Estruturas de dados lineares - Alocação encadeada - Prática

Marco A L Barbosa malbarbo.pro.br

1) Para criar um encadeamento de nós com os itens 7, 1 e 2 podemos escrever `No(7, No(1, No(2, None)))`. Quando o Python avalia esta expressão, o primeiro nó criado contém o 2, o segundo o 1 e o terceiro o 7. Escreva um trecho de código que crie o mesmo encadeamento mas que os nós sejam criados na ordem: primeiro o nó com o 7, depois o nó com o 1 e depois o nó com o 2.

2) Projete uma função que receba como parâmetro uma lista de números (`list[int]`) e devolva um encadeamento com os mesmos itens da lista.

3) Projete uma função que receba como parâmetro uma lista de números (`list[int]`) e devolva um encadeamento com os mesmos itens da lista, mas em ordem contrária.

4) Projete uma função que receba como parâmetro o início de um encadeamento (nó ou `None`) e determine quantos elementos existem no encadeamento.

```
def num_itens(p: No | None) -> int:
    '''
    Determina quantos itens existem no encadeamento
    que começa com *p*.

    Exemplos
    >>> num_itens(None)
    0
    >>> num_itens(No(10, None))
    1
    >>> num_itens(No(20, No(10, None)))
    2
    >>> num_itens(No(4, No(20, No(10, None))))
    3
    '''
    return 0
```

5) Projete uma função que receba como parâmetro o início de um encadeamento (nó ou `None`) e determine a soma dos itens no encadeamento.

6) Projete uma função que receba como parâmetro o início de um encadeamento (nó ou `None`) e modifique cada nó do encadeamento somando 1 ao item do nó.

```
def soma1(p: No | None):
    '''
    Modifica cada nó do encadeamento que começa em *p* somando
    1 ao item do nó.

    Exemplos
    >>> p = No(10, No(20, No(30, None)))
    >>> soma1(p)
    >>> p
    No(item=11, prox=No(item=21, prox=No(item=31, prox=None)))
    '''
    return
```

7) Projete uma função que receba como parâmetro um nó que representa o início de um encadeamento e encontre o valor máximo entre todos os itens do encadeamento.

8) Projete uma função que receba como parâmetro o início de um encadeamento (nó ou `None`) e devolva o início de um outro encadeamento que é uma cópia do encadeamento de entrada.

```
def copia(p: No | None) -> No | None:
    '''
    Cria e devolve uma cópia do encadeamento que inicia em *p*.

    Exemplos
    >>> p = No(10, No(20, No(30, None)))
    >>> q = copia(p)
    >>> # quando mudamos p,
    >>> # q não é alterado pois é uma cópia
    >>> p.item = 1
    >>> p.prox.item = 2
    >>> p.prox.prox.item = 3
    >>> p
    No(item=1, prox=No(item=2, prox=No(item=3, prox=None)))
    >>> q
    No(item=10, prox=No(item=20, prox=No(item=30, prox=None)))

    Exemplo para o None
    >>> copia(None)
    '''
    return None
```

9) Projete uma função que receba como parâmetro o início de um encadeamento (nó ou `None`) e modifique o encadeamento duplicando cada nó (a cópia de cada nó deve ficar após o nó original no encadeamento). Seria possível fazer os exemplos a seguir funcionar se as cópias dos nós ficassem antes dos nós originais? Explique.

```
def duplica_nos(p: No | None):
    '''
    Modifica o encadeamento que começa em *p* criando uma
    cópia de cada nó e colocando a cópia após o nó original
    no encadeamento.

    Exemplos
    >>> p = No(1, No(2, None))
    >>> duplica_nos(p)
    >>> p
    No(item=1, prox=No(item=1, prox=No(item=2, prox=No(item=2, prox=None))))
    >>> # A modificação do primeiro
    >>> # não pode alterar o segundo!
    >>> p.item = 20
    >>> p.prox.item
    1
    '''
    return
```

10) Modifique a especificação do TAD Pilha e a implementação que usa encadeamento para que a função `desempilha` devolva `None` caso a pilha esteja vazia.

11) Modifique a função `grupos_corretos` (veja o material de estruturas de dados lineares com alocação contígua) para que ela use a versão do TAD Pilha que devolve `None` quando a pilha está vazia. Qual das duas versão você prefere? Por que?

12) Modifique a especificação do TAD Fila e a implementação que usa encadeamento com início e fim para que a função `desenfileira` devolva `None` caso a fila esteja vazia.

13) O professor Confuso da Silva apresentou a seguinte implementação de Fila para os alunos

```
class Fila:
    inicio: No | None
    fim: No | None

    def __init__(self) -> None:
        self.inicio = None
        self.fim = None

    def enfileira(self, item: str):
        if self.fim is None:
            self.inicio = No(item, None)
            self.fim = self.inicio
        else:
            self.fim.prox = No(item, None)
            self.fim = self.fim.prox

    def desenfileira(self) -> str:
        if self.inicio is None:
            raise ValueError('fila vazia')
        item = self.inicio.item
        self.inicio = self.inicio.prox
        return item

    def vazia(self) -> bool:
        return self.inicio is None
```

Crie um exemplo de uso de fila que demostre que implementação está incorreta. Explique e corrija a implementação.

14) Projete uma função que receba como parâmetro uma lista de números (`list[int]`) e devolva um encadeamento duplo com os mesmos itens da lista.

15) Projete uma função que receba como parâmetro um nó `p` de um encadeamento duplo, onde `p.prox` não é `None`, e troque de lugar os nós `p` e `p.prox`.

16) Projete uma função que receba como parâmetro um nó `p` de um encadeamento duplo, onde `p.ante` não é `None`, e troque de lugar os nós `p` e `p.ante`.

17) Implemente o TAD Lista usando encadeamento simples. Dica: use um nó sentinela.

18) Implemente o TAD Lista usando encadeamento duplo com sentinela. Mantenha um campo com a quantidade de elementos na lista e use essa informação para implementar as funções de inserção e remoção em posição navegando pela menor quantidade de nós possível.

19) (Desafio) Suponha que o Python não permitisse estruturas com autorreferência e crie um esquema para fazer encadeamento duplo usando arranjos. Implemente uma fila dupla usando esse esquema. Dica: crie um arranjo de nós, onde os campos `ante` e `prox` de cada nó são índices para elementos do arranjo.