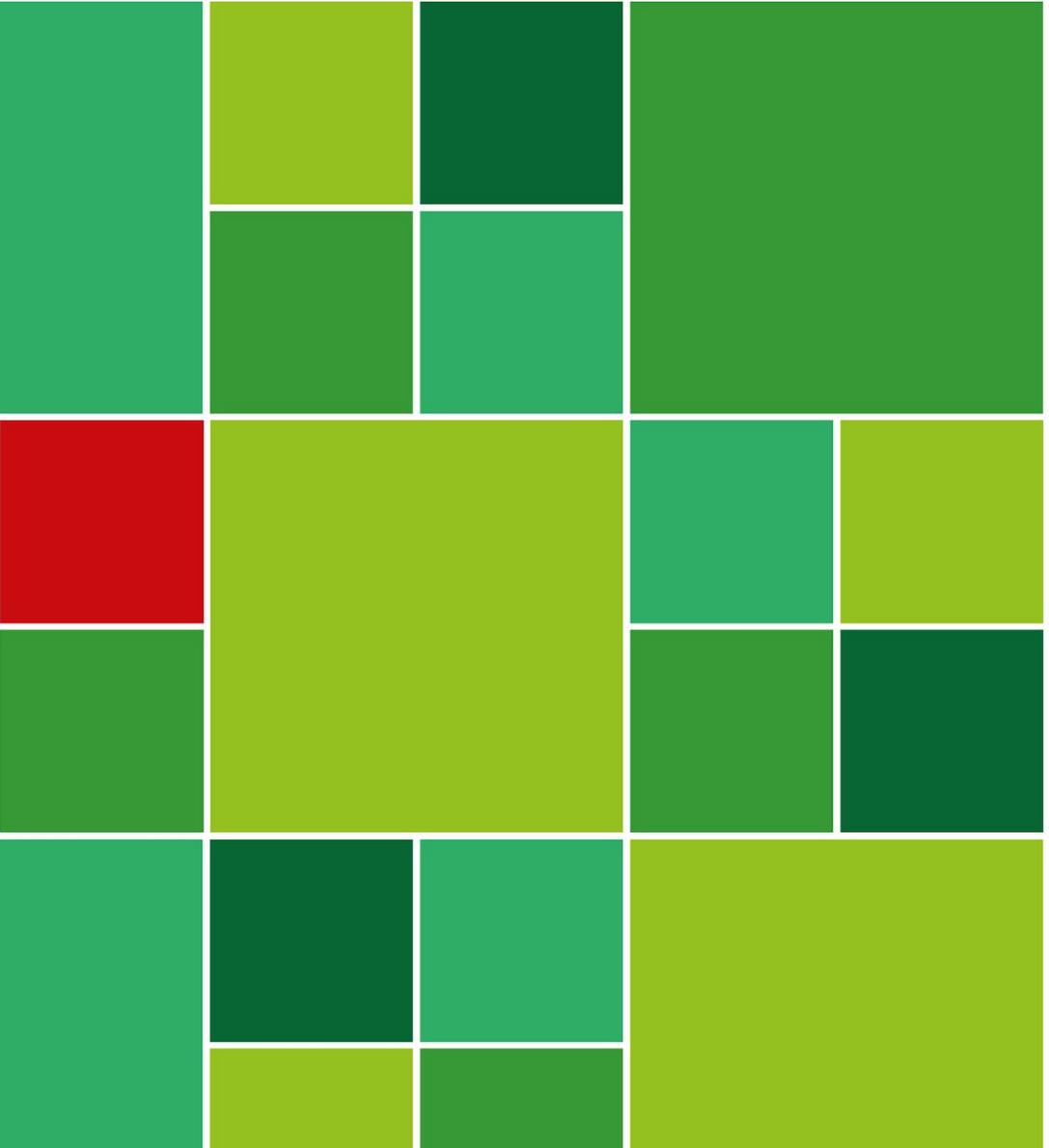


Estrutura de Dados

Listas Encadeadas

Prof. Dr. Danilo Barbosa

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO

O que vamos ver nessa aula?

