Pontifícia Universidade Católica do Paraná
Escola Politécnica – Bacharelado em Ciência da Computação – 7º Período
Prof. Edson Emílio Scalabrin

Nome: Gustavo Hammerschmidt.

Trabalho: 01
Data de entrega: quinta-feira seguinte a 1ª avaliação somativa.
Grupo: até 5 integrantes.
Peso na nota do RA01: 40%

Descrição do problema: Dado 50 números inteiros de 1 a 50 e as seguintes combinações.

$a) \binom{n=50}{p=5}$ = 2.118.760

$b) \binom{n=50}{p=4}$ = 230.300

$c) \binom{n=50}{p=3}$ = 19.600

$d) \binom{n=50}{p=2}$ = 1.225

Dica: acompanhe o código em python (trabalho1.py) e output gerado e salvo (PRIME_console.txt) junto das questões a seguir.

Pede-se para:

1. gerar as combinações tal que: em (a) se obtenha 2.118.760 sequencias de 5 números diferentes, em (b) se obtenha 230.300 sequencias de 4 números diferentes, em (c) se obtenha 19.600 sequencias de 3 números diferentes, em (d) se obtenha 1.225 sequencias de 2 números diferentes. PROGRAMA 1.

```python

"""
    | Dados 50 números inteiros de 1 a 50 e as seguintes comninações.


    a) [ n = 50 | p = 5 ] = 2.118.760

    b) [ n = 50 | p = 4 ] = 230.300

    c) [ n = 50 | p = 3 ] = 19.600

    d) [ n = 50 | p = 2] = 1.225
"""
class Programas():


    # C Total -> 17 -> O(1)
    def __init__(self) -> None:

        self.values = list(range(1,51)) # Números de 1 a 50.                                                                                  # C1 <- 1

        self.combinations_a, self.combinations_b = list(combinations(self.values, 5)), list(combinations(self.values, 4))                    # C2 <- 2
        self.combinations_c, self.combinations_d = list(combinations(self.values, 3)), list(combinations(self.values, 2))                    # C3 <- 2

        self.sceneryC1, self.sceneryC2, self.sceneryC3, self.over = None, None, None, False                                                  # C4 <- 4

        self.database, self.database_header, self.db_backtest, self.db_backtest_header = [], [], [], []                                      # C5 <- 4

        self.all_gains, self.apostas = {}, {}                                                                                                 # C6 <- 2


    # C Total -> 1 -> O(1)
    # Dump scenery results.
    def storage(self, var, name) -> None:
```

```python

    # C Total -> 4 -> O(1)
    """
        1.  gerar as combinações tal que: em (a) se obtenha 2.118.760 sequencias de
            5 números diferentes, em (b) se obtenha 230.300 sequencias de 4 números
            diferentes, em (c) se obtenha 19.600 sequencias de 3 números diferentes,
            em (d) se obtenha 1.225 sequencias de 2 números diferentes. PROGRAMA 1.
    """
    @decorator_over
    def programa1(self) -> None:

        print("\nPrograma 1:\n")                                                                                                               # C1 <- 1
        print("\tSequências:")                                                                                                               # C2 <- 1
        print("\n\t\ta: {} combinações.\n\t\tb: {} combinações.".format(len(self.combinations_a), len(self.combinations_b)))                 # C3 <- 1
        print("\t\tc: {} combinações.\n\t\td: {} combinações.\n".format(len(self.combinations_c), len(self.combinations_d)))                 # C4 <- 1


    # C Total -> 5 + k*n -> O(k * n)
    """
        2.  encontrar o menor conjunto de sequencias de 5 números que contem todas
```

```python

##########################################################
#########   ####    ####    #####  ###    ###  ########
#########  # ## #  ###  #  ####  ###  #  ##  ########
#########  ##  ##  ##  ###  ###  ###  ##  #  ########
#########  ######  #  #####  ##  ###  ####   ########
##########################################################


prog = Programas()


##########################################################
##########################################################

prog.console(last_console_show=True, store_console=False)  # Mostrar o último console OU salvar os resultados dessa execução no arquivo console.txt.


prog.quick_test(test=False, programa_2_a_4=prog.programa2, programa_5_a_8=prog.programa5) # Testa duas funções e faz a análise temporal delas.

# last_console_show=True bloqueia a função quick_test de executar. Defina l.c.s. como False e test como True em quick test para fazer testes rápidos.

##########################################################
##########################################################


prog.load_database() # Carrega a base de dados csv.


prog.programa1() # Carrega as combinações e mostra as sequências.


##########################################################
##########################################################
```

trabalho1.py | PRIME_console.txt

```
Programa 1:

    Sequências:

        a: 2118760 combinações.
        b: 230300 combinações.
        c: 19600 combinações.
        d: 1225 combinações.


Gerando cenários:


     Programa 4:  Phase take5 finished -> len(conjunto) == 33572 sequências.
     Programa 2:  Phase take5 finished -> len(conjunto) == 60 sequências.
```

2. encontrar o menor conjunto de sequencias de 5 números que contém todas as sequencias de 2 números, cenário C1. PROGRAMA 2.

```
    # C Total -> 5 + k*n -> O(k * n)
    """
        2.  encontrar o menor conjunto de sequencias de 5 números que contem todas
            as sequencias de 2 números, cenário C1. PROGRAMA 2.

            Obs.: entendo uma sequência de 5 números que contém o maior número de
            sequências de 2 como sendo:

                    Ex.:   (  1 -> 5 -> 16 -> 27 -> 50 )

                    contendo as sequências:

                        - 1 -> 5 | 1 -> 16 | 1 -> 27 | 1 -> 50
                        - 5 -> 16 | 5 -> 27 | 5 -> 50
                        - 16 -> 27 -> 50 | 16 -> 50
                        - 27 -> 50

                    Pois, se for esperado que a sequência 1->50 seja encontrada
                    lado a lado, precisariamos de uma sequência com limite acima de 52.
                    - Menor combinação possível nesse cenário: (1->50->51->52->53).

                    E o exercício pede TODAS as sequências de 2 números!


        Explicação sobre as funções takes nos programas 2, 3 e 4:

                Combinação de 5, 2 a 2; 10 resultados possíveis.
                >>> print(list(combinations(['1','2','3','4','5'], 2)))
                ... [('1', '2'), ('1', '3'), ('1', '4'), ('1', '5'), ('2', '3'), ('2', '4'), ('2', '5'), ('3', '4'), ('3', '5'), ('4', '5')]

                Combinações de 5, 2 a 2, com um elemento repetido; 6 resultados possíveis.
                >>> print(list(combinations(['1','2','3','4','A'], 2)))
                ... [('1', '2'), ('1', '3'), ('1', '4'), ('1', 'A'), ('2', '3'), ('2', '4'), ('2', 'A'), ('3', '4'), ('3', 'A'), ('4', 'A')]


                !!!
                    Algumas funções takes tem o limite abaixo dos resultados possíveis para otimizar tempo de execução
                    e tamanho da função; obtive o número de comparação nas funções lambdas através de execução prévia
                    e outros testes.
                !!!
    """
    @decorator_over
```

```
@decorator_over
def programa2(self, load=False) -> None:

    take5 = lambda arr: not reduce( (lambda a, b: a or b), arr) # All true.                                    # C1 <- 1
    take4 = lambda arr: arr.count(False) == 6                                                                  # C2 <- 1
    take3 = lambda arr: arr.count(False) == 2                                                                  # C3 <- 1
    take2 = lambda arr: arr.count(False) == 1   # last chance.                                                 # C4 <- 1

    index_markup = (lambda x: str(x[0])+":"+str(x[1]))                                                         # C5 <- 1

    self.sceneryC1 = self.program_handler(program_number=2, savename='sceneryC1', combination_p=self.combinations_d,
                        combination_number=2, conjunto_index=index_markup, takes=[take5, take4, take3, take2], load=load)   # C6 <- k*n
```

```
    """
        C Total -> 11 + 2m + 5n + 4(n * o) + 2p

                -- m é, no pior caso, 20% de n -> m == 0.2n.

                --  n * o ~ constante * n

                -- p ocorre 20% das vezes -> p == 0.2 n * o

                -- k < 5

        C Total -> 11 + 5.4n + 4.4 * (k * n) -> O(k * n)
    """
    # Algoritmo para encontrar menor número de sequências.
    def program_handler(self, program_number=0, savename="", combination_p=[], combination_number=0, conjunto_index=(), takes=[], load=False) -> None:
```

```
    # Algoritmo para encontrar menor número de sequências.
    def program_handler(self, program_number=0, savename="", combination_p=[], combination_number=0, conjunto_index=(), takes=[], load=False) -> None:

        if load:                                                                                                    # C1 <- 1

            return self.loader(f'arrs/{savename}.pkl')                                                              # C2 <- 1

        conjunto, conjunto_matrix = list(), {}                                                                      # C3 <- 2

        for k in combination_p:                                                                                     # C4 <- m (m < C(50,5)==n)

            conjunto_matrix[conjunto_index(k)] = False                                                              # C5 <- m

        control, next_ = takes[0], 1    # Assumed take_order_based = [5, 4, 3, 2, 1] || Take 5 elements array with all combinations...   # C6 <- 2

        while True:                                                                                                 # C7 <- n := len(takes)

            for _ in self.combinations_a:                                                                           # C8 <- n * (o := 2.118.760)

                indexes = list(map(conjunto_index, list(combinations( _ , combination_number))))                    # C9 <- n * o

                taken = list(map((lambda i: conjunto_matrix[i]), indexes ))                                         # C10 <- n * o

                if control(taken):                                                                                  # C11 <- n * o

                    [ conjunto_matrix.__setitem__(i, True) for i in indexes ]                                       # C12 <- p (p * n < 0)

                    conjunto.append(_)                                                                              # C13 <- p

            print(f"\n\t Programa {program_number}:  Phase take{6-next_} finished -> len(conjunto) == {len(conjunto)} sequências.", end='')   # C14 <- n

            if list(conjunto_matrix.values()).count(False) == 0:  # comb(50, program number) = len(programa1 -> Cenário x)   # C15 <- n

                break                                                                                               # C16 <- 1

            else:                                                                                                   # C17  -----

                control = takes[next_]                                                                              # C18 <- n-1

                next_ += 1                                                                                          # C19 <- n-1

        self.storage(conjunto, f'arrs/{savename}.pkl')                                                              # C20 <- 1

        print(f"\n\n\t Cenário {program_number-1}: ", len(conjunto), "sequências.")                                # C21 <- 1

        del conjunto_matrix, control, next_                                                                         # C22 <- 3

        return conjunto[:]                                                                                          # C24 <- 1


# C Total -> 4 -> O(1)
```

