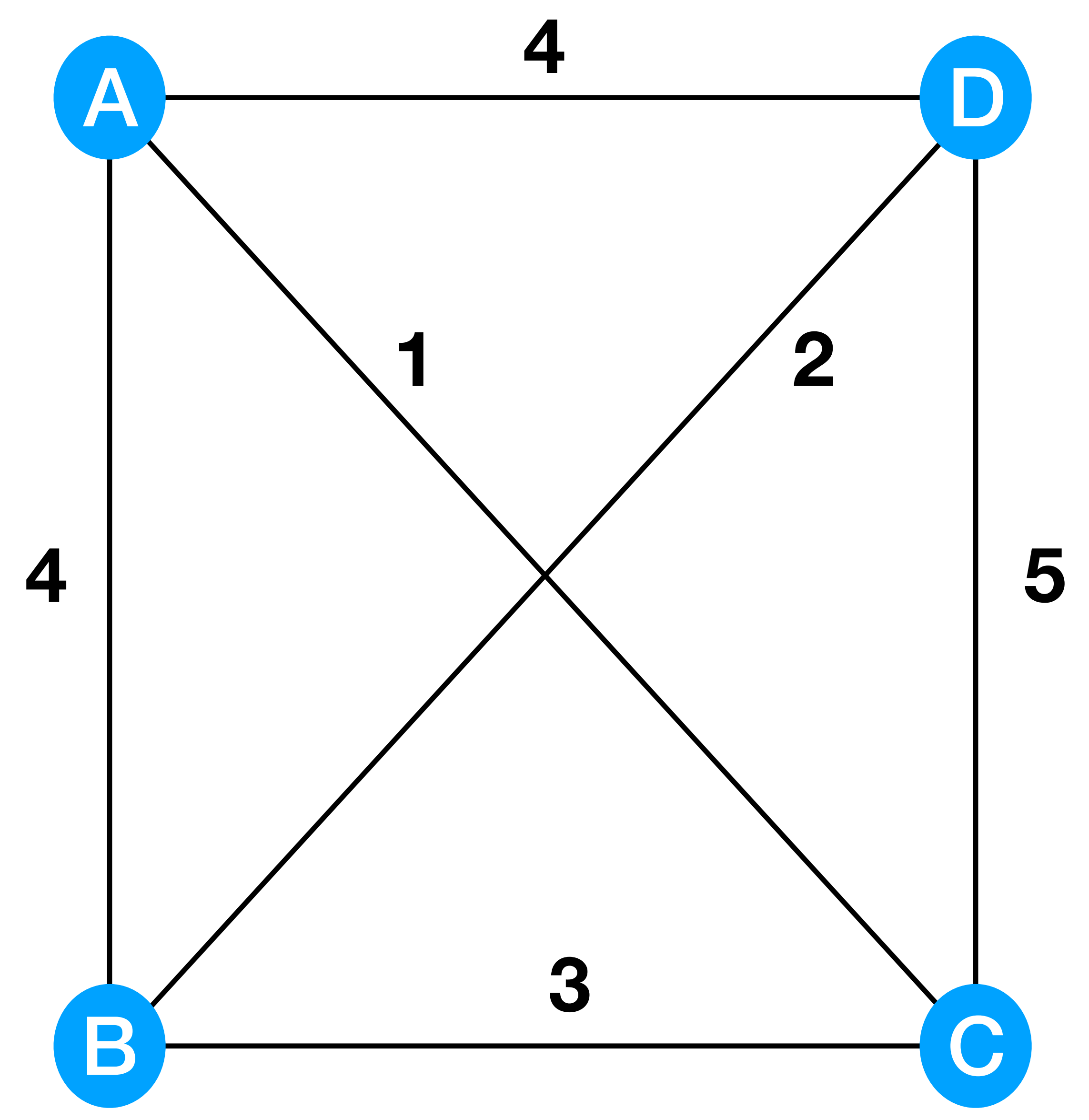
M	**1**	**2**	**3**	**4**	**5**
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	0	1	1	0





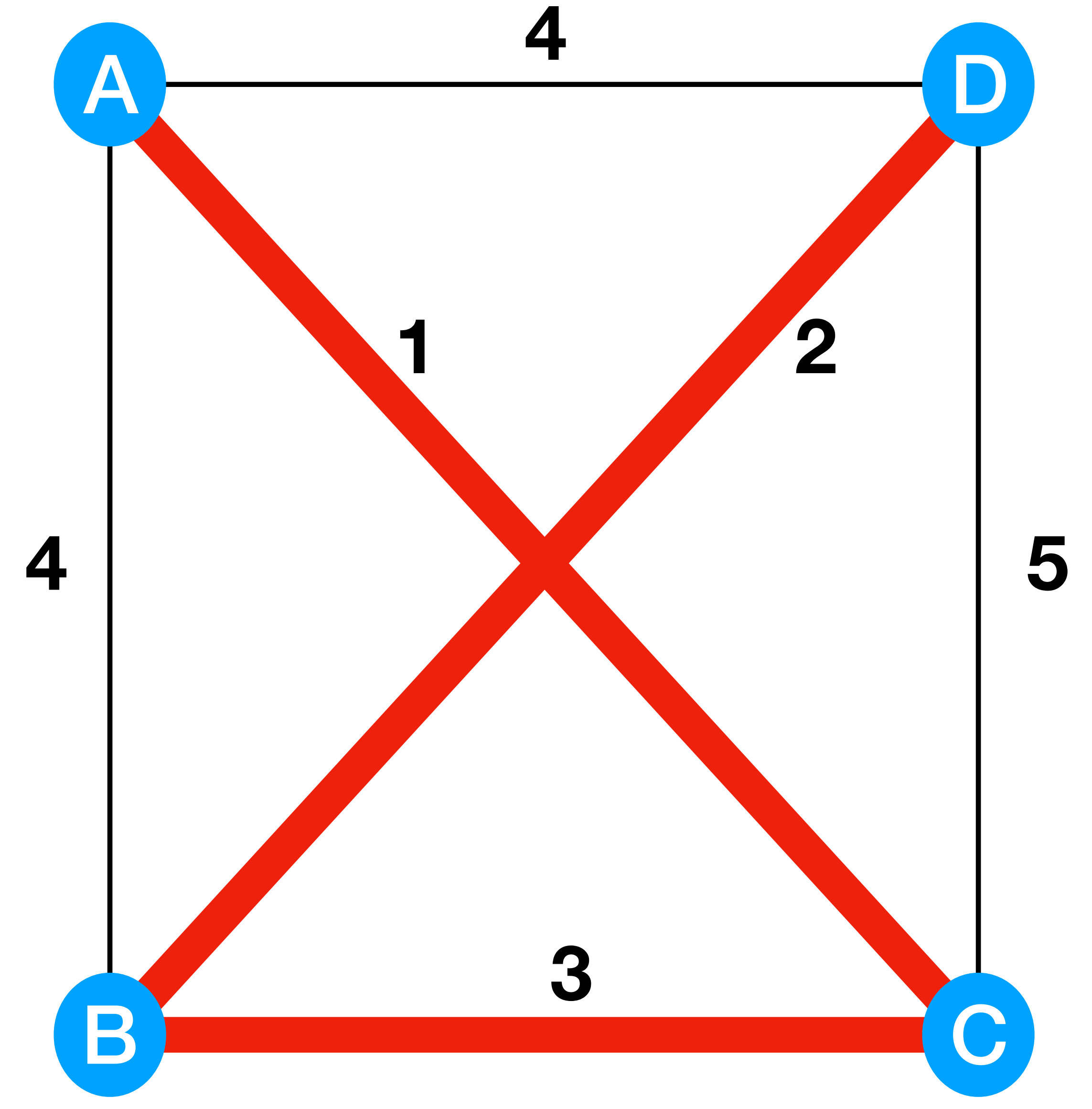
Maximum Parsimony

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



Maximum Parsimony

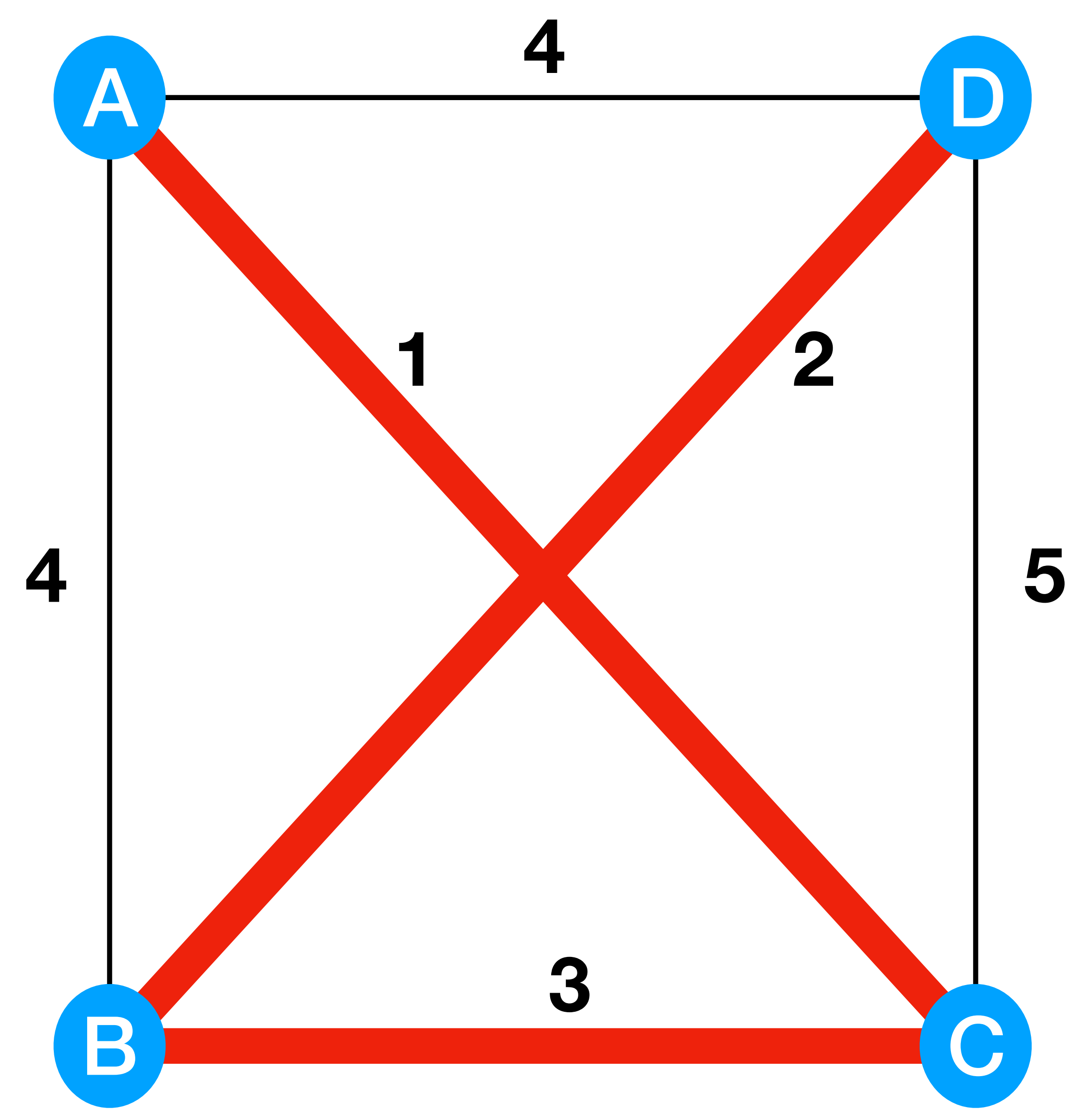
M	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	0	1	1	0



