Este é o CS50

Introdução à Ciência da Computação (CS50)

OpenCourseWare

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

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

DNA

Problema a resolver

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

Em um arquivo chamado `<nome_do_arquivo>` localizado `dna.py` em uma pasta chamada `<nome_da_pasta>` `dna`, implemente um programa que identifique a quem pertence uma sequência de DNA.

Demonstração

```
$ python dna.py databases/small.csv sequences/1.txt
Bob
$ python dna.py databases/small.csv sequences/2.txt
No match
$ python dna.py databases/large.csv sequences/19.txt
Fred
$
```

Recorded with **asciinema**

Código de Distribuição

Para este problema, você deverá estender a funcionalidade do código fornecido pela equipe do CS50.

▼ **Baixe o código de distribuição.**

Faça login em cs50.dev (https://cs50.dev/) , clique na janela do terminal e execute `cd` o comando. Você verá que o prompt do terminal será semelhante ao seguinte:

```
$
```

Em seguida, execute

```
wget https://cdn.cs50.net/2024/fall/psets/6/dna.zip
```

para baixar um arquivo ZIP chamado `dna.zip` para o seu espaço de código.

Em seguida, execute

```
unzip dna.zip
```

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

```
rm dna.zip
```

e responda com “y” seguido de Enter quando solicitado para remover o arquivo ZIP que você baixou.

Agora digite

```
cd dna
```

Em seguida, pressione Enter para entrar (ou seja, abrir) esse diretório. Seu prompt agora deve ser semelhante ao abaixo.

```
dna/ $
```

Execute o programa `ls` isoladamente e você deverá ver alguns arquivos e pastas:

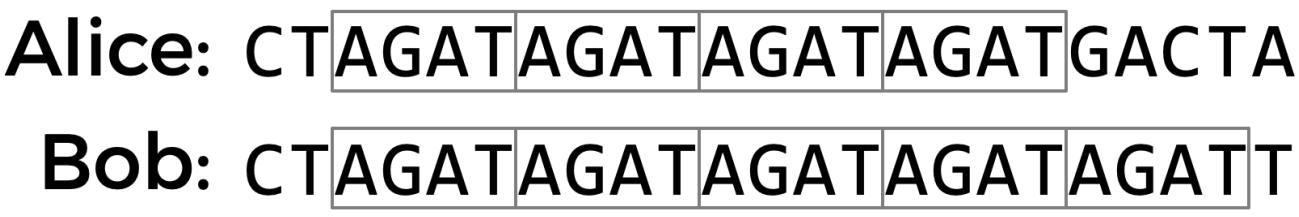
```
databases/ dna.py sequences/
```

Se você encontrar algum problema, siga esses mesmos passos novamente e veja se consegue determinar onde errou!

Fundo

O DNA é, na verdade, uma sequência de moléculas chamadas nucleotídeos, organizadas em uma forma específica (uma dupla hélice). Cada célula humana possui bilhões de nucleotídeos dispostos em sequência. Cada nucleotídeo do DNA contém uma das quatro bases nitrogenadas: adenina (A), citosina (C), guanina (G) ou timina (T). Algumas porções dessa sequência (ou seja, o genoma) são iguais, ou pelo menos muito semelhantes, em quase todos os seres humanos, mas outras porções da sequência apresentam maior diversidade genética e, portanto, variam mais entre as populações.

Uma região do DNA onde a diversidade genética tende a ser alta é nas Repetições em Tandem Curtas (STRs). Uma STR é uma pequena sequência de bases de DNA que tende a se repetir consecutivamente inúmeras vezes em locais específicos do DNA de uma pessoa. O número de repetições de uma STR específica varia bastante entre os indivíduos. Nas amostras de DNA abaixo, por exemplo, Alice tem a STR `AGAT` repetida quatro vezes em seu DNA, enquanto Bob tem a mesma STR repetida cinco vezes.



