CSCA08 Assignment 2

Due: December 3, 2017. 11:55pm

Introduction

In this assignment, we will be working with dictionaries and methods, using OOP to solve problems. The goal of this assignment is to make sure that you are comfortable with all the technical elements covered in the course, as well as giving you a bit more opportunity for freedom of choice and design.

Important Notes

In this assignment, you may use any material covered in the course, or any built-in elements of python that do not require importing. Your code should not use `import` or `print` statements anywhere.

As with previous assignments, getting the code working will only constitute a relatively small part of your mark. A large portion will be dedicated to efficiency and good design. Once again, you should remember that genes can be very long, and there will be many genes stored in our objects, so think carefully about copying data or looping over data more times than necessary.

Also, in this assignment, a large portion of the marks will be dedicated to how well your code works within the software development principles we've discussed in lecture. Working code is important, but so is code that can easily be maintained, updated and modified in the future following the ideals of encapsulation and abstraction as presented in lecture.

DNA Sequencing

Thanks to our good work in previous assignments, we have been hired to build the first version of a software system for a genetic engineering firm[1]. This firm works with humans exclusively at the moment, but may want to work with other animal DNA at a later date. As you know by now, all animals have genetic data arranged in long strings of paired nucleotides call chromosomes. Humans have 23 pairs of chromosomes (other animals have more or less). 22 of the pairs are normal pairings, and a final pair (23) called the sex chromosomes which are different in males and females. Individual pairs of nucleotides can be found either by referring to their chromosome pair and location (e.g., chromosome pair 7, position 17), or by their marker (e.g., rs1799883). Interestingly, the vast majority of human DNA is identical among all people. So we normally only bother to record the pairs that differ between individuals.

The scientists also want to be able to query an individual's genetic sequence. A query is a way of asking whether certain nucleotide patterns exist. So they may wish to query something like: "Is marker rs4543123 equal to AG?", or "Are the nucleotides at chromosome pair 9 position 74 the reverse order of the nucleotides at position 95?". In order to do this, the geneticists create special query chromosome pairs that are blank everywhere except the pairs they wish to analyze. They can either use normal nucleotides (A, T, G, or C) or memory nucleotides (1, 2, 3, 4, etc). If a normal nucleotide in a query chromosome doesn't equal the nucleotide in the chromosome being compared against, then the whole query is rejected. The query is also rejected if every memory nucleotide with the same value isn't compared with nucleotides of the same value (e.g., every 3 can be compared against a 'C' or against a 'T', but we can't have one 3 compared against a 'C' and another compared against a 'T'). As an example; to solve our query "Is marker rs4543123 equal to AG?", we would create a query chromosome pair with rs4543123 equal to AG, and compare it with the chromosomes of the individual; to solve our query "Are the nucleotides at chromosome pair 9 position 74 the reverse order of the nucleotides at position 95?", we would create a query chromosome pair, and set position 74 to be 12, and position 95 to be 21. A memory nucleotide can appear as many times as necessary in a chromosome pair or across multiple chromosome pairs, but there are only 10 different memory nucleotide types, so all

[1] In the previous assignments we said that not everything was fully biologically credible... we're now taking a much larger leap into science fiction

memory nucleotides are represented by a number between 0 and 9. If a query nucleotide compares against an unknown nucleotide (i.e., the model simply doesn't have any information on it) it should not cause the query to be rejected, except in the case where it is being compared against the male sex chromosome pair (23), in which case comparing a query nucleotide against something unknown should reject the query.

The other major interest of the geneticists is creating custom offspring[2]. When two individuals mate to produce offspring, each chromosome pair of the offspring is filled with nucleotide pairs where one nucleotide comes from the mother and one comes from the father. These scientists have discovered a way to choose which nucleotide will come from each parent by building binding chromosomes that can set a given position or marker to be either left-maternal (the child gets the mothers left nucleotide and the father's right) or right-maternal (the child gets the fathers left nucleotide and the mothers right).

Your Tasks

For this assignment, you will be required to build classes for and methods for the system outlined above. There will be specific requirements given below, but you will likely also want other classes that are not directly mentioned. You should first construct your UML diagram to make sure that you've thought through your design before writing your code.

The clients don't really understand much about computer science. So they aren't going to specify how you should build your code. But they do have some requirements. They've had difficulty articulating their needs, so instead, we built them a very simple prototype, and recorded some sample operations that they would like to perform. The list of those operations is attached to this assignment. Notice that they haven't provided exact details about what should happen in some of the boundary cases, so you'll have to make some sensible decisions for yourself.

What to Submit

Submit `a2.py` and `a2.test` on MarkUs along with your UML diagram which can be named one of `a2.jpg`, `a2.png` or `a2.pdf`. Note that MarkUs cannot deal with multiple possible file types, so it WILL NOT warn you if your UML file is missing or named incorrectly. It willy only be checking for the .py and .test files. Your UML diagram can be hand drawn and photographed/scanned or created on a computer, but it must be clearly readable by the TAs, and have a maximum file size of 1MB.

[2]We told you this was science fiction

Annotated Sample User Operations

```
# create a new Male client with ID 12345
father = Male('12345')
# create a new Female client with ID 67890
mother = Female('67890')
# set chromosome pair 12 position 45 to be AG
father.set_by_pos(12, 45, 'AG')
mother.set_by_pos(12, 45, 'CT')
# set chromosome marker rs12345 to refer to chromosome
# pair 3, position 97
father.set_marker('rs12345', 3, 97)

# set marker rs12345 to be GT
father.set_by_marker('rs12345', 'GT')
# this should return "AG"
result_str = father.get_by_pos(12, 45)
# this should return "GT"
result_str2 = father.get_by_marker('rs12345')
c = father.get_chromosome(3)
# This will set father's pair 3-85 to be "TA"
c.set_by_pos(85, 'TA')
# Now mother and father share a chromosome pair, updating one will update the other
mother.set_chromosome(7, c)

# create a new query object
query = Query()
query.set_by_pos(12, 45, 'AG')
# this should return True since 12-45 in the query matches with 12-45 in father
result = father.test(query)
query.set_by_pos(12, 45, 'A1')
query.set_marker('rs12345', 3, 97)
# now the query will only work if 12-45 is AX and 3-97 is XT for some value of X
query.set_by_marker('rs12345', '1T')

# create a new binder object
binder = Binder()
# set chromosome pair position 45 to be left material
# (this means that the offspring will have 12-45 equal to CG
# e.g., gets the left C from mother and the right G from father)
binder.set_by_pos(12, 45, 'LM')

# this means any offspring created with this binder will be female
binder.set_sex("F")
child = mother.procreate(father, binder)
```