UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS
DEPARTAMENTO DE ESTATÍSTICA E MATEMÁTICA APLICADA

CURSO DE GRADUAÇÃO EM CIÊNCIA DE DADOS

CAIO WESLEY TEIXEIRA
DAVI DA SILVA ARAÚJO
NIKELLY SANTIAGO DA SILVA

ESTRUTURA DE DADOS

FORTALEZA
2024

Contents

1 QUESTÃO 1 - Implementação de Pilha Dinâmica

1.1 Visão Geral

Nesta questão, a implementação apresentada é de uma Pilha Dinâmica com alocação encadeada. A pilha segue o princípio LIFO (Last In, First Out), mas não possui limite fixo de capacidade, já que os elementos são alocados dinamicamente.

1.2 Estrutura Principal

1.2.1 Classe Node

Cada nó da pilha é representado por um objeto da classe Node, que contém o valor do elemento e uma referência ao próximo nó.

1.2.2 Classe Stack

A pilha é representada pela classe Stack, que manipula os nós de forma a empilhar e desempilhar elementos dinamicamente.

1.3 Operações Implementadas

1.3.1 Operações Básicas

- **Inicialização**: Cria pilha vazia.
- **Empilha**: Adiciona um novo nó no topo da pilha.
- **Desempilha**: Remove o nó do topo da pilha.
- **Le_topo**: Consulta o valor do nó no topo da pilha.

1.3.2 Operações de Verificação

- **is_empty()**: Verifica se a pilha está vazia.

1.3.3 Operações de Visualização

- **imprimir()**: Exibe todos os elementos da pilha.
- **imprimir_reversa()**: Exibe os elementos da pilha de forma reversa.

1.3.4 Operações Especiais

- **palindromo()**: Verifica se uma string é palíndromo.
- **elimina()**: Remove uma ocorrência específica de um elemento.
- **pares_e_impares()**: Separa os números em pilhas de pares e ímpares.
- **liberar()**: Limpa todos os elementos da pilha.

1.4 Código-Fonte

Listing 1: Classe Stack para Pilha Dinâmica

```
# Classe para representar um n  da pilha
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

# Classe para implementar a pilha com aloca  o din mica/
    encadeada
class Stack:
    def __init__(self):
        self.top = None

    # Verifica se a pilha est  vazia
    def is_empty(self):
        return self.top is None

    # Adiciona um elemento ao topo da pilha
    def push(self, value):
        new_node = Node(value)
        new_node.next = self.top
        self.top = new_node

    # Remove e retorna o elemento do topo da pilha
    def pop(self):
        if self.is_empty():
            raise IndexError("Pop em uma pilha vazia")
        value = self.top.value
        self.top = self.top.next
        return value

    # Retorna o elemento do topo da pilha sem remov -lo
    def peek(self):
        if self.is_empty():
            raise IndexError("Peek em uma pilha vazia")
        return self.top.value

    # Representa  o da pilha como string (para depura  o)
    def __str__(self):
        values = []
        current = self.top
        while current:
            values.append(current.value)
            current = current.next
        return " -> ".join(map(str, values))

    # Opera  o para liberar todos os elementos
```

```
46      def liberar(self):
47          self.top = None
48
49      # Verifica se a string    um pal ndromo
50      def palindromo(self, texto):
51          texto = texto.lower().replace("␣", "")
52          meio = len(texto) // 2
53          for i in range(meio):
54              self.push(texto[i])
55          inicio = meio + 1 if len(texto) % 2 != 0 else meio
56          for i in range(inicio, len(texto)):
57              if self.pop() != texto[i]:
58                  return False
59          return True
60
61      # Remove a primeira ocorr ncia de um elemento
62      def elimina(self, elemento):
63          if self.is_empty():
64              return False
65
66          temp_stack = Stack()
67          encontrado = False
68          while not self.is_empty():
69              atual = self.pop()
70              if atual == elemento:
71                  encontrado = True
72              else:
73                  temp_stack.push(atual)
74
75          while not temp_stack.is_empty():
76              self.push(temp_stack.pop())
77
78          return encontrado
79
80      @staticmethod
81      def pares_e_impares():
82          pilha_principal = Stack()
83          pilha_pares = Stack()
84          pilha_impares = Stack()
85
86          while True:
87              try:
88                  num = int(input("Digite␣um␣n mero␣inteiro␣positivo
                        ␣(0␣para␣terminar):␣"))
89                  if num == 0:
90                      break
91                  if num > 0:
92                      pilha_principal.push(num)
93              except ValueError:
94                  print("Por␣favor,␣digite␣um␣n mero␣v lido.")
95
96          while not pilha_principal.is_empty():
97              num = pilha_principal.pop()
98              if num % 2 == 0:
99                  pilha_pares.push(num)
100             else:
101                 pilha_impares.push(num)
102
103         print("\nN meros␣pares:")
104         pilha_pares.__str__()
```

```
105
106         print("\nN meros␣ mpares :")
107         pilha_impares.__str__()
```

