Introdução a Técnicas de Programação

Recursão

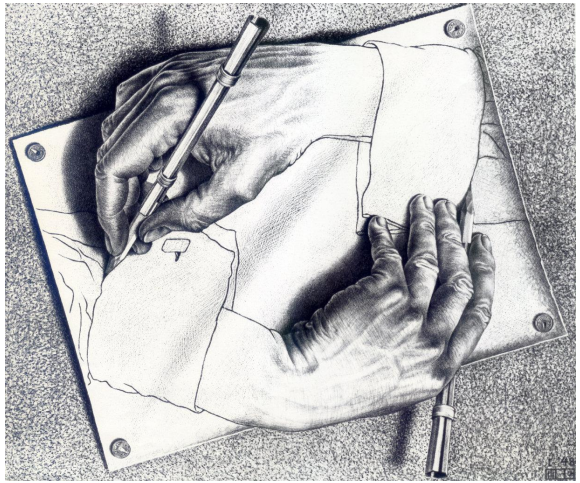
Prof. André Campos
DIMAp/UFRN

Situação-problema

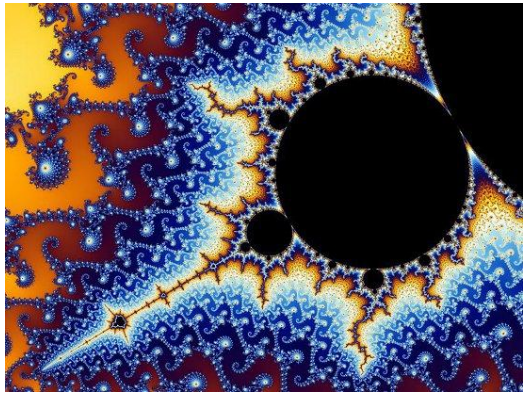
Descreva um algoritmo para traçar um caminho entre dois pontos quaisquer em um labirinto

O que é recursão (ou recursividade)?

Descreve algo definido através de um processo de repetição de si mesmo.

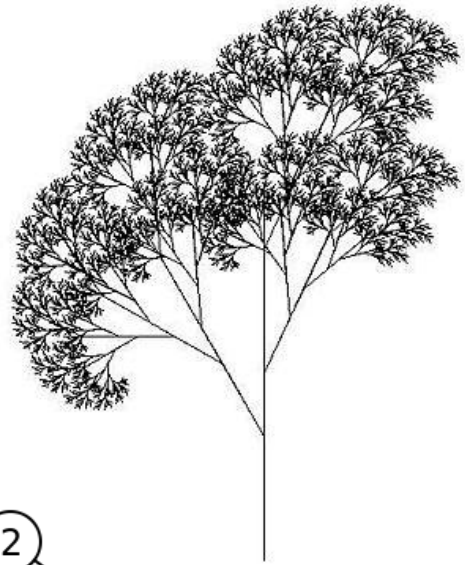


M.C.Escher



$$\mathbb{N} = \begin{cases} 0 \\ n+1 & \text{se } n \in \mathbb{N} \end{cases}$$

Recursividade em computação

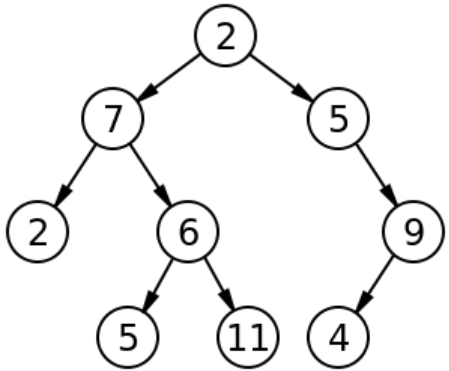


Técnica de resolução de problemas

- Redução de um problema maior em um menor
- As vezes fica mais fácil resolver problemas menores

Áreas

- Computação gráfica (subdivisões espaciais)
- Algoritmos (ordenação)
- Estruturas de dados (árvores de busca)
- Compiladores (árvores sintáticas)
- Jogos (geração procedural de conteúdo)
- ...



