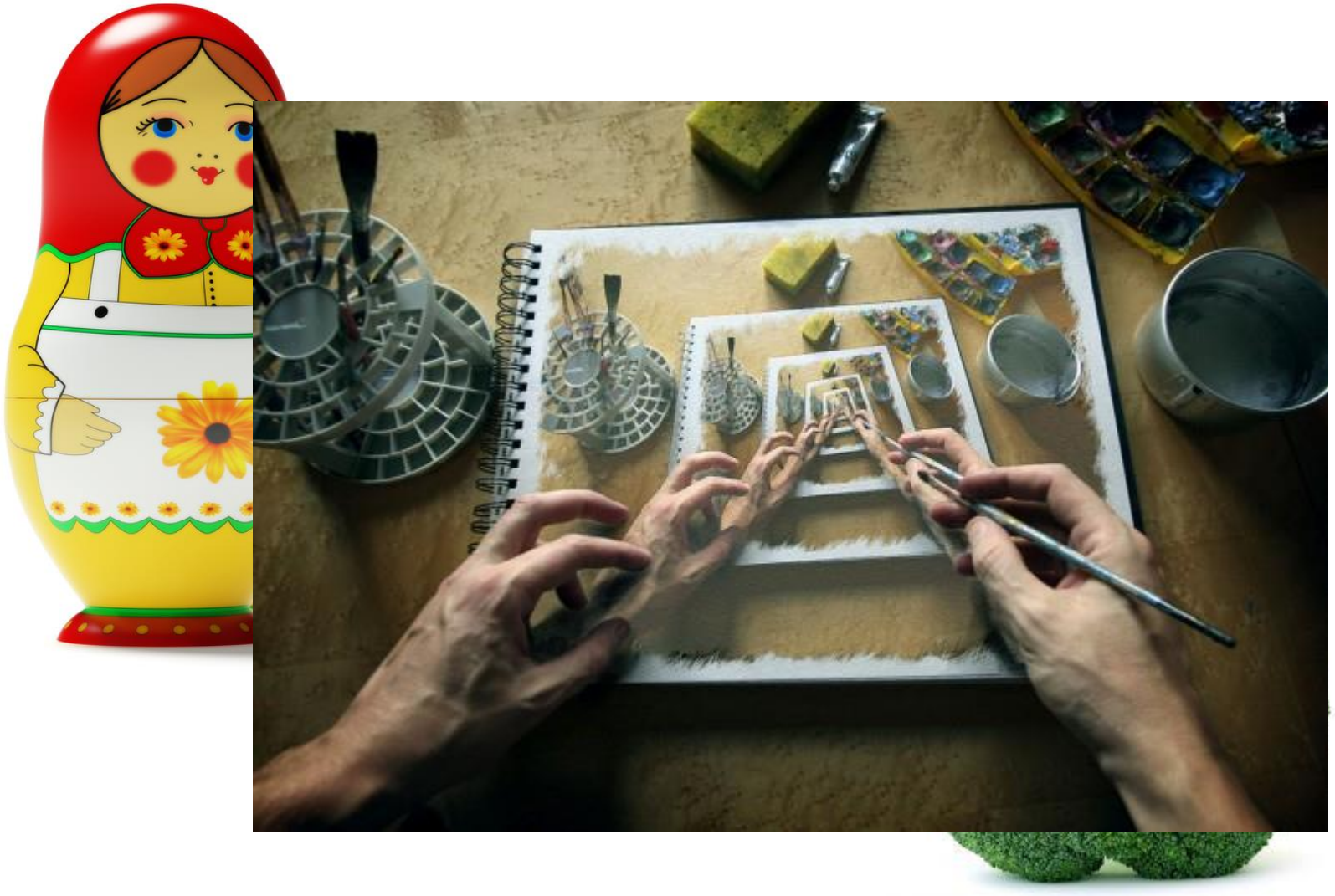
NOVA SCHOOL OF SCIENCE & TECHNOLOGY
NOVALINCS
LABORATORY FOR COMPUTER SCIENCE AND INFORMATICS

ALGORITMOS E ESTRUTURAS DE DADOS 2023/2024
RECURSIVIDADE

Armanda Rodrigues

4 de outubro de 2023

Introduzir a ideia de recursividade…



Tokyo Skytree – A mais alta torre do mundo …

Subir uma torre com 2523 degraus

- Temos de subir ao topo de uma torre com 2523 degraus
- Como avançar com esta tarefa?
- Uma tarefa comprida como esta começa com um passo: Saber como reduzir a dimensão do problema
- Queremos subir ao topo da torre (2523 degraus). Se subirmos um degrau, o problema torna-se um pouco menor
- Claro que agora temos de reduzir a dimensão do novo problema, subindo um novo degrau

- Ao fim de algum tempo, a dimensão do problema estará de tal forma reduzida, que poderemos terminar a tarefa subindo o último degrau.