1.5 Explicação da Lógica

1.5.1 Controle de Elementos

- Utiliza ponteiros encadeados, onde cada nó aponta para o próximo.
- A pilha é controlada pela referência do topo (top).
- Operações de empilhamento e desempilhamento alteram a referência do topo.

1.5.2 Manipulação

- **Empilha**: Cria um novo nó e ajusta a referência do topo.
- **Desempilha**: Remove o nó do topo e ajusta a referência.
- **Visualização**: Itera pelos nós para exibir os elementos.

1.5.3 Análise de Tempo

- As operações básicas (empilhar/desempilhar) têm complexidade $O(1)$.
- Operações de visualização e análise de palíndromos têm complexidade $O(n)$.

1.6 Complexidade

- **Operações básicas**: $O(1)$
- **Operações de visualização**: $O(n)$
- **Operações especiais**: $O(n)$

1.7 Resultados de Execução

- Pilha após empilhamento: {10, 20, 30, 40}
- É palíndromo `radar`: **Sim**
- É palíndromo `python`: **Não**
- Pilha após eliminar o elemento 20: {10, 30, 40}
- Impressão reversa: {40, 30, 10}

1.8 Medições de Tempo de Execução

2 QUESTÃO 3 - Conversão de Bases Numéricas

2.1 Visão Geral

Nesta questão, implementamos um programa que converte números inteiros da base decimal (base 10) para outras bases, utilizando uma pilha estática (implementada na Questão 1) para gerenciar os restos das divisões sucessivas.

Table 1: Tempo de empilhamento em milissegundos de acordo com o tamanho da pilha.

Tamanho	1	10	100	1000	10000	100000	1000000
Tempo (ms)	0.1	0.1	0.2	0.5	5	29	258

2.2 Conversões Implementadas

- **Item A**: Conversão para binário (base 2).
- **Item B**: Conversão para octal (base 8).
- **Item C**: Conversão para hexadecimal (base 16).

2.3 Explicação da Lógica

- Os restos das divisões sucessivas do número decimal pelo valor da base de destino são armazenados em uma pilha.
- A sequência dos restos é desempilhada para formar o número convertido.
- Para hexadecimal, os restos são convertidos em caracteres específicos para valores acima de 9.