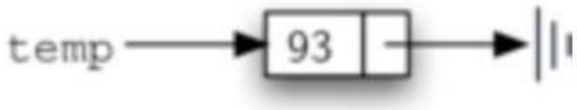
- Listas encadeadas

Listas Encadeadas

- Já utilizamos a lista (list) em Python
  - Coleção de itens poderosa e simples, que fornece ao programador uma variedade de operações
- Mas nem todas as linguagens de programação incluem uma lista
  - Nesses casos a lista deve ser implementada pelo programador.
- Uma lista é uma coleção de itens em que cada item tem uma posição relativa em relação aos outros.

Listas Encadeadas

- No entanto, não conseguimos garantir que os elementos armazenados vão ocupar um espaço de memória contíguo
  - Portanto, não temos acesso direto aos elementos da lista
- Para percorrer todos os elementos da lista, devemos explicitamente guardar o seu encadeamento
  - Isso é feito armazenando-se um ponteiro para o próximo elemento junto com a informação de cada elemento



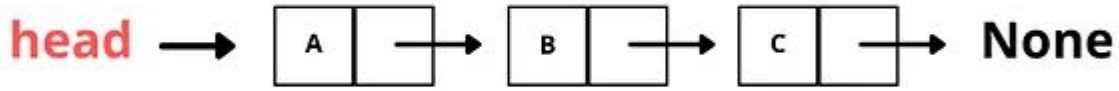
Listas Encadeadas

- A estrutura consiste em uma sequência encadeada de elementos
  - Nós da lista
- Um nó da lista é representado por dois campos:
  - Informação armazenada (valor)
  - Referência para o próximo elemento da lista
- A lista é representada por uma referência ao primeiro elemento da lista
  - Cabeça (head) da lista

Listas Encadeadas

- Do primeiro elemento, podemos alcançar o segundo, seguindo o encadeamento, e assim por diante

- O último nó da lista armazena, como próximo nó, um ponteiro inválido com valor None e sinaliza assim que não existe próximo nó



Listas Encadeadas

- O bloco básico para a construção da implementação da lista é o nó (Node)
  - Possui as duas informações: informação e a referência para o próximo nó

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

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext
```

Listas Encadeadas

- Como dissemos, a lista será construída a partir de uma coleção de nós, cada um ligado ao próximo por referências explícitas
  - É necessário encontrar o primeiro nó (que contém o primeiro item), cada item pode ser encontrado seguindo sucessivamente os próximos itens
  - Então é necessário manter uma referência ao primeiro nó
- O valor None indica que a cabeça da lista não se refere a nada

```
class UnorderedList:

    def __init__(self):
        self.head = None
```

```
>>> mylist = UnorderedList()
```

Listas Encadeadas

- O método isEmpty() verifica se a cabeça da lista é None
  - Será verdadeiro se não há nós na lista encadeada

```
def isEmpty(self):
    return self.head == None
```

Listas Encadeadas

- E aí como inserimos itens na lista?
  - Nesse caso, vamos inserir o novo nó no lugar mais fácil da lista que nesse caso é a cabeça

- Cada item na lista fica em um objeto Node
  - Então é necessário criar um objeto Node, fazer com que sua referência próximo aponte para a cabeça e fazer com que a cabeça vire esse novo nó

```
def add(self,item):
    temp = Node(item)
    temp.setNext(self.head)
    self.head = temp
```

Listas Encadeadas

- size(), search() e remove() vão utilizar uma técnica chamada de varredura de lista encadeada
  - Vai se percorrer a lista

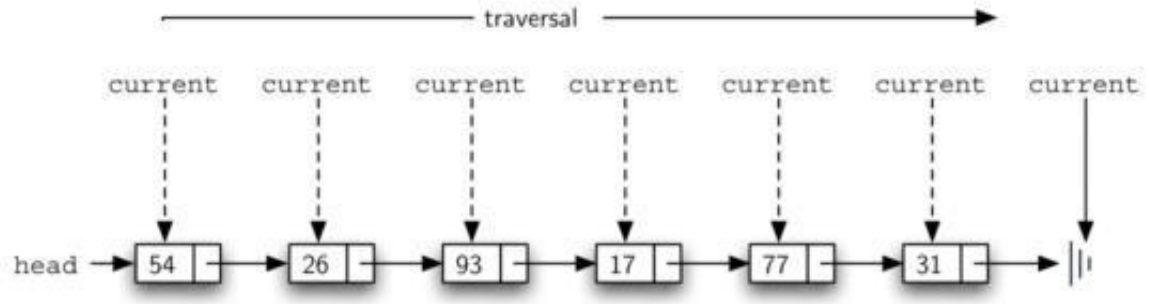
- Varredura refere-se ao processo de visitar sistematicamente cada nó
  - Ao visitarmos cada nó, movemos a referência para o próximo nó percorrendo a próxima referência

Listas Encadeadas

- size() percorremos a lista encadeada e contamos o número de nós que existe

