FACULDADE DE TECNOLOGIA SENAC RIO

FACULDADE DE TECNOLOGIA SENAC RIO	
Curso: Análise e Desenvolvimento de Sistemas	**Semestre letivo: 2023.1**
Unidade Curricular: Estrutura de Dados	**Módulo:** Modelagem do Projeto de Sistemas
Professor: Bruno Santos do Nascimento	**Data:** 09/05/2023
Competências a serem avaliadas: • Desenvolver estruturas de dados para armazenar e organizar informações de um sistema computacional de forma eficientemente, facilitando sua busca e modificação.	**Indicadores de Competência:** • Implementa estrutura de dados de acordo com as necessidades projetadas de um sistema computacional. • Desenvolve sistemas computacionais utilizando estruturas de dados linear e não linear.
Aluno: Danielle Oliveira Moreno de Souza	**Conceito:**

Prezado aluno,

No dia 09/05 realizaremos uma recuperação para TODOS. Durante a semana de 02/05 a 08/05 você terá a oportunidade de refazer a Avaliação Prática.

Todas as 3 questões devem ser refeitas! Juntamente com a solução, deverá ser encaminhado um relatório apontando o que foi corrigido (compare as implementações: a entregue em 02/05 e a nova implementação - corrigida).

OBS. O envio da recuperação é individual!

Itens a serem encaminhados – Prazo 09/05/2023:

- Código com as correções (nova implementação) das 3 questões referentes a Avaliação Prática;
- Relatório contendo uma explicação sobre o que foi corrigido (ou implementado diferente)

Questões

1) Escreva um método em python para remover elementos repetidos em uma pilha (estrutura encadeada).

Dia: 02/05/2023

- Este é um programa que remove os elementos duplicados de uma lista encadeada;
- Não criamos uma class Pilha;
- Criamos uma função em vez de método;
- O programa usa uma lista para implementar a estrutura de pilha, em vez de criar uma classe Pilha própria e usar seus métodos empilhar, desempilhar e esta_vazia. O código da função remover_duplicatas usa a estrutura da lista para remover os elementos duplicados, quando na verdade deveria ser implementado de acordo com a estrutura da pilha.

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def remove_duplicates(head):
    current = head
    while current is not None and current.next is not None:
        if current.value == current.next.value:
            current.next = current.next.next
        else:
            current = current.next
    return head


head = Node(3)
head.next = Node(3)
head.next.next = Node(3)
head.next.next.next = Node(6)
head.next.next.next.next = Node(6)
head.next.next.next.next.next = Node(9)
head.next.next.next.next.next.next = Node(10)
head.next.next.next.next.next.next.next = Node(12)


current = head
while current is not None:
    print(current.value, end=" → ")
    current = current.next
print("None")


head = remove_duplicates(head)


current = head
while current is not None:
    print(current.value, end=" → ")
    current = current.next
print("None")
```

- O código usando a classe Pilha é mais apropriado para o problema de remover duplicatas de uma sequência de elementos, pois utiliza a estrutura de dados adequada para tal tarefa. Enquanto o código original usa uma lista encadeada, que não é uma pilha e não oferece a mesma eficiência para manipulação de elementos em uma sequência que a pilha oferece;
- Foi criado desta vez a classe Pilha;
- A pilha é utilizada para armazenar uma sequência de números inteiros;
- O método "remover_duplicatas" é utilizado para remover elementos duplicados da pilha. A estratégia utilizada é criar um conjunto de itens já vistos, e percorrer a pilha do topo até o fundo. Para cada item da pilha, se ele não está no conjunto de itens vistos, ele é adicionado na lista de resultado e no conjunto de itens vistos. No final, a pilha original é atualizada com a lista de resultado sem os elementos duplicados;
- O código final mostra a pilha original, chama o método para remover as duplicatas e exibe a pilha resultante sem as duplicatas.

