PLTW COMPUTER SCIENCE

Project 3.2.6

Supplement E: The Alignment Problem

Introduction

Computation has turned many disciplines on their heads, from engineering design to architecture, from visual arts to music. Computation has revolutionized many areas of biology, and in this lesson, we aim to understand the impact of computation on genetics. One of the most important problems that had to be solved in computer science to assist the advancement of biology was the problem of alignment. GitHub solved a similar problem; given two files, one of which is a copy of the other one but with some insertion, deletions, and replacements, how can you identify which parts of the two files are the same? The BLAST algorithm, invented in 1994, solved the alignment problem for the special case of DNA, which has only the four letters G, C, A, and T. In this supplementary activity, you learn what the alignment problem is.

Resources

3.2.6.e.P source File

Procedure

1. Open the program `mutation.pyin` Canopy and execute it. From the iPython session, run `simulate()` and observe the output.

2. This function simulates the reproduction of a sentence, generation after generation. With each reproduction, some changes might occur, much like a game of “telephone.” These changes, called mutations, accumulate with each generation.

   The program defines a class called `Individual`. Each `Individual` is represented by a string that generates “children” with a 90% chance of a single mutation occurring in each child. A mutation can be a single character insertion, a single character deletion, or a single character point mutation.

   The program also defines a function

   ```
   simulate(numGenerations=4, luca='knowledge of computation is power!')
   ```

   which starts with a string luca (an acronym for the “last universal common ancestor”) and creates several generations of children. In each generation a string might create 0, 1, 2, 3 or 4 children.

   The function produces two sections of output:

   - A numbered list of the fourth and final generation.

- A family tree descending from the initial string, with indentation to display the parent-child relationship. This tree is called a phylogeny, which is a visualization of the relationships between different organisms or species.

Here is one example of the program's output using a variation of `simulate()`. The output has two parts: the current generation and the phylogeny. Only the first part is shown here; the second part is shown in the next step.

```
In []: simulate(numGenerations=4,
  ...: luca='knowledge of computation is power!')
--------------------------------

Current generation:

0 'knowledge of coputatUon is powXer!'
1 'kOnowledge ofScoputation is power!'
2 'knowledge ofScoputation iGs power!'
3 'knowledLe ofScoputation is power!'
4 'kWnowledge f coputation is power!'
5 'knowledge f copWtation is power!'
6 'knowledge f coputation is poer!'
7 'knowledgeF of computationCis powerE'
8 'knowlede of computation i powerU!'
9 'knowlede of computationGP is power!'
10 'knowledY of computation is pwer!'
11 'knowlede of computation is pwr!'
12 'knowledeHof computation is pwer!'
```

The first part of this output is a puzzle. It is a lot like the evidence we look at when we look at the DNA of existing animals. It shows only the current generation of "organisms." We can see that some groups of the strings are related, since they share a mutation. Some mutations must have happened earlier in the family tree than others, since they affect larger groups.

Notice, for instance, that sequences `0` through `6` are all missing the `'m'` in `'computation'`. That is a deletion. You might also notice that sequences `1` through `3` all have the space in `'of computer'` changed to an `'S'`. That is a point mutation. Can you find another mutation that two or more strings have?

3. The second part of the output is a pedigree, as shown below. (Generational references `G#-#` have been added to help in your analysis.) The output is the answer to the puzzle, showing which mutations happened first and how the strings in the current generation are related to each other. Underneath each string, its descendants are listed. An indentation shows successive generations.

```
--------------------------------

Phylogeny:

 'knowledge of computation is power!'           G1-1
    'knowledge of coputation is power!'         G2-1
       'knowledge of coputatUon is power!'      G3-1
```

```
          'knowledge of coputatUon is powXer!'  G4-1
       'knowledge ofScoputation is power!'      G3-2
          'kOnowledge ofScoputation is power!'  G4-2
          'knowledge ofScoputation iGs power!'  G4-3
          'knowledLe ofScoputation is power!'   G4-4
       'knowledge f coputation is power!'       G3-3
          'kWnowledge f coputation is power!'   G4-5
          'knowledge f copWtation is power!'    G4-6
          'knowlege f coputation is poer!'      G4-7
    'knowledge of computationCis power!'        G2-2
       'knowledgeF of computationCis power!'    G3-4
          'knowledgeF of computationCis powerE' G4-8
    'knowdlede of computation is power!'        G2-2
       'knowlede of computation i power!'       G3-5
          'knowlede of computation i powerU!'   G4-9
       'knowlede of computationG is power!'     G3-6
          'knowlede of computationGP is power!' G4-10
       'knowlede of computation is pwer!'       G3-7
          'knowledY of computation is pwer!'    G4-11
          'knowlede of computation is pwr!'     G4-12
          'knowledeHof computation is pwer!'    G4-13
```