Utilizar múltiplos STRs, em vez de apenas um, pode melhorar a precisão do perfilamento de DNA. Se a probabilidade de duas pessoas terem o mesmo número de repetições para um único STR for de 5%, e o analista examinar 10 STRs diferentes, então a probabilidade de duas amostras de DNA coincidirem puramente por acaso é de cerca de 1 em 1 quatrilhão (assumindo que todos os STRs sejam independentes entre si). Portanto, se duas amostras de DNA coincidirem no número de repetições para cada um dos STRs, o analista pode ter bastante certeza de que elas vieram da mesma pessoa. O CODIS, o banco de dados de DNA

(https://www.fbi.gov/services/laboratory/biometric-analysis/codis/codis-and-ndis-fact-sheet) do FBI , utiliza 20 STRs diferentes como parte de seu processo de perfilamento de DNA.

Como seria um banco de dados de DNA desse tipo? Bem, em sua forma mais simples, você pode imaginar um banco de dados de DNA formatado como um arquivo CSV, onde cada linha corresponde a um indivíduo e cada coluna corresponde a um STR específico.

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

Os dados do arquivo acima sugerem que Alice possui a sequência `AGAT` repetida 28 vezes consecutivas em algum lugar do seu DNA, a sequência `AATG` repetida 42 vezes e a sequência `TATC` repetida 14 vezes. Bob, por sua vez, possui esses mesmos três STRs repetidos 17, 22 e 19 vezes, respectivamente. E Charlie possui esses mesmos três STRs repetidos 36, 18 e 25 vezes, respectivamente.

Então, dada uma sequência de DNA, como você poderia identificar a quem ela pertence? Bem, imagine que você procurou na sequência de DNA a maior sequência consecutiva de repetições de `AGAT` STRs e descobriu que a sequência mais longa tinha 17 repetições. Se você então descobrisse que a sequência mais longa de STRs `AATG` tinha 22 repetições e a sequência mais longa de STRs `TATC` tinha 19 repetições, isso forneceria uma evidência bastante forte de que o DNA era de Bob. Claro, também é possível que, ao contabilizar cada um dos STRs, não haja correspondência com ninguém em seu banco de dados de DNA, caso em que você não teria nenhuma correspondência.

Na prática, como os analistas sabem em qual cromossomo e em qual localização do DNA um STR será encontrado, eles podem restringir sua busca a uma seção estreita do DNA. Mas vamos ignorar esse detalhe para este problema.

Sua tarefa é escrever um programa que receba uma sequência de DNA e um arquivo CSV contendo a contagem de STRs para uma lista de indivíduos e, em seguida, indique a quem o DNA (mais provavelmente) pertence.

Especificação

- O programa deve exigir como primeiro argumento da linha de comando o nome de um arquivo CSV contendo a contagem de STRs para uma lista de indivíduos e como segundo argumento da linha de comando o nome de um arquivo de texto contendo a sequência de DNA a ser identificada.
    - Se o seu programa for executado com um número incorreto de argumentos de linha de comando, ele deverá imprimir uma mensagem de erro de sua escolha (com `--error` `print`). Se o número correto de argumentos for fornecido, você pode assumir

que o primeiro argumento é de fato o nome de um arquivo CSV válido e que o segundo argumento é o nome de um arquivo de texto válido.

- Seu programa deve abrir o arquivo CSV e ler seu conteúdo para a memória.
  - Você pode assumir que a primeira linha do arquivo CSV conterá os nomes das colunas. A primeira coluna será a palavra `name` e as colunas restantes serão as próprias sequências STR.
- Seu programa deve abrir a sequência de DNA e ler seu conteúdo para a memória.
- Para cada um dos STRs (da primeira linha do arquivo CSV), seu programa deve calcular a maior sequência de repetições consecutivas do STR na sequência de DNA a ser identificada. Observe que definimos uma função auxiliar para você, `longest_match` que fará exatamente isso!
- Se a contagem de STR corresponder exatamente a algum dos indivíduos no arquivo CSV, seu programa deverá imprimir o nome do indivíduo correspondente.
  - Pode-se assumir que a contagem de STR não corresponderá a mais de um indivíduo.
  - Se a contagem de STR não corresponder exatamente a nenhum dos indivíduos no arquivo CSV, seu programa deverá imprimir `No match`.

