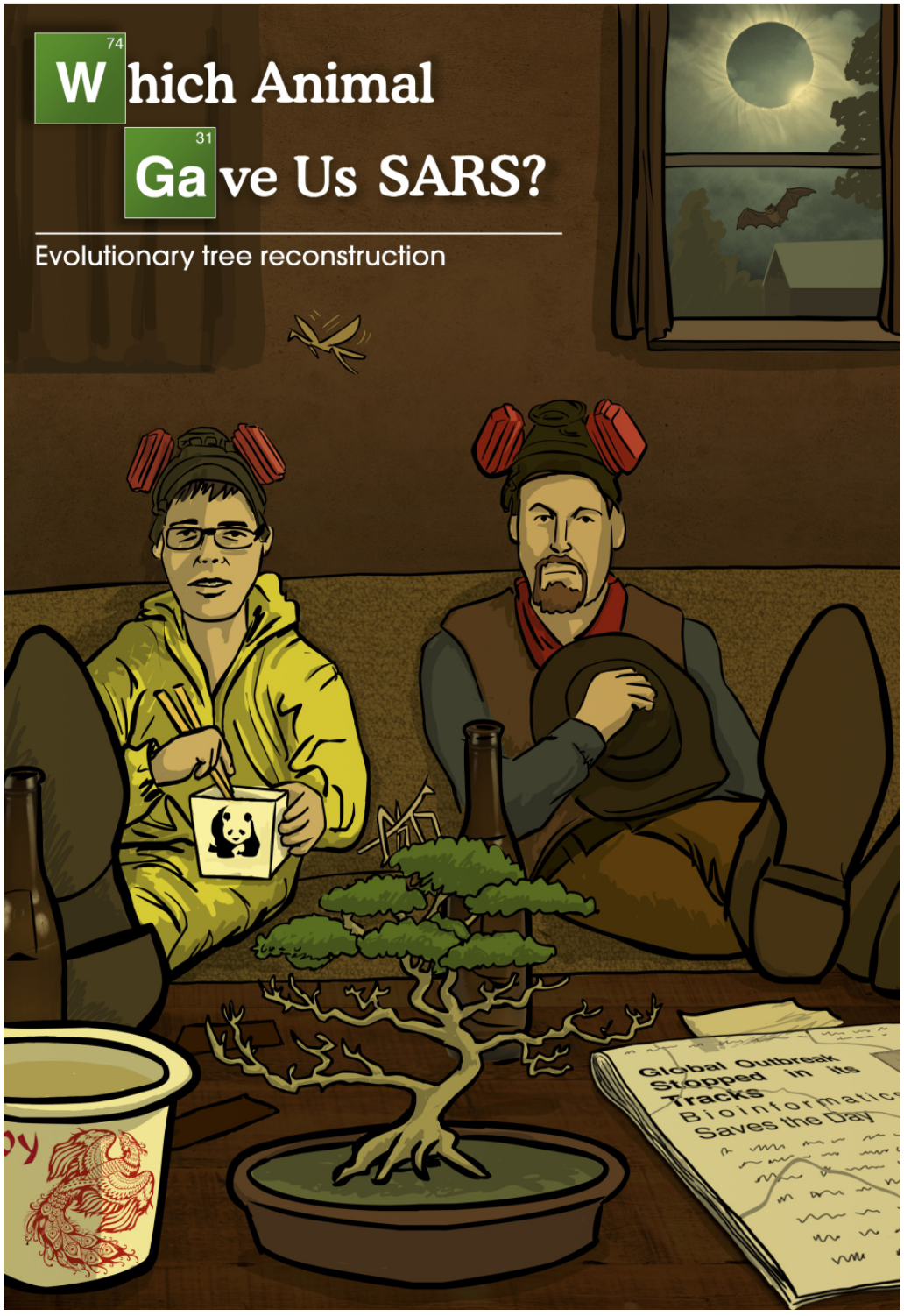
7 Which nimal Gave Us S RS?

Evolutionary Tree Reconstruction



7 Compute Distances Between Leaves

Distance Between Leaves Problem
Compute the distances between leaves in a weighted tree.

Input: An integer n followed by the adjacency list of a weighted tree with n leaves.
Output: An n x n matrix $(d_{i,j})$, where $d_{i,j}$ is the length of the path between leaves i and j.

Formatting

Input: An integer n, followed by an adjacency list representing a weighted tree.
Output: A tab-separated integer matrix of distances between leaves.

Constraints

- n will be between 1 and 10^2.

Test Cases

Case 1

Description: This sample dataset tests your base case. This tree is very small, consisting of only 2 leaves. Your code should work well with this tree.

Input:

```
2
0->1:2
```

Output:

```
0 2
2 0
```

Case 2

Description: This adjacency list represents a tree of which all the edge weights are the same.

Input:

```
4
0->4:1
1->4:1
2->5:1
3->5:1
4->5:1
```

Output:

```
0 2 3 3
2 0 3 3
3 3 0 2
3 3 2 0
```

Case 3

Description: This adjacency list represents an unbalanced tree, where there is at least 1 internal node between 1 leaf and another internal node. This dataset tests if your code is traversing correctly from 1 leaf to another through more than 1 internal node.

Input:

```
4
0->4:1
1->4:2
2->5:2
3->6:6
4->5:4
5->6:5
```

Output:

```
0 3 7 16
3 0 8 17
7 8 0 13
16 17 13 0
```

Case 4

Description: This dataset represents a special "star" tree, which has 1 node with degrees larger than 3.

Input:

```
5
0->5:1
1->5:7
2->5:4
3->5:9
4->5:6
```

Output:

```
0 8 5 10 7
8 0 11 16 13
5 11 0 13 10
10 16 13 0 15
7 13 10 15 0
```

Case 5

Description: This is a tree with more than 1 node whose degree is larger than 3. (Which nodes have degree 3?)

Input:

```
6
0->6:11
1->6:7
2->6:4
3->7:15
4->7:8
5->7:6
6->7:24
```

Output:

```
0 18 15 50 43 41
18 0 11 46 39 37
15 11 0 43 36 34
50 46 43 0 23 21
43 39 36 23 0 14
41 37 34 21 14 0
```

Case 6

Description: Another "star" tree but with 4 leaves instead of 5 leaves.

Input:

```
4
0->4:1
1->4:2
2->4:5
3->4:6
```

Output:

```
0 3 6 7
3 0 7 8
6 7 0 11
7 8 11 0
```

Case 7

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.

7B Compute Limb Lengths in a Tree

Limb Length Problem
Find the limb length for a leaf in a tree.

Input: An integer i, followed by an additive distance matrix D.
Output: The limb length of the leaf in $Tree(D)$ corresponding to row i of D.

Formatting

Input: An integer i, followed by a tab-separated additive distance matrix of integers D.
Output: An integer representing the limb length of leaf i.

Constraints

- i will be between 1 and 10^2.
- D will be an n x n matrix where n is between 1 and 10^2.

Test Cases

Case 1

Description: This sample dataset tests your base case. This tree is very small, consisting of only 2 leaves. Your code should work well with this tree!

Input:

```
1
0 2
2 0
```

Output:

```
2
```

Case 2

Description: This matrix would give you a tree whose all edges have the same length/weight. All limbs then also have the same length. If you don't believe that, you might want to try with other limbs by changing the first parameter to any from 0 to 3.

Input:

```
2
0 2 3 3
2 0 3 3
3 3 0 2
3 3 2 0
```

Output:

```
1
```

Case 3

Description: Another dataset to test your code.

Input:

```
3
0 3 7 16
3 0 8 17
7 8 0 13
16 17 13 0
```

Output:

```
11
```

Case 4

Description: A "star" tree which has 1 node with degrees larger than 3 might fit this distance matrix.

Input:

```
2
0 8 5 10 7
8 0 11 16 13
5 11 0 13 10
10 16 13 0 15
7 13 10 15 0
```

Output:

```
4
```

Case 5

Description: This distance matrix could be fit with a tree with more than 1 node whose degree is larger than 3.

Input:

```
3
0 18 15 50 43 41
18 0 11 46 39 37
15 11 0 43 36 34
50 46 43 0 23 21
43 39 36 23 0 14
41 37 34 21 14 0
```

Output:

```
15
```

Case 6

Description: This tree looks like a star with all leaves connected to 1 internal node.

Input:

```
3
0 3 6 7
3 0 7 8
6 7 0 11
7 8 11 0
```

Output:

```
6
```

Case 7

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.

7C Implement dditivePhylogeny

dditive Phylogeny Problem
Construct the simple tree fitting in an additive distance matrix.

Input: An additive distance matrix D.
Output: A weighted adjacency list for the simple tree fitting D.

□c7/logos/7C.png

Formatting

Input: A tab-separated additive distance matrix of integers D.
Output: A new-line separated weighted adjacency list with integer node labels and edge weights.