Programas 3 e 4 chamam essa função também.

```
##########################################################
##########################################################

if not prog.over:

    print("\nGerando cenários:\n")


thread_sceneries = []

#                                           var load | (if True) ~ carrega os cenários salvos em arquivos .pkl.
#                                                    V
[thread_sceneries.append(threading.Thread(target=p, args=(False,))) for p in [prog.programa2, prog.programa3, prog.programa4]]


[thread.start() for thread in thread_sceneries]


[thread.join() for thread in thread_sceneries]

##########################################################
##########################################################
```

```
Gerando cenários:


    Programa 4:  Phase take5 finished -> len(conjunto) == 33572 sequências.
    Programa 2:  Phase take5 finished -> len(conjunto) == 60 sequências.
    Programa 4:  Phase take4 finished -> len(conjunto) == 52248 sequências.
    Programa 3:  Phase take5 finished -> len(conjunto) == 1422 sequências.
    Programa 2:  Phase take4 finished -> len(conjunto) == 161 sequências.
    Programa 4:  Phase take3 finished -> len(conjunto) == 58660 sequências.

    Cenário 3:  58660 sequências.

    Programa 2:  Phase take3 finished -> len(conjunto) == 170 sequências.
    Programa 3:  Phase take4 finished -> len(conjunto) == 2735 sequências.
    Programa 2:  Phase take2 finished -> len(conjunto) == 171 sequências.

    Cenário 1:  171 sequências.

    Programa 3:  Phase take3 finished -> len(conjunto) == 2799 sequências.

    Cenário 2:  2799 sequências.
```

3. encontrar o menor conjunto de sequencias de 5 números que contem todas as sequencias de 3 números, cenário C2. PROGRAMA 3.

```python
    # C Total -> 4 + k*n -> O(k * n)
    """
        3.  encontrar o menor conjunto de sequencias de 5 números que contem todas
            as sequencias de 3 números, cenário C2. PROGRAMA 3.
    """
    @decorator_over
    def programa3(self, load=False) -> None:

        take5 = lambda arr: not reduce( (lambda a, b: a or b), arr) # All true.                                  # C1 <- 1
        take4 = lambda arr: arr.count(False) == 4                                                                # C2 <- 1
        take3 = lambda arr: arr.count(False) == 2 # last chance.                                                 # C3 <- 1

        index_markup = (lambda x: str(x[0])+":"+str(x[1])+":"+str(x[2]))                                         # C4 <- 1

        self.sceneryC2 = self.program_handler(program_number=3, savename='sceneryC2', combination_p=self.combinations_c,
                                    combination_number=3, conjunto_index=index_markup, takes=[take5, take4, take3], load=load)          # C5 <- k*n
```

```python

##################################################
##################################################

if not prog.over:

    print("\nGerando cenários:\n")


thread_sceneries = []

#                                           var load | (if True) ~ carrega os cenários salvos em arquivos .pkl.
#                                                    V
[thread_sceneries.append(threading.Thread(target=p, args=(False,))) for p in [prog.programa2, prog.programa3, prog.programa4]]


[thread.start() for thread in thread_sceneries]


[thread.join() for thread in thread_sceneries]

##################################################
##################################################
```

```
Gerando cenários:


    Programa 4:  Phase take5 finished -> len(conjunto) == 33572 sequências.
    Programa 2:  Phase take5 finished -> len(conjunto) == 60 sequências.
    Programa 4:  Phase take4 finished -> len(conjunto) == 52248 sequências.
    Programa 3:  Phase take5 finished -> len(conjunto) == 1422 sequências.
    Programa 2:  Phase take4 finished -> len(conjunto) == 161 sequências.
    Programa 4:  Phase take3 finished -> len(conjunto) == 58660 sequências.

    Cenário 3:  58660 sequências.

    Programa 2:  Phase take3 finished -> len(conjunto) == 170 sequências.
    Programa 3:  Phase take4 finished -> len(conjunto) == 2735 sequências.
    Programa 2:  Phase take2 finished -> len(conjunto) == 171 sequências.

    Cenário 1:  171 sequências.

    Programa 3:  Phase take3 finished -> len(conjunto) == 2799 sequências.

    Cenário 2:  2799 sequências.
```

4. encontrar o menor conjunto de sequencias de 5 números que contem todas as sequencias de 4 números, cenário C3. PROGRAMA 4.

```python
    # C Total -> 4 + k*n -> O(k * n)
    """
        4.  encontrar o menor conjunto de sequencias de 5 números que contem todas
            as sequencias de 4 números, cenário C3. PROGRAMA 4.
    """
    @decorator_over
    def programa4(self, load=False) -> None:

        take5 = lambda arr: not reduce( (lambda a, b: a or b), arr) # All true.                                    # C1 <- 1
        take4 = lambda arr: arr.count(False) == 3                                                                  # C2 <- 1
        take3 = lambda arr: arr.count(False) == 1   # last chance.                                                 # C3 <- 1

        index_markup = (lambda x: str(x[0])+":"+str(x[1])+":"+str(x[2])+":"+str(x[3]))                             # C4 <- 1

        self.sceneryC3 = self.program_handler(program_number=4, savename='sceneryC3', combination_p=self.combinations_b,
                                              combination_number=4, conjunto_index=index_markup, takes=[take5, take4, take3], load=load)  # C5 <- k*n
```

```python
    ##################################################
    ##################################################

    if not prog.over:

        print("\nGerando cenários:\n")


    thread_sceneries = []

    #                                           var load | (if True) ~ carrega os cenários salvos em arquivos .pkl.
    #                                                    V
    [thread_sceneries.append(threading.Thread(target=p, args=(False,))) for p in [prog.programa2, prog.programa3, prog.programa4]]


    [thread.start() for thread in thread_sceneries]


    [thread.join() for thread in thread_sceneries]

    ##################################################
    ##################################################
```

```
Gerando cenários:


    Programa 4:  Phase take5 finished -> len(conjunto) == 33572 sequências.
    Programa 2:  Phase take5 finished -> len(conjunto) == 60 sequências.
    Programa 4:  Phase take4 finished -> len(conjunto) == 52248 sequências.
    Programa 3:  Phase take5 finished -> len(conjunto) == 1422 sequências.
    Programa 2:  Phase take4 finished -> len(conjunto) == 161 sequências.
    Programa 4:  Phase take3 finished -> len(conjunto) == 58660 sequências.

    Cenário 3:  58660 sequências.

    Programa 2:  Phase take3 finished -> len(conjunto) == 170 sequências.
    Programa 3:  Phase take4 finished -> len(conjunto) == 2735 sequências.
    Programa 2:  Phase take2 finished -> len(conjunto) == 171 sequências.

    Cenário 1:  171 sequências.

    Programa 3:  Phase take3 finished -> len(conjunto) == 2799 sequências.

    Cenário 2:  2799 sequências.
```

5. supondo que tais combinações representem um sistema de aposta de algum pais europeu, calcular o retorno de um(a) apostador(a) em cada um dos cenários: C1, C2 e C3. Para o cálculo do retorno considere as seguintes regras e valores:

   Resultados do item 5 depois dos items: a, b, c e d.

   a) se acertou 2 números em um cartão, então a renumeração é 4,16 Euros. PROGRAMA 5.

```
    # C Total -> 33 + n + k * n -> O(k * n)
    """
        5.  supondo que tais combinações representem um sistema de aposta de algum país europeu,
            calcular o retorno de um(a) apostador(a) em cada um dos cenários: C1, C2 e C3.
            Para o cálculo do retorno considere as seguintes regras e valores:

                a)  se acertou 2 números em um cartão, então a renumeração é 4,16 Euros. PROGRAMA 5.

                C1, C2, C3 -> Dar preferência a cartões com 4 acertos e, aí, os de 2.
    """
    @decorator_over
    def programa5(self) -> None:

        if self.database == []:                                                                                         # C1 <- 1

            self.load_database()                                                                                        # C2 <- n

        if not self.load_sceneries():                                                                                   # C3 <- 1

            print("\n\t\tCenários não carregados.\n")                                                                   # C4 <- 1

            return;                                                                                                     # C5 <- 0

        result = {}                                                                                                     # C6 <- 1

        def all_cards(call, scenery):                                                                                   # C7 <- 1

            points = []                                                                                                 # C8 <- 1

            [points.append(max([2 if m > 1 else m for m in [self.point_hits(i, db_i) for db_i in self.database]] )) for i in scenery]            # C9 <- k * n

            result[call] = int( ( points.count(2) ) * 4.19 )                                                            # C10 <- 1

            print('\t Programa 5 --> '+call+' terminou.')                                                               # C11 <- 1

        data_ = [self.sceneryC1, self.sceneryC2, self.sceneryC3[:15000],self.sceneryC3[15000:30000],
                                                 self.sceneryC3[30000:45000], self.sceneryC3[45000:]]                   # C12 <- 1

        threads = []                                                                                                    # C13 <- 1

        [threads.append(threading.Thread(target=all_cards, args=('thread '+str(i), data_[i],))) for i in range(0, len(data_))]          # C14 <- 6

        [t.start() for t in threads]                                                                                    # C15 <- 6

        [t.join() for t in threads]                                                                                     # C16 <- 6

        res = [result['thread 0'], result['thread 1'], sum(result[i] for i in list(result.keys())[2:])]                 # C17 <- 3

        self.all_gains['Programa 5'] = res                                                                              # C18 <- 1

        self.apostas['5'] = (f"\nPrograma 5: \n\n\t\tCenário 1: {res[0]} euros.\n\t\tCenário 2: {res[1]} euros.\n\t\tCenário 3: {res[2]} euros.\n") # C19 <- 1
```

b) se acertou 3 números em um cartão, então a renumeração é 11,89 Euros. Deve-se observar que neste caso a remuneração máxima inclui 1 cartão com 3 números e 3 cartões com 2 números. PROGRAMA 6.

```python
    # C Total -> 33 + n + k * n -> O(k * n)
    """
        b)  se acertou 3 números em um cartão, então a renumeração é 11,89 Euros. Deve-se observar
            que neste caso a remuneração máxima inclui 1 cartão com 3 números e 3 cartões com 2 números.
            PROGRAMA 6.
    """
    @decorator_over
    def programa6(self) -> None:

        if self.database == []:                                                                                                 # C1 <- 1

            self.load_database()                                                                                                # C2 <- n

        if not self.load_sceneries():                                                                                           # C3 <- 1

            print("\n\t\tCenários não carregados.\n")                                                                            # C4 <- 1

            return;                                                                                                             # C5 <- 0

        result = {}                                                                                                             # C6 <- 1

        def all_cards(call, scenery):                                                                                           # C7 <- 1

            points = []                                                                                                         # C8 <- 1

            [points.append(max([3 if m > 2 else m for m in [self.point_hits(i, db_i) for db_i in self.database]] )) for i in scenery]  # C9 <- k * n

            result[call] = int( ( points.count(3) + int(points.count(2)/3) ) * 11.89 )                                          # C10 <- 1

            print('\t Programa 6 --> '+call+' terminou.')                                                                       # C11 <- 1

        data_ = [self.sceneryC1, self.sceneryC2, self.sceneryC3[:15000], self.sceneryC3[15000:30000],
                                                 self.sceneryC3[30000:45000], self.sceneryC3[45000:]]                           # C12 <- 1

        threads = []                                                                                                            # C13 <- 1

        [threads.append(threading.Thread(target=all_cards, args=('thread '+str(i), data_[i],))) for i in range(0, len(data_))]  # C14 <- 6

        [t.start() for t in threads]                                                                                            # C15 <- 6

        [t.join() for t in threads]                                                                                             # C16 <- 6

        res = [result['thread 0'], result['thread 1'], sum(result[i] for i in list(result.keys())[2:])]                         # C17 <- 3

        self.all_gains['Programa 6'] = res                                                                                      # C18 <- 1

        self.apostas['6'] = (f"\nPrograma 6: \n\n\t\tCenário 1: {res[0]} euros.\n\t\tCenário 2: {res[1]} euros.\n\t\tCenário 3: {res[2]} euros.\n") # C19 <- 1
```

