

Estruturas de dados lineares

Alocação encadeada

Marco A L Barbosa

malbarbo.pro.br

O código inicial dos exercícios está disponível na página <https://malbarbo.pro.br/ensino/2025/6884/>.

Introdução

- 1) Qual é utilidade do `None`?
- 2) O que é uma autorreferência na definição de um tipo de dado?
- 3) Qual é a utilidade de um tipo de dado com autorreferência?

Encadeamento simples

- 4) 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.
- 5) 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. A implementação deve analisar os elementos da lista de entrada na ordem que eles aparecem na lista.
- 6) Projete uma função que receba como parâmetro o início de um encadeamento (nó ou `None`) e determine quantos elementos existem no encadeamento (código inicial em `num_itens.py`).

```
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
```

- 7) Projete uma função que receba como parâmetro o início de um encadeamento (nó ou `None`) e determine a soma dos itens no encadeamento.
- 8) 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 (código inicial em `soma1.py`).

```
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
```

- 9) 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.
- 10) Projete uma função que receba como parâmetro um número e o início de um encadeamento (nó ou `None`) e adicione o número (novo nó) no fim no encadeamento. A função deve devolver o (novo) início do encadeamento.
- 11) Projete uma função que receba como parâmetro uma lista de números (`list[int]`) e devolva um encadeamento com os mesmos itens da lista. A implementação deve analisar os elementos da lista de entrada na ordem que eles aparecem na lista e deve ter complexidade de tempo $O(n)$, onde n é a quantidade de elementos da lista.
- 12) 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 (código inicial em `copia.py`).

```
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
```

- 13) 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. (código inicial em `duplica_nos.py`)

```
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

```

Pilhas e filas

- 14) Modifique a especificação do TAD Pilha e a implementação que usa encadeamento para que a função `desempilha` devolva `None` caso a pilha esteja vazia (modifique o arquivo `pilha_encadeamento.py`).
- 15) 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? (modifique o arquivo `grupos_corretos.py`)
- 16) Modifique o TAD Pilha e adicione o método `inverte`, que inverte a ordem dos elementos da pilha. Implemente o método para pilhas com encadeamento (modifique o arquivo `pilha_encadeamento.py`).
- 17) 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 demonstre que a implementação está incorreta. Explique e corrija a implementação.

- 18) 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 (modifique o arquivo

`fila_encadeada_inicio_fim.py`).

- 19) Modifique o TAD Fila e adicione o método `junta(a, b)`, que move todos os elementos da fila `b` para o final da fila `a`. Implemente o método para filas com encadeamento. A sua implementação deve ter complexidade de tempo de $O(1)$ (modifique o arquivo `fila_encadeada_inicio_fim.py`).
- 20) Diferente de uma fila comum, que obedece a regra “o primeiro a entrar é o primeiro a sair”, em uma fila de prioridades, cada item tem um prioridade e o item que deve sair primeiro é o que tem a maior prioridade. Projete um TAD para uma fila de prioridade e implemente usando encadeamento. Qual é a complexidade de tempo das operações de enfileira e desenfileira?

Encadeamento duplo

- 21) Projete uma função que receba como parâmetro uma lista de números (`list[int]`) e devolva um encadeamento duplo com os mesmos itens da lista. A implementação deve analisar os elementos da lista de entrada na ordem que eles aparecem na lista.
- 22) Projete uma função que receba como parâmetro uma lista de números (`list[int]`) e devolva um encadeamento duplo com os mesmos itens da lista, mas em ordem contrária. A implementação deve analisar os elementos da lista de entrada na ordem que eles aparecem na lista.
- 23) 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`.
- 24) 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`.
- 25) Projete uma função que receba como parâmetro o início e fim de um encadeamento duplo (dois nós), e determine se o encadeamento é palíndromo, isto é, os elementos são os mesmo do início para o fim e do fim para o início.
- 26) Projete uma função que receba como parâmetro o início de um encadeamento duplo (nó ou `None`), e modifique o encadeamento invertendo a ordem dos elementos. A sua função não deve criar nenhum nó e deve devolver o (novo) início do encadeamento.
- 27) Projete uma função que receba como parâmetro um número e o início de um encadeamento duplo (nó ou `None`), onde os itens estão em ordem não decrescente, e insira o número (em um novo nó) no encadeamento mantendo ele ordenado. A sua função deve devolver o (novo) início do encadeamento.

Listas

- 28) Implemente o TAD Lista usando encadeamento simples. Dica: use um nó sentinela.
- 29) 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.
- 30) Proponha um forma de implementar o TAD Lista com encadeamento duplo que permita inverter a ordem dos elementos da lista em tempo $O(1)$.
- 31) Projete e implemente um TAD que representa um conjunto (coleção de elementos únicos). O TAD deve conter operações de inserção, remoção e verificação de pertinência de valores. Inclua também operações de união e interseção de dois conjuntos (a implementação deve ter tempo $(n + m)$, onde n e m são os tamanhos dos conjuntos).

Desafios

- 32) Supondo que o Python não permitisse estruturas com autorreferência, 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.