Maximum Parsimony

2-Approximation Algorithm

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

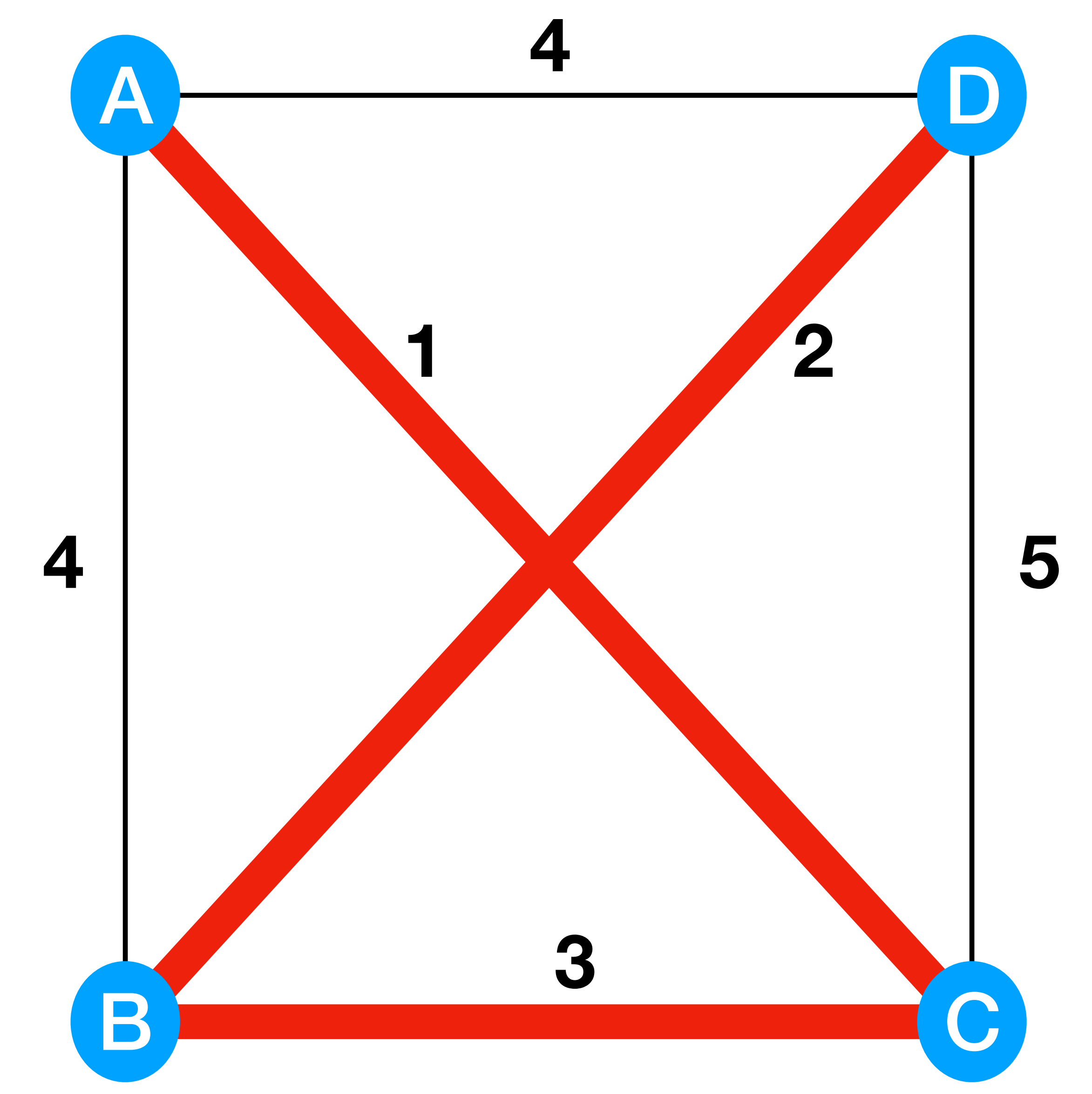


Maximum Parsimony

2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

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



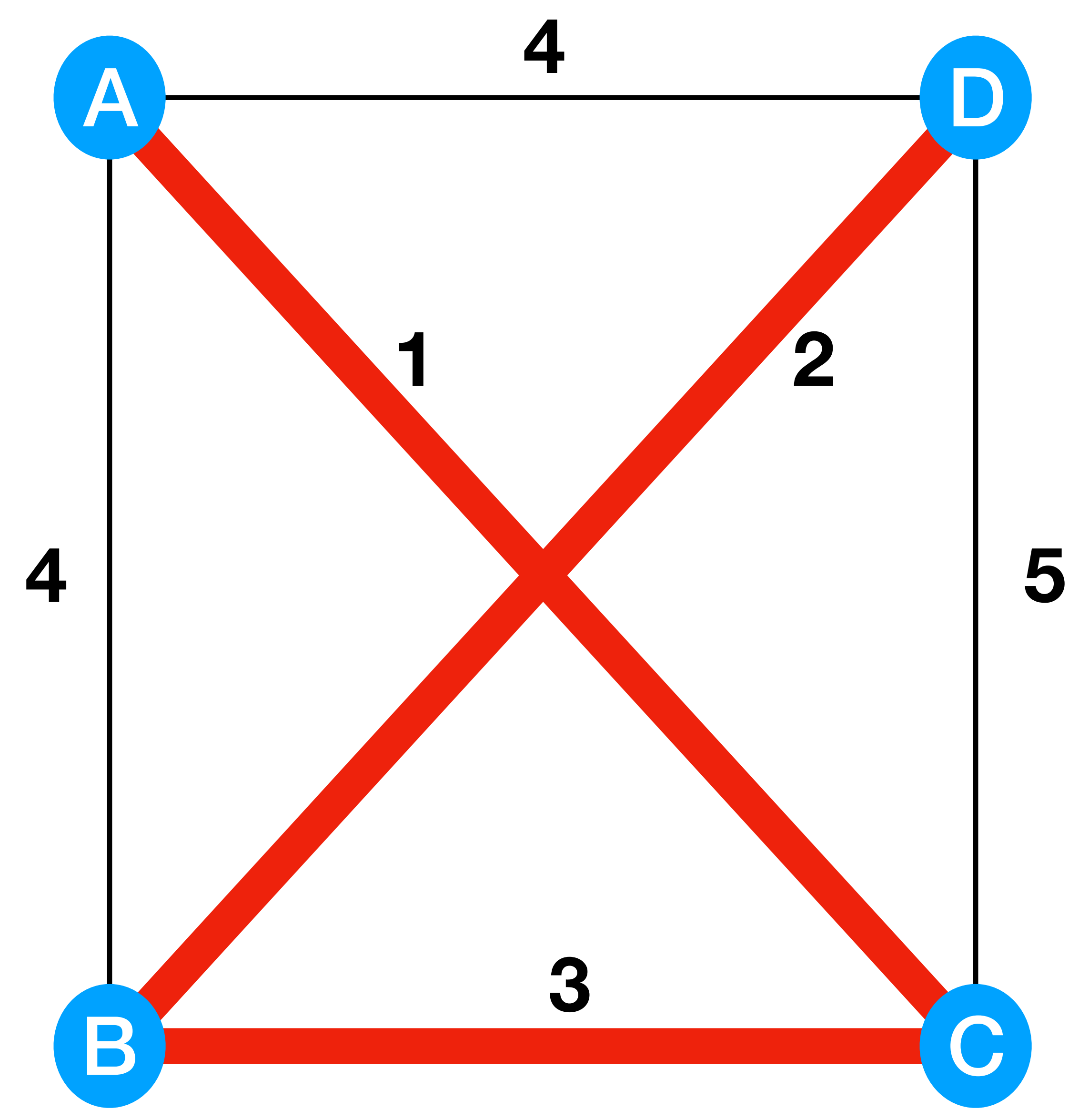
Maximum Parsimony

2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.

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



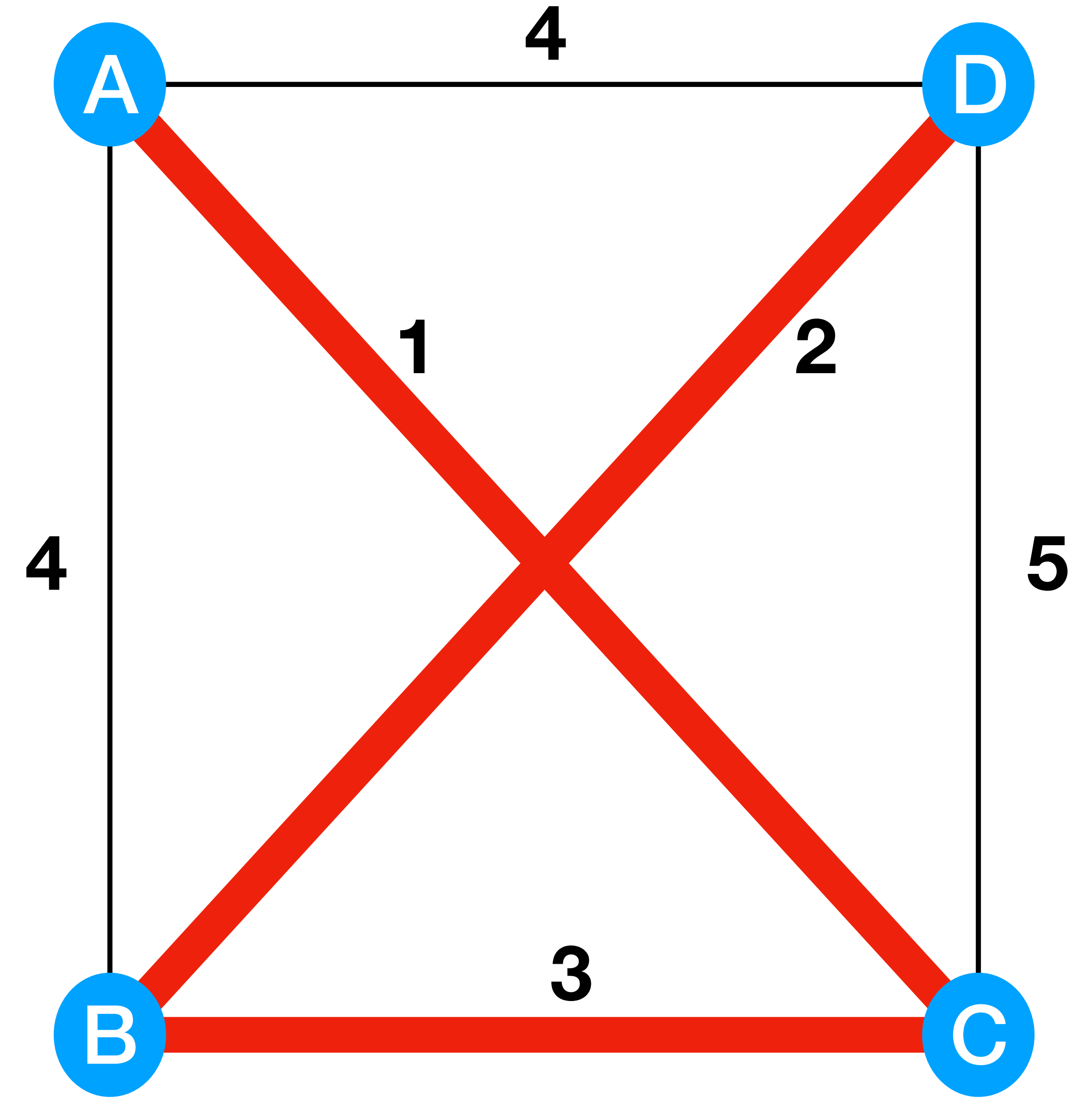
Maximum Parsimony

2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.
- Let *C* be the Euler cycle of the tree (it contains each edge twice).

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



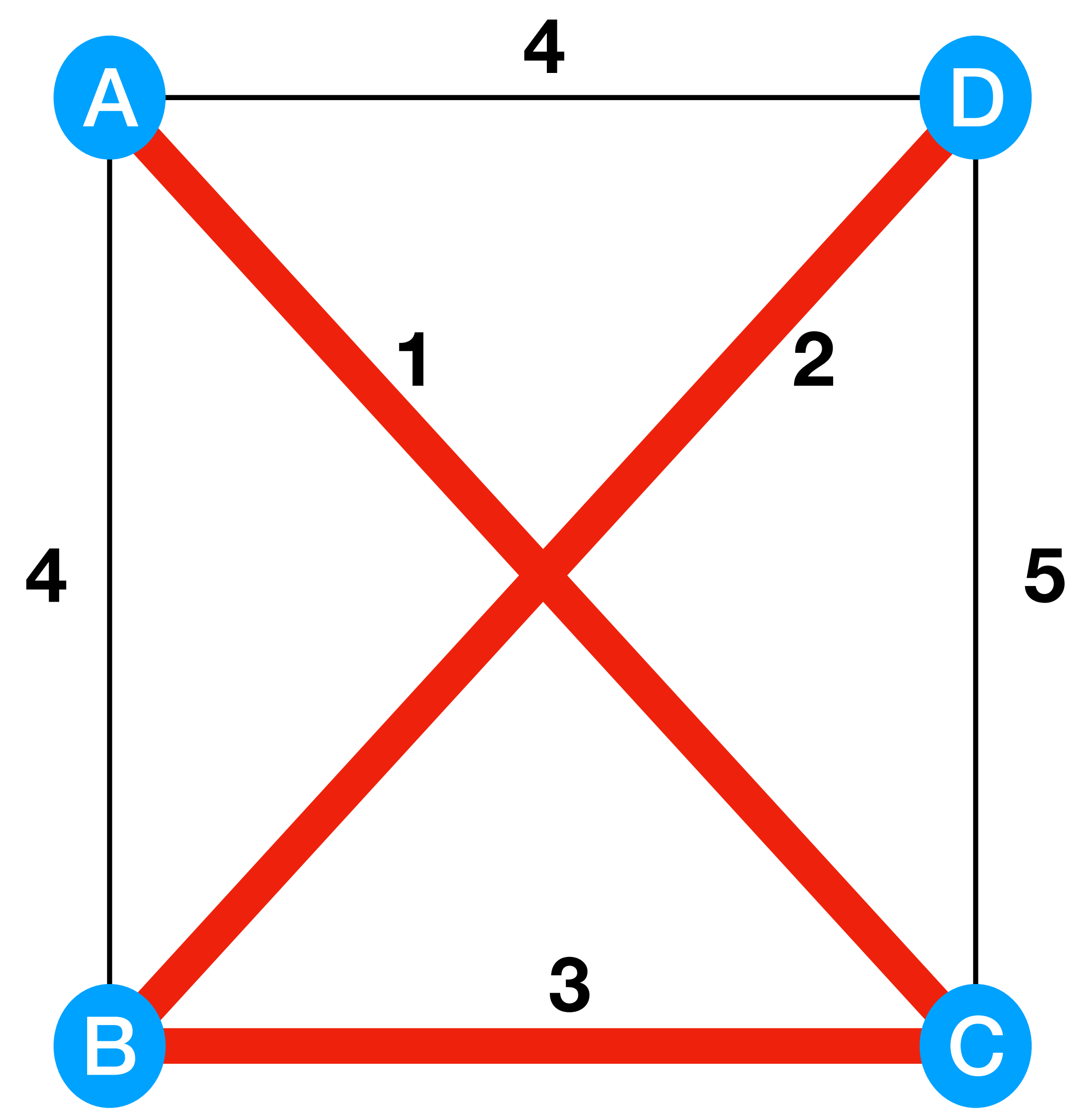
Maximum Parsimony

2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.
- Let *C* be the Euler cycle of the tree (it contains each edge twice).
- Define *w(H)* for some graph *H* to be the sum of the weights of all edges.

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



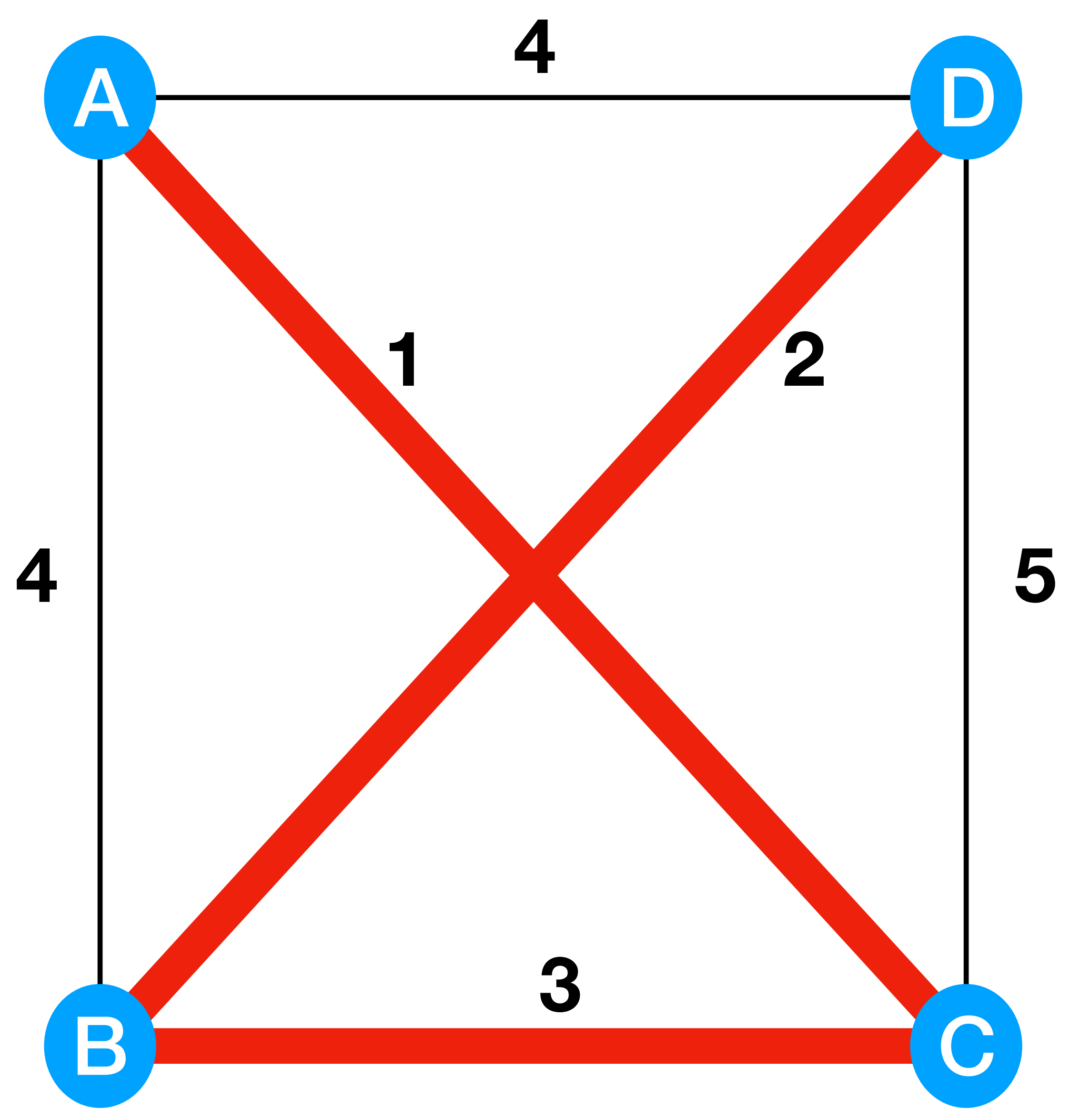
Maximum Parsimony

2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.
- Let *C* be the Euler cycle of the tree (it contains each edge twice).
- Define *w(H)* for some graph *H* to be the sum of the weights of all edges.
- Let *P* be the path containing all nodes of the graph constructed from *M* ordered by their first occurrence in *C*.