A **tree** is a data structure where every **node** (except the **root node**) is the **child** of exactly one **parent**. The G# annotation represents generations 1 through 4. Based on the pedigree, strings G4-9, G4-10, and G4-11 are siblings. Another set of siblings is G3-5, G3-6, and G3-7. Can you identify another set of siblings?

4. Return to the first part of the output. Could you figure out the pedigree by only looking at the first part of the output? In the case of biological organisms, this is exactly the puzzle we face: we can examine present-day organisms but must infer how they are related in the past.

   To infer the most likely historical relationships based on the evidence, we look for commonalities in the sequences. For example, sentences 8 through 12 are all missing the `'g'` in `'knowledge'`. These patterns are easier to detect, both for our visual system and for a computer algorithm, if the sequences are first aligned. Alignment places each corresponding piece of information in its own column. To create an alignment, we insert gaps into the sequences so that insertions and deletions don't get the sequences out of alignment with each other. We fill these gaps with a character that is not used in the sequences so that we don't confuse the symbol with a change. We will use the equal sign.

   Looking at the first part of the output, you could produce the following alignment by inserting equal signs.

   ```
   'k=nowled=====f=co=p=tat=on==i===p====r=='
   ```

   The line above shows the characters that were retained by *all* members of the current generation. Each equal sign in the output below represents a place where a point mutation, insertion, or deletion occurred.

   ```
   0 'k=nowledge= of co=putatUon==i=s powXer=!'
   1 'kOnowledge= ofSco=putation==i=s pow=er=!'
   2 'k=nowledge= ofSco=putation==iGs pow=er=!'
   3 'k=nowledgLe= ofSco=putation==i=s pow=er=!'
   ```

```
4 'kWnowledge= =f co=putation==i=s pow=er=!'
5 'k=nowledge= =f co=pWtation==i=s pow=er=!'
6 'k=nowledge= =f co=putation==i=s po==er=!'
7 'k=nowledgeF of computation==i=sCpow=erE='
8 'k=nowled=e= of computation==i==pow=erU!'
9 'k=nowled=e= of computationGP i=s pow=er=!'
10 'k=nowled=Y= of computation==i=s p=w=er=!'
11 'k=nowled=e= of computation==i=s p=w==r=!'
12 'k=nowled=e=Hof computation==i=s p=w=er=!'
```

What does it mean if a letter appears in every member of the current generation?

5. Here is output from an additional run of simulate, using a different starting string `luca`. See if you can create the tree and pedigree with only access to the current generation. This is akin to the problem you will solve in Project 3.2.6, in which you only have access to existing organisms.

