



Pilhas

Objetivos

1. Entender o tipo abstrato de dados pilhas
2. Ser capaz de implementar o TAD pilhas usando listas em python
3. Compreender o desempenho da implementação
4. Entender como aplicar o TAD para resolver problemas.

Conteúdo

1. O que são Pilhas?
2. TAD Pilhas
3. Implementação do TAD Pilhas
4. Análise de desempenho
5. Aplicações
6. Exercícios

Pilhas

- Uma Pilha (stack) é uma coleção de objetos que são inseridos e removidos segundo o princípio **last-in, first-out (LIFO)**, último a entrar, primeiro a sair.
- Um usuário pode inserir objetos numa pilha a qualquer momento (se ela tiver espaço), mas somente pode remover o último objeto inserido (o topo da pilha).
- O nome Pilha (stack) é derivada da metáfora e pilha de pratos.

Onde as Pilhas são utilizadas?

- Endereços visitados recentemente em um Browser
- Inverter dados (de uma String ou coleção)
- Realizar match de parênteses ou tags html
- Converter número para binário
- Chamadas de rotinas em um Sistema Operacional
- Tratar expressões infixas, prefixas e pós-fixas

Tipo de Dado Abstrato Pilha

- Pilhas estão entre as estruturas de dados mais importantes.
- Uma instância do tipo Pilha suporta as seguintes funções/métodos:

- `p.push(e)`: adiciona um elemento `e` na pilha `p`
- `p.pop()`: remove e retorna o elemento no topo da pilha `p`
- Outras operações úteis:
 - `s.top()`: retorna uma referência ao topo da pilha, sem removê-lo.
 - `p.is_empty()`: retorna `True` se a pilha está vazia e `False` caso contrário.
 - `len(p)` ou `p.size()`: retorna o número de elementos na pilha.

Exemplo de série de operações com pilha

.Operação.	.Valor Retornado.	.Conteúdo da Pilha.
<code>p.push(5)</code>	–	[5]
<code>p.push(3)</code>	–	[5, 3]
<code>len(p)</code>	2	[5, 3]
<code>p.pop()</code>	3	[5]
<code>p.is_empty()</code>	False	[5]
<code>p.pop()</code>	5	[]
<code>p.is_empty()</code>	True	[]
<code>p.pop()</code>	“erro”	[]
<code>p.push(7)</code>	–	[7]
<code>p.push(9)</code>	–	[7, 9]
<code>p.top()</code>	9	[7, 9]
<code>p.push(4)</code>	–	[7, 9, 4]
<code>len(p)</code>	3	[7, 9, 4]
<code>p.pop()</code>	4	[7, 9]

]

Implementação baseada em Array

- É possível implementar uma pilha armazenando seus elementos em uma lista



Implementação de uma Pilha usando lists (python)

1ª pergunta: como se deseja utilizar a pilha?

```
p = PilhaArray()    # conteúdo [ ]
p.push(5)           # conteúdo [5]
p.push(3)           # conteúdo [5, 3]
print(len(p))       # conteúdo [5, 3]; retorna 2
print(p.pop())      # conteúdo [5]; retorna 3
print(p.is_empty()) # conteúdo [5]; retorna False
print(p.pop())      # conteúdo [ ]; retorna 5
print(p.is_empty()) # conteúdo [ ]; retorna True
p.push(7)           # conteúdo [7]
p.push(9)           # conteúdo [7, 9]
print(p.top())      # conteúdo [7, 9]; retorna 9
p.push(4)           # conteúdo [7, 9, 4]
print(p.size())     # conteúdo [7, 9, 4]; retorna 3
print(p.pop())      # conteúdo [7, 9]; retorna 4
p.push(6)           # conteúdo [7, 9, 6]
p.push(8)           # conteúdo [7, 9, 6, 8]
print(p.pop())      # conteúdo [7, 9, 6]; retorna 8
```

Classe Pilha

```
class PilhaArray:

    def __init__(self):
        self._pilha = []

    def __len__(self):
        return len(self._pilha)

    def size(self):
        return self.__len__()

    def is_empty(self):
        return len(self._pilha) == 0

    def push(self, e):
        self._pilha.append(e)

    def top(self):
        if self.is_empty():
            raise PilhaVazia('A pilha está vazia')
        return self._pilha[-1]

    def pop(self):
        if self.is_empty():
            raise PilhaVazia('A pilha está vazia')
        return self._pilha.pop()
```

Exceção

Pilha Vazia

```
class PilhaVazia(Exception):  
    pass
```

Lançando exceção

```
def pop(self):  
    if self.is_empty():  
        raise PilhaVazia('A pilha está vazia')  
    return self._pilha.pop()
```

Análise da implementação da Pilha

Operação	Tempo de Execução
S.push(e)	O(1)*
S.pop()	O(1)*
S.top()	O(1)
S.is empty()	O(1)
len(S)	O(1)

*amortizado

Invertendo dados

- Como consequência do protocolo LIFO, uma pilha pode ser utilizada para inverter dados

Invertendo uma String

```
def inverte_texto(texto):  
    pilha = PilhaArray()  
    for letra in texto:  
        pilha.push(letra)  
    palavra = ""  
    while not pilha.is_empty():  
        palavra += pilha.pop()  
    return palavra
```

Invertendo um arquivo

```
def inverte_arquivo(nome_arquivo):
    p = PilhaArray()
    arquivo = open(nome_arquivo)    # abre o arquivo

    for linha in arquivo:            # lê cada linha do arquivo
        p.push(linha.rstrip("\n"))  # insere a linha na pilha
    arquivo.close()                 # fecha o arquivo

    output = open(nome_arquivo, 'w') # reabre o arquivo e sobrescreve
    while not p.is_empty():
        output.write(p.pop() + '\n') # re-insert newline characters
    output.close()
```

Matching Parênteses ou Tags HTML

- Quais dessas expressões estão corretas? 1. `((()){([()])}` 2. `((()(){([()])}))` 3. `)(()){([()])}` 4. `{[]}` 5. `(`
- Como seria o algoritmo para verificar se os delimitadores de uma expressão estão corretos ou não?

Algoritmo para verificar delimitadores

```
def is_matched(expr):
    abre = '(['      # delimitadores - abertura
    fecha = ')]'     # delimitadores - fechamento
    pilha = PilhaArray()
    for d in expr:
        if d in abre:
            pilha.push(d)    # push o delimitador de abertura
        elif d in fecha:
            if pilha.is_empty():
                return False    # não bate
            if fecha.index(d) != abre.index(pilha.pop()):
                return False    # não bate
    return pilha.is_empty()    # tem mais delimitadores na pilha?
```

Código fonte

Disponível em <https://replit.com/@RicardoRubens/pilhas-21>

Para estudar

- Seções 3.1 a 3.9 do livro [5] https://panda.ime.usp.br/pythonds/static/pythonds_pt/03-EDBasicos/toctree.html
- Módulo 2, Seção 2.2 do livro [4] <http://educapes.capes.gov.br/handle/capes/176522>
- Capítulo 18 (Pilhas) do livro [2] (texto bastante resumido!) <https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf>

Referências

- Tradução do livro *Howto Think Like a Computer Scientist: Interactive Version*, de Brad Miller e David Ranum. link: <https://panda.ime.usp.br/pensepy/static/pensepy/index.html>
- Allen Downey, Jeff Elkner and Chris Meyers. *Aprenda Computação com Python 3.0*. link: <https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf>
- SANTOS, A. C. *Algoritmo e Estrutura de Dados I*. 2014. Disponível em

<http://educapes.capes.gov.br/handle/capes/176522>

4. SANTOS, A. C. *Algoritmo e Estrutura de Dados II*. 2014. Disponível em 2014. Disponível em <https://educapes.capes.gov.br/handle/capes/176557>
5. Tradução do livro [5] *Problem Solving with Algorithms and Data Structures using Python* de Brad Miller and David Ranum. link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
6. Brad Miller and David Ranum. *Problem Solving with Algorithms and Data Structures using Python* link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
7. Caelum. Algoritmos e Estruturas Dados em Java. Disponível em <https://www.caelum.com.br/download/caelum-algoritmos-estruturas-dados-java-cs14.pdf>

That's all Folks
