

## Relatório do projeto “Trabalho final” de Estruturas Discretas de Computação

Relativamente o 1º ponto do trabalho “Implemente uma função denominada: **ou\_exclusivo** ( $p, q$ )  $\equiv (p \oplus q)$ ”, criámos a função conforme pedido, em que dentro da mesma estão as condições relativas ao XOR. Se “p” ou “q” for “1” e o outro for “0” o resultado é “1”. Caso contrário, o resultado é “0”. Colocámos os input’s do utilizador fora da função.

```
#1. Implemente uma função denominada: ou_exclusivo (p, q)  $\equiv (p \oplus q)$ 

def ou_exclusivo(p,q):
    if (p==1 and q==0) or (p==0 and q==1):
        valor = print("(p  $\oplus$  q) = 1")
    elif (p==0 and q==0) or (p==1 and q==1):
        valor = print("(p  $\oplus$  q) = 0")

p = int(input("Indique os valores entre 0 e 1 para 'P': "))
q = int(input("Indique os valores entre 0 e 1 para 'Q': "))
print(ou_exclusivo(p,q))
```

Em relação ao 2º ponto do trabalho “Implemente uma função que receba um número inteiro positivo e devolva uma lista com os respetivos fatores primos.”, criámos uma função chamada “fator\_primo”, com o (y) como valor de input colocado pelo utilizador. Dentro da função colocámos uma lista “fatores\_primos”, que recebe os fatores primos de “Y”. Criámos uma variável “divisor” que recebeu o valor 2. De seguida colocámos uma condição: enquanto o divisor 2 for menor ou igual ao “Y” (colocado pelo utilizador), se “Y%2 == 0”, a lista recebe esse valor e o valor Y vai recebendo o resultado gerado pela operação anterior.

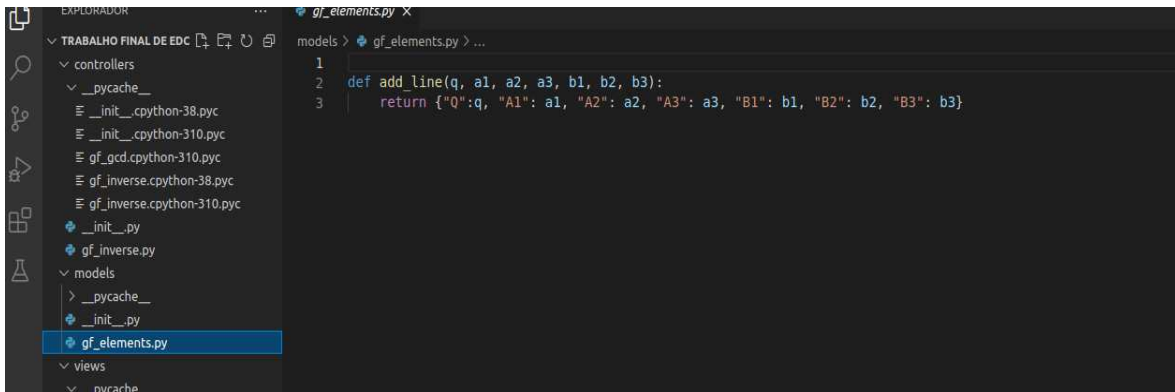
Caso o valor “Y” dividido por 2 não nos dê “resto 0”, o divisor “2” recebe “+1”. No final da função, a mesma deve retornar a lista “fatores primos” com os valores dentro da mesma. O input para “Y” foi colocado fora da função.

```
#2. Implemente uma função que receba um número inteiro positivo e devolva uma
#lista com os respectivos factores primos.

def fator_primo(y):
    fatores_primos = []
    divisor = 2
    while divisor <= y:
        if y%divisor == 0:
            fatores_primos.append(divisor)
            y = y/divisor
        else:
            divisor += 1
    return fatores_primos

print(fator_primo(int(input("Indique um valor para calcular os seus fatores primos: "))))
```

Sobre a terceira parte do trabalho, tendo em conta o enunciado, começámos por implementar em `models/gf_elements.py` uma função denominada `add_line`. Nesta, atribuímos variáveis (`"Q":q`, `"A1": a1`, `"A2": a2`, `"A3": a3`, `"B1": b1`, `"B2": b2`, `"B3": b3`) às respetivas colunas de modo a podermos manipulá-las.



```
1
2 def add_line(q, a1, a2, a3, b1, b2, b3):
3     return {"Q":q, "A1": a1, "A2": a2, "A3": a3, "B1": b1, "B2": b2, "B3": b3}
```