```
def size(self):
    current = self.head
    count = 0
    while current != None:
        count = count + 1
        current = current.getNext()

    return count
```

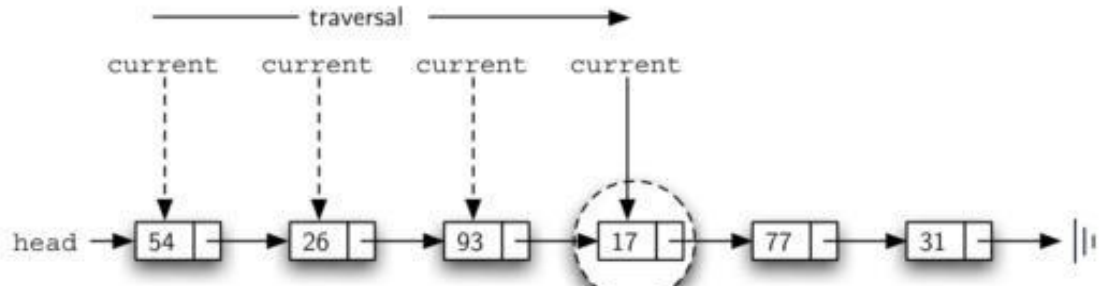


Listas Encadeadas

- search() a varredura começa na cabeça da lista e enquanto não chega no fim da lista ou não encontra o item, vai procurando o item

```
def search(self,item):
    current = self.head
    found = False
    while current != None and not found:
        if current.getData() == item:
            found = True
        else:
            current = current.getNext()

    return found
```



Listas Encadeadas

- remove() requer dois passos
  - Procurar o item que queremos remover
  - Quando o item for encontrado, devemos removê-lo

```
def remove(self,item):
    current = self.head
    previous = None
    found = False
    while not found:
        if current.getData() == item:
            found = True
        else:
            previous = current
            current = current.getNext()

    if previous == None:
        self.head = current.getNext()
    else:
        previous.setNext(current.getNext())
```

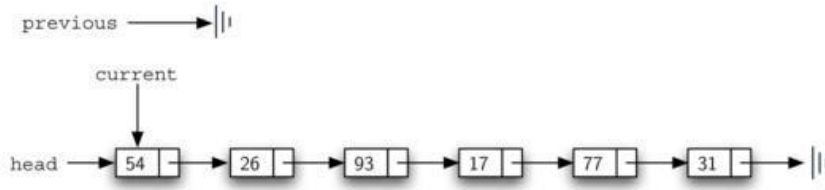
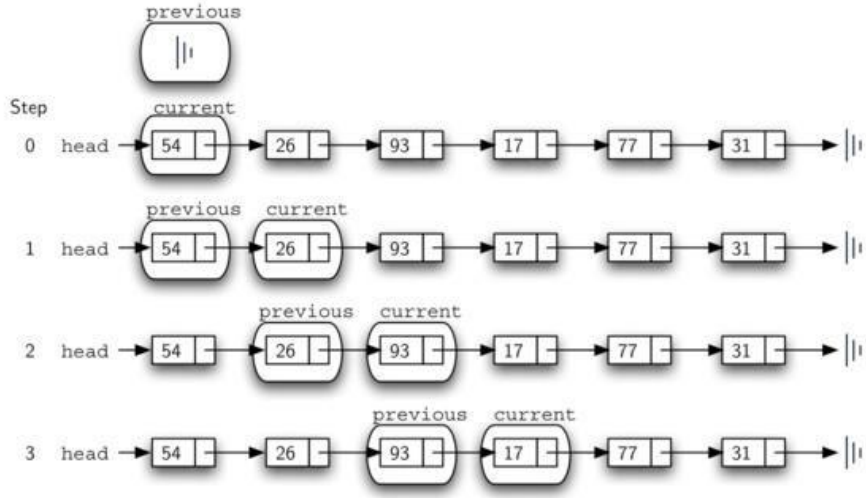


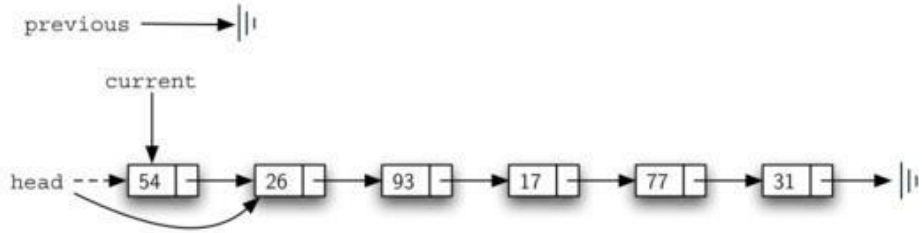
Figura 11: Valores Iniciais de `previous` and `current`



Listas Encadeadas