Constraints

- D will be an n x n matrix where n is between 1 and 10^2.

Test Cases

Case 1

Description: This sample dataset tests your base case. This tree is very small, consisting of only 2 leaves. Your code should work well with this tree.

Input:

```
0 2
2 0
```

Output:

```
0->1:2
```

Case 2

Description: This distance matrix might be fitted with an unbalanced tree, where there is at least 1 internal node between 1 leaf and another internal node.

Input:

```
0 13 21 22
13 0 12 13
21 12 0 13
22 13 13 0
```

Output:

```
0->4:11
1->4:2
2->5:6
3->6:7
4->5:4
```

Case 3

Description: This matrix would give you a tree whose all edges have the same length/weight. All limbs then also have the same length. If you don't believe that, you might want to try with other limbs by changing the first parameter to any from 0 to 3.

Input:

```
0 2 3 3
2 0 3 3
3 3 0 2
3 3 2 0
```

Output:

```
0->4:1
1->4:1
2->5:1
3->5:1
4->5:1
```

Case 4

Description: This distance matrix might be fitted with an unbalanced tree, where there is at least 1 internal node between 1 leaf and another internal node.

Input:

```
0 3 7 16
3 0 8 17
7 8 0 13
16 17 13 0
```

Output:

```
0->4:1
1->4:2
2->5:2
3->6:6
4->5:4
5->6:5
```

Case 5

Description: A "star" tree which has 1 node with degrees larger than 3 might fit this distance matrix.

Input:

```
0 8 5 10 7
8 0 11 16 13
5 11 0 13 10
10 16 13 0 15
7 13 10 15 0
```

Output:

```
0->5:1
1->5:7
2->5:4
3->5:9
4->5:6
```

Case 6

Description: This distance matrix could be fit with a tree with more than 1 node whose degree is larger than 3.

Input:

```
0 18 15 50 43 41
18 0 11 46 39 37
15 11 0 43 36 34
50 46 43 0 23 21
43 39 36 23 0 14
41 37 34 21 14 0
```

Output:

```
0->6:11
1->6:7
2->6:4
3->7:15
4->7:8
5->7:6
6->7:24
```

Case 7

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.

7D Implement UPGM

UPGM Problem
Construct the ultrametric tree resulting from UPGM .

Input: An $n \times n$ distance matrix followed by leaf node labels.
Output: A weighted adjacency list for the ultrametric tree output by UPGMA.

c7/logos/7D.png

Formatting

Input: A tab-separated $n \times n$ integer distance matrix.
Output: A newline-separated weighted adjacency list with integer node labels.

Constraints

- n will be between 1 and 10^3.

Test Cases

Case 1

Description: A general case where there are 4 input species with a 4×4 distance matrix.

Input:

```
0 3 4 3
3 0 4 5
4 4 0 2
3 5 2 0
```

Output:

```
4->2:1
4->3:1
5->0:1.5
5->1:1.5
6->4:0.5
6->5:1
```

Case 2

Description: Another general case where there are 4 input species with a 4×4 distance matrix.

Input:

```
0 20 17 11
20 0 20 13
17 20 0 10
11 13 10 0
```

Output:

```
4->2:5
4->3:5
5->0:7
5->4:2
6->1:8.8333
6->5:1.8333
```

Case 3

Description: This dataset will check your base case when there are only 2 leaves in the tree. Your code should produce a rooted tree with 2 leaves.

Input:

```
0 2
2 0
```

Output:

```
2->0:1
2->1:1
```

Case 4

Description: This dataset has some skewed pattern in the length between leaves, producing an unbalanced and unrooted tree.

Input:

```
0 3 48 12
3 0 42 3
48 42 0 15
12 3 15 0
```

Output:

```
4->0:1.5
4->1:1.5
5->3:3.75
5->4:2.25
6->2:17.5
6->5:13.75
```

Case 5

Description: This dataset will challenge the tree that fits perfectly to the matrix.

Input:

```
0 13 21 22
13 0 12 13
21 12 0 13
22 13 13 0
```

Output:

```
4->1:6
4->2:6
5->3:6.5
5->4:0.5
6->0:9.3333
6->5:2.8333
```

Case 6

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.

7E Implement the Neighbor-Joining lgorithm

Neighbor Joining Problem
Construct the tree resulting from applying the neighbor-joining algorithm to a distance matrix.

Input: An $n \times n$ distance matrix.
Output: A weighted adjacency list for the tree resulting from applying the neighbor-joining algorithm.

