Universidade do Estado do Amazonas
Aluno: Natan Siqueira dos Santos
Disciplina: Tópicos Especiais em Eng. De Software

Introdução

Este trabalho tem como objetivo apresentar o teste prático solicitado na atividade. O programa escolhido para a realização dessa tarefa foi 1164 - Número Perfeito encontrado na plataforma BeeCrowd.

O objetivo da realização deste programa é encontrar quais os números positivos são números perfeitos, o número perfeito é um inteiro para o qual a soma de todos os seus divisores positivos próprios (excluindo ele mesmo) é igual ao próprio número.

Entrada

A entrada contém vários casos de teste. A primeira linha da entrada contém um inteiro **N** ($1 \leq N \leq 20$), indicando o número de casos de teste da entrada. Cada uma das **N** linhas seguintes contém um valor inteiro **X** ($1 \leq X \leq 10^8$), que pode ser ou não, um número perfeito.

Saída

Para cada caso de teste de entrada, imprima a mensagem "**X** eh perfeito" ou "**X** nao eh perfeito", de acordo com a especificação fornecida.

Figura 1: Entrada e saída do problema do URI

O código foi implementado na linguagem Pyhton 3.10.4, também foram adicionadas propostas para o aumento da eficiência dele. A versão testada é a versão pós aplicação das propostas de eficiência. Abaixo segue a função e a saída de teste do programa.

```python
def numPerfeito(num):
    soma = 0

    if(num < 1 or num > 10 ** 8): # Verifica se o valor pertence ao intervalo
        print("Entrada Inváilida")
        return


    for i in range(1,(num//2)+1):# Testa divisões até o maior divisor possível
        if num % i ==0:          # Verica se o numero testado é um divisor
            soma +=i             # Incrementa o divisor ao somátorio final
    if soma != num:              # Ao fim caso não seja perfeito
        print("{} nao eh perfeito".format(num)) # Saída negatica
    else:                        # Se for perfeito
        print("{} eh perfeito".format(num))     # Saída positiva
```

Universidade do Estado do Amazonas
Aluno: Natan Siqueira dos Santos
Disciplina: Tópicos Especiais em Eng. De Software
Figura 2: Função que testa se um número é perfeito ou não

```
# Número de testes
qtdNumeros = int(input())
# Laço de testes
for a in range (qtdNumeros):
    # Chamada de teste da função
    numPerfeito()
```

Figura 3: Chamada função para teste

```
C:\Users\Natan\AppData\Local\Programs\Python\P
3
15
15 nao eh perfeito
8
8 nao eh perfeito
28
28 eh perfeito

Process finished with exit code 0
```

Figura 4: Saída para três testes

Universidade do Estado do Amazonas
Aluno: Natan Siqueira dos Santos
Disciplina: Tópicos Especiais em Eng. De Software

Técnica Funcional

Abaixo está representado o conjunto universo do problema a ser solucionado onde todos elementos fora de **G** são considerados elementos inválidos para o programa. Utilizaremos este diagrama abaixo para representar os limites e o conjunto de respostas válidas para criação dos nossos roteiros de teste e analisar nosso valor limite.