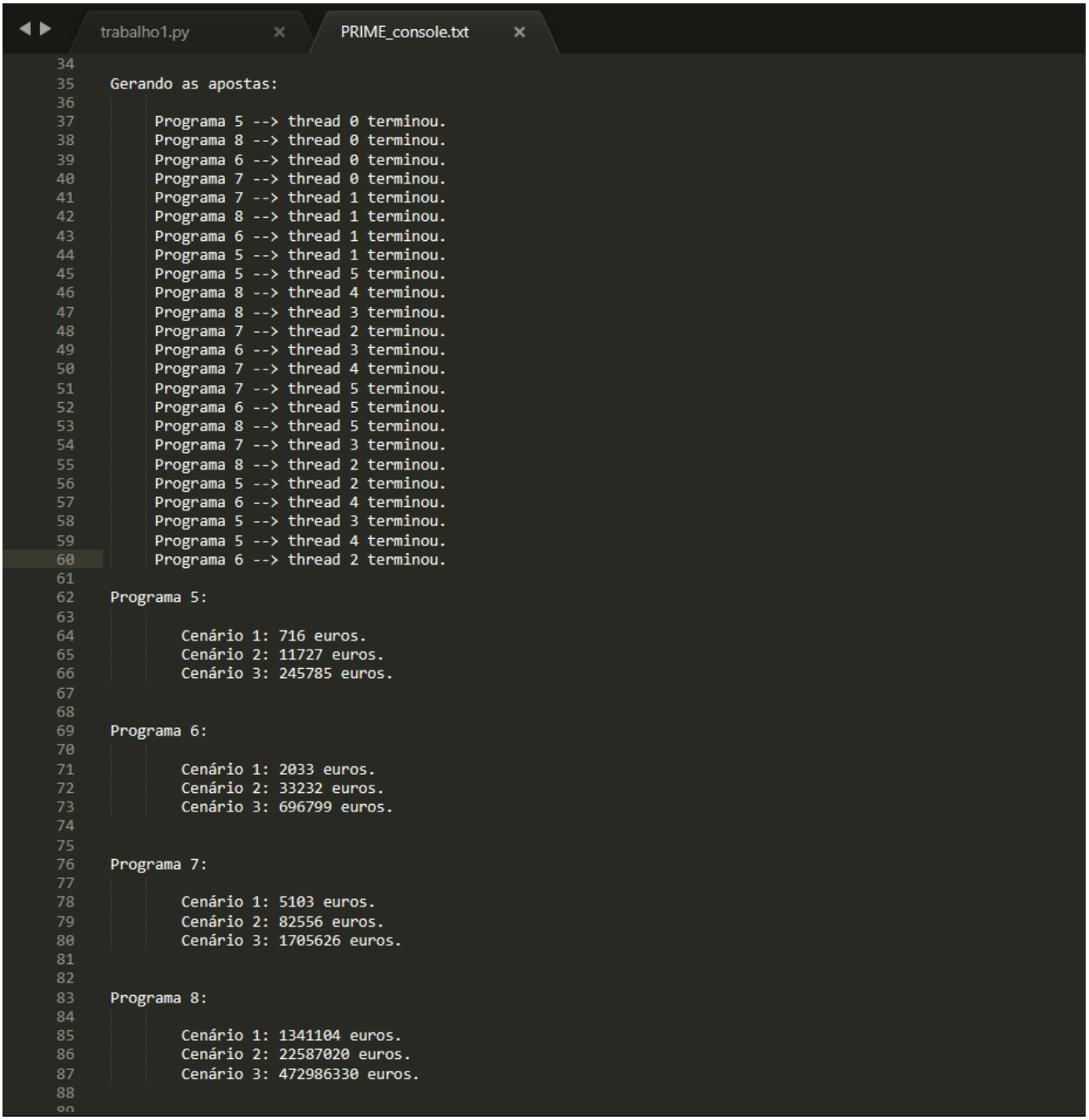
c) se acertou 4 números em um cartão, então a renumeração é 82,31 Euros. Deve-se observar que neste caso a remuneração máxima inclui 1 cartão com 4 números, 4 cartões com 3 números e 6 cartões com 2 números. PROGRAMA 7.

```python
    # C Total -> 33 + n + k * n -> O(k * n)
    """
        c)  se acertou 4 números em um cartão, então a renumeração é 82,31 Euros. Deve-se observar que neste
            caso a remuneração máxima inclui 1 cartão com 4 números, 4 cartões com 3 números e 6 cartões com
            2 números. PROGRAMA 7.
    """
    @decorator_over
    def programa7(self) -> None:

        if self.database == []:                                                                                                 # C1 <- 1

            self.load_database()                                                                                                # C2 <- n

        if not self.load_sceneries():                                                                                           # C3 <- 1

            print("\n\t\tCenários não carregados.\n")                                                                            # C4 <- 1

            return;                                                                                                             # C5 <- 0

        result = {}                                                                                                             # C6 <- 1

        def all_cards(call, scenery):                                                                                           # C7 <- 1

            cards = []                                                                                                          # C8 <- 1

            [cards.append(max([4 if m > 3 else m for m in [self.point_hits(i, db_i) for db_i in self.database]] )) for i in scenery]  # C9 <- k * n

            result[call] = int( ( cards.count(4) + int(cards.count(3)/4) + int(cards.count(2)/6) ) *  82.31 )                   # C10 <- 1

            print('\t Programa 7 --> '+call+' terminou.')                                                                       # C11 <- 1

        data_ = [self.sceneryC1, self.sceneryC2, self.sceneryC3[:15000], self.sceneryC3[15000:30000],
                                                 self.sceneryC3[30000:45000], self.sceneryC3[45000:]]                           # C12 <- 1

        threads = []                                                                                                            # C13 <- 1

        [threads.append(threading.Thread(target=all_cards, args=('thread '+str(i), data_[i],))) for i in range(0, len(data_))]  # C14 <- 6

        [t.start() for t in threads]                                                                                            # C15 <- 6

        [t.join() for t in threads]                                                                                             # C16 <- 6

        res = [result['thread 0'], result['thread 1'], sum(result[i] for i in list(result.keys())[2:])]                         # C17 <- 3

        self.all_gains['Programa 7'] = res                                                                                      # C18 <- 1

        self.apostas['7'] = (f"\nPrograma 7: \n\n\t\tCenário 1: {res[0]} euros.\n\t\tCenário 2: {res[1]} euros.\n\t\tCenário 3: {res[2]} euros.\n") # C19 <- 1
```

d) se acertou 5 números em um cartão, então a renumeração é 70.584,44 Euros. Deve-se observar que neste caso a remuneração máxima inclui 1 cartão com 5 números, 5 cartões com 4 números, 10 cartões com 3 números, 10 cartões com 2 números. PROGRAMA 8.

```
    # C Total -> 33 + n + k * n -> O(k * n)
    """
        d)  se acertou 5 números em um cartão, então a renumeração é 70.584,44 Euros. Deve-se observar
            que neste caso a remuneração máxima inclui 1 cartão com 5 números, 5 cartões com 4 números,
            10 cartões com 3 números, 10 cartões com 2 números. PROGRAMA 8.
    """
    @decorator_over
    def programa8(self) -> None:

        if self.database == []:                                                                                                 # C1 <- 1

            self.load_database()                                                                                                # C2 <- n

        if not self.load_sceneries():                                                                                           # C3 <- 1

            print("\n\t\tCenários não carregados.\n")                                                                            # C4 <- 1

            return;                                                                                                             # C5 <- 0

        result = {}                                                                                                             # C6 <- 1

        def all_cards(call, scenery):                                                                                           # C7 <- 1

            cards = []                                                                                                          # C8 <- 1

            [cards.append(max([ m for m in [self.point_hits(i, db_i) for db_i in self.database]] )) for i in scenery]           # C9 <- k * n

            result[call] = int( ( cards.count(5) + int(cards.count(4)/5) + int(cards.count(3)/10) + int(cards.count(2)/10) ) *  70584.44 )  # C10 <- 1

            print('\t Programa 8 --> '+call+' terminou.')                                                                       # C11 <- 1

        data_ = [self.sceneryC1, self.sceneryC2, self.sceneryC3[:15000], self.sceneryC3[15000:30000],
                                                 self.sceneryC3[30000:45000], self.sceneryC3[45000:]]                           # C12 <- 1

        threads = []                                                                                                            # C13 <- 1

        [threads.append(threading.Thread(target=all_cards, args=('thread '+str(i), data_[i],))) for i in range(0, len(data_))]  # C14 <- 6

        [t.start() for t in threads]                                                                                            # C15 <- 6

        [t.join() for t in threads]                                                                                             # C16 <- 6

        res = [result['thread 0'], result['thread 1'], sum(result[i] for i in list(result.keys())[2:])]                         # C17 <- 3

        self.all_gains['Programa 8'] = res                                                                                      # C18 <- 1

        self.apostas['8'] = (f"\nPrograma 8: \n\n\t\tCenário 1: {res[0]} euros.\n\t\tCenário 2: {res[1]} euros.\n\t\tCenário 3: {res[2]} euros.\n") # C19 <- 1
```

```
    ######################################################
    ######################################################

    if not prog.over:

        print("\n\nGerando as apostas:\n")


    threads_apostas = []

    [threads_apostas.append(threading.Thread(target=p)) for p in [prog.programa5, prog.programa6, prog.programa7, prog.programa8]]


    [thread.start() for thread in threads_apostas]


    [thread.join() for thread in threads_apostas]


    prog.resultado_apostas() # Centavos foram desconsiderados para normalizar o output.

    ######################################################
    ######################################################
```



6. Tomando como exemplo o histórico de prêmios dos últimos anos disponíveis no "euro_concursos.csv", escreva um programa de *backtest* e gere um relatório que mostre os valores investidos e os retornos obtidos. PROGRAMA 9.