Definição recursiva

Elementos essenciais

1. **Caso base**
   caso trivial que permite "parar" a recursividade

2. **Caso recursivo**
   caso cuja definição se repete... até chegar no caso base

Exemplos

Definição de **descendente**

1. Os filhos de uma pessoa são seus descendentes
2. Os filhos dos descendentes de uma pessoa são também seus descendentes

Definição dos **números naturais**

1. 0 é um número natural
2. Se n é um número natural, $n+1$ também é

Definição de **fatorial**

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n.(n-1)! & \text{se } n > 1 \end{cases}$$

Resolução de problemas usando recursão

Ótima ferramenta para **quebrar problemas complexos** em mais simples

Ao tentar resolver um problema complexo, reduz o problema e aplica a mesma estratégia no problema mais simples.

Para ilustrar (algo simples)... calcular o somatório de uma sequência de inteiros

$$\textbf{Soma} \text{ de } \{ s_1, s_2, s_3, \ldots, s_n \} = s_1 + \textbf{Soma} \text{ de } \{ s_2, s_3, \ldots, s_n \}$$

$$\textbf{Soma} \text{ de } \{ s_2, s_3, \ldots, s_n \} = s_2 + \textbf{Soma} \text{ de } \{ s_3, \ldots, s_n \}$$

$$\ldots$$

$$\textbf{Soma} \text{ de } \{ s_n \} = s_n$$

Função recursiva

1. **Caso base**

   Define quando a recursão deve parar.

   Se não houver, haverá loop infinito!

   No exemplo, a recursão para quando n = 0

2. **Caso recursivo**

   Define quando o caso pode ser resolvido usando casos menores.

   As chamadas recursivas devem conduzir ao caso base, senão haverá loop infinito!

   No exemplo, há chamada recursiva quando n > 0

Fatorial

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n.(n-1)! & \text{se } n > 1 \end{cases}$$

```c
int fat(int n) {
  if (n == 0) {
    return 1;
  }
  return n * fat(n-1);
}
```

Exemplo: calcular o fatorial

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n.(n-1)! & \text{se } n > 1 \end{cases}$$

```
1  int fat(int n) {
2    if (n == 0) {
3      return 1;
4    }
5    return n * fat(n-1);
6  }
```

Cálculo de fatorial de 5: **`fat(5)`**

```
fat(5) = 5.fat(4) = 5.24 = 120
fat(4) = 4.fat(3) = 4.6 = 24
fat(3) = 3.fat(2) = 3.2 = 6
fat(2) = 2.fat(1) = 2.1 = 2
fat(1) = 1.fat(0) = 1.1 = 1
fat(0) = 1
```

CASO BASE!!!

Exemplo: calcular o nésimo valor de Fibonacci

$$f(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ f(n-1) + f(n-2) & \text{se } n > 1 \end{cases}$$

```
1  int fib(int n) {
2    if (n <= 1) {
3      return n;
4    }
5    return fib(n-1) + fib(n-2);
6  }
```

Cálculo de fibonacci de 6: **`fib(6)`**

```
fib(6) = fib(5) + fib(4) = 5 + 3 = 8
fib(5) = fib(4) + fib(3) = 3 + 2 = 5
fib(4) = fib(3) + fib(2) = 2 + 1 = 3
fib(3) = fib(2) + fib(1) = 1 + 1 = 2
fib(2) = fib(1) + fib(0) = 1 + 0 = 1
fib(1) = 1 CASO BASE!!!
fib(0) = 0 CASO BASE!!!
```

Solução recursiva vs. solução iterativa

Alguns problemas são **naturalmente recursivos**.
Nesses casos, uma solução recursiva é mais simples de implementar.
Exemplo: Série de Fibonacci

versão iterativa

```
1  int fib(int n) {
2    int f = 0, aa = 0, a = 1;
3    for (int i = 0; i < n; i++) {
4      f = a + aa;
5      aa = a;
6      a = f;
7    }
8    return f;
9  }
```

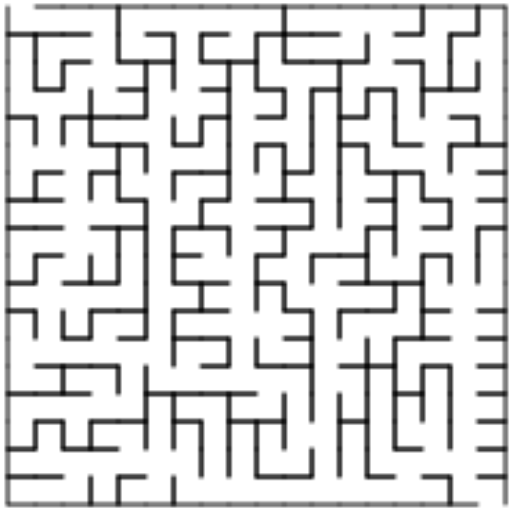
versão recursiva

```
1  int fib(int n) {
2    if (n <= 1) {
3      return n;
4    }
5    return fib(n-1) + fib(n-2);
6  }
```

Explorando as possibilidades

A recursão pode ser usada como procedimento para explorar as possibilidades de um problema. Exemplo: responder se há caminho entre duas posições quaisquer.

```
algoritmo tem_caminho(posição O, posição D)
  se O = D então
    retorna verdadeiro
  senão
    marca O como percorrido (“já passei por aqui”)
    para cada posição P adjacente a O nas quatro direções
      se 1. não há parede entre P e O
         2. P ainda não foi marcado como percorrido
         3. tem_caminho(P, saída) então
        retorna verdadeiro
    retorna falso
```

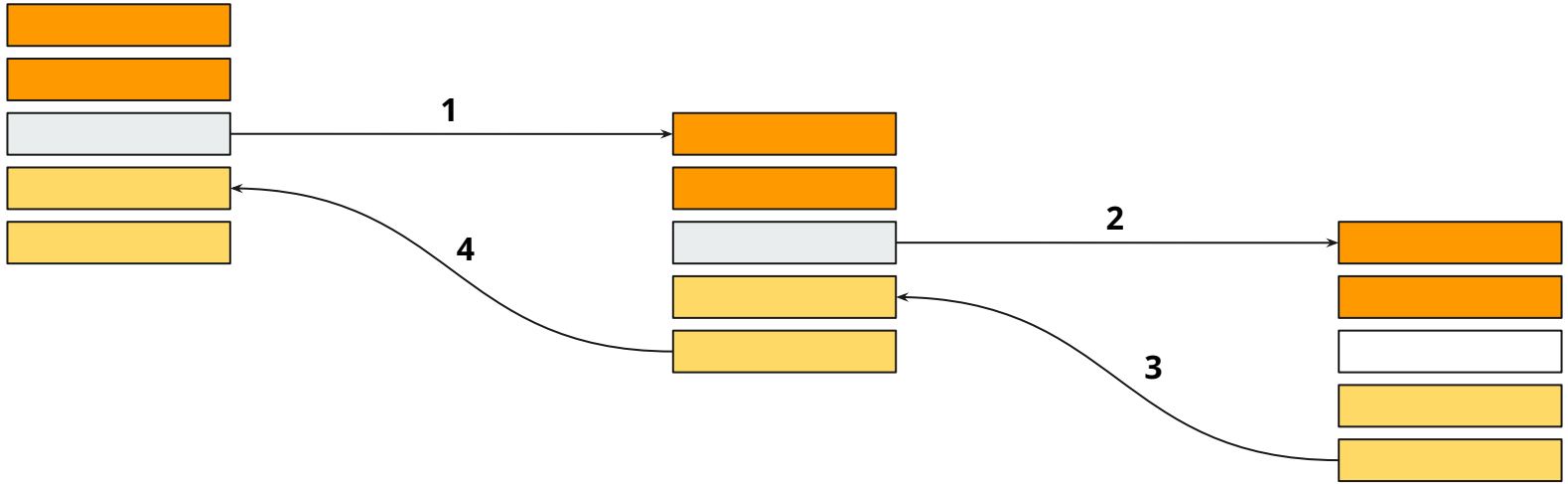


Processamento da recursão

Pode haver processamento:

- Antes da chamada recursiva
- Depois da chamada recursiva

Dependendo de quando a chamada é feita o resultado é diferente



Exemplo de processamento pré e pós-chamada

Problema

Implemente um programa que converte um número de base decimal para base binária. Exs: 15 → 1111, 10 → 1010