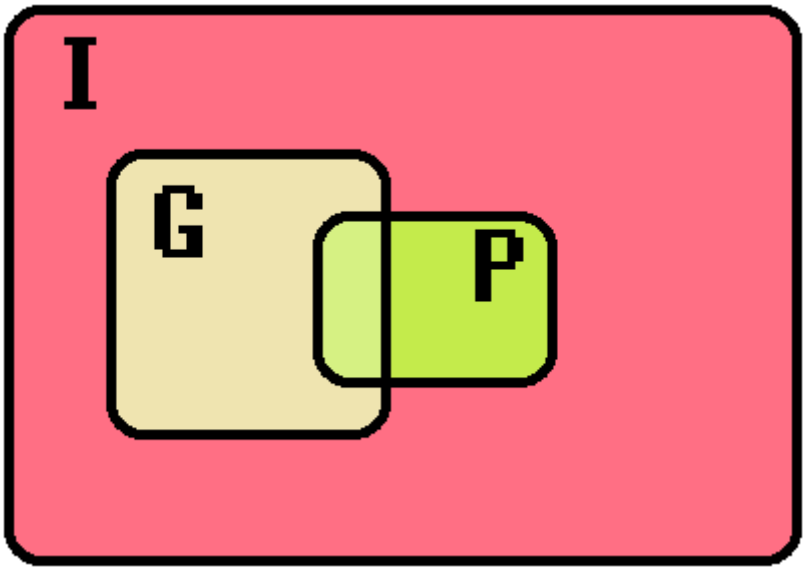

Figura 5: o conjunto de testes do programa

Na figura 5 temos que:

- I: Representa os números inteiros
- G: Representa o intervalo do problema
- P: Representa os números perfeitos

Pela lógica, chegamos a conclusão que G são resultados positivos, a intersecção entre G e P são os resultados positivos e perfeitos e tudo que está fora de G são resultados de entradas inválidas.

Limite Mínimo	Limite Máximo
{0,1,2}	{99.999.999, 100.000.000, 100.000.001 }

Como os valores perfeitos presentes no intervalo de 1 a 10^8 se limitam a cinco podemos representados diretamente como abaixo:

Números Perfeitos existentes entre os limites mínimos e máximos
6, 28, 496, 8.128, 33.550.336

Universidade do Estado do Amazonas

Aluno: Natan Siqueira dos Santos

Disciplina: Tópicos Especiais em Eng. De Software

Roteiro de Testes

CT_ID	ENTRADA	SAÍDA
CT_01	0	Entrada Inválida
CT_02	1	nao eh perfeito
CT_03	5	nao eh perfeito
CT_04	6	eh perfeito
CT_05	7	nao eh perfeito
CT_06	27	nao eh perfeito
CT_07	28	eh perfeito
CT_08	29	nao eh perfeito
CT_09	495	nao eh perfeito
CT_10	496	eh perfeito
CT_11	497	nao eh perfeito
CT_12	8.127	nao eh perfeito
CT_13	8.128	eh perfeito
CT_14	8.129	nao eh perfeito
CT_15	33.550.335	nao eh perfeito
CT_16	33.550.336	eh perfeito
CT_17	33.550.337	nao eh perfeito
CT_18	99.999.999	nao eh perfeito
CT_19	100.000.000	nao eh perfeito
CT_20	100.000.001	Entrada Inválida

Universidade do Estado do Amazonas
Aluno: Natan Siqueira dos Santos
Disciplina: Tópicos Especiais em Eng. De Software

Refinando o roteiro obtemos a seguinte tabela de teste:

CT_ID	ENTRADA	SAÍDA
CT_01	0	Entrada Inválida
CT_02	1	nao eh perfeito
CT_04	6	eh perfeito
CT_05	7	nao eh perfeito
CT_07	28	eh perfeito
CT_08	29	nao eh perfeito
CT_10	496	eh perfeito
CT_11	497	nao eh perfeito
CT_13	8.128	eh perfeito
CT_14	8.129	nao eh perfeito
CT_16	33.550.336	eh perfeito
CT_17	33.550.337	nao eh perfeito
CT_20	100.000.001	Entrada Inválida

Universidade do Estado do Amazonas
Aluno: Natan Siqueira dos Santos
Disciplina: Tópicos Especiais em Eng. De Software

