Trabalho 1 – Análise de Algoritmos

Componentes do grupo

Nome: Kauan Toledo Camargo matrícula 0051352311031
Nome: Henrique Rodrigues de Freitas matrícula 0051352311010
Nome: _____ matrícula _____
Nome: _____ matrícula _____

Instruções

- data de referência para entrega: 10/04.
- em grupos de até 4 alunos.
- considerar no trabalho o *print do* código fonte e o *print* das simulações;
- nomear o arquivo como "**Trabalho 1 de Análise de Algoritmos + 1º nome de um dos componentes do grupo**";
- entregar o trabalho em arquivo PDF pelo *email* do professor: mauricio.mario@fatec.sp.gov.br
assunto = **trabalho 1 de Análise de Algoritmos**.

Exercício 1:

Demonstrar a relação entre os tempos de execução do produto e da soma entre as matrizes A e B; se necessário executar n vezes a parte do algoritmo que executa a operação dominante em cada operação, afim de que seja possível quantificar os tempos de execução.

Matriz A + Matriz B

Matriz A * Matriz B

$$A = \begin{pmatrix} 2 & -1 & 3 \\ 0 & 4 & -6 \\ -6 & 10 & -5 \end{pmatrix} \quad B = \begin{pmatrix} 4 & 7 & -8 \\ 9 & 3 & 5 \\ 1 & -1 & 2 \end{pmatrix}$$

Tempo execução 1 Tempo execução 2

Relação = Tempo execução 2 ÷ Tempo execução 1

→ *Demonstrar qual a relação entre os tempos de execução em função de* m_{linhas} e n_{colunas}. *Justificar os resultados em função do tamanho das entradas e da quantidade de execuções da operação dominante (passos).*

código fonte:

```
import time

A = [[2,-1,3],[0,4,-6],[-6,10,-5]]
B = [[4,7,-8],[9,3,5],[1,-1,2]]
S = [[0,0,0],[0,0,0],[0,0,0]]

temp_start = time.time()

for k in range(1):
  for i in range(3):
    for j in range(3):
      S[i][j] = 0
      S[i][j] = A[i][j] + B[i][j]

temp_finish = time.time()

print(S)
print(f'Tempo de Processamento: {temp_finish - temp_start}')

temp_start = time.time()

for k in range(1):
  for i in range(3):
    for j in range(3):
      S[i][j] = 0
      for l in range(3):
        S[i][j] = S[i][j] + A[i][l] * B[l][j]

temp_finish = time.time()

print(S)
print(f'Tempo de Processamento: {temp_finish - temp_start}')
```

resultados da execução no console + conclusões sobre os diferentes tempos de execução:

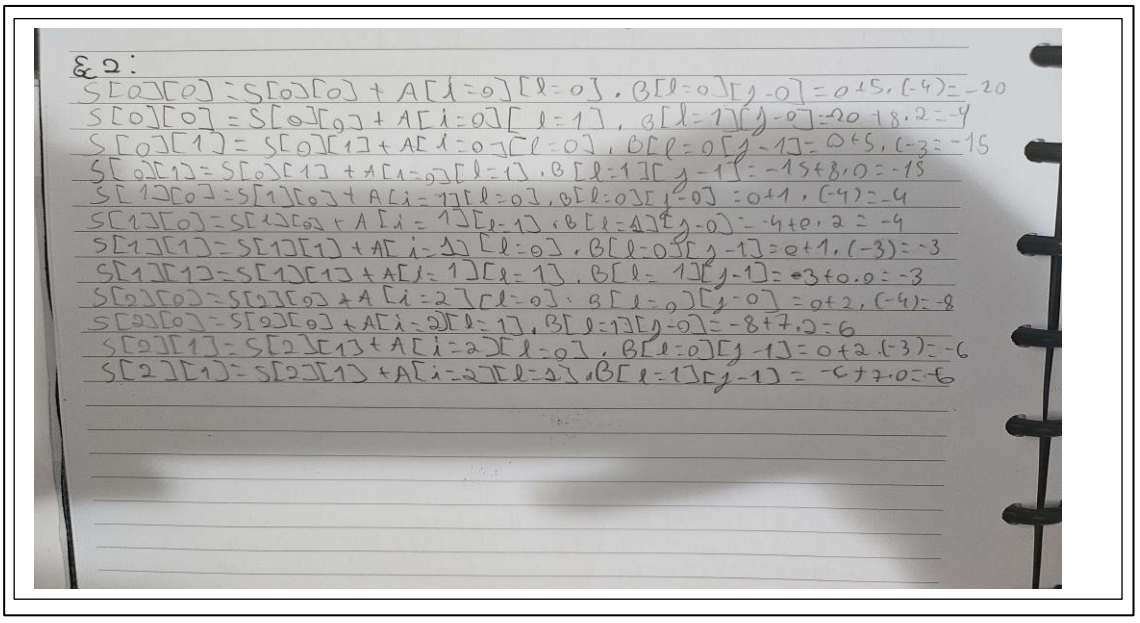
```
[[6, 6, -5], [9, 7, -1], [-5, 9, -3]]
Tempo de Processamento: 0.26236605644226074
[[2, 8, -15], [30, 18, 8], [61, -7, 88]]
Tempo de Processamento: 0.8445768356323242
```

Exercício 2 - Para o trecho de código *Python* abaixo, simular a execução do algoritmo para i e j variando conforme o laço for aninhado. Escrever os valores finais de C para cada caso.

```
A = [[5, 8], [1, 0], [2, 7]]
B = [[-4, -3], [2, 0]]

C =  [[0, 0], [0, 0], [0, 0]]
for i in range(0, 3):
    for j in range(0, 2):
       C[i][j] = 0
       for k in range(0, 2):
            C[i][j] =  C[i][j] + A[i][k]*B[k][j]
```

resultados das simulações:



Exercício 3:

1. Aplicar o algoritmo de ordenação por inserção em uma lista de 50 elementos ordenados em sequência crescente;

2. Aplicar o algoritmo de ordenação por inserção em uma lista de 50 elementos ordenados em sequência decrescente; comparar os tempos de execução com o exemplo do exercício 1. Concluir.

3. Aplicar o algoritmo de ordenação por inserção em uma lista de 50 elementos em sequência aleatória; comparar os tempos de execução com os exemplos dos exercícios 1 e 2. Concluir.

4. Aplicar o algoritmo de ordenação por inserção em uma lista de 100 elementos ordenados em sequência crescente; comparar os tempos de execução com os exemplos dos exercícios 1 e 2. Concluir.

5. Aplicar o algoritmo de ordenação por inserção em uma lista de 100 elementos ordenados em sequência decrescente; comparar os tempos de execução com os exemplos dos exercícios 1, 2, 3 e 4. Concluir.

6. Aplicar o algoritmo de ordenação por inserção em uma lista de 100 elementos em sequência aleatória; comparar os tempos de execução com os exemplos dos exercícios 1, 2, 3, 4 e 5. Concluir.

código fonte:

```
import time
import random

A = list(range(1,51))

print('Matriz não ordenada = ', A)
print('Len = ', len(A))

temp_start = time.time()

for k in range(1000):
  for i in range(1,len(A)):
    temp = A[i]
    j = i -1
    while j >= 0 and A[j] > temp:
      A[j + 1] = A[j]
      j -= 1
    A[j + 1] = temp

temp_finish = time.time()

print('Matriz ordenada = ',A)
print('Tempo Execução = ', temp_finish - temp_start)
```

```
A = list(range(50,0,-1))

print('Matriz não ordenada = ', A)
print('Len = ', len(A))

temp_start = time.time()

for k in range(1000):
  for i in range(1,len(A)):
    temp = A[i]
    j = i - 1
    while j >= 0 and A[j] > temp:
      A[j + 1] = A[j]
      j -= 1
    A[j + 1] = temp

temp_finish = time.time()

print('Matriz ordenada = ',A)
print('Tempo Execução = ', temp_finish - temp_start)

A = list(range(1,51))
random.shuffle(A)

print('Matriz não ordenada = ', A)
print('Len = ', len(A))

temp_start = time.time()

for k in range(1000):
  for i in range(1,len(A)):
    temp = A[i]
    j = i - 1
    while j >= 0 and A[j] > temp:
      A[j + 1] = A[j]
      j -= 1
    A[j + 1] = temp

temp_finish = time.time()

print('Matriz ordenada = ',A)
print('Tempo Execução = ', temp_finish - temp_start)

A = list(range(1,101))

print('Matriz não ordenada = ', A)
print('Len = ', len(A))

temp_start = time.time()
```

```
for k in range(1000):
  for i in range(1,len(A)):
    temp = A[i]
    j = i -1
    while j >= 0 and A[j] > temp:
      A[j + 1] = A[j]
      j -= 1
    A[j + 1] = temp

temp_finish = time.time()

print('Matriz ordenada = ',A)
print('Tempo Execução = ', temp_finish - temp_start)

A = list(range(100,0,-1))

print('Matriz não ordenada = ', A)
print('Len = ', len(A))

temp_start = time.time()

for k in range(1000):
  for i in range(1,len(A)):
    temp = A[i]
    j = i - 1
    while j >= 0 and A[j] > temp:
      A[j + 1] = A[j]
      j -= 1
    A[j + 1] = temp

temp_finish = time.time()

print('Matriz ordenada = ',A)
print('Tempo Execução = ', temp_finish - temp_start)

A = list(range(1,101))
random.shuffle(A)

print('Matriz não ordenada = ', A)
print('Len = ', len(A))

temp_start = time.time()

for k in range(1000):
  for i in range(1,len(A)):
    temp = A[i]
    j = i - 1
    while j >= 0 and A[j] > temp:
      A[j + 1] = A[j]
      j -= 1
    A[j + 1] = temp
```

```
temp_finish = time.time()

print('Matriz ordenada = ',A)
print('Tempo Execução = ', temp_finish - temp_start)
```

resultados da execução no console + conclusões sobre os diferentes tempos de execução:

```
Matriz não ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50]
Len =  50
Matriz ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50]
Tempo Execução =  0.008572578430175781
Matriz não ordenada =  [50, 49, 48, 47, 46, 45, 44, 43, 42, 41,
40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25,
24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9,
8, 7, 6, 5, 4, 3, 2, 1]
Len =  50
Matriz ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50]
Tempo Execução =  0.008504152297973633
Matriz não ordenada =  [6, 36, 11, 31, 42, 48, 23, 21, 3, 4,
13, 28, 27, 25, 45, 16, 17, 49, 1, 41, 19, 7, 44, 39, 20, 46,
43, 33, 8, 12, 24, 40, 5, 34, 10, 14, 38, 22, 30, 15, 29, 37,
18, 2, 50, 47, 9, 26, 32, 35]
Len =  50
Matriz ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50]
Tempo Execução =  0.009791135787963867
Matriz não ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
93, 94, 95, 96, 97, 98, 99, 100]
Len =  100
```

```
Matriz ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100]
Tempo Execução =  0.017846107482910156
Matriz não ordenada =  [100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90,
89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73,
72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56,
55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39,
38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22,
21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3,
2, 1]
Len =  100
Matriz ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100]
Tempo Execução =  0.01859760284423828
Matriz não ordenada =  [6, 36, 62, 64, 23, 100, 90, 50, 42, 3, 17,
54, 53, 97, 72, 84, 44, 51, 28, 22, 93, 13, 96, 68, 24, 16, 14, 58,
15, 1, 67, 4, 48, 69, 80, 95, 21, 40, 88, 76, 33, 46, 38, 61, 78,
45, 2, 74, 18, 9, 11, 60, 12, 66, 20, 19, 55, 7, 82, 87, 65, 34, 52,
99, 79, 63, 77, 81, 47, 86, 29, 26, 91, 25, 75, 71, 39, 8, 83, 31,
41, 85, 27, 89, 35, 5, 49, 98, 32, 43, 59, 10, 57, 30, 56, 92, 73,
70, 94, 37]
Len =  100
Matriz ordenada =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100]
Tempo Execução =  0.017939805984449707
```

Exercício 4

Dada as listas A e B:

A = [0, “kombi”, “fusca”, “belina”, 4, “dodge” “gordini”, “tl”, 6, “variant”, “brasília”, “opala”, 7, “corcel”, “santana”, “quantum”, 11, “scort”, “rural”, “variante”, 23, “sp2”, “willis”, ”simca”, 44, “decavê”, “uno”, “veraneio”]

B = [21, “macaco”, “zebra”, “tartaruga”, “tubarão”, “formiga”, 55, “leão”, “elefante”, “girafa”, “coelho”, “pato”, 60, “gato”, “cobra”, “aranha”, “cabra”, “besouro”, 73, “borboleta”, “boi”, “tigre”, “leopardo”, “anta”, 101, “mula”, “burro”, “calango”, “lagartixa”, “sapo”, 202, “baleia”, “urso”, “hiena”, “rato”, “gorila”]

***Exercício* 1:** Retornar cada nó e os respectivos campos da lista A em chamadas individuais do algoritmo de busca sequencial, comparando os tempos de execução para cada caso.

***Exercício* 2:** Retornar cada nó e os respectivos campos da lista A em chamadas individuais do algoritmo de busca sequencial que adiciona incondicionalmente o nó procurado ao fim da lista, comparando os tempos de execução para cada caso.

***Exercício* 3:** Retornar cada nó e os respectivos campos da lista B em chamadas individuais do algoritmo de busca sequencial, comparando os tempos de execução para cada caso.

***Exercício* 4:** Retornar cada nó e os respectivos campos da lista B em chamadas individuais do algoritmo de busca sequencial que adiciona incondicionalmente o nó procurado ao fim da lista, comparando os tempos de execução para cada caso.

***Exercício* 5:** Comparar os tempos de execução do algoritmo de busca sequencial e busca sequencial que adiciona incondicionalmente o nó procurado ao fim da lista, aplicado à lista A.

***Exercício* 6:** Comparar os tempos de execução do algoritmo de busca sequencial e busca sequencial que adiciona incondicionalmente o nó procurado ao fim da lista, aplicado à lista B.

***Exercício* 7:** Comparar os tempos de execução do algoritmo de busca sequencial quando aplicado à lista A e à lista B.

***Exercício* 8:** Comparar os tempos de execução do algoritmo de busca sequencial que adiciona incondicionalmente o nó procurado ao fim da lista, quando aplicado à lista A e à lista B.

***Exercício* 9:** Justificar os resultados.

código fonte:

```
import time

A =
[0,'kombi','fusa','belina',4,'dodge','gordino','tl',6,'variant','brasília','opala',7,'corcel','santana','
quantum',11,'scort','rural','variante',23,'sp2','willis','simca',44,'decavê','uno','veraneio']

B = [21, "macaco", "zebra", "tartaruga", "tubarão", "formiga", 55, "leão", "elefante",
"girafa", "coelho", "pato", 60, "gato", "cobra", "aranha", "cabra", "besouro", 73,
"borboleta", "boi", "tigre", "leopardo", "anta", 101, "mula", "burro", "calango", "lagartixa",
"sapo", 202, "baleia", "urso", "hiena", "rato", "gorila"]

na = len(A)
nb = len(B)

def linear_search_a(x):
  for p in range(1):
    i=0
    while i < na:
      if A[i] == x:
        return A[i], A[i+1], A[i+2], A[i+3]
      else:
        i += 1

def linear_search_a_end(x):
  for p in range(1):
    i=0
    while i < na:
      if A[i] == x:
        return A[i+1], A[i+2], A[i+3], A[i]
      else:
        i += 1
```

```
def linear_search_b(x):
  for p in range(1):
    i=0
    while i < nb:
      if B[i] == x:
        return B[i],B[i+1],B[i+2],B[i+3],B[i+4],B[i+5]
      else:
        i += 1

def linear_search_b_end(x):
  for p in range(1):
    i=0
    while i < nb:
      if B[i] == x:
        return B[i+1], B[i+2], B[i+3], B[i]
      else:
        i += 1

#1
for i in [0,4,6,7,11,23,44]:
  start_time = time.time();
  for j in range(10000):
    linear_search_a(i)
  print(f'List A, Key ({i}) = {linear_search_a(i)}')
  zawarudo = time.time()
  print(f'tempo de execução= {zawarudo-start_time}')

print('\n')
#2
for i in [0,4,6,7,11,23,44]:
  start_time = time.time();
  for j in range(10000):
    linear_search_a_end(i)
  print(f'List A end, Key ({i}) = {linear_search_a_end(i)}')
  zawarudo = time.time()
  print(f'tempo de execução= {zawarudo-start_time}')

print('\n')
#3
for i in [21,55,60,73,101,202]:
  start_time = time.time();
  for j in range(10000):
    linear_search_b(i)
  print(f'List B, Key ({i}) = {linear_search_b(i)}')
  zawarudo = time.time()
  print(f'tempo de execução= {zawarudo-start_time}')

print('\n')
#4
for i in [21,55,60,73,101,202]:
  start_time = time.time();
  for j in range(10000):
    linear_search_b_end(i)
  print(f'List B end, Key ({i}) = {linear_search_b_end(i)}')
  zawarudo = time.time()
  print(f'tempo de execução= {zawarudo-start_time}')
```

