Prof. Kennedy Lopes Algoritmos e Estrutura de Dados II 2023, 1º Semestre

LISTA DE EXERCÍCIOS

RECURSÃO

01. Através do algoritmo abaixo, calcule o fibonacci de 8. Neste procedimento, verifique quantas vezes o fibonacci(4) foi calculado.

```
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

```
fibonacci(8)
├── fibonacci(7)
|   ├── fibonacci(6)
|   |   ├── fibonacci(5)
|   |   |   ├── fibonacci(4)     ← (1ª vez)
|   |   |   └── fibonacci(3)
|   |   |       ├── fibonacci(2)
|   |   |       └── fibonacci(1)
|   |   └── fibonacci(4)     ← (2ª vez)
|   |       ├── fibonacci(3)
|   |       |   ├── fibonacci(2)
|   |       |   └── fibonacci(1)
|   |       └── fibonacci(2)
|   └── fibonacci(5)
|       ├── fibonacci(4)     ← (3ª vez)
|       |   ├── fibonacci(3)
|       |   |   ├── fibonacci(2)
|       |   |   └── fibonacci(1)
|       |   └── fibonacci(2)
|       └── fibonacci(3)
|           ├── fibonacci(2)
```

```
│           └── fibonacci(1)
└── fibonacci(6)
    ├── fibonacci(5)
    │   ├── fibonacci(4)          ← (4ª vez)
    │   │   ├── fibonacci(3)
    │   │   │   ├── fibonacci(2)
    │   │   │   └── fibonacci(1)
    │   │   └── fibonacci(2)
    │   └── fibonacci(3)
    │       ├── fibonacci(2)
    │       └── fibonacci(1)
    └── fibonacci(4)          ← (5ª vez)
        ├── fibonacci(3)
        │   ├── fibonacci(2)
        │   └── fibonacci(1)
        └── fibonacci(2)
```

02. Generalize a operação da questão 1 anterior considerando que estou calculando o fibonacci de qualquer valor maior do que 4.

operação = chamadas de fibonacci (4) em fibonacci (n) = fibonacci (n − 3)

(vale para n > 4)

1,1,2,3,5,8,13,21, 34.

03. Generalize a questão 2 considerando que estou calculando quantas vezes o fibonacci(n) é calculado para encontrar o fibonacci(m), sendo m > n.

chamadas de fibonacci (n) em fibonacci (m) = fibonacci (m − n + 1)

04. No algoritmo abaixo, considere os seguintes tempos:

- Chamada recursiva demora 2ns
- Retorno da chamada recursiva demora 1ns;
- Atribuição e soma demora 0.5ns;
- Divisão e multiplicação demora 1.5ns

```c
int funcRecursiva(int n) {
    if (n == 0) {
        return 1;
    }
    return funcRecursiva(n-1) + 1/funcRecursiva(n-1);
}
```

- Calcule o tempo total para funcRecursiva(5).

Código modificado:

```c
#include <stdio.h>

int contadorChamadas = 0; // variável global para contar o número de chamadas
double funcRecursiva(int n) {      // função recursiva
    contadorChamadas++;         // conta cada chamada feita

    if (n == 0) {
        return 1;
    } else {
        double resultado = funcRecursiva(n - 1);
        return resultado + 1 / resultado;
    }
}

int main() { // função principal
    int n = 5;

    contadorChamadas = 0; // zera o contador antes de começar
    double resultado = funcRecursiva(n);

    printf("Resultado de funcRecursiva(%d) = %.6lf\n", n, resultado);
    printf("Numero de chamadas recursivas: %d\n", contadorChamadas);

    return 0;
}
```

Saída do código:

```
PROBLEMAS    SAÍDA    CONSOLE DE DEPURAÇÃO    TERMINAL    PORTAS

PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02> cd 'c:\Users\Isa\Desktop\Isabel\UFERSA\II
I Período\Algoritmos & estrutura de dados II\listas\lista02\output'
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output> & .\'questao01.exe'
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output> cd 'c:\Users\Isa\Desktop\Isabel\UF
ERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output'
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output> & .\'questao04.exe'
Resultado de funcRecursiva(5) = 3.553010
Numero de chamadas recursivas: 6
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output>
```

Calculando o tempo total:

1. Tempos de cada operação:

   Chamada recursiva → 2 ns

   Retorno da chamada → 1 ns

   Atribuição → 0,5 ns

   Divisão → 1,5 ns

   Soma → 0,5 ns

   Retorno da função → 1 ns
2. Tempo por cada chamada = 6,5
3. Tempo do retorno = 1 ns
4. Tempo para T(5) = 6,5 × 5 + 1 = 33,5 ns

05. Refaça o procedimento da questão 4 considerando uma modificação no algoritmo:

```
int funcRecursiva(int n) {
    if (n == 0) {
        return 1;
    }
    k = funcRecursiva(n-1);
    return k + 1/k;
}
```

Mudança observada no código:

Antes fazia 2 chamadas recursivas. Agora faz 1 chamada recursiva, guarda o valor em k e usa.

1. Tempos por chamada:

   Chamada recursiva → 2 ns

   Retorno da chamada → 1 ns

   Atribuição → 0,5 ns

   Divisão (1/k) → 1,5 ns

   Soma (k + 1/k) → 0,5 ns

   Retorno final da função → 1 ns
2. Somando os tempos = 6,5
3. Para T(5) = (6,5 × 5) + 1 = 33,5 ns

06. Escreva, passo a passo, a execução do algoritmo fatorial em seu formato recursivo. Evidencie as chamadas recursivas e retornos da recursão.

Entendendo a questão em linguagem simples:

função fatorial(n)

  se n == 0 ou n == 1 então

    retorne 1

senão

retorne n * fatorial(n - 1)

Retorno da recursão:

fatorial(1) retorna 1

fatorial(2) retorna 2 * 1 = 2

fatorial(3) retorna 3 * 2 * 1 = 6

fatorial(4) retorna 4 * 3 * 2 * 1 = 24

fatorial(5) retorna 5 *4 * 3 * 2 * 1 = 120

07. No calendário gregoriano, geralmente um ano X é bissexto se o ano (x-4) também foi. Este pode ser uma etapa para calcular o algoritmo recursivo para um ano bissexto, mas não está correto por completo. Explique o porquê e apresente uma solução.

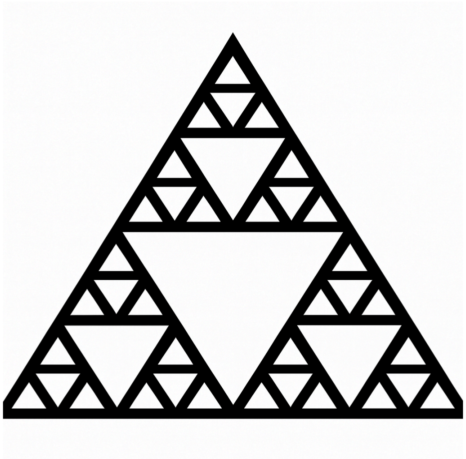
Explicação: A ideia de usar o ano (X - 4) para determinar se o ano X é bissexto não é suficiente porque não leva em conta a exceção dos anos múltiplos de 100 — que só são bissextos se também forem múltiplos de 400. A solução correta está nas regras fixas baseadas em divisibilidade por 4, 100 e 400.

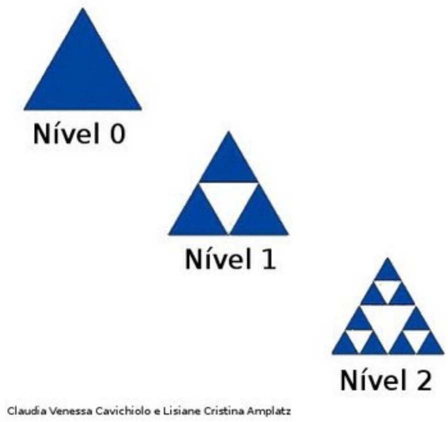
Solução:

