Recursividade - Estrutura de Dados

Emannuel Levi e Henrique Fernandes

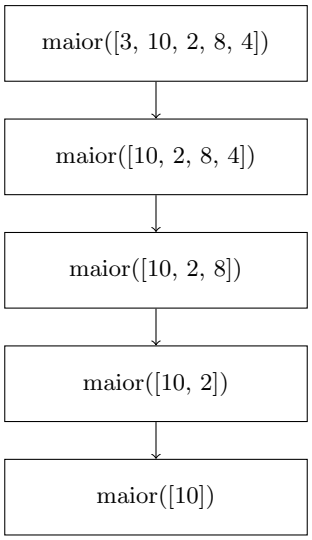
Novembro de 2024

Problema 1. Projete um algoritmo recursivo para determinar o maior elemento em uma sequência de inteiros S com n elementos. Qual é a complexidade do seu algoritmo? Desenhe a árvore de recursão.

Solução. Segue o algoritmo, sua complexidade e a sua árvore de recursão: □

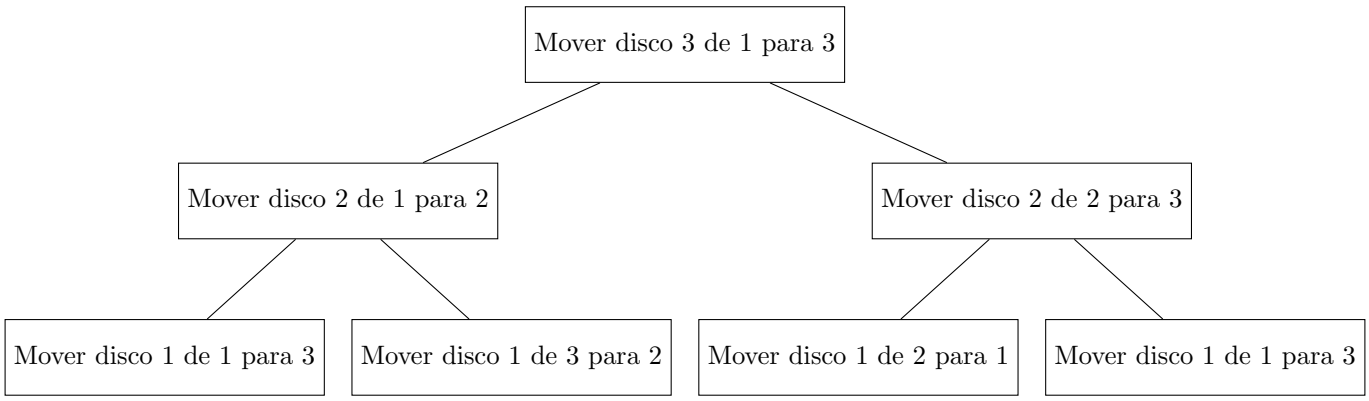
```python
def maior(lista: list) -> int:
    if len(lista) == 1:
        return lista[0]
    if lista[0] > lista[-1]:
        return maior(lista[:-1])
    return maior(lista[1:])
```

O algoritmo tem complexidade O(n) como pode ser observado pela sua árvore de recursão abaixo:



Problema 2. Faça um diagrama das chamadas recursivas de uma função que resolve o problema das torres de hanoi.

Solução. □



Problema 3. Escreva um algoritmo recursivo que organize uma sequência de inteiros de tal forma que os valores pares apareçam antes do que os ímpares.

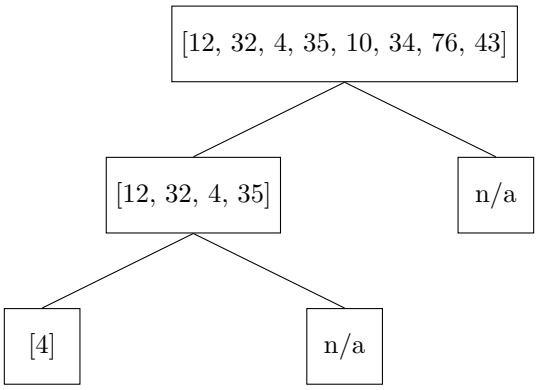
Solução. Segue código python:

```
def sortedOrderEven(lista: list) -> list:
    if len(lista) == 1:
        return lista
    if lista[0] % 2 == 0:
        return lista[0:1] + sortedOrderEven(lista[1:])
    return sortedOrderEven(lista[1:]) + lista[0:1]
```

□

Problema 4. Faça um diagrama das chamadas recursivas da pesquisa binária para a sequência $S = [12, 32, 4, 35, 10, 34, 76, 43]$, quando o valor pesquisado é 4.