```
class Pilha:
    def __init__(self):
        self.itens = []

    def empilhar(self, item):
        self.itens.append(item)

    def desempilhar(self):
        return self.itens.pop()

    def esta_vazia(self):
        return len(self.itens) == 0

    def remover_duplicatas(self):
        itens_vistos = set()
        resultado = []
        while not self.esta_vazia():
            item = self.desempilhar()
            if item not in itens_vistos:
                resultado.append(item)
                itens_vistos.add(item)
        resultado.reverse()
        self.itens = resultado


s = Pilha()
s.empilhar(3)
s.empilhar(3)
s.empilhar(3)
s.empilhar(6)
s.empilhar(6)
s.empilhar(9)
s.empilhar(10)
s.empilhar(12)

print("Pilha original:", s.itens)
s.remover_duplicatas()
print("Pilha sem duplicatas:", s.itens)
```

2) Escreva um método em python para remover todos elementos de um range específico em um lista (estrutura encadeada).

Dia: 02/05/2023

- O código original está errado porque não realiza a remoção dos elementos da lista que estão dentro do range especificado. Ele apenas verifica se o elemento atual está dentro do range e, em seguida, passa para o próximo elemento.
- Não eliminamos todo os NODOS dentro de um determinado range;
- O primeiro código não tem uma função para remover nós de uma lista encadeada com base em um intervalo de valores;
- A lógica de remoção não está adequada, pois a remoção de um nó pode afetar a referência do próximo nó na lista, fazendo com que alguns nós sejam perdidos na remoção.

```
class Nodo:
    def __init__(self, dado=None, proximo=None):
        self.dado = dado
        self.proximo = proximo

class ListaEncadeada:
    def __init__(self):
        self.cabeca = None

    def __str__(self):
        resultado = ""
        nodo_atual = self.cabeca
        while nodo_atual:
            resultado += str(nodo_atual.dado) + " -> "
            nodo_atual = nodo_atual.proximo
        resultado += "None"
        return resultado

    def insere_no_inicio(self, valor):
        novo_nodo = Nodo(valor, self.cabeca)
        self.cabeca = novo_nodo

    def remove(self, valor):
        assert self.cabeca, "Não foi possivel remover valor de lista
vazia."

        if self.cabeca.dado == valor:
            self.cabeca = self.cabeca.proximo
        else:
            anterior = None
            corrente = self.cabeca
            while corrente and corrente.dado != valor:
                anterior = corrente
                corrente = corrente.proximo
            if corrente:
                anterior.proximo = corrente.proximo
            else:
                anterior.proximo = None


lista = ListaEncadeada()
for i in range(15):
```

```
    lista.insere_no_inicio(i)
print("Lista:", lista)

print("====*====="*15)
print("
Removendo elementos")
print("====*====="*15)
for i in range(15):
    lista.remove(i)
    print("Removendo o elemento {0}: {1}".format(i, lista))
```

Dia: 09/05/2023

- O segundo código adiciona um novo método chamado remove_range na classe ListaEncadeada, que permite remover todos os nós que contêm um valor dentro de um intervalo determinado. Esse método itera por todos os nós da lista, verifica se o valor está dentro do intervalo e, em caso afirmativo, remove o nó correspondente usando o método remove já definido;
- Foi adicionado o método "remove_range" que recebeu os parâmetros "inicio" e "fim" que definem o range de valores a serem removidos.
- Em seguida, foi percorrida a lista e verificado se o valor do nodo está dentro do range especificado. Caso esteja, o método "remove" é utilizado para remover o valor da lista. Por fim, é impressa a lista após a remoção.

```
class Nodo:
    def __init__(self, dado=None, proximo=None):
        self.dado = dado
        self.proximo = proximo


class ListaEncadeada:
    def __init__(self):
        self.cabeca = None

    def __str__(self):
        resultado = ""
        nodo_atual = self.cabeca
        while nodo_atual:
            resultado += str(nodo_atual.dado) + " -> "
            nodo_atual = nodo_atual.proximo
        resultado += "None"
        return resultado

    def insere_no_inicio(self, valor):
        novo_nodo = Nodo(valor, self.cabeca)
        self.cabeca = novo_nodo

    def remove(self, valor):
        assert self.cabeca, "Não foi possivel remover valor de lista
vazia."