```
função bissexto {
   se ano < 1582 }
então
      retorne "Ano fora do calendário gregoriano"
   se { ano % 400 == 0 }
então
      retorne verdadeiro
   se { ano % 100 == 0 }
 então
      retorne falso
   se { ano % 4 == 0 }
 então
      retorne verdadeiro
   senão
         retorne falso
fim algoritmo.
```

08. Um fractal é uma estrutura geométrica complexa que exibe repetição infinita de padrões semelhantes em diferentes escalas. Um exemplo de fractal simples é o triângulo de Sierpinski. Ele começa com um triângulo equilátero grande. Em seguida, cada lado desse triângulo é dividido em

três partes iguais e o triângulo central é removido. Esse processo é repetido para os triângulos restantes, criando uma sequência infinita de triângulos menores que se assemelham ao triângulo original. Desenhe um triângulo de Sierpinski tal qual descrito no texto.





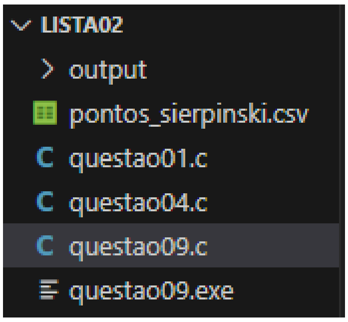
09. Desenhe um triângulo Sierpinski (descrito na questão 8) computacionalmente através do seguinte algoritmo:
    a) Marque os pontos: A = (0, 1); B = (−1,−1); C = (1,−1).
    b) Escolha um ponto aleatório P que esteja no interior do triângulo formado por pelos pontos A, B e C.
    c) Marque o ponto médio entre P e um ponto escolhido aleatoriamente entre A, B e C.

Algoritmo:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() { // função principal
    double A_x = 0, A_y = 1; // definindo os vértices do triângulo
    double B_x = -1, B_y = -1;
    double C_x = 1, C_y = -1;

    double P_x = 0, P_y = 0; // ponto inicial

    int num_pontos = 10000; // número de pontos a serem gerados

       FILE  *arquivo  =  fopen("pontos_sierpinski.csv",  "w");  //  abrir
arquivo para salvar os pontos
    if (arquivo == NULL) {
        printf("Erro ao criar o arquivo.\n");
        return 1;
    }
```

```
    fprintf(arquivo, "x,y\n"); // cabeçalho do csv

    srand(time(NULL)); // inicializar o gerador de números aleatórios

    for (int i = 0; i < num_pontos; i++) {
        int escolha = rand() % 3; // para escolher um vértice aleatório
        double alvo_x, alvo_y;

        if (escolha == 0) {
            alvo_x = A_x;
            alvo_y = A_y;
        } else if (escolha == 1) {
            alvo_x = B_x;
            alvo_y = B_y;
        } else {
            alvo_x = C_x;
            alvo_y = C_y;
        }

          P_x = (P_x + alvo_x) / 2.0; // função para calcular o ponto
médio
        P_y = (P_y + alvo_y) / 2.0;

         fprintf(arquivo, "%lf,%lf\n", P_x, P_y); // função para salvar
o ponto no arquivo
    }

    fclose(arquivo);

                printf("Pontos    gerados    e    salvos    no    arquivo
'pontos_sierpinski.csv'.\n");

    return 0;
}
```

Saída do código:



10. Apresente um algoritmo de recursão com e sem cauda executa a seguinte expressão:

$$p(x, n) = \prod_{k=0}^{n} (x - k)$$

Algoritmo sem cauda:

```c
#include <stdio.h>

double p_sem_cauda(double x, int n) { // recursão sem cauda
    if (n < 0) {
        return 1;
    } else {
        return (x - n) * p_sem_cauda(x, n - 1);
    }
}

int main() {
    double x;
    int n;

    printf("Digite o valor de x: ");
    scanf("%lf", &x);

    printf("Digite o valor de n: ");
    scanf("%d", &n);

    double resultado = p_sem_cauda(x, n);

    printf("Resultado (recursao sem cauda): %lf\n", resultado);

    return 0;
}
```

Algoritmo com cauda:

```
#include <stdio.h>

double com_cauda_aux(double x, int n, double acumulador) {// função auxiliar para recursão com cauda
    if (n < 0) {
        return acumulador;
    } else {
        return com_cauda_aux(x, n - 1, acumulador * (x - n));
    }
}

double com_cauda(double x, int n) { //função principal que chama a auxiliar
    return com_cauda_aux(x, n, 1);
}

int main() {
    double x;
    int n;

    printf("Digite o valor de x: ");
    scanf("%lf", &x);

    printf("Digite o valor de n: ");
    scanf("%d", &n);

    double resultado = com_cauda(x, n);

    printf("Resultado (recursao com cauda): %lf\n", resultado);

    return 0;
}
```

11. Apresente versões recursivas de cauda para cada uma das expressões abaixo:

a) f(n) = n!