Primeiro: função do programa 9.

```
    # C Total -> 22 + 9n + k * n -> O(k * n)
    """
        6.  Tomando como exemplo o histórico de prêmios dos últimos anos disponíveis no "euro_concursos.csv",
            escreva um programa de backtest e gere um relatório que mostre os valores investidos e os
            retornos obtidos.  PROGRAMA 9.
    """
    @decorator_over
    def programa9(self) -> None:

        compare = lambda arr_i:  sorted([[ len(set(i[1:6]) & set(arr_i)), i] for i in self.db_backtest], key=(lambda elem: elem[0]))[-1]           # C1 <- n

        readjust = lambda arr_i, match: [ (i if i in match else chr(248)) for i in arr_i]                                                       # C2 <- n

        winners = lambda scenery, limit: [ [ readjust(i, compare(i)[1][1:6]), compare(i)[1] ] if compare(i)[0] > limit else
                                    [readjust([-1,-1,-1,-1,-1], compare(i)[1][1:6]), compare(i)[1]] for i in scenery if compare(i)[0] > 0]      # C3 <- n

        content = lambda scenery, res: [ res.append("\t\t"+(str(i+1)+":  ").ljust(8)+str(scenery[i][0]).ljust(25)+
                                                                                "   -->  "+str(scenery[i][1])) for i in range(0, len(scenery))]     # C4 <- n

        display = lambda scenery, title, res: [res.append("\n\t\t"+title+"   ('ø': número não hit).\n\n\t\t"+str(self.database_header[0:5])+
                                                                "       ---  "+str(self.db_backtest_header)+"\n\n"), content(scenery, res)]        # C5 <- n

        compare_db_i = lambda scenery, min_points: [ [i[0], len( [k for k in [ int(len(set(i[1:6]) & set(j))/min_points)
                                        for j in scenery] if k > 0])] for i in self.db_backtest] # [Card number, number of hits].      # C6 <- k * n

        db_display = lambda scenery, res: [ [res.append("CON\t\t\t"),
                                        [res.append('CON[No:'+(str(i[0])+',').ljust(5)+' hits='+(str(i[1])+']').ljust(13)) for i in scenery[i:i+5]],
                                        res.append("")] for i in range(0, len(scenery), 10)]                                                    # C7 <- n

        if len(self.all_gains) < 4:                                                                                                             # C8 <- 1

            self.all_gains = self.loader('last_gains/all_gains.pkl')                                                                            # C9 <- 1

        invested = lambda scenery: [str(k)+" -> "+str(len(scenery))+" cartões." for k in self.all_gains.keys()]                                 # C10 <- 1
```

```
        if len(self.all_gains) < 4:                                                                                                             # C8 <- 1

            self.all_gains = self.loader('last_gains/all_gains.pkl')                                                                            # C9 <- 1

        invested = lambda scenery: [str(k)+" -> "+str(len(scenery))+" cartões." for k in self.all_gains.keys()]                                 # C10 <- 1

        lucro = lambda _0_to_2: [str(k)+" -> "+str(self.all_gains[k][_0_to_2])+" euros." for k in self.all_gains.keys()]                        # C11 <- 1

        op = lambda scenery: 0 if scenery == self.sceneryC1 else ( 1 if scenery == self.sceneryC2 else 2)                                       # C12 <- 1

        inv_lucro = lambda scenery: [ ["Investidos:  "+invested(scenery)[i], "Remuneração: "+lucro(op(scenery))[i]]
                                                                        for i in range(0, len(self.all_gains))]                                 # C13 <- 1

        scenery_relatory = lambda scenery, res: [res.append("\n\t\tRelatório (Cenário "+str(op(scenery)+1)+"):\n\n\t\t\t\t\t"),
                                    (res.append("CON\t\t\t\t"+"\n\n\t\t\t\t\t".join(["\n\t\t\t\t".join(i) for i in inv_lucro(scenery)])+"\n\n"))]    # C14 <- 1

        control = lambda scenery, res: [display(winners(scenery, op(scenery)+1), ("Cenário "+str(op(scenery)+1)+":"), res),
                        res.append(f"\n\n\t\tTodos os hits do cenário {str(op(scenery)+1)} que as entradas(No) em db_backtest receberam:\n"),
                                                    db_display(compare_db_i(scenery, 2), res), scenery_relatory(scenery, res)   ]              # C15 <- 1

        resultC1, resultC2, resultC3 = [], [], []                                                                                               # C16 <- 3

        print("\n\nRelatório backtest:")                                                                                                         # C17 <- 1

        threads = []                                                                                                                            # C18 <- 1

        [threads.append(threading.Thread(target=control, args=( i[0], i[1], ) )) for i in [
                                                [self.sceneryC1, resultC1], [self.sceneryC2, resultC2], [self.sceneryC3, resultC3]] ]           # C19 <- 3

        [t.start() for t in threads]                                                                                                            # C20 <- 3

        threads[0].join()                                                                                                                       # C21 <- 1

        [print(line) if line[0:3]!="CON" else print(line[3:], end='') for line in resultC1]                                                     # C22 <- n

        threads[1].join()                                                                                                                       # C23 <- 1

        [print(line) if line[0:3]!="CON" else print(line[3:], end='') for line in resultC2]                                                     # C24 <- n

        threads[2].join()                                                                                                                       # C25 <- 1

        [print(line) if line[0:3]!="CON" else print(line[3:], end='') for line in resultC3]                                                     # C26 <- n
```

```
    ######################################################
    ######################################################


    prog.programa9() # Backtest.


    ######################################################
    ######################################################
```

Segundo: Output do programa 9 (relatório).

- Detalhe: o relatório gerado possui ~ 62.000 linhas. Para cada cenário, o relatório define três blocos: (1) a lista dos cartões apostados e, ao seu lado, o cartão da base euro_concursoos.csv que teve mais acertos para o cartão

apostado (i), ø significa que o número no cartão apostado i não teve um hit no cartão com mais hits da base .csv em relação a (i); (2) contém todos os cartões da base (representados por 'No:' e seu número) e o número de cartões apostados que tiveram 2 hits ou mais; e (3) o total de cartões investidos no cenário X com a remuneração para as regras de contagem de cartões e remuneração nos programas 5, 6, 7 e 8 – basicamente, o que aconteceria se o apostador trocasse seus cartões seguindo as regras de um dos programas para obter a melhor remuneração.

-- Detalhe 2: como os cenários 1, 2 e 3 seguem a mesma lógica de output, vou mostrar apenas os blocos 1, 2 e 3 do cenário 1 para efeitos explicativos. Os resultados completos dos cenários estão salvos no arquivo txt: PRIME_console.txt no folder console.

Bloco (1):

```
Relatório backtest:

        Cenário 1:   ('ø': número não hit).

        ['B1', 'B2', 'B3', 'B4', 'B5']      ---  ['No', 'B1', 'B2', 'B3', 'B4', 'B5', 'LS1', 'LS2', 'Jackpot', 'Wins']


        1:      [1, 2, 'ø', 'ø', 5]         -->  [1343, 1, 2, 5, 15, 42, 5, 8, 44373299, 0]
        2:      [1, 6, 'ø', 'ø', 9]         -->  [1199, 1, 6, 9, 34, 47, 7, 12, 14552000, 0]
        3:      [1, 10, 'ø', 12, 'ø']       -->  [1332, 1, 10, 12, 21, 26, 5, 11, 46859882, 0]
        4:      [1, 'ø', 15, 16, 'ø']       -->  [1184, 1, 6, 13, 15, 16, 6, 10, 68153942, 0]
        5:      [1, 'ø', 19, 'ø', 21]       -->  [1314, 1, 5, 19, 21, 28, 4, 7, 22310697, 0]
        6:      [1, 22, 23, 24, 'ø']        -->  [650, 1, 22, 23, 24, 31, 6, 11, 37664283, 0]
        7:      ['ø', 26, 27, 28, 'ø']      -->  [959, 21, 26, 27, 28, 38, 9, 11, 14494200, 0]
        8:      [1, 30, 31, 'ø', 'ø']       -->  [1224, 1, 13, 30, 31, 39, 2, 7, 15152100, 0]
        9:      [1, 'ø', 35, 36, 'ø']       -->  [1206, 1, 28, 35, 36, 46, 6, 9, 14701600, 0]
        10:     ['ø', 'ø', 39, 40, 41]      -->  [1059, 14, 16, 39, 40, 41, 8, 10, 34502478, 0]
        11:     ['ø', 42, 43, 'ø', 45]      -->  [1354, 34, 38, 42, 43, 45, 9, 12, 37525275, 0]
        12:     [1, 'ø', 47, 48, 'ø']       -->  [1156, 1, 3, 29, 47, 48, 3, 12, 43581440, 0]
        13:     [2, 'ø', 10, 'ø', 18]       -->  [1389, 2, 10, 18, 34, 35, 6, 9, 37176013, 0]
        14:     [2, 'ø', 'ø', 15, 19]       -->  [1271, 2, 3, 15, 19, 34, 5, 8, 22196254, 0]
        15:     [2, 8, 'ø', 16, 'ø']        -->  [1376, 2, 5, 8, 14, 16, 8, 9, 146729731, 0]
        16:     [2, 'ø', 13, 17, 21]        -->  [1382, 2, 3, 13, 17, 21, 7, 8, 28785119, 0]
        17:     ['ø', 22, 'ø', 30, 34]      -->  [1066, 4, 22, 30, 32, 34, 3, 4, 94304641, 0]
        18:     ['ø', 'ø', 27, 31, 35]      -->  [1391, 25, 27, 31, 35, 43, 5, 6, 60310182, 0]
        19:     [2, 'ø', 28, 'ø', 36]       -->  [928, 2, 15, 28, 36, 50, 2, 11, 33820376, 0]
        20:     ['ø', 25, 29, 33, 'ø']      -->  [1181, 22, 25, 29, 33, 35, 3, 6, 42137452, 0]
        21:     [2, 'ø', 42, 'ø', 50]       -->  [1383, 2, 10, 16, 42, 50, 4, 11, 39844969, 1]
        22:     [3, 'ø', 11, 'ø', 21]       -->  [1302, 3, 11, 21, 27, 36, 9, 10, 50321191, 0]
        23:     [3, 'ø', 10, 17, 'ø']       -->  [1310, 3, 10, 17, 33, 39, 1, 4, 46937085, 0]
        24:     ['ø', 8, 13, 14, 'ø']       -->  [1345, 8, 13, 14, 42, 50, 3, 9, 66130274, 0]
        25:     ['ø', 'ø', 12, 15, 18]      -->  [812, 11, 12, 15, 18, 44, 3, 9, 17408173, 0]
        26:     [3, 22, 27, 32, 'ø']        -->  [1322, 3, 18, 22, 27, 32, 2, 7, 25387006, 0]
        27:     [3, 23, 26, 'ø', 'ø']       -->  [1168, 3, 10, 12, 23, 26, 1, 12, 43374190, 0]
        28:     [3, 'ø', 29, 'ø', 35]       -->  [1193, 3, 15, 29, 35, 47, 3, 4, 22256333, 0]
        29:     [3, 'ø', 28, 31, 34]        -->  [1259, 3, 28, 31, 32, 34, 4, 5, 22772993, 0]
        30:     ['ø', 39, 'ø', 47, 50]      -->  [1020, 17, 35, 39, 47, 50, 6, 8, 87570000, 1]
        31:     ['ø', 6, 12, 17, 'ø']       -->  [1347, 6, 12, 17, 34, 42, 5, 12, 87266444, 0]
        32:     [4, 7, 13, 'ø', 'ø']        -->  [1171, 4, 7, 13, 23, 42, 1, 3, 69712546, 0]
        33:     [4, 8, 10, 'ø', 21]         -->  [71, 4, 8, 10, 18, 21, 1, 7, 28094862, 0]
        34:     [4, 9, 11, 'ø', 'ø']        -->  [1329, 4, 9, 11, 17, 39, 2, 10, 15245600, 0]
        35:     ['ø', 22, 'ø', 33, 35]      -->  [1181, 22, 25, 29, 33, 35, 3, 6, 42137452, 0]
        36:     ['ø', 23, 'ø', 32, 34]      -->  [1320, 11, 23, 32, 34, 39, 1, 6, 65175962, 1]
        37:     ['ø', 24, 'ø', 31, 37]      -->  [1397, 5, 10, 24, 31, 37, 9, 10, 127347464, 0]
        38:     [4, 'ø', 27, 30, 'ø']       -->  [1052, 4, 17, 23, 27, 30, 3, 8, 49712130, 0]
        39:     [4, 'ø', 'ø', 48, 50]       -->  [1129, 4, 6, 27, 48, 50, 1, 11, 15126600, 0]
        40:     [5, 6, 'ø', 15, 'ø']        -->  [1365, 5, 6, 15, 37, 42, 3, 4, 59778232, 0]
        41:     [5, 7, 'ø', 14, 21]         -->  [175, 5, 7, 14, 21, 40, 3, 8, 18315755, 1]
        42:     [5, 'ø', 11, 17, 'ø']       -->  [1326, 5, 11, 17, 24, 37, 3, 6, 25999539, 0]
```

