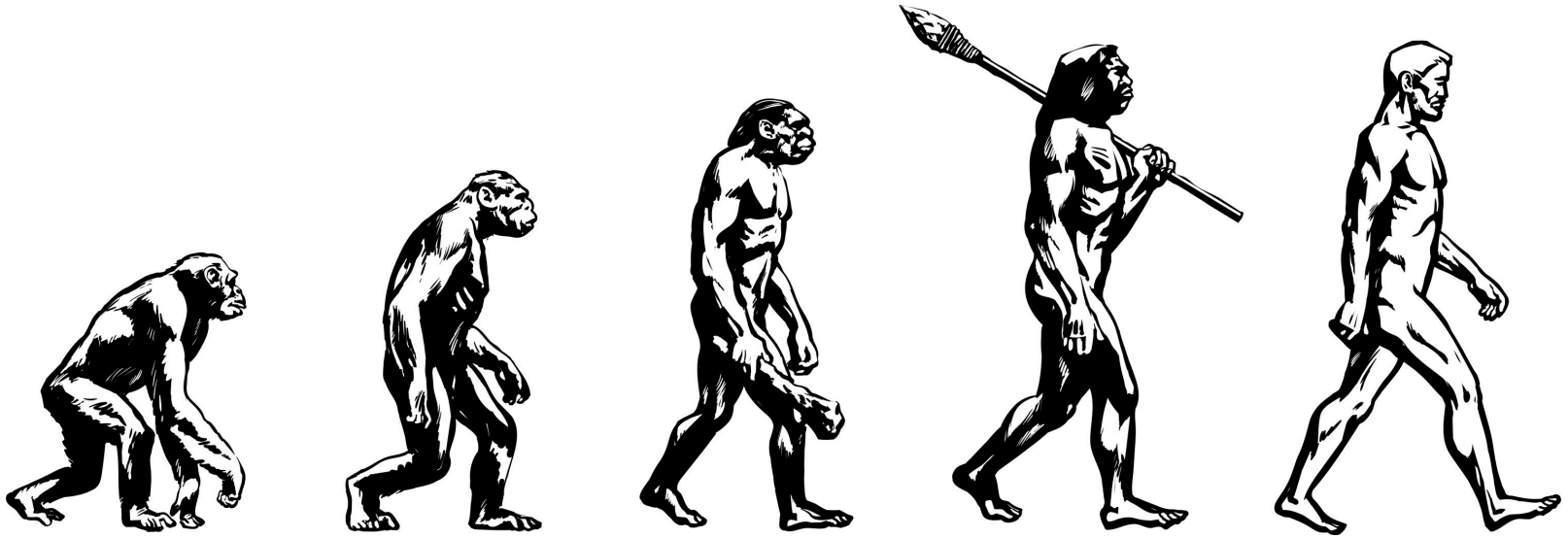
Fun with Phylogenetic Trees

Erin Molloy
University of Maryland, College Park

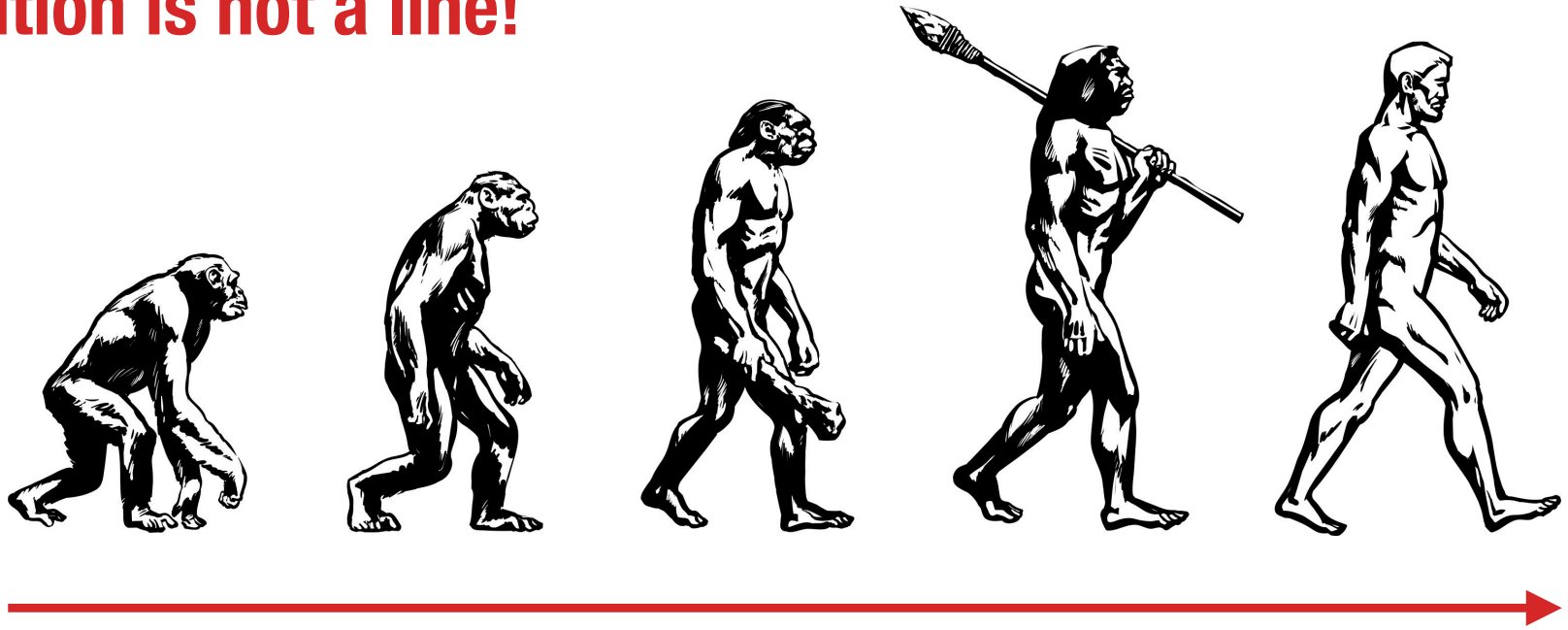
Guest Lecture in CMSC 423 Bioinformatics
April 25, 2023

Who has seen a similar picture?

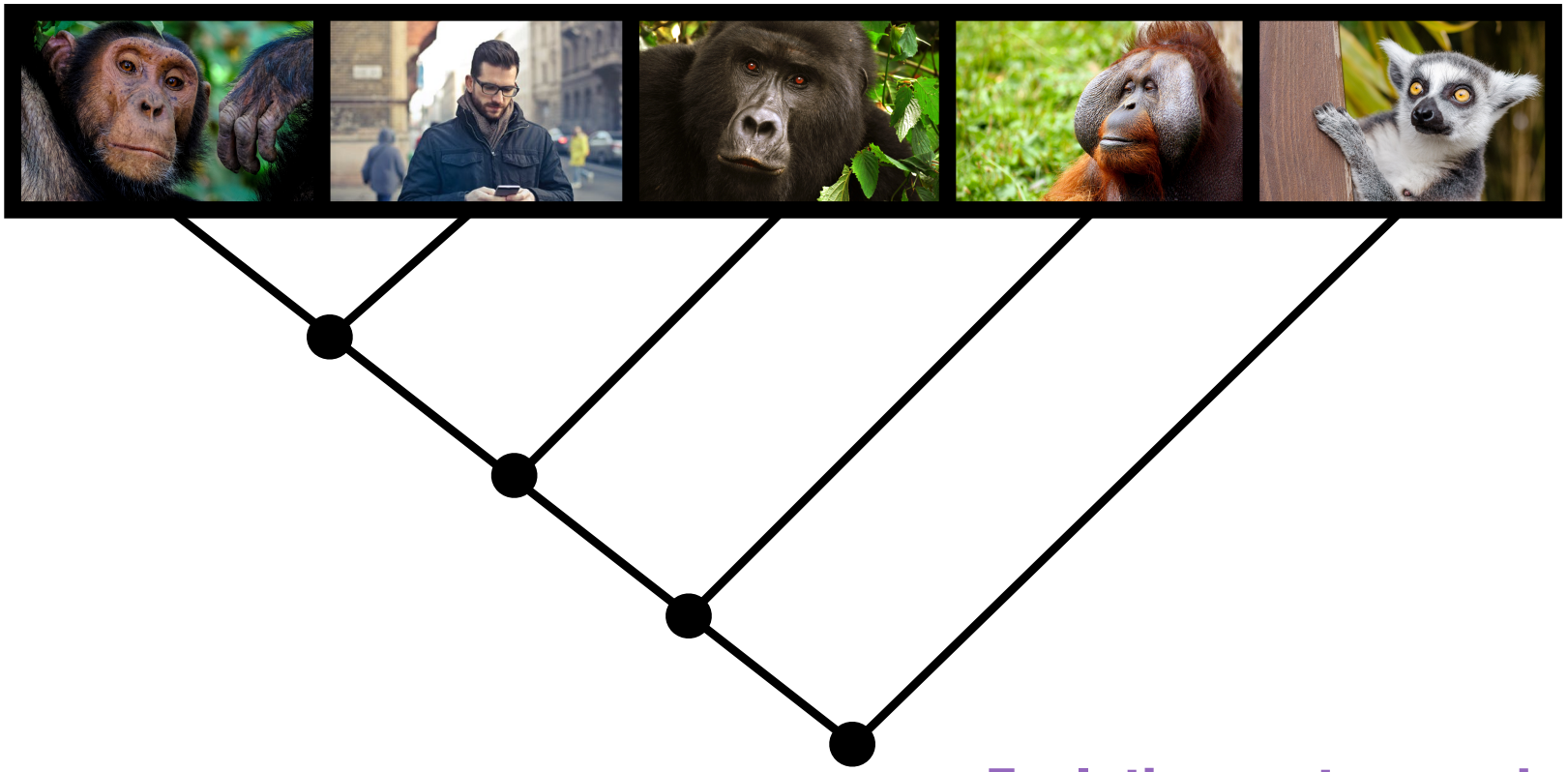


Who has seen a similar picture?

Evolution is not a line!

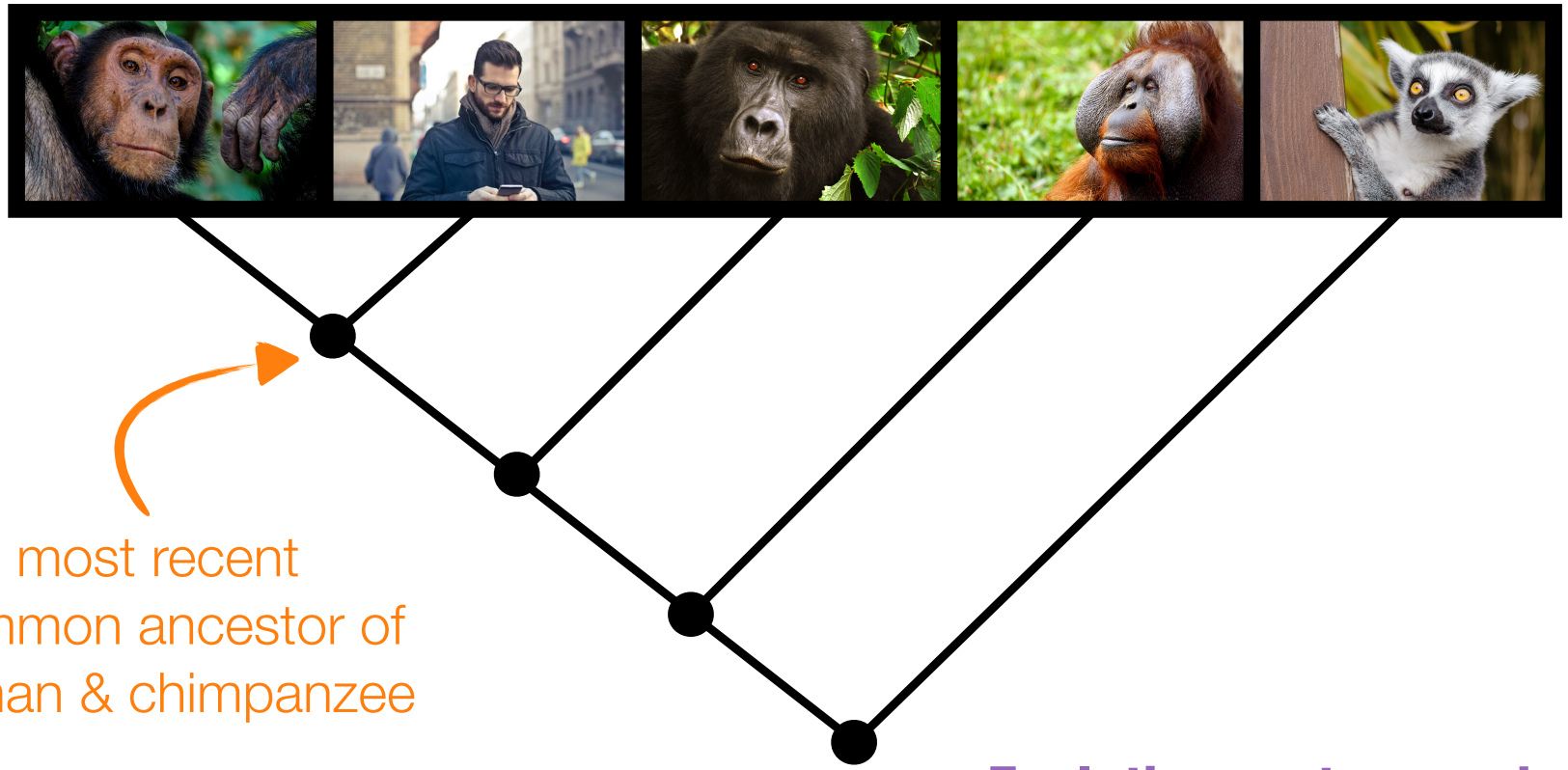


Evolution is a tree!



Evolutionary tree = phylogeny

Evolution is a tree!



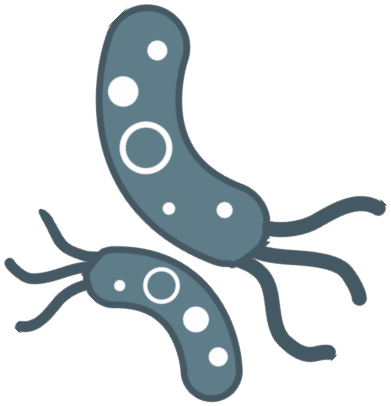
most recent
common ancestor of
human & chimpanzee

Evolutionary tree = phylogeny

Evolutionary trees are used in **bioinformatics analyses**.