Técnica Estrutural

Grafo de Fluxo de Controle

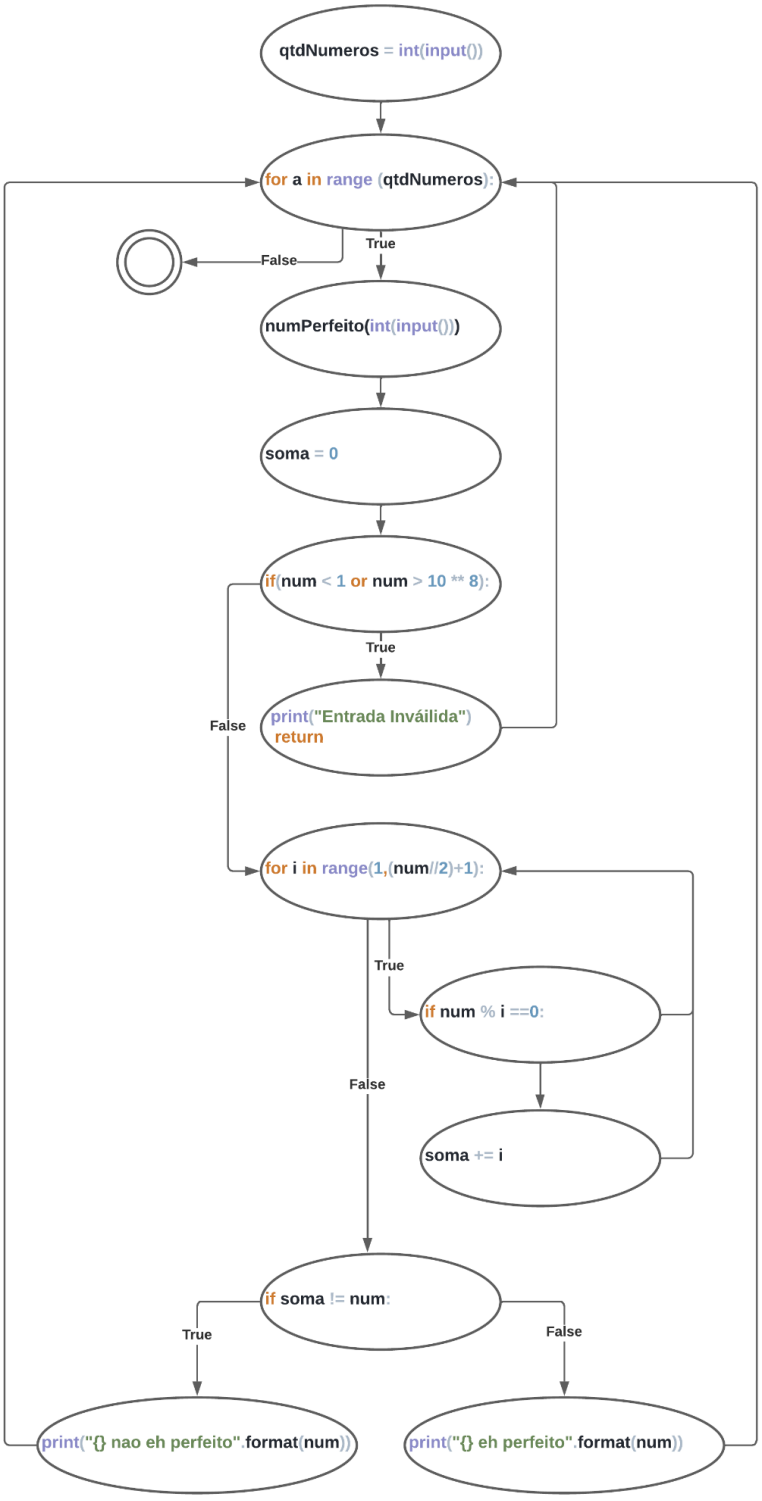

Figura 6: GFC do programa

Universidade do Estado do Amazonas
Aluno: Natan Siqueira dos Santos
Disciplina: Tópicos Especiais em Eng. De Software

Implementação da ferramenta de testes

Para testar a aplicação foram utilizadas as bibliotecas pytest e unittest. As ferramentas possuem uma limitação com as funções que foram utilizadas para o teste. Então foi necessário trocar os prints por returns para que a validação das saídas fossem feitas e também isolar a função do resto do código o comentando. O resultado ficou demonstrado na figura 7.