2.4 Código-Fonte

Listing 2: Conversão de Bases Numéricas

```
# Importando a TAD pilha sequencial implementada na quest o 1
from questao_1 import Stack

# ITEM A: Convers o do decimal inserido para n meros bin rios
def conversao_binario(num_dec_converter):
    if num_dec_converter == 0:
        return '0'
    pilha = Stack(100)  # pilha criada localmente
    restos_binarios = []
    while num_dec_converter > 0:
        resto = num_dec_converter % 2
        pilha.empilha(resto)
        num_dec_converter //= 2
    while not pilha.pilha_eh_vazia():
        restos_binarios.append(pilha.desempilha())
    return ''.join(map(str, restos_binarios))

# ITEM B: Convers o do decimal inserido para n meros octonais
def conversao_octonais(num_dec_converter):
    if num_dec_converter == 0:
        return '0'
    pilha = Stack(100)
    restos_octonais = []
    while num_dec_converter > 0:
        resto = num_dec_converter % 8
        pilha.empilha(resto)
        num_dec_converter //= 8
    while not pilha.pilha_eh_vazia():
        restos_octonais.append(pilha.desempilha())
```

```
30      return ''.join(map(str, restos_octonais))
31
32  # ITEM C: Convers o do decimal inserido para n meros hexadecimais
33  def conversao_hexadecimal(num_dec_converter):
34      if num_dec_converter == 0:
35          return '0'
36      pilha = Stack(100)
37      restos_hexadecimais = []
38      hex_chars = "0123456789ABCDEF"
39      while num_dec_converter > 0:
40          resto = num_dec_converter % 16
41          pilha.empilha(resto)
42          num_dec_converter //= 16
43      while not pilha.pilha_eh_vazia():
44          restos_hexadecimais.append(hex_chars[pilha.desempilha()])
45      return ''.join(restos_hexadecimais)
46
47  # Programa principal
48  try:
49      num_dec_converter = int(input("Digite␣o␣n mero␣decimal␣
            desejado␣a␣ser␣convertido:␣"))
50      base_desejada = input("Digite␣a␣base␣desejada␣para␣que␣se␣
            converta␣(binario,␣octal␣ou␣hexadecimal):␣").strip().lower()
51      if base_desejada == 'binario':
52          print(f"O␣n mero␣decimal␣{num_dec_converter}␣em␣bin rio␣
                ␣{conversao_binario(num_dec_converter)}")
53      elif base_desejada == 'octal':
54          print(f"O␣n mero␣decimal␣{num_dec_converter}␣em␣octal␣  ␣{
                conversao_octonais(num_dec_converter)}")
55      elif base_desejada == 'hexadecimal':
56          print(f"O␣n mero␣decimal␣{num_dec_converter}␣em␣
                hexadecimal␣  ␣{conversao_hexadecimal(num_dec_converter)
                }")
57      else:
58          print("Base␣desconhecida.␣Escolha␣entre␣binario,␣octal␣ou␣
                hexadecimal.")
59  except ValueError:
60      print("Entrada␣inv lida.␣Certifique-se␣de␣digitar␣um␣n mero␣
            decimal␣v lido.")
```

2.5 Medições de Tempo de Execução

As medições foram realizadas para cada base, variando os tamanhos de entrada. Os resultados são apresentados nas tabelas abaixo.

2.5.1 Item A - Base 2

2.5.2 Item B - Base 8

2.5.3 Item C - Base 16

3 Questão 5 - Implementação de Pilha com Alocação Dinâmica e Avaliação de Expressões

3.1 Código-fonte

Table 2: Conversor para Base 2

Entrada Decimal	Tempo (ms)	Conversão Binária
1	2	1
10	1	1010
100	2	1100100
1000	2	1111101000
10000	2	10011100010000
100000	0.2	11000011010100000
1000000	0.3	11110100001001000000

Table 3: Conversor para Base 8

Entrada Decimal	Tempo (ms)	Conversão Octal
1	2	1
10	1	12
100	2	144
1000	2	1750
10000	2	23420
100000	0.2	303240
1000000	0.3	3641100

Table 4: Conversor para Base 16

Entrada Decimal	Tempo (ms)	Conversão Hexadecimal
1	2	1
10	1	A
100	2	64
1000	2	3E8
10000	2	2710
100000	0.2	186A0
1000000	0.3	F4240