Código:

```
#include <stdio.h>

int fatorial_tail(int n, int acc) {// função fatorial com recursão de cauda
```

```
    if (n == 0) return acc;
    return fatorial_tail(n - 1, acc * n);
}

int fatorial(int n) { // função wrapper (interface para o usuário)
    return fatorial_tail(n, 1);
}


int main() { //função principal
    int num;

    printf("Digite um numero inteiro: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("Nao existe fatorial de numero negativo\n");
    } else {
        printf("Fatorial de %d e: %d\n", num, fatorial(num));
    }

    return 0;
}
```

Saída do código:

```
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output> & .\'11a.exe'
Digite um numero inteiro: 3
Fatorial de 3 e: 6
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output>
```

b) $f(n) = 2f(n-1) + 3f(n-2)$, $f(0) = 1$, $f(1) = 2$

(c) $\sum_{k=1}^{M} k$

Código:

```
#include <stdio.h>

int f_tail(int n, int a, int b) { // função recursiva de cauda
    if (n == 0) return a;
    if (n == 1) return b;
    return f_tail(n - 1, b, 2 * b + 3 * a);
}
```

```c
int f(int n) { // função wrapper (interface simples)
    return f_tail(n, 1, 2);
}


int main() { //função principal
    int num;

    printf("Digite um numero inteiro: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("Erro. Digite valores positivos\n");
    } else {
        printf("f(%d) = %d\n", num, f(num));
    }

    return 0;
}
```

Saída do código:

```
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output> & .\'questao11b.exe'
Digite um numero inteiro: 3
f(3) = 20
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output>
```

12. Calcule o sin(80) considerando como caso base o resultado que $\sin(x) = x - x^3/6$ e que:

$$\begin{cases} \sin(x) & = \sin\left(\frac{x}{3}\right)\left(\frac{3-\tan^2\left(\frac{x}{3}\right)}{1+\tan^2\left(\frac{x}{3}\right)}\right) \\ \tan(x) & = \frac{\sin(x)}{\cos(x)} \\ \cos(x) & = 1 - \sin\left(\frac{x}{2}\right) \end{cases}$$

Passo 1: Converter 80° para radianos

$x = 80° = (80 * \pi) / 180 = (4\pi / 9) \approx 1{,}3963$ rad

Passo 2: Aplicar sin(x/3)

$x_1 = x / 3 \approx 1{,}3963 / 3 = 0{,}4654$ rad

Como $x_1 < 0{,}5$, podemos usar a aproximação:

$\sin(x_1) \approx x_1 - x_1^3/6 \approx 0{,}4654 - (0{,}4654^3)/6 \approx 0{,}4654 - 0{,}0168 = 0{,}4486$

Passo 3: Calcular $\tan(x_1)$
$\cos(x_1) = 1 - \sin(x_1 / 2)$
$x_1 / 2 = 0{,}2327$
$\sin(0{,}2327) \approx 0{,}2327 - (0{,}2327^3)/6 \approx 0{,}2306$
$\cos(0{,}4654) \approx 1 - 0{,}2306 = 0{,}7694$
$\tan(0{,}4654) \approx \sin(0{,}4654) / \cos(0{,}4654) \approx 0{,}4486 / 0{,}7694 \approx 0{,}5829$

Passo 4: Aplicar fórmula principal
$\tan^2(0{,}4654) \approx (0{,}5829)^2 \approx 0{,}3398$
$\text{Fator} = (3 - 0{,}3398) / (1 + 0{,}3398) = 2{,}6602 / 1{,}3398 \approx 1{,}985$
$\sin(1{,}3963) \approx 0{,}4486 * 1{,}985 \approx 0{,}890$

13. Implemente uma versão recursiva dos algoritmos abaixo:

a) Somas sucessivas para calcular o produto de dois números.
b) Divisão inteira entre dois números através de abstrações sucessivas.
c) Verificação se uma palavra é um palíndromo.
d) Inversão de uma string.
e) Geração de todos os números da mega sena (6 números entre 1 e 60).

