

## Atividade Avaliativa 1 Complexidade de Algoritmos e Recursão

### 1. Exercício: A Torre dos Algoritmos (teórico) (5 pontos)

Você foi convocado para escalar a lendária Torre dos Algoritmos, composta por três desafios. Em cada andar, é proposto um algoritmo diferente, e cabe a você analisar a eficiência de cada um deles. Considere que cada operação básica (atribuição) leva, em média, 1 microsegundo ( $10^{-6}$  segundos). Analise os algoritmos a seguir, escritos em pseudocódigo, e responda às perguntas.

```
Algoritmo_A(n) {  
  for i = 1 to n {  
    for j = i to n  
      processa(i, j)  
    }  
  }  
}
```

```
Algoritmo_B(n) {  
  if n <= 1  
    return 1  
  Algoritmo_B(n/2)  
  Algoritmo_B(n/2)  
  for i = 1 to n  
    processa(i)  
}
```

```
Algoritmo_C(n) {  
  for i = 1 to n {  
    j = 1  
    while j <= n {  
      processa(i, j)  
      j += 2 // vai de 2 em 2 (percorre metade dos elementos no while)  
    }  
  }  
}
```

Suponha que as primitivas `processa(i)` e `processa(i, j)` possuem complexidade  $O(1)$

- Mostre matematicamente quem é a função  $T(n)$  calcule a complexidade do algoritmo A.
- Mostre matematicamente quem é a função  $T(n)$  calcule a complexidade do algoritmo B.
- Mostre matematicamente quem é a função  $T(n)$  calcule a complexidade do algoritmo C.
- Usando a complexidade assintótica calculada no item anterior, qual(is) algoritmo(s) pode(m) ser executado(s) em tempo menor que 1 segundo para  $n = 10^5$ ?
- Se a primitiva `processa(i, j)` levar tempo proporcional a diferença entre  $i$  e  $j$ , mostre matematicamente como fica a complexidade dos algoritmos A e C). Dica: em A, ao invés de 1 some  $(j - i)$  em cada iteração e em C, ao invés de 1 some  $(i - j)$  em cada iteração.

## 2. O Problema do Reality Show (teórico/prático) (3 pontos)

Em um reality show, os participantes precisam decidir quem será o novo líder. A produção do programa organiza uma dinâmica descrita como segue: os  $n$  participantes estão dispostos em um círculo. A cada rodada, o apresentador sorteia um número  $k$  e conta  $k$  participantes no sentido horário, a partir da posição atual. O  $k$ -ésimo participante é eliminado do jogo. O processo se repete com os participantes restantes, começando a contagem a partir do próximo após o eliminado, até que apenas um participante reste, o vencedor e novo líder.

### Sua tarefa:

Desenvolva uma função recursiva que, dado o número total de participantes  $n$  e o passo de eliminação  $k$ , determine a posição do vencedor no círculo inicial.

### Exemplo:

Se  $n = 7$ ,  $k = 2$  e iniciarmos no primeiro participante, a eliminação ocorre assim:

```
1 2 3 4 5 6 7 → remove 3
1 2 4 5 6 7   → remove 6
1 2 4 5 7     → remove 2
1 4 5 7       → remove 7
1 4 5         → remove 5
1 4           → remove 1
4             Vencedor
```

Ordem de eliminação: 3, 6, 2, 7, 5, 1.

Sobrevivente: 4.

- a) Explique como a recursão pode modelar esse problema. (teórico)
- b) Implemente a solução recursiva em uma linguagem de sua escolha. (prático)
- c) Analise a complexidade do algoritmo (tempo). (teórico)

## 3. Problema: O Labirinto Mágico do Feiticeiro (prático) (2 pontos)

Você é um aventureiro preso em um labirinto mágico criado por um feiticeiro. As paredes do labirinto são vivas e se rearranjam sempre que você faz um movimento! Felizmente, você encontrou um pergaminho antigo com as seguintes regras:

O labirinto é uma matriz  $N \times N$ , onde você começa no canto superior esquerdo  $(0, 0)$  e deve chegar ao canto inferior direito  $(N-1, N-1)$ .

Você só pode se mover para a direita ( $\rightarrow$ ), para baixo ( $\downarrow$ ) ou na diagonal inferior-direita ( $\searrow$ ).

Algumas células estão amaldiçoadas (representadas por 0) e não podem ser pisadas. Células seguras são representadas por 1.

O labirinto é mágico: sempre que você pisa em uma célula segura, ela vira 0 (amaldiçoada) após sua passagem, para dificultar o caminho de volta.

**Sua missão:**

Implemente uma função recursiva que imprima todos os caminhos possíveis para escapar do labirinto, respeitando as restrições acima.

**Labirinto:**

```
[ [1, 1, 1],  
  [1, 0, 1],  
  [0, 1, 1] ]
```

**Saída:**

Caminho 1:  $\rightarrow \rightarrow \downarrow \downarrow$

Caminho 2:  $\rightarrow \searrow \downarrow$

Caminho 3:  $\downarrow \searrow \rightarrow$

**Requisitos:**

Condição de parada: pare quando alcançar (N-1, N-1) ou quando não houver movimentos válidos.

**Passo Recursivo:** Para cada célula atual, explore as 3 direções possíveis ( $\rightarrow$ ,  $\downarrow$ ,  $\searrow$ ) se a célula de destino for segura (1).

**Backtracking:** Marque células visitadas como 0 e as restaure após a recursão para permitir novos caminhos.

**Dica:** Use uma matriz auxiliar para rastrear o caminho atual.

Explique o seu algoritmo recursivo em detalhes e teste sua implementação com a seguinte matriz:

```
[[1, 1, 1, 0, 1],  
 [1, 0, 1, 1, 1],  
 [0, 1, 1, 1, 0],  
 [1, 0, 1, 0, 1],  
 [0, 1, 1, 1, 1]]
```

Quantos caminhos foram encontrados? Liste cada um deles.

**Observação:** em todos exercícios práticos, a resolução deve vir acompanhada de um link para execução do código fonte em ambiente virtual (onlineGDB ou Google Colab). O não envio do link acarretará em penalização da nota.

*"You will never speak to anyone more than you speak to yourself in your head. Be kind to yourself."*  
-- Author Unknown