Na figura 8 temos a criação do arquivo de teste que utiliza os casos de teste presentes no roteiro de teste. E na figura 9 temos a saída da classe de teste, onde cada ponto verde representa um teste realizado, e abaixo destes pontos se encontra o status final dos teste, onde podem observar que o programa passou a todos os teste submetidos.

```
def numPerfeito(num):
    soma = 0

    if(num < 1 or num > 10 ** 8): # Verifica se o valor pertence ao intervalo
        return(print("Entrada Inválida"))


    for i in range(1,(num//2)+1):# Testa divisões até o maior divisor possível
        if num % i ==0:          # Verica se o numero testado é um divisor
            soma +=i             # Incrementa o divisor ao somátorio final
    if soma != num:              # Ao fim caso não seja perfeito
        return(print("nao eh perfeito")) # Saída negatica
    else:                        # Se for perfeito
        return(print("eh perfeito"))     # Saída positiva


'''
# Número de testes
qtdNumeros = int(input())
# Laço de testes
for a in range (qtdNumeros):
    # Chamada de teste da função
    numPerfeito(int(input()))
'''
```

Figura 7: Código adaptado para os testes

Universidade do Estado do Amazonas
Aluno: Natan Siqueira dos Santos
Disciplina: Tópicos Especiais em Eng. De Software

```python
class test_numPerfeito (unittest.TestCase):
    def test_ct01(self):
        self.assertEqual(numPerfeito.numPerfeito(0), "Entrada Inválida")
    def test_ct02(self):
        self.assertEqual(numPerfeito.numPerfeito(1), "nao eh perfeito")
    def test_ct04(self):
        self.assertEqual(numPerfeito.numPerfeito(6), "eh perfeito")
    def test_ct05(self):
        self.assertEqual(numPerfeito.numPerfeito(7), "nao eh perfeito")
    def test_ct07(self):
        self.assertEqual(numPerfeito.numPerfeito(28), "eh perfeito")
    def test_ct08(self):
        self.assertEqual(numPerfeito.numPerfeito(29), "nao eh perfeito")
    def test_ct10(self):
        self.assertEqual(numPerfeito.numPerfeito(496), "eh perfeito")
    def test_ct11(self):
        self.assertEqual(numPerfeito.numPerfeito(497), "nao eh perfeito")
    def test_ct13(self):
        self.assertEqual(numPerfeito.numPerfeito(8128), "eh perfeito")
    def test_ct14(self):
        self.assertEqual(numPerfeito.numPerfeito(8129), "nao eh perfeito")
    def test_ct16(self):
        self.assertEqual(numPerfeito.numPerfeito(33550336), "eh perfeito")
    def test_ct17(self):
        self.assertEqual(numPerfeito.numPerfeito(33550337), "nao eh perfeito")
    def test_ct20(self):
        self.assertEqual(numPerfeito.numPerfeito(100000001), "Entrada Inválida")
```

Figura 8: Código da classe de teste

```
C:\Users\Natan\Desktop\Programação\weasel>pytest test_numPerfeito.py
============================================= test session starts =============================================
platform win32 -- Python 3.10.4, pytest-7.1.1, pluggy-1.0.0
rootdir: C:\Users\Natan\Desktop\Programação\weasel
collected 13 items

test_numPerfeito.py .............                                                                    [100%]

============================================== 13 passed in 1.51s ==============================================
```

Figura 9: Resultado dos testes

Conclusões

Com a implementação deste trabalho foi possível desenvolver melhor o entendimento sobre técnicas e ferramentas úteis que podem ser utilizadas durante o teste de softwares. Com a necessidade de decidir qual aplicativo seria utilizado nesse trabalho também foi possível conhecer ferramentas de testes para ambientes de desenvolvimento de jogos.

Ao fim, o software testado demonstrou possuir as saídas esperadas para o caso de teste. Segue abaixo o link do Github para acessar os arquivos python:

https://github.com/NatanSIsantos/Topico-Especiais-Engenharia-de-Software/tree/main/trabalho_pratico_1