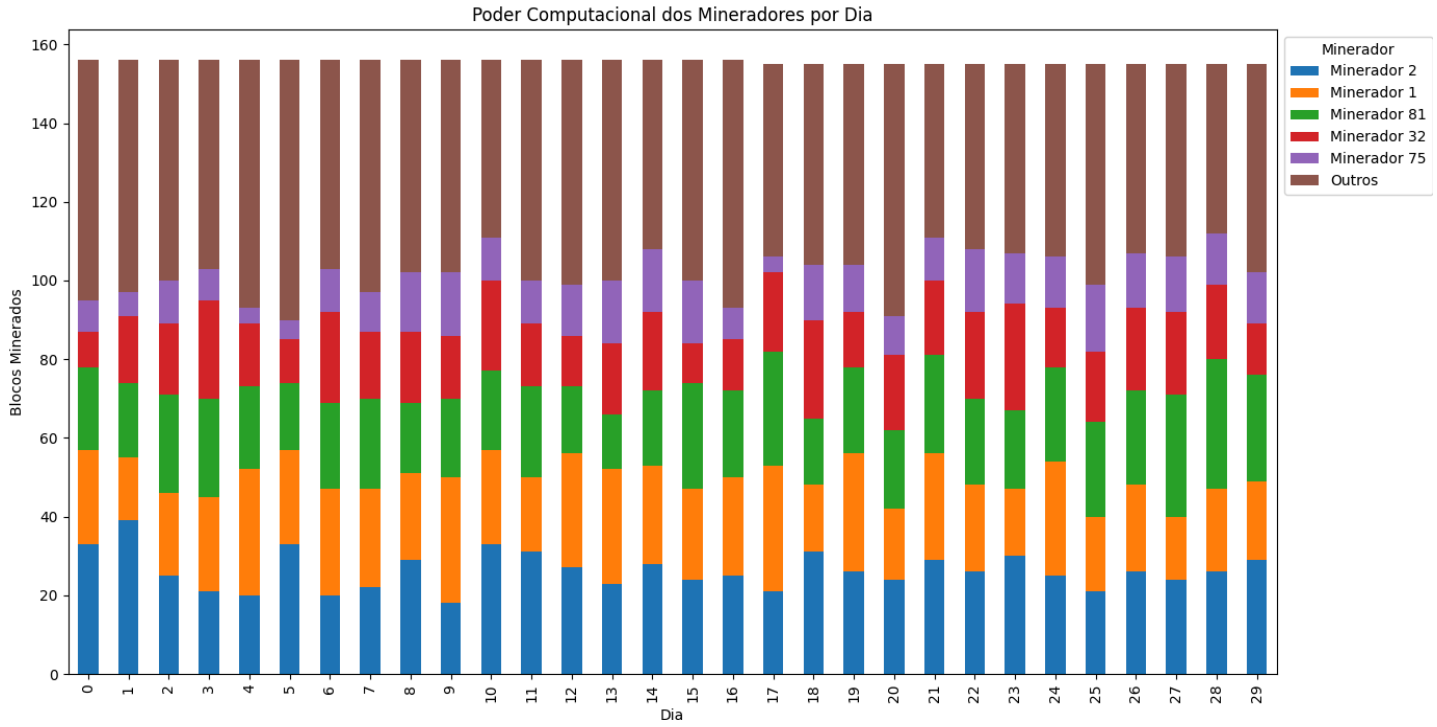
John Vitor da Silva Cunha
11821BCC005

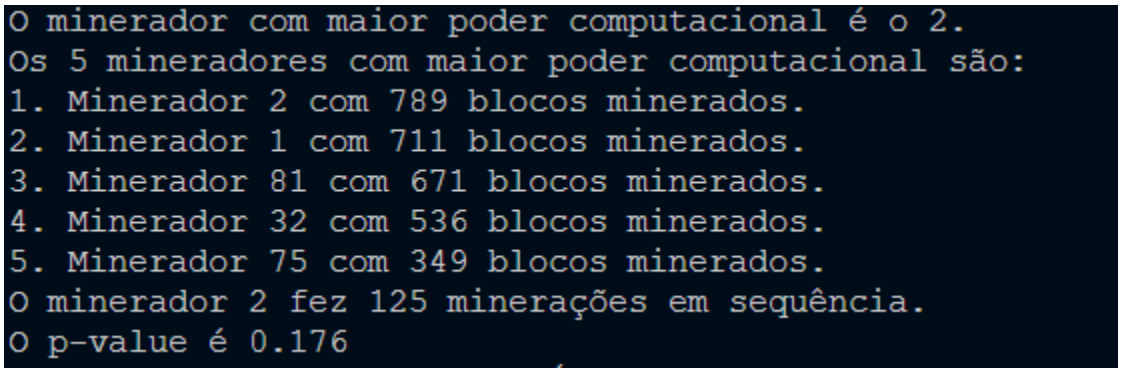
Link do GitHub com o código: https://github.com/johnsigma/mineracao_egoista

Gráfico do poder computacional de cada minerador por dia:



5 maiores mineradores do mês de junho foram os mineradores de ID 2, 1, 81, 32, 75, respectivamente do maior para o menor.

O minerador ID 2 teve 125 minerações em sequência e seu p-value com 1000 permutações foi de 0.176.



Código main.py:

```
import numpy as np
import funcoes

# Carregar o arquivo npy
data = np.load('block_integer_array.npy')


# Divide o arquivo em 12 subarrays, um para cada mês
data2 = np.array_split(data, 12)

# O mês do meu aniversário é junho, como o array começa em 0, o mês de junho é o 5
mineracoesMes = data2[5]

# Divide o array de junho em 30 subarrays, um para cada dia
mineracoesDias = np.array_split(mineracoesMes, 30)

# cria um array de array que quantos blocos cada minerador minerou em cada dia
repeticoesPorDia = funcoes.cria_repeticoes_por_dia(mineracoesDias)
df = funcoes.cria_dataframe(repeticoesPorDia, 5)

# plota o gráfico, feche a janela do gráfico para continuar a execução
funcoes.plota_grafico(df)

# imprime os 5 mineradores que mais mineraram em junho
funcoes.imprime_mineradores(df, 5)

# busca o minerador que mais minerou em junho
maior_minerador = funcoes.busca_minerador_mais_poderoso(df)