Código:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 6  // para as combinações da Mega-Sena
#define LIMITE_COMBINACOES 1  // função para limitar os números da mega
sena

int contador_combinacoes = 0;  // contador global

/*-----------------------------------------------
a - produto por somas sucessivas (recursivo)
-----------------------------------------------*/
int produto(int a, int b) {
    if (b == 0) return 0;
    return a + produto(a, b - 1);
}

/*-----------------------------------------------
b - divisão inteira por subtrações sucessivas
-----------------------------------------------*/
int divisao_inteira(int dividendo, int divisor) {
    if (dividendo < divisor) return 0;
```

```c
    return 1 + divisao_inteira(dividendo - divisor, divisor);
}

/*------------------------------------------------
c - verificar se uma palavra é palíndromo (recursivo)
------------------------------------------------*/
bool palindromo(const char *str, int inicio, int fim) {
    if (inicio >= fim) return true;
    if (str[inicio] != str[fim]) return false;
    return palindromo(str, inicio + 1, fim - 1);
}

/*------------------------------------------------
d - inverter uma string (recursivo)
------------------------------------------------*/
void inverter_string(char *str, int inicio, int fim) {
    if (inicio >= fim) return;
    char temp = str[inicio];
    str[inicio] = str[fim];
    str[fim] = temp;
    inverter_string(str, inicio + 1, fim - 1);
}

/*------------------------------------------------
e - gerar combinações da mega sena
------------------------------------------------*/
void gerar_combinacoes(int start, int profundidade, int combinacao[]) {
    if (contador_combinacoes >= LIMITE_COMBINACOES) return;

    if (profundidade == MAX) {
        for (int i = 0; i < MAX; i++) {
            printf("%02d ", combinacao[i]);
        }
        printf("\n");
        contador_combinacoes++;
        return;
    }

    for (int i = start; i <= 60; i++) {
        combinacao[profundidade] = i;
        gerar_combinacoes(i + 1, profundidade + 1, combinacao);

        if (contador_combinacoes >= LIMITE_COMBINACOES) break;
```

```
    }
}


int main() { //função principal

    int a = 4, b = 3;  // teste do produto
    printf("Produto de %d e %d = %d\n", a, b, produto(a, b));


    int dividendo = 15, divisor = 2;   // teste da divisão inteira
     printf("Divisao inteira de %d por %d = %d\n", dividendo, divisor,
divisao_inteira(dividendo, divisor));


    const char *palavra = "arara"; // teste de palíndromo
     printf("'%s' e palindromo? %s\n", palavra, palindromo(palavra, 0,
strlen(palavra) - 1) ? "Sim" : "Nao");


    char texto[] = "cacau"; // teste de inversão de string
    inverter_string(texto, 0, strlen(texto) - 1);
    printf("String invertida: %s\n", texto);


          printf("Combinacoes da Mega-Sena (limitadas a %d):\n",
LIMITE_COMBINACOES); // teste da mega sena
    int combinacao[MAX];
    gerar_combinacoes(1, 0, combinacao);

    return 0;
}
```

Saída do código:

```
ERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output'
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output> & .\'questao13.exe'
Produto de 4 e 3 = 12
Divisao inteira de 15 por 2 = 7
'arara' e palindromo? Sim
String invertida: uacac
Combinacoes da Mega-Sena (limitadas a 1):
01 02 03 04 05 06
PS C:\Users\Isa\Desktop\Isabel\UFERSA\III Período\Algoritmos & estrutura de dados II\listas\lista02\output>
```

COMPLEXIDADE

14. Explique as seguintes afirmações e questionamentos:

a) A função f(n) tem complexidade O(n2).

O tempo cresce rápido, o quadrado do tamanho da entrada. Se a entrada dobra, o tempo fica 4x maior.

b) O tempo necessário para execução do algoritmo tem complexidade Θ(n log n)

O tempo cresce mais devagar que o quadrado, algo entre linear e quadrático. É mais eficiente que $O(n^2)$.

c) Qual o algoritmo mais veloz, o que tem complexidade O(n) ou um outro com Θ(n2) ?

Algoritmo com O(n) é mais rápido que um com $\Theta(n^2)$, principalmente quando recebe uma entrada grande, mas tudo depende da comparação entre os algoritmos.

15. Quais as complexidades de tempo dos algoritmos abaixo (big O):

a) A1: Ordenação de uma lista sequencial não ordenada;

   Depende do algoritmo

b) A2: Busca de um elemento em uma pilha formado por lista encadeada.

   O(n)

c) A3: Busca de elementos em uma lista linear encadeada ordenada;

   O(n)

d) A4: Inserção de elemento numa fila formado por lista encadeada;

   1

e) Remoção de elemento em uma lista sequencial;

   O(n)

16.Quais as complexidades de memória dos algoritmos abaixo (big O):

a) Inserção de elementos em uma pilha; O(n)

b) Inserção de elementos em uma fila; O(n)

c) Remoção de elementos em uma pilha; O

d) Remoção de elementos em uma fila; O

17. Prove se é verdadeiro ou falso:

a) n2 é O(n3); Verdadeiro

b) n3 é O(n2); Falso

c) log10(n2) é O(lg(n)) Verdadeiro

d) n2 sin 2(n) é O(n2) Verdadeiro

18. O algoritmo A possui complexidade O(n5) e o algoritmo B possui complexidade O(1.5n). Ambos realizam a mesma operação. Qual dos dois você utilizaria?

O Algoritmo A $(O(n^5))$, porque, apesar de ser pesado, ele é polinomial. O B é exponencial, e exponenciais sempre ficam piores à medida que n cresce, apesar disso, ambos são ruins por não suportarem grandes entradas.

19. A quantidade de operações de um algoritmo A é de TA(n) = 2n2+5, do algoritmo B é TB(n) = 100n. Até qual tamanho de problema o algoritmo A é mais eficiente do que o B? O algoritmo A pode ser mais eficiente do que o algoritmo B, mas deve-se levar em consideração sua função, utilidade e complexidade.

20. Calcule a complexidade do algoritmo abaixo: O(n)

```
int f(int n) {
    int s = 0;
    for(int i=0; i<n; i++)
        for(int k=n; k<i; k++)
            s = s + i;
}
```

21. Calcule a complexidade do da função main: $O(n^2)$

```
int f(int n) {
    int s = 0;
    for(int i=0; i<n*n; i++)
        s = s + i;
}
int g(int n){
    f(n/2) * f(n/2);
}
int main(){
    int d;
    scanf("%d\n", d);
    g(d);
}
```

22. Prove se é verdadeiro ou falso:

a) n2 é Θ(n3); Falso

b) n3 é Θ(n2); Falso

c) log 10(n2) é Θ(lg(n)) Verdadeiro

d) n2 cos2(n) é Θ(n2) Falso

23. Prove se é verdadeiro ou falso:

a) n2 é Ω(n3); Falso

b) n3 é Ω(n2); Verdadeiro

c) log 10(n2) é Ω(lg(n)) Verdadeiro

d) n2 cos2(n) é Ω(n2) Falso

24. Julgue as afirmções em (V) Verdadeiro ou (F) Falso:

a) c1O(f(n)) = O(c1 * f(n)) Verdadeiro

b) O(f(n) + g(n)) = O(f(n) * g(n)) Falso

c) O(f(n)) + O(g(n)) = O(max(f(n), g(n))) Verdadeiro

d) f(n)O(g(n)) = O(g(n)) Falso

CONCEITOS INICIAIS DE ÁRVORES

25. Construa uma árvore binária qualquer com os elementos:

49 50 44 54 12 61 68 87 59 30 42 51 33 41 27