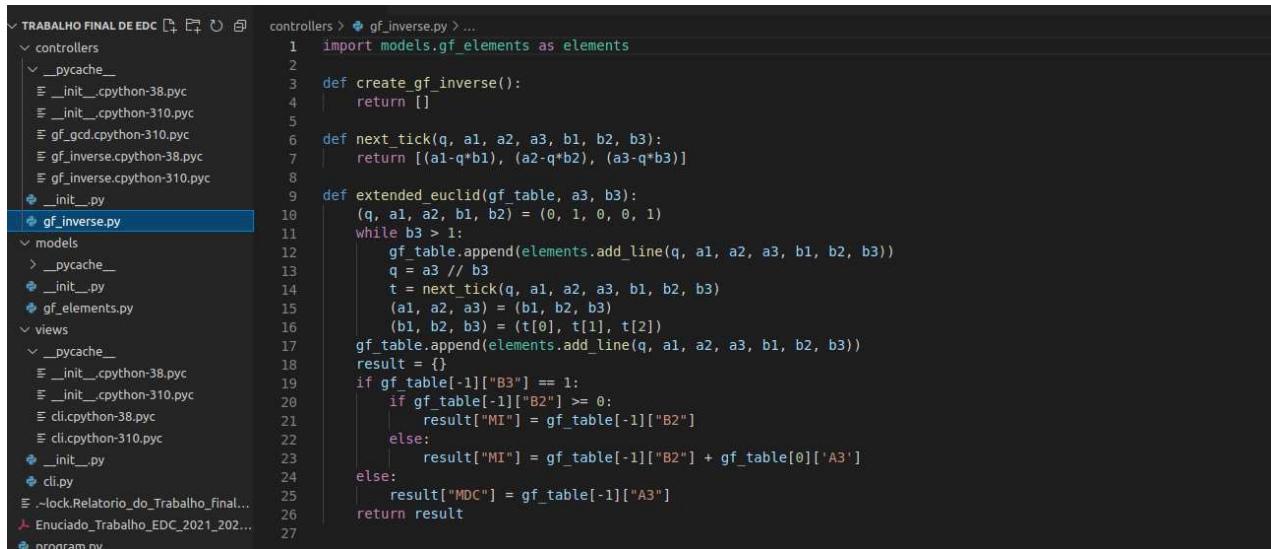
De seguida, em `controllers/gf_inverse.py`, importámos os elementos da função contidos em `"models"` e criámos as funções:

- `"create_gf_inverse"`, retorna uma lista vazia. Essa lista é guardada na variável `gf_tables`.
- `"next_tick"`, que faz os cálculos de modo a que consigamos obter os valores das colunas seguintes;
- `"extended_euclid"`, usámos o algoritmo de "Euclides expandido", conforme pedido no enunciado. Usando como parâmetros de entrada a `"gf_tabela"` e os valores indicados pelo utilizador (para serem atribuídos às variáveis `"a3"` e `"b3"`). As variáveis `"q"`, `"a1"`, `"a2"`, `"b1"`, `"b2"` recebem respetivamente os valores (0,1,0,0,1) por ordem. Enquanto `"b3"` for maior que 1, são adicionados à variável `"gf_table"`, os elementos retornados pela função `"add_line"`.

`"q"` recebe a divisão inteira de `"a3"` por `"b3"` e `"t"` retorna os valores calculados com `"a1"`, `a2`, `a3"` a receberem os valores `"b1"`, `b2`, `b3"`. De maneira a que o algoritmo continue a funcionar.

Criando um dicionário, vemos colocar neste, os valores das respetivas condições, seja `"MI"` ou `"MDC"`

