Professor
Edson Pedro FERLIN

Estrutura de Dados

Estruturas Lineares

Prof. Edson Pedro Ferlin

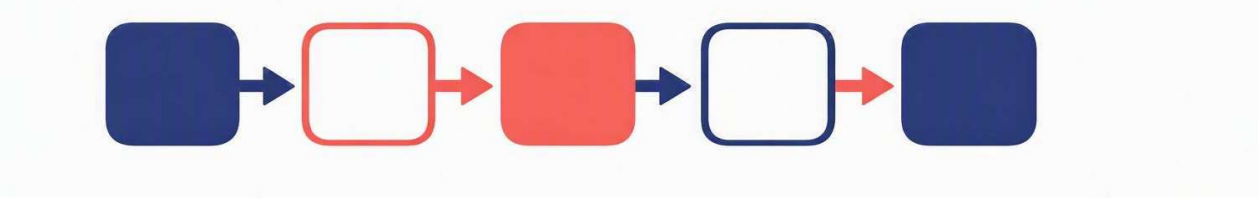
Professor
Edson Pedro FERLIN

Estrutura de Dados

AULA 2 - ESTRUTURAS LINEARES

Estruturas Lineares

Vetores • Listas Encadeadas • Filas • Pilhas

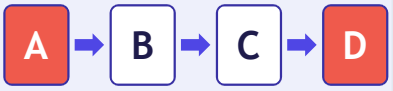


O que são Estruturas Lineares?

Uma estrutura linear organiza seus elementos em sequência, na qual cada elemento possui, no máximo, um predecessor e um sucessor.

- **Organização sequencial**
- **Acesso aos elementos**
- **Inserção e remoção**
- **Eficiência**

SEQUÊNCIA



Cada elemento aponta para o próximo da sequência.

Classificação

Estáticas	Dinâmicas
Vetores	Listas Encadeadas
Tamanho fixo	Tamanho variável
Memória contínua	Memória distribuída

Comparação Geral

Estrutura	Acesso	Inserção	Remoção
Vetor	Muito rápido	Média	Média
Lista	Lento	Rápida	Rápida
Pilha	Topo	Muito rápida	Muito rápida
Fila	Extremidades	Muito rápida	Muito rápida

Vetores

Vetores são coleções de elementos do mesmo tipo armazenados em posições consecutivas de memória.

Características dos Vetores

- Memória contínua
- Índices
- Acesso direto
- Tamanho fixo

Vetores em Python

```python
numeros = [10, 20, 30, 40, 50]
print(numeros)
```

Criando e exibindo um vetor (lista) em Python.

Acessando Elementos do Vetor

```python
nomes = ["Ana", "Carlos", "Maria"]
print(nomes[0])
print(nomes[2])
```

Acesso direto pelo índice — a contagem começa em 0.

Percorrendo Vetores

```python
idades = [18, 20, 25, 30]
for idade in idades:
    print(idade)
```

Percorrendo todos os elementos com um laço for.

Inserção no Vetor

```python
numeros = [10, 20, 40]
numeros.insert(2, 30)
print(numeros)
```

insert(posição, valor) adiciona um elemento na posição indicada.

Remoção

```python
numeros = [10, 20, 30, 40]
numeros.remove(20)
print(numeros)
```

remove(valor) exclui a primeira ocorrência do valor.

Atividade - Vetores

Faça um programa que:

- leia 10 números;
- encontre o maior;
- encontre o menor;
- calcule a média.

Strings em Python

- Sequência de caracteres usada para representar texto em Python;
- Strings podem conter letras, números, espaços e símbolos;
- Cada caractere é acessado por um índice que começa em 0;
- São do tipo **str**.

```
minhaString = "Estamos em 2025."
print(minhaString)
print(type(minhaString))
```

```
Estamos em 2025.
<class 'str'>
```

Acessando usando índices

- Índices iniciam sempre no zero;
- Índices podem ser negativos ("contagem de trás para frente").

```
indices = "String Python"
print(indices[0])
print(indices[2])
print(indices[-1])
print(indices[-3])
```

```
S
r
n
h
```


