*PROVA SUBSTITUTIVA*

**MAC0119 INTRODUÇÃO À PROGRAMAÇÃO DE COMPUTADORES (ICB)**

***2º SEMESTRE DE 2025***

**Nome completo:** _____

**NUSP:** _____

**Instruções:**

(*1*) Esta prova é individual.
(*2*) A prova consiste de **4** questões (contando a Questão 0 nesta página). Note que é possível tirar mais de 10 nesta prova `:-)`
(*3*) Para ter nota integral em uma questão, a escrita de sua solução deve estar boa.
(*4*) As respostas devem estar nos locais indicados.
(*5*) Não é permitido o uso de aparelhos eletrônicos de qualquer natureza.
(*6*) Não destaque as folhas deste caderno.
(*7*) Não use folhas avulsas para rascunho. Não é necessário apagar seus rascunhos.
(*8*) Não é permitido consultar colegas.
(*9*) O único material que você pode consultar é seu *cheat sheet* (uma folha A4, frente e verso).

*Assinatura*:

*Sua assinatura acima atesta a autenticidade e originalidade de seu trabalho e que você compromete-se a seguir o código de ética da USP em todas as suas atividades, incluindo esta prova.*

Boa sorte!

| **Q** | 0 | 1 | 2 | 3 | **Total** |
|---|---|---|---|---|---|
| **Nota** | | | | | |

**Q0.** **[0.5 pontos]** Leia o conteúdo desta página e preencha os itens requisitados. Assine acima, e atente ao significado de sua assinatura.

*Data*: 2026/2/2, 2:30pm

**Q1.** [**3.5 pontos**] Considere o seguinte código.

```
aa = ['F','L','I','M','V','S','P','T','A','Y',
      '|','H','Q','N','K','D','E','C','W','R',
      'G']

codons = [['TTT', 'TTC'],
          ['TTA', 'TTG', 'CTT', 'CTC', 'CTA', 'CTG'],
          ['ATT', 'ATC', 'ATA'],
          ['ATG'],
          ['GTT', 'GTC', 'GTA', 'GTG'],
          ['TCT', 'TCC', 'TCA', 'TCG', 'AGT', 'AGC'],
          ['CCT', 'CCC', 'CCA', 'CCG'],
          ['ACT', 'ACC', 'ACA', 'ACG'],
          ['GCT', 'GCC', 'GCA', 'GCG'],
          ['TAT', 'TAC'],
          ['TAA', 'TAG', 'TGA'],
          ['CAT', 'CAC'],
          ['CAA', 'CAG'],
          ['AAT', 'AAC'],
          ['AAA', 'AAG'],
          ['GAT', 'GAC'],
          ['GAA', 'GAG'],
          ['TGT', 'TGC'],
          ['TGG'],
          ['CGT', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG'],
          ['GGT', 'GGC', 'GGA', 'GGG']]

def compBase(b):
    if b == 'A':
        return 'T'
    if b == 'T':
        return 'A'
    if b == 'C':
        return 'G'
    if b == 'G':
        return 'C'

def reverseComplement(DNA):
    rc = ''
    for x in DNA[::-1]:
        rc += compBase(x)
    return rc

def amino(codon):
    for i in range(len(aa)):
        if codon in codons[i]:
            return aa[i]
```

```
def codingStrandToAA(DNA):
    p = ''
    for i in range(0, len(DNA), 3):
        codon = DNA[i : i + 3]
        p += amino(codon)
    return p

def codingRCStrandToAA(DNA):
    rc = reverseComplement(DNA)
    return codingStrandToAA(rc)
```

Em cada item abaixo, é dada uma chamada de função, complementando o código acima. Você deve dizer em cada item o valor devolvido pela chamada dada.

($i$) `reverseComplement('GCTTGTCTCCTTGTGCAG')`

*Resposta*:

($ii$) `amino('GCT')`

*Resposta*:

($iii$) `codingStrandToAA('GCTTGTCTCCTTGTGCAG')`

*Resposta*:

($iv$) `codingRCStrandToAA('GCTTGTCTCCTTGTGCAG')`

*Resposta*:

**Q2.** [**4 pontos**] Considere as seguintes funções:

```
def epower(a, b):
    if b == 0:
        return 1
    else:
        return a * epower(a, b - 1)

def power(a, b):
    if b == 0:
        return 1
    t = power(a, b // 2)
    if b % 2 == 0:  # b even
        return t * t
    else:
        return a * t * t
```

(*i*) Qual é a saída do seguinte código?
`print(epower(2, 3))`

*Resposta*:

(*ii*) Qual é a saída do seguinte código?
`print(power(2, 3))`

*Resposta*:

(*iii*) Em geral, o que é o valor devolvido por `power(a, b)`, quando os argumentos `a` e `b` são inteiros positivos? Esboce uma justificativa para sua resposta.

*Resposta*:

*Resposta* (continuação):

(*iv*) Estamos agora interessados em **quantos produtos são executados no total** quando executamos `epower()` e `power()`. Note que, no caso de `epower()`, ocorre **um** produto quando a expressão `a * epower(a, b - 1)` é avaliada, mas mais produtos podem ser executados quando a expressão `epower(a, b - 1)` for avaliada.

Quantos produtos são executados **no total** quando executamos `epower(2, 3)`?

*Resposta*:

(*v*) Note que, no caso de `power()`, ocorre **um** produto quando a expressão `t * t` é avaliada, ocorrem **dois** produtos quando a expressão `a * t * t` é avaliada, e ocorre um certo número de produtos quando a chamada recursiva em

```
t = power(a, b // 2)
```

é executada.