Bloco (2):

```
Todos os hits do cenário 1 que as entradas(No) em db_backtest receberam:

    [No:1,    hits=12]        [No:2,    hits=9]         [No:3,    hits=19]        [No:4,    hits=10]        [No:5,    hits=10]
    [No:11,   hits=10]        [No:12,   hits=12]        [No:13,   hits=18]        [No:14,   hits=13]        [No:15,   hits=10]
    [No:21,   hits=9]         [No:22,   hits=11]        [No:23,   hits=9]         [No:24,   hits=13]        [No:25,   hits=17]
    [No:31,   hits=11]        [No:32,   hits=10]        [No:33,   hits=12]        [No:34,   hits=15]        [No:35,   hits=10]
    [No:41,   hits=14]        [No:42,   hits=15]        [No:43,   hits=13]        [No:44,   hits=12]        [No:45,   hits=11]
    [No:51,   hits=13]        [No:52,   hits=11]        [No:53,   hits=11]        [No:54,   hits=14]        [No:55,   hits=10]
    [No:61,   hits=11]        [No:62,   hits=11]        [No:63,   hits=10]        [No:64,   hits=13]        [No:65,   hits=12]
    [No:71,   hits=7]         [No:72,   hits=21]        [No:73,   hits=13]        [No:74,   hits=15]        [No:75,   hits=12]
    [No:81,   hits=11]        [No:82,   hits=11]        [No:83,   hits=10]        [No:84,   hits=25]        [No:85,   hits=15]
    [No:91,   hits=14]        [No:92,   hits=19]        [No:93,   hits=13]        [No:94,   hits=13]        [No:95,   hits=11]
    [No:101,  hits=9]         [No:102,  hits=10]        [No:103,  hits=10]        [No:104,  hits=12]        [No:105,  hits=10]
    [No:111,  hits=9]         [No:112,  hits=11]        [No:113,  hits=20]        [No:114,  hits=14]        [No:115,  hits=16]
    [No:121,  hits=11]        [No:122,  hits=8]         [No:123,  hits=14]        [No:124,  hits=11]        [No:125,  hits=19]
    [No:131,  hits=18]        [No:132,  hits=11]        [No:133,  hits=13]        [No:134,  hits=11]        [No:135,  hits=16]
    [No:141,  hits=10]        [No:142,  hits=11]        [No:143,  hits=10]        [No:144,  hits=13]        [No:145,  hits=19]
    [No:151,  hits=23]        [No:152,  hits=13]        [No:153,  hits=7]         [No:154,  hits=8]         [No:155,  hits=11]
    [No:161,  hits=15]        [No:162,  hits=10]        [No:163,  hits=9]         [No:164,  hits=13]        [No:165,  hits=10]
    [No:171,  hits=12]        [No:172,  hits=11]        [No:173,  hits=10]        [No:174,  hits=18]        [No:175,  hits=5]
    [No:181,  hits=12]        [No:182,  hits=10]        [No:183,  hits=8]         [No:184,  hits=12]        [No:185,  hits=13]
    [No:191,  hits=12]        [No:192,  hits=17]        [No:193,  hits=14]        [No:194,  hits=12]        [No:195,  hits=23]
    [No:201,  hits=8]         [No:202,  hits=10]        [No:203,  hits=11]        [No:204,  hits=11]        [No:205,  hits=9]
    [No:211,  hits=10]        [No:212,  hits=10]        [No:213,  hits=12]        [No:214,  hits=10]        [No:215,  hits=11]
    [No:221,  hits=14]        [No:222,  hits=12]        [No:223,  hits=10]        [No:224,  hits=11]        [No:225,  hits=8]
    [No:231,  hits=21]        [No:232,  hits=13]        [No:233,  hits=11]        [No:234,  hits=9]         [No:235,  hits=24]
    [No:241,  hits=23]        [No:242,  hits=13]        [No:243,  hits=10]        [No:244,  hits=11]        [No:245,  hits=11]
    [No:251,  hits=10]        [No:252,  hits=8]         [No:253,  hits=12]        [No:254,  hits=9]         [No:255,  hits=18]
    [No:261,  hits=10]        [No:262,  hits=9]         [No:263,  hits=22]        [No:264,  hits=12]        [No:265,  hits=13]
    [No:271,  hits=10]        [No:272,  hits=11]        [No:273,  hits=10]        [No:274,  hits=7]         [No:275,  hits=21]
    [No:281,  hits=9]         [No:282,  hits=14]        [No:283,  hits=10]        [No:284,  hits=11]        [No:285,  hits=13]
    [No:291,  hits=11]        [No:292,  hits=11]        [No:293,  hits=10]        [No:294,  hits=15]        [No:295,  hits=11]
    [No:301,  hits=10]        [No:302,  hits=10]        [No:303,  hits=9]         [No:304,  hits=19]        [No:305,  hits=13]
    [No:311,  hits=9]         [No:312,  hits=11]        [No:313,  hits=14]        [No:314,  hits=11]        [No:315,  hits=10]
    [No:321,  hits=10]        [No:322,  hits=8]         [No:323,  hits=16]        [No:324,  hits=14]        [No:325,  hits=8]
    [No:331,  hits=12]        [No:332,  hits=16]        [No:333,  hits=11]        [No:334,  hits=18]        [No:335,  hits=11]
    [No:341,  hits=10]        [No:342,  hits=13]        [No:343,  hits=11]        [No:344,  hits=11]        [No:345,  hits=10]
    [No:351,  hits=11]        [No:352,  hits=10]        [No:353,  hits=8]         [No:354,  hits=11]        [No:355,  hits=9]
    [No:361,  hits=13]        [No:362,  hits=14]        [No:363,  hits=9]         [No:364,  hits=8]         [No:365,  hits=12]
    [No:371,  hits=11]        [No:372,  hits=13]        [No:373,  hits=10]        [No:374,  hits=9]         [No:375,  hits=11]
    [No:381,  hits=16]        [No:382,  hits=9]         [No:383,  hits=12]        [No:384,  hits=12]        [No:385,  hits=22]
    [No:391,  hits=9]         [No:392,  hits=10]        [No:393,  hits=19]        [No:394,  hits=8]         [No:395,  hits=13]
    [No:401,  hits=10]        [No:402,  hits=10]        [No:403,  hits=9]         [No:404,  hits=18]        [No:405,  hits=15]
    [No:411,  hits=13]        [No:412,  hits=9]         [No:413,  hits=12]        [No:414,  hits=10]        [No:415,  hits=14]
```

Bloco (3):

```
    [No:1341, hits=19]        [No:1342, hits=10]        [No:1343, hits=20]        [No:1344, hits=15]        [No:1345, hits=17]
    [No:1351, hits=10]        [No:1352, hits=10]        [No:1353, hits=14]        [No:1354, hits=13]        [No:1355, hits=10]
    [No:1361, hits=10]        [No:1362, hits=11]        [No:1363, hits=10]        [No:1364, hits=10]        [No:1365, hits=8]
    [No:1371, hits=11]        [No:1372, hits=12]        [No:1373, hits=10]        [No:1374, hits=12]        [No:1375, hits=11]
    [No:1381, hits=10]        [No:1382, hits=10]        [No:1383, hits=11]        [No:1384, hits=14]        [No:1385, hits=10]
    [No:1391, hits=10]        [No:1392, hits=8]         [No:1393, hits=8]         [No:1394, hits=12]        [No:1395, hits=16]
    [No:1401, hits=10]        [No:1402, hits=12]        [No:1403, hits=13]        [No:1404, hits=9]

Relatório (Cenário 1):


        Investidos:  Programa 7 -> 171 cartões.
        Remuneração: Programa 7 -> 5103 euros.

        Investidos:  Programa 8 -> 171 cartões.
        Remuneração: Programa 8 -> 1341104 euros.

        Investidos:  Programa 5 -> 171 cartões.
        Remuneração: Programa 5 -> 716 euros.

        Investidos:  Programa 6 -> 171 cartões.
        Remuneração: Programa 6 -> 2033 euros.
```

7. Como foco na disciplina é análise de complexidade de algoritmos, pede-se como último item do trabalho a realização da análise de complexidade de tempo para cada um dos seguintes itens: PROGRAMA 1, PROGRAMA 2, PROGRAMA 3, PROGRAMA 4, PROGRAMA 5, PROGRAMA 6, PROGRAMA 7, PROGRAMA 8, PROGRAMA 9.