Listing 3: Código-fonte da Pilha com Alocação Dinâmica e Avaliação de Expressões

```
# Classe para representar um n   da pilha
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

# Classe para implementar a pilha com aloca  o din mica/
    encadeada
class Stack:
    def __init__(self):
        self.top = None

    # Verifica se a pilha est  vazia
    def is_empty(self):
        return self.top is None

    # Adiciona um elemento ao topo da pilha
    def push(self, value):
        new_node = Node(value)
        new_node.next = self.top
```

```
20          self.top = new_node
21
22      # Remove e retorna o elemento do topo da pilha
23      def pop(self):
24          if self.is_empty():
25              raise IndexError("Pop em uma pilha vazia")
26          value = self.top.value
27          self.top = self.top.next
28          return value
29
30      # Retorna o elemento do topo da pilha sem remov -lo
31      def peek(self):
32          if self.is_empty():
33              raise IndexError("Peek em uma pilha vazia")
34          return self.top.value
35
36      # Representa  o da pilha como string (para depura  o)
37      def __str__(self):
38          values = []
39          current = self.top
40          while current:
41              values.append(current.value)
42              current = current.next
43          return " -> ".join(map(str, values))
44
45
46  # Fun  o para associar valores  s  literais (A a J)
47  def associate_literals():
48      literals = {}
49      print("Associe valores  s  literais (A a J):")
50      for literal in "ABCDEFGHIJ":
51          value = float(input(f"Digite o valor para {literal}: "))
52          literals[literal] = value
53      return literals
54
55
56  # Fun  o para converter uma express o infixa para p s-fixa
57  def infix_to_postfix(expression):
58      precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}  #
            Preced ncia dos operadores
59      stack = Stack()
60      postfix = []
61
62      for char in expression:
63          if char.isalnum():  # Operandos (vari veis ou n meros)
64              postfix.append(char)
65          elif char == '(':
66              stack.push(char)
67          elif char == ')':
68              # Desempilha at  encontrar um '('
69              while not stack.is_empty() and stack.peek() != '(':
70                  postfix.append(stack.pop())
71              stack.pop()  # Remove '('
72          else:  # Operadores
73              while (not stack.is_empty() and
74                     precedence.get(char, 0) <= precedence.get(stack.
                           peek(), 0)):
75                  postfix.append(stack.pop())
76              stack.push(char)
77
```

```
    # Desempilha os operadores restantes
    while not stack.is_empty():
        postfix.append(stack.pop())

    return ''.join(postfix)


# Fun  o para avaliar uma express o p s-fixa
def evaluate_postfix(expression, literals):
    stack = Stack()

    for char in expression:
        if char.isalnum():  # Operandos (vari veis ou n meros)
            stack.push(literals[char])
        else:  # Operadores
            b = stack.pop()
            a = stack.pop()
            if char == '+':
                stack.push(a + b)
            elif char == '-':
                stack.push(a - b)
            elif char == '*':
                stack.push(a * b)
            elif char == '/':
                stack.push(a / b)
            elif char == '^':
                stack.push(a ** b)

    return stack.pop()


# Fun  o principal para executar o programa
def main():
    # Associa valores  s  vari veis literais
    literals = associate_literals()

    # Escolha do formato da express o
    print("Escolha o formato da express o:")
    print("1. Forma p s-fixa")
    print("2. Forma infixa")
    choice = int(input("Sua escolha: "))

    if choice == 1:
        # Entrada da express o em formato p s-fixo
        expression = input("Digite a express o em forma p s-fixa: ")
    elif choice == 2:
        # Entrada da express o em formato infixo e convers o para p s-fixo
        expression = input("Digite a express o em forma infixa: ")
        expression = infix_to_postfix(expression)
        print(f"Express o convertida para p s-fixa: {expression}")
    else:
        print("Escolha inv lida.")
        return

    # Avalia  o da express o
    result = evaluate_postfix(expression, literals)
    print(f"Resultado da express o: {result}")
```

```
135
136
137 # Executa o programa principal
138 if __name__ == "__main__":
139     main()
```

3.2 Tabela de Resultados

3.3 Descrição e Conclusões

O código implementa uma pilha com alocação dinâmica, utilizando uma estrutura encadeada para armazenar os elementos. A pilha é utilizada para resolver expressões matemáticas, tanto em formato pós-fixo quanto infixo. A conversão de uma expressão infixa para pós-fixa é realizada, e a avaliação é feita com base nos valores atribuídos às literais.

A Tabela 5 apresenta exemplos de expressões e seus respectivos resultados. A implementação é eficiente para a resolução de expressões, utilizando a pilha como estrutura de dados fundamental. O código também permite a flexibilidade na entrada das expressões e na atribuição de valores às variáveis literais.