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



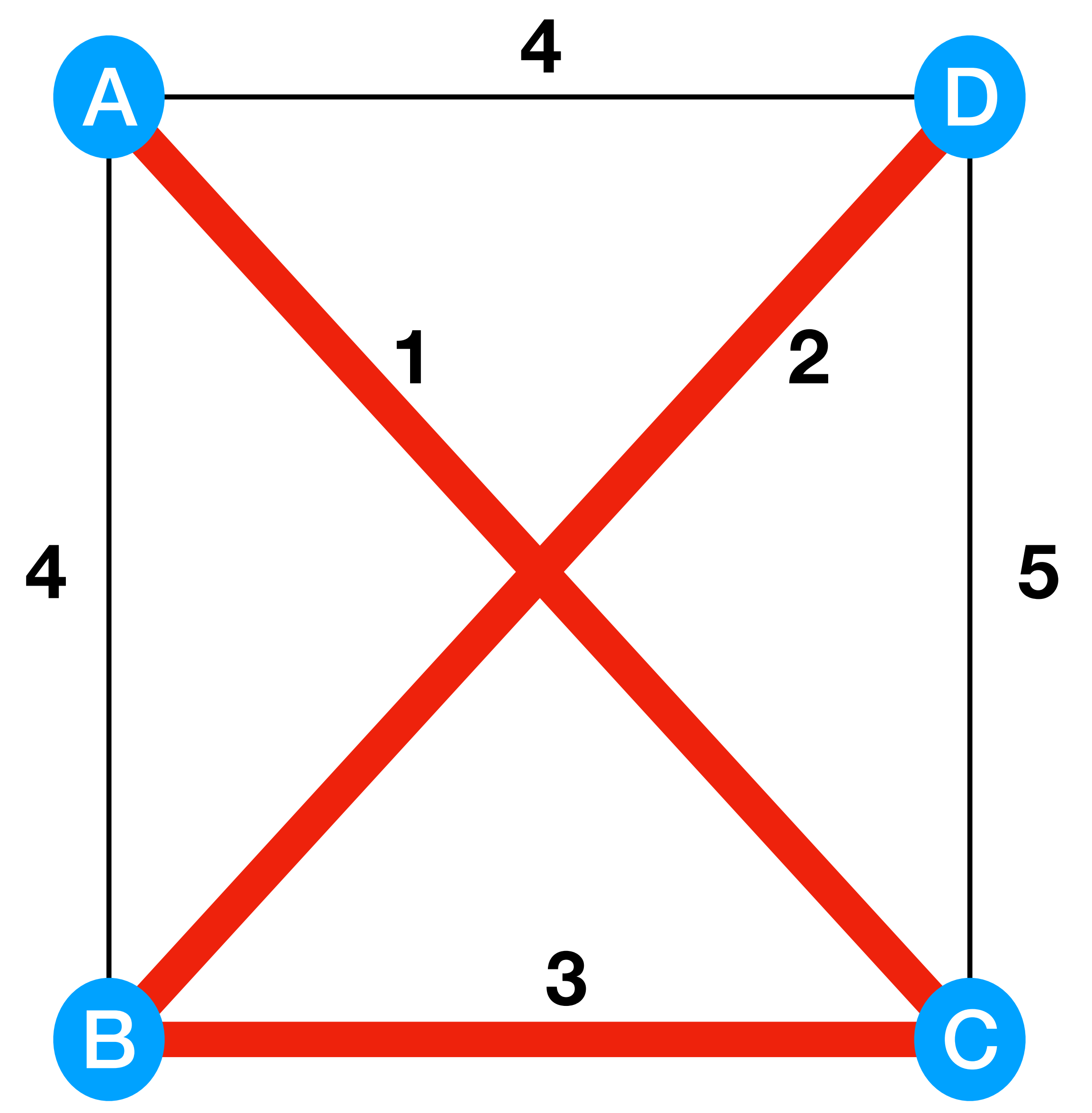
Maximum Parsimony

2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.
- Let *C* be the Euler cycle of the tree (it contains each edge twice).
- Define *w(H)* for some graph *H* to be the sum of the weights of all edges.
- Let *P* be the path containing all nodes of the graph constructed from *M* ordered by their first occurrence in *C*.
- $w(T') \leq w(P) \leq w(C) = 2w(T^*)$
  (T' is the minimum spanning tree of the graph)

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



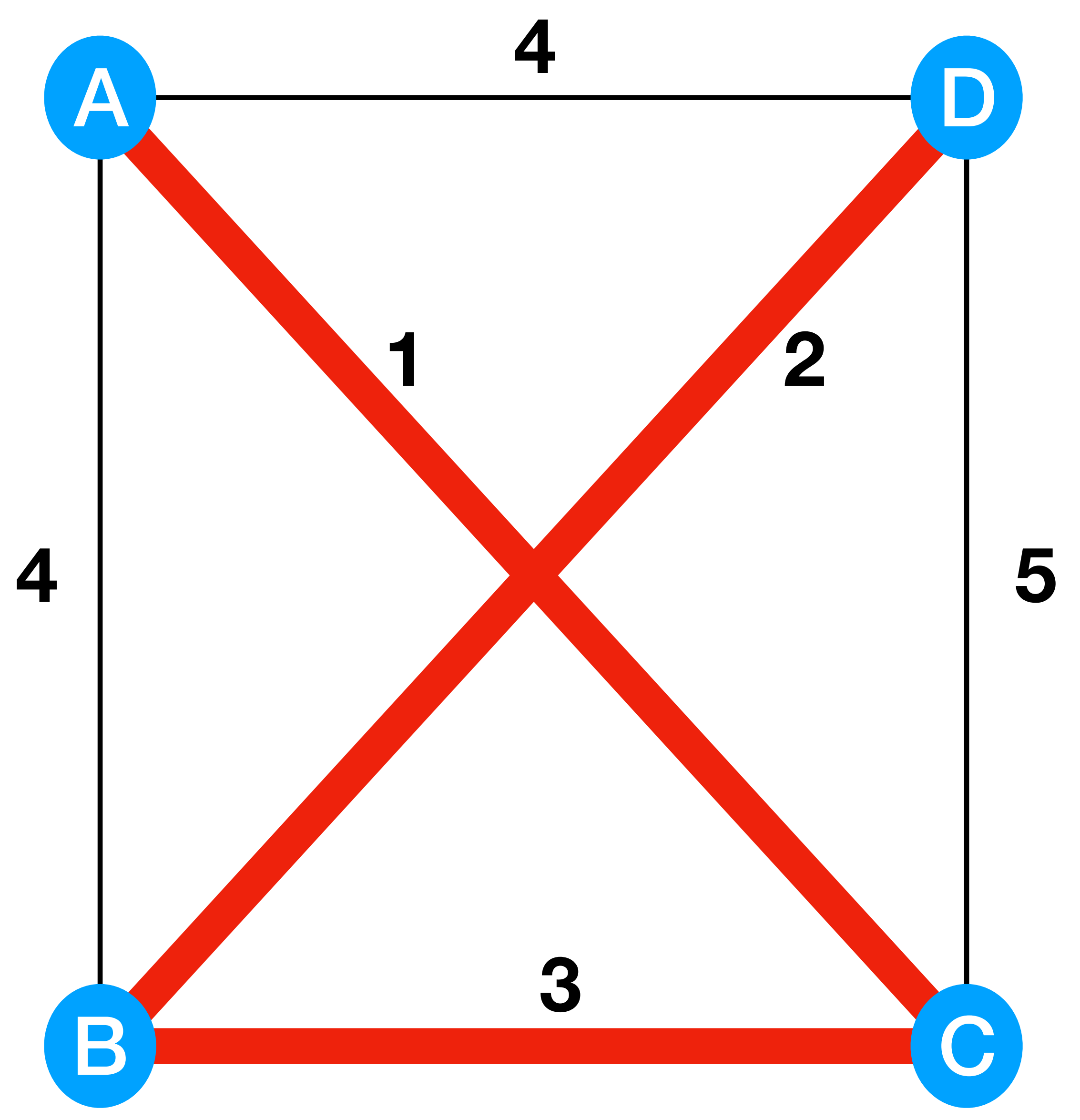
Maximum Parsimony

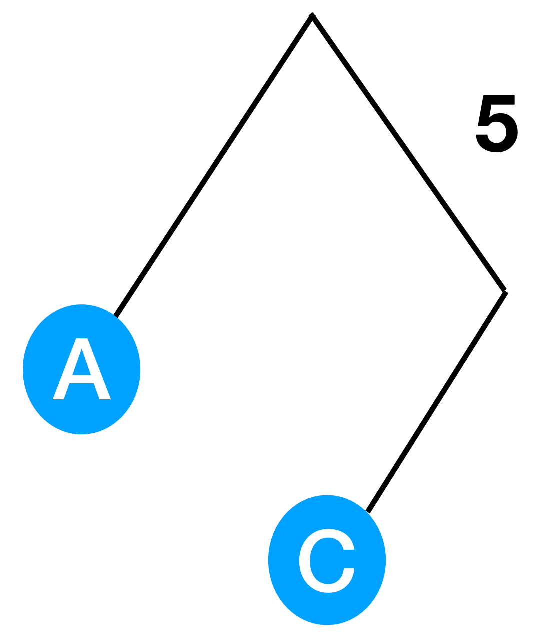
2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.
- Let *C* be the Euler cycle of the tree (it contains each edge twice).
- Define *w(H)* for some graph *H* to be the sum of the weights of all edges.
- Let *P* be the path containing all nodes of the graph constructed from *M* ordered by their first occurrence in *C*.
- $w(T') \leq w(P) \leq w(C) = 2w(T^*)$
  (T' is the minimum spanning tree of the graph)

M	**1**	**2**	**3**	**4**	**5**
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	0	1	1	0





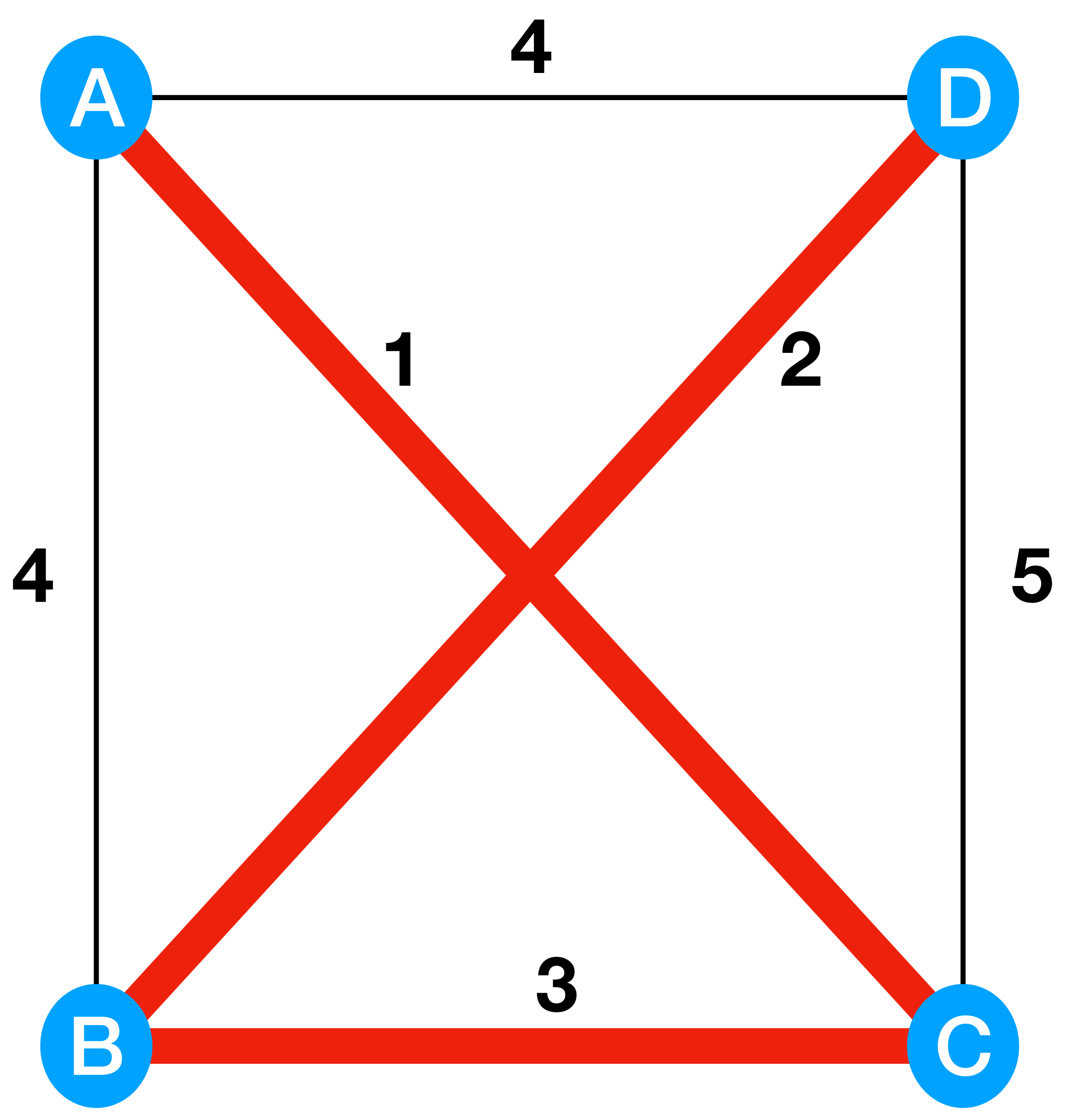
Maximum Parsimony

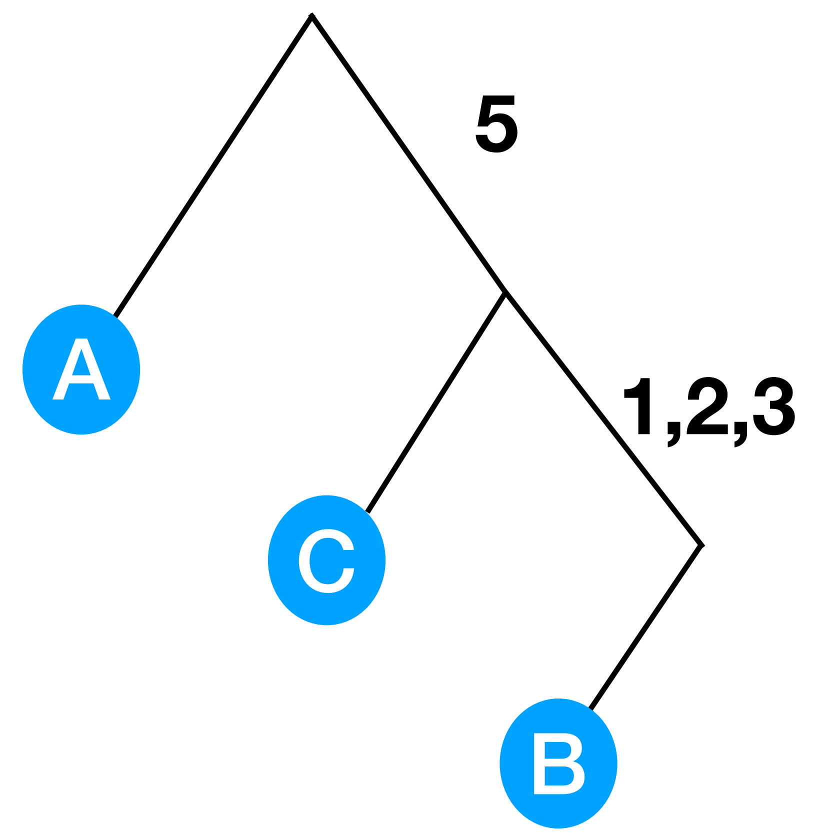
2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.
- Let *C* be the Euler cycle of the tree (it contains each edge twice).
- Define *w(H)* for some graph *H* to be the sum of the weights of all edges.
- Let *P* be the path containing all nodes of the graph constructed from *M* ordered by their first occurrence in *C*.
- $w(T') \leq w(P) \leq w(C) = 2w(T^*)$
  (T' is the minimum spanning tree of the graph)