Primeiro: o algoritmo que faz a análise temporal para todos os programas acima.

```python
    # C Total -> 5 + 12n + 5*n*r -> (r < n) -> O(k * n)
    """
        7.  Como foco na disciplina é análise de complexidade de algoritmos, pede-se como último item do trabalho
            a realização da análise de complexidade de tempo para cada um dos seguintes itens: PROGRAMA 1, PROGRAMA 2,
            PROGRAMA 3, PROGRAMA 4, PROGRAMA 5, PROGRAMA 6, PROGRAMA 7, PROGRAMA 8, PROGRAMA 9.
    """
    @decorator_over
    def time_analysis(self, funcs=[], repeat=0, call_name='', show_fig=False) -> None:

        def output(x):                                                                          # C1 <- 3
            sys.stdout = last                                                                   # C1.1 <- 1
            print(x, end='')                                                                    # C1.2 <- 1
            sys.stdout = open('console/hidden.txt', 'w')                                        # C1.3 <- 1

        print("\nTime Analysis:\n\n")                                                           # C2 <- 1

        last = sys.stdout                                                                       # C3 <- 1

        sys.stdout = open('console/hidden.txt', 'w')                                            # C4 <- 1

        for f in funcs:                                                                         # C5 <- n

            time_arr = []                                                                       # C6 <- n

            output("\t"+str(f.__name__)+"(): [repeat==5 => ")                                   # C7 <- n*3

            for r in range(0, repeat):                                                          # C8 <- n*r

                output(f"{r} ")                                                                 # C9 <- n*r*3

                track = time.time()                                                             # C10 <- n*r

                f()                                                                             # C11 <- n*r

                time_arr.append(time.time() - track)                                            # C12 <- n*r

            output(f"]. Average Time: {sum(time_arr)/repeat} seconds.\n")                       # C13 <- n*3

            sys.stdout = last                                                                   # C14 <- n

            plt.plot(time_arr, label=''+f.__name__)                                             # C15 <- n
            plt.title("When f() == "+f.__name__)                                                # C16 <- n
            plt.legend(loc='best')                                                              # C17 <- n
            plt.savefig("time_analysis/"+f.__name__+"_fig_"+call_name+".png")                   # C18 <- n

            if show_fig:                                                                        # C19 <- n
                plt.show()                                                                      # C20 <- n

            sys.stdout = open('console/hidden.txt', 'w')                                        # C21 <- n

        sys.stdout = last                                                                       # C22 <- 1

        print("\n")                                                                             # C23 <- 1
```

```python


    ####################################################
    ####################################################

    funcs = [prog.programa1, prog.programa2, prog.programa3, prog.programa4, prog.programa5, prog.programa6, prog.programa7, prog.programa8, prog.programa9]


    prog.time_analysis(funcs=funcs, repeat=5, show_fig=False) # Testando a complexidade temporal de cada função.

    ####################################################
    ####################################################


    # Você pode deixar de executar uma função em específico usando essas instruções.
    #
    # prog.over = True
    #
    #   func()
    #
    # prog.over = False


```

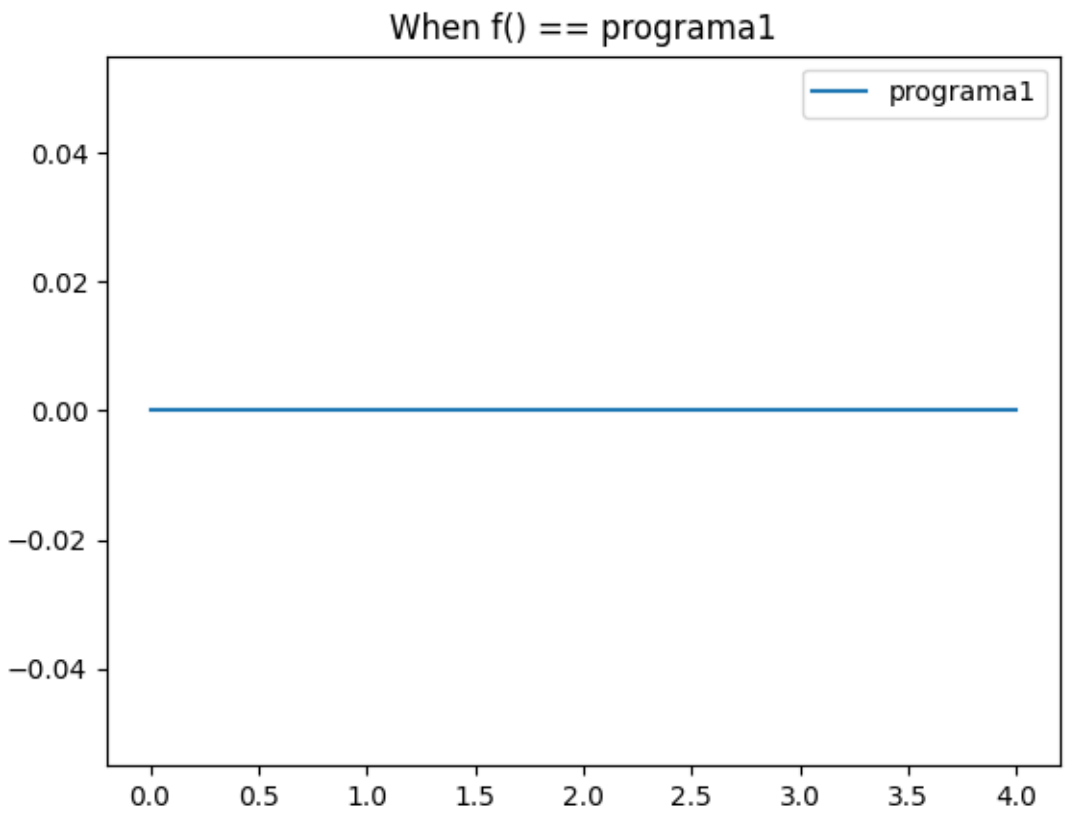
Segundo: o output do algoritmo.

```
62223
62224 ▼ Time Analysis:
62225
62226
62227       programa1(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 0.0 seconds.
62228       programa2(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 71.7330518245697 seconds.
62229       programa3(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 71.5942967891693 seconds.
62230       programa4(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 51.054732704162596 seconds.
62231       programa5(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 106.80825762748718 seconds.
62232       programa6(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 106.95024647712708 seconds.
62233       programa7(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 107.39654083251953 seconds.
62234       programa8(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 104.65906548500061 seconds.
62235       programa9(): [repeat==5 => 0 1 2 3 4 ]. Average Time: 554.3489911079407 seconds.
62236
62237
62238   [Finished in 7035.8s]
```

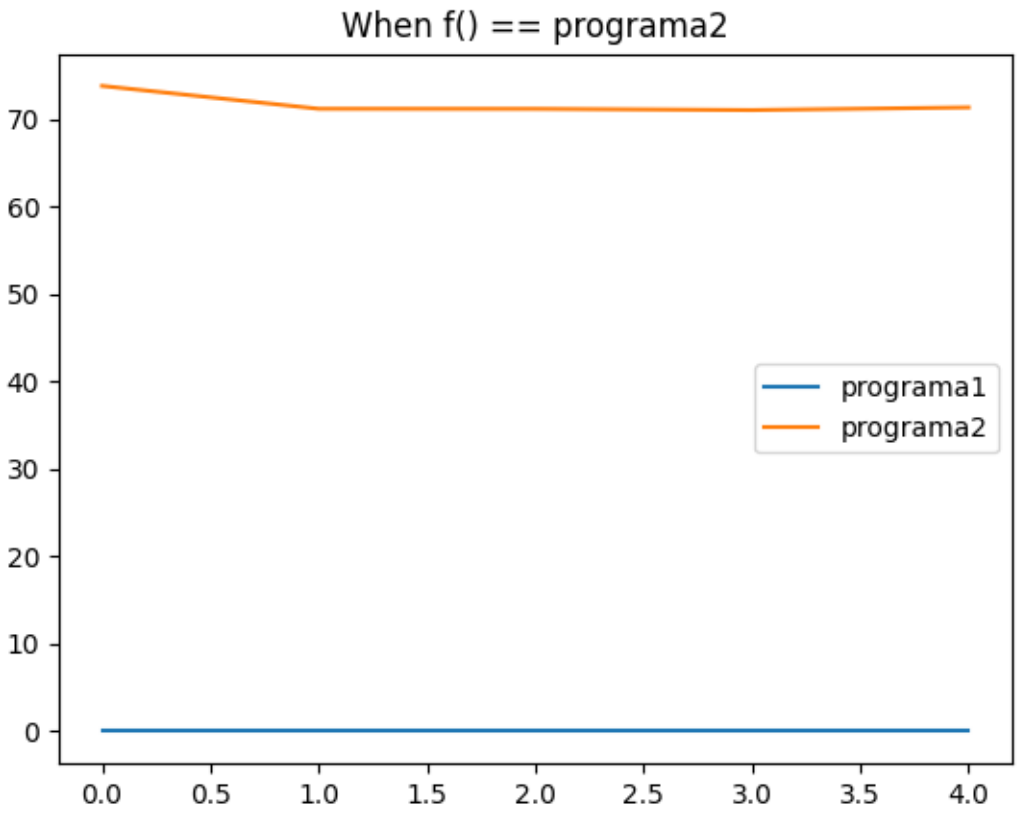
São ~ 62.000 linhas mesmo, haha.

Terceiro: os plots para cada programa de 1 a 9 de suas execuções.