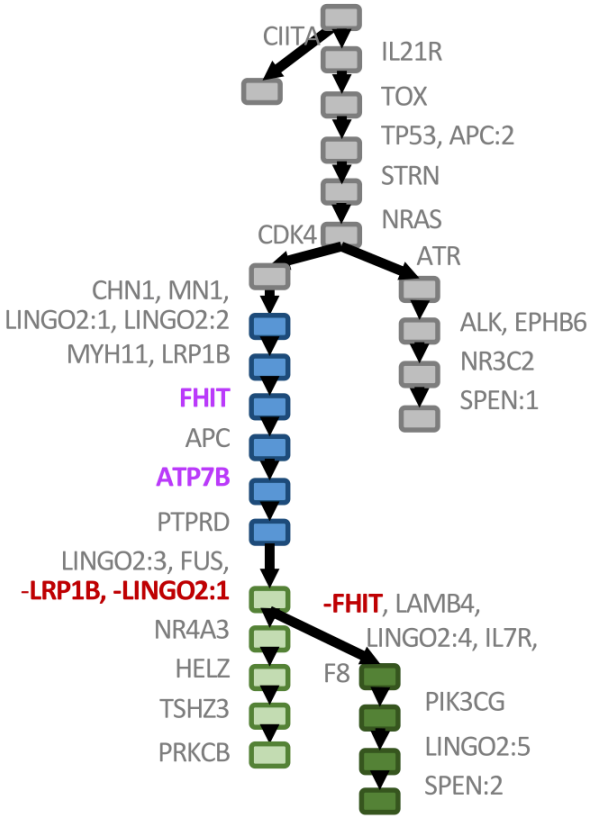
vaccine design & strain tracking

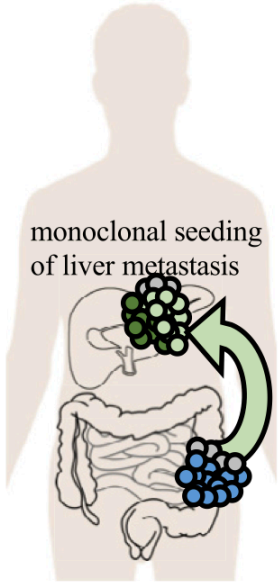
[Qiu et al., 2019; Hodcroft et al., 2021]



microbe identification

[Nguyen et al., 2014; Amato et al., 2019; Meyer et al., 2019; Shah et al., 2021]



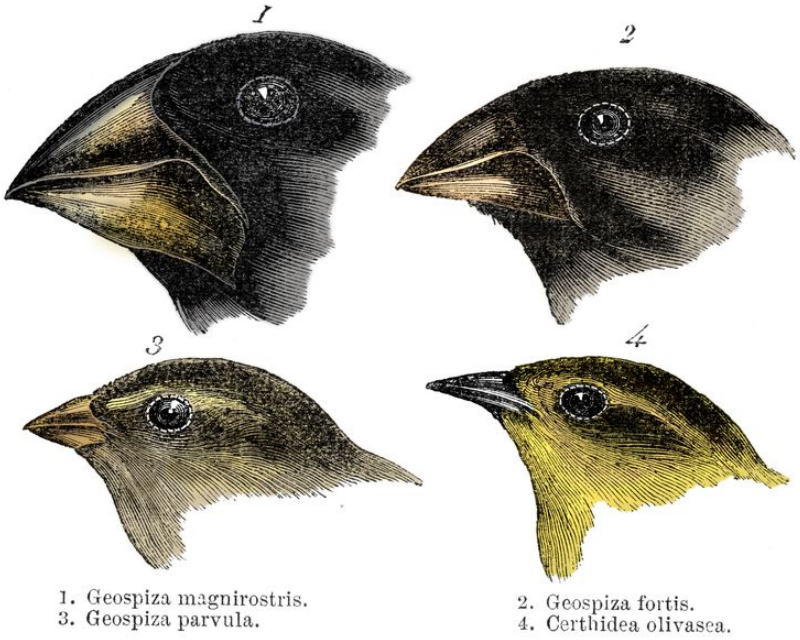


tumor evolution

[Satas et al., 2021]

Evolutionary trees are reconstructed from **data**.

Morphology



Darwin's finches

Genomics

1,000 bat genomes

5,000 insect genomes

10,000 plant genomes

>60,000 vertebrate genomes

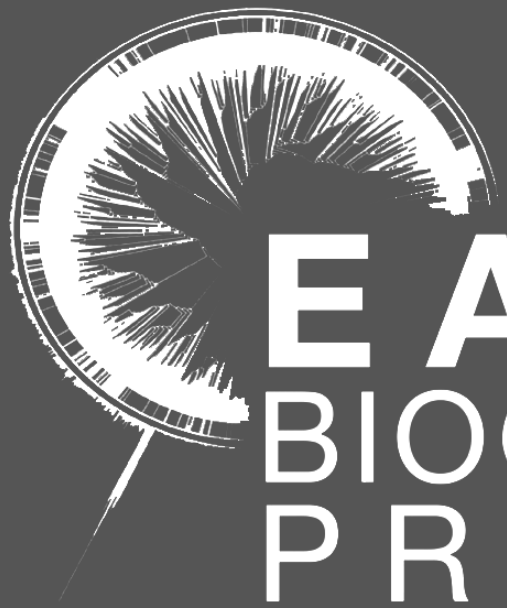
BAT 1K

i5K

10KP

VGP VERTEBRATE GENOMES PROJECT

A PROJECT OF THE G10K CONSORTIUM



EARTH BIOGENOME PROJECT

~1.5 million eukaryotic genomes
in next 10 years

"Resolving the Tree of Life is unquestionably among the most complex scientific problems facing biology and presents challenges much greater than sequencing the human genome."

From: "Assembling the Tree of Life: Harnassing Life's History to Benefit Science and Society," NSF, 2002.

The human genome project "ended" in 2003 with ~92%…

And then it took ~20 more years to "complete" final ~8%.

RESEARCH ARTICLE

HUMAN GENOMICS

The complete sequence of a human genome

Sergey Nurk[1]†, Sergey Koren[1]†, Arang Rhie[1]†, Mikko Rautiainen[1]†, Andrey V. Bzikadze[2], Alla Mikheenko[3], Mitchell R. Vollger[4], Nicolas Altemose[5], Lev Uralsky[6,7], Ariel Gershman[8], Sergey Aganezov[9]‡, Savannah J. Hoyt[10], Mark Diekhans[11], Glennis A. Logsdon[4], Michael Alonge[9], Stylianos E. Antonarakis[12], Matthew Borchers[13], Gerard G. Bouffard[14], Shelise Y. Brooks[14], Gina V. Caldas[15], Nae-Chyun Chen[9], Haoyu Cheng[16,17], Chen-Shan Chin[18], William Chow[19], Leonardo G. de Lima[13], Philip C. Dishuck[4], Richard Durbin[19,20], Tatiana Dvorkina[3], Ian T. Fiddes[21], Giulio Formenti[22,23], Robert S. Fulton[24], Arkarachai Fungtammasan[18], Erik Garrison[11,25], Patrick G. S. Grady[10], Tina A. Graves-Lindsay[26], Ira M. Hall[27], Nancy F. Hansen[28], Gabrielle A. Hartley[10], Marina Haukness[11], Kerstin Howe[19], Michael W. Hunkapiller[29], Chirag Jain[1,30], Miten Jain[11], Erich D. Jarvis[22,23], Peter Kerpedjiev[31], Melanie Kirsche[9], Mikhail Kolmogorov[32], Jonas Korlach[29], Milinn Kremitzki[26], Heng Li[16,17], Valerie V. Maduro[33], Tobias Marschall[34], Ann M. McCartney[1], Jennifer McDaniel[35], Danny E. Miller[4,36], James C. Mullikin[14,28], Eugene W. Myers[37], Nathan D. Olson[35], Benedict Paten[11], Paul Peluso[29], Pavel A. Pevzner[32], David Porubsky[4], Tamara Potapova[13], Evgeny I. Rogaev[6,7,38,39], Jeffrey A. Rosenfeld[40], Steven L. Salzberg[9,41], Valerie A. Schneider[42], Fritz J. Sedlazeck[43], Kishwar Shafin[11], Colin J. Shew[44], Alaina Shumate[41], Ying Sims[19], Arian F. A. Smit[45], Daniela C. Soto[44], Ivan Sović[29,46], Jessica M. Storer[45], Aaron Streets[5,47], Beth A. Sullivan[48], Françoise Thibaud-Nissen[42], James Torrance[19], Justin Wagner[35], Brian P. Walenz[1], Aaron Wenger[29], Jonathan M. D. Wood[19], Chunlin Xiao[42], Stephanie M. Yan[49], Alice C. Young[14], Samantha Zarate[9], Urvashi Surti[50], Rajiv C. McCoy[49], Megan Y. Dennis[44], Ivan A. Alexandrov[3,7,51], Jennifer L. Gerton[13,52], Rachel J. O'Neill[10], Winston Timp[8,41], Justin M. Zook[35], Michael C. Schatz[9,49], Evan E. Eichler[4,53]*, Karen H. Miga[11,54]*, Adam M. Phillippy[1]*

We cannot analyze these forthcoming **BIG** datasets with the methods we have.

Challenge #1: Too many species

Best methods are heuristics for NP-hard optimization problems

Solution space (i.e., set of all possible phylogenetic trees) grows exponentially in number of species!

Challenge #1:
Too many species

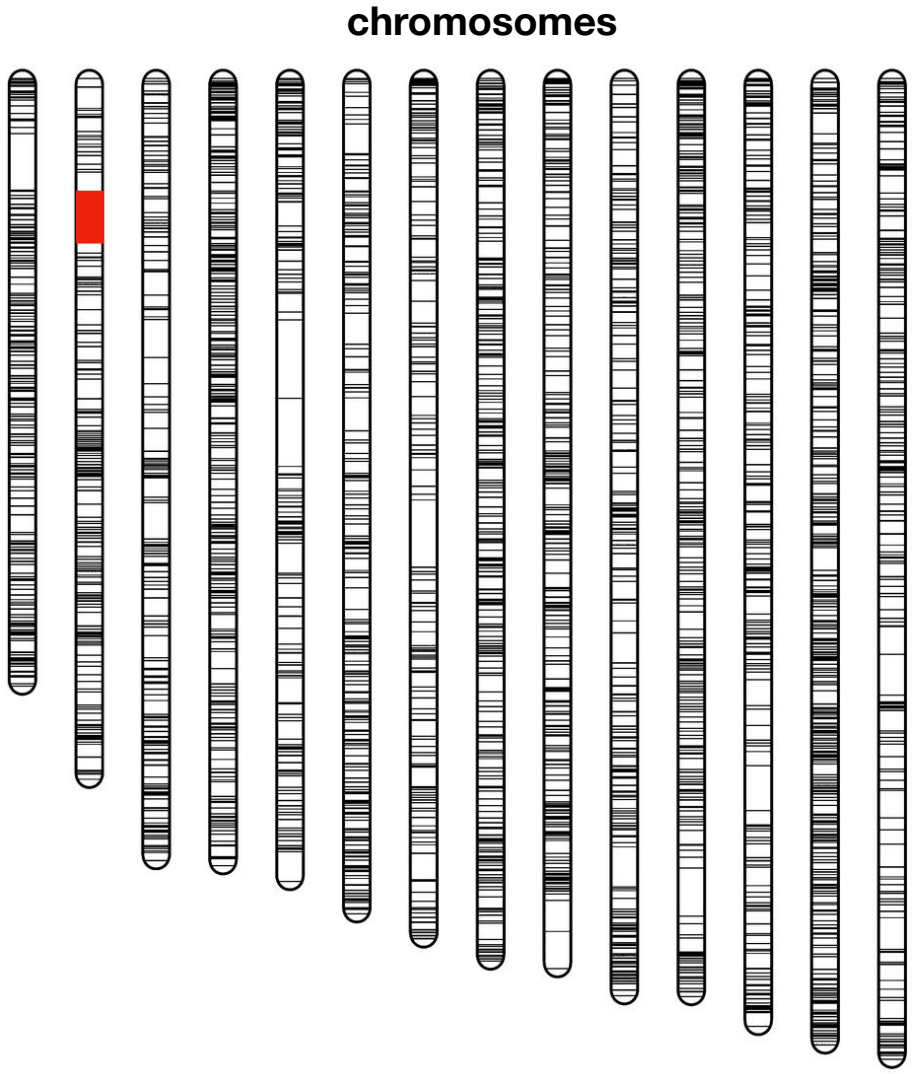
# leaves	#trees
4	3
5	15
6	105
7	945
8	10,395
9	135,135
10	2,027,025

Challenge #2: Genome-scale data

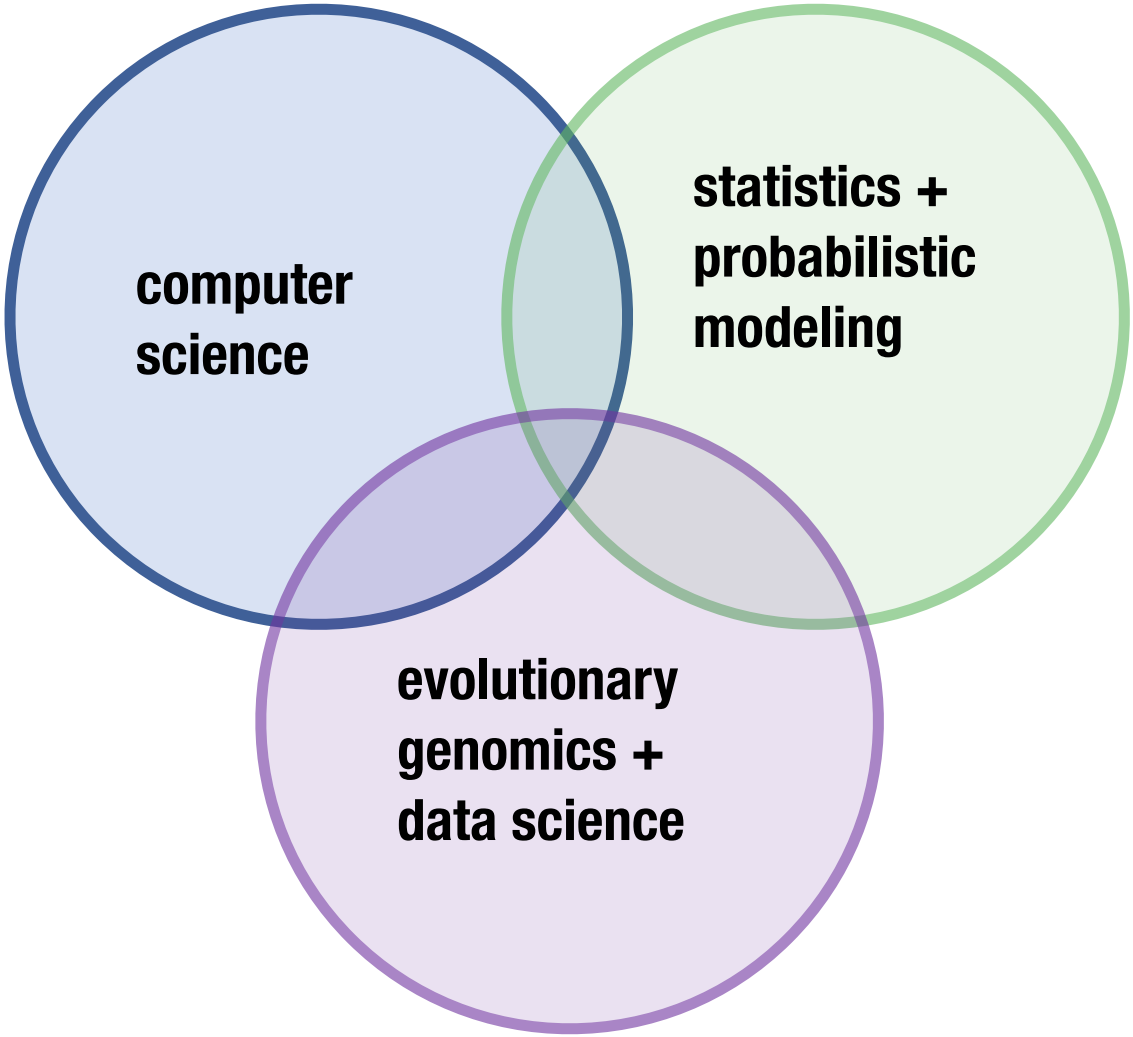
More data per species

Error, missing data, etc.