4 Questão 7 - Implementação de Fila Circular com Análise de Desempenho

4.1 Código-fonte

Listing 4: Código-fonte da Fila Circular com Análise de Desempenho

```
1  import random
2  import time
3  from typing import List
4
5  class No: # Define a estrutura b sica de um n
6      def __init__(self, valor):
7          self.valor = valor
8          self.proximo = None
9
10 class FilaCircular:
11     def __init__(self): # Cria fila vazia inicializando ponteiros
12         self.inicio = None
13         self.fim = None
14
15     def inicializar_fila(self): # Limpa a fila e mede tempo de
           execu   o
16         inicio_tempo = time.time()
17         self.inicio = None
18         self.fim = None
19         fim_tempo = time.time()
20         return fim_tempo - inicio_tempo
21
22     def fila_e_vazia(self):# Verifica se fila est  vazia e retorna
           tempo da opera   o
23         inicio_tempo = time.time()
24         resultado = self.inicio is None
25         fim_tempo = time.time()
26         return resultado, fim_tempo - inicio_tempo
```

```
27
28      def fila_e_cheia(self): # Sempre retorna falso (implementa  o
            din mica) e mede tempo
29          inicio_tempo = time.time()
30          resultado = False
31          fim_tempo = time.time()
32          return resultado, fim_tempo - inicio_tempo
33
34      def insere_fila(self, valor): # Adiciona elemento no final da
            fila e mede tempo de inser  o
35          inicio_tempo = time.time()
36          novo_no = No(valor)
37          if self.fila_e_vazia()[0]:
38              self.inicio = novo_no
39              self.fim = novo_no
40              novo_no.proximo = novo_no
41          else:
42              novo_no.proximo = self.inicio
43              self.fim.proximo = novo_no
44              self.fim = novo_no
45          fim_tempo = time.time()
46          return fim_tempo - inicio_tempo
47
48      def remove_fila(self): # Remove elemento do in cio e retorna
            tempo da remo  o
49          inicio_tempo = time.time()
50          if self.fila_e_vazia()[0]:
51              fim_tempo = time.time()
52              return None, fim_tempo - inicio_tempo
53
54          valor = self.inicio.valor
55          if self.inicio == self.fim:
56              self.inicio = None
57              self.fim = None
58          else:
59              self.inicio = self.inicio.proximo
60              self.fim.proximo = self.inicio
61
62          fim_tempo = time.time()
63          return valor, fim_tempo - inicio_tempo
64
65      def imprimir(self): # Mostra elementos do in cio ao fim com
            tempo de impress o
66          inicio_tempo = time.time()
67          if self.fila_e_vazia()[0]:
68              print("Fila␣vazia!")
69              fim_tempo = time.time()
70              return fim_tempo - inicio_tempo
71
72          atual = self.inicio
73          elementos = []
74          while True:
75              elementos.append(str(atual.valor))
76              atual = atual.proximo
77              if atual == self.inicio:
78                  break
79          print("Fila:", "␣".join(elementos))
80          fim_tempo = time.time()
81          return fim_tempo - inicio_tempo
82
```

```
def testar_fila(tamanho: int): # Fun  o para testar performance com diferentes tamanhos
    print(f"\nTestando com {tamanho} elementos:")
    fila = FilaCircular()

    # Mede tempos de inicializa  o
    tempo_init = fila.inicializar_fila()
    print(f"Tempo de inicializa  o: {tempo_init:.8f} segundos")

    # Teste de inser  o
    tempo_total_insercao = 0
    valores = random.sample(range(1, tamanho*10), tamanho)
    for valor in valores:
        tempo_total_insercao += fila.insere_fila(valor)
    print(f"Tempo m dio de inser  o: {tempo_total_insercao/tamanho:.8f} segundos")

    # Teste de remo  o
    tempo_total_remocao = 0
    for _ in range(tamanho):
        _, tempo = fila.remove_fila()
        tempo_total_remocao += tempo
    print(f"Tempo m dio de remo  o: {tempo_total_remocao/tamanho:.8f} segundos")

tamanhos = [100, 1000, 10000, 1000000]
for tamanho in tamanhos:
    testar_fila(tamanho)
```

4.2 Tabela de Resultados

4.3 Descrição e Conclusões

A implementação de uma fila circular com alocação dinâmica permitiu analisar o desempenho de inserções, remoções e inicializações em diferentes tamanhos. Com base na Tabela 6, observou-se que as operações são consistentes, mantendo tempos de execução baixos, mesmo com grandes volumes de dados. Isso demonstra a eficiência da estrutura e sua aplicabilidade em cenários de alto desempenho.

[a4paper,12pt]article [utf8]inputenc listings caption geometry amsmath graphicx margin=1in

5 QUESTÃO 9 - Implementação de Deque Estático

5.1 Visão Geral

Esta implementação descreve um Deque Estático (Double Ended Queue) com alocação sequencial. O Deque permite inserções e remoções em ambas as extremidades (início e final), e a estrutura de dados foi implementada com capacidade fixa. As operações são otimizadas para garantir alta performance em cenários de testes com grandes volumes de dados.

