**Fundação CECIERJ - Vice Presidência de Educação Superior a Distância**
**Curso de Tecnologia em Sistemas de Computação**
**Disciplina: Programação com Interfaces Gráficas**
**AD2 $1^o$ semestre de 2020.**
**Professores: Mario Benevides e Paulo Roma**

---

# AD2: DNA Strand

## 1 Objetivo

O objetivo da AD2 é complementar as tarefas que ficaram faltando na AD1:

- Implemente uma interface gráfica, em TkInter, para visualizar o processo de casamento de fitas de DNA, e que permita executar todas as tarefas necessárias, permitindo:
  - obter duas fitas de DNA via arquivo (utilize um formato adequado)
  - ou digitando-as diretamente na interface.
- A saída do programa também deve ser efetuada usando componentes de interface apropriados.
  - permitir deslizar continuamente a segunda fita, em relação a primeira, para a direita ou esquerda, para cima ou para baixo,
  - e obter o número de casamentos em cada configuração.
- A interface implementada deve ser a mais adequada possível aos requisitos da aplicação. Não há um formato fixo. Use a sua criatividade e bom senso.

## 2 Diretivas Gerais

Uma possibilidade interessante é fornecer parâmetros diretamente na linha de comando, ao estilo Unix. Perceba que são necessários poucos parâmetros, como duas strings de DNA e talvez um indicador de saída verbosa, para impressão de dados de *debugging*. As strings

de DNA podem ser geradas internamente, como caracteres aleatórios, a partir de tamanhos fornecidos, ou serem passadas diretamente na linha de comando.

Python possui um módulo muito útil, chamado getopt [1], que pode ser usado para obter valores passados na linha de comando:

Listing 1: getopt - Linha de Comando

```
##
 # Instancia um objeto da classe Tk, responsável por criar
 # a janela principal. Aceita, na linha de comando,
 # quatro argumentos:
 #
 #  @param h help
 #  @param n tamanho do DNA1
 #  @param m tamanho do DNA2
 #  @param v modo verboso
 #
 #  Uso:
 #  -  move.py -n 6 -m 7 -v or
 #  -  move.py --dna1=6 --dna2=7 -v or
 #  -  move.py --help
 #
def main(argv=None):
    if argv is None:
       argv = sys.argv

    n1 = n2 = 0
    debug = False
    try:
        try:
            # Opções, que requeiram um argumento, devem ser
            # seguidas por dois pontos (:).
            # Opções longas, que requeiram um argumento,
            # devem ser seguidas por um sinal de igual ('=').
            opts, args = getopt.getopt(argv[1:], "hn:m:v", \
                         ["help","dna1=","dna2=","verbose"])
        except getopt.GetoptError as msg:
            raise ValueError (str(msg))
        # opts é uma lista de opções com pares [(option1, argument1),
        #                                       (option2, argument2)]
        # args é a lista de argumentos de programa que sobra
        # após a lista de opções ser removida,
        # por exemplo, "move.py -h --help 1 2",
        # faz opts e args serem:
        # [('-h', ''), ('--help', '')] ['1', '2']
        for opt,arg in opts:  # alguma coisa como [('-h', '')] or
                              #                   [('--help', '')]
            if opt in ( "-h", "--help" ):
               print ("Usage move.py -n1 <DNA1_length> \
                                     -n2 <DNA2_length> -v")
               return 1
            elif opt in ("-n", "--dna1"):
```

[1] https://docs.python.org/3.1/library/getopt.html

```
            n1 = int(arg)
        elif opt in ("-m", "--dna2"):
            n2 = int(arg)
        elif opt in ( "-v", "--verbose" ):
             debug = True
except ValueError as err:
    print (str(err) + "\nFor help, type: %s --help" % argv[0])
    return 2

master = Tk()
master.title ("DNA Strand")
gfg = GFG(master, n1, n2, debug)

            .....
```

O código 1 permite fornecer quatro argumentos opcionais, na linha de comando, e ao final da sua execução, as variáveis n1, n2 e debug terão seus valores possivelmente modificados:

```
move.py -m7 -v -n 6 ou
move.py --dna1=6 --dna2=7 -v ou
move.py --help
```

A string "hn:m:v" no primeiro try, indica as iniciais dos parâmetros, onde ':' obriga um valor associado ao parâmetro:

1. h help
2. n: tamanho do primeiro DNA (requer valor)
3. m: tamanho do segundo DNA (requer valor)
4. v modo verboso

Use o código 1 para obter valores a serem passados ao construtor de um DNAStrand. Procure entendê-lo completamente, principalmente o porque dos dois **try** aninhados.

**Dica**: Pode-se sempre utilizar o método **move** do componente canvas para mover um texto na sua implementação [2]. Uma possibilidade é usar as setas do teclado para deslocar a string uma posição para direita, esquerda, para cima ou para baixo. O mais difícil é calcular a largura das letras para mover adequadamente. Procure usar uma fonte de texto mono espaçada (com largura fixa).

# 3 Tarefas complementares

1. Inclua uma opção **reset** para retornar o texto para a posição inicial. Se o texto sair totalmente da janela, o reset deve ser chamado automaticamente [3].

[2] https://www.geeksforgeeks.org/python-tkinter-moving-objects-using-canvas-move-method/
[3] http://orion.lcg.ufrj.br/python/ADs/move.mp4

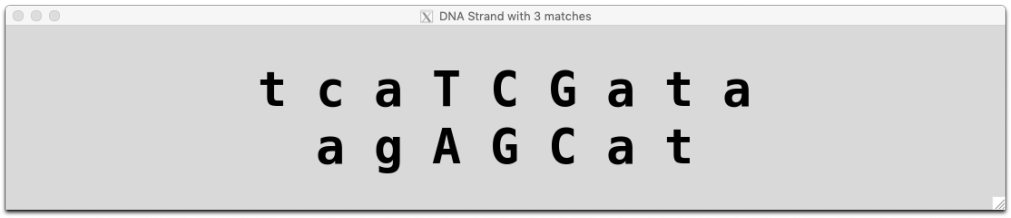


Figura 1: DNA Strand

2. Implemente uma opção para posicionar o texto na posição que gera o maior número possível de casamentos. É necessário alterar algum método do DNAStrand para isso? Qual?

3. Eu escolhi as seguintes teclas para controlar a aplicação:
DNA Strand - T matches A and C matches G

   $\rightarrow$ : move right

   $\leftarrow$ : move left

   $\uparrow$ : move up

   $\downarrow$ : move down

   $Shift - L$: shuffle

   $Tab$: reset

   $Escape$: exit

   h: help

   m: go to position of maximum matches

   Utilizem, adicionalmente, botões, para as funções correspondentes.

4. Permita que o usuário redimensione a janela, mas sempre mantendo os dois textos centrados. Use o bind $< Configure >$ [4], associado a um método resize, para conseguir isso.

[4] `http://archive.oreilly.com/oreillyschool/courses/Python2/MoreAboutGraphicalUserInterfaces.html`