Modeling evolution of single **gene** vs. **genome** (many genes)



Addressing these challenges requires **interdisciplinary** research!



Agenda

Part 1: Perfect Phylogenies

Part 2: Small Parsimony Problem & Fitch's Algorithm

Part 3: Large Parsimony Problem

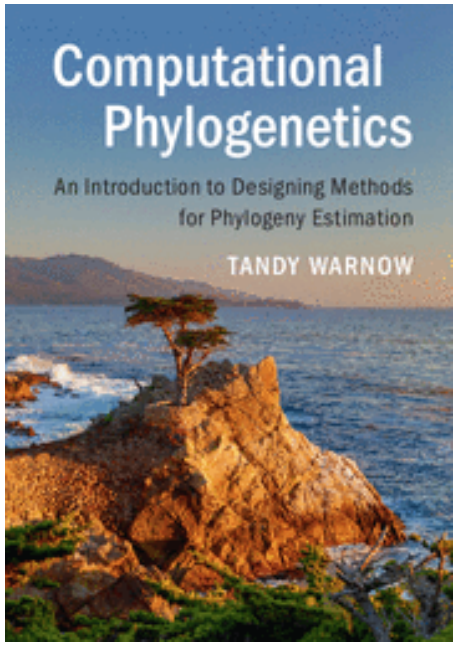
Part 4: Maximum Parsimony Methods

Acknowledgements

Material in slides based on *Computational Phylogenetics* by Prof. Tandy Warnow.

Tandy Warnow

University of Illinois at Urbana-Champaign



Let's begin with a formal definition of **phylogenetic tree**.

Definition. A *rooted phylogenetic tree* T is a triplet (t, S, ϕ), where

- t is a **rooted binary tree**
- S is a **set** of labels typically representing species, and
- ϕ is a **bijection** i.e., 1-1 function, mapping leaf vertices in t to labels in S

In this presentation

For simplicity, we do not make an explicit distinction between leaf nodes and their labels.

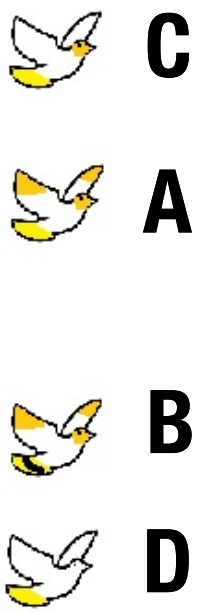
We will simply say that T is a phylogenetic tree on label set S, omitting the reference to ϕ.

We let

- $L(T)$ denote the **leaf set** of T
- $V(T)$ denote the **vertex set** of T
- $E(T)$ denote the **edge set** of T

Now we have a definition for phylogenetic trees — but what about the **data**?

Let's consider a simple example.



Definition. A *character* c is a surjection $c : S \to \{1,2,\ldots,k\}$ mapping labels (species) onto k states.


C


A

state 0 = trait is absent
state 1 = trait present


B


D

Definition. A *character* c is a surjection $c : S \to \{1,2,\ldots,k\}$ mapping labels (species) onto k states.

	c1
	Black Stripe
A =	0
B =	1
C =	0
D =	0


C


A

state 0 = trait is absent
state 1 = trait present


B


D

Class question — What are the characters implied by these birds?

	c1
	Black Stripe
A =	0
B =	1
C =	0
D =	0

state 0 = trait is absent
state 1 = trait present


C


A


B


D

Class question — What are the characters implied by these birds?

	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1


C

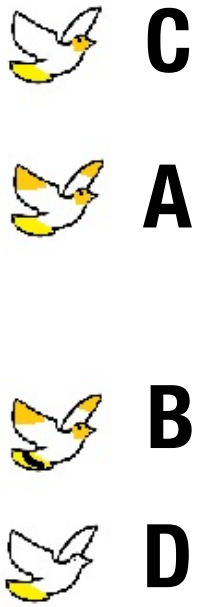

A


B


D

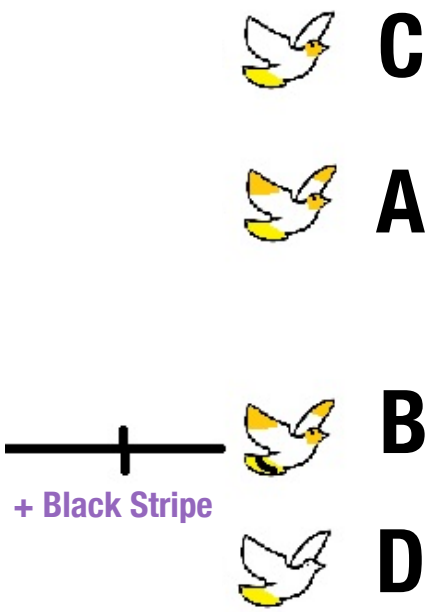
In practice, we are **given data** (characters) and **seek a phylogenetic tree** that best explains it.

Let's consider a simple example, again.



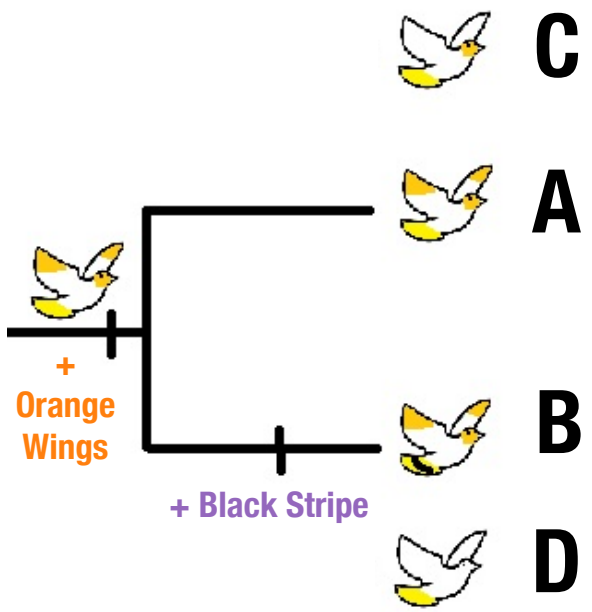
Let's consider a simple example, again.

	c1
	Black Stripe
A =	0
B =	1
C =	0
D =	0



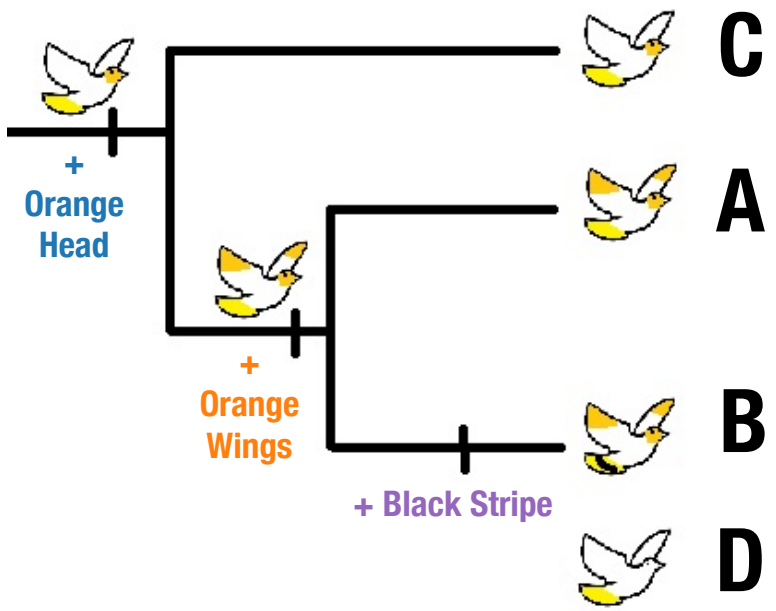
Let's consider a simple example, again.

	c1	c2
	Black Stripe	Orange Wings
A =	0	1
B =	1	1
C =	0	0
D =	0	0



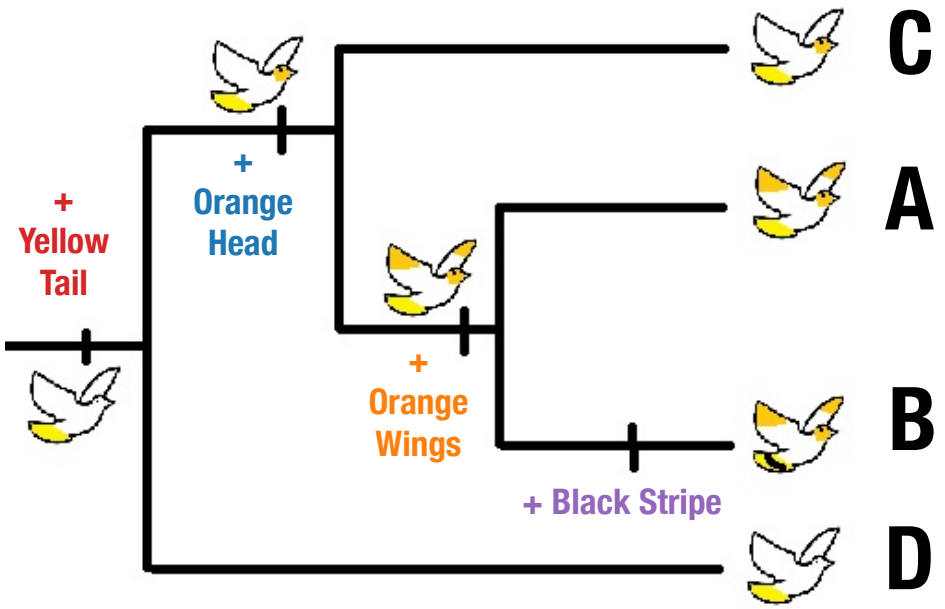
Let's consider a simple example, again.

	c1	c2	c3
	Black Stripe	Orange Wings	Orange Head
A =	0	1	1
B =	1	1	1
C =	0	0	1
D =	0	0	0



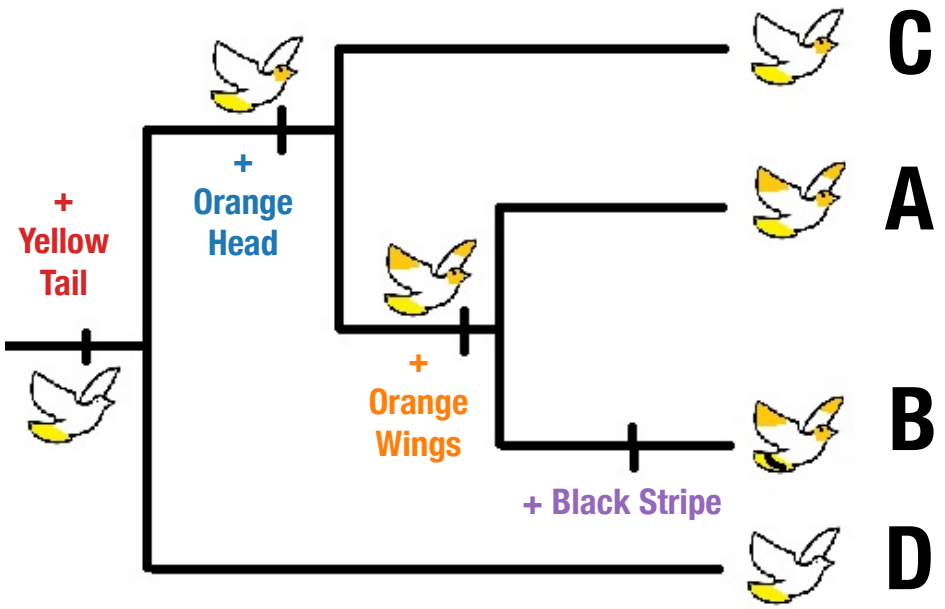
Let's consider a simple example, again.

	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1



Definition. A phylogenetic tree T is called a *perfect phylogeny* for a set $\mathscr{C}$ of characters if every character can be explained by a trait arising on exactly one branch of T.

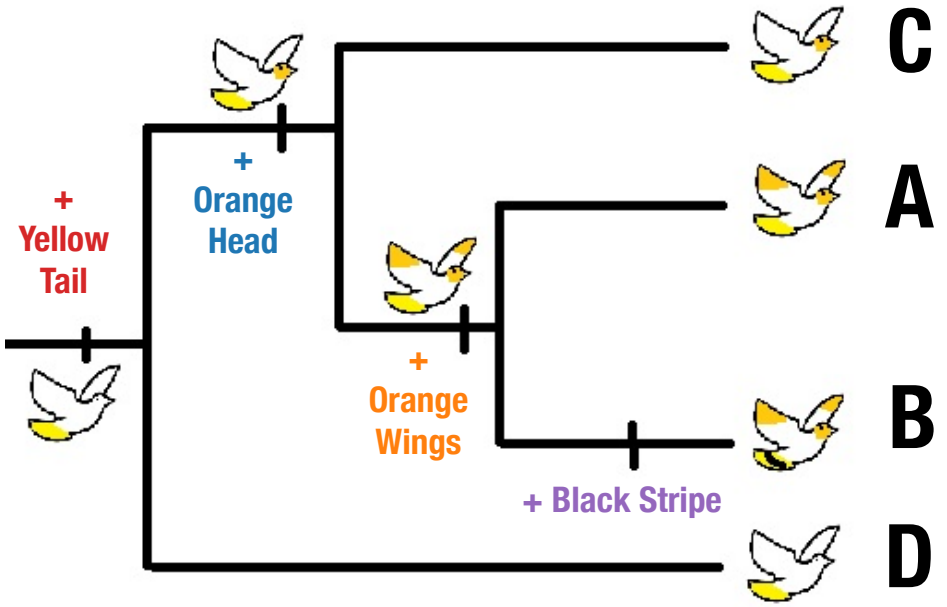
	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1



Now we want an **algorithm** to find a perfect phylogeny for our data if one exists.

Observation — There is a relationship between characters and clades (subsets of species).

	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1



Observation — There is a relationship between characters and clades (subsets of species).

	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1

Algorithm Sketch:

1. Assume 0 is ancestral state and 1 is mutated state. Write down subset of species implied by each character:

{ {B}, {A, B}, {A, B, C}, {A,B,C,D} }

Observation — There is a relationship between characters and clades (subsets of species).

	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1

Algorithm Sketch:

1. Assume 0 is ancestral state and 1 is mutated state. Write down subset of species implied by each character:

   { {B}, {A, B}, {A, B, C}, {A,B,C,D} }

2. Add trivial sets (set of 1 species and set of all species).

Observation — There is a relationship between characters and clades (subsets of species).

	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1

Algorithm Sketch:

1. Assume 0 is ancestral state and 1 is mutated state. Write down subset of species implied by each character:

   { {B}, {A, B}, {A, B, C}, {A,B,C,D} }

2. Add trivial sets (set of 1 species and set of all species).

3. Build tree using Hasse Diagram.

For step 3, need to:

Define a partial order on clades so that the Hasse Diagram produces a phylogeny.

But first some definitions…

Definition. A *relation* on set X is subset of the Cartesian product $X \times X$, which is the set formed by taking exactly two elements from X, in all possible ways.

But first some definitions…

Definition. A *relation* on set X is subset of the Cartesian product $X \times X$, which is the set formed by taking exactly two elements from X, in all possible ways.

