












Isso é CS50

Introdução do CS50 à Ciência da Computação

OpenCourseWare

Doar  (<https://cs50.harvard.edu/donate>)

David J. Malan (<https://cs.harvard.edu/malan/>)
malan@harvard.edu

 (<https://www.clubhouse.com/@davidjmalan>)  (<https://www.facebook.com/dmalan>) 
(<https://github.com/dmalan>)  (<https://www.instagram.com/davidjmalan/>) 
(<https://www.linkedin.com/in/malan/>)  (<https://orcid.org/0000-0001-5338-2522>) 
(<https://www.quora.com/profile/David-J-Malan>)  (<https://www.reddit.com/user/davidjmalan>) 
(<https://www.tiktok.com/@davidjmalan>)  (<https://davidjmalan.t.me/>) 
(<https://twitter.com/davidjmalan>)

DNA

Implement a program that identifies a person based on their DNA, per the below.

```
$ python dna.py databases/large.csv sequences/5.txt  
Lavender
```

Getting Started

Faça login em code.cs50.io (<https://code.cs50.io>), clique na janela do seu terminal e execute `cd` -o sozinho. Você deve descobrir que o prompt da janela do terminal é semelhante ao abaixo:

```
$
```

Próxima execução

```
wget https://cdn.cs50.net/2022/fall/psets/6/dna.zip
```

para baixar um ZIP chamado `dna.zip` em seu codespace.

Então execute

```
unzip dna.zip
```

para criar uma pasta chamada `dna`. Você não precisa mais do arquivo ZIP, então você pode executar

```
rm dna.zip
```

e responda com “y” seguido de Enter no prompt para remover o arquivo ZIP que você baixou.

Agora digite

```
cd dna
```

seguido de Enter para entrar (ou seja, abrir) nesse diretório. Seu prompt agora deve se parecer com o abaixo.

```
dna/ $
```

Execute `ls` por si só e você verá alguns arquivos e pastas:

```
databases/ dna.py sequences/
```

Se você tiver algum problema, siga estas mesmas etapas novamente e veja se consegue determinar onde errou!

Fundo

O DNA, o portador da informação genética nos seres vivos, tem sido usado na justiça criminal há décadas. Mas como exatamente funciona o perfil de DNA? Dada uma sequência de DNA, como os investigadores forenses podem identificar a quem ela pertence?

Bem, o DNA é realmente apenas uma sequência de moléculas chamadas nucleotídeos, arranjadas em uma forma particular (uma dupla hélice). Cada célula humana tem bilhões de nucleotídeos dispostos em sequência. Cada nucleotídeo do DNA contém uma das quatro bases diferentes: adenina (A), citosina (C), guanina (G) ou timina (T). Algumas porções dessa sequência (ou seja, genoma) são as mesmas, ou pelo menos muito semelhantes, em quase todos os seres humanos, mas outras porções da sequência têm uma diversidade genética maior e, portanto, variam mais na população.

Um lugar onde o DNA tende a ter alta diversidade genética é em Short Tandem Repeats (STRs). Um STR é uma sequência curta de bases de DNA que tende a se repetir consecutivamente várias vezes em locais específicos dentro do DNA de uma pessoa. O número de vezes que qualquer STR específico se repete varia muito entre os indivíduos. Nas amostras de DNA abaixo, por exemplo, Alice tem o STR `AGAT` repetido quatro vezes em seu DNA, enquanto Bob tem o mesmo STR repetido cinco vezes.

Alice: CTAGATAGATAGATAGATGACTA

Bob: CTAGATAGATAGATAGATAGATT

Using multiple STRs, rather than just one, can improve the accuracy of DNA profiling. If the probability that two people have the same number of repeats for a single STR is 5%, and the analyst looks at 10 different STRs, then the probability that two DNA samples match purely by chance is about 1 in 1 quadrillion

(assuming all STRs are independent of each other). So if two DNA samples match in the number of repeats for each of the STRs, the analyst can be pretty confident they came from the same person. CODIS, the FBI's [DNA database \(https://www.fbi.gov/services/laboratory/biometric-analysis/codis/codis-and-ndis-fact-sheet\)](https://www.fbi.gov/services/laboratory/biometric-analysis/codis/codis-and-ndis-fact-sheet), uses 20 different STRs as part of its DNA profiling process.

What might such a DNA database look like? Well, in its simplest form, you could imagine formatting a DNA database as a CSV file, wherein each row corresponds to an individual, and each column corresponds to a particular STR.

```
name,AGAT,AATG,TATC
Alice,28,42,14
Bob,17,22,19
Charlie,36,18,25
```

The data in the above file would suggest that Alice has the sequence `AGAT` repeated 28 times consecutively somewhere in her DNA, the sequence `AATG` repeated 42 times, and `TATC` repeated 14 times. Bob, meanwhile, has those same three STRs repeated 17 times, 22 times, and 19 times, respectively. And Charlie has those same three STRs repeated 36, 18, and 25 times, respectively.

So given a sequence of DNA, how might you identify to whom it belongs? Well, imagine that you looked through the DNA sequence for the longest consecutive sequence of repeated `AGAT`s and found that the longest sequence was 17 repeats long. If you then found that the longest sequence of `AATG` is 22 repeats long, and the longest sequence of `TATC` is 19 repeats long, that would provide pretty good evidence that the DNA was Bob's. Of course, it's also possible that once you take the counts for each of the STRs, it doesn't match anyone in your DNA database, in which case you have no match.

In practice, since analysts know on which chromosome and at which location in the DNA an STR will be found, they can localize their search to just a narrow section of DNA. But we'll ignore that detail for this problem.

