"Não existe almoço grátis!"

Removendo Recursão

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Removendo recursão

Quando a recursão não é de cauda (e não conseguimos transformá-la em uma recursão de cauda)

Precisamos salvar as informações em uma pilha para remover a recursão

Removendo recursão

Quando a recursão não é de cauda (e não conseguimos transformá-la em uma recursão de cauda)

Precisamos salvar as informações em uma pilha para remover a recursão

Fazer o papel do compilador de salvar o “Frame” na memória

Ideia Geral

Geralmente, o que se faz é

- Substituir chamadas recursivas por “empilhar”
- Retornos se tornam “desempilhar”
- Realizar o processamento em um loop enquanto a pilha não estiver vazia

Exemplo

```
função quickSort(v,a,b)
se a ≥ b
      retorne
m ← particionar(v,a,b)
quickSort(v,a,m-1)
quickSort(v,m+1,b)
retorne
```


Obs.: o parâmetro *p* é uma pilha

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

Teste de mesa

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	42	15	23	8	4	16

Teste de mesa

quickSort		
a	b	m
1	6	

Pilha p


```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

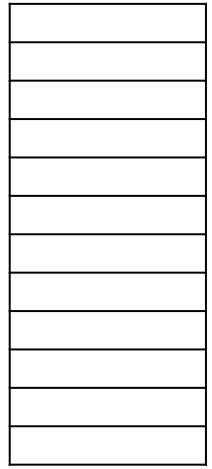
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	42	15	23	8	4	16

Teste de mesa

quickSort		
a	b	m
1	6	

Pilha p



Topo

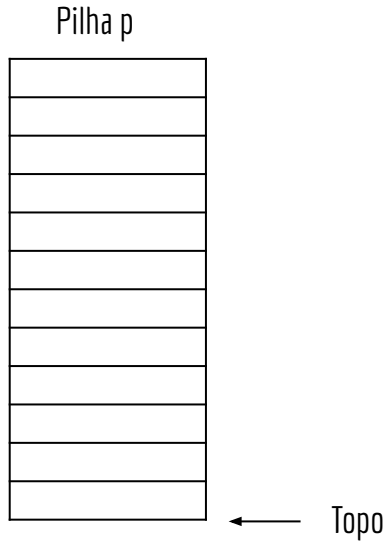
```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	42	15	23	8	4	16