Dicas

- Você pode achar `csv` (https://docs.python.org/3/library/csv.html)o módulo do Python útil para ler arquivos CSV na memória. De particular ajuda pode ser o módulo `read_csv` `csv.DictReader` (https://docs.python.org/3/library/csv.html#csv.DictReader).
  - Por exemplo, se um arquivo como esse `foo.csv` tiver uma linha de cabeçalho, onde cada string é o nome de algum campo, veja como você poderia imprimi-las `fieldnames` como um `<li>` `list`:

    ```
    import csv

    with open("foo.csv") as file:
        reader = csv.DictReader(file)
        print(reader.fieldnames)
    ```

  - E aqui está como você lê todas as (outras) linhas de um arquivo CSV para uma lista `list`, onde cada elemento é uma lista `dict` que representa essa linha:

    ```
    import csv

    rows = []
    with open("foo.csv") as file:
        reader = csv.DictReader(file)
        for row in reader:
            rows.append(row)
    ```

- As funções `open` (https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files)`and` `read` (https://docs.python.org/3/tutorial/inputoutput.html#methods-of-file-objects)também podem ser úteis para ler arquivos de texto na memória.

- Considere quais estruturas de dados podem ser úteis para manter o controle das informações em seu programa. Um `ArrayList` `list` (https://docs.python.org/3/tutorial/introduction.html#lists)ou um `ArrayList` `dict` (https://docs.python.org/3/tutorial/datastructures.html#dictionaries)podem ser úteis.
- Lembre-se de que definimos uma função (`longest_match`) que, dado uma sequência de DNA e um STR como entradas, retorna o número máximo de vezes que o STR se repete. Você pode então usar essa função em outras partes do seu programa!

Passo a passo

Como testar

Embora `check50` exista uma solução disponível para este problema, recomendamos que você primeiro teste seu código por conta própria para cada um dos seguintes itens.

- Execute seu programa como `python dna.py databases/small.csv sequences/1.txt`. Seu programa deve exibir a saída `Bob`.
- Execute seu programa como `python dna.py databases/small.csv sequences/2.txt`. Seu programa deve exibir a saída `No match`.
- Execute seu programa como `python dna.py databases/small.csv sequences/3.txt`. Seu programa deve exibir a saída `No match`.
- Execute seu programa como `python dna.py databases/small.csv sequences/4.txt`. Seu programa deve exibir a saída `Alice`.
- Execute seu programa como `python dna.py databases/large.csv sequences/5.txt`. Seu programa deve exibir a saída `Lavender`.

- Execute seu programa como `python dna.py databases/large.csv sequences/6.txt`. Seu programa deve exibir a saída `Luna`.
- Execute seu programa como `python dna.py databases/large.csv sequences/7.txt`. Seu programa deve exibir a saída `Ron`.
- Execute seu programa como `python dna.py databases/large.csv sequences/8.txt`. Seu programa deve exibir a saída `Ginny`.
- Execute seu programa como `python dna.py databases/large.csv sequences/9.txt`. Seu programa deve exibir a saída `Draco`.
- Execute seu programa como `python dna.py databases/large.csv sequences/10.txt`. Seu programa deve exibir a saída `Albus`.
- Execute seu programa como `python dna.py databases/large.csv sequences/11.txt`. Seu programa deve exibir a saída `Hermione`.
- Execute seu programa como `python dna.py databases/large.csv sequences/12.txt`. Seu programa deve exibir a saída `Lily`.
- Execute seu programa como `python dna.py databases/large.csv sequences/13.txt`. Seu programa deve exibir a saída `No match`.
- Execute seu programa como `python dna.py databases/large.csv sequences/14.txt`. Seu programa deve exibir a saída `Severus`.
- Execute seu programa como `python dna.py databases/large.csv sequences/15.txt`. Seu programa deve exibir a saída `Sirius`.
- Execute seu programa como `python dna.py databases/large.csv sequences/16.txt`. Seu programa deve exibir a saída `No match`.
- Execute seu programa como `python dna.py databases/large.csv sequences/17.txt`. Seu programa deve exibir a saída `Harry`.
- Execute seu programa como `python dna.py databases/large.csv sequences/18.txt`. Seu programa deve exibir a saída `No match`.
- Execute seu programa como `python dna.py databases/large.csv sequences/19.txt`. Seu programa deve exibir a saída `Fred`.
- Execute seu programa como `python dna.py databases/large.csv sequences/20.txt`. Seu programa deve exibir a saída `No match`.

Correção

```
check50 cs50/problems/2025/x/dna
```

Estilo

```
style50 dna.py
```

Como enviar

```
submit50 cs50/problems/2025/x/dna
```