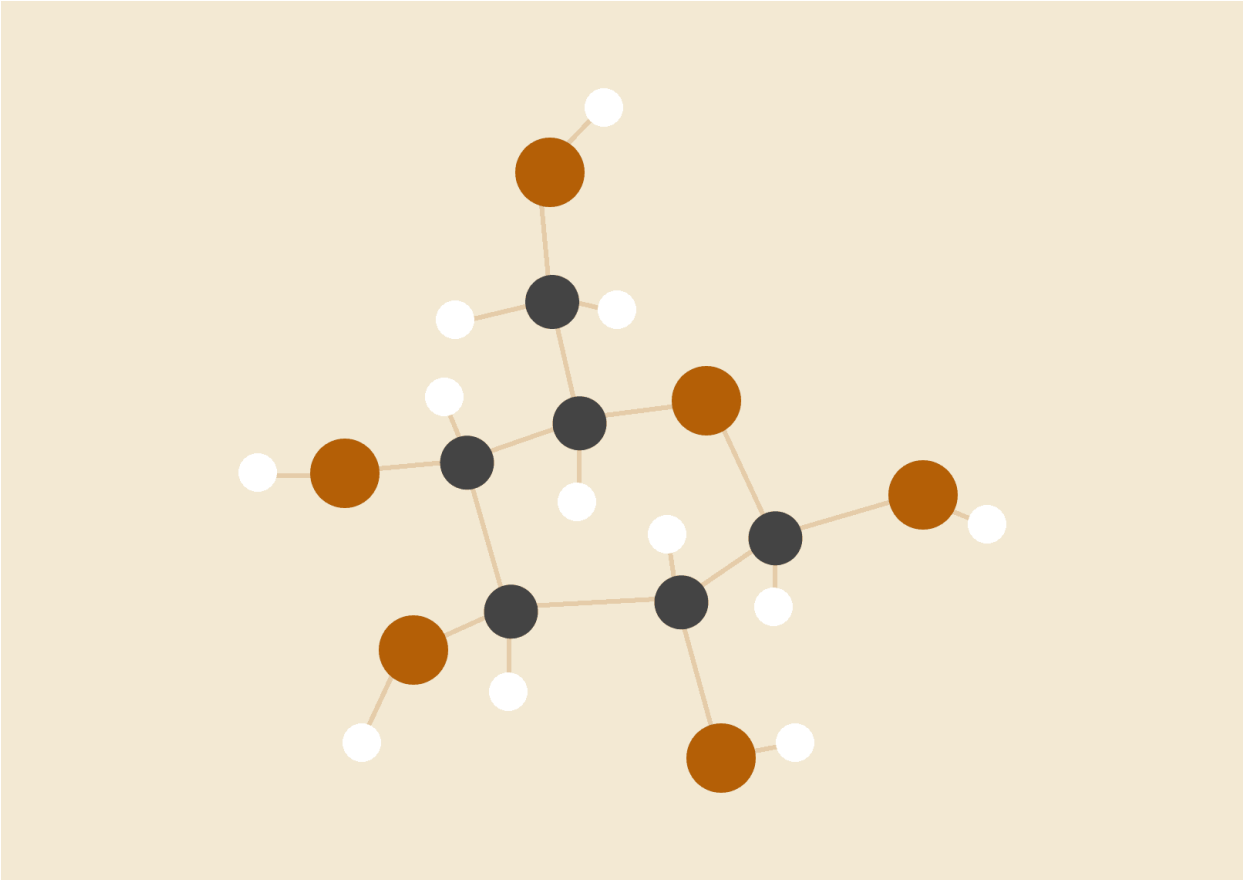
Relatório Trabalho 2



Vitor Yuji F. Matsushita (11921BCC021)

17/11/2023

Análise de Algoritmos

INTRODUÇÃO

Implementar um algoritmo para solucionar o problema de:

Dada uma matriz de inteiros, com cada linha ordenada e cada coluna ordenada, desenvolver um algoritmo para saber se um inteiro k pertence à matriz.

HIPÓTESE 1

- Primeira solução: Simples verificação de cada elemento na matriz (complexidade O(n^2))

```
import random

def check_in_matriz(matriz, k):
    for i in range(len(matriz)):
        for j in range(len(matriz[0])):
            if matriz[i][j] == k:
                return True
    return False

# Define o tamanho da matriz
linhas = 100
colunas = 100
```

```
# Cria a matriz e preenche
matriz = []
for i in range(200):
    matriz.append(sorted(random.sample(range(0, 10000), 200)))


# Imprime a matriz
for linha in matriz:
    print(linha)
print("Digite o valor a ser encontrado: ")
k = int(input())


print(check_in_matriz(matriz,k))
```

SAÍDA

```
...[321, 371, 393, 399, 429, 463, 532, 574, 580, 581, 597, 656, 751, 805, 810, 846, 966, 989,
1035, 1044, 1064, 1116, 1195, 1219, 1278, 1437, 1438, 1447, 1460, 1477, 1668, 1687, 1690,
1704, 1736, 1739, 1832, 1916, 1936, 1937, 1964, 1979, 2011, 2141, 2153, 2195, 2300, 2318,
2395, 2436, 2491, 2574, 2577, 2605, 2625, 2759, 2817, 2824, 2999, 3076, 3082, 3087, 3173,
3178, 3209, 3211, 3283, 3327, 3336, 3361, 3441, 3542, 3551, 3600, 3621, 3622, 3684, 3788,
3863, 3929, 3957, 3971, 4139, 4207, 4230, 4241, 4250, 4287, 4296, 4317, 4332, 4413, 4420,
4494, 4499, 4536, 4555, 4558, 4571, 4708, 4712, 4719, 4837, 4897, 4934, 4957, 4994, 5017,
5279, 5299, 5370, 5414, 5497, 5514, 5552, 5622, 5661, 5711, 5757, 5777, 5931, 5935, 5944,
5996, 6031, 6035, 6053, 6111, 6113, 6120, 6127, 6148, 6155, 6174, 6329, 6377, 6431, 6439,
6488, 6541, 6576, 6615, 6637, 6655, 6752, 6770, 6810, 6917, 6994, 7039, 7058, 7060, 7172,
7203, 7236, 7377, 7381, 7449, 7454, 7466, 7570, 7640, 7701, 7732, 7824, 7869, 7896, 7916,
8074, 8166, 8238, 8248, 8268, 8274, 8298, 8503, 8741, 8845, 8930, 8991, 9008, 9037, 9038,
9104, 9163, 9174, 9181, 9239, 9261, 9387, 9429, 9539, 9592, 9678, 9699, 9744, 9830, 9840,
9906, 9937]
```

Digite o valor a ser encontrado:

850

True

HIPÓTESE 2

- Segunda solução: Verificação mais eficiente (complexidade O(n)):
- Se o elemento na parte inferior direita for maior que o elemento que estamos procurando, o algoritmo recorta a linha inferior e tenta novamente. Se o elemento na parte inferior direita for menor que o elemento que estamos procurando, o algoritmo recorta a coluna da direita e tenta novamente.
- Dessa forma, o algoritmo é capaz de determinar se um elemento k pertence à matriz ordenada em tempo O(log(m*n)), onde m e n são as dimensões da matriz.
-

```
import random

import time


inicio = time.time()

def check_matriz(matriz, k):
    n = len(matriz)
    m = len(matriz[0])

    linha = 0
    coluna = m - 1

    while linha < n and coluna >= 0:
        if matriz[linha][coluna] == k:
            return True
        elif matriz[linha][coluna] > k:
            coluna -= 1
```

```
        else:
            linha += 1

    return False


linhas = 200
# Cria a matriz e preenche
matriz = []
for i in range(200):
    matriz.append(sorted(random.sample(range(0, 10000), 200)))

# Imprime a matriz
for linha in matriz:
    print(linha)

print("Digite o valor a ser encontrado: ")
k = int(input())


print(check_matriz(matriz,k))

fim = time.time()
print("Tempo de execução = ",fim-inicio)
```

SAÍDA

```
...
[46, 74, 101, 146, 150, 249, 264, 421, 422, 568, 614, 617, 660, 688, 782, 797, 948, 958, 982,
1053, 1081, 1423, 1503, 1521, 1553, 1584, 1687, 1692, 1755, 1816, 1822, 1889, 1910, 1937,
2030, 2086, 2174, 2177, 2240, 2317, 2351, 2364, 2371, 2381, 2474, 2502, 2593, 2601, 2623,
2649, 2660, 2677, 2689, 2693, 2714, 2738, 2754, 2798, 2804, 2852, 2917, 2983, 3003, 3036,
3073, 3318, 3319, 3329, 3357, 3419, 3481, 3519, 3524, 3530, 3533, 3551, 3571, 3624, 3645,
3703, 3795, 3899, 3912, 4039, 4099, 4228, 4257, 4305, 4345, 4398, 4450, 4458, 4639, 4716,
4751, 4781, 4792, 4809, 4819, 4901, 4941, 4945, 4965, 5087, 5098, 5106, 5129, 5152, 5157,
5227, 5334, 5337, 5417, 5535, 5536, 5588, 5589, 5615, 5625, 5685, 5687, 5734, 5794, 5849,
5904, 6040, 6116, 6167, 6225, 6268, 6391, 6393, 6396, 6441, 6498, 6545, 6730, 6794, 6810,
6879, 6907, 6922, 7144, 7155, 7198, 7227, 7278, 7343, 7350, 7381, 7406, 7424, 7458, 7520,
7670, 7675, 7749, 7807, 7887, 7969, 8052, 8080, 8089, 8178, 8244, 8249, 8303, 8305, 8360,
8370, 8413, 8449, 8463, 8555, 8562, 8584, 8605, 8791, 8799, 8834, 8913, 9057, 9109, 9270,
9277, 9316, 9319, 9426, 9428, 9544, 9573, 9725, 9780, 9853, 9874, 9875, 9896, 9927, 9951,
```

```
9953]
Digite o valor a ser encontrado:
8178
True
Tempo de execução = 4.867913722991943
```

HIPÓTESE 3

- O algoritmo é bastante eficiente, pois possui complexidade de tempo O(log(nm)) na média, onde n e m são o número de linhas e colunas da matriz, respectivamente.

```python
import random

def busca_binaria(vet, esq, dir, k):
    if dir >= esq:
        mid = (dir + esq) // 2
        if vet[mid] == k:
            return True
        elif vet[mid] > k:
            return busca_binaria(vet, esq, mid - 1, k)
        else:
            return busca_binaria(vet, mid + 1, dir, k)
    else:
        return False

def search(mat, n, m, k):
    esq = 0
    dir = n * m - 1
    while esq <= dir:
        mid = (dir + esq) // 2
        if mat[mid // m][mid % m] <= k:
            esq = mid + 1
        else:
            dir = mid - 1

    i = 0
    j = 0
    while i < n and j < m:
        if mat[i][j] <= k:
```

```
            j += 1
        else:
            i += 1

    if i < n:
        if busca_binaria(mat[i], 0, m - 1, k):
            return True

    if j < m:
        if busca_binaria(mat[i - 1], 0, m - 1, k):
            return True

    return False


# Define o tamanho da matriz
linhas = 200

# Cria a matriz e preenche
matriz = []
for i in range(200):
    matriz.append(sorted(random.sample(range(0, 10000), 200)))

# Imprime a matriz
for linha in matriz:
    print(linha)

print("Digite o valor a ser encontrado: ")
k = int(input())


print(search(matriz,linhas,colunas,k))
```

SAÍDA

```
...
[144, 198, 277, 278, 308, 366, 388, 404, 468, 518, 529, 613, 653, 681, 684, 700, 749, 755,
802, 884, 962, 1027, 1068, 1082, 1086, 1137, 1161, 1173, 1219, 1224, 1226, 1413, 1419,
1434, 1436, 1453, 1463, 1508, 1541, 1543, 1623, 1673, 1675, 1843, 1851, 1907, 1939, 1977,
2053, 2058, 2206, 2221, 2282, 2302, 2329, 2377, 2431, 2463, 2468, 2490, 2580, 2644, 2793,
2880, 2967, 3032, 3056, 3080, 3086, 3104, 3105, 3174, 3246, 3247, 3252, 3260, 3335, 3468,
3579, 3639, 3759, 3768, 3769, 3901, 3938, 3952, 3976, 4022, 4102, 4129, 4132, 4162, 4164,
4262, 4379, 4598, 4644, 4702, 4828, 4885, 4887, 4946, 4958, 4985, 4997, 5028, 5045, 5075,
```

```
5141, 5147, 5174, 5198, 5224, 5291, 5365, 5372, 5481, 5495, 5752, 5840, 5874, 5901, 5965,
5984, 6002, 6036, 6180, 6190, 6259, 6282, 6353, 6468, 6514, 6531, 6538, 6623, 6742, 6813,
6828, 7050, 7063, 7069, 7143, 7189, 7232, 7254, 7263, 7267, 7280, 7407, 7480, 7501, 7523,
7579, 7609, 7632, 7670, 7694, 7699, 7703, 7778, 7795, 7903, 7918, 7970, 8083, 8107, 8133,
8155, 8164, 8207, 8226, 8285, 8566, 8636, 8693, 8905, 8928, 8939, 8984, 8998, 9059, 9103,
9145, 9162, 9209, 9289, 9381, 9475, 9618, 9715, 9772, 9776, 9783, 9784, 9848, 9883, 9888,
9893, 9908]
Digite o valor a ser encontrado:
900
False
```

INTRODUÇÃO

Implementar um algoritmo para solucionar o problema de:

- Dado um vetor de strings, determine o prefixo comum máximo.

HIPÓTESE 1

- Forma Direta

```python
import time


inicio = time.time()

def max_prefixo(vetor):
    prefixo = vetor[0]
    for string in vetor[1:]:
        while string[:len(prefixo)] != prefixo and prefixo:
            prefixo = prefixo[:len(prefixo)-1]
```

```
        if not prefixo:
            break
    return prefixo


vetor = ["abacateiro"
,"abacateiroso"
,"abacateirosofia"
,"abacateirosofialidade"
,"abacateirosofialidadedos"
,"abacateirosofialidadedosolhos"
,"abacateirosofialidadedosolhosverdes"
,"abacateirosofialidadedosolhosverdesde"
,"abacateirosofialidadedosolhosverdesdeum"
,"abacateirosofialidadedosolhosverdesdeumgato"]

print("O maior prefixo comum é ", max_prefixo(vetor))

fim = time.time()
print("Tempo de execução = ",fim-inicio)
```

SAÍDA

```
O maior prefixo comum é  abacateiro
Tempo de execução =  0.0009331703186035156
```

HIPÓTESE 2

- Três funções para encontrar o prefixo comum máximo. A função “max_prefixo_DC” usa a técnica de Divisão e Conquista. A função usa um vetor para armazenar o prefixo comum máximo de substrings até a posição i.

```
def max_prefixo_DC(vet, esq, dir):
    if esq == dir:
        return vet[esq]
```

```
    mid = (esq + dir) // 2
    return max_prefixo_DC(vet, esq, mid) + max_prefixo_DC(vet, mid + 1,
dir)

def max_prefixo_DC(vet, n):
    if n == 0:
        return ""
    prefixo = [0] * (n + 1)
    len1 = len(vet[0])
    prefixo[1] = len1
    x = prefixo[1]
    for i in range(2, n + 1):
        len1 = len(vet[i - 1])
        len2 = len(vet[i - 2])
        count = 0
        while (count < len1 and count < len2):
            if (vet[i - 1][count] != vet[i - 2][count]):
                break
            count += 1
        prefixo[i] = count
        x = min(x, prefixo[i])
    return vet[0][:x]


vet = ["flagra", "flacido", "flanela", "flagelo", "flatulência",
"flamingo", "flanco", "flauta", "flamenguista"]
n = len(vet)
print("Prefixo: ", max_prefixo_DC(vet, n))
##Saida = fla
```

SAÍDA

```
Prefixo:  fla
Tempo de execução =  0.0015189647674560547
```

INTRODUÇÃO

Implementar um algoritmo para solucionar o problema de:

- (Target Sum) Dado um vetor de inteiros e um “alvo” K, pergunta-se de quantos formas diferentes todos elementos do vetor podem ser somados ou subtraídos para somarem k. Por exemplo, com $V=[1,2,1,3]$ e $k=1$, temos 1 = +1 -2 -1 +3 = -1 -2 +1 +3= +1 -2 -1 +3 = ...

HIPÓTESE 1

- Algoritmo com complexidade de O(2^N)

```python
from random import randint



def target_sum(V, k):
    count = 0
    subset = []

    def subvet(i, k1):
        nonlocal count
        if k1 == k:
            count += 1
            subset.append(list(numbers))
        if k1 < k and i < len(V):
            numbers.append(V[i])
            subvet(i + 1, k1 + V[i])
            numbers.pop()
            subvet(i + 1, k1 - V[i])

    numbers = []
    subvet(0, 0)

    return count


# Example usage:
```

```
v=[]
for i in range(100):
        random_number = randint(0, 10)
        v.append(random_number)
print(v)
print("Digite o valor de k: ")
k = int(input())
print(target_sum(v, k))
```

SAÍDA

```
[7, 3, 8, 10, 3, 0, 1, 6, 0, 5, 4, 10, 8, 6, 1, 7, 4, 2, 4, 2, 10, 0, 4, 2, 3]
Digite o valor de k:
2
50866
Tempo de execução =  6.38218355178833


Obs: Acima de 25 elementos, demora de mais
```

HIPÓTESE 2

- Técnica de Divisão e Conquista utiliza uma tabela para calcular a quantidade de combinações diferentes possíveis de elementos do vetor. A tabela R armazena sub-soluções parciais do problema. O algoritmo percorre cada elemento do vetor e calcula o número de combinações diferentes que somam k.

```
from random import randint
import time

inicio = time.time()
def target_sum(V, K):
```

```
    # Criar tabela para armazenar sub-soluções
    R = [[0 for _ in range(K+1)] for _ in range(len(V)+1)]

    # Ao inicializar a tabela, todos os subconjuntos vazios tem a soma 0
    for i in range(len(V)+1):
        R[i][0] = 1

    # Iterar por cada elemento do vetor e calcular sub-soluções
    for i in range(1, len(V)+1):
        for j in range(1, K+1):
            if V[i-1] <= j:
                # Se o elemento atual for menor ou igual ao valor j,
então escolher incluí-lo
                R[i][j] = R[i-1][j-V[i-1]] + R[i-1][j]
            else:
                R[i][j] = R[i-1][j]

    # Retornar o número de combinações possíveis
    return R[-1][-1]


v=[]
for i in range(1000):
        random_number = randint(0, 10)
        v.append(random_number)
print(v)
print("Digite o valor de k: ")
k = int(input())
print(target_sum(v, k))

fim = time.time()
print("Tempo de execução = ",fim-inicio)
```

SAÍDA

```
[1, 1, 4, 3, 2, 4, 4, 1, 10, 1, 6, 2, 2, 3, 2, 3, 6, 3, 5, 5, 10, 7, 10, 8, 9, 4, 9, 1, 7, 2, 10, 6, 8, 5, 8, 4,
1, 6, 0, 0, 5, 1, 2, 10, 3, 9, 5, 6, 10, 10, 5, 1, 1, 7, 8, 3, 2, 7, 9, 1, 5, 5, 10, 5, 8, 3, 9, 6, 9, 6, 6, 4,
10, 5, 4, 6, 9, 3, 9, 10, 8, 5, 6, 5, 9, 1, 5, 1, 1, 7, 9, 8, 1, 0, 10, 2, 9, 3, 0, 6, 10, 3, 4, 3, 7, 8, 2, 5, 9,
4, 4, 1, 3, 8, 6, 4, 3, 4, 0, 3, 8, 0, 7, 6, 8, 1, 3, 7, 10, 8, 3, 1, 1, 5, 6, 9, 5, 1, 8, 3, 3, 6, 6, 7, 4, 0, 10,
3, 4, 8, 10, 5, 5, 10, 3, 7, 8, 8, 0, 1, 7, 9, 2, 9, 7, 5, 2, 9, 10, 1, 2, 8, 4, 4, 2, 1, 7, 1, 1, 2, 6, 2, 8, 0,
4, 10, 3, 9, 5, 4, 7, 4, 0, 10, 0, 8, 8, 9, 3, 5, 5, 0, 3, 9, 5, 5, 3, 5, 2, 0, 0, 3, 5, 9, 1, 6, 3, 2, 9, 9, 8, 6,
6, 6, 7, 10, 8, 10, 7, 1, 1, 9, 6, 9, 7, 3, 4, 8, 0, 2, 1, 6, 7, 2, 7, 9, 3, 5, 3, 6, 4, 1, 1, 7, 5, 9, 0, 4, 7, 2,
9, 9, 6, 8, 10, 3, 0, 5, 1, 5, 8, 0, 7, 5, 1, 2, 0, 9, 6, 8, 10, 10, 2, 8, 5, 7, 2, 7, 7, 0, 7, 10, 10, 3, 9, 1,
9, 9, 9, 4, 6, 4, 10, 1, 1, 10, 1, 10, 7, 3, 0, 6, 10, 9, 8, 3, 6, 3, 6, 9, 2, 9, 0, 2, 1, 7, 8, 2, 4, 2, 0, 5, 3,
1, 5, 4, 5, 1, 0, 5, 5, 1, 10, 5, 9, 10, 4, 6, 10, 10, 2, 10, 7, 2, 5, 3, 6, 9, 3, 5, 3, 2, 2, 1, 3, 8, 7, 6, 9,
```