Teste de mesa

quickSort		
a	b	m
1	6	4



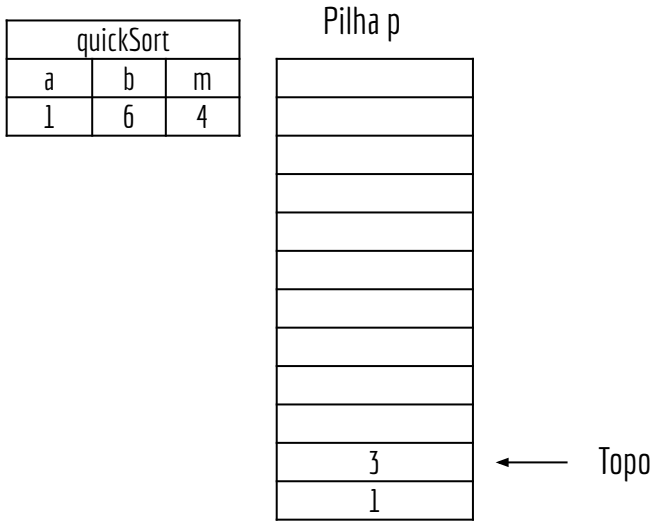

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
1	6	4



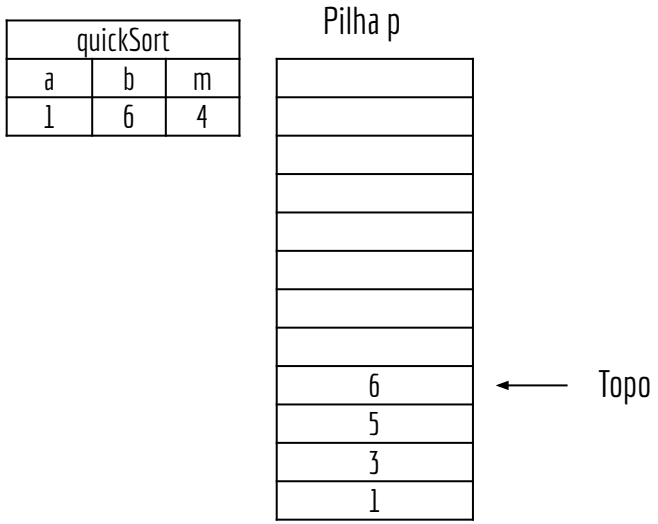

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
1	6	4




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

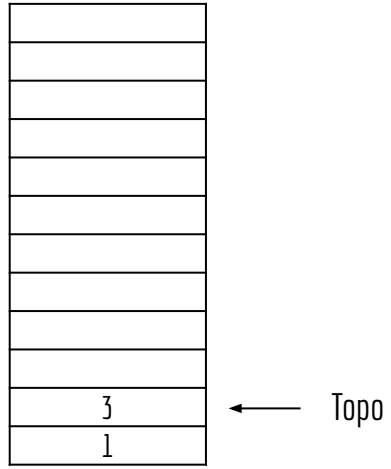
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
5	6	4

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

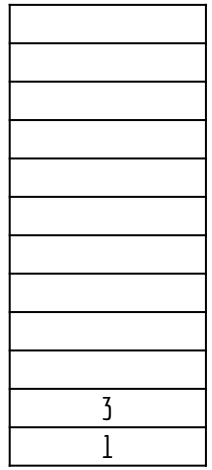
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
5	6	6

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

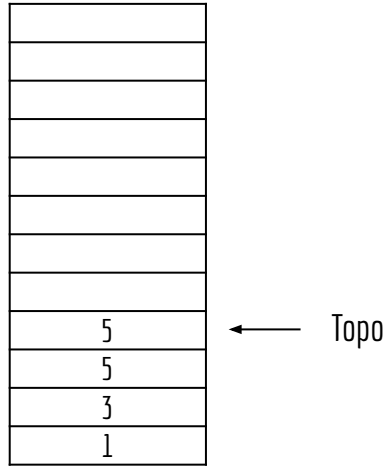
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
5	6	6

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
5	6	6

Pilha p

6
7
5
5
3
1

← Topo

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
7	6	6

Pilha p

5
5
3
1

← Topo

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

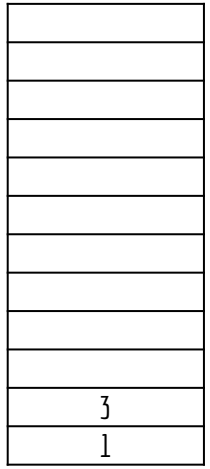
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
5	5	6

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
1	3	6

Pilha p

Topo

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

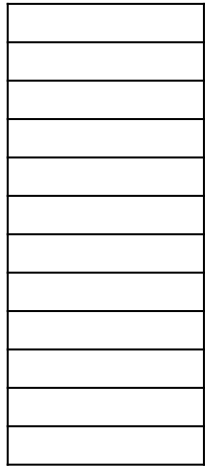
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	15	8	4	16	23	42

Teste de mesa

quickSort		
a	b	m
1	3	1

Pilha p



Topo

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

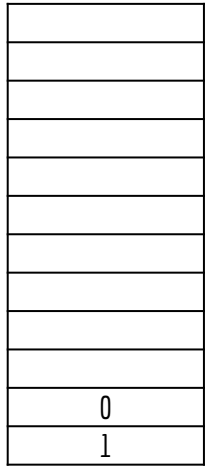
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
1	3	1

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

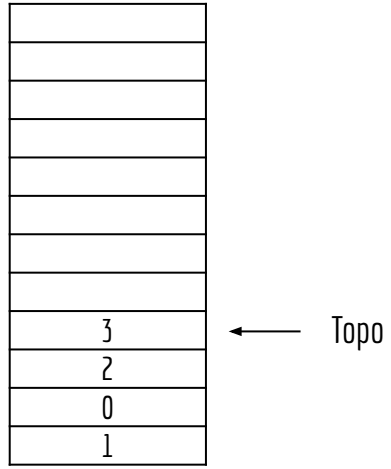
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
1	3	1

Pilha p



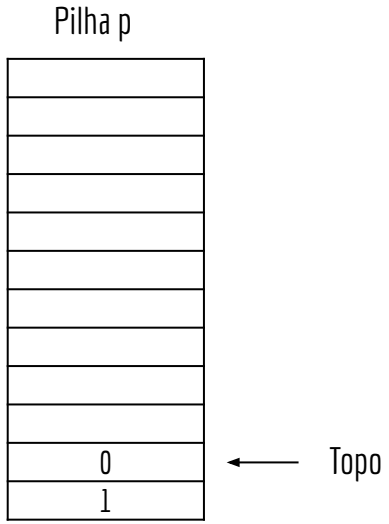

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
2	3	1




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

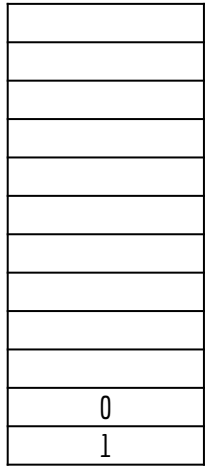
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
2	3	3

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

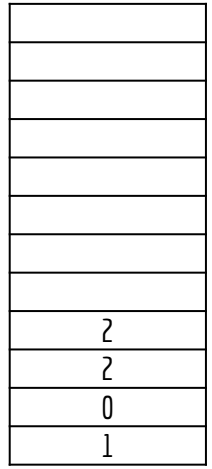
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
2	3	3

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
2	3	3

Pilha p

3 ← Topo
4
2
2
0
1

```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

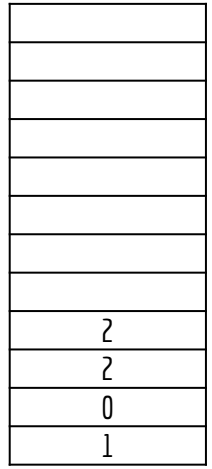
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
4	3	3

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

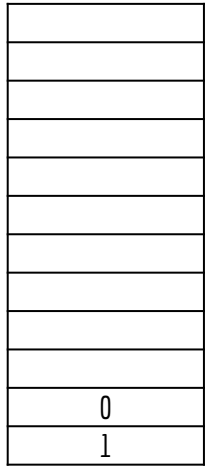
```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
2	2	3

Pilha p




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
1	0	3

Pilha p

Topo

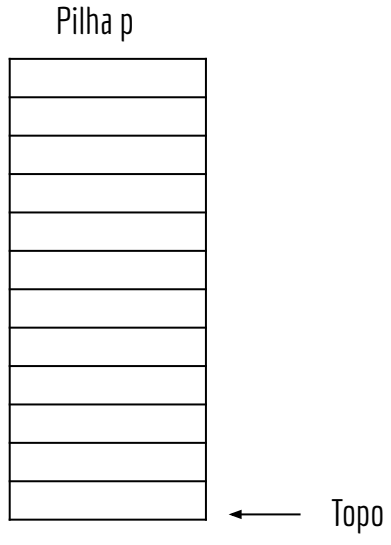
```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Teste de mesa

quickSort		
a	b	m
1	0	3




```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

```
função particionar(v,a,b)
x ← v[b] //pivô
m ← a
para i ← a até b-1
      se v[i] ≤ x
            trocar(v,m,i)
            m ← m + 1
trocar(v,m,b)
retorne m
```

i	1	2	3	4	5	6
v[i]	4	8	15	16	23	42

Questão de ordem

A ordem em que o quicksort é executado para os subproblemas é a mesma nas duas versões do algoritmo?

```
função quickSort(v,a,b)
se a ≥ b
      retorne
m ← particionar(v,a,b)
quickSort(v,a,m-1)
quickSort(v,m+1,b)
retorne
```



```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

Questão de ordem

A ordem em que o quicksort é executado para os subproblemas é a mesma nas duas versões do algoritmo?

Não.

Lembre-se que uma pilha é uma LIFO (Last In First Out). Se a ordem era importante no algoritmo recursivo, leve isso em consideração ao empilhar os dados.

```
função quickSort(v,a,b)
se a ≥ b
      retorne
m ← particionar(v,a,b)
quickSort(v,a,m-1)
quickSort(v,m+1,b)
retorne
```

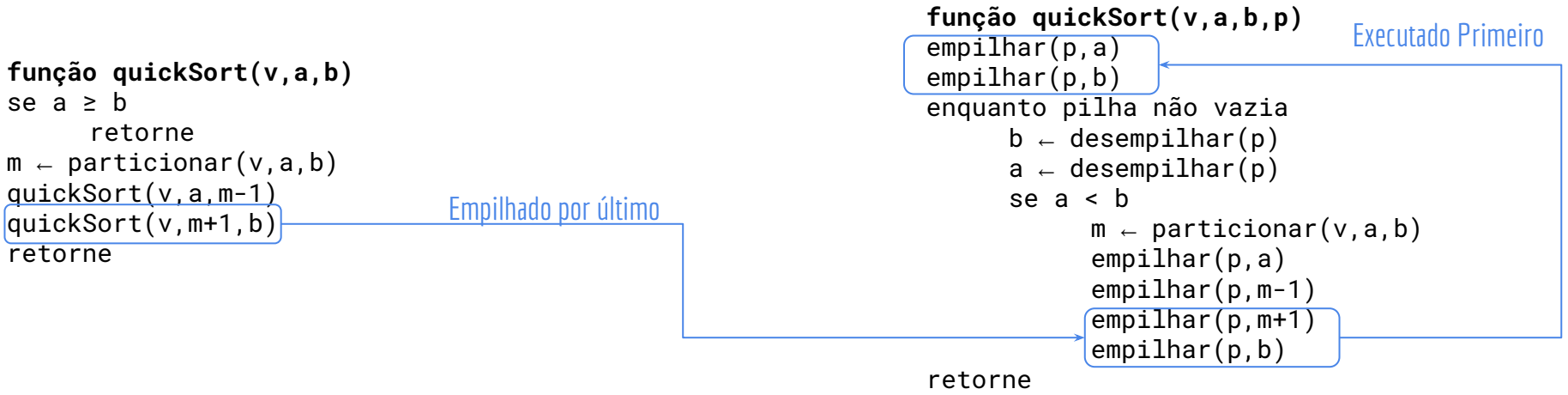


```
função quickSort(v,a,b,p)
empilhar(p,a)
empilhar(p,b)
enquanto pilha não vazia
      b ← desempilhar(p)
      a ← desempilhar(p)
      se a < b
            m ← particionar(v,a,b)
            empilhar(p,a)
            empilhar(p,m-1)
            empilhar(p,m+1)
            empilhar(p,b)
retorne
```

Questão de ordem

A ordem em que o quicksort é executado para os subproblemas é a mesma nas duas versões do algoritmo?

Não.



Removendo recursão

Recursão de cauda geralmente é trivial de se remover

- Geralmente uma boa ideia
  - Redução do custo de memória e processamento

Removendo recursão

Recursão de cauda geralmente é trivial de se remover

- Geralmente uma boa ideia
  - Redução do custo de memória e processamento

Se não é uma recursão de cauda, podemos tentar transformar em uma recursão de cauda

Removendo recursão

Recursão de cauda geralmente é trivial de se remover

- Geralmente uma boa ideia
  - Redução do custo de memória e processamento

Se não é uma recursão de cauda, podemos tentar transformar em uma recursão de cauda

Caso não seja possível transformar em recursão de cauda

- Podemos ainda usar uma pilha para remover a recursão
  - Mas tenha em mente que estamos fazendo algo que o compilador estava fazendo automaticamente

Por que remover

Remover recursão de cauda quase sempre vai resultar em um programa mais eficiente

Remover uma recursão usando pilha também pode ser vantajoso

- A pilha de Frames gerada pelo compilador precisa guardar várias outras informações que custam memória, que a nossa pilha não precisa ter
  - Ponto de execução atual, fim do frame atual, ponto de retorno, ...
- Algumas linguagens de programação ou hardwares podem não possuir suporte para recursão
  - E.g., Microcontroladores

Desvantagem de remover a recursão

- Muitos programas se tornam complicados quando a recursão é removida

Faça você mesmo

Considere a seguinte solução para o problema do labirinto

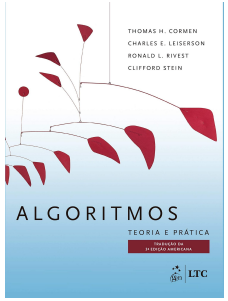
```
função labirinto(L,p,f)
entrada: labirinto L de MxN, a posição atual p e a posição de saída f
saída: caminho da entrada até a saída.

se p = f
      retorne encontrou saída
n ← listarProximosPassosValidos(L,p)
para i de 1 até |n|
      marcarComoPassou(L,n[i])
      se labirinto(L,n[i],f) = encontrou saida
            retorne encontrou saída
      removerPassou(L,n[i])
retorne sem saída
```

Exercícios

1. Implemente os algoritmos da aula em C.
2. Remova a recursão do problema das 8 -rainhas.
3. Encontre outros algoritmos recursivos (e.g., algoritmos dados em aula) e remova a recursão através de pilhas.

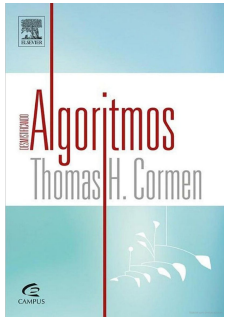
Referências

T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 3a ed. 2012

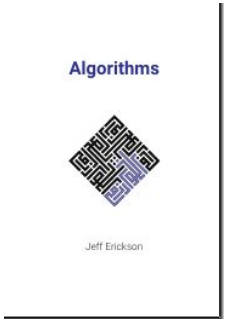


R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2014

T. Cormen. Desmistificando algoritmos. 2017.



J. Erickson. Algorithms. 2019.



Licença