# conta quantas mineracoes em sequencia o minerador mais poderoso fez
mineracoes_sequencia = funcoes.conta_mineracoes_sequencia(
    mineracoesDias, maior_minerador)

print(f'O minerador {maior_minerador} fez {
      mineracoes_sequencia} minerações em sequência.')

# faz 1000 permutações para calcular o p-value
mineracoes_sequenciais_do_maior_minerador = funcoes.teste_k_permutacoes(
    mineracoesDias, maior_minerador, 1000)

# ordena o array para calcular o p-value
mineracoes_sequenciais_do_maior_minerador.sort()
```

```
# calcula o p-value
p_value = funcoes.calcula_p_value(
    mineracoes_sequenciais_do_maior_minerador, mineracoes_sequencia)

print(f'O p-value é {p_value}')
```

Código funcoes.py:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


def cria_repeticoes_por_dia(mineracoesDias):
    repeticoesPorDia = []

    for dia in mineracoesDias:
        repeticoesPorDia.append(np.bincount(dia))

    return repeticoesPorDia


def cria_dataframe(repeticoesPorDia, k_maiores=5):
    df = pd.DataFrame(repeticoesPorDia).fillna(0).astype(int)
    df.columns = [f'Minerador {i}' for i in range(df.shape[1])]
    df.index.name = 'Dia'

    soma_mineradores = df.sum(axis=0)

    maiores_mineradores = soma_mineradores.nlargest(k_maiores).index

    # Agrupar os demais mineradores em "outros"
    df['Outros'] = df.loc[:, ~df.columns.isin(maiores_mineradores)].sum(axis=1)
    df = df[maiores_mineradores.tolist() + ['Outros']]

    return df


def plota_grafico(df):

