

SME0827 - Estruturas de Dados

Pilhas e Filas em arrays Aula 10



Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Moisés Rocha dos Santos
Monitor: Marília da Silva

Implementação em Python

- Implementação de TADs baseadas em listas
- Mais comuns:
 - Pilhas e filas
- Duas opções de implementação:
 - Arrays
 - Listas encadeadas
 - Simplesmente encadeadas
 - Duplamente encadeadas

© André de Carvalho - ICMC/USP

4

Tópicos

- Design patterns
- TAD Pilha (Stack)
- Implementação de pilhas em arrays
- TAD Fila (queue)
- Implementação de filas em arrays
- Deque

© André de Carvalho - ICMC/USP

2

Implementação por arrays

- A implementação do array é simples e eficiente
 - Executa geralmente em $O(1)$
 - O tamanho da pilha / fila tem um limite superior, N
- Valor de N é arbitrário, e pode se:
 - Pequeno, pilha / fila ter tamanho insuficiente
 - Grande, levar a um desperdício de memória

© André de Carvalho - ICMC/USP

5

Design patterns

- Alguns códigos no livro estão em uma linguagem abstrata
 - Usa padrões de projeto (design patterns) orientados a objetos
 - Código padrão para problemas que ocorrem frequentemente em projetos de software
 - Permite organizar os códigos em components reutilizáveis
 - Códigos são facilmente traduzidos para uma linguagem de programação
 - Ex.: Python

© André de Carvalho - ICMC/USP

3

TAD Pilha (Stack)

- Uma dos TADs mais comuns, pode ser usada para:
 - Definir sequência em que carros são retirados de estacionamento
 - Decidir que documento vai assinar primeiro
 - Implementação de funções recursivas
 - Cálculo de uma expressão matemática
 - Ler artigos e livros para estudar estruturas de dados




© André de Carvalho - ICMC/USP

6

TAD Pilha (Stack)

- A inserção e remoção de itens seguem o princípio de que o **último** a entrar é o **primeiro** a sair
 - Last-in-first-out** (LIFO)
 - Os itens são inseridos no **final** da pilha e removidos a partir do **final** da pilha



© André de Carvalho - ICMC/USP 7

Exemplo TAD Pilha

```


>>> pilha = [3, 4, 5]
>>> pilha.push(6)
>>> pilha.push(7)
>>> pilha

>>> pilha.pop()

>>> pilha

>>> pilha.pop()
>>> pilha.pop()
>>> pilha.isEmpty()
>>> pilha

```



© André de Carvalho - ICMC/USP 10

TAD Pilha (design pattern)

- Construtor:
 - make():Stack** # cria uma pilha
- Funções de acesso:
 - top(S:Stack):item** # retorna item no topo da pilha S
 - size(S:Stack):integer** # retorna o tamanho da pilha
 - isEmpty(S:Stack):boolean** # retorna se a pilha está vazia
- Funções de manipulação:
 - push(S:Stack, o:element):Stack** # Insere item no topo da pilha
 - pop(S:Stack):Stack** # Remove item no topo da pilha

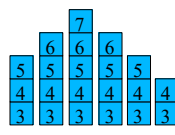
© André de Carvalho - ICMC/USP 8

Exemplo TAD Pilha

```

>>> pilha = [3, 4, 5]
>>> pilha.append(6)
>>> pilha.append(7)
>>> pilha
[3, 4, 5, 6, 7]
>>> pilha.pop()
7
>>> pilha
[3, 4, 5, 6]
>>> pilha.pop()
6
>>> pilha.pop()
5
>>> pilha.isEmpty()
False
>>> pilha
[3, 4]

```



© André de Carvalho - ICMC/USP 11

Pseudo código

```

# Supor variável t, índice do
# item no topo da pilha

Algorithm size()
return t+1

Algorithm isEmpty()
return (t<0)

Algorithm top()
if isEmpty() then
return Error
return S[t]

# Supor variável N, o tamanho
# topo da pilha

Algorithm push(x)
if size()==N then
return Error
t=t+1
S[t]=x


Algorithm pop()
if isEmpty() then
return Error
S[t]=null
t=t-1