10, 3, 10, 2, 1, 4, 3, 2, 9, 6, 3, 9, 10, 5, 3, 1, 6, 7, 10, 5, 7, 1, 0, 4, 7, 10, 2, 8, 3, 3, 1, 2, 10, 7, 2, 5, 7, 6, 7, 9, 4, 10, 0, 9, 10, 9, 3, 0, 1, 4, 4, 5, 0, 4, 6, 4, 0, 6, 10, 3, 4, 9, 2, 0, 4, 9, 9, 3, 0, 4, 6, 0, 5, 9, 0, 1, 3, 3, 1, 5, 6, 8, 4, 7, 4, 6, 0, 8, 0, 1, 4, 0, 3, 7, 6, 8, 5, 9, 2, 5, 5, 0, 3, 4, 2, 10, 3, 8, 8, 4, 10, 2, 3, 8, 0, 6, 2, 3, 3, 4, 5, 0, 0, 7, 9, 3, 0, 0, 4, 4, 8, 10, 5, 6, 1, 2, 10, 6, 5, 3, 5, 8, 2, 6, 6, 6, 1, 2, 1, 4, 5, 3, 7, 1, 9, 4, 3, 4, 8, 1, 1, 2, 10, 8, 6, 2, 1, 0, 0, 9, 2, 4, 9, 9, 0, 10, 3, 5, 3, 6, 6, 5, 1, 4, 6, 8, 10, 0, 9, 10, 7, 2, 3, 3, 3, 1, 2, 1, 9, 0, 3, 9, 4, 10, 1, 4, 3, 1, 6, 6, 5, 6, 10, 1, 3, 3, 5, 10, 9, 6, 3, 10, 3, 1, 2, 4, 7, 2, 1, 3, 3, 5, 7, 5, 4, 0, 7, 6, 0, 1, 3, 5, 2, 1, 4, 0, 0, 5, 8, 4, 7, 10, 0, 6, 1, 3, 1, 3, 3, 9, 2, 10, 4, 8, 5, 4, 4, 2, 7, 1, 6, 9, 4, 2, 1, 5, 5, 2, 2, 7, 5, 3, 2, 5, 0, 6, 5, 9, 7, 9, 6, 9, 6, 9, 4, 2, 10, 1, 3, 1, 0, 0, 9, 8, 9, 7, 10, 2, 4, 7, 7, 3, 1, 1, 0, 10, 7, 6, 0, 6, 7, 8, 9, 3, 9, 1, 3, 6, 8, 4, 6, 1, 7, 1, 9, 4, 1, 3, 0, 3, 5, 1, 4, 0, 7, 4, 1, 0, 7, 10, 6, 1, 9, 7, 5, 0, 0, 6, 7, 10, 3, 1, 6, 3, 6, 4, 0, 5, 8, 4, 8, 1, 9, 1, 10, 6, 2, 0, 9, 5, 5, 8, 5, 5, 7, 6, 4, 4, 5, 3, 9, 7, 6, 0, 6, 2, 6, 0, 6, 4, 3, 1, 5, 1, 7, 4, 5, 8, 4, 0, 2, 6, 0, 7, 3, 8, 8, 0, 3, 9, 1, 6, 3, 2, 6, 5, 6, 4, 6, 0, 6, 10, 3, 8, 4, 4, 5, 5, 7, 8, 0, 1, 6, 4, 3, 6, 6, 9, 5, 3, 9, 10, 9, 0, 10, 0, 5, 6, 2, 8, 3, 7, 4, 8, 1, 3, 7, 8, 0, 6, 9, 0, 4, 0, 3, 5, 10, 8, 4, 8, 6, 0, 10, 8, 6, 3, 9, 8, 3, 1, 9, 0, 8, 2, 4, 8, 5, 2, 10, 9, 5, 5, 10, 0, 1, 8, 2, 7, 7, 7, 2, 10, 5, 7, 8, 3, 10, 8, 0, 1, 8, 7, 2, 3, 10, 6, 6, 8, 5, 4, 1, 1, 10, 9, 3, 9, 10, 5, 0, 10, 7, 3, 3, 8, 3, 6, 5, 9, 9, 10, 2, 6, 7, 4, 7, 9, 6, 0, 10, 8, 9, 10, 3, 8, 8, 4, 3, 6, 6, 2, 3, 5, 2, 8, 3, 6, 7, 1, 3, 10, 2, 3, 8, 6, 10, 7, 2, 2, 8, 6, 6, 5, 4, 2, 10, 4, 10, 4, 7, 4, 4, 0, 1, 2, 1, 8, 6, 4, 6, 0, 9, 0, 3, 0, 9, 0, 7, 0, 1, 7, 7, 9, 10, 6, 1, 5, 10, 6, 2, 10, 6]
Digite o valor de k:
2
237554450769298245010204129986
Tempo de execução = 1.915