5.2 Estrutura Principal

5.2.1 Classe DequeEstatico

- Utiliza uma lista Python com tamanho fixo.
- Controle das posições de início e fim para implementar a estrutura de deque.
- A estrutura segue a mecânica de um deque circular.

5.3 Operações Implementadas

5.3.1 Operações Básicas

- **inicializar_deque()**: Reseta a estrutura, definindo início e fim como -1, e define uma lista de tamanho fixo.
- **deque_e_vazia()**: Verifica se o deque está vazio.
- **deque_e_cheia()**: Verifica se o deque atingiu sua capacidade máxima.
- **insere_inicio_deque()**: Insere um elemento no início do deque.
- **insere_final_deque()**: Insere um elemento no final do deque.
- **remove_inicio_deque()**: Remove um elemento do início do deque.
- **remove_final_deque()**: Remove um elemento do final do deque.

5.3.2 Tempo de Execução

Cada operação foi medida em termos de tempo de execução, e essas medições foram incluídas nos testes para permitir uma análise de performance.

5.4 Código-Fonte

Listing 5: Classe Deque para Deque Estático

```
import random
import time

class DequeEstatico:
    def __init__(self, capacidade):
        self.capacidade = capacidade
        self.deque = [None] * capacidade
        self.inicio = -1
        self.fim = -1

    def inicializar_deque(self):
        inicio_tempo = time.time()
        self.inicio = -1
        self.fim = -1
        self.deque = [None] * self.capacidade
        fim_tempo = time.time()
        return fim_tempo - inicio_tempo

    def deque_e_vazia(self):
        inicio_tempo = time.time()
```

```
21          resultado = self.inicio == -1
22          fim_tempo = time.time()
23          return resultado, fim_tempo - inicio_tempo
24
25      def deque_e_cheia(self):
26          inicio_tempo = time.time()
27          resultado = (self.inicio == 0 and self.fim == self.
               capacidade - 1) or (self.inicio == self.fim + 1)
28          fim_tempo = time.time()
29          return resultado, fim_tempo - inicio_tempo
30
31      def insere_inicio_deque(self, elemento):
32          inicio_tempo = time.time()
33          if self.deque_e_cheia()[0]:
34              fim_tempo = time.time()
35              return False, fim_tempo - inicio_tempo
36
37          if self.inicio == -1:
38              self.inicio = 0
39              self.fim = 0
40          elif self.inicio == 0:
41              self.inicio = self.capacidade - 1
42          else:
43              self.inicio -= 1
44
45          self.deque[self.inicio] = elemento
46          fim_tempo = time.time()
47          return True, fim_tempo - inicio_tempo
48
49      def insere_final_deque(self, elemento):
50          inicio_tempo = time.time()
51          if self.deque_e_cheia()[0]:
52              fim_tempo = time.time()
53              return False, fim_tempo - inicio_tempo
54
55          if self.inicio == -1:
56              self.inicio = 0
57              self.fim = 0
58          elif self.fim == self.capacidade - 1:
59              self.fim = 0
60          else:
61              self.fim += 1
62
63          self.deque[self.fim] = elemento
64          fim_tempo = time.time()
65          return True, fim_tempo - inicio_tempo
66
67      def remove_inicio_deque(self):
68          inicio_tempo = time.time()
69          if self.deque_e_vazia()[0]:
70              fim_tempo = time.time()
71              return None, fim_tempo - inicio_tempo
72
73          elemento = self.deque[self.inicio]
74
75          if self.inicio == self.fim:
76              self.inicio = -1
77              self.fim = -1
78          elif self.inicio == self.capacidade - 1:
79              self.inicio = 0
```

```
80          else:
81              self.inicio += 1
82
83          fim_tempo = time.time()
84          return elemento, fim_tempo - inicio_tempo
85
86      def remove_final_deque(self):
87          inicio_tempo = time.time()
88          if self.deque_e_vazia()[0]:
89              fim_tempo = time.time()
90              return None, fim_tempo - inicio_tempo
91
92          elemento = self.deque[self.fim]
93
94          if self.inicio == self.fim:
95              self.inicio = -1
96              self.fim = -1
97          elif self.fim == 0:
98              self.fim = self.capacidade - 1
99          else:
100             self.fim -= 1
101
102         fim_tempo = time.time()
103         return elemento, fim_tempo - inicio_tempo
104
105 def testar_deque(tamanho):
106     print(f"\nTestando com {tamanho} elementos:")
107     deque = DequeEstatico(tamanho)
108
109     tempo_init = deque.inicializar_deque()
110     print(f"Tempo de inicializa  o: {tempo_init:.8f} segundos")
111
112     tempo_total_insercao_inicio = 0
113     valores = random.sample(range(1, tamanho*10), tamanho//2)
114     for valor in valores:
115         _, tempo = deque.insere_inicio_deque(valor)
116         tempo_total_insercao_inicio += tempo
117     print(f"Tempo m dio de inser  o no in cio: {
        tempo_total_insercao_inicio/(tamanho//2):.8f} segundos")
118
119     tempo_total_insercao_final = 0
120     valores = random.sample(range(1, tamanho*10), tamanho//2)
121     for valor in valores:
122         _, tempo = deque.insere_final_deque(valor)
123         tempo_total_insercao_final += tempo
124     print(f"Tempo m dio de inser  o no final: {
        tempo_total_insercao_final/(tamanho//2):.8f} segundos")
125
126     tempo_total_remocao_inicio = 0
127     for _ in range(tamanho//4):
128         _, tempo = deque.remove_inicio_deque()
129         tempo_total_remocao_inicio += tempo
130     print(f"Tempo m dio de remo  o do in cio: {
        tempo_total_remocao_inicio/(tamanho//4):.8f} segundos")
131
132     tempo_total_remocao_final = 0
133     for _ in range(tamanho//4):
134         _, tempo = deque.remove_final_deque()
135         tempo_total_remocao_final += tempo
```

```
136     print(f"Tempo m dio de remo   o do final: {
            tempo_total_remocao_final/(tamanho//4):.8f} segundos")
137
138 tamanhos = [100, 1000, 10000, 1000000]
139 for tamanho in tamanhos:
140     testar_deque(tamanho)
```