```

© André de Carvalho - ICMC/USP 9

Implementação por arrays

- Uma forma simples de usar pilhas é por meio um array (vetor) de tamanho N
 - Pilha é representada por um vetor S com N itens
 - Uma variável inteira t indica o item no topo da pilha



- Os índices do vetor começam em 0
 - Para pilha vazia, t = -1

© André de Carvalho - ICMC/USP 12

Pilha em Python por Arrays

```
# Cria uma pilha e inicializa seu tamanho como 0
def createStack():
    stack = []
    return stack

# Verifica se pilha está vazia
def isEmpty(stack):
    return len(stack) == 0

# Adiciona um item à pilha e incrementa seu tamanho
def push(stack, item):
    stack.append(item)
    print("Item " + item + " foi colocado na pilha")

# Remove um item da pilha e decrementa seu tamanho
def pop(stack):
    if not isEmpty(stack):
        return stack.pop()

# Retorna, sem remover, o item no topo da pilha
def peek(stack):
    if not isEmpty(stack):
        return stack[len(stack) - 1]

# Função para imprimir itens da pilha
def printPilha(stack):
    print("Os itens da pilha, a partir do topo, são:")
    temp = -1
    while (not isEmpty(stack)):
        print(stack[temp], end=">")
        temp = temp + 1
    pop(stack)
```

© André de Carvalho - ICMC/USP 13

Exercício

- Em uma equação matemática, cada parêntese aberto deve ser fechado
- Escrever código em Python usando a TAD pilha que
 - Recebe uma equação com um número qualquer de parênteses
 - Retorna se todo parêntese aberto foi fechado (ou todo fechado foi aberto) e quantas vezes isso ocorreu

© André de Carvalho - ICMC/USP 16

Exemplo

```
stack = createStack()
push(stack, str(10))
push(stack, str(20))
push(stack, str(30))
print("Item " + pop(stack) + " foi retirado da pilha")
printPilha(stack)
```

```
10 foi colocado na pilha
20 foi colocado na pilha
30 foi colocado na pilha
Item 30 foi retirado da stack
Os itens da pilha, a partir do topo, são:
20->10->
```

© André de Carvalho - ICMC/USP 14

Pausa



© André de Carvalho - ICMC/USP 17



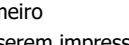
Implementação em Python

TAD	Python	Complexidade
S.push(x)	lista.append(x)	O(1)*
S.pop()	lista.pop()	O(1)*
S.top()	lista[-1]	O(1)
S is empty()	len(lista) == 0	O(1)
len (s)	len (lista)	O(1)

* O(n) se usando array dinâmico, pois a lista precisa redefinir o tamanho de seu array interno

© André de Carvalho - ICMC/USP 15


TAD Fila (Queue)

- Outra TAD muito comum, pode ser usada para:
 - Quem pega o taxi no aeroporto 
 - Quando recebe restituição do IR 
 - Quem vai entrar em um dado curso após prestar o ENEM 
 - Que avião decola ou aterrissa primeiro
 - Indicar sequência de arquivos a serem impressos em uma impressora

© André de Carvalho - ICMC/USP 18

TAD Fila (Queue)

- A inserção e remoção de itens seguem o princípio de que o **primeiro** a entrar é o **primeiro** a sair
 - First-in-first-out (FIFO)**
 - Os itens são inseridos no **final** da fila e removidos a partir do **início** da fila



© André de Carvalho - ICMC/USP 19

Exemplo TAD Fila

```
>>> fila = ["Pedro", "Joao", "Jose"]
>>> fila.append("Luiz") # Luiz chegou
>>> fila.append("Mario") # Mario chegou
>>> fila.pop(0)

>>> fila.pop(0)

>>> fila
```

© André de Carvalho - ICMC/USP 22

TAD Fila (Queue)

- Construtor:
 - make()**: *Queue* # cria uma fila vazia
- Funções de acesso:
 - size**(*S:Queue*): *integer* # retorna tamanho da fila *S*
 - isEmpty**(*S:Queue*): *Boolean* # retorna se a fila está vazia
 - front**(*S:Queue*): *element* # retorna item na frente da fila
- Funções de manipulação:
 - Enqueue**(*S:Queue, o:element:Queue* # insere item no final da fila
 - Dequeue**(*S:Queue*): *Queue* # remove item da frente da fila

© André de Carvalho - ICMC/USP 20

Exemplo TAD Fila

```
>>> fila = ["Pedro", "Joao", "Jose"]
>>> fila.append("Luiz") # Luiz chegou
>>> fila.append("Mario") # Mario chegou
>>> fila.pop(0)
'Pedro'
>>> fila.pop(0)
'Joao'
>>> fila
['Jose', 'Luiz', 'Mario']
```

© André de Carvalho - ICMC/USP 23

Pseudo código

```
Algorithm size()
return N-f+r

Algorithm isEmpty()
return size()=0

Algorithm front()
if isEmpty() then
return Error
return Q[f]

Algorithm dequeue()
if isEmpty() then
return Error
Q[f]=null
f = f+1

Algorithm enqueue(o)
if size = N - 1 then
return Error
Q[r]=o
r = r+1
```

© André de Carvalho - ICMC/USP 21

Observação

```
L = [1,2,3]
for i in range(0,len(L)):
    print (i) # imprime o indice i da lista L

L = [1,2,3]
for i in range(0,len(L)):
    print (L[i]) # imprime o item de indice i da lista L
```

© André de Carvalho - ICMC/USP 24

Implementação por vetores

- Também é simples de criar uma fila por meio um vetor de tamanho N
 - Representa fila por um vetor S com N itens
 - Usar uma variável inteira, f, para indicar o final da fila
- A fila começa no item de índice 0
 - Para fila vazia, f = -1

013...fN-2N-1

© André de Carvalho - ICMC/USP

25

Exemplo

```
queue1 = createQueue()
enqueue(queue1, str(10))
enqueue(queue1, str(20))
enqueue(queue1, str(30))
print("Item " + dequeue(queue1) + " foi retirado da fila")
printfila(queue)
```

Item 10 foi colocado na fila
Item 20 foi colocado na fila
Item 30 foi colocado na fila
Item 10 foi retirado da fila
Os itens da fila, a partir do início, são:
...

© André de Carvalho - ICMC/USP

28

Fila em Python por Arrays

```
## Cria uma fila e inicializa seu tamanho como 0
def createQueue():
    queue = []
    return queue

# Verifica se fila está vazia
def isEmpty(queue):
    return len(queue) == 0

# Adiciona um item à fila e incrementa seu tamanho
def enqueue(queue, item):
    queue.append(item)
    print("Item "+ item + " foi colocado na fila")

# Remove um item da fila e decrementa seu tamanho
def dequeue(queue):
    if not(isEmpty(queue)):
        return queue.pop(1)

# Retorna, sem remover, o item na frente da fila
def peek(stack):
    if not(isEmpty(stack)):
        return stack[len(stack) - 1]

# Função para imprimir itens da fila
Exercício
```

© André de Carvalho - ICMC/USP

26

Implementação em Python

TAD	Python	Complexidade
S.enqueue(x)	lista.append(x)	O(1)*
S.dequeue()	lista.pop(0)	O(1)*
S.first()	lista[0]	O(1)
S is empty()	len(lista) == 0	O(1)
len (s)	len (lista)	O(1)

* O(n) caso repetidamente sejam retirados e incluídos na lista

© André de Carvalho - ICMC/USP

29

Exercício

- Escrever e testar um método de impressão para imprimir o conteúdo de uma fila
 - Se necessário, pode modificar um pouco as demais funções do slide anterior

© André de Carvalho - ICMC/USP

27

Ineficiência 1

- Remover item do início da fila da fila é computacionalmente ineficiente
 - Operação lista.pop[0]
- Quando o item de índice 0 é retirado, itens remanescente são movidos para a esquerda

0123...N-3N-2N-1

© André de Carvalho - ICMC/USP

30

Ineficiência 1

- Remover item do início da fila da fila é computacionalmente ineficiente
 - Operação `lista.pop[0]`

0

1

2

3

...

N-3

N-2

N-1

0

1

2

3

...

N-3

N-2

N-1
 - Quando o item de índice 0 é retirado, itens remanescente são movidos para a esquerda
 - Alternativa: criar um índice `i`, para o início da fila

0

i

2

3

...

N-3

N-2

N-1

0

1

2

3

...

N-3

N-2

N-1

© André de Carvalho - ICMC/USP31

TAD Deque

- Fila com entrada e saída (direção) dupla
 - Double-ended queue
 - Permite inserção e remoção no início e no final
 - Mais geral que fila e pilha
 - Possui várias aplicações
 - Ex.: Uma pessoa no final da fila de um hospital pode começar a passar mal
 - Deve ser atendida antes de quem está no início da fila
 - Em geral implementada em array dinâmico
 - Pode ser implementada por um array circular

© André de Carvalho - ICMC/USP34

Ineficiência 2

- Esta implementação de fila pode ainda ser ineficiente
 - A medida que itens saem da fila, e novos itens entram, possível tamanho da fila diminui

© André de Carvalho - ICMC/USP32

TAD Deque

- O TAD deque possui seis métodos (funções) de acesso e manipulação
 - `insertFirst(S:Deque, x:item):Deque`
 - `insertLast(S:Deque, x:item):Deque`
 - `removeFirst(S:Deque):Deque`
 - `removeLast(S:Deque):Deque`
 - `first(S:Deque):item`
 - `last(S:Deque):item`

© André de Carvalho - ICMC/USP35

Ineficiência

- Esta implementação de fila pode ser ineficiente
 - A medida que itens saem da fila, e novos itens entram, possível tamanho da fila diminui
 - Alternativa: Vetor circular
 - Após final do vetor, índice recomeça no início do vetor

$N-f+r \bmod N$

$f \leftarrow (f+1) \bmod N$

$r \leftarrow (r+1) \bmod N$

f

1

2

i

...

N-3

N-2

N-1

f

1

2

i

...

N-3

N-2

N-1

© André de Carvalho - ICMC/USP33

Implementação em Python

TAD	Python	Complexidade
<code>S.add_first(x)</code>	<code>lista.appendleft(x)</code>	$O(1)$
<code>S.add_last(x)</code>	<code>lista.append(x)</code>	$O(1)$
<code>S.delete_first()</code>	<code>lista.popleft()</code>	$O(1)$
<code>S.delete_last()</code>	<code>lista.pop()</code>	$O(1)$
<code>S.first</code>	<code>lista[0]</code>	$O(1)$
<code>S.last</code>	<code>lista[-1]</code>	$O(1)$
<code>len(s)</code>	<code>len(lista)</code>	$O(1)$

© André de Carvalho - ICMC/USP36

Exercício

- Escrever e testar código Python para implementar as funções da estrutura dequeue
 - Além de uma função de impressão


© André de Carvalho - ICMC/USP 37

Arrays dinâmicos

- Python usa arrays dinâmicos
 - Tamanho de uma lista pode crescer sem limite ao longo da execução de um código
 - Para isso, uma instância da lista em geral mantém um array auxiliar com capacidade maior que a lista
 - Usa operações **append** para aumentar tamanho
 - Uma análise (análise de amortização) mostra que arrays dinâmicos são eficientes
 - Mas custo se torna alto para sistemas de tempo real
 - Operações de inserção e remoção em posições internas de um array tem custo elevado

© André de Carvalho - ICMC/USP 38

Questions



© André de Carvalho - ICMC/USP 39