Formatting

Input: A tab-separated $n \times n$ integer distance matrix.
Output: A weighted adjacency list with integer node labels and edge weights.

Constraints

- n will be between 1 and 10^2.

Test Cases

Case 1

Description: This is a dataset to quick check for your code to make sure you consider the base case. This tree is very small and straight forward, you would be able to expect how the tree looks like with the given algorithm without much thinking. Your code should produce an unrooted tree with only 2 leaves connected.

Input:

```
0 5
5 0
```

Output:

```
0->1:5.000
1->0:5.000
```

Case 2

Description: This dataset will produce a tree where all edges are equal. This unrooted tree only has 3 leaves. Have you observed anything different here?

Input:

```
0 2 2
2 0 2
2 2 0
```

Output:

```
0->3:1.000
1->3:1.000
2->3:1.000
```

Case 3

Description: This distance matrix is additive. How will this affect your tree?

Input:

```
0 3 4 3
3 0 4 5
4 4 0 2
3 5 2 0
```

Output:

```
0->4:1.000
1->4:2.000
2->5:1.000
3->5:1.000
4->5:1.500
```

Case 4

Description: A general case to test your code.

Input:

```
0 13 21 22
13 0 12 13
21 12 0 13
22 13 13 0
```

Output:

```
0->4:11.000
1->4:2.000
2->5:6.000
3->5:7.000
4->5:4.000
```

Case 5

Description: This is a tricky dataset. Look at your answer and tell me what you notice here?

Input:

```
0 1 5 8
1 0 3 4
5 3 0 2
8 4 2 0
```

Output:

```
0->4:2.000
1->4:-1.000
2->5:0.000
3->5:2.000
4->5:3.500
```

Case 6

Description: This dataset gives you another perspective on a 3-degree node. We have learned about the degree of a node. For example, a leaf is a node with degree 1 while an internal node is with degree 3. However, what if the edge weight between the 2 3-degree nodes is 0? This means, technically, we could represent a tree having nodes with degree higher than 3.

Input:

```
0 2 4 6
2 0 4 6
4 4 0 8
6 6 8 0
```

Output:

```
0->4:1.000
1->4:1.000
2->5:3.000
3->5:5.000
4->5:0.000
```

Case 7

Description: This dataset tests if your code can handle a larger unbalanced tree.

Input:

```
0 2 10 20 24 30 30 14 13
2 0 10 20 24 30 30 14 13
10 10 0 18 22 28 28 12 17
20 20 18 0 6 12 12 8 27
24 24 22 6 0 10 10 12 31
30 30 28 12 10 0 6 18 37
30 30 28 12 10 6 0 18 37
14 14 12 8 12 18 18 0 21
13 13 17 27 31 37 37 21 0
```

Output:

```
0->14:1.000
1->14:1.000
2->13:4.000
3->11:1.000
4->10:2.000
5->9:3.000
6->9:3.000
7->12:1.000
8->15:10.000
9->10:5.000
10->11:3.000
11->12:6.000
12->13:7.000
13->15:3.000
14->15:2.000
```

Case 8

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.

7F Implement SmallParsimony

Small Parsimony Problem
Find the most parsimonious labeling of the internal nodes of a rooted tree.

Input: A list of n DNA strings, followed by an adjacency list for a rooted binary tree with n leaves.
Output: The minimum parsimony score of this tree, followed by the labeling of the nodes by DNA strings minimizing the parsimony score of the tree.

Formatting

Input: Newline-separated DNA strings, followed by a newline-separated unweighted adjacency list for a rooted binary tree with n leaves labeled 0 through n 1.
Output: An integer representing the minimum parsimony score of the tree, followed by a newline-separated list of DNA strings representing the labeling of all nodes in the tree minimizing the parsimony score of the tree.

Constraints

- n will be between 1 and 10^2.
- The length of a DNA string will be between 1 and 10^2.

Test Cases

Case 1

Description: This small and easy dataset is a quick check for your logic at the base level so that your debugging is at least faster because you can do this one by hand.

Input:

```
 C
 C
--
2->0
2->1
```

Output:

```
0
 C
 C
 C
```

Case 2

Description: This sample dataset is not actually run on your code.

Input:

```
G G TCCC
 TTG G T
CTGCGC T
 TGG CG
--
4->0
4->1
5->2
5->3
6->4
6->5
```

Output:

```
13
G G TCCC
 TTG G T
CTGCGC T
 TGG CG
 TGG C T
 TGG C T
 TGG C T
```

Case 3

Description: This is an easy check if your code is working at the most basic case when all edges are 0.

Input:

```
G G
G G
G G
G G
--
4->0
4->1
5->2
5->3
6->4
6->5
```

Output:

```
0
G G
G G
G G
G G
G G
G G
G G
```

Case 4

Description: A balance tree which you can calculate parsimony score by hand, where some of the weights are 0.

Input:

```
TTT

CCC
--
4->0
4->1
5->2
5->3
6->4
6->5
```

Output:

```
6

TTT

CCC
```

Case 5

Description: This dataset makes sure that your code works with more than just 4-leaf unbalanced trees.

Input:

```
GTC T
G C T
C G T
 TC G
 TTCT
 TCCG
--
6->0
6->1
7->2
7->3
8->4
8->5
9->6
9->7
10->8
10->9
```

Output:

```
9
GTC T
G C T
C G T
 TC G
 TTCT
 TCCG
GTC T
 TC T
 TCCT
 TC T
 TC T
```

Case 6

Description: This dataset tests your dynamic programming code logic. Your code would need to, for each position in the sequence, consider the whole tree to get the global optimal solution.

Input:

```
 GTT
GCTC
TG
CC T
GG T
--
5->0
5->1
6->2
6->5
7->3
7->6
8->4
8->7
```

Output:

```
8
 GTT
GCTC
TG
CC T
GG T
GGTT
GG T
GG T
GG T
```

Case 7

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.

7G dapt SmallParsimony to Unrooted Trees

Small Parsimony in an Unrooted Tree Problem
Find the most parsimonious labeling of the internal nodes in an unrooted tree.

Input: A list of n DNA strings, followed by an adjacency list for an unrooted binary tree with n leaves.
Output: An integer representing the minimum parsimony score of the tree, followed by a newline-separated list of DNA strings representing the labeling of all nodes in the tree minimizing the parsimony score of the tree.

Formatting

Input: Newline-separated DNA strings, followed by a newline-separated unweighted adjacency list for an unrooted binary tree with n leaves labeled 0 through n 1.
Output: An integer representing the minimum parsimony score of the tree, followed by a newline-separated list of DNA strings representing the labeling of all nodes in the tree minimizing the parsimony score of the tree.

Constraints

- n will be between 1 and 10^2.
- The length of a DNA string will be between 1 and 10^2.

Test Cases

Case 1

Description: This small and easy dataset is a quick check for your logic at the base level so that your debug at least faster because you can do this one by hand.

Input:

```
 C
TC
GC
 G
--
0->4
1->4
2->5
3->5
4->5
```

Output:

```
3
 C
TC
GC
 G
 C
 C
```

Case 2

Description: This sample dataset is not actually run on your code.

Input:

```
C   TCTC
 TTGCG C
TTGCGCT
 TGGCCG
--
0->4
1->4
2->5
3->5
4->5
```

Output:

```
16
C   TCTC
 TTGCG C
TTGCGCT
 TGGCCG
 T GCCTC
 TGGCCT
```

Case 3

Description: This is an easy check if your code is working at the most basic case when all edges are 0.

Input:

```
G G
G G
G G
G G
--
4->0
4->1
5->2
5->3
6->4
6->5
```

Output:

```
0
G G
G G
G G
G G
G G
G G
G G
```

Case 4

Description: Another easy balanced tree whose parsimony score can be calculated by hand.

Input:

```
TTT

CCC
--
0->4
1->4
2->5
3->5
4->5
```

Output:

```
6

TTT

CCC
```

Case 5

Description: This dataset makes sure that your code can handle more than 4-leaf trees and with an unbalanced tree.

Input:

```
GTC C
G C T
C G T
 TGGT
 TTC
 TCC
--
0->6
1->6
2->7
3->7
4->8
5->8
6->9
7->9
8->9
```

Output:

```
10
GTC C
G C T
C G T
 TGGT
 TTC
 TCC
GTC T
 TG T
 TCC
 TC T
```

Case 6

Description: This dataset tests your dynamic programming code logic. Your code would need to, for each position in the sequence, consider the whole tree to get the global optimal solution.

Input:

```
 GTT
GCTC
TG
CC T
GG T
--
0->5
1->5
2->6
3->7
4->7
5->6
6->7
```

Output:

```
8
 GTT
GCTC
TG
CC T
GG T
GGTT
GG T
GG T
```

Case 7

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.