Exercício 5: Dadas as listas:

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

Letras = ["A", "B", "C", "D", "F", "G", "H", "I", "J", "K", "L"]
```

Alterar o algoritmo de busca binária para que o mesmo possa retornar além dos nós procurados, os campos presentes em lista indexada.

Executar o algoritmo de busca para cada nó individualmente, verificando se o tempo de busca aumenta à medida que os nós estão mais próximos do início ou do fim da lista. Se os tempos medidos forem desprezíveis, multiplicar o número de buscas para cada nó.

```
import time
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Letras = ["A", "B", "C", "D", "E", "F", "G", "H", "I","J","K","L"]
n = len(L)
print ("Dimensão da lista L = ", n)
print("\n")
def busca_bin(x):
  for p in range(1):
    posição_inferior = 0
    posição_superior = n
    while posição_inferior <= posição_superior:
      meio = int((posição_inferior + posição_superior)/2)
      if meio < n:
        if L[meio] == x:
          return L[meio], Letras[meio]
          posição_inferior = posição_superior + 1
        else:
          if L[meio]< x:
            posição_inferior = meio + 1

          else:
            if L[meio]> x:
              posição_superior = meio - 1

      else:
        return "Chave não encontrada"

tempo_inicial = time.time()
print("Chave (5), campo da chave = ", busca_bin(5))
print("Chave (1), campo da chave = ", busca_bin(1))
print("Chave (4), campo da chave = ", busca_bin(4))
print("Chave (3), campo da chave = ", busca_bin(3))
print("Chave (2), campo da chave = ", busca_bin(2))
print("Chave (0), campo da chave = ", busca_bin(0))
print("Chave (6), campo da chave = ", busca_bin(6))
print("Chave (7), campo da chave = ", busca_bin(7))
print("Chave (10), campo da chave = ", busca_bin(10))
print("Chave (9), campo da chave = ", busca_bin(9))
print("Chave (8), campo da chave = ", busca_bin(8))
print("Chave (20), campo da chave = ", busca_bin(20))
tempo_final = time.time()
print("Templo de execução = ", tempo_final - tempo_inicial)
```

```
Dimensão da lista L =  10


Chave (5), campo da chave =  (5, 'E')
Chave (1), campo da chave =  (1, 'A')
Chave (4), campo da chave =  (4, 'D')
Chave (3), campo da chave =  (3, 'C')
Chave (2), campo da chave =  (2, 'B')
Chave (0), campo da chave =  None
Chave (6), campo da chave =  (6, 'F')
Chave (7), campo da chave =  (7, 'G')
Chave (10), campo da chave =  (10, 'J')
Chave (9), campo da chave =  (9, 'I')
Chave (8), campo da chave =  (8, 'H')
Chave (20), campo da chave =  Chave não encontrada
Templo de execução =  0.0010278224945068 36
```