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





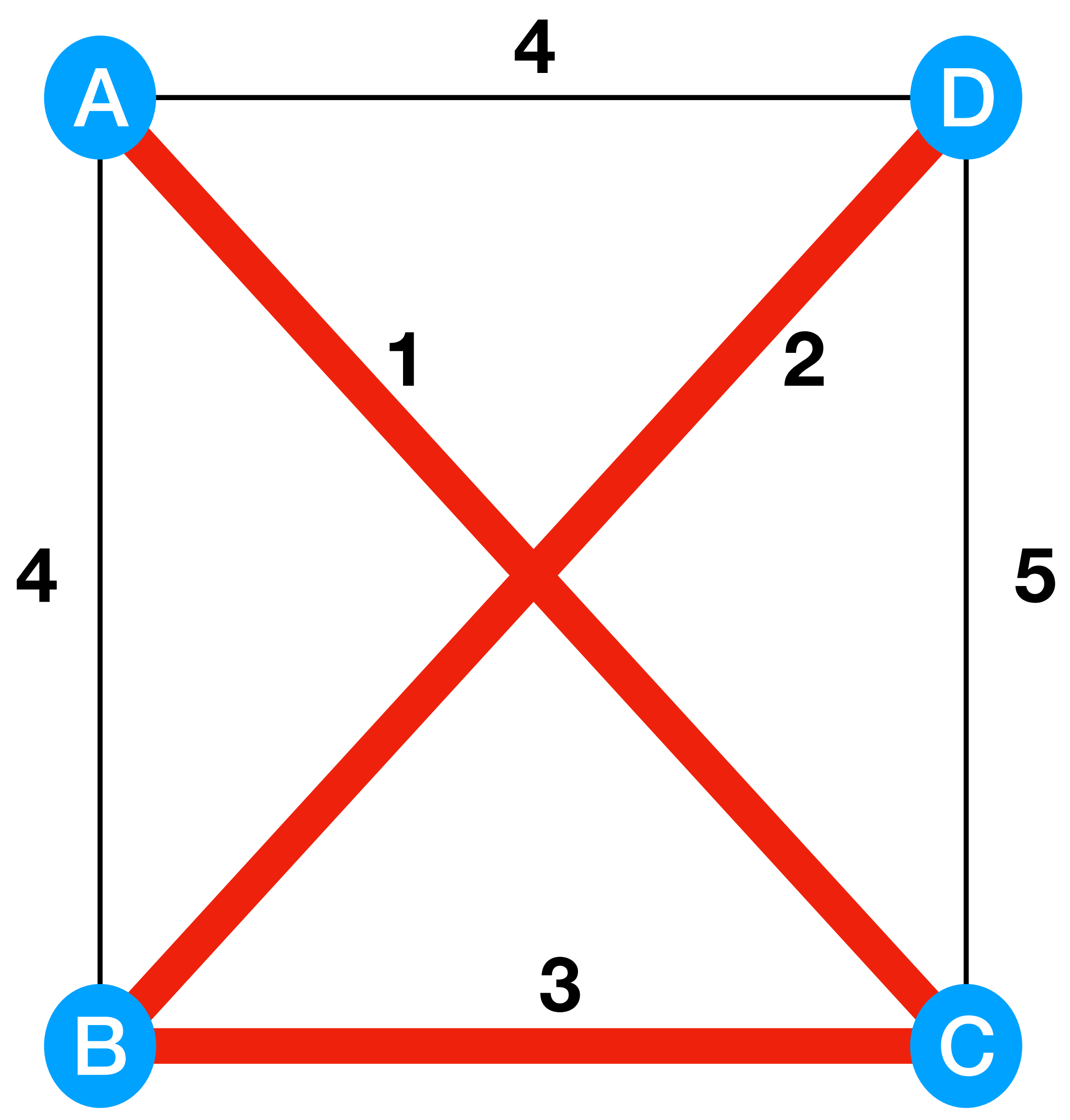
Maximum Parsimony

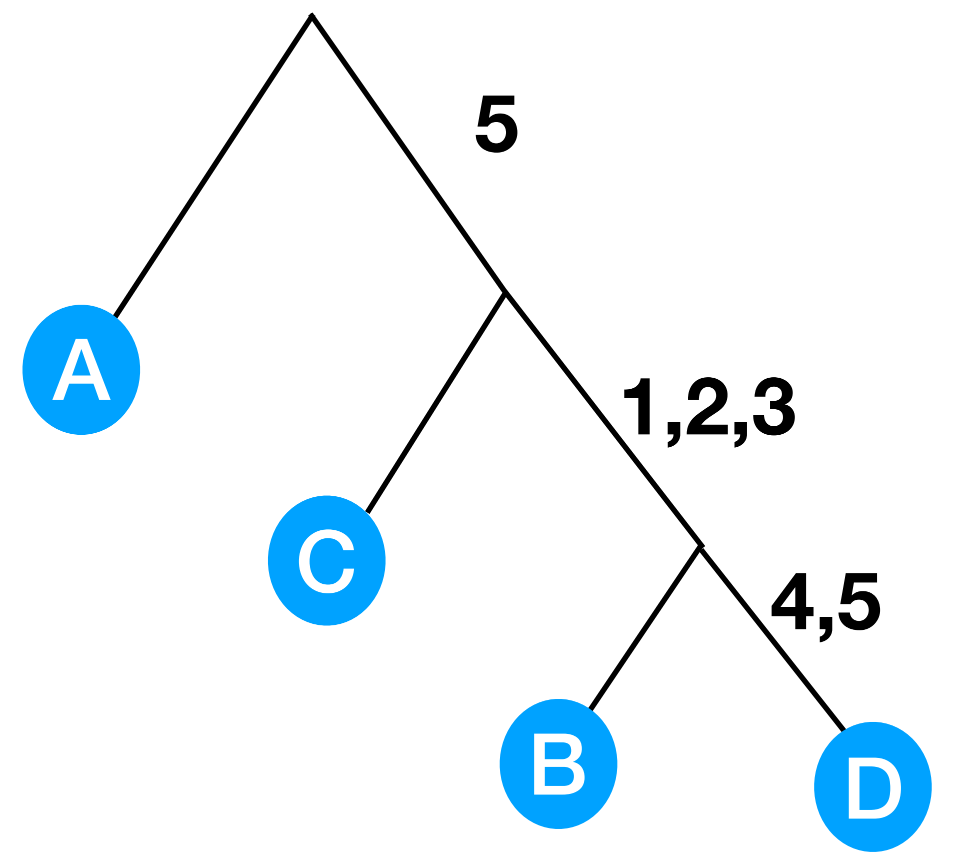
2-Approximation Algorithm

The approximation guarantee is based on an Euler cycle of the best tree.

- Let *T** be the optimal tree for *M*.
- Let *C* be the Euler cycle of the tree (it contains each edge twice).
- Define *w(H)* for some graph *H* to be the sum of the weights of all edges.
- Let *P* be the path containing all nodes of the graph constructed from *M* ordered by their first occurrence in *C*.
- $w(T') \leq w(P) \leq w(C) = 2w(T^*)$
  (T' is the minimum spanning tree of the graph)

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





Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

- let *Z* = *{{1},{2},{3},..,{n}}* (* the set of initial clusters *)

Neighbor Joining

Algorithm Given a distance matrix M with rows labeled $(1,2,3....n)$

- let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$

Neighbor Joining

Algorithm Given a distance matrix M with rows labeled $(1,2,3....n)$

- let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
- while $|Z|>1$

Neighbor Joining

```
Algorithm Given a distance matrix $M$ with rows labeled $(1,2,3....n)$
    • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
    • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
    • while $|Z|>1$
        • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
```

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

- let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
- while $|Z|>1$
  - define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
  - $(A, B) = arg \min_{(A,B) \in Z} D(A, B) - u_A - u_B$

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

- let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
- while $|Z|>1$
    - define $u_A = 1/(n-2) * \Sigma_{F\in Z} D(A,F)$ for all $A \in Z$
    - $(A, B) = arg \min_{(A,B)\in Z} D(A, B) - u_A - u_B$
    - form *C* by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

- let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
- while $|Z|>1$
  - define $u_A = 1/(n-2) * \Sigma_{F\in Z} D(A,F)$ for all $A \in Z$
  - $(A, B) = arg \min_{(A,B)\in Z} D(A,B) - u_A - u_B$
  - form *C* by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
  - $Z = Z \cup \{C\} - \{A,B\}$

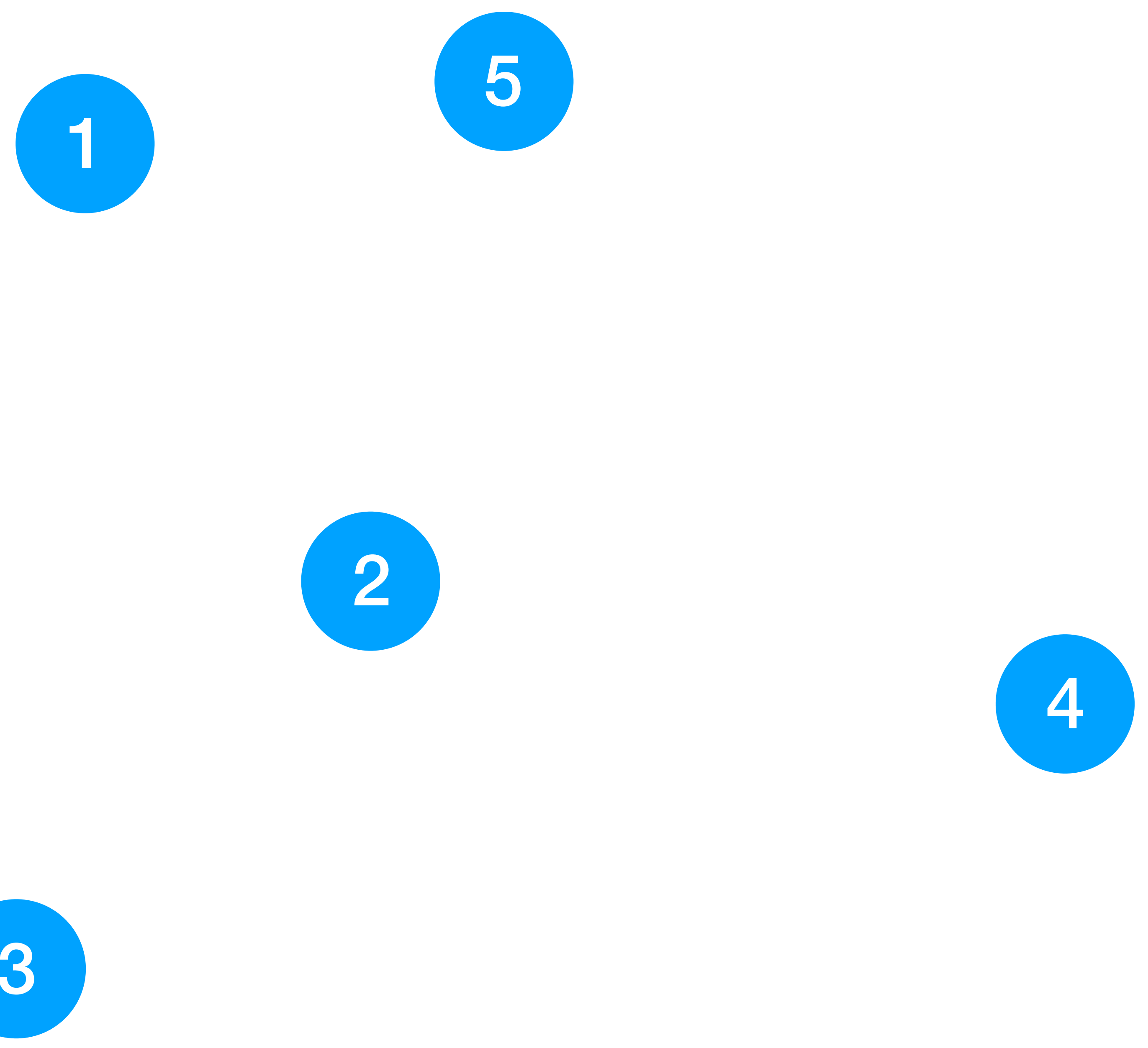
Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
• let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
• for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
• while $|Z|>1$
    • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
    • $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
    • form $C$ by creating a new cluster root and connecting it to the two cluster roots
      with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
    • $Z = Z \cup \{C\} - \{A,B\}$
    • define $D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )$
```

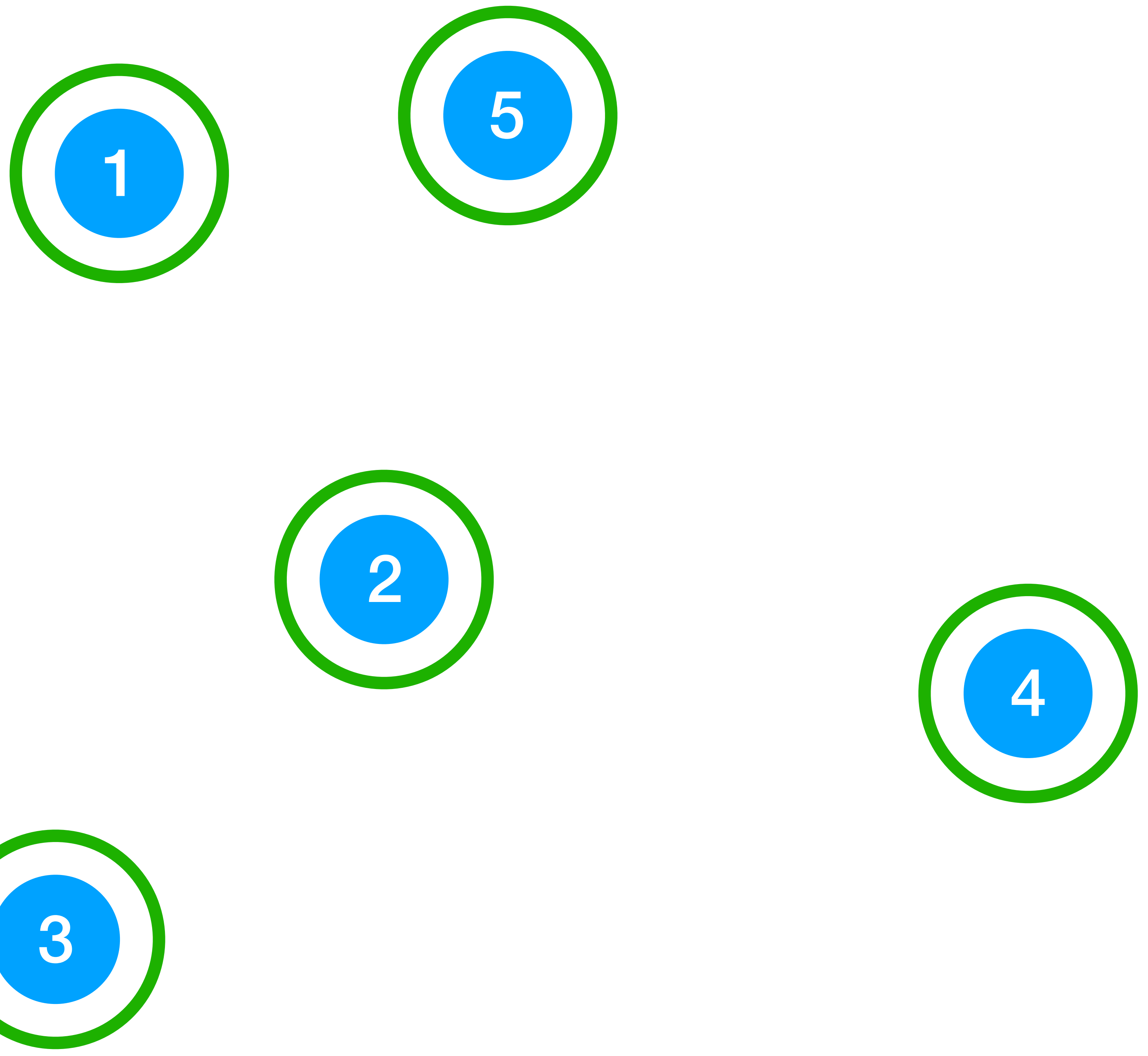
Neighbor Joining

M	1	2	3	4	5
1		8	8	5	3
2	8		3	8	8
3	8	3		8	8
4	5	8	8		5
5	3	8	8	5	



Neighbor Joining

D	{1}	{2}	{3}	{4}	{5}
{1}		8	8	5	3
{2}	8		3	8	8
{3}	8	3		8	8
{4}	5	8	8		5
{5}	3	8	8	5	

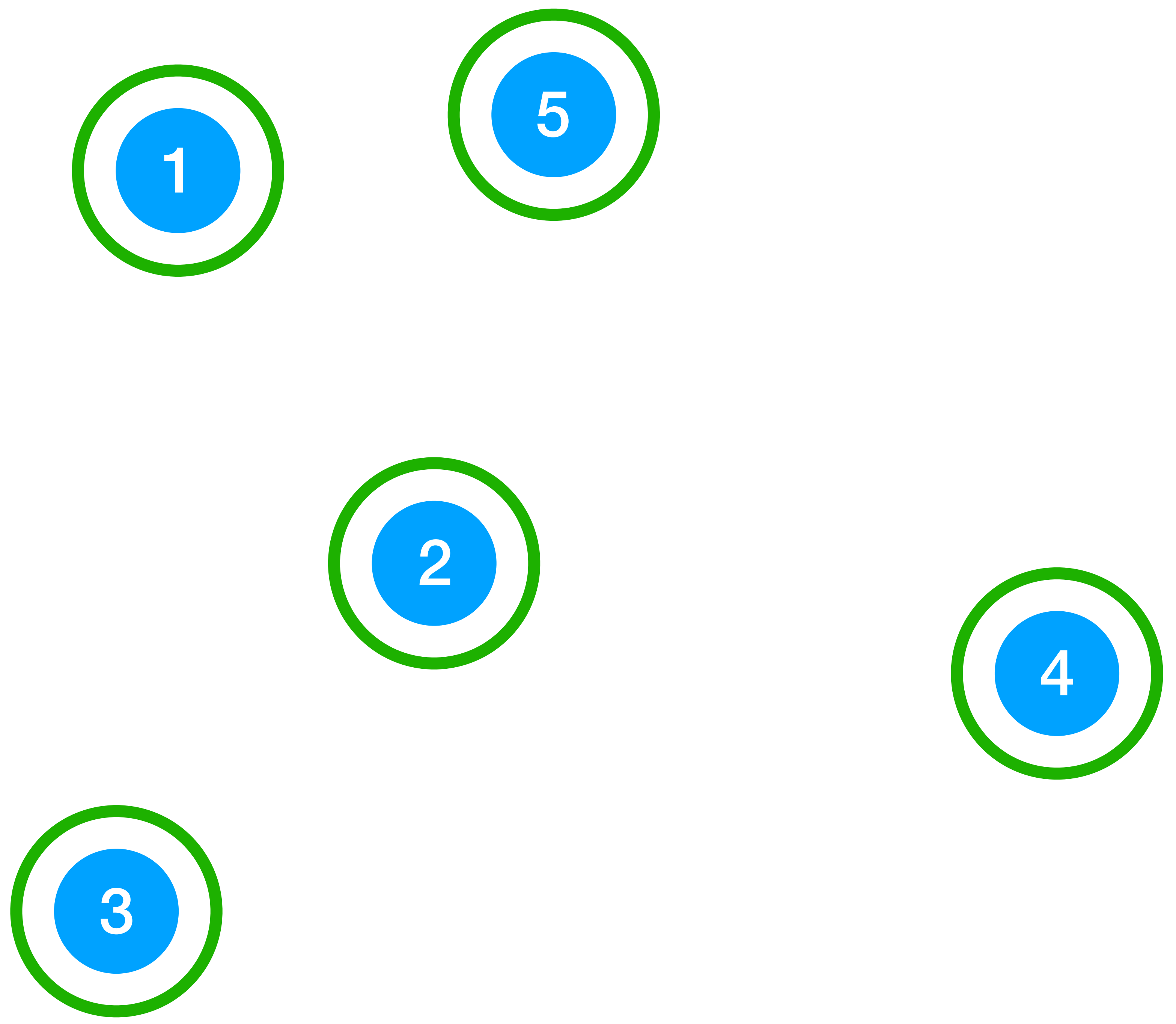


- let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$

Neighbor Joining

D	{1}	{2}	{3}	{4}	{5}
{1}		8	8	5	3
{2}	8		3	8	8
{3}	8	3		8	8
{4}	5	8	8		5
{5}	3	8	8	5	

	u_A
{1}	8
{2}	9
{3}	9
{4}	8.66
{5}	8

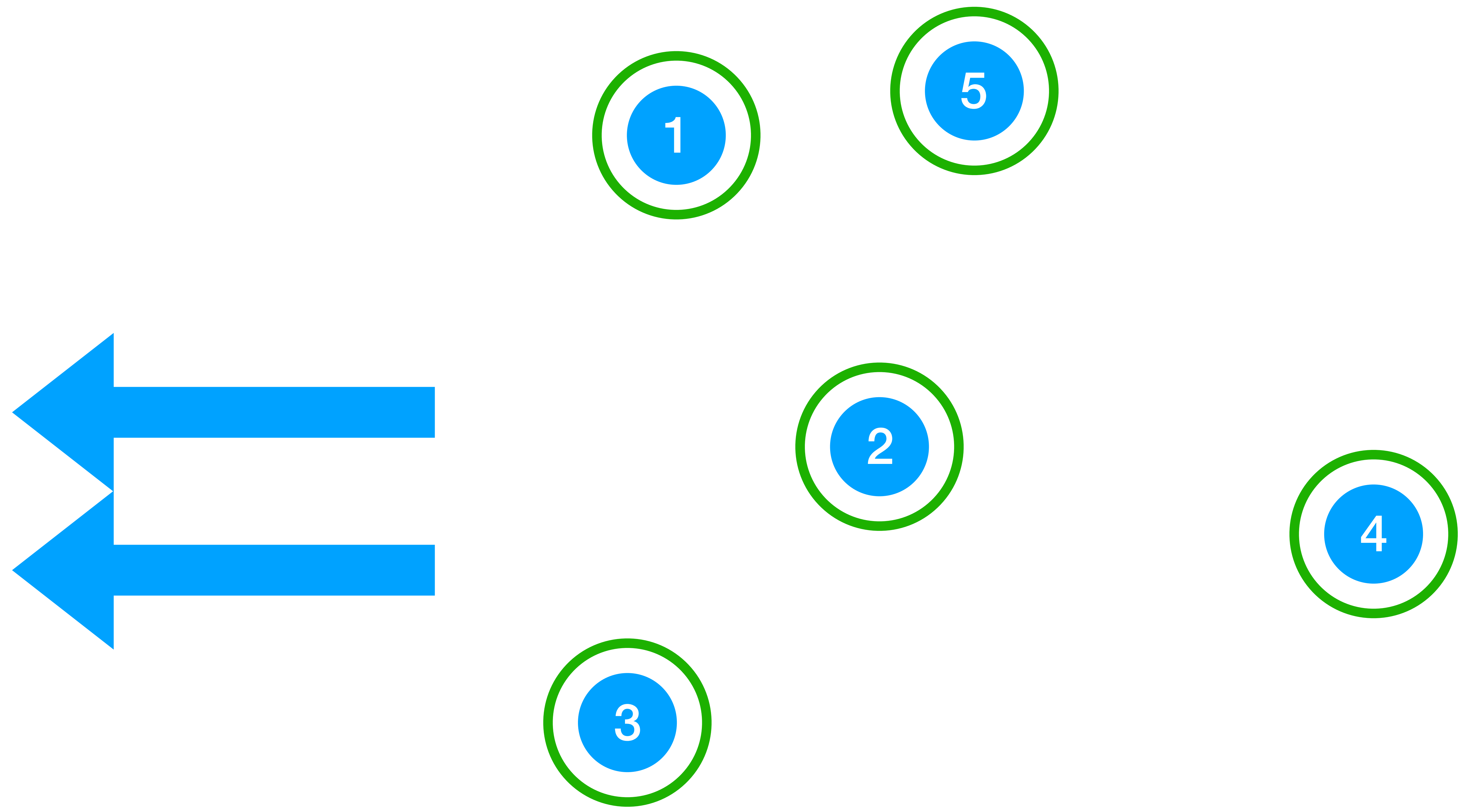


- while $|Z|>1$
  - define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$

Neighbor Joining

D	{1}	{2}	{3}	{4}	{5}
{1}		8	8	5	3
{2}	8		3	8	8
{3}	8	3		8	8
{4}	5	8	8		5
{5}	3	8	8	5	

	u_A
{1}	8
{2}	9
{3}	9
{4}	8.66
{5}	8

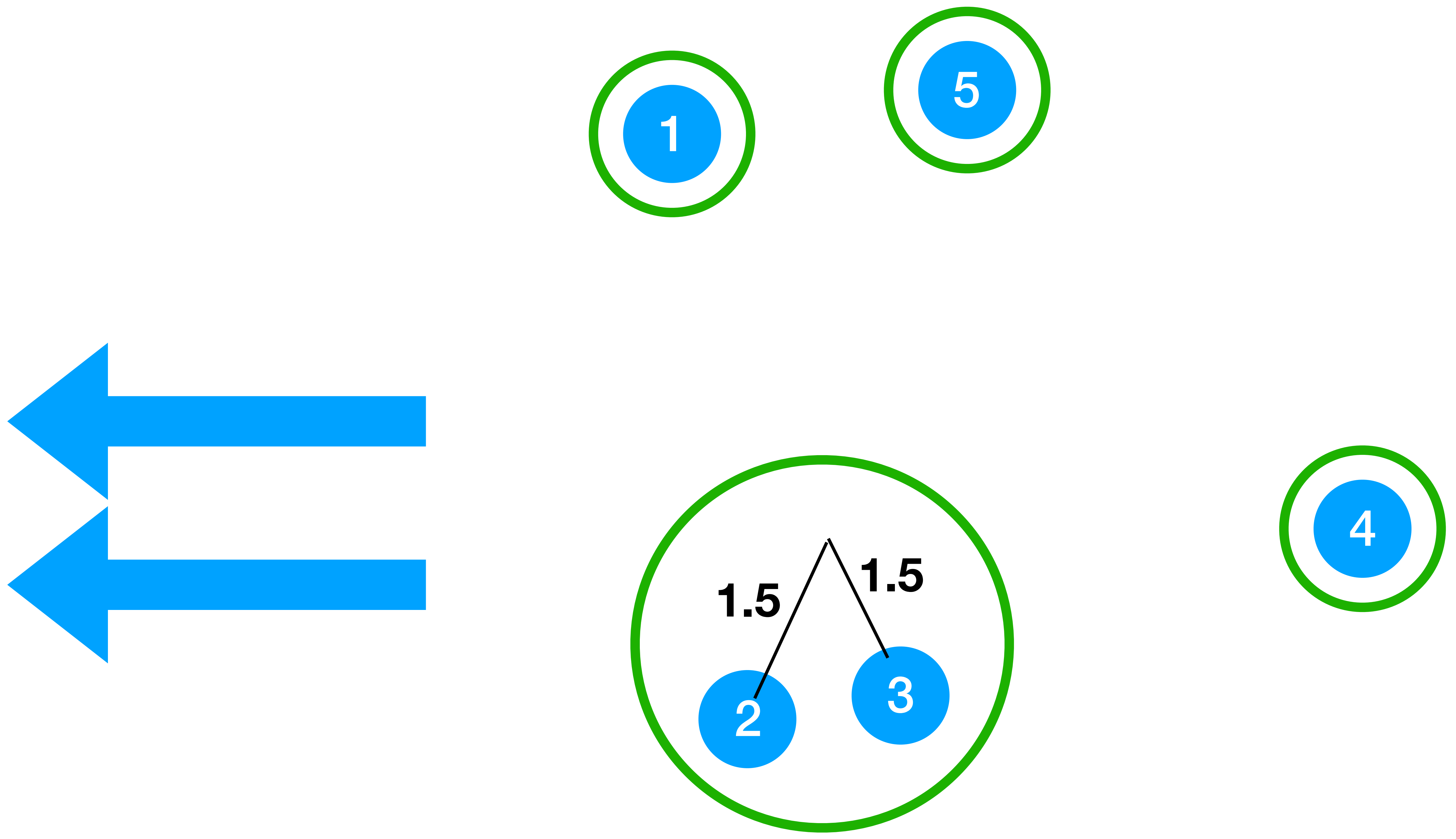


- while $|Z|>1$
  - $(A,B) = arg \min_{(A,B)\in Z} D(A,B) - u_A - u_B$

Neighbor Joining

D	{1}	{2}	{3}	{4}	{5}
{1}		8	8	5	3
{2}	8		3	8	8
{3}	8	3		8	8
{4}	5	8	8		5
{5}	3	8	8	5	

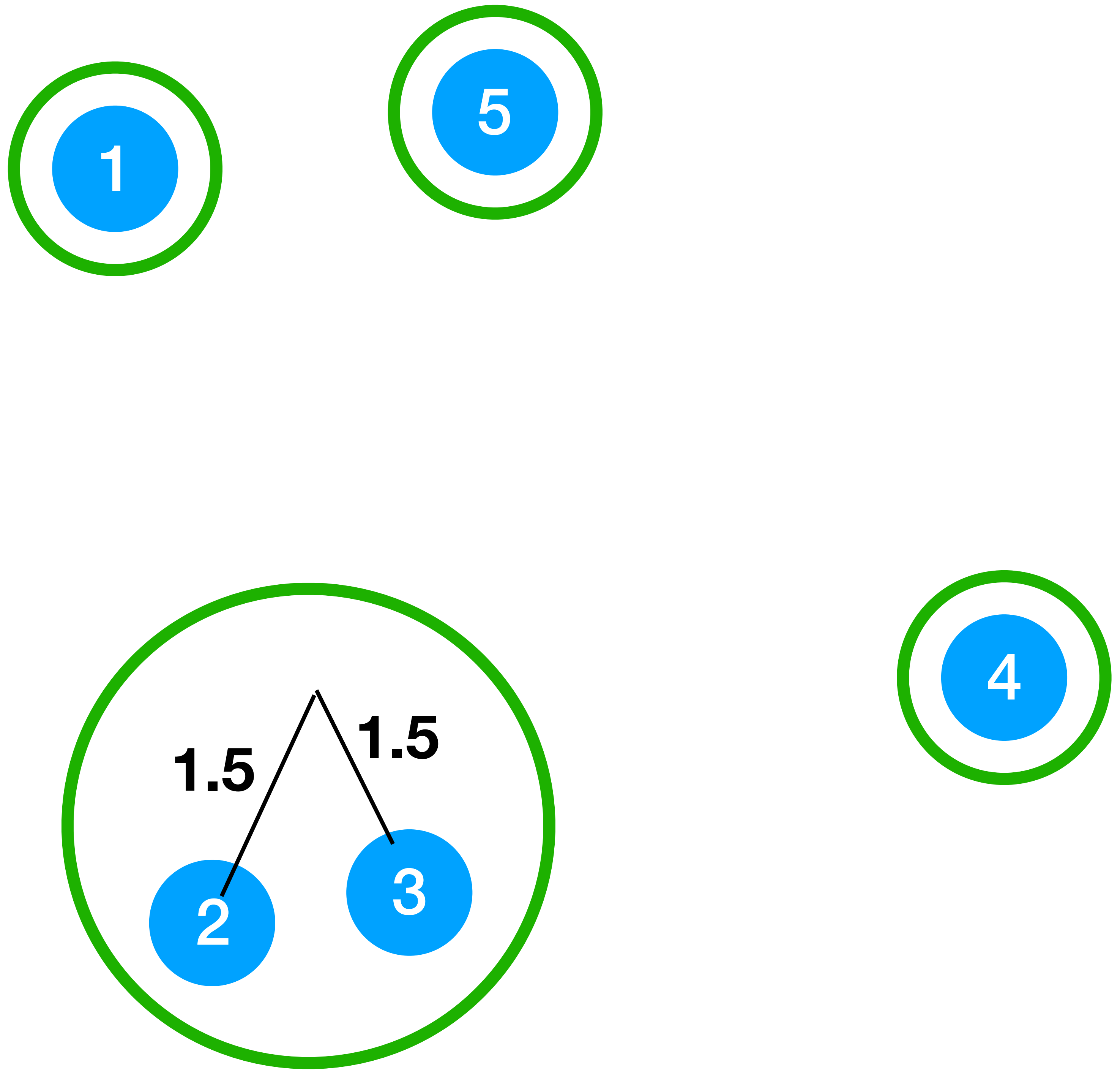
	u_A
{1}	8
{2}	9
{3}	9
{4}	8.66
{5}	8



- while $|Z|>1$
  - form C by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B)+(u_A-u_B)\right)$ and $\frac{1}{2}\left(D(A,B)+(u_B-u_A)\right)$ respectively.

Neighbor Joining

D	{1}	{2,3}	{4}	{5}
{1}		6.5	5	3
{2,3}	6.5		6.5	8
{4}	5	6.5		5
{5}	3	6.5	5	

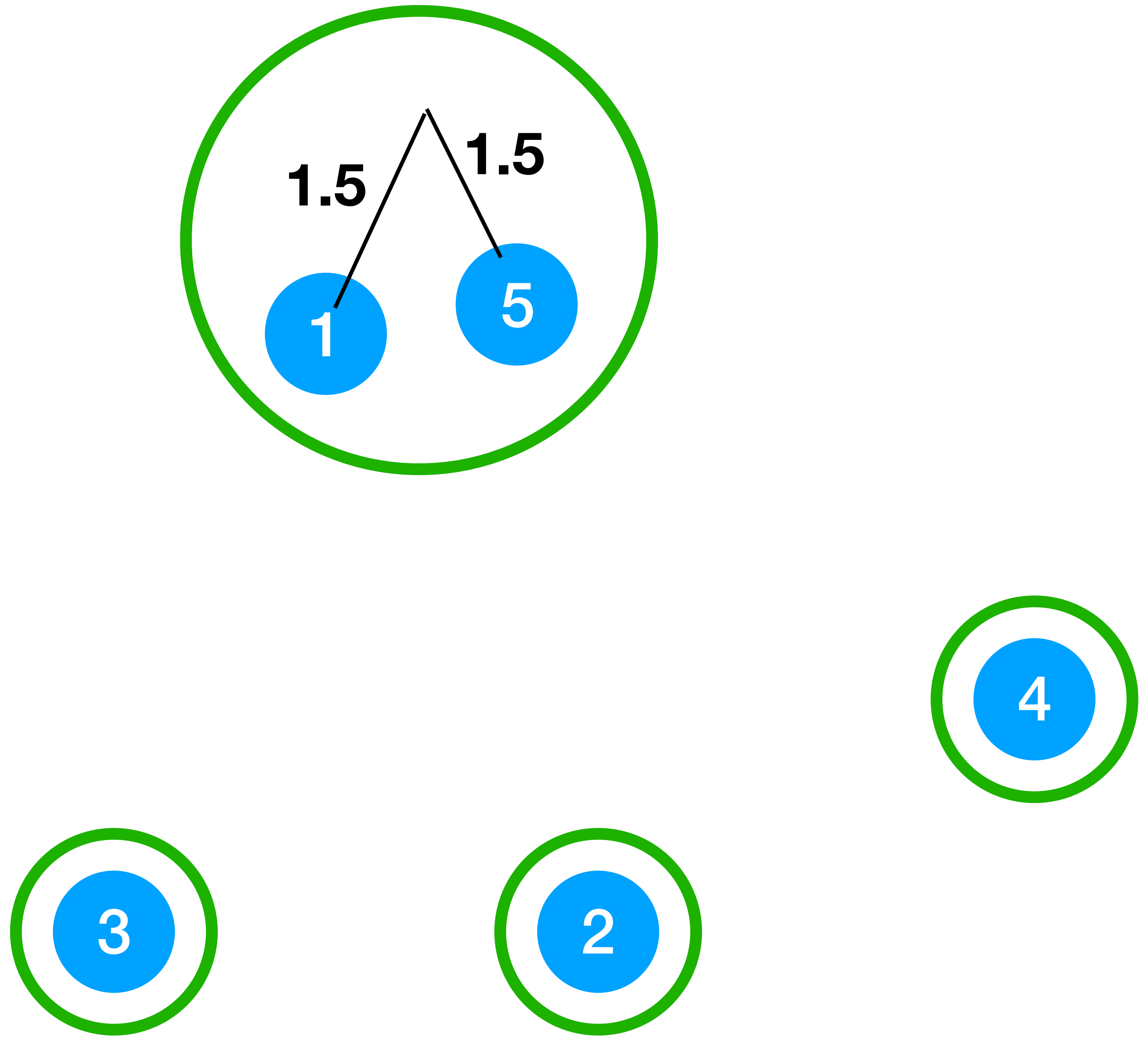


- while $|Z|>1$
  - $Z = Z \cup \{C\} - \{A,B\}$
  - define $D(F,C) = D(C,F) = 1/2 \, (D(A,F) + D(B,F) - D(A,B))$

Neighbor Joining

D	{1}	{2,3}	{4}	{5}
{1}		6.5	5	3
{2,3}	6.5		6.5	8
{4}	5	6.5		5
{5}	3	6.5	5	

	u_A
{1}	4.833
{2,3}	7
{4}	5.5
{5}	4.833

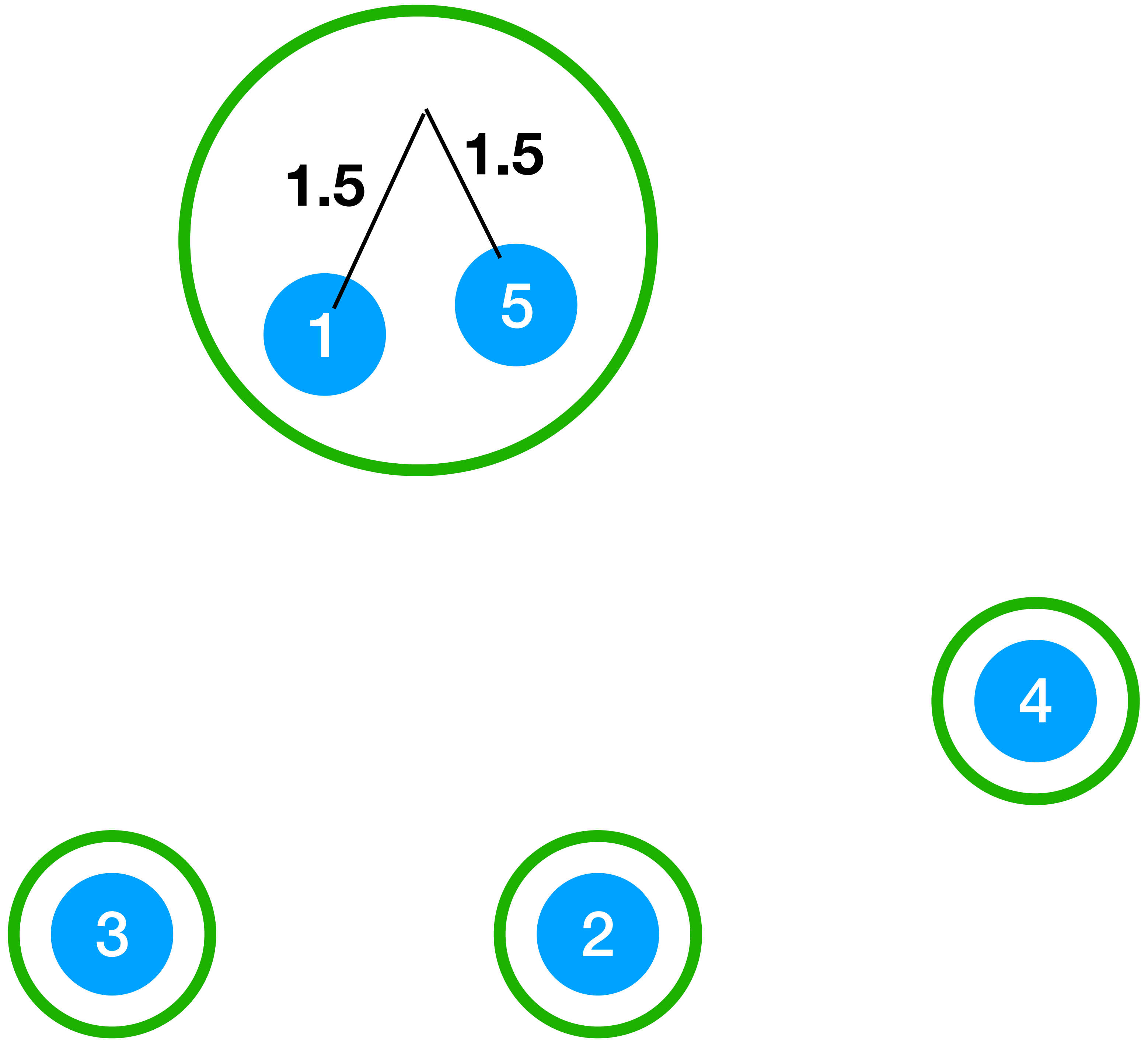


- while $|Z|>1$
  - define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$

Neighbor Joining

D	{1}	{2,3}	{4}	{5}
{1}		6.5	5	3
{2,3}	6.5		6.5	8
{4}	5	6.5		5
{5}	3	6.5	5	

	u_A
{1}	4.833
{2,3}	7
{4}	5.5
{5}	4.833

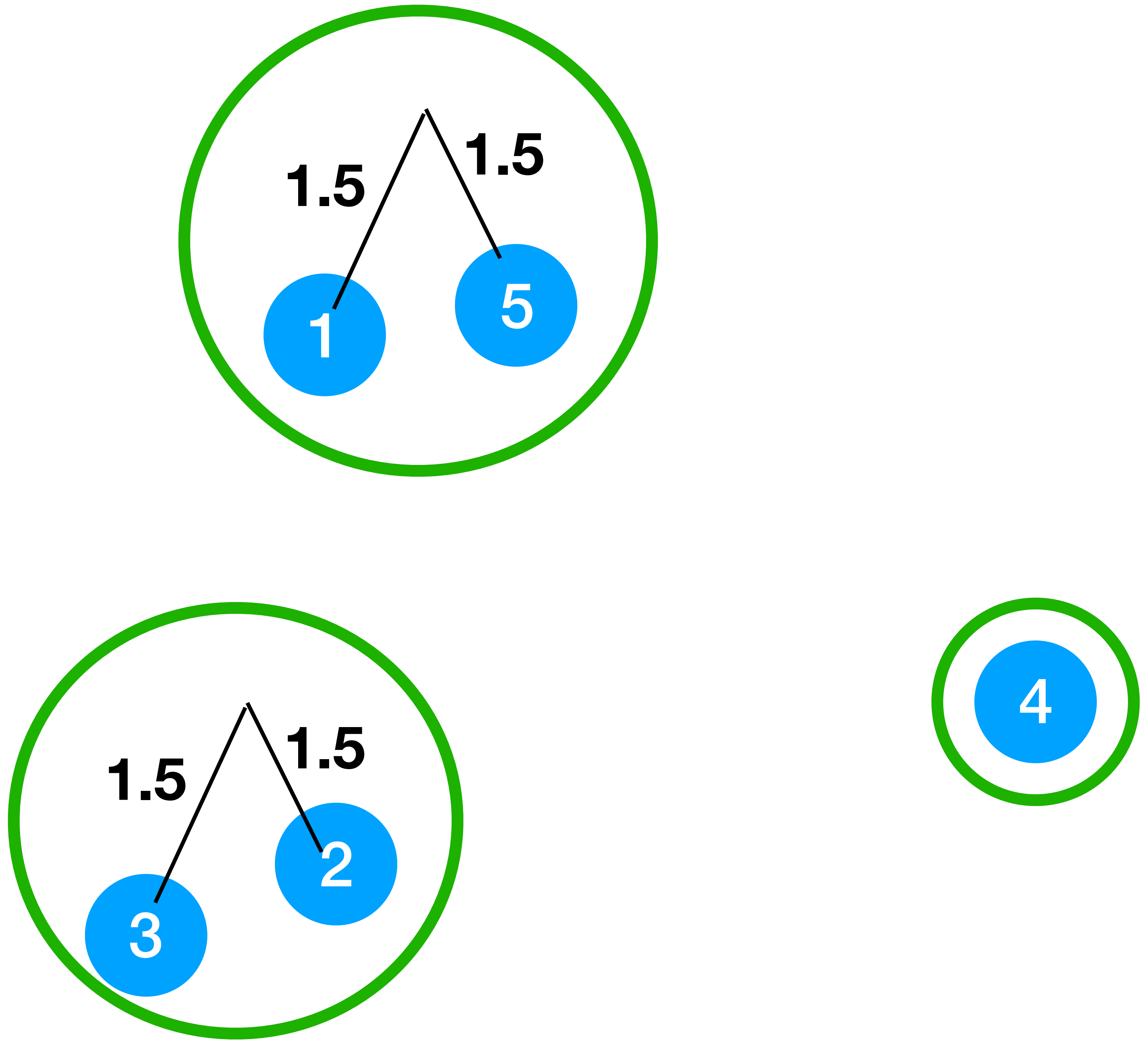


- while $|Z|>1$
  - $(A,B) = arg \min_{(A,B)\in Z} D(A,B) - u_A - u_B$

Neighbor Joining

D	{1}	{2,3}	{4}	{5}
{1}		6.5	5	3
{2,3}	6.5		6.5	8
{4}	5	6.5		5
{5}	3	6.5	5	

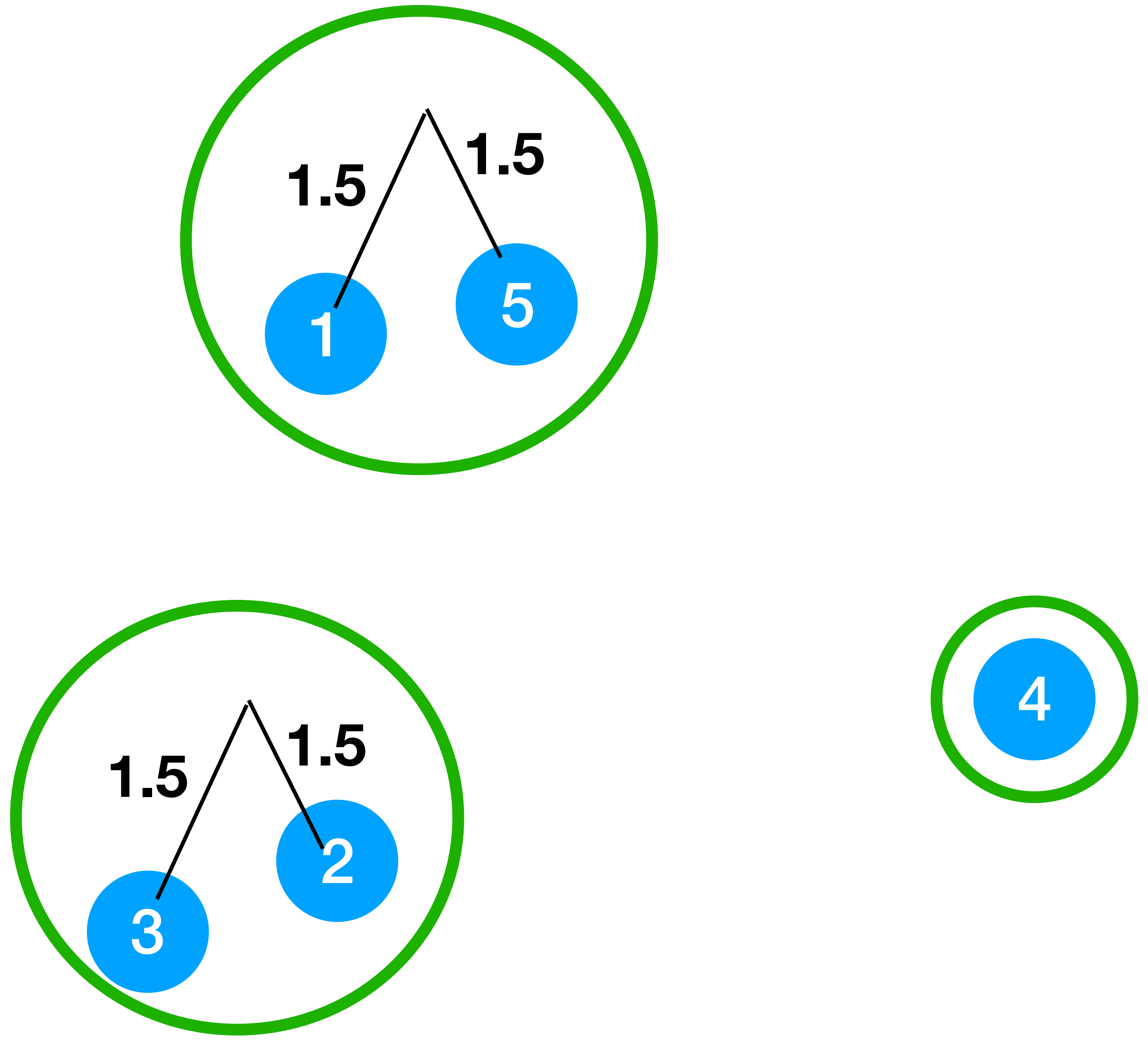
	u_A
{1}	4.833
{2,3}	7
{4}	5.5
{5}	4.833



- while $|Z|>1$
  - form C by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B)+(u_A-u_B)\right)$ and $\frac{1}{2}\left(D(A,B)+(u_B-u_A)\right)$ respectively.

Neighbor Joining

D	{1,5}	{2,3}	{4}
{1,5}		5.75	3.5
{2,3}	5.75		6.5
{4}	3.5	6.5	

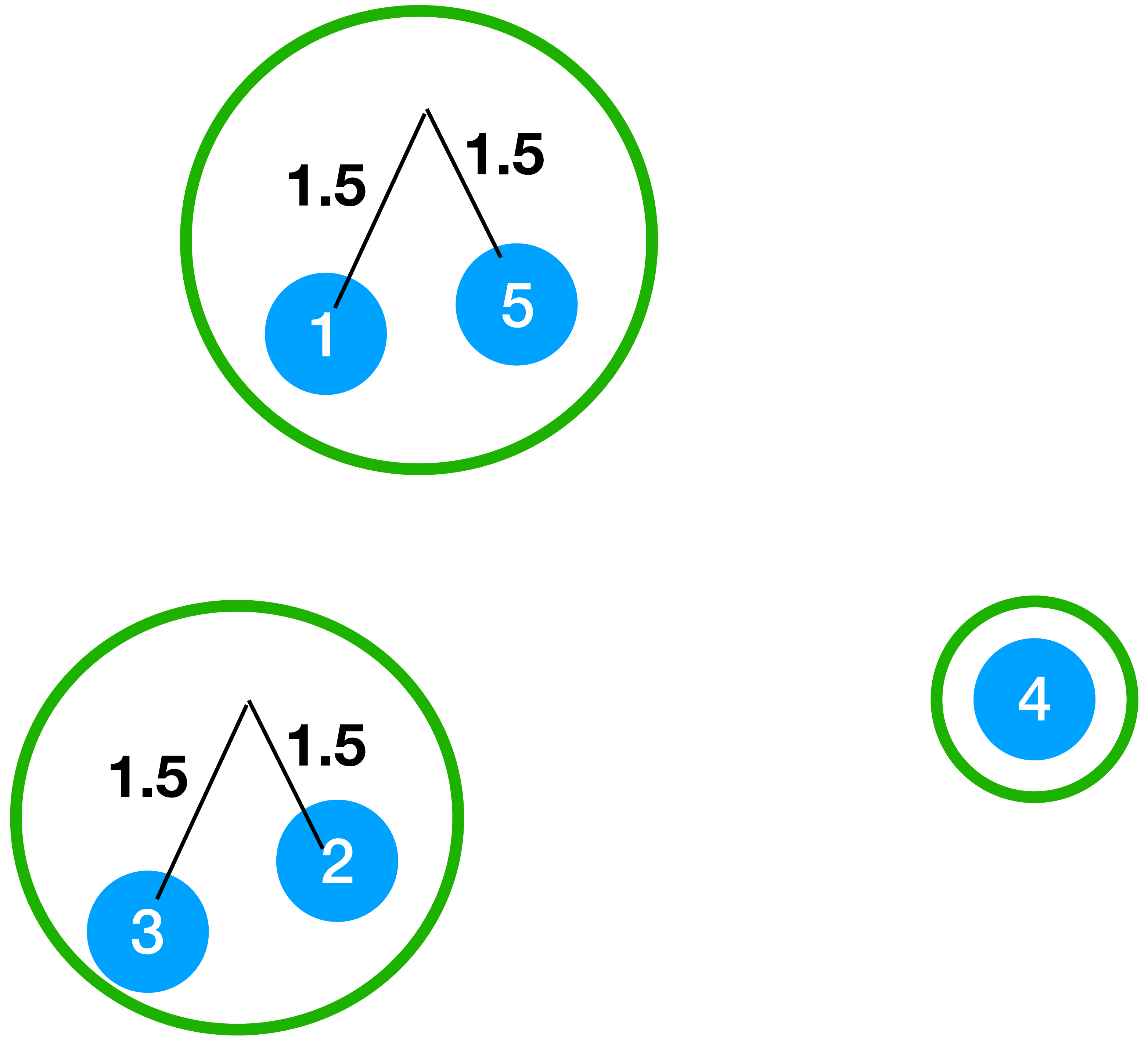


- while $|Z|>1$
  - $Z = Z \cup \{C\} - \{A,B\}$
  - define $D(F,C) = D(C,F) = 1/2\ (\ D(A,F) + D(B,F) - D(A,B)\ )$

Neighbor Joining

D	{1,5}	{2,3}	{4}
{1,5}		5.75	3.5
{2,3}	5.75		6.5
{4}	3.5	6.5	

	u
{1,5}	3.083
{2,3}	4.083
{4}	3.33

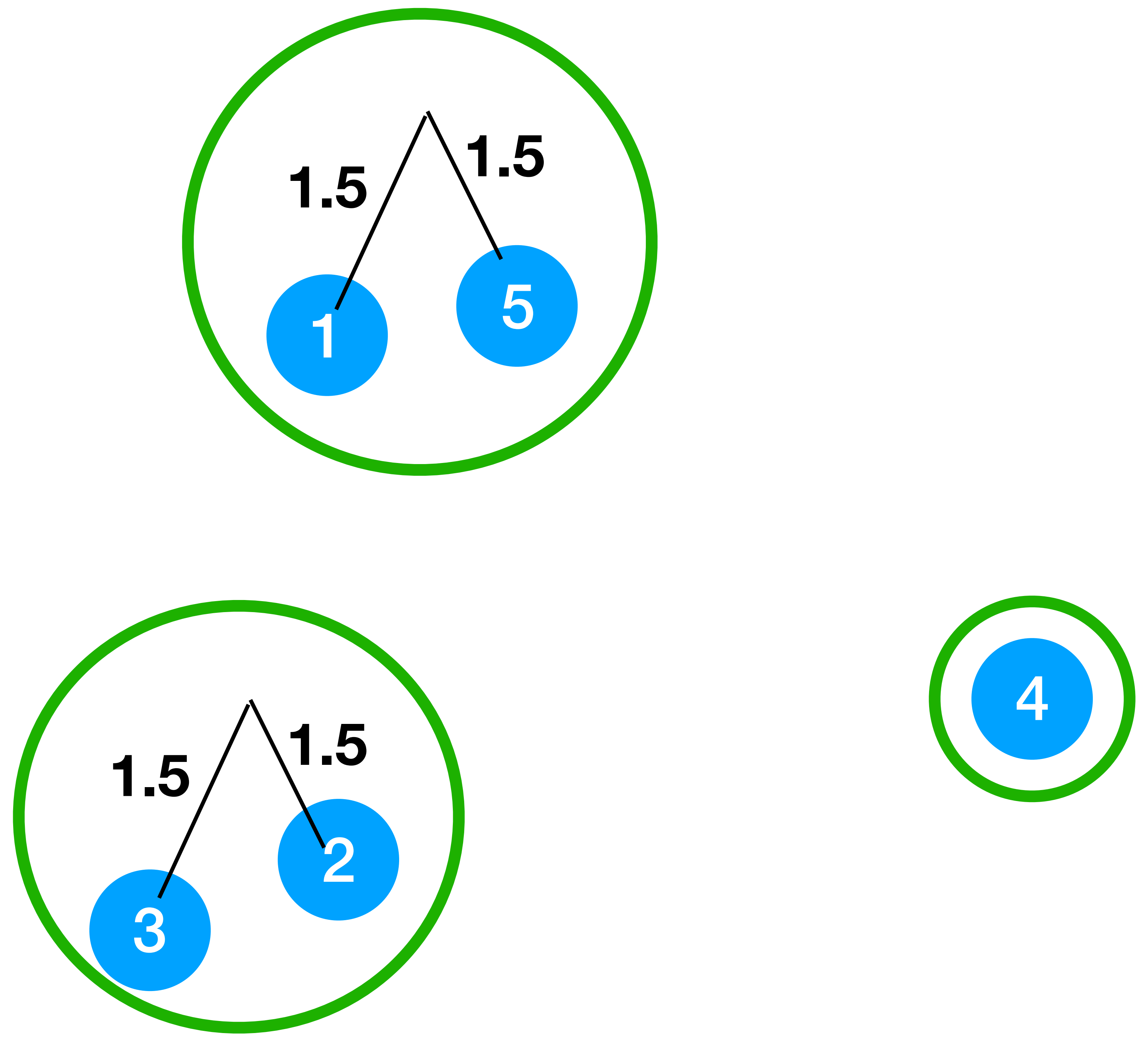


- while $|Z|>1$
  - define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$

Neighbor Joining

D	{1,5}	{2,3}	{4}
{1,5}		5.75	3.5
{2,3}	5.75		6.5
{4}	3.5	6.5	

	u
{1,5}	2.833
{2,3}	3.833
{4}	3.33

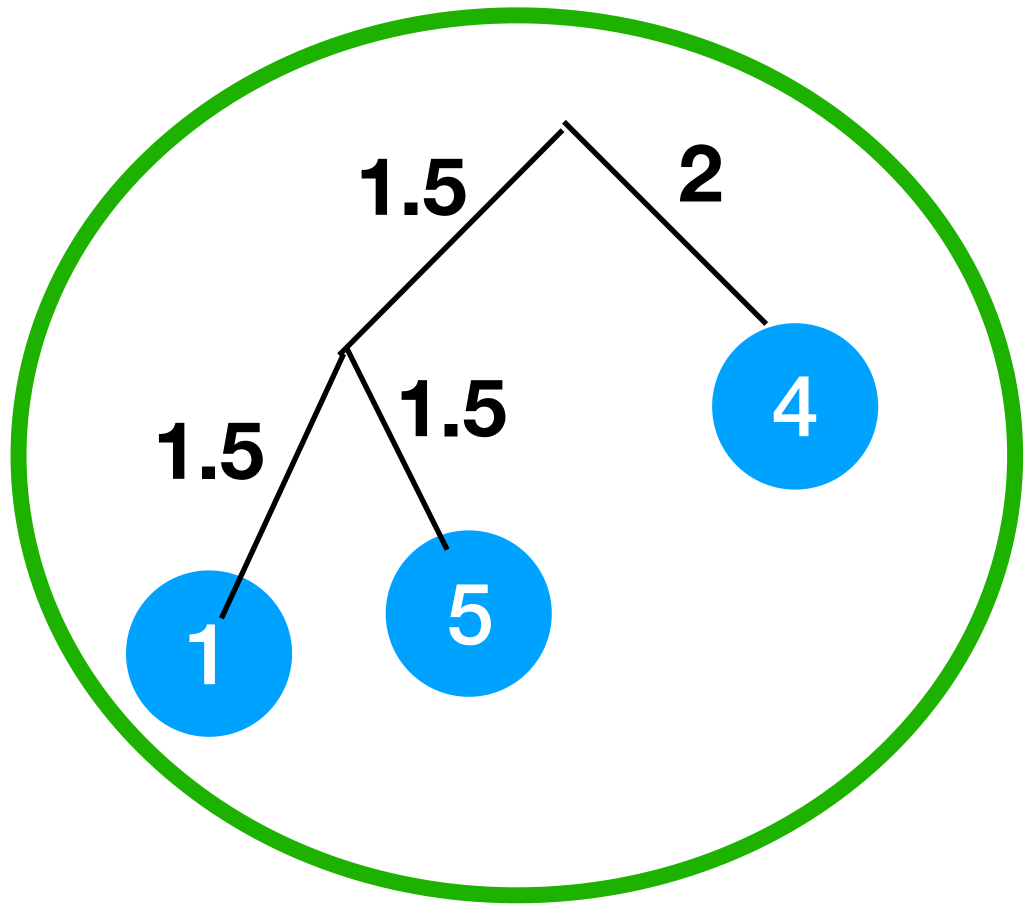


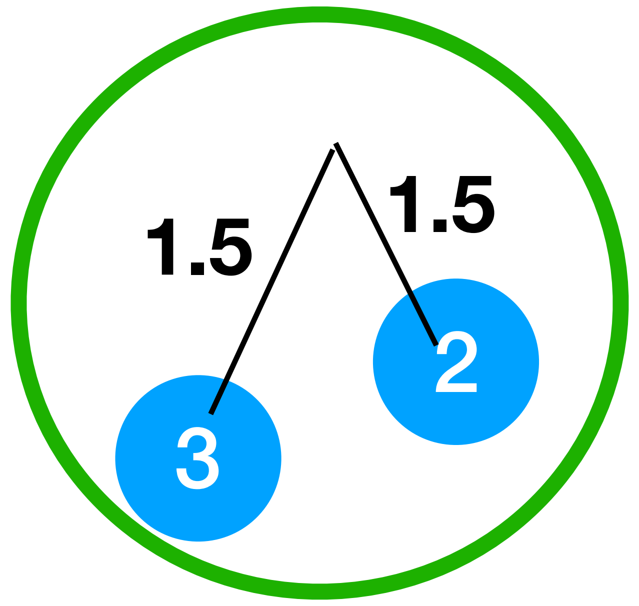
- while $|Z|>1$
  - $(A, B) = arg \min_{(A,B)\in Z} D(A, B) - u_A - u_B$

Neighbor Joining

D	{1,5}	{2,3}	{4}
{1,5}		5.75	3.5
{2,3}	5.75		6.5
{4}	3.5	6.5	

	u
{1,5}	2.833
{2,3}	3.833
{4}	3.33

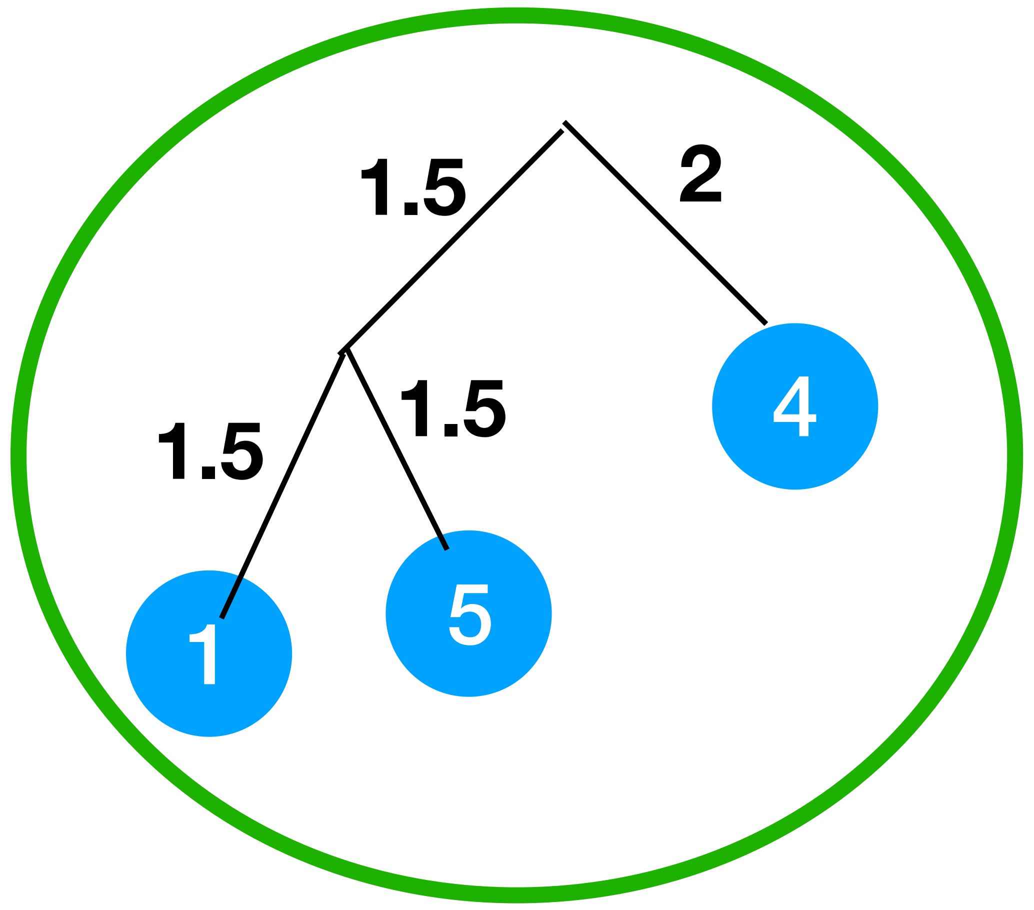


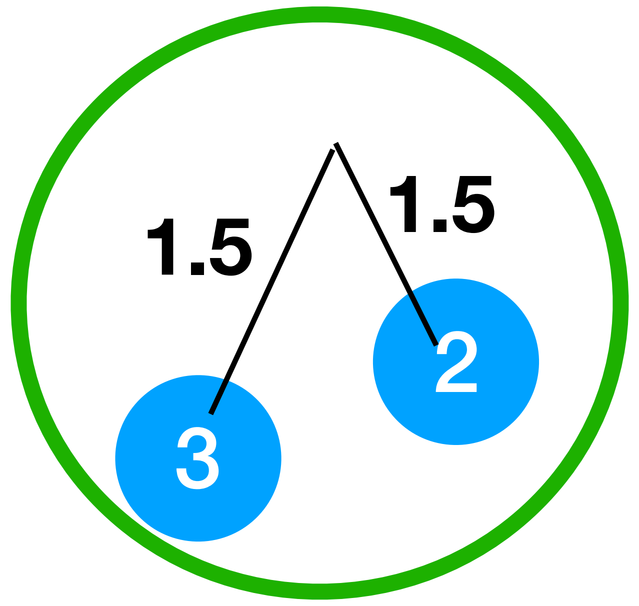


- while $|Z|>1$