```
def remove(self,item):
    current = self.head
    previous = None
    found = False
    while not found:
        if current.getData() == item:
            found = True
        else:
            previous = current
            current = current.getNext()

    if previous == None:
        self.head = current.getNext()
    else:
        previous.setNext(current.getNext())
```



Exercício

- Exercício - Implemente os métodos (definições do método a seguir):
  - append(item)
    - https://www.programiz.com/python-programming/methods/list/append
  - insert(index, item)
    - https://www.programiz.com/python-programming/methods/list/insert
  - pop(index)
    - https://www.programiz.com/python-programming/methods/list/pop
  - Index(item, start, end)
    - https://www.programiz.com/python-programming/methods/list/index

Exercício

As listas encadeadas simples são normalmente utilizadas para relacionar itens que precisam ser exibidos ou manipulados por meio de estruturas dinâmicas. Em relação a manipulação de uma lista encadeada simples, analise os itens:

I. É preciso inicializar a lista antes de inserir algum elemento.

II. A inclusão de um elemento em uma lista encadeada simples pode ser realizada somente de duas maneiras: no início e no final da lista.

Exercício

III. Um elemento de uma lista encadeada simples não pode ser excluído quando está no meio da lista.
IV. Uma lista encadeada está vazia se ela aponta para nulo.
São verdadeiros somente os seguintes:
Alternativas
A Apenas I, e V.
B Apenas I e II
C Apenas I e IV
D Apenas, II e V.
E Apenas, III e V.

Exercício

Em relação ao tipo de estrutura de dados conhecido como lista ligada ou lista encadeada, é correto afirmar que:
Alternativas
A Um elemento deve entrar por uma extremidade e ser removido pela outra extremidade.
B Não é uma estrutura flexível, pois há necessidade de definição de um tamanho máximo de elementos.
C O primeiro elemento que entrar só poderá ser removido por último, após todos os outros elementos serem removidos.

Exercício

Em relação ao tipo de estrutura de dados conhecido como lista ligada ou lista encadeada, é correto afirmar que:
Alternativas
D É uma estrutura multidimensional e homogênea.
E A sucessão dos elementos é determinada por um ponteiro que indica a posição do próximo elemento.

Exercício

Uma _____ é uma estrutura de dados linear que apresenta uma série de nós, cada um contendo um elemento de dados, uma referência para o nó anterior e outra referência para o próximo nó, permitindo, assim, a manipulação de elementos em ambas as direções. Assinale a alternativa que preenche corretamente a lacuna do trecho acima.

Alternativas

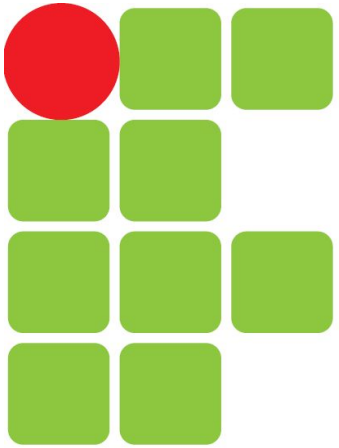
A árvore B pilha C lista encadeada D lista duplamente encadeada E lista circular

Exercício

1 - C
2- E
3- D

Referência

- MENEZES, N. N. C. Introdução à programação com Python. Algoritmos e lógica de programação para iniciantes. São Paulo: Editora Novatec, 2019.
- MILLER, B. N.; RANUM, D. L. Resolução de Problemas com Algoritmos e Estruturas de Dados usando Python. Disponível em: <https://panda.ime.usp.br/panda/static/pythonds_pt/index.html> Acesso em: 10 nov. 2024.



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO