

# Estruturas de Dados

## Algoritmos e Recursividade

Universidade Estadual Vale do Acaraú – UVA

---

Paulo Regis Menezes Sousa

paulo\_regis@uvanet.br

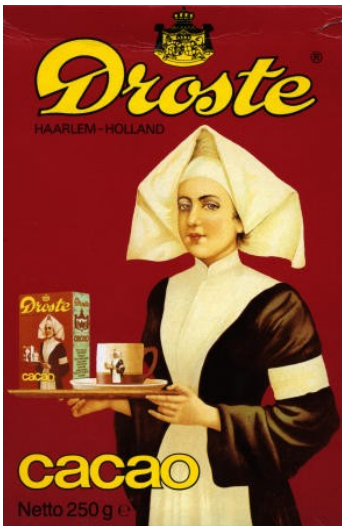
# Implementações recursivas

Dividir para Conquistar

- O que é recursão?
  - É um método de programação no qual uma função pode chamar a si mesma
  - O termo é usado de maneira mais geral para descrever o processo de repetição de um objeto de um jeito similar ao que já fora mostrado
- Por que precisamos aprender recursão?
  - Paradigma de programação poderoso
  - Nova maneira de pensar
- Muitas estruturas têm natureza recursiva:
  - Estruturas encadeadas
  - Fatorial, máximo divisor comum
  - Uma pasta que contém outras pastas e arquivos

- **Efeito Droste** é o nome dado ao efeito de uma imagem que aparece dentro de si mesma, em um lugar onde seria realista esperar uma imagem semelhante aparecer.





- A origem do efeito ocorreu no ano de 1904 e ficou conhecido como Droste devido a imagem das caixas de cacau Droste, que exibia uma enfermeira carregando uma bandeja com uma xícara de chocolate e uma caixa com a mesma imagem.

- $\text{mdc}(p, q)$ : encontre o maior divisor comum entre  $p$  e  $q$ ;
- $\text{mdc}(180, 240) = ?$

180	240	2
90	120	2
45	60	2
45	30	2
45	15	3
15	5	3
5	5	5
1	1	$\text{mdc}(180, 240) = 2^2 \times 3 \times 5 = 60$

- $\text{mdc}(p, q)$ : encontre o maior divisor comum entre  $p$  e  $q$ ;
- $\text{mdc}(180, 240) = ?$

180	240	2
90	120	2
45	60	2
45	30	2
45	15	3
15	5	3
5	5	5
1	1	

$$\text{mdc}(180, 240) = 2^2 \times 3 \times 5 = 60$$

- Uso de mdc:
  - Simplificação de frações:

$$\frac{180}{240} = \frac{36}{48}$$

- Importante em mecanismos de criptografia.
- Algoritmo de Euclides:


$$\text{mdc}(p, q) = \begin{cases} p, & \text{se } q = 0 \\ \text{mdc}(q, p \% q), & \text{caso contrário} \end{cases}$$

$$\begin{aligned} \text{mdc}(240, 180) &= \text{mdc}(180, 60) & 240 \% 180 &= 60 \\ &= \text{mdc}(60, 0) & 180 \% 60 &= 0 \\ &= 60 \end{aligned}$$



### ● Implementação em C

```
1  int mdc(int p, int q) {  
2      if (q == 0)  
3          return p; // caso base  
4      else  
5          return mdc(q, p % q); // passo de redução  
6  }
```

```
 int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```

```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```


```
int mdc(int p, int q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```

```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```


```
int mdc(int p, int q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```

```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```


```
int mdc(int p, int q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```




```
int mdc(int 6p, int 4q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 4p, int 2q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```




```
int mdc(int 6p, int 4q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 4p, int 2q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```



```
int mdc(int 6p, int 4q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 4p, int 2q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 2p, int 0q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```







```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```




```
int mdc(int 6p, int 4q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 4p, int 2q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 2p, int 0q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```




```
int mdc(int 6p, int 4q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 4p, int 2q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```




```
int mdc(int 2p, int 0q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```



```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```



```
int mdc(int 6p, int 4q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```



```
int mdc(int 4p, int 2q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```


2

```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```


```
int mdc(int p, int q){  
    if (q == 0)  
        return p;  
    else  
        return mdc(q, p % q);  
}
```

Diagram illustrating the recursive call for `mdc(6, 4)`:

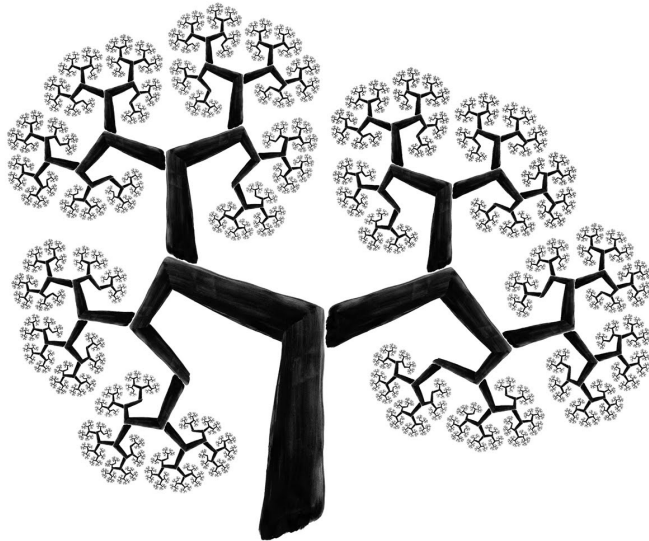
- Initial call: `mdc(6, 4)` (p=6, q=4)
- First recursive call: `mdc(4, 2)` (p=4, q=2)
- Second recursive call: `mdc(2, 0)` (p=2, q=0)
- Base case reached: `mdc(2, 0)` returns 2.

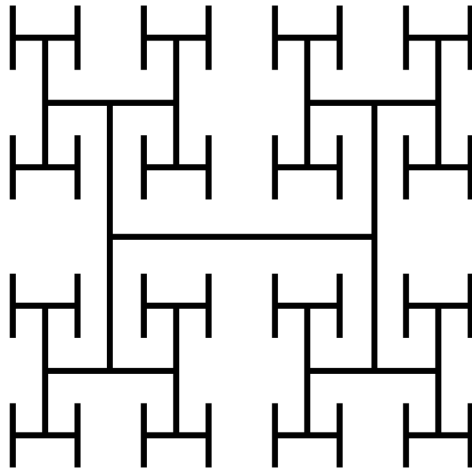


```
int main() {  
    int n = mdc(6,4);  
    return 0, 2;  
}
```



```
int main() {  
    int n = mdc(6,4);  
    return 0;  
}
```

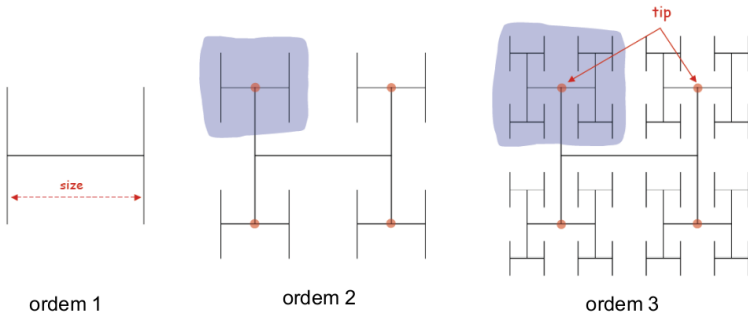






## Árvore H

- Árvore-H de ordem  $n$ 
  - Desenha uma letra H
  - Recursivamente desenha 4 árvores-H da ordem de  $n-1$  (e metade do tamanho), cada árvore conectada em um “topo” (*tip*).

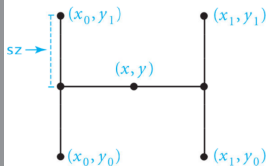


## Árvore H (em C)

```
1 void draw(int n, double len, double x, double y) {  
2     double x0, x1, y0, y1;  
3     if (n > 0) {  
4         x0 = x - len/2;  
5         x1 = x + len/2;  
6         y0 = y - len/2;  
7         y1 = y + len/2;  
8  
9         drawLine(x0, y, x1, y);  
10        drawLine(x0, y0, x0, y1);  
11        drawLine(x1, y0, x1, y1);  
12  
13        draw(n-1, len/2, x0, y0);  
14        draw(n-1, len/2, x0, y1);  
15        draw(n-1, len/2, x1, y0);  
16        draw(n-1, len/2, x1, y1);  
17    }  
18 }
```

*desenha o H centralizado em  $(x,y)$*

*recursivamente  
desenha 4 Hs  
com a metade do  
tamanho*



- Consiste em dividir o problema em problemas menores
- Problemas menores são resolvidos recursivamente usando o mesmo método
- Resultados são combinados para resolver problema original
- Vários algoritmos são resolvidos com essa técnica (e.x., quicksort, mergesort)

## Exercício 1

Crie uma função recursiva para a impressão de um vetor de números inteiros.

## Exercício 2

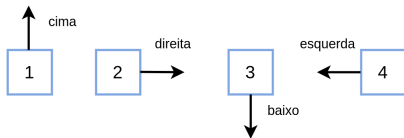
Crie uma função recursiva para realizar a soma de todos os elementos de um vetor de números reais.

## Exercício 3

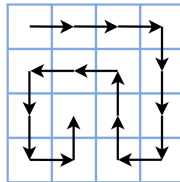
A figura abaixo descreve um caminho dentro de uma matriz  $4 \times 4$  com um código de direções em que os números 1, 2, 3 e 4 representam respectivamente as direções: cima, direita, baixo e esquerda como mostra a figura. O caminho se inicia na posição (0,0).

Crie uma matriz  $4 \times 4$  de números inteiros e a inicialize como a da figura. Crie uma função recursiva que percorre a matriz obedecendo ao código de direções, até encontrar o número 8, a cada passo que a função der na matriz imprima a linha e coluna da posição visitada.

OBS: não use nenhuma estrutura de repetição, apenas recursão.



2	2	2	3
3	4	4	3
3	8	1	3
2	1	1	4



#### Exercício 4

Implemente uma função recursiva que, dados dois números inteiros  $x$  e  $n$ , calcula o valor de  $x^n$ .

#### Exercício 5

Crie uma função recursiva imprimir um número inteiro positivo na sua forma binária.