Definition. A *partial order* is relation R on set X satisfying three properties:

- $\langle x, y \rangle \in R$ and $\langle y, z \rangle \in R$ implies that $\langle x, z \rangle \in R$ (TRANSITIVITY)
- $\langle x, x \rangle \in R$ for all $x \in X$
- $\langle x, y \rangle \in R$ and $\langle y, x \rangle \in R$ implies $x = y$

But first some definitions…

Definition. The *Hasse Diagram* for a set X with a partial order R is constructed in three steps:

(1) create vertex for each element in X,

(2) add directed edge $x \to y$ if $\langle x, y \rangle \in R$ and $x \neq y$, and

(3) remove arrows implied by transitivity.

Going back to our example…

	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1

Algorithm Sketch:

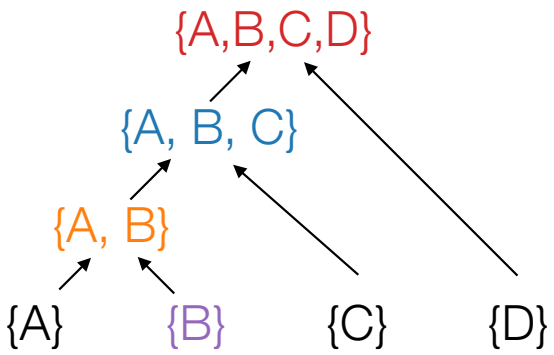
1. Assume 0 is ancestral state and 1 is mutated state. Write down subset of species implied by each character:

   { {B}, {A, B}, {A, B, C}, {A,B,C,D} }

2. Add trivial sets (set of 1 species and set of all species).

3. Build tree using Hasse Diagram.

Going back to our example…



	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1

Algorithm Sketch:

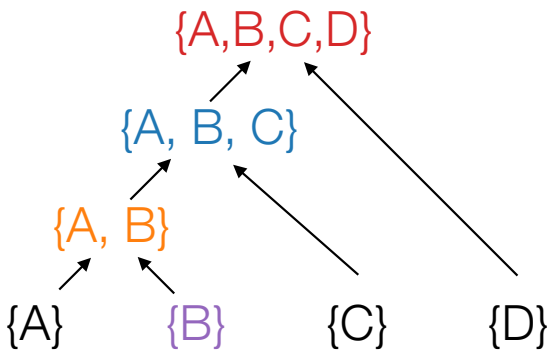
1. Assume 0 is ancestral state and 1 is mutated state. Write down subset of species implied by each character:

   { {B}, {A, B}, {A, B, C}, {A,B,C,D} }

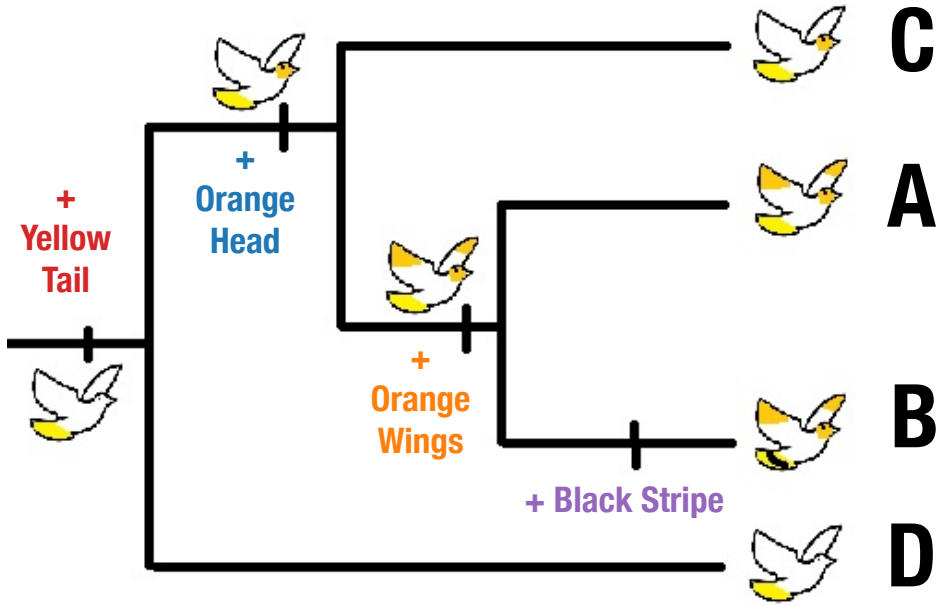
2. Add trivial sets (set of 1 species and set of all species).

3. Build tree using Hasse Diagram.

Going back to our example…



	c1	c2	c3	c4
	Black Stripe	Orange Wings	Orange Head	Yellow Tail
A =	0	1	1	1
B =	1	1	1	1
C =	0	0	1	1
D =	0	0	0	1



Now we have an **algorithm** to find a perfect phylogeny for our data if one exists.

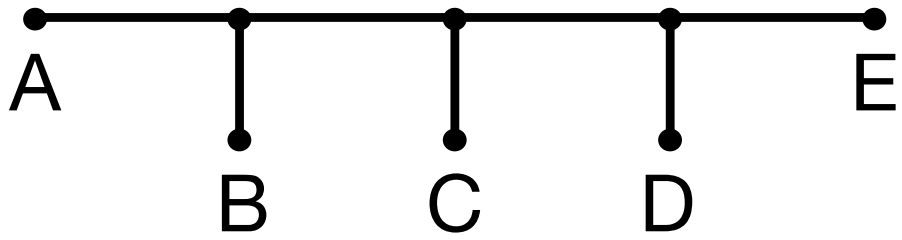
However, perfect phylogenies unlikely to exist in practice!

A perfect phylogenies will exist if characters

- **evolve without homoplasy AND**
- **are correctly called+coded** for all labels in the set S,
  - no error
  - no missing or ambiguous states

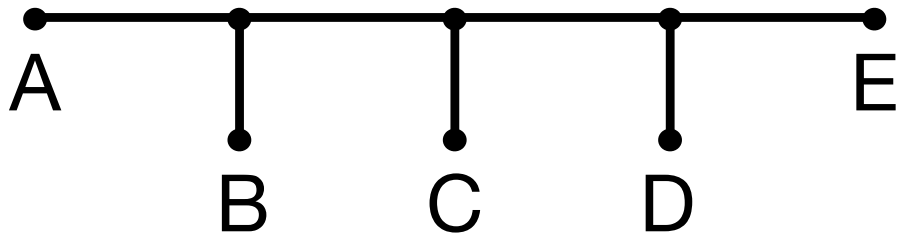
Now let's define **homoplasy** for the case where characters are *undirected* (i.e., we don't know which state is ancestral or mutated).

	c_1	c_2
A =	**1**	**1**
B =	**0**	**1**
C =	**0**	**0**
D =	**0**	**0**
E =	**1**	**0**



Definition. Given a tree T and a k-state character c, we say that c evolves without *homoplasy* if the internal nodes can be labeled with states so that each substitution produces a new state.

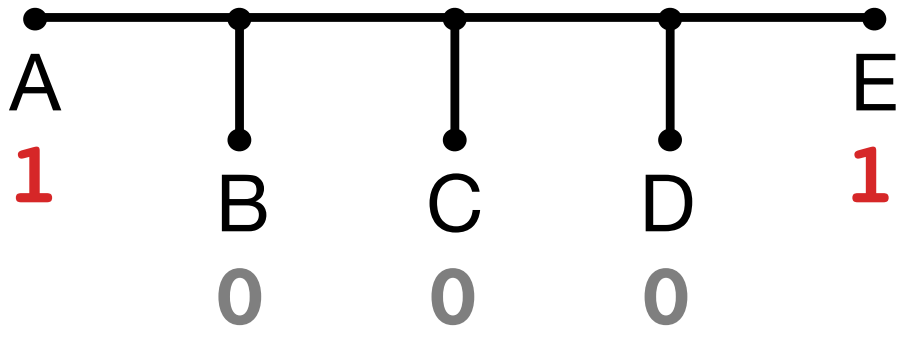
	c_1	c_2
A =	**1**	**1**
B =	**0**	**1**
C =	**0**	**0**
D =	**0**	**0**
E =	**1**	**0**



Definition. Given a tree T and a k-state character c, we say that c evolves without *homoplasy* if the internal nodes can be labeled with states so that each substitution produces a new state.

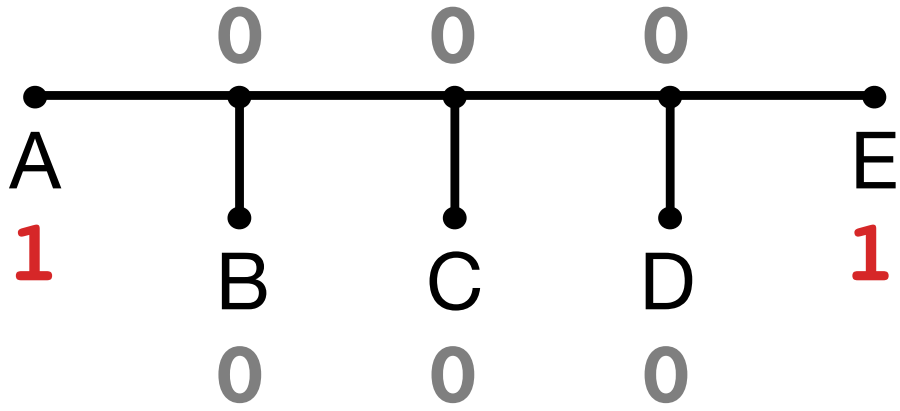
Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

	c_1	c_2
A =	1	1
B =	0	1
C =	0	0
D =	0	0
E =	1	0



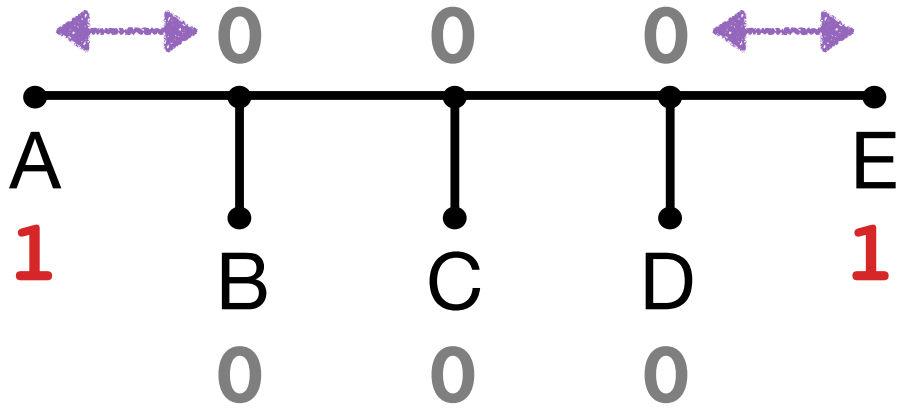
Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

	c_1	c_2
A =	1	1
B =	0	1
C =	0	0
D =	0	0
E =	1	0



Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

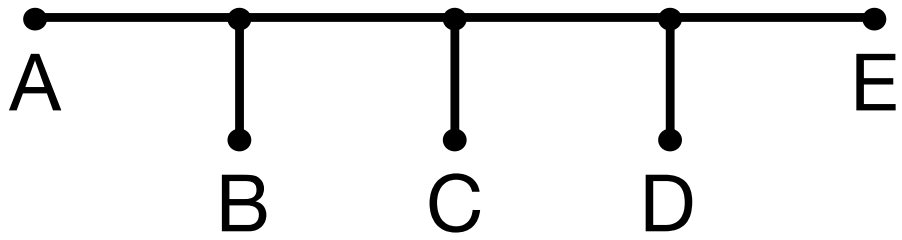
	c_1	c_2
A =	1	1
B =	0	1
C =	0	0
D =	0	0
E =	1	0



At least 2 substitutions is required to explain c1, across all possible labelings of the internal nodes… so it evolved with homoplasy!

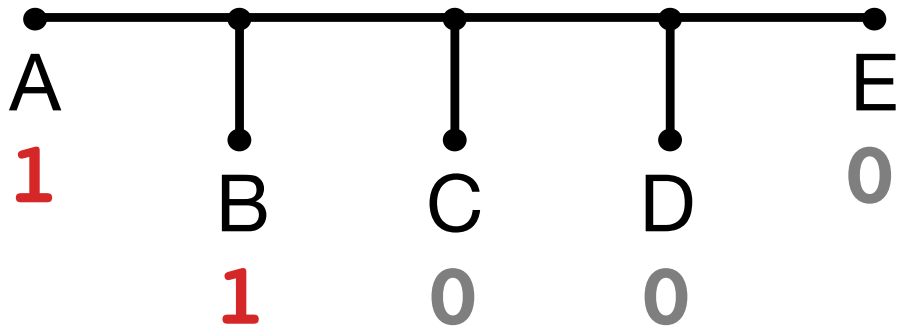
Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

	c_1	c_2
A =	**1**	**1**
B =	**0**	**1**
C =	**0**	**0**
D =	**0**	**0**
E =	**1**	**0**



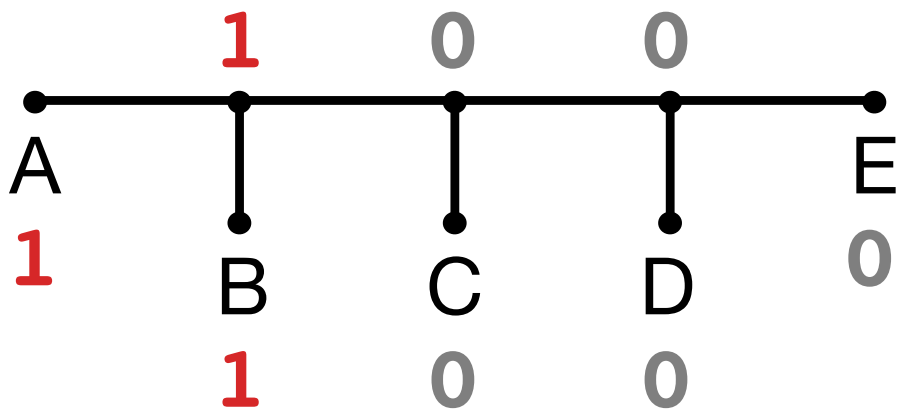
Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

	c_1	c_2
A =	1	1
B =	0	1
C =	0	0
D =	0	0
E =	1	0



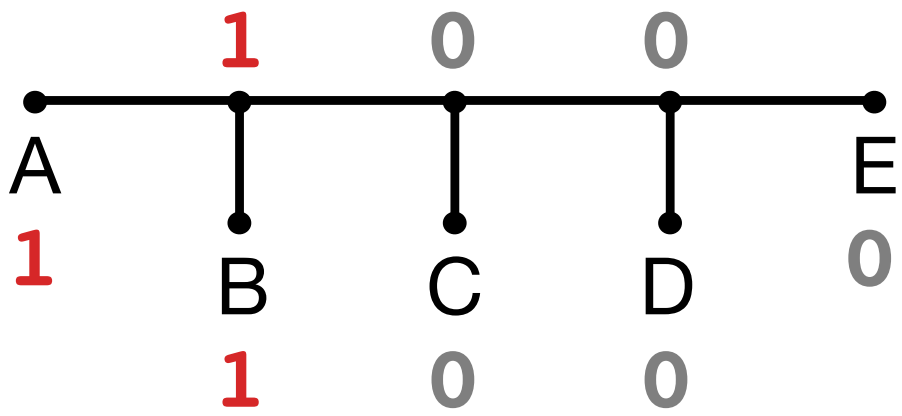
Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

	c_1	c_2
A =	1	1
B =	0	1
C =	0	0
D =	0	0
E =	1	0



Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

	c_1	c_2
A =	1	1
B =	0	1
C =	0	0
D =	0	0
E =	1	0



Only 1 substitution needed to explain c2, across all possible labelings of the internal nodes… so it evolved WITHOUT homoplasy!