Resolver um problema de forma recursiva

- Nós sabemos como subir ao cimo da torre. Podemos descrever o processo como “subir degraus até chegarmos ao cimo”.

- Vamos descrever a solução com mais detalhe:
  - Se falta subir um degrau, subir o degrau e terminou
  - Se falta subir N degraus (sendo N <= 2523), dividir a tarefa remanescente em duas partes:
    - Subir um degrau
    - Subir N-1 degraus (equivalente **a subir ao topo de uma torre com N-1 degraus**)
- Esta é a forma recursiva de descrever a solução do problema

Os dois componentes da Recursividade

- A recursividade tem duas partes:
  - Se o problema é simples (ou fácil, ou básico), resolve-se imediatamente
  - Se o problema não pode ser resolvido imediamente, então divida-se em problemas de dimensão menor e depois:
    - Aplique-se o mesmo procedimento aos problemas menores

- A estratégia é encontrar uma acção fácil de aplicar que divida o problema em problemas menores
  - Alguns destes problemas menores podem ser resolvidos imediatamente – são básicos
  - Outros irão ser, também eles, divididos em problemas de menor dimensão

Dividir uma linha em 16 segmentos iguais…

- Pretendemos dividir uma linha em 16 partes iguais
- Qual é a primeira coisa a fazer ?

- Temos agora 2 partes (segmentos), cada um com metade do comprimento original
- O problema converteu-se em “dividir cada um dos segmentos em 8 partes iguais”
- Como resolver os problemas menores ?

Aplicando-lhes o mesmo processo

Dividir uma linha em 16 segmentos iguais…

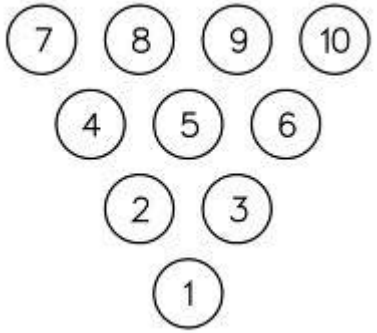
- Dividimos cada um dos segmentos ao meio (resultam 4 partes iguais)
- Voltamos a aplicar o processo aos 4 segmentos

- Ficamos com 8, e voltamos a aplicar

- E ficamos com 16…

Números Triangulares

- Se adicionássemos uma linha ao arranjo, quantos pinos teria?
- Em 4 linhas, temos 10 pinos
- Quantos pinos teríamos em 5 linhas?
- Ao número total de pinos num arranjo triangular chamamos um **Número Triangular**
- O número total de pinos num arranjo triangular de 5 linhas é a soma de 5 com o número de pinos num arranjo triangular de 4 linhas




```
triangle(5) = 5 + triangle(4)
```

Números Triangulares

- Quanto é `triangle(10)`?

Número Total de pinos em 10 linhas = Número de pinos na 10ª linha + Número Total de pinos em 9 linhas

Ou seja, podemos escrever a seguinte fórmula:

```
triangle(N) = N + triangle(N-1)
```

Números Triangulares

- Quanto é `triangle(10)`?

Número Total de pinos em 10 linhas = Número de pinos na 10ª linha +
Número Total de pinos em 9 linhas

Números Triangulares

- Quanto é `triangle(10)`?

Número Total de pinos em 10 linhas = Número de pinos na 10ª linha +
(Número de pinos na 9ª linha +
Número Total de pinos em 8 linhas)

Números Triangulares

- Quanto é `triangle(10)`?

Número Total de pinos em 10 linhas = Número de pinos na 10ª linha +
Número de pinos na 9ª linha +
(Número de pinos na 8ª linha +
Número Total de pinos em 7 linhas)

Números Triangulares

- Quanto é `triangle(10)`?

Número Total de pinos em 10 linhas = Número de pinos na 10ª linha +
Número de pinos na 9ª linha +
Número de pinos na 8ª linha +
… +
… +
Número de pinos na 1ª linha

Número de pinos na 1ª linha = 1

Isto é o que chamamos o **Caso Base**

Caso Base

Número de pinos na 1ª linha = 1

- O caso base é um problema que pode ser resolvido imediatamente

- As duas partes da recursividade:
  - Se o problema é simples, resolve-se imediatamente – Este é o Caso Base
  - Se o problema não pode ser resolvido imediamente, divida-se em problemas de dimensão menor e depois:
    - Aplique-se o mesmo procedimento aos problemas menores

```
triangle(1) = 1
triangle(N) = N + triangle(N-1)
```

Números Triangulares em Java

```
public static int triangle( int n )
{
  if ( n == 1 )
    return 1;
  else
    return n + triangle( n-1 );
}
```

Caso Base

Problema de dimensão menor

Visão estática da recursividade

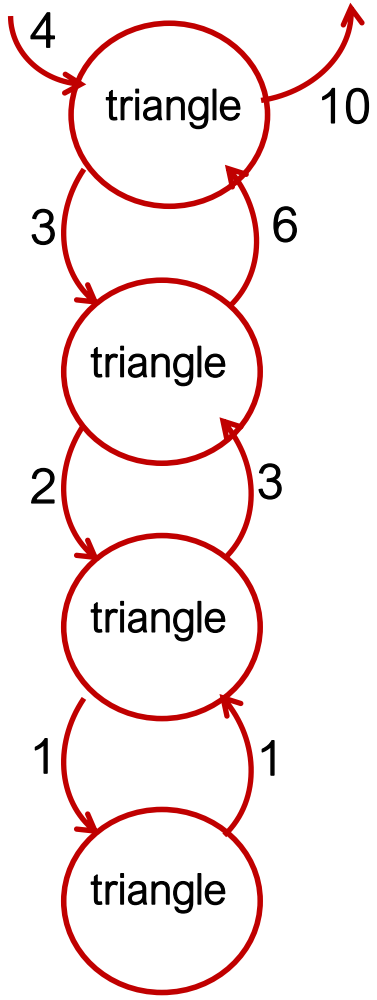
```
triangle(1) = 1
triangle(N) = N + triangle(N-1)
```

Os símbolos da definição são reorganizados e a sintaxe necessária é adicionada

```
public static int triangle( int n )
{
  if ( n == 1 )
    return 1;
  else
    return n + triangle( n-1 );
}
```

Visão Dinâmica da recursividade

```
public static int triangle( int n )
{
  if ( n == 1 )
    return 1;
  else
    return n + triangle( n-1 );
}
```



Vamos correr `triangle(4)`:

Que vai chamar `triangle(3)`

Que vai chamar `triangle(2)`

Que vai chamar `triangle(1)`

`triangle(1)` devolve 1 e termina

`triangle(2)` devolve 3 e termina

`triangle(3)` devolve 6 e termina

`triangle(4)` devolve 10 e termina

Erros comuns

```
static int triangle( int n ) {
 return n + triangle( n-1 );
}
```

Qual o problema com esta versão de triangle ?

1. Na perspetiva estática – não tem caso base
2. Na perspetiva dinâmica – o método faz sempre uma chamada recursiva, a cadeia de ativações está sempre a crescer

Vantagens da recursividade

- A recursividade é útil porque existem problemas que são naturalmente recursivos
- Neste caso, o que temos que fazer é transformar a sua descrição em código Java
- É sempre possível transformar um método recursivo num iterativo e vice-versa. Mas pode não ser natural ou fácil.
- Geralmente, o método recursivo ocupa mais memória do que o iterativo, devido a necessidade de guardar todas as ativações do método
- A utilização de recursividade não traz mais poder ao Java
- A maior parte das linguagens modernas permitem recursividade e iteratividade porque é conveniente.

Função Soma de um vetor recursiva

```
public static int arraySum(int[] n){
   return arraySum(n, 0);
}
```

Método público que permite fazer a 1ª chamada recursiva, definindo o problema completo

```
protected static int arraySum(int n[], int pos){

   if (pos==n.length)
      return 0;
   else return n[pos]+arraySum(n,++pos);
}
```

Caso Base

Problema menor: É a forma como é feita a chamada recursiva que permite diminuir o problema

Pesquisa Binária Recursiva

- Pensemos num vetor ordenado de números inteiros
- Podemos então executar sobre o vetor um algoritmo de pesquisa binária

- Vamos ver uma versão recursiva deste algoritmo
- A solução é apresentada como um método estático que efetua a pesquisa binária de um valor sobre um vetor de inteiros já ordenado, que é passado como parâmetro

Pesquisa Binária recursiva em vetor de inteiros

```
public static int binarySearch(int[] v, int aim){
   return binarySearch(v, aim, 0, v.length-1);
}

protected static int binarySearch(int[] v, int aim,
                                  int low, int high){
   int mid;
   int current;

   if ( low > high )
     return -1;
   else {
         mid = ( low + high ) / 2;
         current = v[mid];
         if ( aim == current )
            return mid;
         else if ( aim < current )
                return binarySearch(v, aim , low, mid-1);
              else return binarySearch(v, aim, mid+1, high);
        }
}
```

Método público que faz a primeira chamada ao método recursivo

Caso Base sem sucesso

Caso Base c/ sucesso

Cada uma destas chamadas divide o espaço de pesquisa ao meio

Função de Fibonacci – mais um exemplo

- A função de Fibonacci modela o crescimento de comunidades de coelhos
- A função Matemática é a seguinte:

$$Fibonacci(n) = \begin{cases} 0, & n = 0; \\ 1, & n = 1; \\ Fibonacci(n-1) + Fibonacci(n-2), & n \geq 2. \end{cases}$$

Fibonacci – Método Recursivo

```
//Requires: n >= 0
public static long fibonacciRec( int n ){

   if ( n == 0 )
      return 0;
   else if ( n == 1 )
         return 1;
         else
             return fibonacciRec(n - 1) + fibonacciRec(n - 2);
}
```

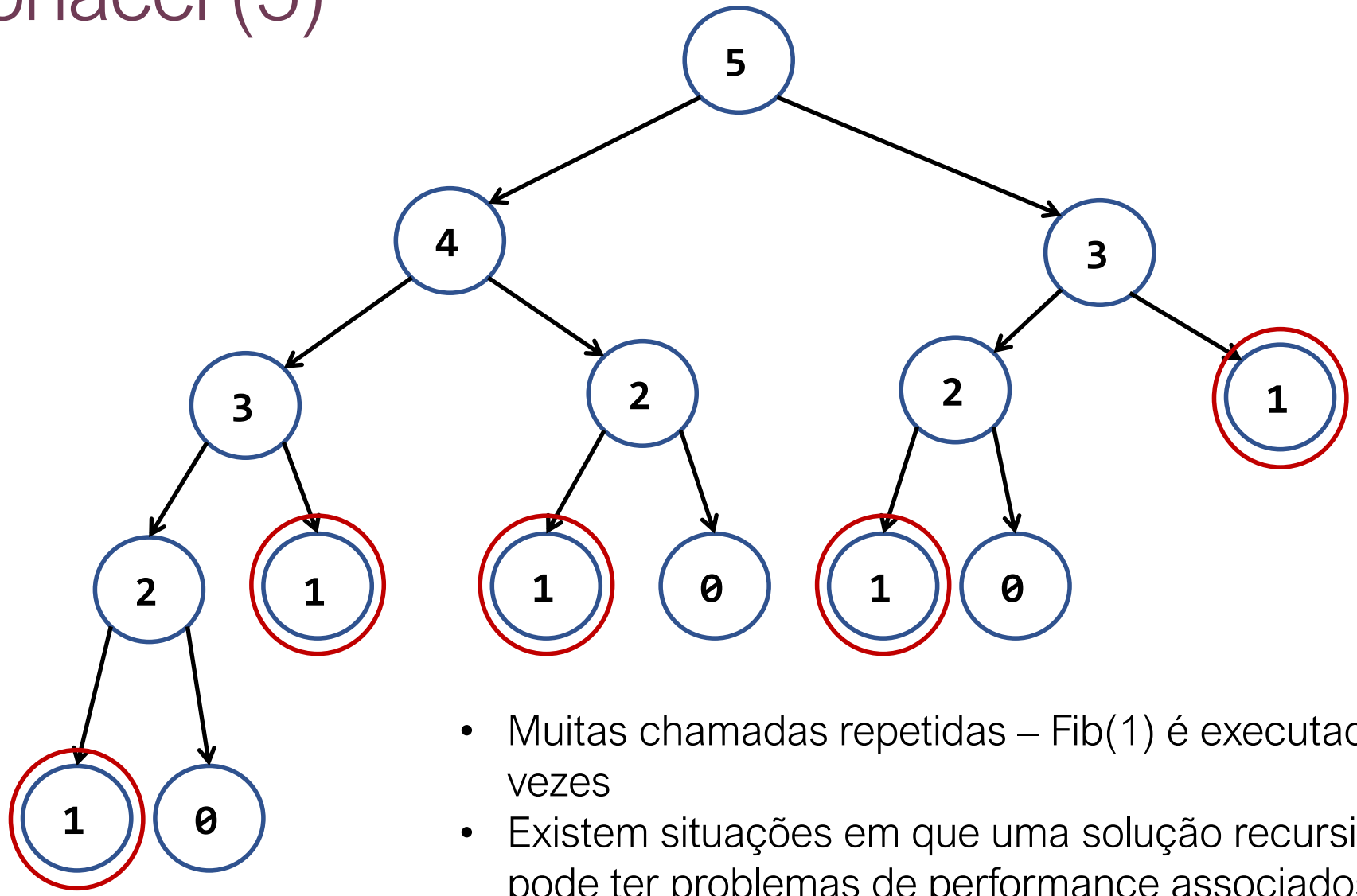
Caso Base

Caso Base

Problemas de menor dimensão

Qual o problema com este método ?

Fibonacci (5)



- Muitas chamadas repetidas – Fib(1) é executado 5 vezes
- Existem situações em que uma solução recursiva pode ter problemas de performance associados