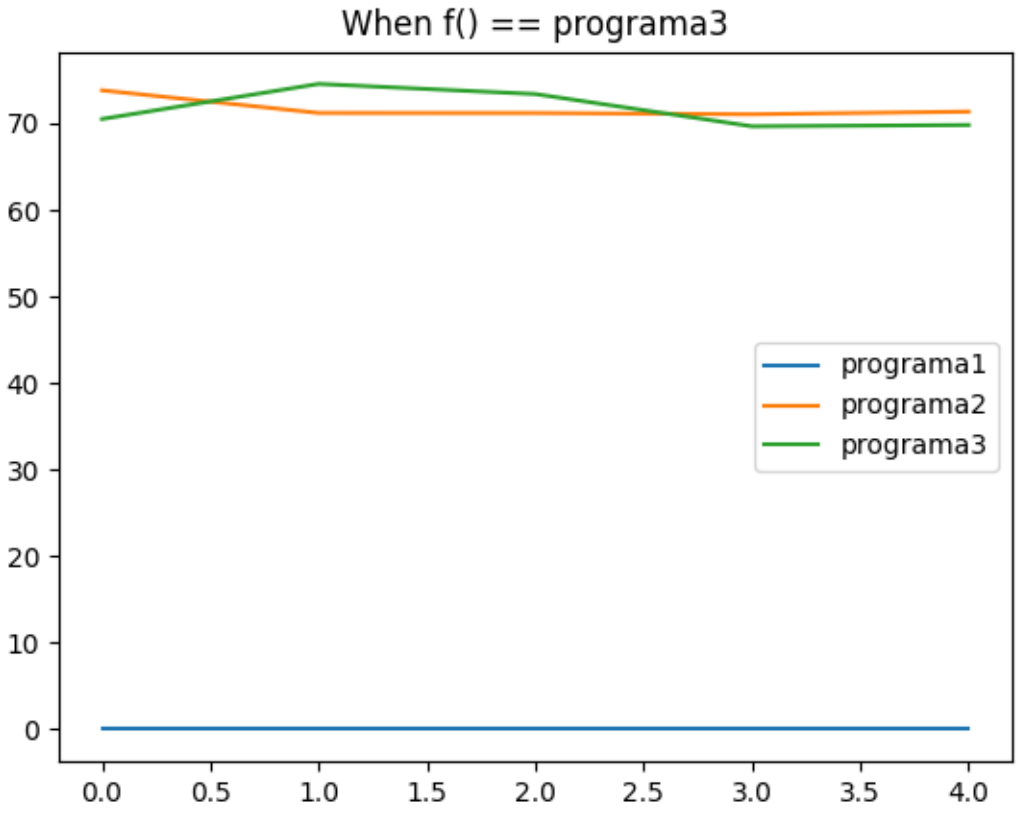
- Programa 1:



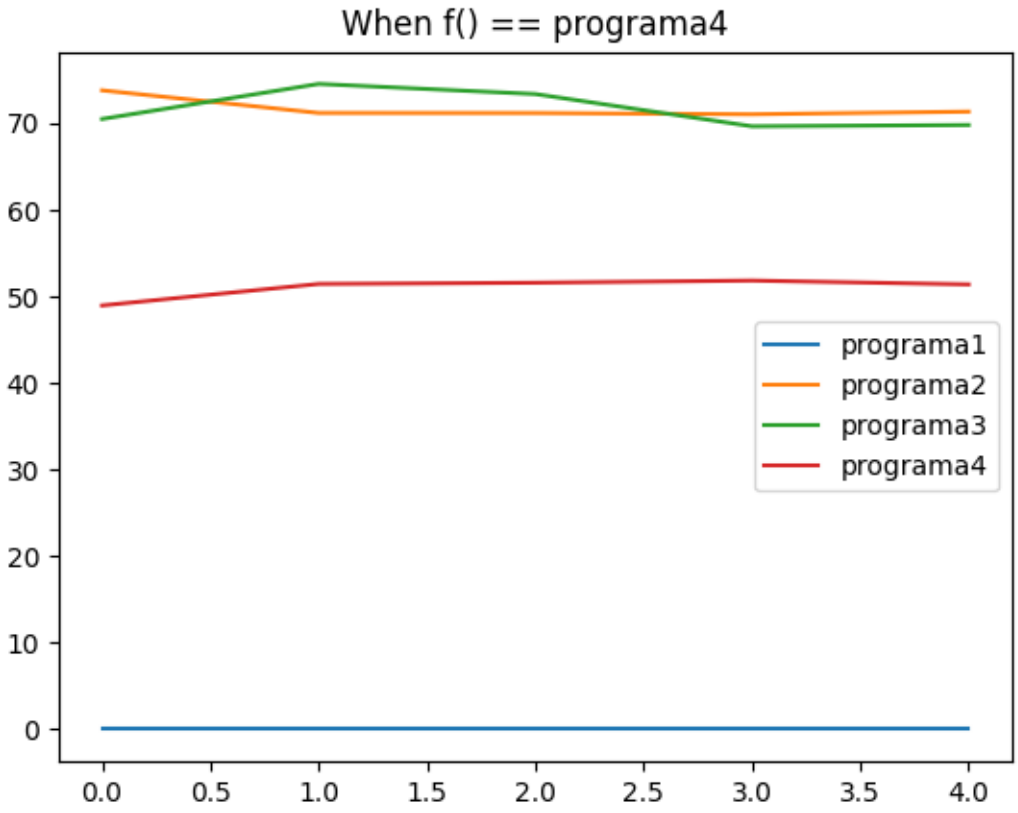
- Programa 2:


When f() == programa2
70
60
50
40
30
20
10
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
programa1
programa2

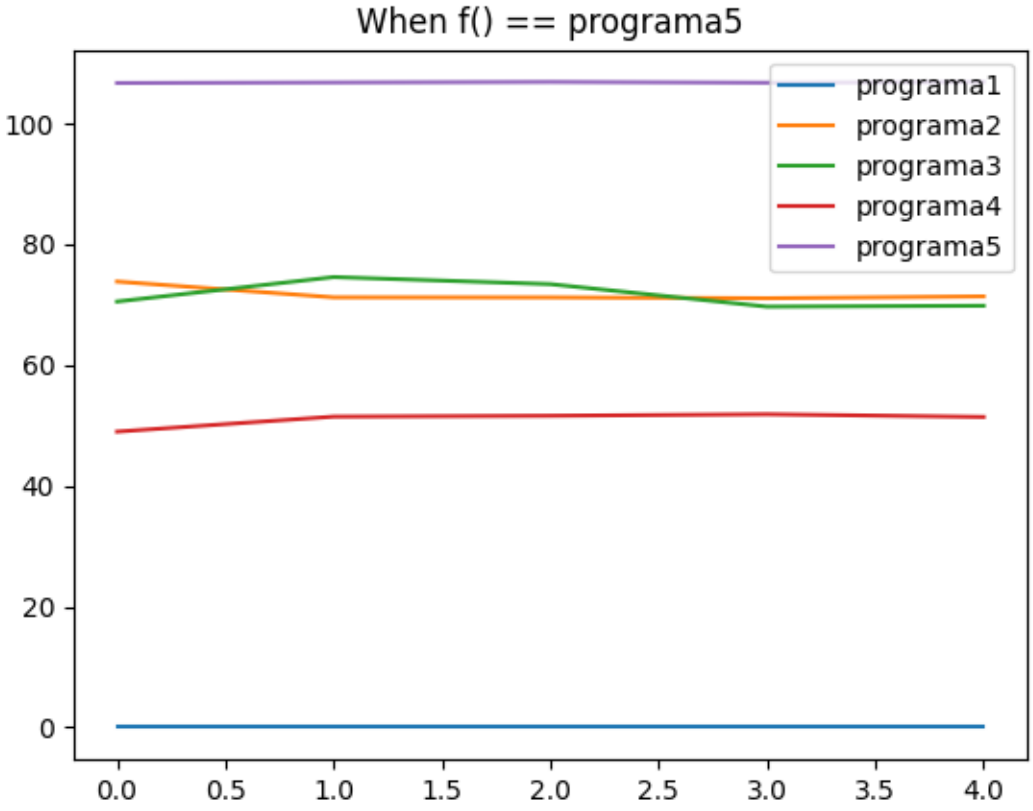
- Programa 3:


When f() == programa3
70
60
50
40
30
20
10
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
programa1
programa2
programa3

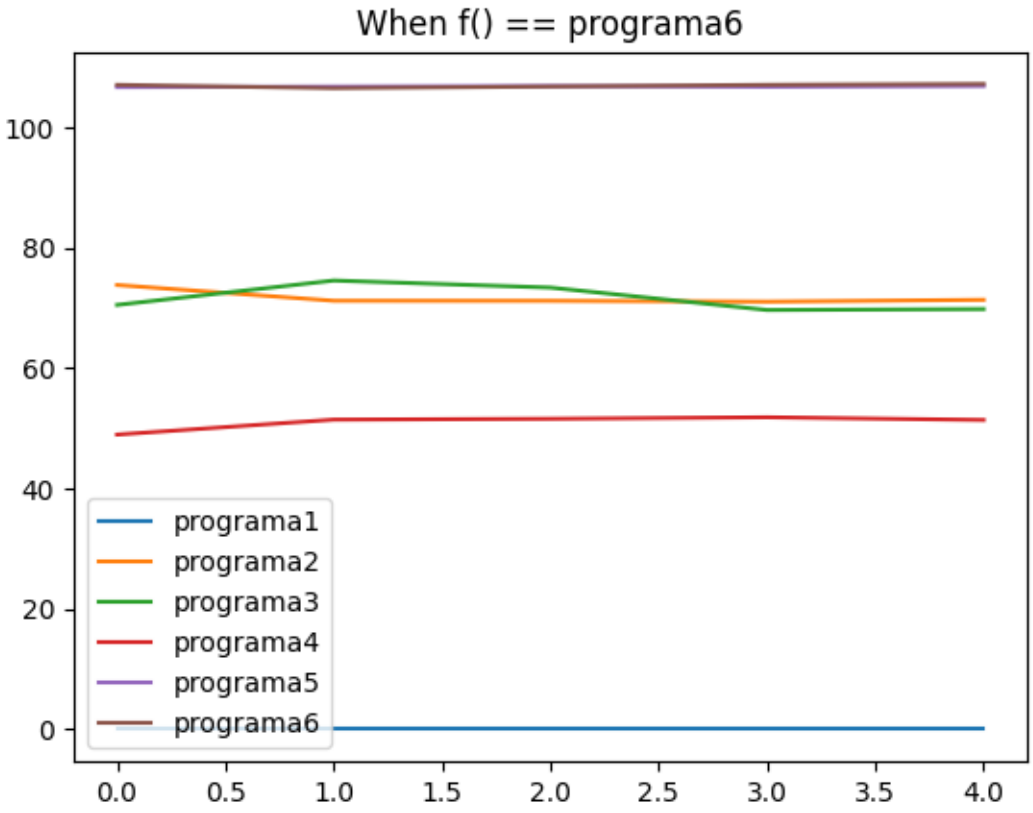
- Programa 4:


