gianfrancojoel_condoriluna_aula7Pratico

October 8, 2024

1 Aula 07 - Exercício prático Variações de listas

Aluno: Gian Franco Joel Condori Luna

1.0.1 1) Resolver os exercícios no Beecrowd Judge: https://judge.beecrowd.com/

1242 Ácido Ribonucleico Alienígena

1083 LEXSIM - Avaliador Lexico e Sintático

- Anexar no classroom um pdf com a análise de complexidade dos algoritmos que vc desenvolveu.

1.0.2 Solução:

1.0.3 1242 Ácido Ribonucleico Alienígena

Passos:

- Para resolver este problema decidi usar Listas Duplas Encadeadas, pois elas me permitiram saber quem era meu nó anterior. É aqui que é armazenada a lista de letras inseridas pelo usuário.
- Então você começa a percorrer a lista letra por letra (S, F, B, C) perguntando se o próximo nó tem sua combinação certa, por exemplo se é a letra F você procura se o próximo nó é a letra S ou se a letra for B, pergunte se o próximo nó é a letra S e assim por diante.
- Quando a combinação de letras apropriada é encontrada no próximo nó, ambos os nós são removidos da nossa lista e adicionamos 1 ao nosso contador que tem o valor da resposta final. Se a combinação apropriada não for encontrada, a lista é rolada até chegar ao último elemento.

[2]:
```python
# PYTHON version 3.10.12

# Classe Nó
class Nodo:
    def __init__(self, valor):
        self.valor = valor  # Valor nó
        self.siguiente = None
        self.anterior = None

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

```
        self.cabeza = None  # Inicialmente a lista está vazia

    def agregar(self, valor):
        nodo_nuevo = Nodo(valor)  # Crie um novo nó
        if not self.cabeza:  # Se a lista estiver vazia
            self.cabeza = nodo_nuevo  # O novo nó é a cabeça
        else:
            nodo_actual = self.cabeza
            while nodo_actual.siguiente:  # Vá até o fim
                nodo_actual = nodo_actual.siguiente
            nodo_actual.siguiente = nodo_nuevo  # Vincule o novo nó ao final
            nodo_nuevo.anterior = nodo_actual # Link para o nó anterior

    def eliminar(self, nodo):
        # Veja se é a cabeça
        if nodo.anterior:
            nodo.anterior.siguiente = nodo.siguiente
            if nodo.siguiente:
                nodo.siguiente.anterior = nodo.anterior
        else:
            if nodo.siguiente:
                nodo.siguiente.anterior = None
                self.cabeza = nodo.siguiente
            else:
                self.cabeza = None

    def imprimir(self):
        nodo_actual = self.cabeza
        while nodo_actual:  # Vá até o fim da lista
            print(nodo_actual.valor)  # Imprimir valor do nó
            nodo_actual = nodo_actual.siguiente  # Mover para o próximo nó
```

[3]:
```
# Função principal
def main(texto):
    texto = texto.upper()
    letras = list(texto)

    # Crie uma lista vinculada
    lista = ListaEnlazada()

    for letra in letras:
        # Adicionar itens
        lista.agregar(letra)

    #--------------------------
    # RESOLVENDO O PROBLEMA
    #--------------------------
```

```
    nodo_actual = lista.cabeza
    # Contador da resposta
    respuesta = 0

    # Vá até o fim da lista
    while nodo_actual:
        # Pergunte que letra é
        letra = nodo_actual.valor
        if nodo_actual.siguiente:
            letra_siguiente = nodo_actual.siguiente.valor
        else:
            letra_siguiente = None
        # Só pode ser S, B, F, C e S->B, B->S, F->C, C->F
        if (letra == 'S' and letra_siguiente == 'B') or (letra == 'B' and␣
↪letra_siguiente == 'S') or (letra == 'F' and letra_siguiente == 'C') or␣
↪(letra == 'C' and letra_siguiente == 'F'):
            # Nó a ser excluído
            nodo_eliminar = nodo_actual
            # Alterar nó atual
            # Pergunte se não é a cabeça
            if nodo_actual.anterior:
                nodo_actual = nodo_actual.anterior
            else:
                nodo_actual = nodo_actual.siguiente.siguiente

            # Excluir segundo nó
            lista.eliminar(nodo_eliminar.siguiente)
            # Excluir primeiro nó
            lista.eliminar(nodo_eliminar)
            # Adicione um à resposta
            respuesta = respuesta+1
        else:
            nodo_actual = nodo_actual.siguiente

    print(respuesta)
    #lista.imprimir()
```

[4]:
```
input_data = []

try:
    while True:
        line = input()  # Capturar entrada do console

        if line == "":  # Condição para finalizar a entrada (pressione Enter em␣
↪uma linha vazia)
            break
```

```
        # Verifique se o comprimento está dentro do intervalo permitido
        if 1 <= len(line) <= 300:
            input_data.append(line)  # Adicione a linha à lista se ela atender␣
 ↪ao comprimento
        else:
            print("Erro: a entrada deve ter entre 1 e 300 caracteres.")  #␣
 ↪Mostra uma mensagem se não estiver em conformidade
except EOFError:
    pass  # Lidar com EOF se ocorrer

# Imprima as entradas capturadas que atendem à condição
for texto in input_data:
    main(texto)
```

1.0.4 Complexidade do código

1. Classe "Nodo"

- Complexidade de tempo para criar um nó: O(1) (constante).

2. Métodos da classe "ListaEnlazada"

a) Método "agregar"

- Complexidade de tempo: O(n) (onde n é o número de nós na lista no momento da adição).

b) Método "eliminar"

- Complexidade de tempo: O(1) (constante).

c) Método "imprimir"

- Complexidade de tempo: O(n) (onde n é o número de nós na lista).

3. Função main

a) Adicionar letras à lista encadeada

- O método agregar é chamado para cada letra. No pior caso, adicionar um elemento tem complexidade de O(n) (percorrendo toda a lista). Portanto, para um texto de comprimento m, a complexidade total de adicionar as letras é: $O(m^2)$ (para adicionar todas as letras).

b) Remover pares de nós

- Como a remoção de cada nó ocorre em tempo constante (O(1)), a complexidade do processo de remoção é: O(m) (para percorrer e eliminar nós).

4. Laço principal para captura de entrada

- Complexidade de tempo: O(p * m), onde p é o número de linhas de entrada e m é o comprimento médio de cada linha.

Resposta:

- Complexidade total de tempo: $O(n^2)$