    # Plotar o gráfico
    df.plot(kind='bar', stacked=True, figsize=(15, 8))
    plt.xlabel('Dia')
    plt.ylabel('Blocos Minerados')
    plt.title('Poder Computacional dos Mineradores por Dia')
    plt.legend(title='Minerador', bbox_to_anchor=(1, 1), loc='upper left')
    plt.show()


def busca_minerador_mais_poderoso(df):
```

```
    soma_mineradores = df.loc[:, df.columns != 'Outros'].sum(axis=0)
    maior_minerador = soma_mineradores.idxmax().split()[-1]
    return maior_minerador


def busca_k_maiores_mineradores(df, k_maiores=5):
    soma_mineradores = df.loc[:, df.columns != 'Outros'].sum(axis=0)
    k_maiores_mineradores = soma_mineradores.nlargest(k_maiores)
    return k_maiores_mineradores


def imprime_mineradores(df, k_maiores=5):

    maior_minerador = busca_minerador_mais_poderoso(df)
    k_maiores_mineradores = busca_k_maiores_mineradores(df, k_maiores)

    print(f'O minerador com maior poder computacional é o {maior_minerador}.')
    print(f'Os {k_maiores} mineradores com maior poder computacional são:')
    for i, (minerador, poder) in enumerate(k_maiores_mineradores.items(), 1):
        print(f'{i}. {minerador} com {poder} blocos minerados.')


def conta_mineracoes_sequencia(mineracoesDias, maior_minerador):
    mineracoes_sequencia = 0
    mineracao_aux = -1

    for dia in mineracoesDias:

        for j in range(len(dia)):

            if mineracao_aux != -1:
                if mineracao_aux == dia[j] and mineracao_aux == maior_minerador:
                    mineracoes_sequencia += 1
                mineracao_aux = -1

            mineracaoAtual = str(dia[j])

            mineracaoProxima = str(dia[j+1]) if j+1 < len(dia) else None

            if mineracaoProxima == None:
                mineracao_aux = mineracaoAtual
                break

            if mineracaoAtual == maior_minerador and mineracaoProxima == mineracaoAtual:
```

```
            mineracoes_sequencia += 1

    return mineracoes_sequencia


def verifica_permutacao(array1, array2):
    # Verificar se os arrays têm o mesmo tamanho
    if len(array1) != len(array2):
        return False

    # Verificar se os arrays têm os mesmos elementos
    elementos_unicos_array1, contagens_array1 = np.unique(
        array1, return_counts=True)
    elementos_unicos_array2, contagens_array2 = np.unique(
        array2, return_counts=True)

    if not np.array_equal(elementos_unicos_array1, elementos_unicos_array2):
        return False

    # Verificar se as contagens de cada elemento são iguais
    if not np.array_equal(contagens_array1, contagens_array2):
        return False

    return True


def criar_novo_array_permutado(data):

    novo_array = []

    for dia in data:
        novo_dia = np.random.permutation(dia)
        novo_array.append(novo_dia)

    return novo_array


def teste_k_permutacoes(data, maior_minerador, k=1000):
    mineracoes_sequencia = []
    for _ in range(k):
        novo_array = criar_novo_array_permutado(data)
        mineracoes = conta_mineracoes_sequencia(
            novo_array, maior_minerador)
        mineracoes_sequencia.append(mineracoes)
```

```
    return mineracoes_sequencia


def calcula_p_value(mineracoes_sequenciais_do_maior_minerador, mineracoes_sequencia):
    p_value = -1

    for i in range(len(mineracoes_sequenciais_do_maior_minerador)):
        numero_sequencias = mineracoes_sequenciais_do_maior_minerador[i]

        if str(numero_sequencias) == str(mineracoes_sequencia):

            p_value = (i+1)/len(mineracoes_sequenciais_do_maior_minerador)
            break

    return p_value
```