Quantos produtos são executados **no total** quando executamos `power(2, 3)`?

*Resposta*:

(*vi*) Quantos produtos ocorrem **no total** quando executamos `epower(2, 1024)`? Esboce como você chegou a sua resposta.

*Resposta*:

(*vii*) Quantos produtos ocorrem **no total** quando executamos `power(2, 1024)`? Esboce como você chegou a sua resposta.

*Resposta*:

**Q3.** [**4 pontos**] Considere a seguinte função, que recebe como entrada duas sequências de caracteres `s` e `t`:

```
def edit_seq(s, t):
    if s == '' and t == '':
        return [0]
    if s == '':
        r = edit_seq('', t[1:])
        return [-1 + r[0]] + ['-/' + t[0]] + r[1:]
    if t == '':
        r = edit_seq(s[1:], '')
        return [-1 + r[0]] + [s[0] + '/-'] + r[1:]
    r1 = edit_seq(s[1:], t[1:])
    if s[0] == t[0]:
        option1 = 1 + r1[0]
    else:
        option1 = -1 + r1[0]
    r2 = edit_seq(s[1:], t) # delete
    option2 = -1 + r2[0]
    r3 = edit_seq(s, t[1:]) # insert
    option3 = -1 + r3[0]
    score = max(option1, option2, option3)
    if option1 == score:
        return [score] + [s[0] + '/' + t[0]] + r1[1:]
    if option2 == score:
        return [score] + [s[0] + '/-'] + r2[1:]
    return [score] + ['-/' + t[0]] + r3[1:]
```

A função acima "tem a estrutura" da função que calcula o "align score" das sequências `s` e `t` dadas.

Lembre que, ao calcularmos o align score de `s` e `t`, percorremos as letras de `s` da esquerda para a direita, e consideramos quatro operações para "transformar `s` em `t`": (0) mantemos a letra de `s` sendo considerada, (1) substituímos essa letra por outra, (2) inserimos uma nova letra naquele ponto de `s` e (3) removemos a letra de `s` sendo considerada. A operação de manter a letra nos dá um ponto positivo, enquanto que todas as outras operações nos dão um ponto negativo no score. O align score de `s` e `t` é o maior score possível.

Seguem alguns exemplos de execução de `edit_seq()`:

```
>>> edit_seq('', 'AT')
[-2, '-/A', '-/T']
>>> edit_seq('ATG', '')
[-3, 'A/-', 'T/-', 'G/-']
>>> edit_seq('ATG', 'CAG')
[0, '-/C', 'A/A', 'T/-', 'G/G']
```

```
>>> edit_seq('ATG', 'CGGAT')
[-2, '-/C', '-/G', '-/G', 'A/A', 'T/T', 'G/-']
>>>
```

A execução de `edit_seq(s, t)` tem como saída uma lista. O 0-ésimo elemento da lista é o align score de `s` e `t`. Os demais elementos da lista indicam como podemos partir da sequência `s` e chegar em `t`. Por exemplo, no caso em que `s = 'ATG'` e `t = CAG`, podemos partir de `s` e chegar em `t` fazendo o seguinte: inserimos `C` (indicado por `-/C`), mantemos `A` (indicado por `A/A`), removemos `T` (indicado por `T/-`) e finalmente mantemos `G` (indicado por `G/G`).

($i$) Escreva uma função de cabeçalho

```
def align(l):
```

que recebe uma lista `l` devolvida por `edit_seq(s, t)` e produz uma lista da forma `[x, s', a, t']`, onde `x` é o align score de `s` e `t`, as sequências `s'` e `t'` são as sequências `s` e `t` com algumas ocorrências adicionais de `-` para efeitos de "alinhamento", e `a` indica quais caracteres de `s` foram mantidos. Mais precisamente, sua função `align()` deve funcionar conjuntamente com a função `print_alignment()` abaixo:

```
def print_alignment(alignment):
    print('Score:', alignment[0])
    print("s':", alignment[1])
    print("   ", alignment[2])
    print("t':", alignment[3])
```

Os seguintes exemplos mostram o que se espera de sua função `align()`:

```
>>> l = edit_seq('ATG', 'CAG')
>>> l
[0, '-/C', 'A/A', 'T/-', 'G/G']
>>> align(l)
[0, '-ATG', '.|.|', 'CA-G']
>>> print_alignment(align(l))
Score: 0
s': -ATG
    .|.|
t': CA-G
>>> l = edit_seq('ATG', 'CGGAT')
>>> print_alignment(align(l))
Score: -2
s': ---ATG
    ...||.
t': CGGAT-
>>> l = edit_seq('GCCTGG', 'ACCGGA')
```

```
>>> l
[1, 'G/A', 'C/C', 'C/C', 'T/-', 'G/G', 'G/G', '-/A']
>>> print_alignment(align(l))
Score: 1
s': GCCTGG-
    .||.||.
t': ACC-GGA
>>>
```

Escreva aqui sua função `align()` por completo:

*Resposta*:

(*ii*) A função `edit_seq()` é bastante ineficiente. Escreva uma versão memoizada de `edit_seq()`. Sua função memoizada deve ter cabeçalho

```
def edit_seq_memo(s, t, memo):
```

onde `memo` é um dicionário.

*Resposta*:

*Rascunho*:

*Rascunho*: