Trabalho 2 – Análise de Algoritmos

Componentes do grupo
Nome: Henrique Rodrigues de Freitas | Matrícula: 0051352311010

Nome: Kauan Toledo Camargo | Matrícula: 0051352311031

Instruções
- data de referência para entrega: 12/06.
- em grupos de até 4 alunos.
- considerar no trabalho o *print do* código fonte e o *print* das simulações;
- nomear o arquivo como "**Trabalho 2 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 2 de Análise de Algoritmos**.

Exercício 1 – Criar uma lista com 10 elementos no formato:

ponteiro anterior	nº nó	"nome do funcionário"	"R$ salário"	ponteiro próximo

-Criar estrutura de dados (objetos) para a lista. Definir a chave da lista e os outros objetos (ponteiros, dados);
-Utilizando o algoritmo de inserção, inserir um 11° elemento na lista, atualizando os ponteiros dos elementos da mesma;
- Utilizando o algoritmo de remoção, remover um elemento da lista, atualizando os ponteiros;
- Executar o algoritmo de busca de nó na lista para cada um dos 10 elementos individualmente, quantificar o tempo de execução para cada um dos elementos, e concluir de acordo com a posição de cada elemento na lista;

Exercício 2

Criaruma árvore binária que contextualize em seus nós os bairros da cidade de Santos. Desenhar a árvore, com os nós, ponteiros e bairros. Executar o algoritmo de percurso dos nós, quantificando o tempo de execução. Executar o algoritmo de pesquisa para cada um dos nós da árvore, quantificando o tempo de execução para cada nó. Verificar o quanto a altura "h" de cada nó influencia no tempo de busca.

Exercício 3

Adaptar o algoritmo ***Merge-Sort*** para ordenar a sequência de números ***B***; comparar o tempo de execução com a ordenação por intercalação executada no conjunto ***A***. Se necessário executar o algoritmo *n* vezes para que o tempo medido seja quantificável.

```
A = [6, 6, 7, 8, 0, 1, 5, 9]
```

B = [15, 22, 39, 44, 51, 55, 67, 83, 44, 50, 61, 72, 74, 83, 96, 98]

Respostas:

Exercício 1:

```
import time

class Lista:
  def Lista(lista,inicio,chave):
    Lista.L = lista
    Lista.inicio = inicio
    Lista.anterior = Lista.L[chave][0]
    Lista.proximo = Lista.L[chave][4]
    Lista.dimensao = len(Lista.L)
    Lista.chave = Lista.L[chave]

def buscar(L, k):
  for p in range(1):
    x = L
    for i in range(len(L)):
      if x[i][1] == k:
        return x, x[i][1], i

def buscar_inserir(L, pos, novo):
  for p in range(1):
    x = L
    x.append(novo)
    p = len(x)-1
    q = len(x)-2
    while p >= pos:
      temp = x[q]
      x[q] = x[p]
      x[p] = temp
      p = p - 1
    for i in range(1, len(x), 1):
      x[i][0] = i-1
    for i in range(0, len(x)-1, 1):
      x[i][4] = i+1
    return x

def delete(L, k):
  for p in range(1):
    x = L
    x.pop(k)
  for i in range(1, len(x), 1):
    x[i][0] = i-1
  for i in range(1, len(x)-1, 1):
    x[i][0] = i+1
  return x

L = [[0, 1, 'João Silva', 4500.0, 1],[1, 2, 'Maria Oliveira', 5200.0, 2],[2, 3,
'Carlos Pereira', 4800.0, 3],[3, 4, 'Ana Souza', 5300.0, 4],[4, 5, 'Pedro Costa',
4600.0, 5],[5, 6, 'Fernanda Lima', 4700.0, 6],[6, 7, 'Rafael Almeida', 5100.0, 7],[7,
```

```
8, 'Juliana Martins', 4900.0, 8],[8, 9, 'Roberto Gonçalves', 5500.0, 9],[9, 10,
'Patricia Ferreira', 5000.0, 10]]

ele = [10, 11, 'Dor e Sofrimento', '9999.9', 12]

x, chave, pos_chave = buscar(L, 5)
Lista.Lista(L, L[0], pos_chave)

print('L padrao: ',x,'\n')

L_novo = buscar_inserir(L, pos_chave, ele)
Lista.Lista(L, L[0], pos_chave)
print('L Insert: ',L_novo,'\n')

L_novo = delete(L, 5)
Lista.Lista(L, L[0], pos_chave)
print('L Delete: ',L_novo,'\n')

for i in range(len(L)):
  srt = time.time()
  for j in range(100000):
    x, chave, pos_chave = buscar(L, 5)
    Lista.Lista(L, L[i], pos_chave)
  fnl = time.time()
  print(f'Tempo de processo da chave {i}: ', fnl-srt)
```

Console ex1:

```
L padrao:  [[0, 1, 'João Silva', 4500.0, 1], [1, 2, 'Maria Oliveira', 5200.0, 2], [2, 3, 'Carlos Pereira', 4800.0, 3], [3, 4, 'Ana Souza', 5300.0, 4], [4, 5, 'Pedro Costa', 4600.0, 5], [5, 6, 'Fernanda Lima',
 4700.0, 6], [6, 7, 'Rafael Almeida', 5100.0, 7], [7, 8, 'Juliana Martins', 4900.0, 8], [8, 9, 'Roberto Gonçalves', 5500.0, 9], [9, 10, 'Patricia Ferreira', 5000.0, 10]]

L Insert:  [[0, 1, 'João Silva', 4500.0, 1], [0, 2, 'Maria Oliveira', 5200.0, 2], [1, 3, 'Carlos Pereira', 4800.0, 3], [2, 4, 'Ana Souza', 5300.0, 4], [3, 6, 'Fernanda Lima', 4700.0, 5], [4, 7, 'Rafael Almeid
a', 5100.0, 6], [5, 8, 'Juliana Martins', 4900.0, 7], [6, 9, 'Roberto Gonçalves', 5500.0, 8], [7, 11, 'Dor e Sofrimento', '9999.9', 9], [8, 5, 'Pedro Costa', 4600.0, 10], [9, 10, 'Patricia Ferreira', 5000.0,
10]]

L Delete:  [[0, 1, 'João Silva', 4500.0, 1], [2, 2, 'Maria Oliveira', 5200.0, 2], [3, 3, 'Carlos Pereira', 4800.0, 3], [4, 4, 'Ana Souza', 5300.0, 4], [5, 6, 'Fernanda Lima', 4700.0, 5], [6, 8, 'Juliana Marti
ns', 4900.0, 7], [7, 9, 'Roberto Gonçalves', 5500.0, 8], [8, 11, 'Dor e Sofrimento', '9999.9', 9], [9, 5, 'Pedro Costa', 4600.0, 10], [8, 10, 'Patricia Ferreira', 5000.0, 10]]

Tempo de processo da chave 0:  0.1912386417388916
Tempo de processo da chave 1:  0.19620418548583984
Tempo de processo da chave 2:  0.19421696662902832
Tempo de processo da chave 3:  0.1976947784423828
Tempo de processo da chave 4:  0.20591378211975098
Tempo de processo da chave 5:  0.19993019104003906
Tempo de processo da chave 6:  0.199232816696167
Tempo de processo da chave 7:  0.1970984935760498
Tempo de processo da chave 8:  0.1882617473602295
Tempo de processo da chave 9:  0.1851797103881836
```

Exercício 2:

```
#ex 2 lista 2

import time

class Node:
    def __init__(self, bairro, esquerda=None, direita=None):
        self.bairro = bairro
        self.esquerda = esquerda
        self.direita = direita

class BinaryTree:
    def __init__(self):
```

```
        self.raiz = None

    def insert(self, bairro):
        if not self.raiz:
            self.raiz = Node(bairro)
        else:
            self._insert(self.raiz, bairro)

    def _insert(self, node, bairro):
        if bairro < node.bairro:
            if node.esquerda:
                self._insert(node.esquerda, bairro)
            else:
                node.esquerda = Node(bairro)
        else:
            if node.direita:
                self._insert(node.direita, bairro)
            else:
                node.direita = Node(bairro)

    def traverse(self):
        self._traverse(self.raiz)

    def _traverse(self, node):
        if node:
            self._traverse(node.esquerda)
            print(node.bairro)
            self._traverse(node.direita)

    def search(self, bairro):
        start_time = time.time()
        result = self._search(self.raiz, bairro)
        end_time = time.time()
        print(f"Tempo de busca para {bairro}: {end_time - start_time:.6f} segundos")
        return result

    def _search(self, node, bairro):
        if node:
            if node.bairro == bairro:
                return True
            elif bairro < node.bairro:
                return self._search(node.esquerda, bairro)
            else:
                return self._search(node.direita, bairro)
        return False

    def findHeight(self, node, x):
        if node is None:
            return -1
        if node.bairro == x:
            return 0
        leftHeight = self.findHeight(node.esquerda, x)
        rightHeight = self.findHeight(node.direita, x)
        if leftHeight > rightHeight:
```

```
            return leftHeight + 1
        else:
            return rightHeight + 1

# Criar a árvore binária
arvore = BinaryTree()

# Inserir os bairros da cidade de Santos
bairros = ["Gonzaga", "Ponta da Praia", "Embaré", "Aparecida", "São Vicente", "Santos
Centro", "Vila Mathias", "Vila Belmiro", "Boqueirão", "Campo Grande"]
for bairro in bairros:
    arvore.insert(bairro)

# Desenhar a árvore
print("Árvore binária:")
arvore.traverse()

# Executar o algoritmo de percurso dos nós
start_time = time.time()
arvore.traverse()
end_time = time.time()
print(f"Tempo de percurso: {end_time - start_time:.6f} segundos")

# Executar o algoritmo de pesquisa para cada um dos nós da árvore
for bairro in bairros:
    arvore.search(bairro)

# Verificar o quanto a altura "h" de cada nó influencia no tempo de busca
print("Influência da altura 'h' no tempo de busca:")
for bairro in bairros:
    altura = arvore.findHeight(arvore.raiz, bairro)
    print(f"Bairro: {bairro}, Altura: {altura}, Tempo de busca:
{arvore.search(bairro)}")
```

Console ex2:

```
Árvore binária:
Aparecida
Boqueirão
Campo Grande
Embaré
Gonzaga
Ponta da Praia
Santos Centro
São Vicente
Vila Belmiro
Vila Mathias
Aparecida
Boqueirão
Campo Grande
Embaré
Gonzaga
Ponta da Praia
Santos Centro
São Vicente
Vila Belmiro
Vila Mathias
Tempo de percurso: 0.000997 segundos
Tempo de busca para Gonzaga: 0.000000 segundos
Tempo de busca para Ponta da Praia: 0.000000 segundos
Tempo de busca para Embaré: 0.000000 segundos
Tempo de busca para Aparecida: 0.000000 segundos
Tempo de busca para São Vicente: 0.000000 segundos
Tempo de busca para Santos Centro: 0.000000 segundos
Tempo de busca para Vila Mathias: 0.000000 segundos
Tempo de busca para Vila Belmiro: 0.000000 segundos
Tempo de busca para Boqueirão: 0.000000 segundos
Tempo de busca para Campo Grande: 0.000000 segundos
Influência da altura 'h' no tempo de busca:
Tempo de busca para Gonzaga: 0.000000 segundos
Bairro: Gonzaga, Altura: 0, Tempo de busca: True
Tempo de busca para Ponta da Praia: 0.000000 segundos
Bairro: Ponta da Praia, Altura: 4, Tempo de busca: True
Tempo de busca para Embaré: 0.000000 segundos
Bairro: Embaré, Altura: 4, Tempo de busca: True
Tempo de busca para Aparecida: 0.000000 segundos
Bairro: Aparecida, Altura: 4, Tempo de busca: True
Tempo de busca para São Vicente: 0.000000 segundos
Bairro: São Vicente, Altura: 4, Tempo de busca: True
Tempo de busca para Santos Centro: 0.000000 segundos
Bairro: Santos Centro, Altura: 4, Tempo de busca: True
Tempo de busca para Vila Mathias: 0.000000 segundos
Bairro: Vila Mathias, Altura: 4, Tempo de busca: True
Tempo de busca para Vila Belmiro: 0.000000 segundos
Bairro: Vila Belmiro, Altura: 4, Tempo de busca: True
Tempo de busca para Boqueirão: 0.000000 segundos
Bairro: Boqueirão, Altura: 4, Tempo de busca: True
Tempo de busca para Campo Grande: 0.000000 segundos
Bairro: Campo Grande, Altura: 4, Tempo de busca: True
```

Exercício 3:

```
#ex 3 lista 2
```

```python
import time
import random

def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    while len(left) > 0 and len(right) > 0:
        if left[0] <= right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    result.extend(left)
    result.extend(right)
    return result

def intercalacao(A):
    n = len(A)
    for i in range(n-1):
        for j in range(n-i-1):
            if A[j] > A[j+1]:
                A[j], A[j+1] = A[j+1], A[j]
    return A

A = [6, 6, 7, 8, 0, 1, 5, 9]
B = [15, 22, 39, 44, 51, 55, 67, 83, 44, 50, 61, 72, 74, 83, 96, 98]

# Executar o algoritmo de ordenação por intercalação em A
start_time = time.time()
for _ in range(1000):  # Executar 1000 vezes para que o tempo seja quantificável
    intercalacao(A.copy())  # Copiar a lista A para não alterar a original
end_time = time.time()
print(f"Tempo de ordenação por intercalação em A: {end_time - start_time:.6f}
segundos")

# Executar o algoritmo Merge-Sort em B
start_time = time.time()
for _ in range(1000):  # Executar 1000 vezes para que o tempo seja quantificável
    merge_sort(B.copy())  # Copiar a lista B para não alterar a original
end_time = time.time()
print(f"Tempo de ordenação por Merge-Sort em B: {end_time - start_time:.6f}
segundos")
```

Console ex3:

```
Tempo de ordenação por intercalação em A: 0.003992 segundos
Tempo de ordenação por Merge-Sort em B: 0.016027 segundos
```