        if self.cabeca.dado == valor:
            self.cabeca = self.cabeca.proximo
        else:
            anterior = None
            corrente = self.cabeca
            while corrente and corrente.dado != valor:
```

```
                anterior = corrente
                corrente = corrente.proximo
            if corrente:
                anterior.proximo = corrente.proximo
            else:
                anterior.proximo = None

    def remove_range(self, inicio, fim):
        assert self.cabeca, "Não foi possivel remover valores de
lista vazia."

        nodo_atual = self.cabeca
        while nodo_atual:
            if inicio <= nodo_atual.dado <= fim:
                self.remove(nodo_atual.dado)
            nodo_atual = nodo_atual.proximo


lista = ListaEncadeada()

lista.insere_no_inicio(3)
lista.insere_no_inicio(5)
lista.insere_no_inicio(6)
lista.insere_no_inicio(8)
lista.insere_no_inicio(10)
lista.insere_no_inicio(12)
print("Lista:", lista)

inicio = 2
fim = 7
lista.remove_range(inicio, fim)
print("Lista após a remoção:", lista)
```

3) Escreva uma função em python para somar todos os valores de uma fila (estrutura encadeada) de forma recursiva.

Dia: 02/05/2023

- O código original utiliza um loop para percorrer todos os elementos da fila e ir somando os valores, sem utilizar recursividade, o que já não está de acordo com o que foi pedido;
- O código apresentado não usa recursão porque a função somatorioFila é implementada usando um loop while para percorrer todos os elementos da fila.
- Embora esse código possa funcionar em alguns casos, ele não é a solução mais adequada para a tarefa proposta pelo professor de somar todos os valores da fila de forma recursiva.

```
from queue import Queue

def somatorioFila(fila: Queue) -> int:
    soma = 0
    while not fila.empty():
        elemento = fila.get()
        if isinstance(elemento, Queue):
            soma += somatorioFila(elemento)
        else:
```

```
            soma += elemento
    return soma

if __name__ == "__main__":
    filaX = Queue()
    filaX.put(25)
    filaX.put(Queue())
    filaX.put(4)
    filaX.put(Queue())
    filaX.put(Queue())
    filaX.put(555)
    print(f"A soma dos elementos da fila é {somatorioFila(filaX)}")
```

Dia: 09/05/2023

- O segundo código é uma nova implementação que segue as regras da recursividade;
- Ele contém uma classe Fila, que é uma estrutura de dados que permite inserir elementos no final e retirar elementos no início da fila;
- A classe tem dois métodos principais, o push, para inserir um novo dado na fila, e o pop, para retirar o primeiro dado da fila;
- A função soma_lista recebe um nó da lista encadeada e retorna a soma dos valores de todos os nós na lista. A função é implementada recursivamente, somando o valor do nó atual e chamando a função novamente com o próximo nó, até que chegue ao final da lista.

```
class Nodo:

    def __init__(self, dado=0, proximo_no=None):
        self.dado = dado
        self.proximo = proximo_no

    def __repr__(self):
        return f"{self.dado} -> {self.proximo}"

class Fila:

    def __init__(self):
        self.primeiro = None
        self.ultimo = None

    def __repr__(self):
        return f"[{self.primeiro}]"

    def push(self, novo_dado):

        novo_no = Nodo(novo_dado)

        if self.primeiro is None:
            self.primeiro = novo_no
            self.ultimo = novo_no
        else:
            self.ultimo.proximo = novo_no
            self.ultimo = novo_no

    def pop(self):

        assert self.primeiro is not None,
        self.primeiro = self.primeiro.proximo
```

```
        if self.primeiro is None:
            self.ultimo = None


def soma_lista(corrente):
    if not corrente:
        return 0
    return corrente.dado + soma_lista(corrente.proximo)


fila = Fila()
for i in range(15):
    fila.push(i)
print(f"Fila formada: {fila}")
soma = soma_lista(fila.primeiro)
print(f"Soma dos elementos da fila: {soma}")
```