Remark. Given a tree T and a k-state character, the character evolves without *homoplasy* if it can be explained with $k-1$ substitutions.

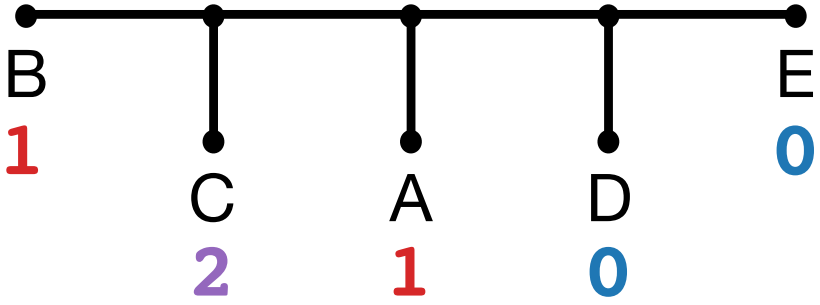
Agenda

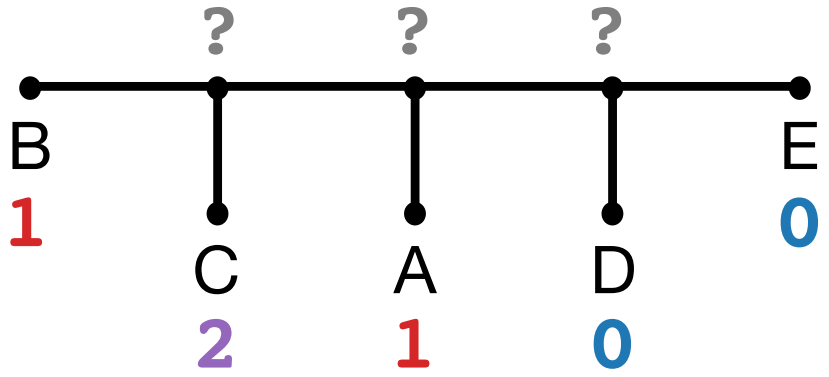
- **Part 1:** Perfect Phylogenies
- **Part 2:** Small Parsimony Problem & Fitch's Algorithm
- **Part 3:** Large Parsimony Problem
- **Part 4:** Maximum Parsimony Methods

Moving on… we want to compute the unordered parsimony score to determine whether an undirected character evolves with homoplasy!

Definition 3 (parsimony score). Given a tree T and a character c, both on label set S, the *parsimony score*, denoted $length(T, c)$, is the *minimum # of substitutions* required to explain the states at the leaves.

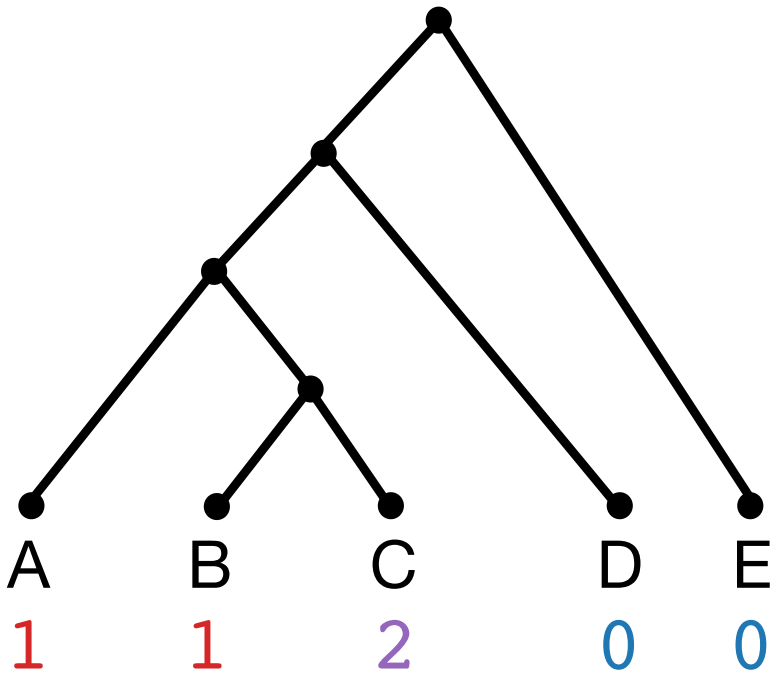
Small Parsimony Problem





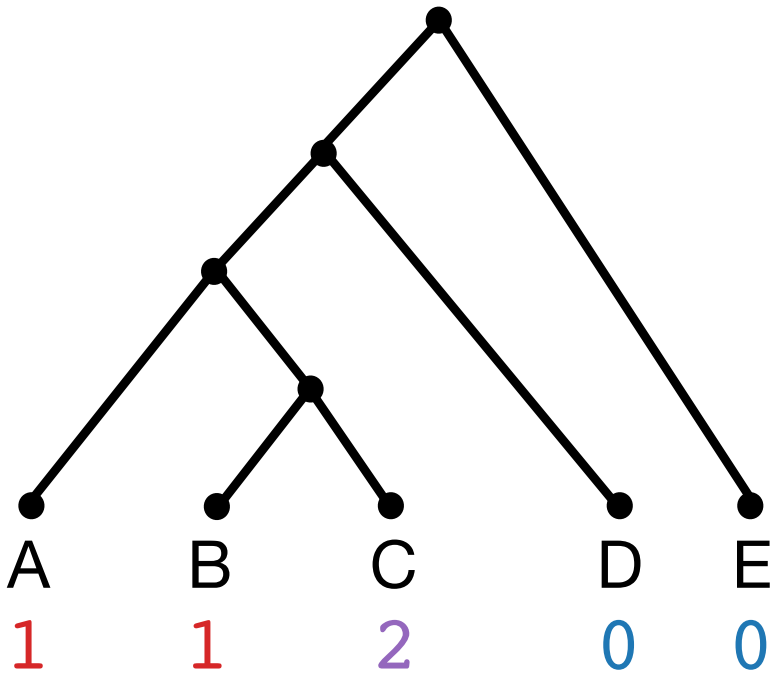
Input. The pair (T, c), where T is a an unrooted binary phylogenetic tree and c is a character, both on label set S.

Output. An assignment of character states to the internal nodes of T to minimize the # of substitutions, i.e. the # of edges $e = (u, v)$ for which $c(u) \neq c(v)$



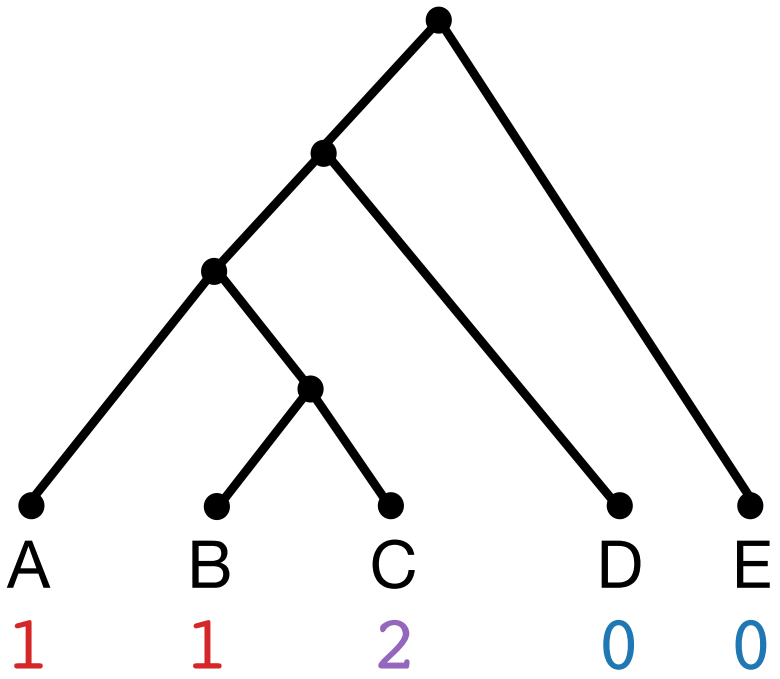
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r



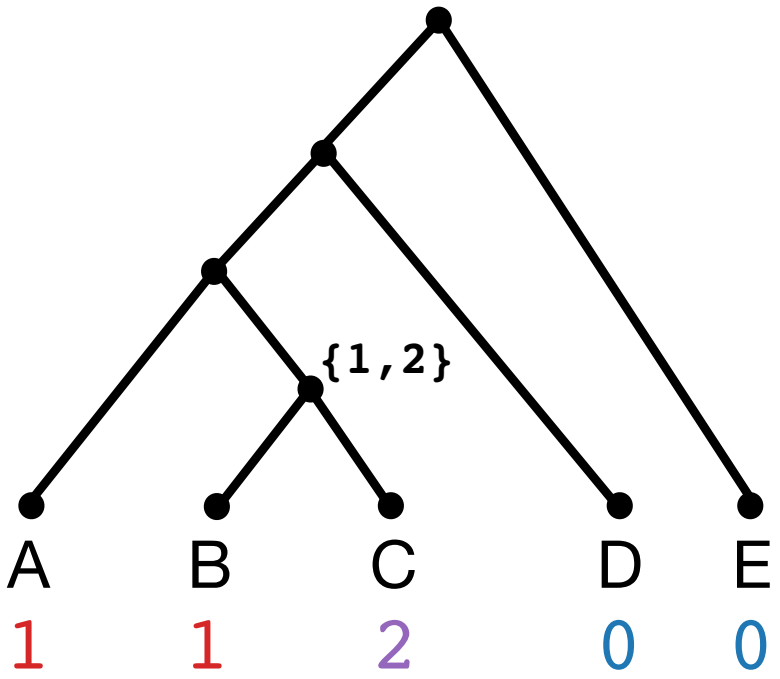
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$



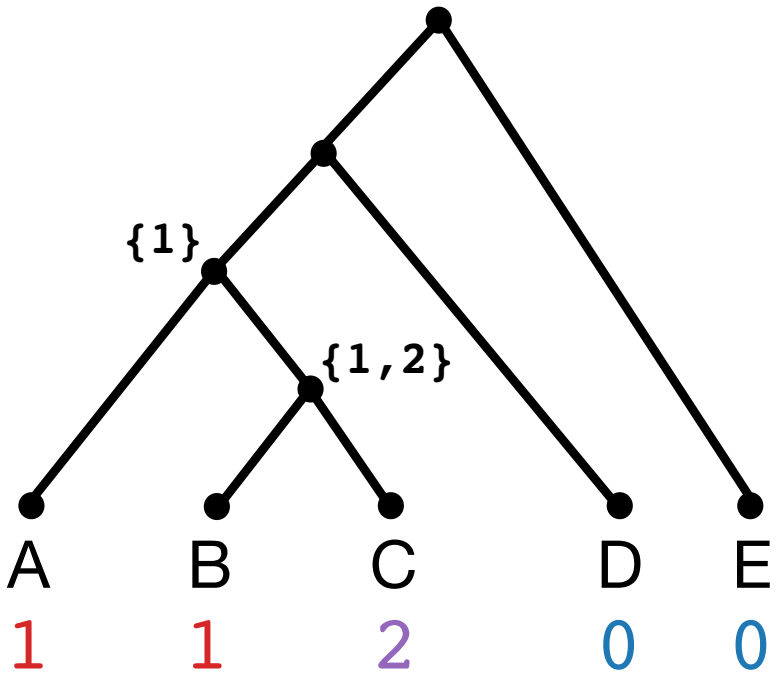
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$



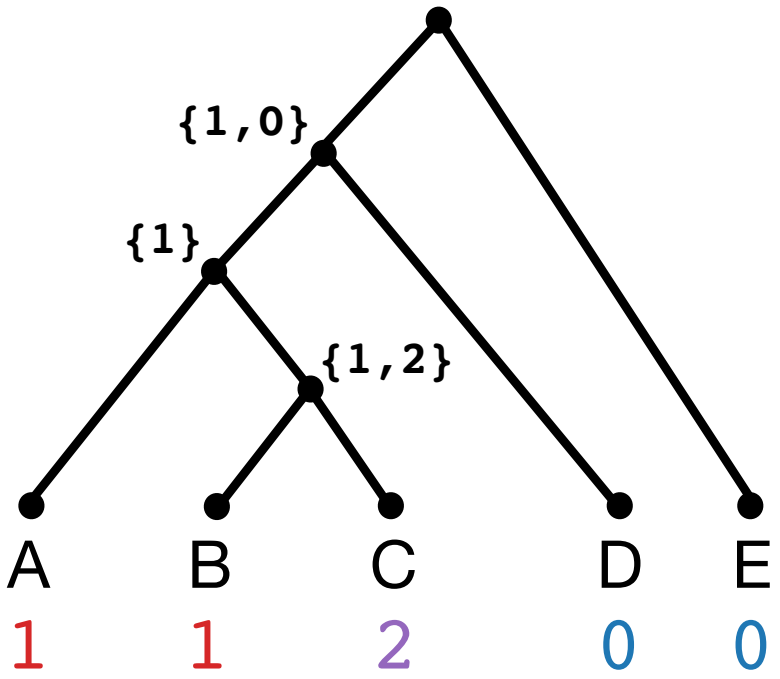
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$



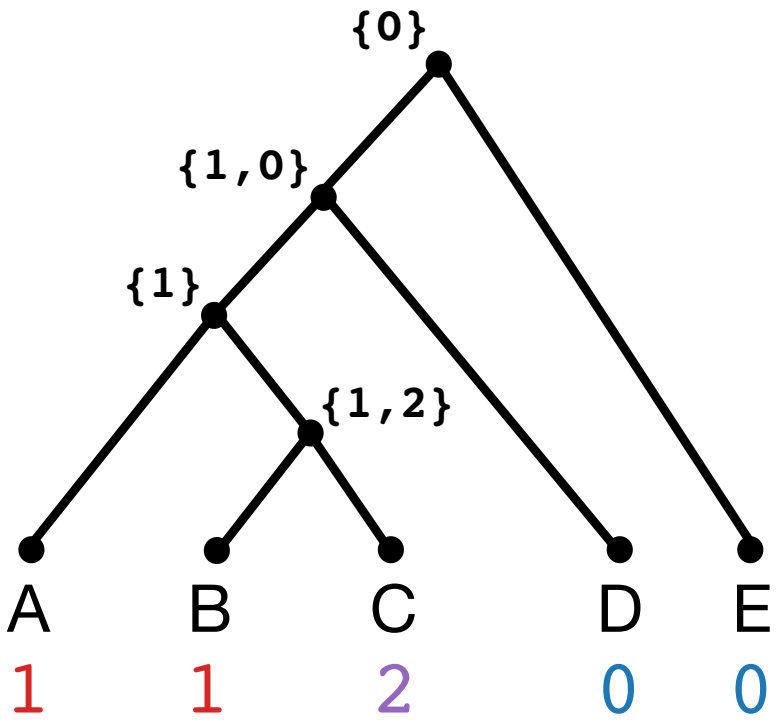
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$