When f() == programa4
70
60
50
40
30
20
10
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
programa1
programa2
programa3
programa4

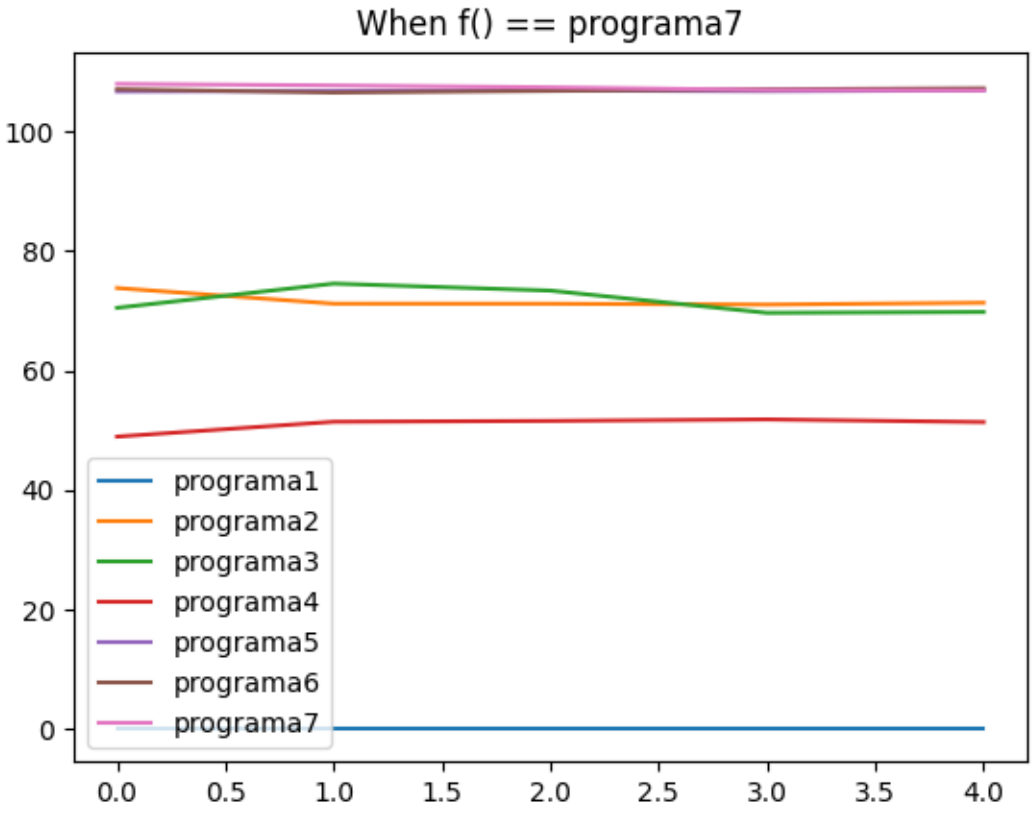
- Programa 5:


When f() == programa5
programa1
programa2
programa3
programa4
programa5
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

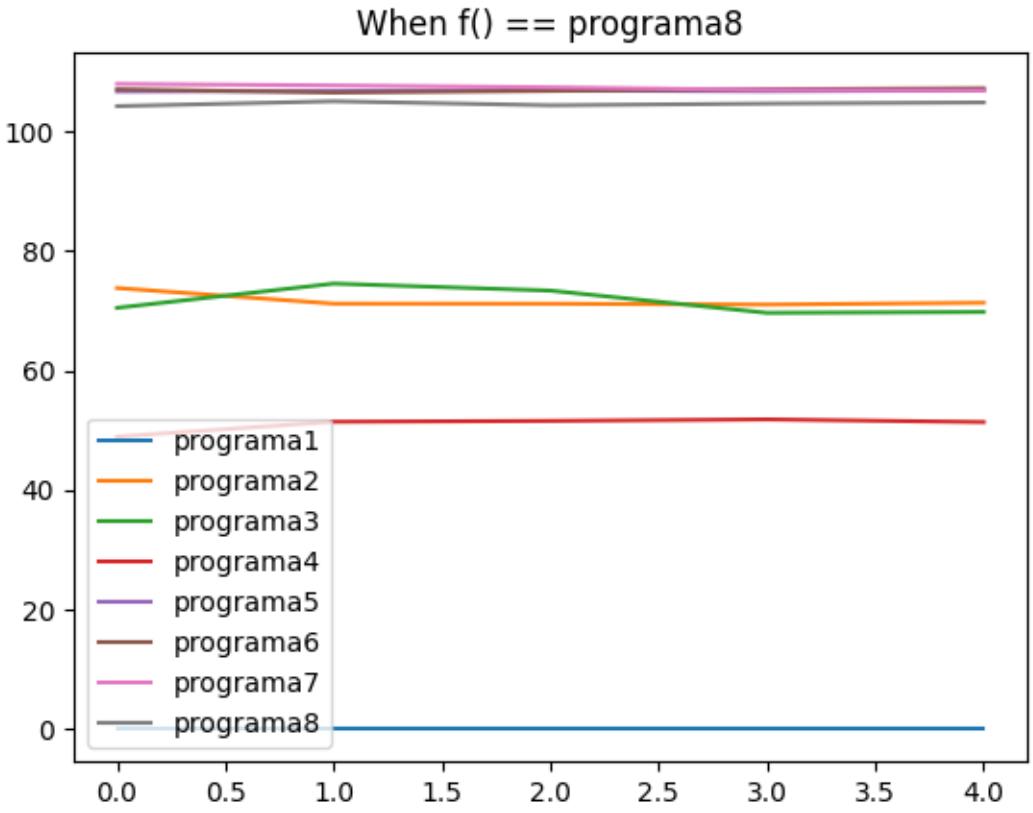
- Programa 6:


When f() == programa6
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
programa1
programa2
programa3
programa4
programa5
programa6

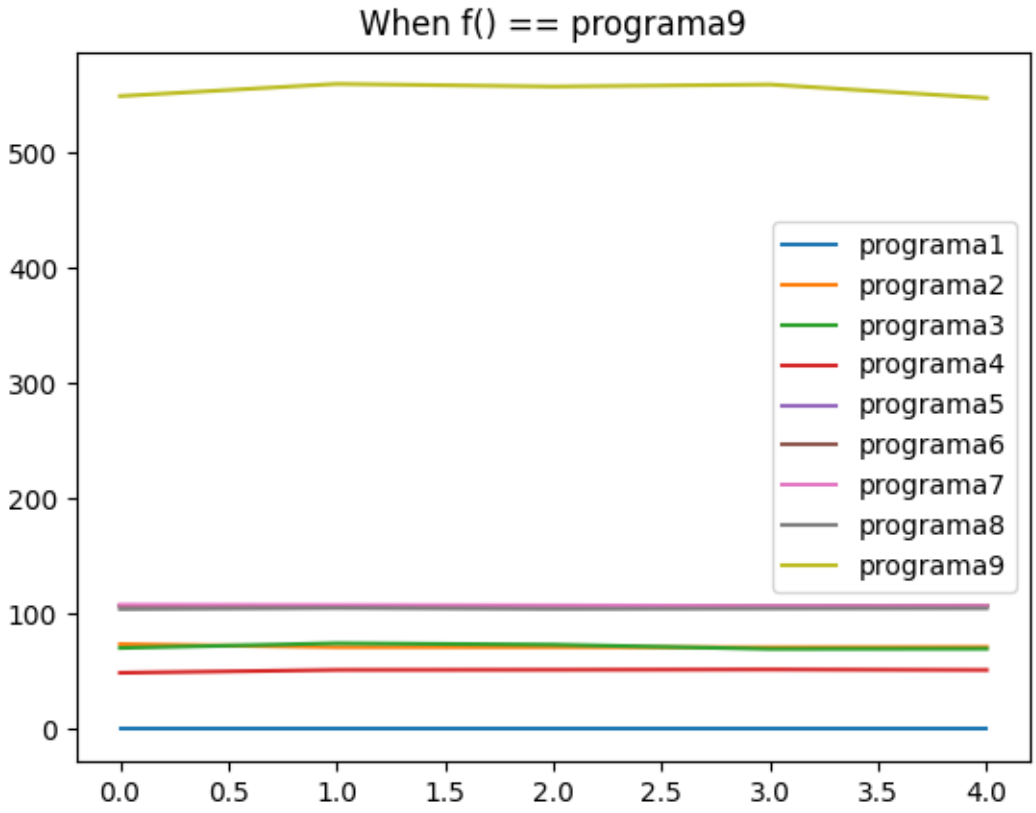
- Programa 7:


When f() == programa7
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
programa1
programa2
programa3
programa4
programa5
programa6
programa7

- Programa 8:


When f() == programa8
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
programa1
programa2
programa3
programa4
programa5
programa6
programa7
programa8

- Programa 9:



O programa leva, em média, 16 minutos para executar por completo, à exceção da função time_analysis(), que executa cada programa 5 vezes, aumentando o tempo total para, aproximadamente, 1 hora e 40 minutos.

Fiz algumas funções e tricks para executar partes do código à sua escolha. Veja o arquivo.py, deixei tudo comentado, etapa a etapa.

Veja o vídeo da execução.

A tabela abaixo mostra uma parte do arquivo de teste ("euro_concursos.csv") com o histórico dos concursos. As colunas **B1, B2, B3, B4** e **B5** são os números sorteados em cada concurso. A

coluna **Jackpot** é o valor do prêmios do concurso e a coluna **Wins** é o número de ganhadores. As colunas **LS1** e **LS2** são números de estrelas do "euro-millions", não consideradas no processo de geração de combinações.

No	B1	B2	B3	B4	B5	LS1	LS2	Jackpot	Wins
1404	4	5	39	46	48	7	10	27.914.329,00	0
1403	2	19	24	26	49	6	7	14.728.800,00	0
1402	6	12	22	29	33	6	11	182.028.000,00	1
1401	9	18	30	39	45	1	3	180.663.000,00	0
...	...	...	...	...	...	...	...	...	....
0004	4	7	33	37	39	1	5	13.870.849,00	1
0003	14	18	19	31	37	4	5	11.880.304,00	0
0002	7	13	39	47	50	2	5	10.111.500,00	0
0001	16	29	32	36	41	7	9	10.143.000,00	1