  - form C by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B)+(u_A-u_B)\right)$ and $\frac{1}{2}\left(D(A,B)+(u_B-u_A)\right)$ respectively.

Neighbor Joining

D	{1,5,4}	{2,3}
{1,5,4}		4
{2,3}	4	



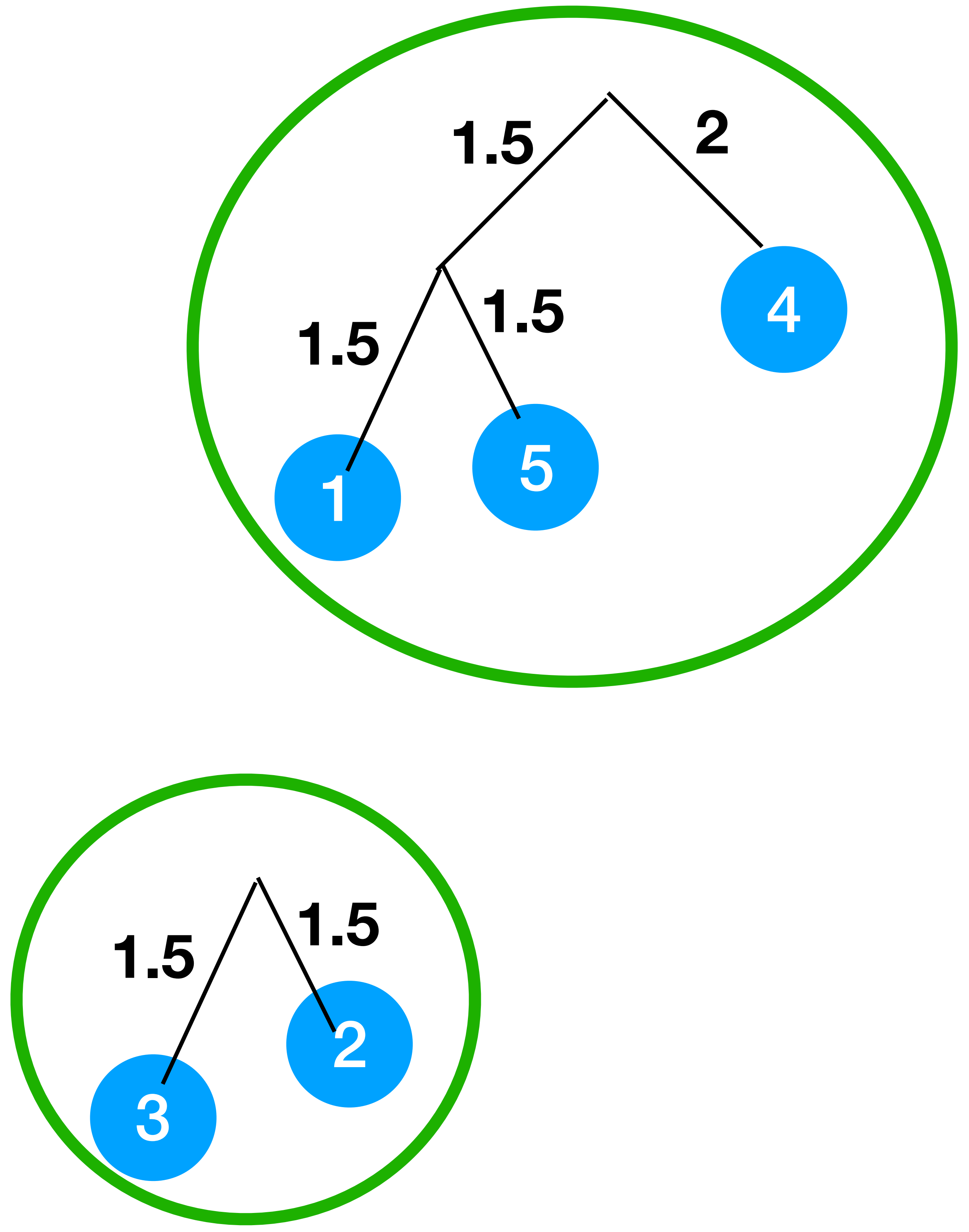


- while $|Z|>1$
  - $Z = Z \cup \{C\} - \{A,B\}$
  - define $D(F,C) = D(C,F) = 1/2\ (\ D(A,F) + D(B,F) - D(A,B)\ )$

Neighbor Joining

D	{1,5,4}	{2,3}
{1,5,4}		4
{2,3}	4	

	u
{1,5,4}	1.33
{2,3}	1.33

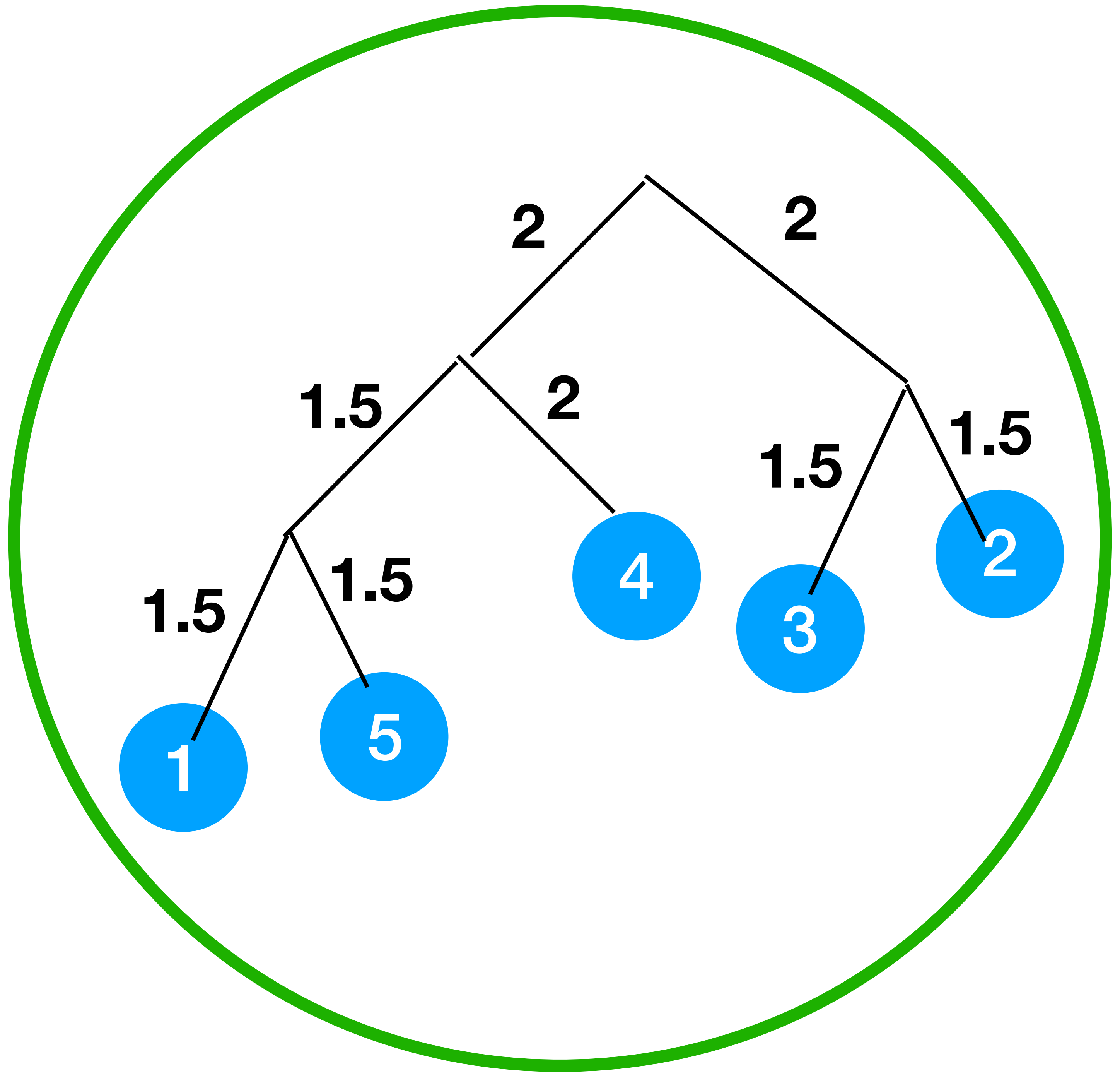


- while $|Z|>1$
  - define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
  - $(A, B) = \mathit{arg} \min_{(A,B) \in Z} D(A, B) - u_A - u_B$

Neighbor Joining

D	{1,5,4}	{2,3}
{1,5,4}		4
{2,3}	4	

	u
{1,5,4}	1.33
{2,3}	1.33



- while $|Z|>1$
  - form C by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B)+(u_A-u_B)\right)$ and $\frac{1}{2}\left(D(A,B)+(u_B-u_A)\right)$ respectively.

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

- let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
- while $|Z|>1$
  - define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
  - $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
  - form *C* by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
  - $Z = Z \cup \{C\} - \{A,B\}$
  - define $D(F,C) = D(C,F) = 1/2 (D(A,F) + D(B,F) - D(A,B))$

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
O(n) • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
     • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
     • while $|Z|>1$
         • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
         • $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
         • form $C$ by creating a new cluster root and connecting it to the two cluster roots
           with edge weights $\frac{1}{2}\left(D(A,B)+(u_A-u_B)\right)$ and $\frac{1}{2}\left(D(A,B)+(u_B-u_A)\right)$ respectively.
         • $Z = Z \cup \{C\} - \{A,B\}$
         • define $D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )$
```