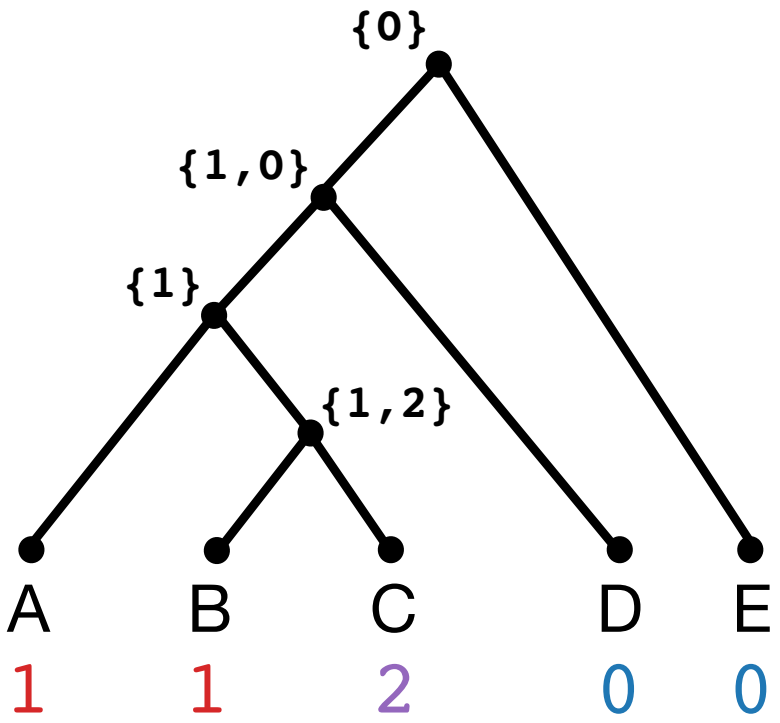
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$



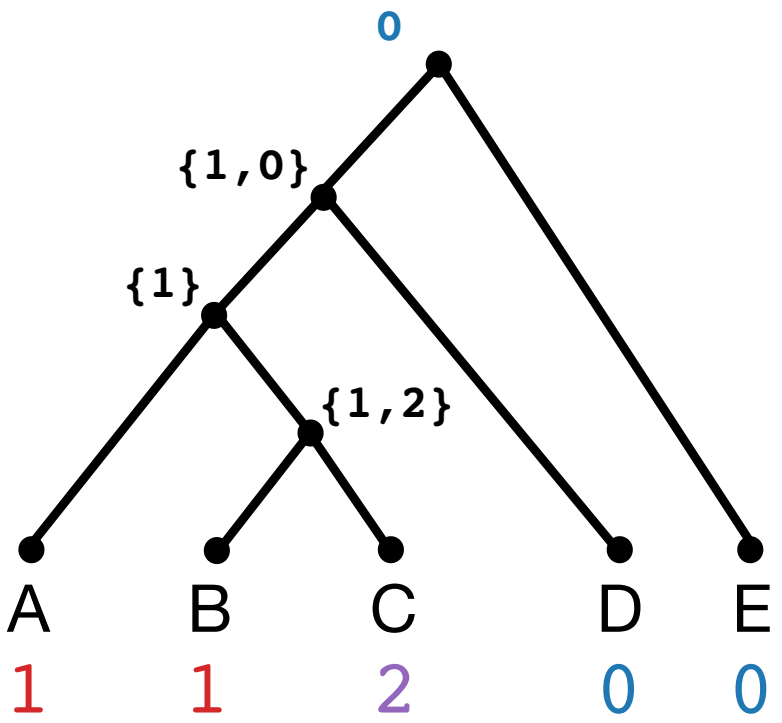
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$



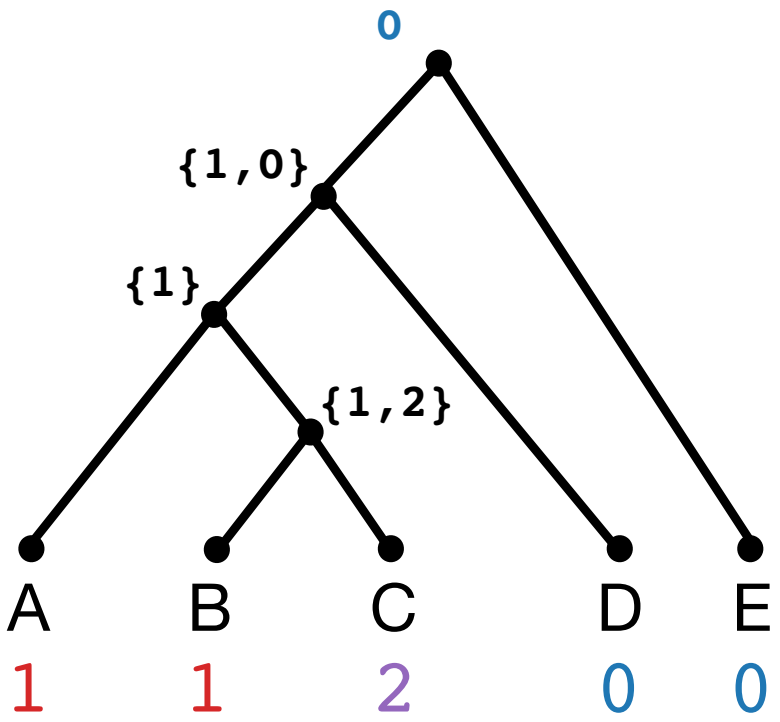
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root



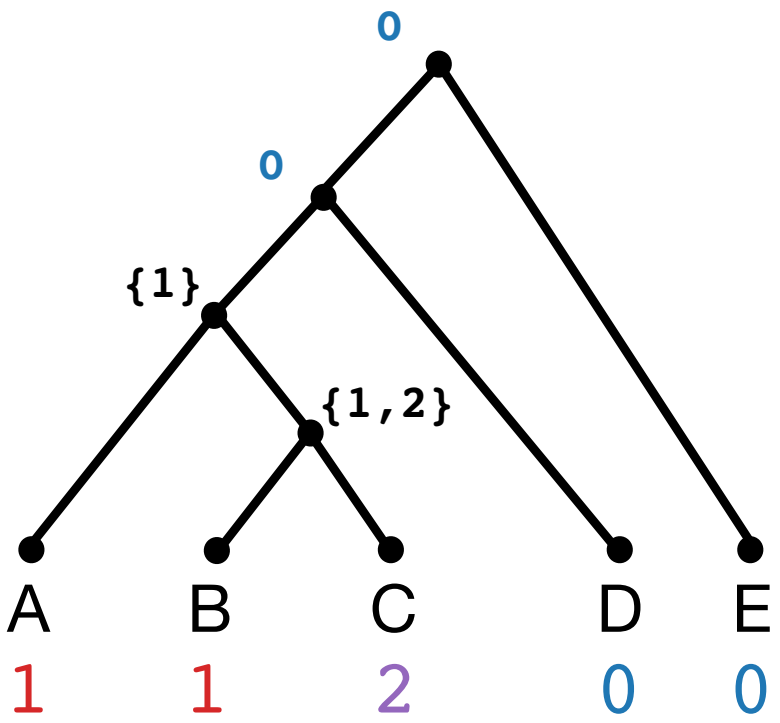
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root



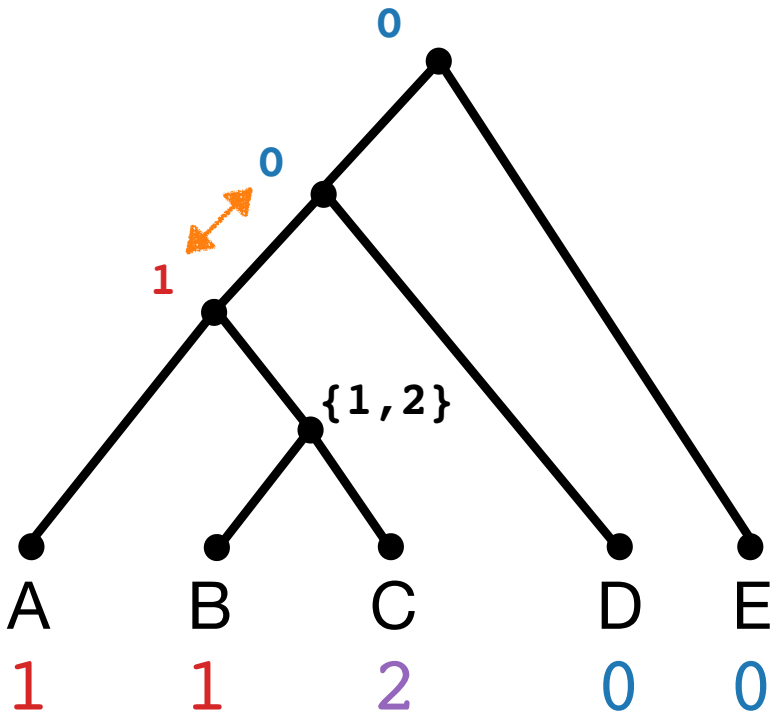
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root
5. Perform a pre-order traversal of T and for each vertex $v \in V(T) \backslash L(T) \backslash \{r\}$:
   a. $u \leftarrow$ parent of v
   b. If $c(u) \in A(v)$: $c(v) \leftarrow c(u)$
   c. Else: $c(v) \leftarrow$ arbitrary state in $A(u)$



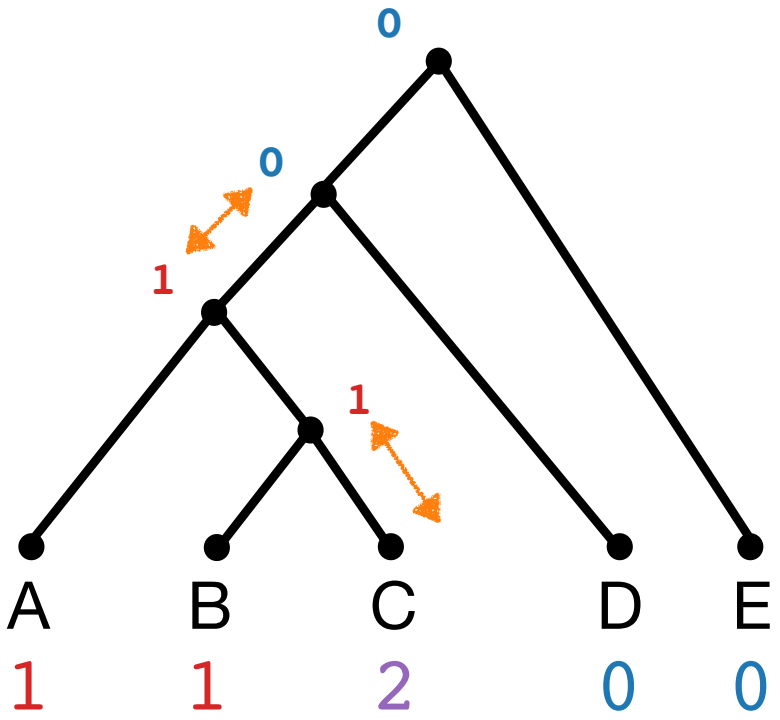
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root
5. Perform a pre-order traversal of T and for each vertex $v \in V(T) \backslash L(T) \backslash \{r\}$:
   a. $u \leftarrow$ parent of v
   b. If $c(u) \in A(v)$: $c(v) \leftarrow c(u)$
   c. Else: $c(v) \leftarrow$ arbitrary state in $A(u)$



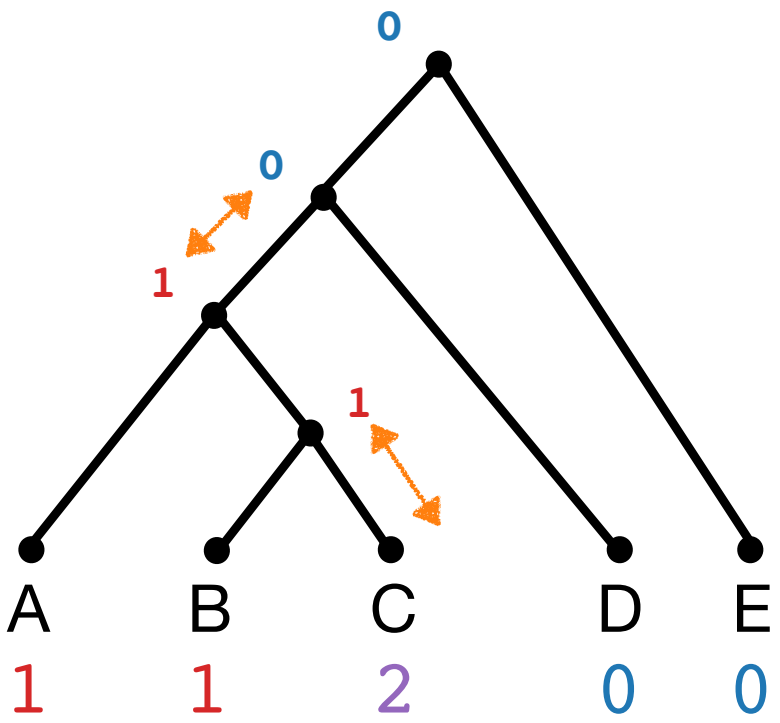
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root
5. Perform a pre-order traversal of T and for each vertex $v \in V(T) \backslash L(T) \backslash \{r\}$:
   a. $u \leftarrow$ parent of v
   b. If $c(u) \in A(v)$: $c(v) \leftarrow c(u)$
   c. Else: $c(v) \leftarrow$ arbitrary state in $A(u)$



FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root
5. Perform a pre-order traversal of T and for each vertex $v \in V(T) \backslash L(T) \backslash \{r\}$:
   a. $u \leftarrow$ parent of v
   b. If $c(u) \in A(v)$: $c(v) \leftarrow c(u)$
   c. Else: $c(v) \leftarrow$ arbitrary state in $A(u)$



FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root
5. Perform a pre-order traversal of T and for each vertex $v \in V(T) \backslash L(T) \backslash \{r\}$:
   a. $u \leftarrow$ parent of v
   b. If $c(u) \in A(v)$: $c(v) \leftarrow c(u)$
   c. Else: $c(v) \leftarrow$ arbitrary state in $A(u)$
6. Return c minus the root

Class Exercise — modify this algorithm to return the parsimony score?

FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root
5. Perform a pre-order traversal of T and for each vertex $v \in V(T) \backslash L(T) \backslash \{r\}$:
   a. $u \leftarrow$ parent of v
   b. If $c(u) \in A(v)$: $c(v) \leftarrow c(u)$
   c. Else: $c(v) \leftarrow$ arbitrary state in $A(u)$
6. Return c minus the root

Class Exercise — modify this algorithm to return the parsimony score?

Add 1 to the parsimony score every time you hit this line!

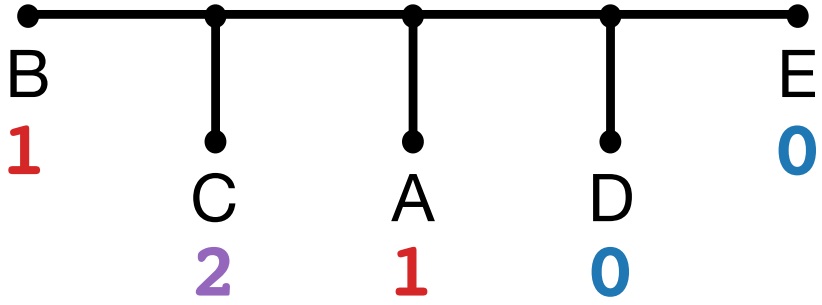
Exit at the end of step 3.

