Algoritmos - Aula 2

Fernando Raposo

Vamos ver

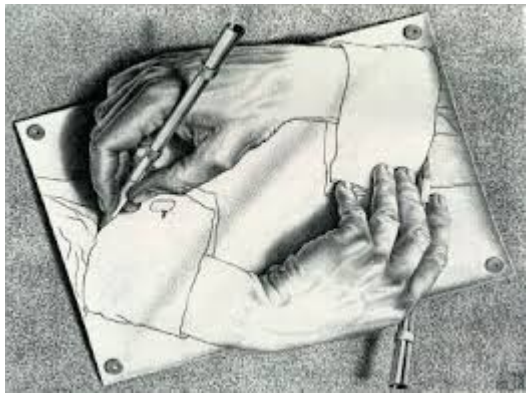
- Iteração (loops)
  - for, while...
- Recursão
  - Funções recursivas
  - Fatorial
  - Torre de Hanói
- Iteração x Recursão
- Estouro de Memória
- Estratégia dividir para conquistar (divide and conquer)

Iteração x Recursão

- Um programa é dito **iterativo** quando ele possui um laço (loop) ou mesmo uma repetição;

FOR, WHILE, GOTO <- bad programming

- Um programa é dito **recursivo** quando ele faz uma chamada a si mesmo;



Maurits Cornelis Escher

Inception - Sonhar dentro de um sonho

Iteração

- Já vimos loops convencionais (*For, While…*)
- Outra forma de se fazer um loop é utilizando **programação funcional**

Java

```
List<String> lista = new ArrayList<String>();
lista.add("Um");
lista.add("Dois");
lista.add("Três");
lista.stream().forEach(item ->
                        System.out.println(item)

                       );
```

Javascript

```
const lista = ["Um", "Dois", "Três"];
lista.forEach(item =>
                console.log(item)
               );
```

Recursão

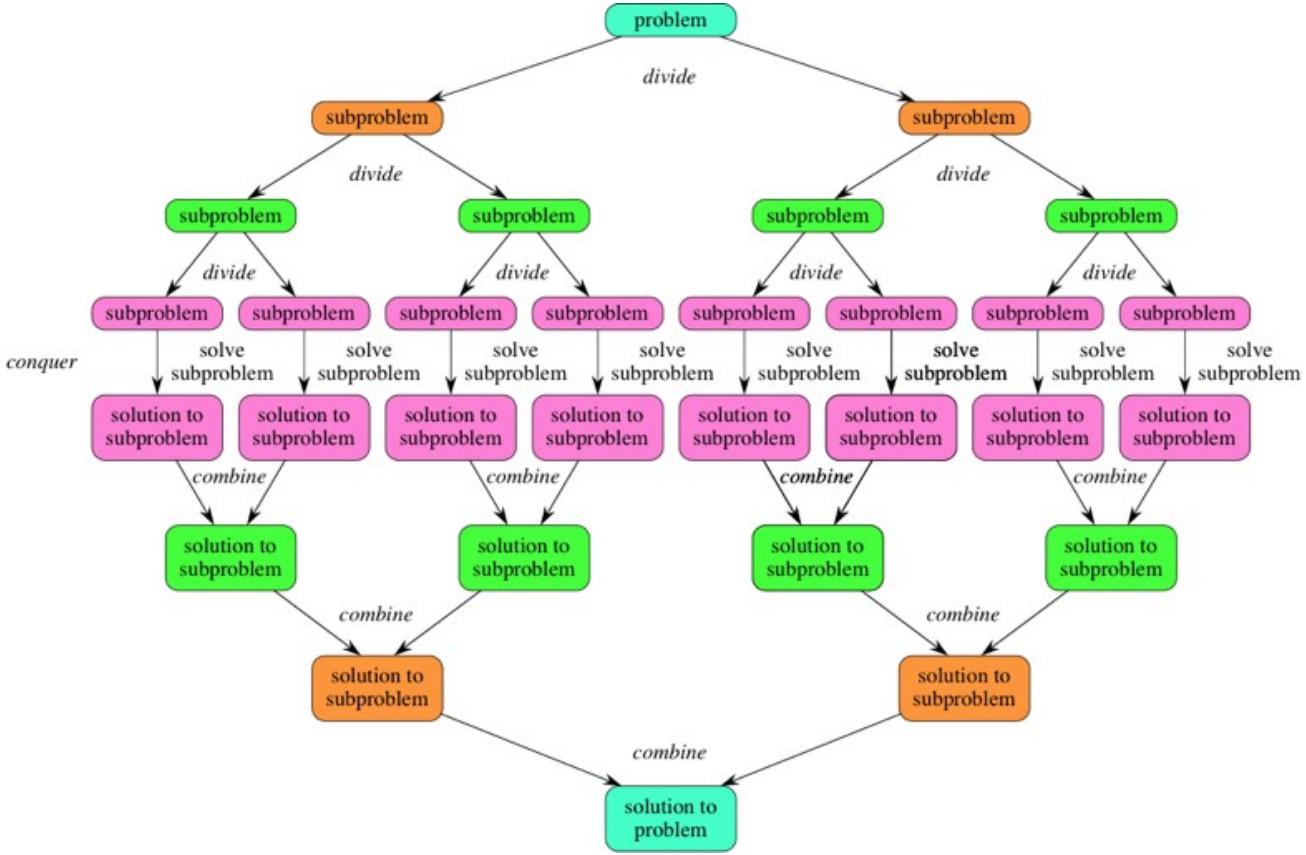
- Ideia básica: Para resolver um **problema**, comece resolvendo um **subproblema**(s) deste problema. Use a(s) solução do(s) subproblema(s) para resolver o problema original.



- Problema: Lavar os pratos sujos
  - Subproblema1: Ensaboar os pratos sujos;
  - Subproblema2: Enxaguar os pratos ensaboados;

Dividir para Conquistar

Dividir para conquistar é um paradigma algorítmico.



Recursão

- Regras da Recursão:
  - Cada chamada recursiva deve ser a uma instância menor do problema;
  - As chamadas recursivas necessitam chegar a um caso base, a qual possui uma resposta a ser retornada sem a necessidade de mais chamadas recursivas.
- Exemplo clássico: **Fatorial**

Fatorial 5: 5! = 5*4!
Fatorial 4: 4! = 4*3!
Fatorial 3: 3! = 3*2!
Fatorial 2: 2! = 2*1!
Fatorial 1: 1! = 1  Caso base!

Recursão

Então podemos deduzir:

n! = n*fat(n-1)

onde fat() é uma chamada recursiva.

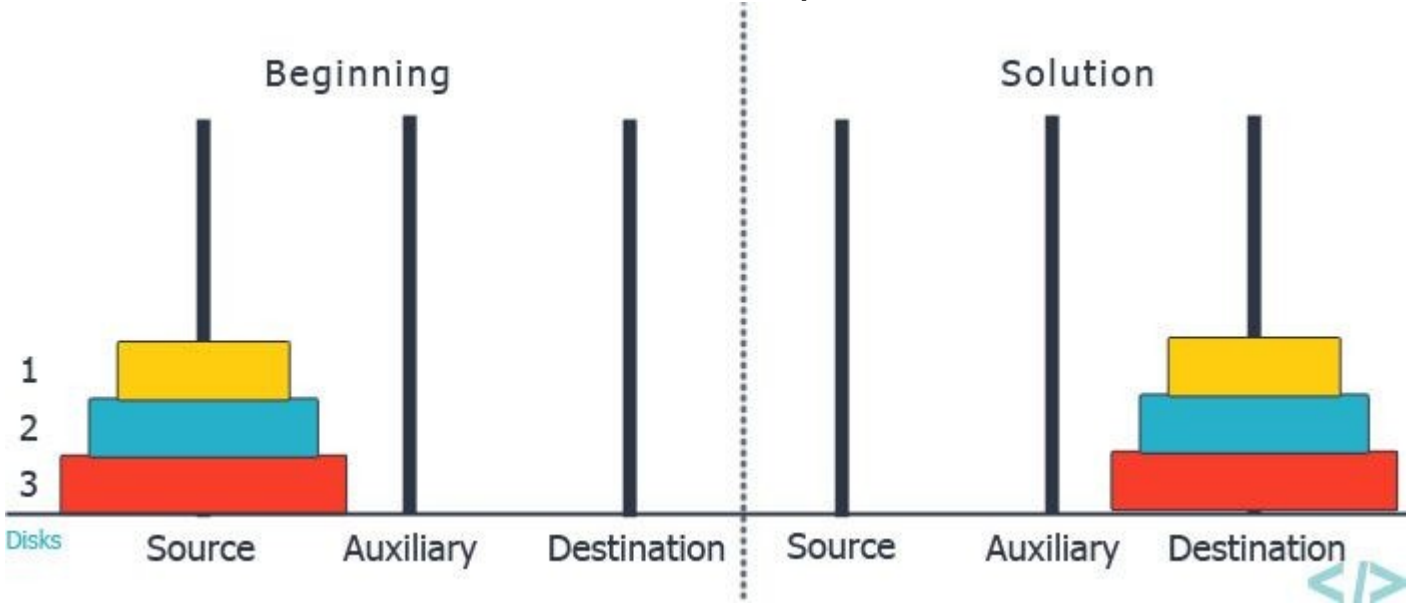
Parte Prática

Vamos implementar em Javascript uma função que calcule o fatorial, primeiro utilizando iterações, e depois utilizando recursividade.

Para testar, vamos calcular 5! e 15!

Recursão

- Alguém já jogou a Torre de Hanói?
- Como levar os discos de ***Source*** para ***Destination***, movendo um disco por vez e nunca deixando um disco maior por cima de um menor.



Recursão

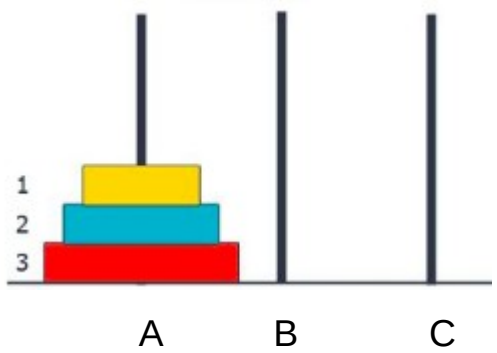
- Entender a lógica por trás de **chamadas recursivas** como a Torre de Hanói e Cálculo de Fatorial nos ajudará a entender algoritmos de ordenação mais a frente.

Parte Prática

Executar o algoritmo numa folha de papel em *pseudo-código*.

Ideia Básica: Origem = A, Destino = C

- Se temos ***n*** discos:
  - Mova os **n-1** primeiros discos para **B** recursivamente;
  - Mova o maior de **A** para **C**;
  - Mova os **n-1** primeiros discos de **B** para **C** recursivame

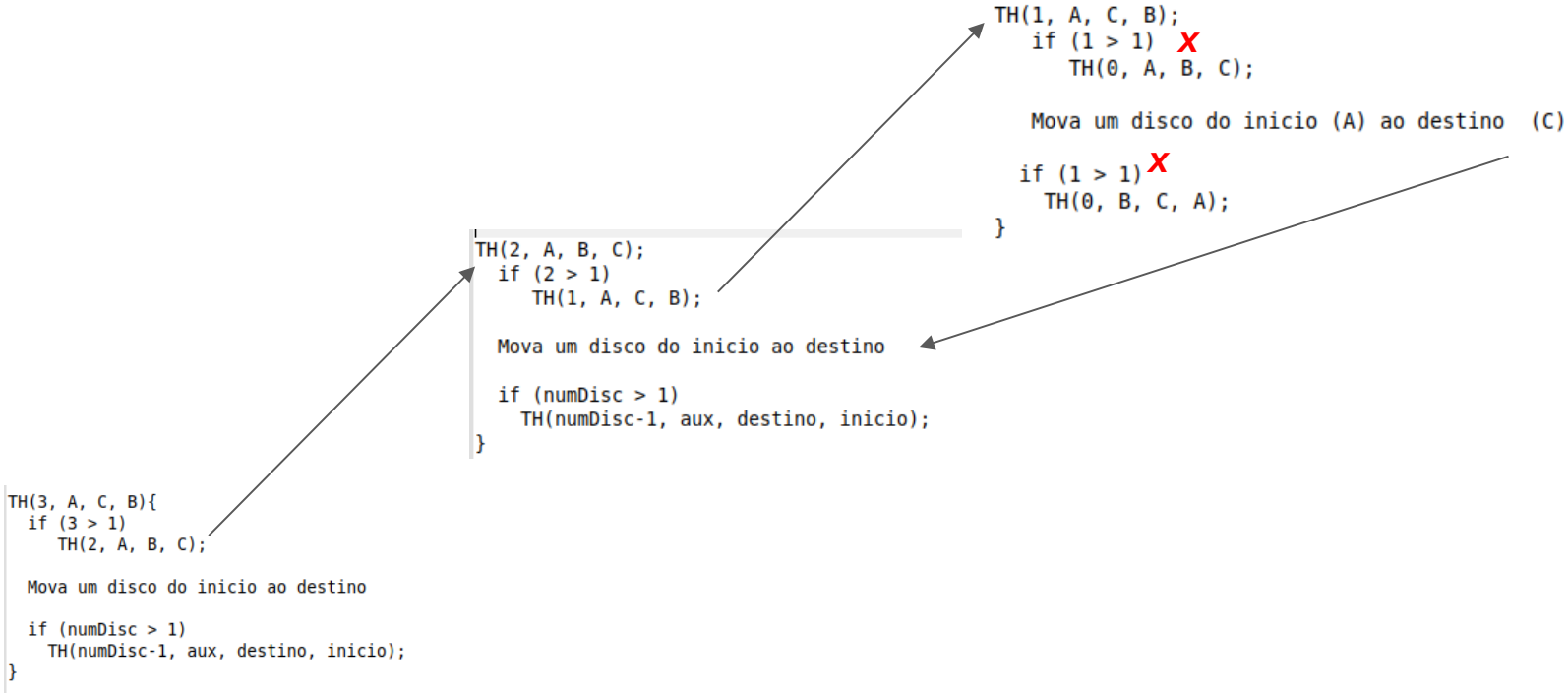


Ou…

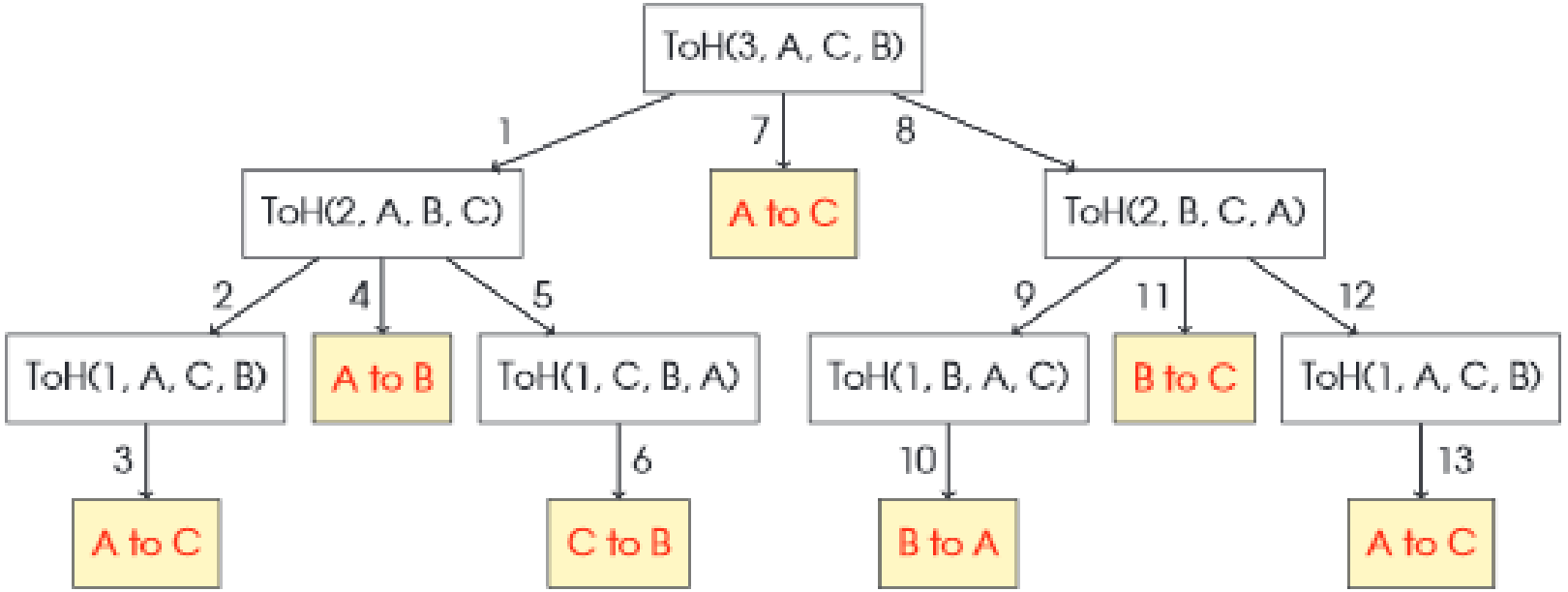
```
TH(numDisc, inicio, destino, aux){
  if (numDisc > 1)
     TH(numDisc-1, inicio, aux, destino);

  Mova um disco do inicio ao destino

  if (numDisc > 1)
     TH(numDisc-1, aux, destino, inicio);

}
```


```
TH(numDisc, inicio, destino, aux){
  if (numDisc > 1)
     TH(numDisc-1, inicio, aux, destino);

  Mova um disco do inicio ao destino

  if (numDisc > 1)
    TH(numDisc-1, aux, destino, inicio);
}
```

Pilha de Invocação



TH(numDisc, inicio, destino, aux)



Mova um disco do inicio ao destino

A to C; A to B; C to B;
A to C;
B to A; B to C; A to C;

Recursão x Iteração: Comparação

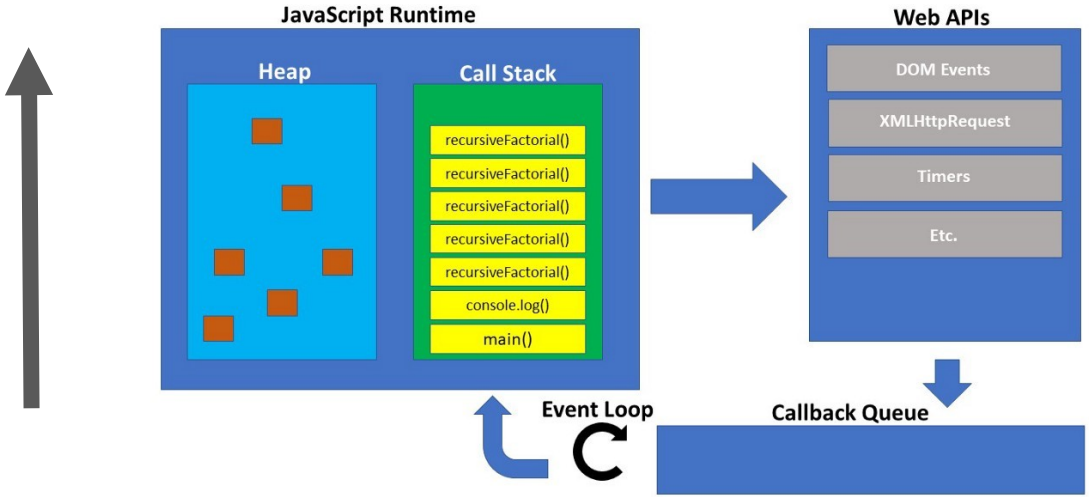
- Vimos que para resolver um problema podemos utilizar iteração ou recursão;
    - Qual é mais extenso (o código)?
    - Qual é mais rápido (de rodar)?

Recursão x Iteração: Fatos

1. Códigos recursivos tendem a ser menores que códigos iterativos;
2. Códigos iterativos tendem a consumir menos tempo de processamento que códigos recursivos;
3. Códigos iterativos levam a estouro de memória (CPU Crash) por má implementação da iteração;
4. Códigos recursivos podem levar a estouro de memória devido a um erro na escolha do caso base;

Estouro de Memória?

- Existem duas partes da memória a Heap e a Stack;
- A memória Heap geralmente é maior que a Stack;
- A cada chamada recursiva aumentamos a pilha da Stack, que pode crescer indefinidamente até estourar se não encontrar o caso base.



Resumo

Aspecto	Recursão	Iteração
Definição	Função que chama a si mesma	Conjunto de instruções repetidas n vezes
Aplicação	Funções	laços
Condição de parada	Caso base	Quando a condição do loop deixa de ser satisfeita
Uso	Quando o código tem que ser pequeno e a complexidade não é problema	Quando o código pode ser maior e existem recursos de memória limitados
Complexidade	Alta (muitas vezes exponencial)	Relativamente baixa (logaritmo polinominal)


Thank You!