Solução. □



Problema 5. Escreva as funções recursivas listas a seguir, todas recebem um inteiro e devolvem um inteiro.

a) **maiorDigito** - retorna o maior dígito de um inteiro.

Solução. Segue código python:

```
def biggestDigit(num: int) -> int:
    return max(num%10, biggestDigit(num//10)) if num >= 10 else num
```

□

b) **menorDigito** - retorna o menor dígito de um inteiro.

Solução. Segue código python:

```
def lowestDigit(num: int) -> int:
    return min(num%10, lowestDigit(num//10)) if num >= 10 else num
```

□

c) **contaDigito** - retorna a quantidade de dígitos de um inteiro.

Solução. Segue código python:

```
def countDigits(num: int) -> int:
    return 1 + countDigits(num//10) if num >= 10 else 1
```

□

d) **somaDigito** - retorna a soma dos dígitos de um inteiro.

Solução. Segue código python:

```
def sumDigits(num: int) -> int:
    return num%10 + sumDigits(num//10) if num >= 10 else num%10
```

□

e) **zeraPares** - retorna um inteiro com os dígitos pares em zero.

Solução. Segue código python:

```
def zeroEvenDigits(num: int) -> int:
    return 10 * zeroEvenDigits(num//10) + (num%10)%2 * num%10 if num >= 10 else (num % 2) * num
```

□

f) **zeraImpares** - retorna um inteiro com os dígitos ímpares em zero.

Solução. Segue código python:

```
def zeroOddDigits(num: int) -> int:
    return 10 * zeroOddDigits(num//10) + ((num%10)%2 == 0) * num%10 if num >= 10 else (num % 2 == 0)
```

□

g) **removePares** - remove os dígitos pares de um inteiro.

Solução. Segue código python:

```
def removeEvenDigits(num: int) -> int:
    return removeEvenDigits(num//10) * (1 + (9 * ((num%10)%2))) + num%10 * ((num%10)%2) if num >= 10
```

□

h) **removeImpares** - remove os dígitos ímpares de um inteiro.

Solução. Segue código python:

```
def removeOddDigits(num: int) -> int:
    return removeOddDigits(num//10) * (1 + (9 * ((num%10)%2 == 0))) + num%10 * ((num%10)%2 == 0) if
```

□

i) **inverteNumero** - inverte os dígitos de um número inteiro.

Solução. Segue código python:

```
def invert(num: int) -> int:
    return (num % 10) * 10**(countDigits(num)-1) + invert(num//10) if num >= 10 else num
```

□

Problema 6. Crie uma função recursiva para verificar se uma string tem mais vogais do que consoantes.

Solução. Segue código python:

```
def moreVowelsThanConsonants(text: str, value: int = 0) -> bool:
    vowels = ['a', 'e', 'i', 'o', 'u']
    if len(text) == 1:
        value += text[0].lower() in vowels
        return value > 0
    if text[0] in vowels:
        return moreVowelsThanConsonants(text[1:], value+1)
    return moreVowelsThanConsonants(text[1:], value-1)
```

□

Problema 7. Implemente o algoritmo de busca binária em um vetor de inteiros ordenado.

Solução. Segue código python:

```
def binarySearch(lista: list, target: int) -> bool:
    if not lista:
        return False
    mid = len(lista) // 2
    if lista[mid] == target:
        return True
    elif lista[mid] < target:
        return binarySearch(lista[mid+1:], target)
    else:
        return binarySearch(lista[:mid], target)
```

□

Problema 8. Dados um vetor de inteiros distintos e ordenados de maneira crescente e um inteiro target, crie um algoritmo recursivo que determine se existem dois inteiros no vetor que a soma seja igual a target.

Solução. Segue código python:

```
def twoIntFromListSumEquals(target: int, lista: list) -> bool:
    if len(lista) < 2:
        return False
    if binarySearch(lista[1:], target - lista[0]):
        return True
    return twoIntFromListSumEquals(target, lista[1:])
```

□

Problema 9. Dado um array S não ordenado de inteiros e um inteiro k, crie um algoritmo recursivo para reorganizar os elementos de S tal que todos os elementos menores ou iguais a K apareçam antes do que os elementos maiores.

Solução. Segue código python:

```
def sortedOrderLessThan(k: int, lista: list) -> list:
    if len(lista) == 0:
        return []
    if lista[0] <= k:
        return [lista[0]] + sortedOrderLessThan(k, lista[1:])
    return sortedOrderLessThan(k, lista[1:]) + [lista[0]]
```

□