```
       49
      /  \
    44    50
   /        \
 12          54
   \        /  \
    30     51   61
   /  \   /      \
 27   42 59       68
     /              \
    33               87
      \
       41
```

Sucessor: 50

Antecessor: 42

Nó	Nível
49	0
44	1
50	1
12	2
54	2
30	3
51	3
61	3
27	4
42	4
59	4

68 4

33 5

87 5

41 6

Nó	Altura
41	0
33	1
42	2
27	0
30	3
12	4
44	5
51	0
59	0
87	0

68	1
61	2
54	3
50	4
49	6

Percurso pré-ordem: 49, 44, 12, 30, 27, 42, 33, 41, 50, 54, 51, 61, 59, 68, 87

In ordem: 12, 27, 30, 33, 41, 42, 44, 49, 50, 51, 54, 59, 61, 68, 87

Pós ordem: 27, 41, 33, 42, 30, 12, 44, 51, 59, 87, 68, 61, 54, 50, 49

26. Construa uma árvore estritamente binária com os elementos:

28 99 78 96 64 63 51 86 43 76 86 76 12 18 89

```
                28
          /           \
        99             78
       /  \          /    \
      96   64       63     51
     / \   / \     / \    / \
    86 43 76 86   76 12  18  89
```

Sucessor: 12

Antecessor: -

Nó	Nível
28	0
99	1
78	1
96	2
64	2

63	2
51	2
86	3
43	3
76	3
86	3
76	3
12	3
18	3
89	3

Nó	Altura
86	0
43	0
76	0
86	0
76	0
12	0
18	0
89	0
96	1
64	1
63	1
51	1

99	2
78	2
28	3

Percurso em pré-ordem: 28, 99, 96, 86, 43, 64, 76, 86, 78, 63, 76, 12, 51, 18, 89

In ordem: 86, 96, 43, 99, 76, 64, 86, 28, 76, 63, 12, 78, 18, 51, 89

Pós-ordem: 86, 43, 96, 76, 86, 64, 99, 76, 12, 63, 18, 89, 51, 78, 28

27. Construa uma árvore binária cheia com os elementos:

61 66 83 39 78 18 95 19 63 45 44 55 47 45 86

```
            61
          /    \
        66       83
       /  \      /  \
     39    78   18   95
     / \   / \ / \  / \
   19 63 45 44 55 47 45 86
```

Sucessor: 66

Antecessor: 39

Nó	Nível
61	0
66	1
83	1
39	2
78	2
18	2
95	2
19	3

63	3
45	3
44	3
55	3
47	3
45	3
86	3

Nó	Altura
19	0
63	0
45	0
44	0
55	0
47	0
45	0
86	0
39	1

78	1
18	1
95	1
66	2
83	2
61	3

Pré-ordem: 61, 66, 39, 19, 63, 78, 45, 44, 83, 18, 55, 47, 95, 45, 86

In ordem: 19, 39, 63, 66, 45, 78, 44, 61, 55, 18, 47, 83, 45, 95, 86

Pós-ordem: 19, 63, 39, 45, 44, 78, 66, 55, 47, 18, 45, 86, 95, 83, 61

28. Construa uma árvore binária completa com os elementos:

84 96 36 30 78 49 77 55 79 76 74 54 67 98 10

```
               84
             /    \
         96          36
        /  \        /   \
      30    78    49     77
     /  \  /  \   / \    /  \
    55 79 76 74 54  67 98    10
```

Antecessor: 79

Sucessor: 98

Nó	Nível
84	0
96	1
36	1

30	2
78	2
49	2
77	2
55	3
79	3
76	3
74	3
54	3
67	3
98	3
10	3

Nó	Altura
55	0
79	0
76	0
74	0
54	0
67	0

98 0

10 0

30 1

78 1

49 1

77 1

96 2

36 2

84 3

Pré-ordem: 84, 96, 30, 55, 79, 78, 76, 74, 36, 49, 54, 67, 77, 98, 10

In ordem: 55, 30, 79, 96, 76, 78, 74, 84, 54, 49, 67, 36, 98, 77, 10

Pós-ordem: 55, 79, 30, 76, 74, 78, 96, 54, 67, 49, 98, 10, 77, 36, 84

29. Construa uma árvore binária de busca com os elementos:

91 32 56 90 63 49 20 62 47 87 16 56 35 32 90

```
        91
        /
       32
    /     \
   20      56
  /       /  \
 16   49     90
     /     /   \
   47    63      90
```

```
   /      /        \
 35     62         87
 /         \
32         56
```

Sucessor: -

Antecessor: 90

Nó	Nível
91	0
32	1
20	2
56	2
16	3
49	3
90	3
47	4
63	4
90	4

35	5
62	5
87	5
32	6
56	7

Nó	Altura
56	0
32	1
35	2
47	3
49	4
16	0
20	5
62	0

63 1

87 0

90 2

56 6

32 7

91 8

Pré-ordem: 91, 32, 20, 16, 56, 49, 47, 35, 32, 56, 90, 63, 62, 90, 87

In ordem: 16, 20, 32, 32, 35, 47, 49, 56, 56, 62, 63, 90, 87, 90, 32, 91

Pós-ordem: 16, 20, 32, 56, 32, 35, 47, 49, 62, 63, 87, 90, 90, 56, 32, 91

30. Construa uma árvore binária de busca Zigue-Zague com os elementos:

19 22 48 46 24 39 80 41 62 73 75 32 46 79 24

```
    19
     \
      22
       \
        48
       /  \
     46    80
    / \    /
  24  46  62
   \       \
    39      73
   /  \      \
 32    41     75
               \
                79
```

Sucessor: 22

Antecessor: -

Nó	Nível
19	0
22	1
48	2
46	3
80	3
24	4
46	4
62	4
39	5
73	5
32	6
41	6
75	6
79	7

Nó	Altura
32	0
41	0
39	1
24	2
46 (dir.)	0
46 (esq.)	3

62	2
73	3
75	1
79	0
80	4
48	5
22	6
19	7

Pré-ordem: 19, 22, 48, 46, 24, 39, 32, 41, 46, 80, 62, 73, 75, 79

In ordem: 19, 22, 24, 32, 39, 41, 46, 46, 48, 62, 73, 75, 79, 80

Pós-ordem: 32, 41, 39, 24, 46, 46, 62, 79, 75, 73, 80, 48, 22, 19

Utilizem das questões 25 a 30 para responder às questões 31 a 36:

31. Calcule os sucessores e antecessores dos nós raízes.
32. Localize o nível de todos os nós. -
33. Localize a altura de todos os nós. -
34. Realize o percurso em pré-ordem. -
35. Realize o percurso em pós-ordem. -
36. Realize o percurso em in-ordem. -
37. Julgue (V) Verdadeiro ou (F) Falso:
    a) Toda árvore binária cheia é completa. (Falso)
    b) Toda árvore binária de busca Zigue-Zague possui como raiz o menor elemento. (Falso)
    c) É possível uma árvore ser simultaneamente: Estritamente binária, Cheia, Completa e Zigue-Zague. (Verdadeiro)
    d) É possível uma árvore ser simultaneamente: Estitamente binária, Cheia, Completa. Mas ser também Zigue-Zague é impossível. (Falso)
    e) Se o antecessor e sucessor de um nó são irmãos, então o nó é pai dos dois. (Falso)
    f) O tio de um nó é ancestral a ele. (Falso)
    g) O neto de um nó é um descendente dele. (Verdadeiro)
38. Qual a altura de uma árvore cheia que possui N=324 nós?

Fórmula:

N=2^h+1−1

324=2^h+1−1

h+1 = log (325) / log (2)

h+1≈ 8.35

h ≈ 7.35

39. Quantos nós faltam para uma árvore cheia com N=348 nós se torne uma árvore completa?

Fórmula:

N=2^h+1−1

altura (7) = 2^8-1 = 255

altura (8) = 2^9-1=511

511−348=163 (já é uma árvore completa, só não está cheia ainda)

40. Quantos nós precisariam ser removidos para que uma árvore cheia com N=538 nós se torne uma árvore completa?

Fórmula:

N=2^h+1−1

Altura 8 ⟶ N = 2^9−1=511

Altura 9 ⟶ N = 2^10−1=1023

1023 − 538 = 485

41. Qual a maior altura do nó raiz em uma árvore estritamente binária com N=251 nós.

Fórmula:

h = n-1 / 2

h = 251 - 1/ 2

h = 250 / 2 = 125