Your task is to write a program that will take a sequence of DNA and a CSV file containing STR counts for a list of individuals and then output to whom the DNA (most likely) belongs.

Specification

In a file called `dna.py`, implement a program that identifies to whom a sequence of DNA belongs.

- The program should require as its first command-line argument the name of a CSV file containing the STR counts for a list of individuals and should require as its second command-line argument the name of a text file containing the DNA sequence to identify.
 - If your program is executed with the incorrect number of command-line arguments, your program should print an error message of your choice (with `print`). If the correct number of arguments are provided, you may assume that the first argument is indeed the filename of a valid CSV file and that the second argument is the filename of a valid text file.
- Your program should open the CSV file and read its contents into memory.
 - You may assume that the first row of the CSV file will be the column names. The first column will be the word `name` and the remaining columns will be the STR sequences themselves.
- Your program should open the DNA sequence and read its contents into memory.
- For each of the STRs (from the first line of the CSV file), your program should compute the longest run of consecutive repeats of the STR in the DNA sequence to identify. Notice that we've defined a helper

function for you, `longest_match`, which will do just that!

- If the STR counts match exactly with any of the individuals in the CSV file, your program should print out the name of the matching individual.
 - You may assume that the STR counts will not match more than one individual.
 - If the STR counts do not match exactly with any of the individuals in the CSV file, your program should print `No match`.

Walkthrough



Usage

Your program should behave per the example below:

```
$ python dna.py databases/large.csv sequences/5.txt
Lavender
```

```
$ python dna.py
Usage: python dna.py data.csv sequence.txt
```

```
$ python dna.py data.csv
Usage: python dna.py data.csv sequence.txt
```

Hints

- You may find Python's `csv` (<https://docs.python.org/3/library/csv.html>) module helpful for reading CSV files into memory. You may want to take advantage of either `csv.reader` (<https://docs.python.org/3/library/csv.html#csv.reader>) or `csv.DictReader` (<https://docs.python.org/3/library/csv.html#csv.DictReader>).

- The `open` (<https://docs.python.org/3.3/tutorial/inputoutput.html#reading-and-writing-files>) and `read` (<https://docs.python.org/3.3/tutorial/inputoutput.html#methods-of-file-objects>) functions may prove useful for reading text files into memory.
- Consider what data structures might be helpful for keeping tracking of information in your program. A `list` (<https://docs.python.org/3/tutorial/introduction.html#lists>) or a `dict` (<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>) may prove useful.
- Remember we've defined a function (`longest_match`) that, given both a DNA sequence and an STR as inputs, returns the maximum number of times that the STR repeats. You can then use that function in other parts of your program!

Testing

While `check50` is available for this problem, you're encouraged to first test your code on your own for each of the following.

- Execute seu programa como `python dna.py databases/small.csv sequences/1.txt`. Seu programa deve produzir `Bob`.
- Execute seu programa como `python dna.py databases/small.csv sequences/2.txt`. Seu programa deve produzir `No match`.
- Execute seu programa como `python dna.py databases/small.csv sequences/3.txt`. Seu programa deve produzir `No match`.
- Execute seu programa como `python dna.py databases/small.csv sequences/4.txt`. Seu programa deve produzir `Alice`.
- Execute seu programa como `python dna.py databases/large.csv sequences/5.txt`. Seu programa deve produzir `Lavender`.
- Execute seu programa como `python dna.py databases/large.csv sequences/6.txt`. Seu programa deve produzir `Luna`.
- Execute seu programa como `python dna.py databases/large.csv sequences/7.txt`. Seu programa deve produzir `Ron`.
- Execute seu programa como `python dna.py databases/large.csv sequences/8.txt`. Seu programa deve produzir `Ginny`.
- Execute seu programa como `python dna.py databases/large.csv sequences/9.txt`. Seu programa deve produzir `Draco`.
- Execute seu programa como `python dna.py databases/large.csv sequences/10.txt`. Seu programa deve produzir `Albus`.
- Execute seu programa como `python dna.py databases/large.csv sequences/11.txt`. Seu programa deve produzir `Hermione`.
- Execute seu programa como `python dna.py databases/large.csv sequences/12.txt`. Seu programa deve produzir `Lily`.
- Execute seu programa como `python dna.py databases/large.csv sequences/13.txt`. Seu programa deve produzir `No match`.
- Execute seu programa como `python dna.py databases/large.csv sequences/14.txt`. Seu programa deve produzir `Severus`.
- Execute seu programa como `python dna.py databases/large.csv sequences/15.txt`. Seu programa deve produzir `Sirius`.

- Execute seu programa como `python dna.py databases/large.csv sequences/16.txt`. Seu programa deve produzir `No match`.
- Execute seu programa como `python dna.py databases/large.csv sequences/17.txt`. Seu programa deve produzir `Harry`.
- Execute seu programa como `python dna.py databases/large.csv sequences/18.txt`. Seu programa deve produzir `No match`.
- Execute seu programa como `python dna.py databases/large.csv sequences/19.txt`. Seu programa deve produzir `Fred`.
- Execute seu programa como `python dna.py databases/large.csv sequences/20.txt`. Seu programa deve produzir `No match`.

Execute o abaixo para avaliar a exatidão do seu código usando `check50`. Mas certifique-se de compilar e testar você mesmo também!

```
check50 cs50/problems/2023/x/dna
```

Execute o abaixo para avaliar o estilo do seu código usando `style50`.

```
style50 dna.py
```

Como enviar

Em seu terminal, execute o abaixo para enviar seu trabalho.

```
submit50 cs50/problems/2023/x/dna
```