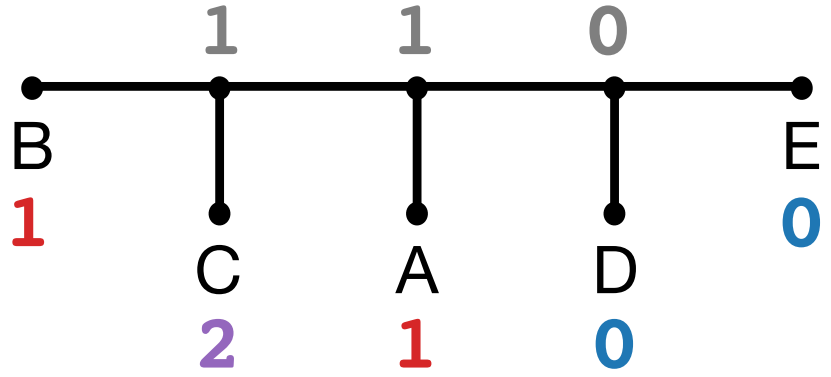
FitchAlgorithm(T, c):

1. Root T, subdividing an arbitrary edge with root r
2. For each $l \in L(T)$: $A(l) \leftarrow \{c(l)\}$
3. Perform a post-order traversal of T and for each vertex $v \in V(T) \backslash L(T)$:
   a. $(w, w') \leftarrow$ children of v
   b. If $A(w) \cap A(w') \neq \emptyset$: $A(v) \leftarrow A(w) \cap A(w')$
   c. Else: $A(v) \leftarrow A(w) \cup A(w')$
4. $c(r) \leftarrow$ arbitrary state in $A(r)$ to root
5. Perform a pre-order traversal of T and for each vertex $v \in V(T) \backslash L(T) \backslash \{r\}$:
   a. $u \leftarrow$ parent of v
   b. If $c(u) \in A(v)$: $c(v) \leftarrow c(u)$
   c. Else: $c(v) \leftarrow$ arbitrary state in $A(u)$
6. Return c minus the root

Small Parsimony Problem

score = 2





Input. The pair (T, c), where T is a an unrooted binary phylogenetic tree and c is a character, both on label set S.

Output. An assignment of character states to the internal nodes of T to minimize the # of substitutions, i.e. the # of edges $e = (u, v)$ for which $c(u) \neq c(v)$

We covered Fitch's algorithm, in which all substitutions have cost 1. Sankoff's algorithm generalizes this idea by allowing substitutions to have different costs!

ToDo: What is the time complexity of Fitch's algorithm…

To prove correctness, define subproblems $Cost(v, x)$ which is the optimal parsimony score of rooted subtree T_v given the assignment $c(v) = x$.

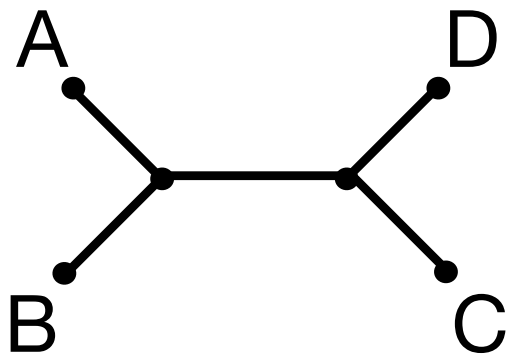
Show this holds for base case (leaves), make inductive hypothesis, and then show it holds for some vertex v.

Agenda

- **Part 1:** Perfect Phylogenies
- **Part 2:** Small Parsimony Problem & Fitch's Algorithm
- **Part 3:** Large Parsimony Problem
- **Part 4:** Maximum Parsimony Methods

Maximum Parsimony (aka Large Parsimony Problem)

	c_1	c_2
A =	**0**	**0**
B =	**0**	**1**
C =	**1**	**0**
D =	**1**	**0**



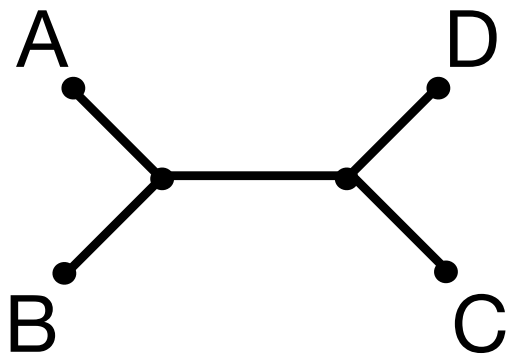
Input. A set $\mathscr{C}$ of k-state character , each on label set S.

Output. A tree T on label set S that *minimizes* the total # of *substitutions* required to explain $\mathscr{C}$.

NP-hard (Foulds and Graham, 1982)

Maximum Parsimony (aka Large Parsimony Problem)

	c_1	c_2
A =	**0**	**0**
B =	**0**	**1**
C =	**1**	**0**
D =	**1**	**0**



Input. A set $\mathscr{C}$ of k-state character , each on label set S.

Output. A tree T on label set S that *minimizes* the total # of *substitutions* required to explain $\mathscr{C}$.

Agenda

- **Part 1:** Perfect Phylogenies
- **Part 2:** Small Parsimony Problem & Fitch's Algorithm
- **Part 3:** Large Parsimony Problem
- **Part 4:** Maximum Parsimony Methods

MP is NP-hard.
Now what?

MP is NP-hard.
Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

MP is NP-hard.
Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

But it's challenging because **tree space** has $(2n-5)!!$ unrooted trees on n leaves!

n	#trees
4	3
5	15
6	105
7	945
8	10,395
9	135,135
10	2,027,025

MP is NP-hard. Now what?

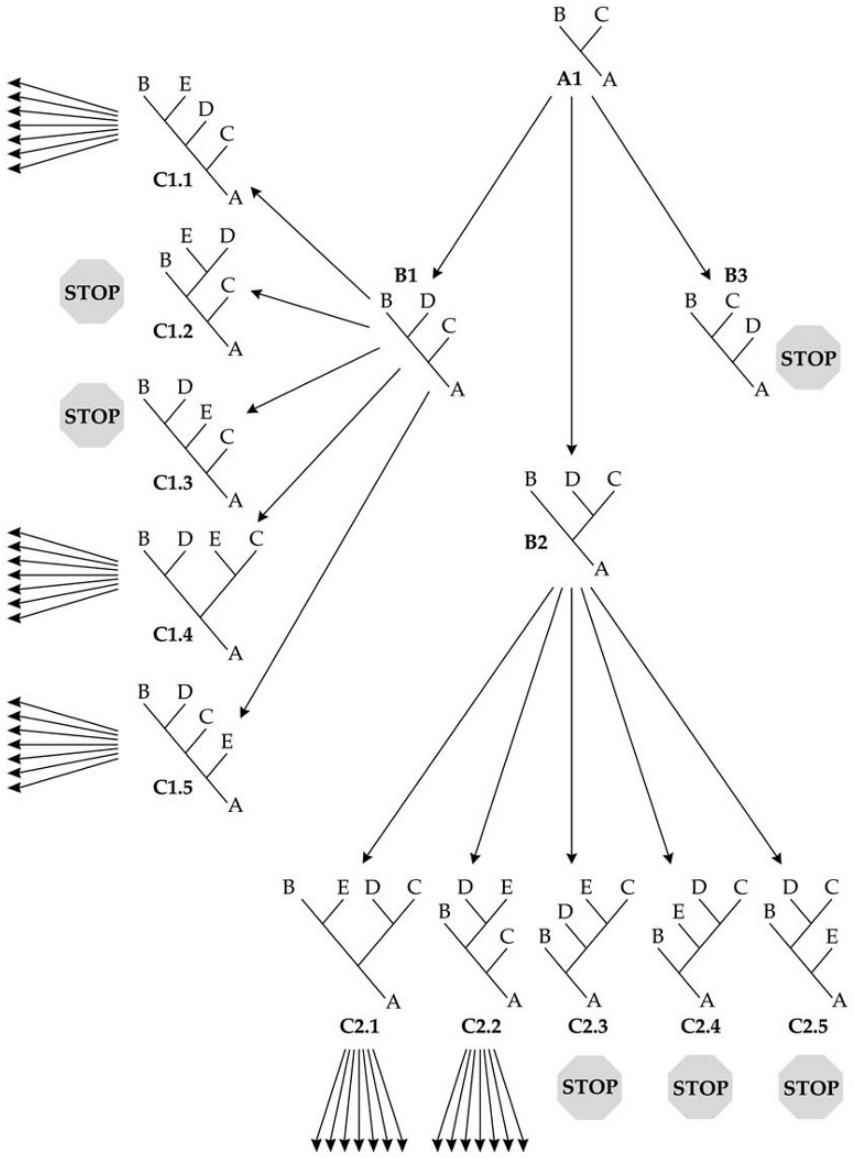
Exhaustive Search:
Evaluate the parsimony score of all trees.

Branch-and-Bound:
Like exhaustive search but better

MP is NP-hard. Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

Branch-and-Bound:
Like exhaustive search but better



MP is NP-hard. Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

Branch-and-Bound:
Like exhaustive search but better

Heuristic (e.g. hill climbing):
Compute a starting tree. Apply some operation to **edit** the tree. Search from new tree if it's score is higher.

MP is NP-hard. Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

Branch-and-Bound:
Like exhaustive search but better

Heuristic (e.g. hill climbing):
Compute a starting tree. Apply some operation to **edit** the tree. Search from new tree if it's score is higher.

But then we need methods to…

1. **Build** a starting tree.

   >randomized taxon addition

2. **Edit** an existing tree.

   >SPR, NNI, TBR moves

MP is NP-hard. Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

Branch-and-Bound:
Like exhaustive search but better

Heuristic (e.g. hill climbing):
Compute a starting tree. Apply some operation to **edit** the tree. Search from new tree if it's score is higher.

But then we need methods to…

1. **Build** a starting tree.

   >randomized taxon addition

2. **Edit** an existing tree.

   >SPR, NNI, TBR moves

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]

RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]

RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]

RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

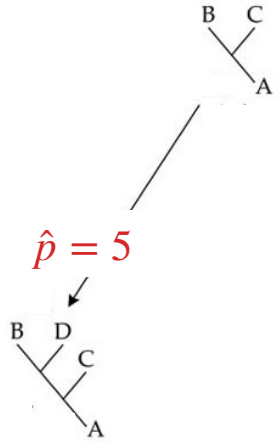
L = [B, C, A, D, E]

RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
    a. $p \leftarrow \infty$
    b. For each $e \in E(T)$:
        i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
        ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
        iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
    c. $T \leftarrow T_{save}$

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```



L = [B, C, A, D, E]

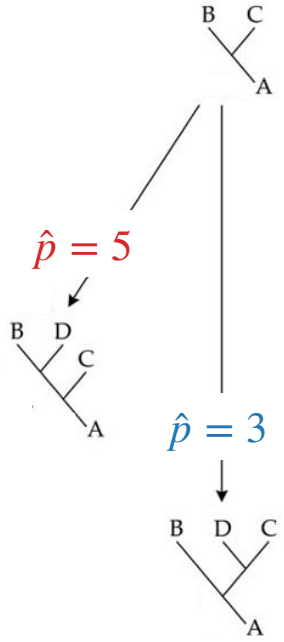
RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
    a. $p \leftarrow \infty$
    b. For each $e \in E(T)$:
        i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
        ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
        iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
    c. $T \leftarrow T_{save}$

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]



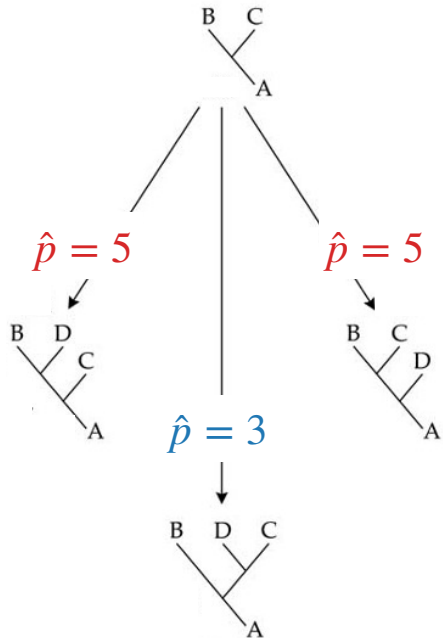
RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
   a. $p \leftarrow \infty$
   b. For each $e \in E(T)$:
      i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
      ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
      iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
   c. $T \leftarrow T_{save}$

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]



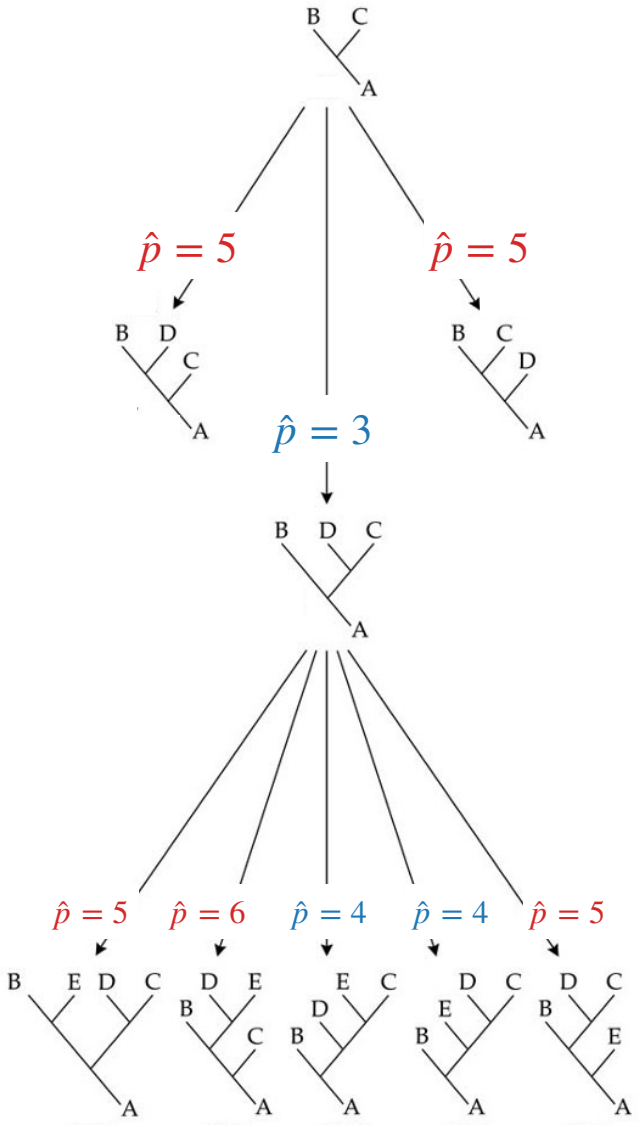
RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
   a. $p \leftarrow \infty$
   b. For each $e \in E(T)$:
      i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
      ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
      iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
   c. $T \leftarrow T_{save}$

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]



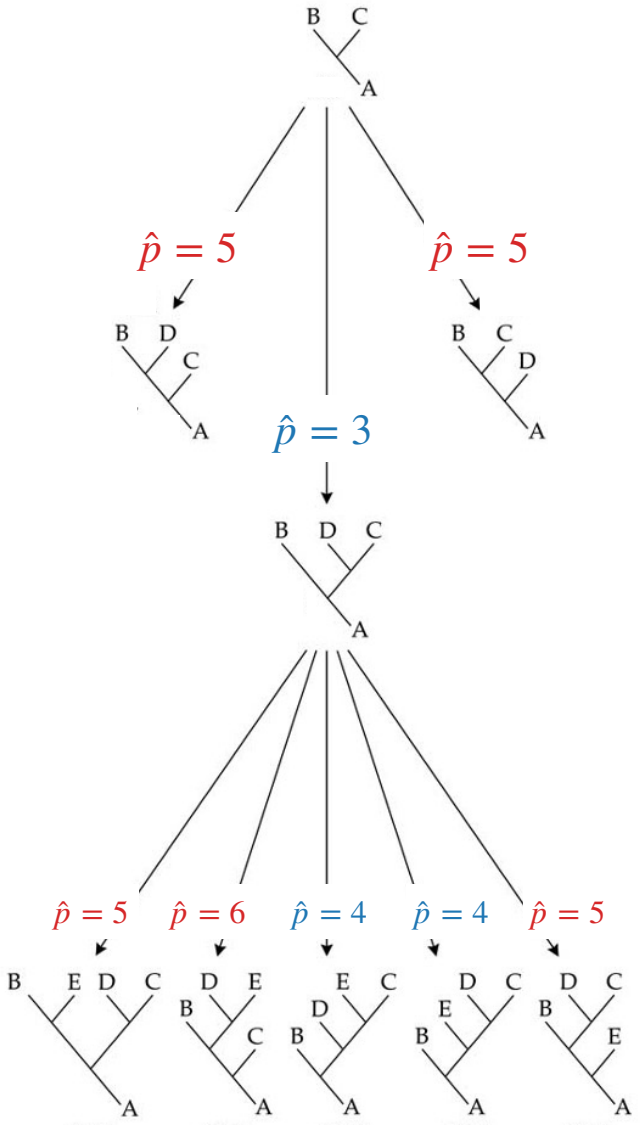
RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
   a. $p \leftarrow \infty$
   b. For each $e \in E(T)$:
      i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
      ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
      iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
   c. $T \leftarrow T_{save}$

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]



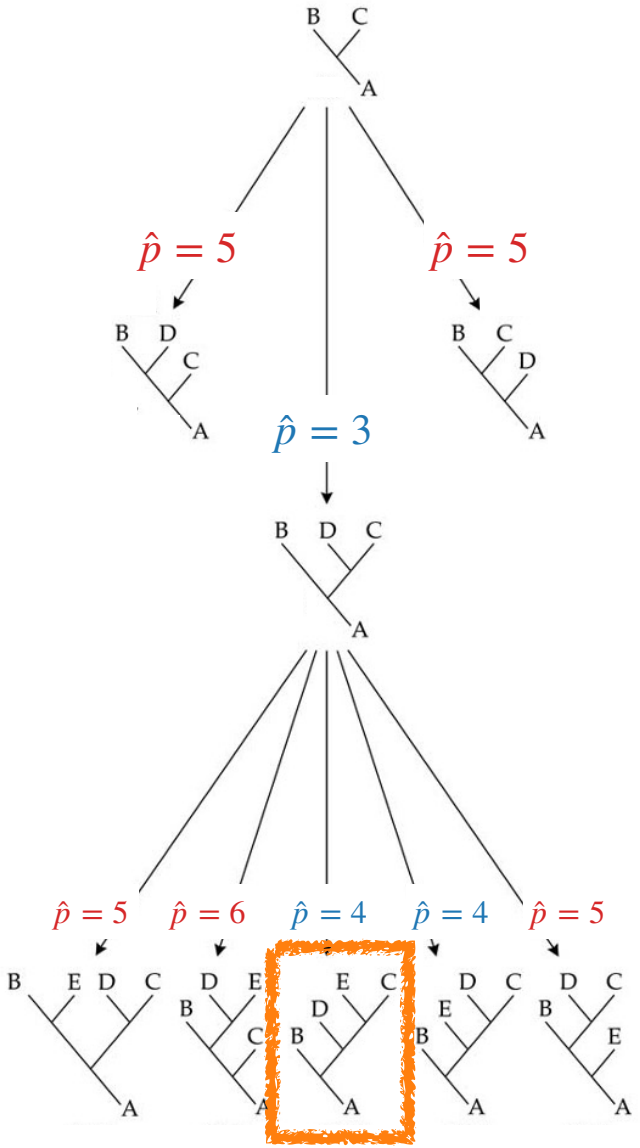
RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
   a. $p \leftarrow \infty$
   b. For each $e \in E(T)$:
      i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
      ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
      iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
   c. $T \leftarrow T_{save}$
4. Return T

Input characters:

```
A = (1,1,1)
B = (1,1,1)
C = (0,0,0)
D = (0,0,1)
E = (0,1,0)
```

L = [B, C, A, D, E]



RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
   a. $p \leftarrow \infty$
   b. For each $e \in E(T)$:
      i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
      ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
      iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
   c. $T \leftarrow T_{save}$
4. Return T

Think about — how many calls will you make to the length function?

Give your answer in Big Oh, where

- n is the # of labels
- m is the #r of characters

RandomizedTaxonAddition($\mathscr{C}, S$):

1. $L \leftarrow$ put labels in S in random order
2. $T \leftarrow$ star tree on first 3 elements of L
3. For each remaining element s in L:
   a. $p \leftarrow \infty$
   b. For each $e \in E(T)$:
      i. $\hat{T} \leftarrow$ Add s to T by subdividing e with new vertex v and creating edge (v, s)
      ii. $\hat{p} \leftarrow length(\hat{T}, \mathscr{C})$
      iii. If $\hat{p} < p$: $T_{save} \leftarrow \hat{T}$ and $p \leftarrow \hat{p}$
   c. $T \leftarrow T_{save}$
4. Return T

MP is NP-hard. Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

Branch-and-Bound:
Like exhaustive search but better

Heuristic (e.g. hill climbing):
Compute a starting tree. Apply some operation to **edit** the tree. Search from new tree if it's score is higher.

But then we need methods to…

1. **Build** a starting tree.

   >randomized taxon addition

2. **Edit** an existing tree.

   >SPR, NNI, TBR moves

MP is NP-hard. Now what?

Exhaustive Search:
Evaluate the parsimony score of all trees.

Branch-and-Bound:
Like exhaustive search but better

Heuristic (e.g. hill climbing):
Compute a starting tree. Apply some operation to **edit** the tree. Search from new tree if it's score is higher.

But then we need methods to…

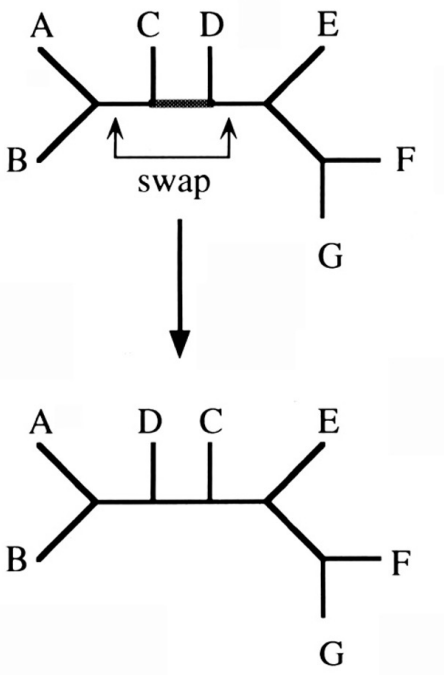
1. **Build** a starting tree.

   >randomized taxon addition

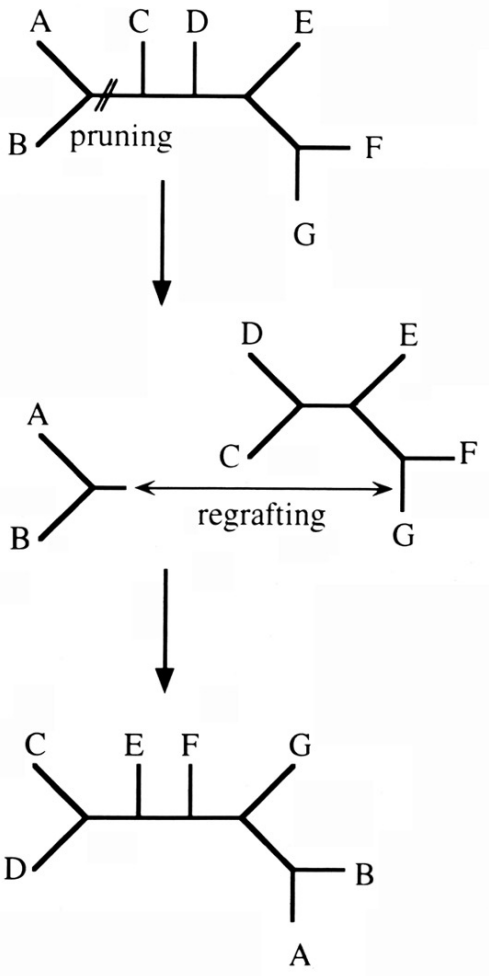
2. **Edit** an existing tree.

   >SPR, NNI, TBR moves

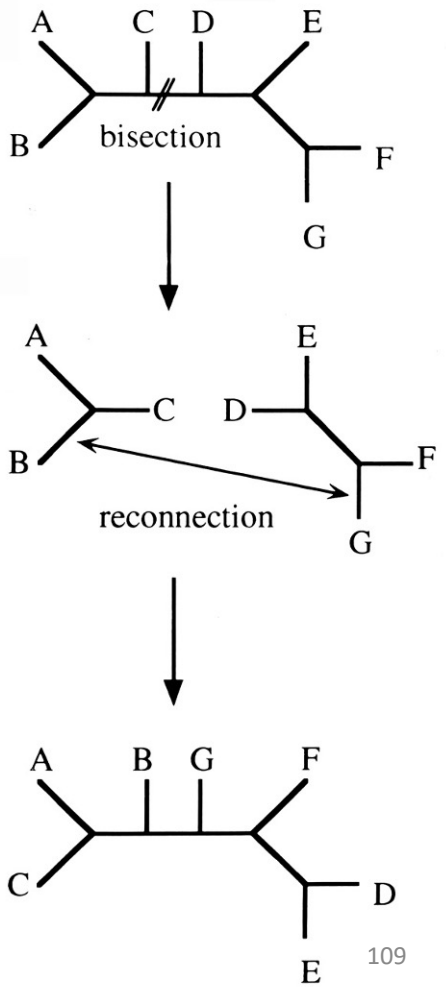
Nearest Neighbor Interchange (NNI) move



Subtree Prune and Regraft (SPR) move



Tree Bisection and Reconnection (TBR) move



Lastly, let's take a closer look at **branch-and-bound**.

Branch-and-Bound Ideas:

1. Take a tree T (e.g. compute a tree with randomized taxon addition) and compute its length L.

Branch-and-Bound Ideas:

1. Take a tree T (e.g. compute a tree with randomized taxon addition) and compute its length L.

2. An optimal solution to maximum parsimony tree must have length $\leq L$.

Branch-and-Bound Ideas:

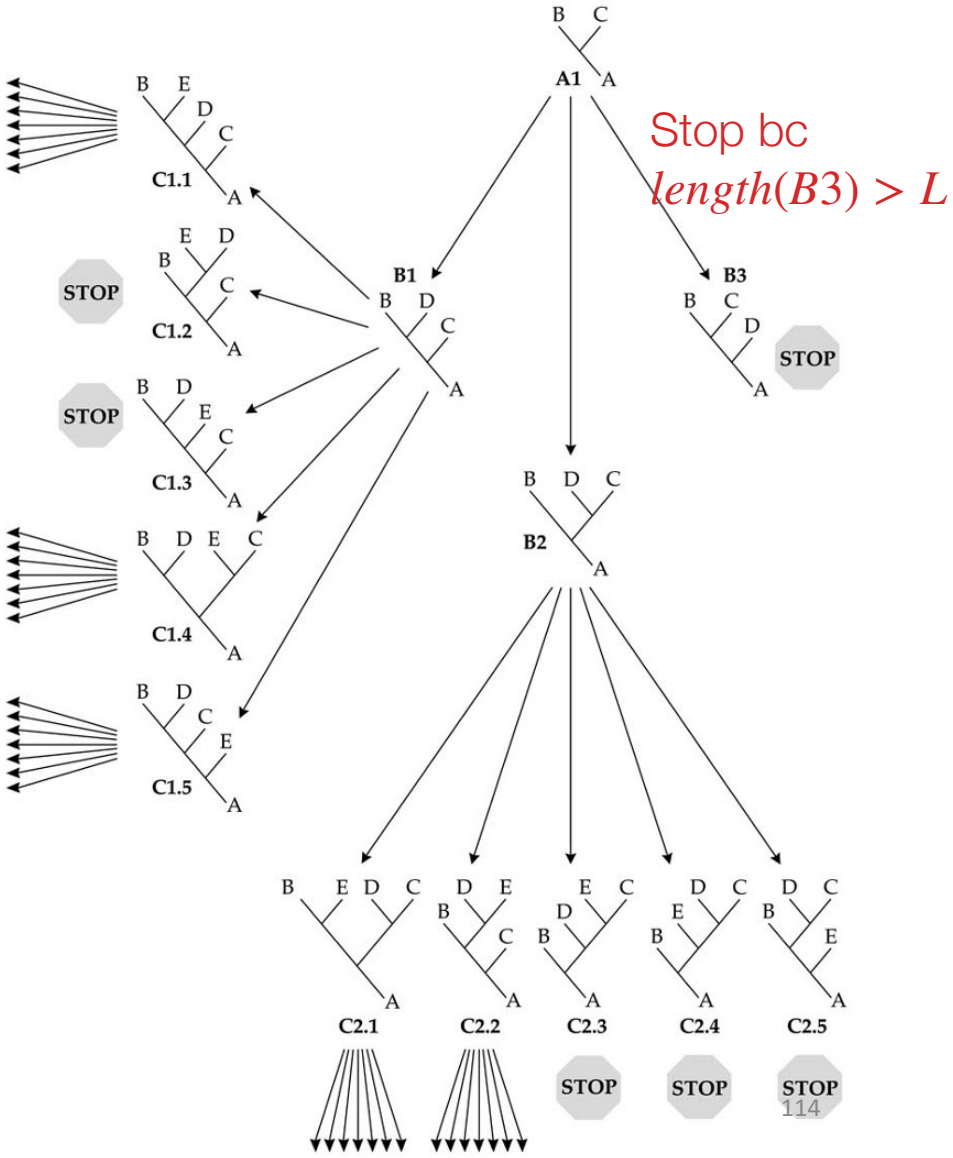
1. Take a tree T (e.g. compute a tree with randomized taxon addition) and compute its length L.

2. An optimal solution to maximum parsimony tree must have length $\leq L$.

3. Now suppose you add a taxon x to a tree t, The length of the resulting tree t_x must be $\geq length(t)$.

Branch-and-Bound Ideas:

1. Take a tree T (e.g. compute a tree with randomized taxon addition) and compute its length L.
2. An optimal solution to maximum parsimony tree must have length $\leq L$.
3. Now suppose you add a taxon x to a tree t, The length of the resulting tree t_x must be $\geq length(t)$.
4. Therefore, you can enumerate all trees via taxon addition and stop enumerating from a given tree t if $length(t) > L$.



Agenda

- **Part 1:** Perfect Phylogenies
- **Part 2:** Small Parsimony Problem & Fitch's Algorithm
- **Part 3:** Large Parsimony Problem
- **Part 4:** Maximum Parsimony Methods

Take-Aways

- Evolutionary trees are reconstructed from data (e.g., characters).
- A perfect phylogeny does not always exist for the data.
- The goal of parsimony is to find the tree that offers the simplest explanation of our data (i.e., minimum substitutions).
- The parsimony score of a given tree can be computed in polynomial time but finding a tree so that the score is minimized is NP-hard.
- Whether it makes sense to reconstruct a tree using maximum parsimony depends on the **model of evolution** — take my undergrad (498Y) or grad class (829A) to learn more!
- You covered a basic model of evolution (substitutions only) in the textbook and considered how to compute likelihood under this model (similar ideas apply for scoring, hardness, and heuristics).