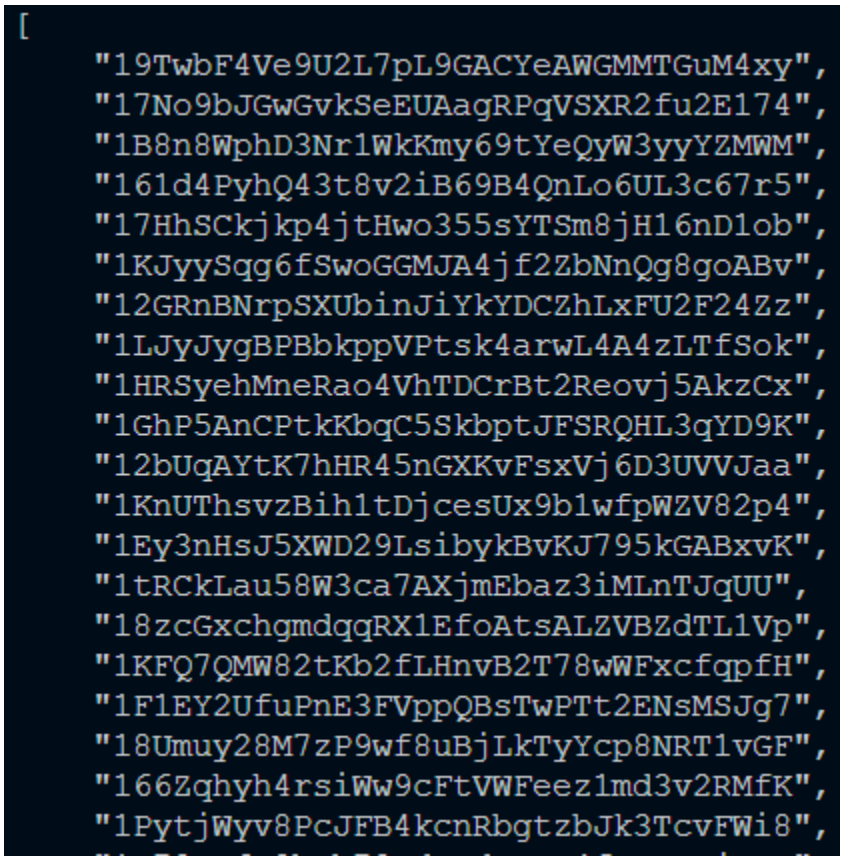
John Vitor da Silva Cunha
11821BCC005

Link do GitHub com o código: [github]

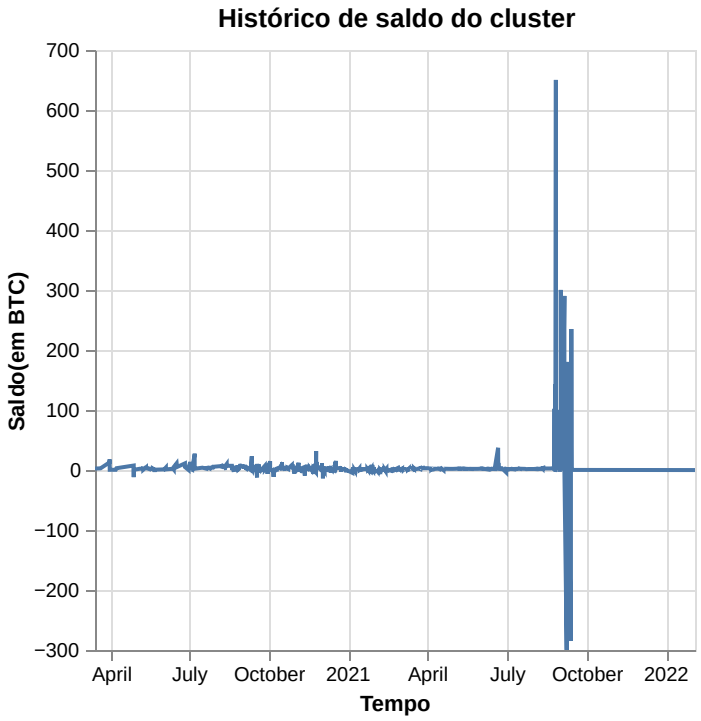
Primeiros endereços do cluster 1JHH1pmHujcVa1aXjRrA13BJ13iCfgfBqj (de um total de 47 endereços):



1. Primeiras entradas do histórico do saldo do cluster (de um total de 789 transações):

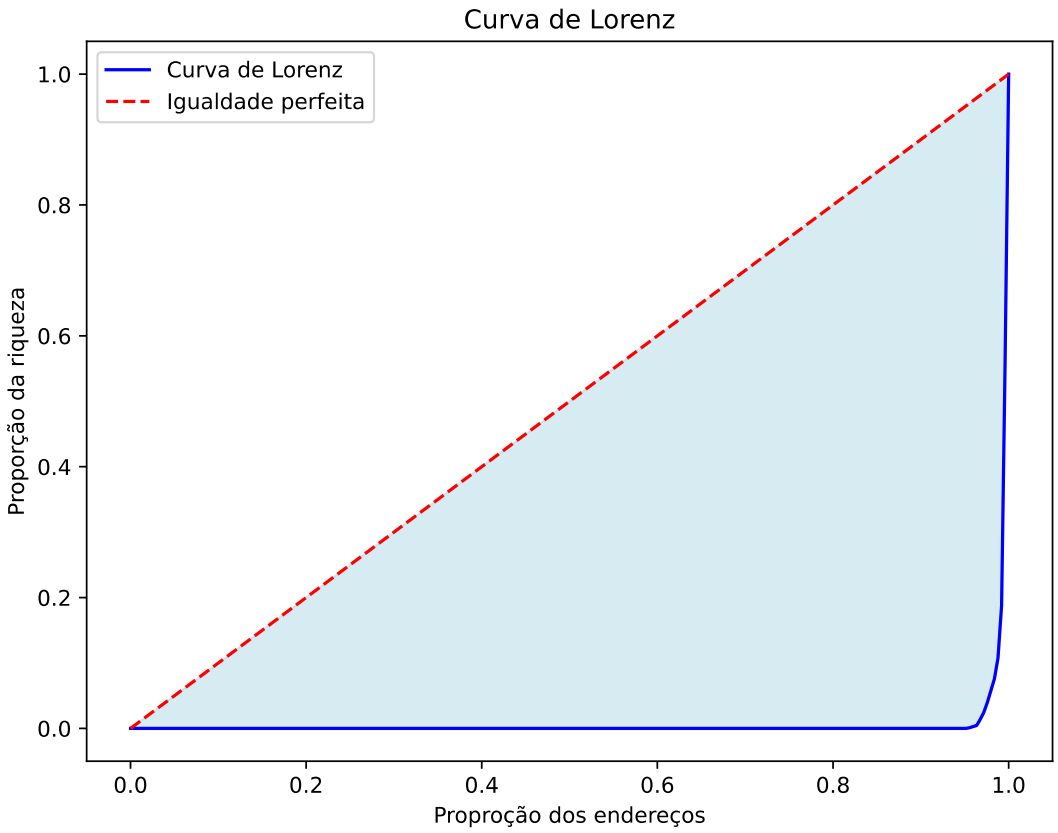
```
Data: 13/03/2020 18:19:37, Saldo: 2.4763 BTC, Transação: 46348c7dfc1437a49dfbe7629594a3b763e742d07077c65a35d3b6c222c23fe0
Data: 16/03/2020 13:38:22, Saldo: 2.592579 BTC, Transação: 5a873e36aa16093736a1bdfc53b65bf16459e1f9846e26c3f6d054c9ba0cec91
Data: 20/03/2020 02:29:07, Saldo: 2.74455779 BTC, Transação: c5881a2669eb1645b71a93ddaa937fe47059a877ed1db55bb49f1491155cf38b
Data: 30/03/2020 14:23:51, Saldo: 12.74455779 BTC, Transação: c38c22752e3c2a6ecdfa74cee0e97acf7b8e5b26b4854856cf9e81ae6003d6b3
Data: 30/03/2020 15:13:42, Saldo: 17.74445779 BTC, Transação: 90b006d3f466747ef555ed04366d45d22b2d2d6edc8c9f656f67f84ceed262cd
Data: 30/03/2020 15:13:42, Saldo: 17.74455779 BTC, Transação: a154dd49c585c0c9f672aee28aeca28d59f88c2f2e8bd0d520b325878a6fdc2e
Data: 30/03/2020 17:17:33, Saldo: 0.26835779 BTC, Transação: b758edef64b484f24e471c11a25ff2b092ac11ed63a310c297efd10fd2d01aa1
Data: 31/03/2020 16:08:36, Saldo: 0.36070379 BTC, Transação: d7819d7b4b7360fc483d7c2dbdd24260e6493c19b93138aa10dbacbf2833c87b
Data: 06/04/2020 16:29:40, Saldo: 0.41757938 BTC, Transação: 1c29c2e372edf38ec974b7dd68727e84b24e69c5efb1832186c04a59d55e93d9
Data: 07/04/2020 13:29:39, Saldo: 3.03881138 BTC, Transação: 2575f1900679f00c70b8159601fad1e0e07fae8683761e6e024ff7dc5aa2d44c
Data: 27/04/2020 09:25:21, Saldo: 7.33881138 BTC, Transação: 707389f3738f08f45717f877afac906ab6091e907167b0c3b58bb80378fc2e30
Data: 27/04/2020 11:41:41, Saldo: -11.90242062 BTC, Transação: 94570fe521ccc90c29d36c0dba2d5fc654800b57734d66a3dd5e8cfbd08eb4cb
Data: 27/04/2020 11:41:41, Saldo: 0.41757938 BTC, Transação: 048080b130d5c0e6b30b092be29652427cd09f0fddee19763e33bf0ef2ab9d63
Data: 04/05/2020 10:06:18, Saldo: 2.01034363 BTC, Transação: 77f9b63095ff2989217e193f5d300011a012a2eaba48809da895a58cf3fcceab
Data: 07/05/2020 14:42:15, Saldo: 2.17440966 BTC, Transação: 5f5a762edad59484d95b49f00f4148f370321244076daa4236fa3ed75741d5f0
Data: 07/05/2020 14:42:15, Saldo: 4.17440966 BTC, Transação: 3bf24b64b0413534cecff91c05b3c572474cd9a0e942751fd6f33cc357dcb4c0
Data: 07/05/2020 15:39:43, Saldo: 2.02348911 BTC, Transação: a8e59c8007db9064e958f5166eb0f8f2dba86fb10ff2af6704b2df6a86bc44c2
Data: 08/05/2020 10:21:33, Saldo: 5.29935118 BTC, Transação: dc718333620eb4f78a08dab0302ab3aae0f628e8bcb1599626dabe79865bc0ee
Data: 08/05/2020 10:25:24, Saldo: 0.58164541 BTC, Transação: 7786a68e965f27724de57fab785c2f2abda6f7440fdefecc066baddf951d7ebd
Data: 11/05/2020 12:52:37, Saldo: 4.44164541 BTC, Transação: d43683449ee5b1ea966a3f05d3d11373dfbd12ebe533c10ae2630109f5de91e5
Data: 11/05/2020 14:06:21, Saldo: 0.64984647 BTC, Transação: 5041cdbed25b0264b7cc3897b89158139d6448c7225203e4e7fdb76943376b19
Data: 11/05/2020 14:53:12, Saldo: 2.64984647 BTC, Transação: 58b8b677d16a6e527f5a4571cc19f113192f0abf750cd8256d9b7ac509f528e5
Data: 11/05/2020 14:53:12, Saldo: 3.09058012 BTC, Transação: 9b149ea0422a9896afe7d90d39d437c6264837dbfcfa828861daed9c6bf33e6e
Data: 13/05/2020 16:12:32, Saldo: 2.44396679 BTC, Transação: 92a1d1098264b000e6ea786dda0c34a41f5cd3ade029c02491b8f3750f3b0bc2
Data: 15/05/2020 14:44:51, Saldo: 1.44348898 BTC, Transação: cca41d1d4d77985c0d7c3f6bb72807aa5911504978353b8e74755b495a97a74f
```

Gráfico do histórico de saldo:



2. Índice de Gini: 0.9872776470147674

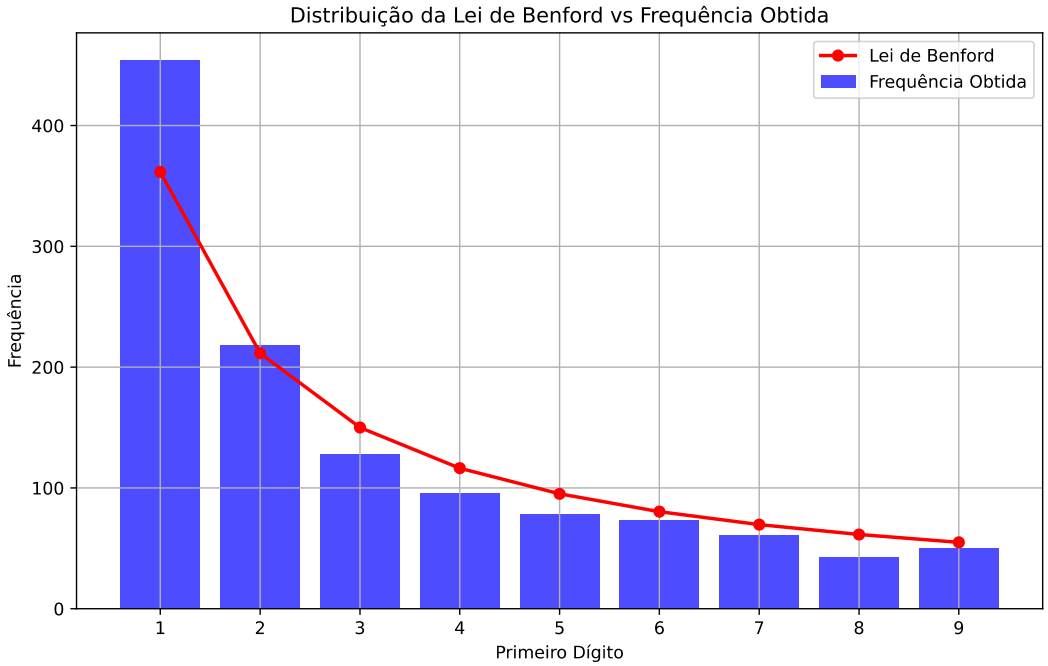
Gráfico de Lorenz:



3.

Esperado: [361.5370247924414, 211.48560212587316, 150.05142266656821, 116.38892562267576, 95.0966765031974, 80.40309434636647, 69.64832832020177, 61.43417945930493, 54.95474616337085]
Obtido: [454, 218, 128, 96, 78, 73, 61, 43, 50]
Qui-Quadrado: 41.46775548369663
P-Valor: 1.7040616134071377e-06

Gráfico de Benford:



Código:

```
import math
from scipy.stats import chisquare
from collections import Counter
from datetime import datetime
import altair as alt
import os
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


def ler_arquivo(base_path):

    lista_dict = []

    for nome_arquivo in os.listdir(base_path):
        if nome_arquivo.endswith('.json'):
            caminho_arquivo = os.path.join(base_path, nome_arquivo)
            with open(caminho_arquivo, 'r', encoding='utf-8') as arquivo:
                conteudo_json = json.load(arquivo)
                lista_dict.append(conteudo_json)

    return lista_dict


def clusterizar_enderecos(enderecos):

    toCluster = {}
    cc = {}

    for endereco in enderecos:

        toCluster[endereco['address']] = []
        for tx in endereco['txs']:
            temp = [i['prev_out']['addr'] for i in tx['inputs']]
```

```
            # temp.extend([i['addr'] for i in tx['out']])
            if endereco['address'] in temp:
                toCluster[endereco['address']].append(temp)

        n_tx = endereco['n_tx']
        done = len(endereco['txs'])

        while done < n_tx:
            for tx in endereco['txs']:
                temp = [i['prev_out']['addr'] for i in tx['inputs']]
                # temp.extend([i['addr'] for i in tx['out']])
                if endereco['address'] in temp:
                    toCluster[endereco['address']].append(temp)

            done += len(endereco['txs'])

    for endereco in enderecos:
        # print('-------------------------')
        clusters = []
        cc[endereco['address']] = []

        for tx in toCluster[endereco['address']]:

            c = []
            for i in range(len(clusters)):
                if any(x in clusters[i] for x in tx):
                    c.append(i)

            if len(c) == 0:
                clusters.append(tx)
            else:
                x = c[0]
                del c[0]

                clusters[x].extend(tx)

                for i in c:
                    clusters[x].extend(clusters[i])

                clusters[x] = list(set(clusters[x]))
```

```
            # print(endereco['address'])

            cc[endereco['address']].extend(clusters[0])

    return cc


def calculaHistoricoSaldo(transacoes, cluster):
    hashs_calculados = []
    transacoes_relevantes = []

    for transacao in transacoes:
        hash_transacao = transacao['hash']

        entradas_no_cluster = False
        saidas_no_cluster = False

        if hash_transacao in hashs_calculados:
            continue

        hashs_calculados.append(hash_transacao)

        for entrada in transacao.get('inputs', []):
            endereco = entrada['prev_out'].get('addr')
            if endereco in cluster:
                entradas_no_cluster = True
                break

        for saida in transacao.get('out', []):
            endereco = saida.get('addr')
            if endereco in cluster:
                saidas_no_cluster = True
                break

        if entradas_no_cluster or saidas_no_cluster:
            transacoes_relevantes.append({
                'time': transacao['time'],
                'hash': hash_transacao,
                'entradas_no_cluster': entradas_no_cluster,
                'saidas_no_cluster': saidas_no_cluster,
                'inputs': transacao.get('inputs', []),
```

```
            'out': transacao.get('out', [])
        })

    transacoes_relevantes.sort(key=lambda x: x['time'])

    historico_saldo = []
    saldo_atual = 0

    for transacao in transacoes_relevantes:
        if transacao['entradas_no_cluster']:
            for entrada in transacao['inputs']:
                if entrada['prev_out']['addr'] in cluster:
                    saldo_atual -= entrada['prev_out']['value']

        if transacao['saidas_no_cluster']:
            for saida in transacao['out']:
                if saida.get('addr') in cluster:
                    saldo_atual += saida['value']

        historico_saldo.append(
            (datetime.fromtimestamp(transacao['time']), saldo_atual/100000000,
transacao['hash']))

    return historico_saldo


def plotar_grafico_linha(historico_saldo, nome):
    tempos_datetime = [t[0] for t in historico_saldo]
    saldos = [saldo[1] for saldo in historico_saldo]

    df = pd.DataFrame({'Tempo': tempos_datetime, 'Saldo': saldos})

    chart = alt.Chart(df).mark_line().encode(
        x=alt.X('Tempo:T', title='Tempo'),
        y=alt.Y('Saldo:Q', title='Saldo(em BTC)'),
        tooltip=['Tempo', 'Saldo']
    ).properties(
        title='Histórico de saldo do cluster'
    ).interactive()

    chart.save(f'linha_{nome}.html')
```

```
def plotar_grafico_area(historico_saldo):
    tempos_datetime = [t[0] for t in historico_saldo]
    saldos = [saldo[1] for saldo in historico_saldo]

    df = pd.DataFrame({'Tempo': tempos_datetime, 'Saldo': saldos})

    chart = alt.Chart(df).mark_area().encode(
        x=alt.X('Tempo:T', title='Tempo'),
        y=alt.Y('Saldo:Q', title='Saldo(em BTC)'),
        tooltip=['Tempo', 'Saldo']
    ).properties(
        title='Histórico de saldo do cluster'
    ).interactive()

    chart.save('area.html')


def plotar_grafico_histograma(historico_saldo):
    tempos_datetime = [t[0] for t in historico_saldo]
    saldos = [saldo[1] for saldo in historico_saldo]

    df = pd.DataFrame({'Tempo': tempos_datetime, 'Saldo': saldos})

    chart = alt.Chart(df).mark_bar().encode(
        x=alt.X('Tempo:T', title='Tempo'),
        y=alt.Y('Saldo:Q', title='Saldo(em BTC)'),
        tooltip=['Tempo', 'Saldo']
    ).properties(
        title='Distribuição de saldo do cluster'
    ).interactive()

    chart.save('hist.html')


def imprimir_historico_saldo(historico_saldo):
    for item in historico_saldo:
        print(f'Data: {item[0].strftime(
            "%d/%m/%Y %H:%M:%S")}, Saldo: {item[1]} BTC, Transação: {item[2]}')
```

```
def naoRepetidos(cluster_h1, cluster):
    set_cluster = set(cluster)
    set_cluster_h1 = set(cluster_h1)

    unique_in_cluster = set_cluster - set_cluster_h1
    unique_in_cluster_h1 = set_cluster_h1 - set_cluster

    return [(item, 'cluster') for item in unique_in_cluster] + [(item, 'cluster_h1') for item in
unique_in_cluster_h1]


def calcular_indice_gini(valores):
    # Converte a lista para um array numpy e ordena os valores
    valores = np.array(valores)
    valores_ordenados = np.sort(valores)

    # Calcula os índices
    n = len(valores)

    # Calcula a soma dos produtos dos valores ordenados
    soma_produtos = np.sum((2 * np.arange(1, n+1) - n - 1) * valores_ordenados)

    # Calcula o índice de Gini
    gini = soma_produtos / (n * np.sum(valores_ordenados))

    return gini


def curva_lorenz(values):
    # Ordena os valores e calcula a soma cumulativa
    values = np.array(values)
    values_sorted = np.sort(values)

    # Soma cumulativa dos valores
    cum_values = np.cumsum(values_sorted)

    # Normaliza para que o total seja 1
    cum_values_normalized = cum_values / cum_values[-1]

    # Adiciona um ponto (0, 0) ao início da curva
```

```
    lorenz_curve = np.insert(cum_values_normalized, 0, 0)

    return lorenz_curve


def plota_curva_lorenz(values):
    # Obtém a curva de Lorenz
    lorenz = curva_lorenz(values)

    # Eixo x: proporção da população
    x = np.linspace(0, 1, len(lorenz))

    # Plotando a curva de Lorenz
    plt.figure(figsize=(8, 6))
    plt.plot(x, lorenz, label='Curva de Lorenz', color='blue')

    # Linha de igualdade perfeita (linha de 45°)
    plt.plot([0, 1], [0, 1], label='Igualdade perfeita',
             color='red', linestyle='--')

    # Preenchendo a área entre a curva de Lorenz e a linha de igualdade
    plt.fill_between(x, lorenz, x, color='lightblue', alpha=0.5)

    # Personalizando o gráfico
    plt.title("Curva de Lorenz")
    plt.xlabel("Proproção dos endereços")
    plt.ylabel("Proporção da riqueza")
    plt.legend()

    plt.savefig("lorenz.svg")


def valores_por_endereco(cluster, transacoes):
    valoresPorEndereco = {endereco: 0 for endereco in cluster}

    for transacao in transacoes:
        for entrada in transacao['inputs']:
            if entrada['prev_out']['addr'] in cluster:
                valoresPorEndereco[entrada['prev_out']
                                   ['addr']] -= entrada['prev_out']['value']
        for saida in transacao['out']:
```

```
            if saida.get('addr') in cluster:
                valoresPorEndereco[saida['addr']] += saida['value']

    for endereco, valor in valoresPorEndereco.items():
        if valor < 0:
            valoresPorEndereco[endereco] = 0
        else:
            valoresPorEndereco[endereco] = valor / 100000000
    return valoresPorEndereco


def obter_primeiro_digito(valor):
    while valor >= 10:
        valor //= 10
    return valor


def calcular_frequencia_benford(transacoes):
    # Extrai os valores das transações
    valores = [
        sum(saida['value'] for saida in tx['out'])
        for tx in transacoes
    ]  # Soma os valores de todas as saídas para obter o valor total da transação

    # Extrai os primeiros dígitos
    primeiros_digitos = [int(str(abs(valor))[0]) for valor in valores]

    # Contando a frequência de cada dígito
    contagem_primeiros_digitos = Counter(primeiros_digitos)

    return contagem_primeiros_digitos


def frequencia_esperada_benford(total_transacoes):
    return {digito: math.log10(1 + 1 / digito) * total_transacoes for digito in range(1, 10)}


def salvar_frequencia_benford(frequencia_obtida, total_transacoes):
    digitos = list(range(1, 10))
    frequencia_esperada = [frequencia_esperada_benford(
        total_transacoes)[digito] for digito in digitos]
```

```
    frequencia_obtida_lista = [
        frequencia_obtida.get(digito, 0) for digito in digitos]

    # Criando o gráfico
    plt.figure(figsize=(10, 6))
    plt.bar(digitos, frequencia_obtida_lista, alpha=0.7,
            label='Frequência Obtida', color='blue')
    plt.plot(digitos, frequencia_esperada, color='red',
             marker='o', label='Lei de Benford', linewidth=2)

    plt.xlabel('Primeiro Dígito')
    plt.ylabel('Frequência')
    plt.title('Distribuição da Lei de Benford vs Frequência Obtida')
    plt.xticks(digitos)
    plt.legend()
    plt.grid(True)

    # Salvando o gráfico em um arquivo
    plt.savefig("benford.svg")
    plt.close()


def teste_qui_quadrado(frequencia_obtida, total_transacoes):
    frequencia_esperada = [frequencia_esperada_benford(
        total_transacoes)[digito] for digito in range(1, 10)]
    frequencia_obtida_lista = [frequencia_obtida.get(
        digito, 0) for digito in range(1, 10)]

    qui_quadrado, p_valor = chisquare(
        frequencia_obtida_lista, frequencia_esperada)

    return qui_quadrado, p_valor


def analisar_benford(transacoes):
    frequencia_obtida = calcular_frequencia_benford(transacoes)
    total_transacoes = sum(frequencia_obtida.values())

    # Salvando o gráfico
    salvar_frequencia_benford(frequencia_obtida, total_transacoes)
```

```
    # Realizando o teste Qui-Quadrado
    qui_quadrado, p_valor = teste_qui_quadrado(
        frequencia_obtida, total_transacoes)

    # Comparação de valores obtidos e esperados
    frequencia_esperada = frequencia_esperada_benford(total_transacoes)
    comparacao = {digito: {"obtida": frequencia_obtida.get(
        digito, 0), "esperada": frequencia_esperada[digito]} for digito in range(1, 10)}

    return comparacao, qui_quadrado, p_valor


def imprime_resultados_benford(comparacao, qui_quadrado, p_valor):

    esperado = []
    obtido = []

    for _, valores in comparacao.items():
        esperado.append(valores['esperada'])
        obtido.append(valores['obtida'])

    print(f'Esperado: {esperado}')
    print(f'Obtido: {obtido}')

    print(f'Qui-Quadrado: {qui_quadrado}')
    print(f'P-Valor: {p_valor}')


def main():
    base_path = 'rawaddr/'

    data_enderecos = ler_arquivo(base_path)

    # print(enderecos)

    clusters = clusterizar_enderecos(data_enderecos)

    # print(len(clusters))

    cluster = clusters['1JHH1pmHujcVa1aXjRrA13BJ13iCfgfBqj']
    set_cluster = set(cluster)
    cluster = list(set_cluster)
```

```
    transacoes = []

    for endereco in data_enderecos:
        if endereco['address'] in cluster:
            transacoes.extend(endereco['txs'])

    historico_saldo = calculaHistoricoSaldo(transacoes, cluster)

    imprimir_historico_saldo(historico_saldo)

    plotar_grafico_linha(historico_saldo, 'historico_saldo')

    valores = valores_por_endereco(cluster, transacoes)

    valores_transacoes = list(valores.values())
    indice_gini = calcular_indice_gini(valores_transacoes)
    print(f'Índice de Gini: {indice_gini}')

    plota_curva_lorenz(valores_transacoes)

    resultados_benford = analisar_benford(transacoes)

    imprime_resultados_benford(*resultados_benford)


if __name__ == '__main__':
    main()
```