Se o último valor em `"b3"` for igual a 1, verificamos se o último valor em `"b2"` é maior que 0, com o `"resultado MI"` a receber o valor contido em `"b2"`. Se o último resultado de `"b2"` for menor que 0, o `"resultado MI"` vai ser a soma do `"b2"` com `"a3"`. Caso o último resultado de `"b3"` não seja igual a 1, então o `"MDC"` vai ser o último valor de `"a3"`. Por fim, mandamos retornar o dicionário `"result"` de maneira a que seja mostrado o output na tela.



```
1 import models.gf_elements as elements
2
3 def create_gf_inverse():
4     return []
5
6 def next_tick(q, a1, a2, a3, b1, b2, b3):
7     return [(a1-q*b1), (a2-q*b2), (a3-q*b3)]
8
9 def extended_euclid(gf_table, a3, b3):
10     (q, a1, a2, b1, b2) = (0, 1, 0, 0, 1)
11     while b3 > 1:
12         gf_table.append(elements.add_line(q, a1, a2, a3, b1, b2, b3))
13         q = a3 // b3
14         t = next_tick(q, a1, a2, a3, b1, b2, b3)
15         (a1, a2, a3) = (b1, b2, b3)
16         (b1, b2, b3) = (t[0], t[1], t[2])
17     gf_table.append(elements.add_line(q, a1, a2, a3, b1, b2, b3))
18     result = {}
19     if gf_table[-1]["B3"] == 1:
20         if gf_table[-1]["B2"] >= 0:
21             result["MI"] = gf_table[-1]["B2"]
22         else:
23             result["MI"] = gf_table[-1]["B2"] + gf_table[0]["A3"]
24     else:
25         result["MDC"] = gf_table[-1]["A3"]
26     return result
27
```

Chegando, por último, ao views, criamos o ficheiro “cli.py”. Importamos as funções do modulo “gf\_inverse” do package “controllers”. Após a importação, foi definida a função cli () que será a interface com o utilizador. A função inicia com o ciclo “While” com a condição “True”, de forma a parar apenas quando indicado pelo user. À variável “gf\_table” foi atribuído o valor retornado pela função “create\_gf\_inverse ()” (uma lista vazia). A variável “numbers” é uma lista que recebe os input’s de “n” e “b”, com “exit” quando o utilizador pretender parar o programa. Utilizamos uma função “split” para separar os valores introduzidos, com o “espaço” como referência.

Criamos uma condição que verifica, no primeiro índice da lista “numbers [0]”, se o utilizador escreveu a palavra “exit”. Caso essa situação ocorra, o programa vai parar.

Desenvolvemos uma segunda condição, que verifica se foram colocados dois valores. Se na lista, não forem colocados valores a mais ou a menos, o programa vai “alertar” o utilizador com a mensagem “Necessários dois números inteiros”.

Caso a variável “numbers” contenha dois elementos, é iniciado um try que irá disparar o erro “Necessários dois números inteiros!” caso ocorra um problema ao se converter as strings na lista “numbers” para números inteiros.

Na condição seguinte, se o “b” for maior que o “n”, o programa vai alertar o utilizador que, “b” tem de ser menor ou igual a “n”. Caso, os valores estejam inseridos da maneira correta, um dicionário vai receber a função do algoritmo de “Euclides expandido”, com “gf\_table” e os valores inteiros de “n” e “b” (linha 19).

Organizamos o output, com os valores a nível estético, com o respetivo espaçamento e respetiva linha de separação, de forma a deixar o código mais organizado aos “olhos do utilizador”.

```

try:
    numbers[0] = int(numbers[0])
    numbers[1] = int(numbers[1])
    if int(numbers[1]) > int(numbers[0]):
        print(f"Erro: b ({numbers[1]}) tem de ser menor ou igual a n ({numbers[0]})")
    else:
        result: dict = gf_inverse.extended_euclid(gf_table, int(numbers[0]), int(numbers[1]))
        print(f"Q': <5>{'A1': ^7}{'A2': ^7}{'A3': ^7}{'B1': ^7}{'B2': ^7}{'B3': ^7}")
        print("-----")
        for i in gf_table:
            print(f"{'Q': <5>}{i['A1']: ^7}{i['A2']: ^7}{i['A3']: ^7}{i['B1']: ^7}{i['B2']: ^7}{i['B3']: ^7}")
        if "MI" in result.keys():
            print(f"Multiplicativo inverso: {result['MI']}")
            print("")
        else:
            print(f"MDC: {result['MDC']}")
            print("")
except ValueError:
    print("Necessários dois números inteiros!")

```

Fazendo um ciclo “for” (linha 22), o programa vai fazer “correr” os valores. Caso o “MI” seja uma chave no dicionário “result” (conforme explicado no controllers), o programa vai apresentar como output o “Multiplicativo Inverso”. Se “MI” não constar em “result”, o programa apresenta o “Máximo Divisor Comum”.

Em relação ao “except”, colocámos este valor de erro de maneira a “obrigar” o utilizador a colocar dois números inteiros.

Foram colocados “init’s”, de modo a ter os ficheiros “.py” em bibliotecas, conforme temos usado em “Algoritmia e Programação” e de forma a termos as pastas, do modo mais organizado possível.