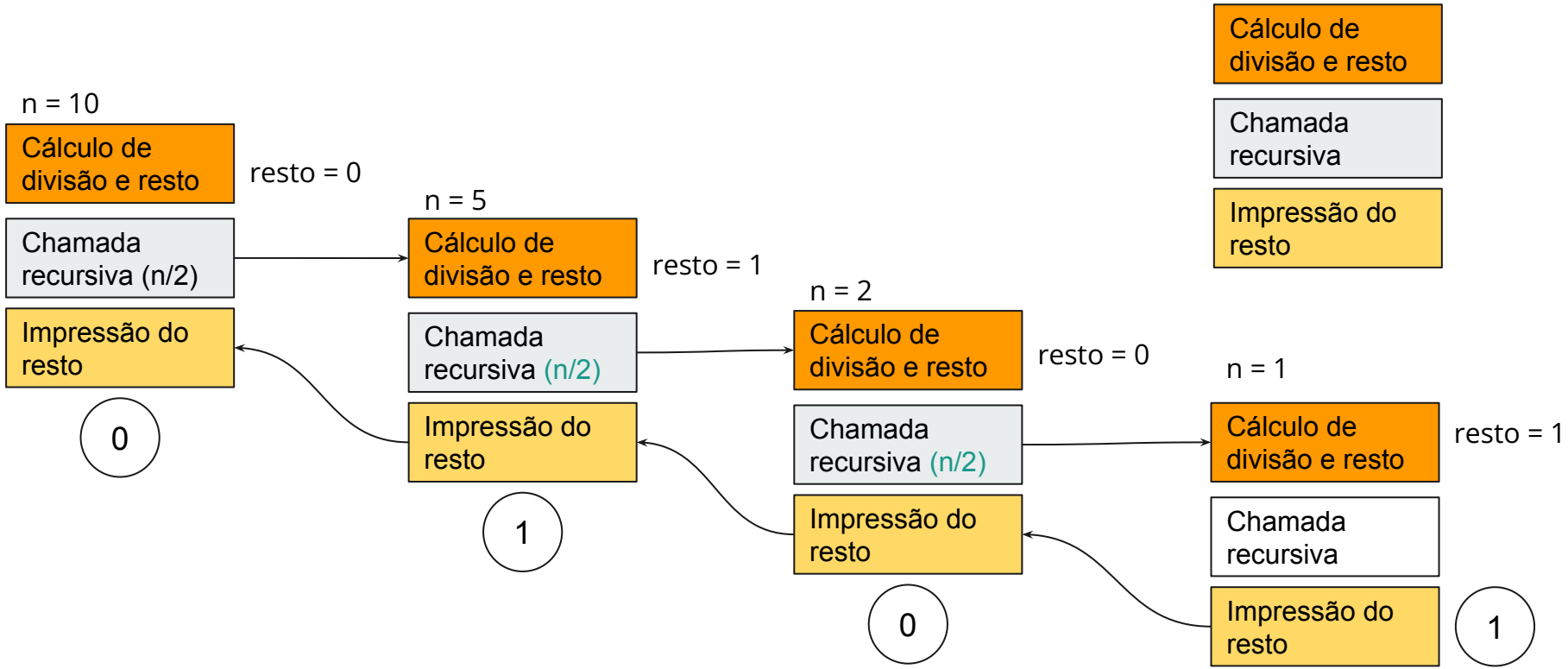
Ideia de solução: dividir o valor por 2, pegar o resto e aplicar a mesma solução para a parte inteira do resultado da divisão

Ex: para valor 10

1. 10 / 2 = 5 (resta **0**)
2. 5 / 2 = 2 (resta **1**)
3. 2 / 2 = 1 (resta **0**)
4. 1 / 2 = 0 (resta **1**)

Resultado é a **sequência de restos na ordem invertida**: 1010

Exemplo de processamento pré e pós-chamada

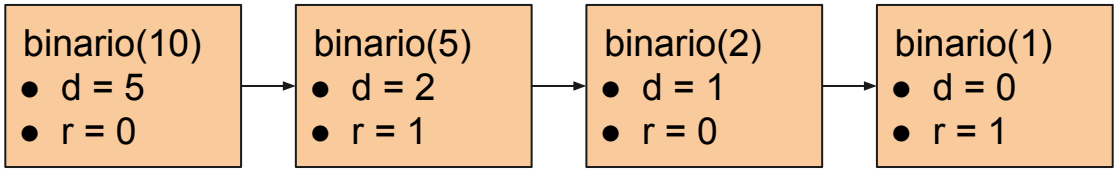


Solução em código

?

As chamadas são “empilhadas” na memória

Para cada chamada um novo escopo é criado



Recursão indireta (ou múltipla)

Um recursão pode ser realizada usando mais de uma função

Por exemplo:

- Um número natural é par se:
  - Ele for 0 **ou**
  - Seu antecessor for ímpar;
- Um número natural é ímpar se:
  - Ele não for 0 **e**
  - Seu antecessor for par.

```
1  bool par(int n) {
2    return (n == 0 or impar(n-1));
3  }
4
5  bool impar(int n) {
6    return (n != 0 and par(n-1));
7  }
```

Práticas

Escreva uma função recursiva para:

1. Calcular a potência de um número inteiro a partir da seguinte relação:

$$b^n = b.b^{n-1}$$

```
int potencia(int base, int expoente) {
  // ...
}
```

Práticas

Escreva funções recursivas para:

1. Achar o maior valor de uma sequência não nula de inteiros.

```
int maior(int tam, int sequencia[]) {
  if (tam == 1) {
    return sequencia[0];
  }
  int max = maior(tam-1, sequencia);
  return max > sequencia[tam-1] ? max : sequencia[tam-];
}
```

Práticas

Escreva uma função recursiva para:

1. Calcular o somatório de um array de inteiros a partir da seguinte relação:

$$\sum_{i=1}^{n} v_i = v_1 + \sum_{i=2}^{n}$$

```
int somatorio(int tam, int sequencia[]) {
  // ...
}
```

Práticas

Escreva funções recursivas para:

1. Verificar se um dado número se encontra entre os números de uma sequência ordenada de números inteiros.
2. Contar quantos caracteres tem uma C string

Problema - Maze runner

Dado um labirinto, escreva um programa para verificar se há saída a partir de uma posição inicial. O labirinto é representado por uma matriz de 0 e 1, onde 0 indica uma posição livre (onde é possível se mover) e 1 indica uma barreira, e só é possível mover nas 4 direções (cima, baixo, direita e esquerda).

Entrada

A primeira linha da entrada contém dois valores inteiros L e C representando as dimensões do labirinto (nº de linhas e nº de colunas, respectivamente). As L linhas seguintes contém C valores 0 ou 1, que indicam cada posição do labirinto, se encontra-se livre (0) ou se há uma barreira (1). As linhas seguintes contêm as coordenadas na posição inicial e a saída do labirinto.

Saída

Seu programa deve imprimir “sim” se houver saída ou “não” caso contrário.

Problema - Maze runner

Exemplos de entrada e saída

Entrada	Saída
4 4 1 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 3 0	sim

Entrada	Saída
6 6 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 5 0	não

Problema - Flood it

Flood it (https://unixpapa.com/floodit/) é um jogo em que precisamos preencher passo-a-passo uma matriz com uma única cor. A cada passo, escolhemos uma cor para preencher a partir da célula superior esquerda. Escreva um programa que realiza um passo do jogo. As cores na matriz do jogo são representadas por valores de 0 a 5.

Entrada
A primeira linha contém a dimensão N de uma matriz quadrada. As N linhas seguintes contêm N valores entre 0 e 5, que indicam as cores de cada célula da matriz. A linha seguinte contém um valor entre 0 e 5 que irá preencher a matriz a partir da célula superior esquerda.

Saída
O programa deve imprimir o resultado da matriz do jogo após o preenchimento da cor passada.

Problema - Flood it

Exemplos de entrada e saída

Entrada	Saída
4 1 1 1 0 1 5 1 4 1 1 2 1 4 5 1 3 4	4 4 4 0 4 5 4 4 4 4 2 1 4 5 1 3

Entrada	Saída
4 4 4 4 3 4 5 4 4 4 4 4 4 4 5 4 3 5	5 5 5 3 5 5 5 5 5 5 5 5 5 5 5 3

Entrada	Saída
4 5 5 5 3 5 5 5 5 5 5 5 5 5 5 5 3 3	3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3