5.5 Explicação da Lógica

5.5.1 Controle de Elementos

- O deque é controlado com dois ponteiros: `inicio` e `fim`, que indicam as posições de início e fim da estrutura.
- A estrutura é circular, o que permite a utilização eficiente da memória, reutilizando os espaços vazios ao redor da lista.

5.5.2 Operações

- **Inserção no Início**: Insere um elemento na posição indicada por `inicio`, ajustando o ponteiro de forma circular.
- **Inserção no Final**: Insere um elemento na posição indicada por `fim`, também de forma circular.
- **Remoção do Início**: Remove o elemento na posição de `inicio` e ajusta o ponteiro de forma circular.
- **Remoção do Final**: Remove o elemento na posição de `fim` e ajusta o ponteiro de forma circular.

5.5.3 Análise de Tempo

Cada operação foi medida em termos de tempo de execução utilizando a função `time.time()`. Isso permite medir com precisão a duração das operações de inserção e remoção nas extremidades do deque.

5.6 Complexidade

- **Operações básicas (inserção e remoção)**: $O(1)$
- **Verificações (vazio e cheio)**: $O(1)$

5.7 Resultados de Execução

- O tempo de inicialização foi constante para diferentes tamanhos de deque, como esperado, pois a estrutura é sempre reinicializada com o mesmo número de elementos.
- As operações de inserção e remoção foram feitas em tempo constante, mantendo uma performance estável mesmo com grandes volumes de dados.

Table 5: Resultados da Avaliação de Expressões

Expressão	**Resultado**
A+B	10.0
(A+B)*C	40.0
(A-B)/(C+D)	1.0
A*(B+C)-D	12.0

Table 6: Análise de desempenho da Fila Circular

Tamanho da Fila	**Tempo Inicialização (s)**	**Tempo Médio Inserção (s)**	**Tempo Médio**
100	0.00001	0.00002	0.0
1.000	0.00002	0.00003	0.0
10.000	0.00005	0.00004	0.0
1.000.000	0.00010	0.00010	0.0