Armazenamento em memória de Strings

- Strings são alocadas sequencialmente na memória
- Cada caracter ocupa 2 espaços da memória (2 bytes)

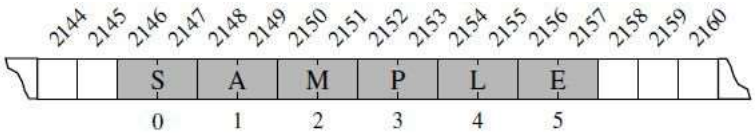


Figure 5.2: A Python string embedded as an array of characters in the computer's memory. We assume that each Unicode character of the string requires two bytes of memory. The numbers below the entries are indices into the string.

Listas Lineares

- Uma lista é uma coleção ordenada, mutável e que permite duplicatas;
- Representada por colchetes [];
- Os elementos podem ser de qualquer tipo (números, strings, outras listas, etc);
- Os elementos podem ser acessados utilizando índices numéricos que iniciam no 0;
- Possui muitos métodos que permitem realizar operações (semelhante as strings).

```
nomes = ['Rodrigo', 'José', 'Olívia']
print(nomes[1])
print(nomes[0])
```

```
José
Rodrigo
```

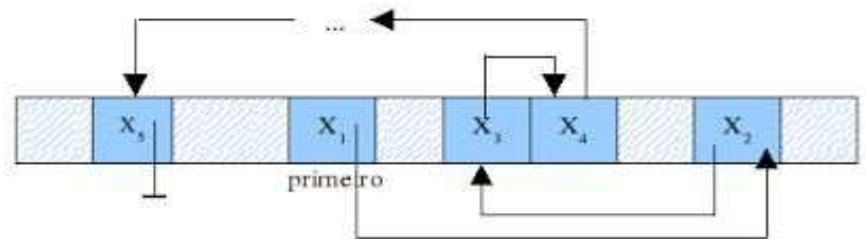

Listas Lineares

- São as estruturas de dados de manipulação mais simples;
- Uma lista agrupa informações referentes a um conjunto de elementos que se relacionam entre si;
- É um conjunto de $n \geq 0$ nós l[1], l[2], ..., l[n] tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da seqüência linear:
  - Se $n > 0$, l[1] é o primeiro nó
  - Para $1 < k \leq n$, o nó l[k] é precedido por l[k – 1]

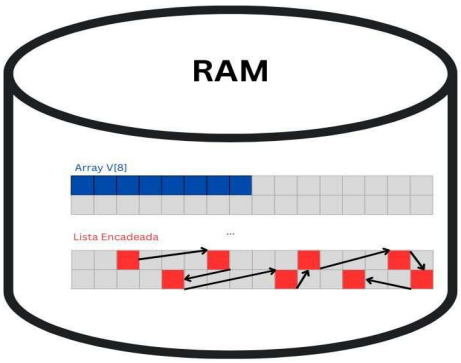
Operações comuns em Listas Lineares

- Inserção
  - Adicionar um nó na lista
- Remoção
  - Deletar um nó na lista
- Busca
  - Procurar um nó na lista

Lista Linear - Representação na memória



Representação na memória



Operações Básicas com Listas Lineares

```
# Modificando um item da lista
nomes[1] = "Clodoaldo"
print("Após modificar o segundo item:", nomes)

# Adicionando novos elementos
nomes.append("Lamine")              # adiciona ao final
nomes.insert(1, "Yamal")              # insere na posição 1
print("Após adicionar itens:", nomes)

# Removendo elementos
nomes.remove("Olívia")                # remove a primeira ocorrência de "Olívia"
nomes.pop()                          # remove o último item
nomes.pop(0)                         # remove o item da posição 0
print("Após remoções:", nomes)

# Ordenando a lista
nomes.sort()
print("Lista ordenada:", nomes)
```

```
Após modificar o segundo item: ['Rodrigo', 'Clodoaldo', 'Olívia']
Após adicionar itens: ['Rodrigo', 'Yamal', 'Clodoaldo', 'Olívia', 'Lamine']
Após remoções: ['Yamal', 'Clodoaldo']
Lista ordenada: ['Clodoaldo', 'Yamal']
```

Tipos de Listas Lineares

- Alocação Contígua ou Sequencial
  - Nós consecutivos da lista são adjacentes na memória
  - Maneira mais simples de se manter uma lista linear na memória do computador
  - Exige alocação prévia da memória do computador (Alocação Estática)
  - Particularmente atraente no caso de filas e pilhas
- Alocação Encadeada
  - O posicionamento dos nós na memória não guarda relação com o da lista
  - Maior complexidade de implementação (uso de ponteiros para manter a sequência)
  - Mais eficiente para operações de inserção e remoção
  - Não exige alocação prévia da memória do computador (Alocação Dinâmica)

Alocação: Contígua × Encadeada

Contígua (Sequencial)	Encadeada
Nós adjacentes na memória	Nós espalhados na memória
Alocação estática (prévia)	Alocação dinâmica
Acesso direto O(1)	Inserção e remoção eficientes

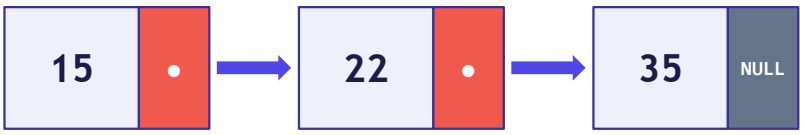
Listas Encadeadas

Lista encadeada é uma coleção de nós ligados por ponteiros.

Cada nó possui:

- **dado**
- **ponteiro para o próximo**

Ilustração - Listas Encadeadas



Cada nó guarda um dado e um ponteiro (coral) para o próximo; o último aponta para NULL.

Listas Encadeadas

- São listas lineares onde a posição na memória dos nós não guarda relação com a posição na lista;
- A posição na lista é mantida através de ponteiros para o próximo nó.

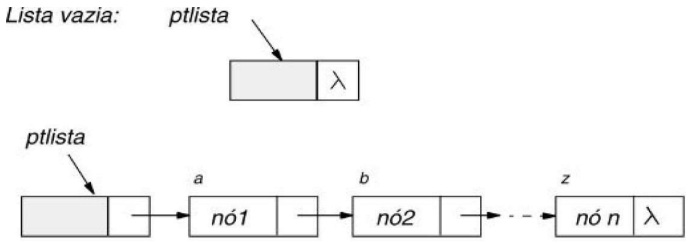


FIGURA 2.5 Lista encadeada com nó-cabeça.

Operação de Inserção - Listas Encadeadas

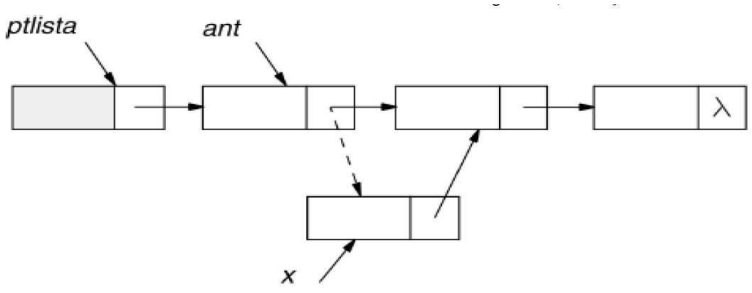


FIGURA 2.6 Inserção de um nó.

Operação de Remoção - Listas Encadeadas

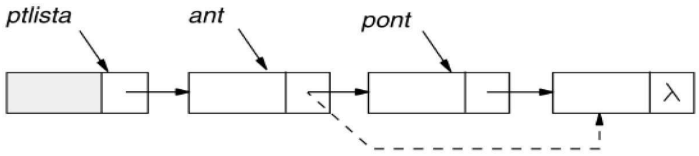


FIGURA 2.7 Remoção de um nó.

As Listas Encadeadas são sempre mais eficientes que as sequenciais?

Depende...

- Apesar das inclusões e remoções em listas encadeadas serem O(1), o acesso a posições do "meio" da lista não é direto
  - O acesso é O(n) em listas encadeadas
  - O acesso em listas encadeadas depende da busca do nó
- Já nas listas sequenciais, temos a situação contrária
  - Pelos nós serem sequenciais em memória, o acesso a qualquer posição é O(1)

Finalizando

- Cada estrutura de dado é mais adequada/eficiente para uma situação diferente;
- Conhecer as diferenças e particularidades de cada uma nos permite empregar a estrutura adequada em cada momento:
  - Listas encadeadas são melhores para situações que não exigem muitos acessos a elementos em posições aleatórias e demandam muitas inserções e remoções nas pontas
  - Listas sequenciais são mais adequadas para situações em que são feitas muitas consultas a elementos em posições aleatórias e poucas operações de inserção e remoção

Implementação Listas Encadeadas em Python

```python
class No:
    def __init__(self, valor):
        self.valor = valor
        self.proximo = None
```

Cada nó guarda um valor e a referência para o próximo.

Listas Encadeadas - Ligando Nós

```python
a = No(10)
b = No(20)
c = No(30)
a.proximo = b
b.proximo = c
```

Encadeando três nós por meio dos ponteiros.

Percorrendo a Lista Encadeada

```python
atual = a
while atual:
    print(atual.valor)
    atual = atual.proximo
```

Percurso sequencial: do primeiro nó até None.

Atividade - Listas Lineares

Inserir um novo nó entre 20 e 30.

Tuplas

- Uma tupla é uma coleção ordenada e imutável;
- Representada por parênteses ();
- Pode conter elementos de tipos variados;
- Ideal para dados que não devem ser modificados;
- Os elementos podem ser acessados utilizando índices numéricos que iniciam no 0.

```
tuplaNomes = ('Rodrigo', 'José', 'Olívia')
print(tuplaNomes[1])
tuplaNomes[1] = 'Clodoaldo' #Erro, porque tuplas são imutáveis
```

```
José
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-811c93028dfe> in <cell line: 0>()
      1 tuplaNomes = ('Rodrigo', 'José', 'Olívia')
      2 print(tuplaNomes[1])
----> 3 tuplaNomes[1] = 'Clodoaldo' #Erro, porque tuplas são imutáveis

TypeError: 'tuple' object does not support item assignment
```


Operações com Tuplas

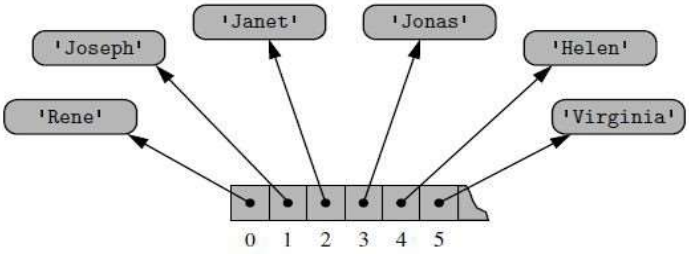
- As Tuplas têm poucos métodos, pois não podem ser alteradas.
  - count(x): retorna a quantidade de vezes que o elemento x aparece na tupla
  - index(x): retorna o índice da primeira ocorrência de x na tupla

```
tuplaNomes = ('Rodrigo', 'José', 'Olívia', 'Rodrigo')
print(tuplaNomes.count('Rodrigo'))
print(tuplaNomes.index('José'))
```

```
2
1
```

Listas e Tuplas em Memória

- Os itens de listas e tuplas são acessados de forma referencial;
- Permite elementos (nós) de diferentes tipos na mesma sequência;



Uso de memória com operações em listas

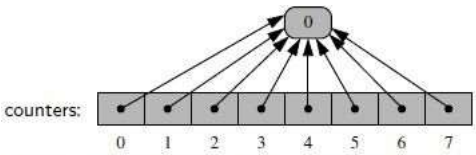


Figure 5.7: The result of the command data = [0] * 8.

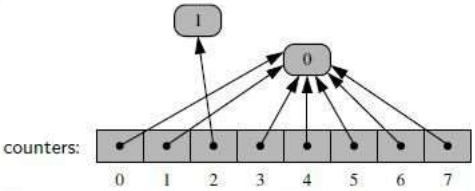


Figure 5.8: The result of command data[2] += 1 upon the list from Figure 5.7.

Eficiência de operações em listas

- Como os nós são alocados sequencialmente, aumentar a lista pode levar o Python a ter que encontrar um novo espaço na memória para armazenar a lista;
- Operações para remover ou adicionar um nó k, exigem o "movimento" de vários outros nós.

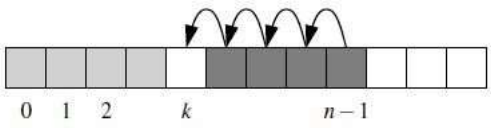


Figure 5.17: Removing an element at index k of a dynamic array.

Vantagens e Desvantagens das Listas Encadeadas

- Crescimento dinâmico;
- Inserção rápida;
- Remoção rápida.

- Não possui acesso direto;
- Consome memória extra;
- Percurso sequencial.

Lista Encadeada × Sequencial

- Inserção e remoção nas pontas são O(1) na lista encadeada;
- O acesso a uma posição aleatória é O(n) na encadeada e O(1) na sequencial;

- Encadeada: melhor quando há muitas inserções e remoções;
- Sequencial: melhor quando há muitas consultas a posições aleatórias.

Complexidades em Listas Lineares

Operação	Lista Sequencial (Array)	Lista Encadeada Simples
Acesso por índice	O(1)	O(n)
Inserção no início	O(n)	O(1)
Inserção no final	O(1)	O(n)
Inserção no meio	O(n)	O(n)
Remoção no início	O(n)	O(1)
Remoção no final	O(1)	O(n)
Remoção no meio	O(n)	O(n)

Operações de Busca

Busca Linear

- Percorre os elementos um a um até encontrar o valor buscado;
- Funciona em qualquer lista, ordenada ou não;
- O algoritmo mais adequado depende de como os dados estão organizados;
- Listas ordenadas permitem algoritmos de busca mais eficientes;
- No pior caso, verifica todos os n elementos.

Operação de Busca

- Para buscar um elemento nos nós de uma lista linear genérica, deve-se percorrer todos seus elementos até encontrar

```
função busca1(x)
   i := 1;     busca1 := 0
   enquanto i≤ n faça
      se L[i] . chave = x então
         busca1 := i          % chave encontrada
         i := n + 1
      senão i := i + 1        % pesquisa prossegue
```


Complexidade do Algoritmo de Busca

- Note que no algoritmo de busca verificamos cada nó da lista comparando com o valor buscado até encontrar o mesmo;
- Logo no pior dos casos, vamos encontrar o valor buscado no último elemento da lista a ser verificado.
- Complexidade: O(n)

Busca em Listas Lineares Ordenadas

Lista Linear Ordenada

- Os elementos seguem uma relação de ordem: crescente, decrescente ou alfabética;
- Conhecendo um elemento, sabemos que os anteriores são menores e os seguintes maiores;
- A ordenação permite descartar partes da lista durante a busca.

Lista Linear Ordenada

- É uma lista linear onde os elementos possuem uma relação de ordem
  - Por exemplo: ordem crescente, decrescente, alfabética, etc
- Isso significa que podemos tirar conclusões sobre os outros elementos da lista através da análise de apenas 1 deles
  - Se a lista é crescente e conhecemos o valor de L[k], podemos concluir que:
    - L[k+1] > L[k], L[k+2] > L[k], …, L[n] > L[k]
    - L[0] < L[k], L[1] < L[k], …, L[k-1] < L[k]

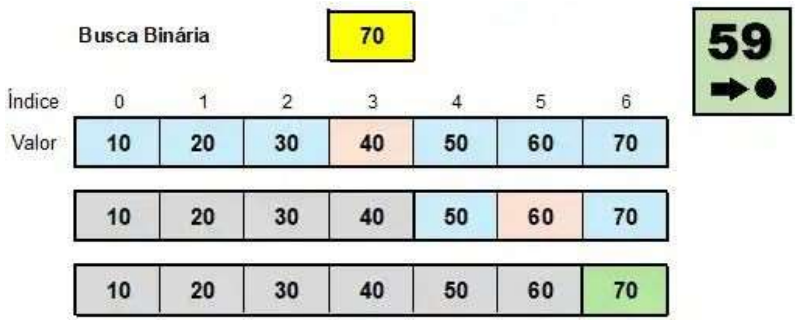
Busca Binária

- As listas lineares ordenadas nos permitem desenvolver um algoritmo de buscas mais eficiente;
- Nessa abordagem, primeiro olhamos o nó posicionado no meio da lista. Caso ele não seja o elemento buscado, podemos excluir metade da lista;
- A operação acima é repetida recursivamente até que o elemento buscado seja encontrado.

Complexidade da Busca Binária

- Compara o elemento central da lista ordenada com o valor buscado;
- A cada passo, descarta metade dos elementos restantes;
- Repete o processo de forma recursiva até encontrar;
- Complexidade: O(log n)

Representação Visual da Busca Binária



Comparação - Busca Linear × Binária

Busca Linear	Busca Binária
Qualquer lista	Somente lista ordenada
Percorre elemento a elemento	Divide a lista ao meio
O(n)	O(log n)

Atividade - Operação de Busca

-Implemente uma busca linear em uma lista linear.

-Implemente uma busca binária em uma lista ordenada.

- Compare o número de operações para buscar o valor 7 em L = [-4, -1, 0, 2, 3, 5, 7, 9, 11, 15].

Exercícios

1) Modifique a implementação de lista encadeada feita em sala para implementar uma lista duplamente encadeada.

Isso significa que além de guardar uma referência para o próximo nó, cada nó também guarda uma referência para o anterior.

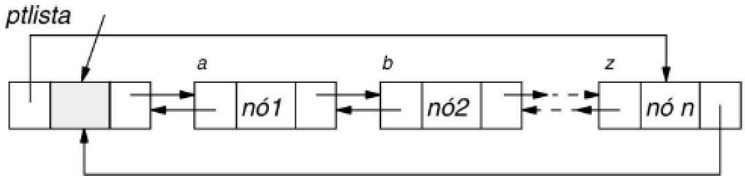


FIGURA 2.10 Lista duplamente encadeada.

2) Implemente um algoritmo de busca em lista linear;

3) Implemente um algoritmo de busca binária em uma lista linear ordenada;

4) Compare o número de operações necessárias para buscar o elemento 7 na lista L usando os dois algoritmos implementados nos exercícios anteriores

L=[-4,-1,0,2,3,5,7,9,11,15]

5) Se utilizarmos a lista duplamente encadeada implementada no exercício 1 para implementar uma fila, suas operações de inserção e remoção seriam mais eficientes do que a fila utilizando listas comuns do Python? Explique.

6) Se utilizarmos a lista duplamente encadeada implementada no exercício 1 para implementar uma fila, suas operações de inserção e remoção seriam mais eficientes do que a fila utilizando listas comuns do Python? Explique.

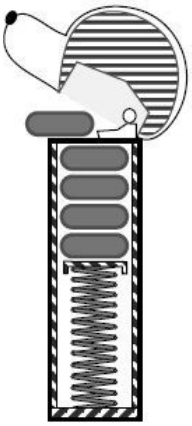
FILAS e PILHAS

LIFO

Pilha – Last In, First Out

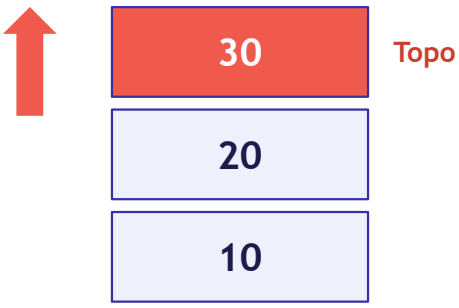
O último elemento a entrar é o primeiro a sair, como uma pilha de pratos.

Pilhas



- Política LIFO (Last In First Out)
  - Último nó a entrar é o primeiro a sair
- Possui dois métodos
  - Push - Insere um nó no topo da pilha
  - Pop - Remove o elemento que está no topo da pilha
- Para implementar deve-se manter uma referência para o topo da estrutura

Ilustração - Pilhas



Operações - Pilhas

Push	Pop	Peek	isEmpty
empilha no topo	remove do topo	consulta o topo	verifica se vazia

Implementação - Pilhas

```python
pilha = []
pilha.append(10)
pilha.append(20)
pilha.append(30)
print(pilha)
```

append() empilha elementos no topo da pilha.

Removendo - Pilhas

```python
pilha.pop()
print(pilha)
```

pop() remove e retorna o elemento do topo (LIFO).

Aplicações - Pilhas

- Desfazer (Undo)
- Navegador Web
- Avaliação de expressões
- Chamadas de funções
- Recursividade

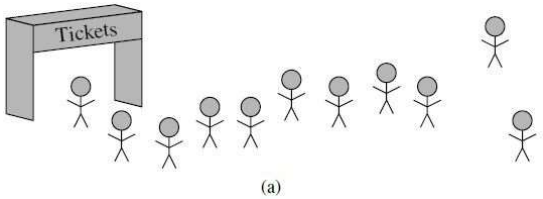
FIFO

Fila – First In, First Out

O primeiro elemento a entrar é o primeiro a sair, como uma fila de atendimento.

Filas

- Política FIFO (First In First Out)
  - Primeiro nó a entrar é o primeiro a sair
- Possui dois métodos
  - Enqueue - Insere um elemento no final da fila
  - Pop - Remove o primeiro elemento da fila
- Para implementar deve-se manter uma referência para o fim da estrutura e uma para o início



(a)

Ilustração - Filas

Entrada →

← Saída

Enfileira no fim, atende pela frente.

Operações - Filas

Enqueue	Dequeue	Front	Rear
insere no fim	remove da frente	consulta a frente	consulta o fim

Implementação - Filas

```python
from collections import deque
fila = deque()
fila.append(10)
fila.append(20)
fila.append(30)
print(fila)
```

deque oferece uma fila eficiente nas duas extremidades.

Removendo - Filas

```python
fila.popleft()
print(fila)
```

popleft() remove o primeiro elemento (FIFO).

Aplicações - Filas

- Impressoras
- Atendimento bancário
- Sistemas Operacionais
- Escalonamento
- Streaming

Comparação - Vetor × Lista Encadeada

Vetor	Lista
Memória contínua	Memória distribuída
Acesso rápido	Acesso sequencial
Inserção lenta	Inserção rápida

Comparação - Pilha × Fila

Pilha	Fila
LIFO	FIFO
Remove o último	Remove o primeiro
Push / Pop	Enqueue / Dequeue

Comparação - Complexidade

Operação	Vetor	Lista	Pilha	Fila
Acesso	O(1)	O(n)	O(1) topo	O(1) extrem.
Busca	O(n)	O(n)	O(n)	O(n)
Inserção	O(n)	O(1)*	O(1)	O(1)
Remoção	O(n)	O(1)*	O(1)	O(1)

** no início ou com referência ao nó*

Exercícios

1) Implementar uma pilha em Python (com seus 2 métodos);

2) Implementar um fila em Python (com seus 2 métodos);

3) Implementar um deque em Python. Um deque é uma estrutura que se pode inserir e excluir elementos nas duas extremidades;

4) Implemente uma função que inverta uma lista, empilhando seus elementos e depois escrevendo-os de volta na lista na ordem inversa.

5) Implemente um método recursivo que remova todos os elementos de uma pilha. (Utilize a pilha implementada no exercício 1)

6) Quais valores são retornados durante a seguinte sequência de operações de pilha, executadas sobre uma pilha inicialmente vazia?

push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop()

7) Repita o exercício anterior para o caso de uma fila;

8) Em certas aplicações utilizando filas é comum fazer:
Q.enqueue (Q.dequeue())

Modifique sua implementação de fila para incluir um métodos rotate() com o mesmo efeito. Exemplo:
- Lista Original: [10, 20, 30, 40]
- Lista após rotate: [20, 30, 40, 10]

Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.com.br

(YOUTUBE) ProfEdsonPedroFerlin