Estrutura de Dados

Aula 05

Prof. Luiz Antonio Schalata Pacheco, Dr. Eng.

Instituto Federal de Santa Catarina
Câmpus Garopaba
Curso Superior de Tecnologia em Sistemas para Internet

schalata@ifsc.edu.br

23/03/2022

Pilhas

Conceito

- A ideia parte do ”empilhamento”de um elemento sobre o outro

Conceito

- Os dados são empilhados e depois desempilhados
- Se quiser tirar um prato ou moeda do meio tem que tirar antes os que estão acima
- Só se tem acesso ao último elemento que foi empilhado
- Se o último for removido, o item anterior, inserido antes dele, pode ser acessado
- LIFO - *Last In First Out* - Último a entrar, primeiro a sair
- Essa é a ideia básica dessa estrutura de dados

Aplicações

- Verificação da sintaxe de expressões aritméticas, tais com $3 * (4 + 5)$
- Percorrimento de uma árvore binária
- Pesquisa do vértice de um grafo
- Microprocessadores com arquitetura baseada em pilhas. Quando um método é chamado, seu endereço de retorno e seus parâmetros são empilhados em uma pilha e quando ele retorna, são desempilhados
- Sistemas Operacionais utilizam muito o conceito de pilha

Operações

- Colocar um item no topo da pilha: empilhar
- Remover um item do topo da pilha: desempilhar
- Não pode ser removido um elemento do meio ou da base da pilha
- Mostrar o elemento do topo
- https://www.cs.usfca.edu/~galles/visualization/StackArray.html

Implementação

- Implementar a classe pilha com os seguintes métodos de empilhar, desempilhar e ver topo.
- Além disso faça duas funções adicionais para verificar se a pilha está cheia e se a pilha está vazia.

Classe Pilha

```
1  class Pilha:
2
3    def __init__(self, capacidade):
4      self.__capacidade = capacidade
5      self.__topo = -1
6      self.__valores = np.empty(self.__capacidade, dtype=
       str)
7
8    def __pilha_cheia(self):
9      if self.__topo == self.__capacidade - 1:
10       return True
11     return False
12
13   def pilha_vazia(self):
14     if self.__topo == -1:
15       return True
16     return False
```


Classe Pilha - continuação

```
18    def empilhar(self, valor):
19      if self.__pilha_cheia():
20        print('A pilha esta cheia')
21      else:
22        self.__topo += 1
23        self.__valores[self.__topo] = valor
24
25    def desempilhar(self):
26      if self.pilha_vazia():
27        print('A pilha esta vazia')
28        return -1
29      else:
30        valor = self.__valores[self.__topo]
31        self.__topo -= 1
32        return valor
```


Classe Pilha - continuação

```
34    def ver_topo(self):
35      if self.__topo != -1:
36        return self.__valores[self.__topo]
37      else:
38        return -1
```


Exercício: Validador de expressão aritmética

- Os delimitadores são { }, [ ], ( )
- Cada delimitador de abertura deve ser casado com um delimitador de fechamento
- Exemplo: Toda { deve ser seguida por uma } que case com ela
  - c[d] (correto)
  - ab[c]de (correto)
  - ab(c]de (incorreto), pois] não casa com (

Exercício: Validador de expressão aritmética

Caractere lido	Conteúdo da pilha
a	
{	{
b	{
(	{(
c	{(
[	{([
d	{([
]	{(
e	{(
)	{
f	{
}	

Exercício: Validador de expressão aritmética

```
1     expressao = str(input('Digite uma expressao: '))
2     pilha = Pilha(len(expressao))
3
4     for i in range(len(expressao)):
5         ch = expressao[i]
6         if ch == '{' or ch == '[' or ch == '(':
7             pilha.empilhar(ch)
8         elif ch == '}' or ch == ']' or ch == ')':
9             if not pilha.pilha_vazia():
10                chx = str(pilha.desempilhar())
11                if (ch == '}' and chx != '{') or (ch == '
    ]' and chx != '[') or (ch == ')' and chx != '('):
12                    print('Erro: ', ch, ' na posicao
13            else:
14                print('Erro: ', ch, ' na posicao ', i)
15    if not pilha.pilha_vazia():
16        print('Erro!')
```