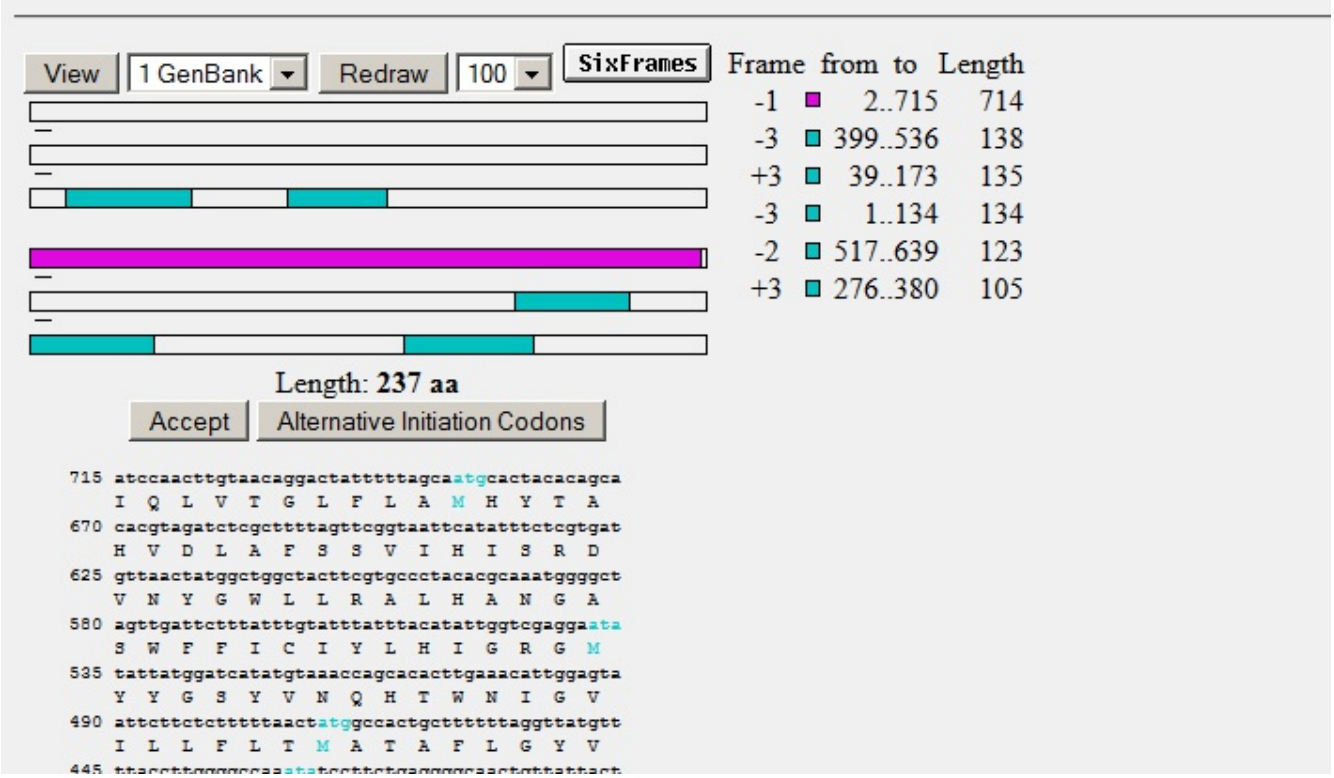
    - Create an alignment by inserting equal signs as necessary.
    - Create a tree showing which strings you think are siblings and first cousins.

```
0 'RHmptation'
1 'comptatCon'
2 'cLmpGation'
3 'QcompGation'
4 'compGatioAn'
5 'comptatSionT'
6 'compStatSion'
7 'comptatSiVon'
8 'comptatBSion'
9 'comptatSon'
10 'comptatSAion'
11 'cmputatioQ'
```

Extension

6. Translation always starts with the amino acid methionine. The following figure is the output of the open reading frame finder, which gives a graphic analysis of pieces of a DNA sequence being read in the six different possible reading frames (ORFF). There are six reading frames for a given piece of DNA: three forward and three reverse complement. In this figure, six open reading frames are displayed, shaded within four of the reading frames. The longest of the six is shaded pink.

    Write code to report the ‘Frame’ and ‘Length’ columns for any FASTA file, as shown in the figure.



Conclusion

1. In computer science a **tree** is a collection of **nodes** and **links** such that each node has exactly one **parent**. What do the nodes and links represent in Part II of this activity?
2. In computer science, a **binary tree** is a tree in which all nodes have exactly 0, 1, or 2 children. Binary trees are important because they lead to fast search algorithms. Are the trees in this activity binary? Explain why or why not.
3. Consider the class of objects, each called an Individual in the code ‘transcribeTranslate.py’. Each of these objects have a variable Individual.children that is a list of other Individual objects. How does this way of structuring the data connect to the underlying concept of a pedigree tree? How are the nodes represented in the program? How are the parent-child relationships of the tree represented in the program?