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
O(n)   • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
O(n²)  • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
       • while $|Z|>1$
           • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
           • $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
           • form C by creating a new cluster root and connecting it to the two cluster roots
             with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
           • $Z = Z \cup \{C\} - \{A,B\}$
           • define $D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )$
```

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

- O(n) • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
- O(n²) • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
- while $|Z|>1$
    - O(n) • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
    - $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
    - form C by creating a new cluster root and connecting it to the two cluster roots with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
    - $Z = Z \cup \{C\} - \{A,B\}$
    - define $D(F,C) = D(C,F) = 1/2 (D(A,F) + D(B,F) - D(A,B))$

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
O(n)    • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
O(n²)   • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
        • while $|Z|>1$
O(n)        • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
O(n²)       • $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
            • form C by creating a new cluster root and connecting it to the two cluster roots
              with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
            • $Z = Z \cup \{C\} - \{A,B\}$
            • define $D(F,C) = D(C,F) = 1/2\ (\ D(A,F) + D(B,F) - D(A,B)\ )$
```

Neighbor Joining

Algorithm Given a distance matrix M with rows labeled $(1,2,3....n)$

```
O(n)     • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
O(n²)    • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
         • while $|Z|>1$
O(n)         • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
O(n²)        • $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
O(1)         • form $C$ by creating a new cluster root and connecting it to the two cluster roots
               with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
             • $Z = Z \cup \{C\} - \{A,B\}$
             • define $D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )$
```

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
O(n)    • let Z = {{1},{2},{3},..,{n}} (* the set of initial clusters *)
O(n²)   • for all {i},{j} ∈ Z set D({i},{j})=$M_{i,j}$
        • while |Z|>1
O(n)        • define $u_A$ = 1/(n-2) * $\Sigma_{F \in Z}$ D(A,F) for all A ∈ Z
O(n²)       • $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
O(1)        • form C by creating a new cluster root and connecting it to the two cluster roots
              with edge weights $\frac{1}{2}\left(D(A,B) + (u_A - u_B)\right)$ and $\frac{1}{2}\left(D(A,B) + (u_B - u_A)\right)$ respectively.
O(1)        • Z = Z ∪ {C} - {A,B}
            • define D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )
```

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
O(n)    • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
O(n²)   • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
        • while $|Z|>1$
O(n)        • define $u_A = 1/(n-2) * \Sigma_{F\in Z} D(A,F)$ for all $A \in Z$
O(n²)       • $(A,B) = arg \min_{(A,B)\in Z} D(A,B) - u_A - u_B$
O(1)        • form C by creating a new cluster root and connecting it to the two cluster roots
              with edge weights $\frac{1}{2}\left(D(A,B)+(u_A-u_B)\right)$ and $\frac{1}{2}\left(D(A,B)+(u_B-u_A)\right)$ respectively.
O(1)        • $Z = Z \cup \{C\} - \{A,B\}$
O(n)        • define $D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )$
```

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
O(n)   • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
O(n²)  • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
       • while $|Z|>1$  O(n)
O(n)       • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
O(n²)      • $(A,B) = arg \min_{(A,B)\in Z} D(A,B) - u_A - u_B$
O(1)       • form C by creating a new cluster root and connecting it to the two cluster roots
             with edge weights $\frac{1}{2}(D(A,B)+(u_A-u_B))$ and $\frac{1}{2}(D(A,B)+(u_B-u_A))$ respectively.
O(1)       • $Z = Z \cup \{C\} - \{A,B\}$
O(n)       • define $D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )$
```

Neighbor Joining

Algorithm Given a distance matrix *M* with rows labeled *(1,2,3....n)*

```
O(n)    • let $Z = \{\{1\},\{2\},\{3\},..,\{n\}\}$ (* the set of initial clusters *)
O(n²)   • for all $\{i\},\{j\} \in Z$ set $D(\{i\},\{j\})=M_{i,j}$
        • while $|Z|>1$   O(n)
O(n)        • define $u_A = 1/(n-2) * \Sigma_{F \in Z} D(A,F)$ for all $A \in Z$
O(n²)       • $(A,B) = arg \min_{(A,B) \in Z} D(A,B) - u_A - u_B$
O(1)        • form C by creating a new cluster root and connecting it to the two cluster roots
              with edge weights $\frac{1}{2}\left(D(A,B)+(u_A-u_B)\right)$ and $\frac{1}{2}\left(D(A,B)+(u_B-u_A)\right)$ respectively.
O(1)        • $Z = Z \cup \{C\} - \{A,B\}$
O(n)        • define $D(F,C) = D(C,F) = 1/2 ( D(A,F) + D(B,F) - D(A,B) )$
```

O(n³) total time

Unweighted Pair Group Method with Arithmetic Mean (UPGMA)

Similar to Neighbor Joining, but does not choose the clusters that are most different (i.e. the use of u_A values).

Uses an arithmetic mean to calculate new cluster distances.