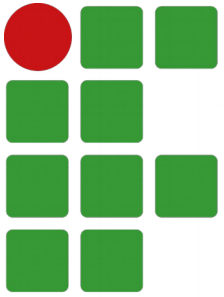
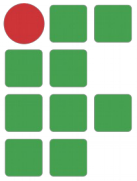


INSTITUTO FEDERAL
Ceará

Recursividade

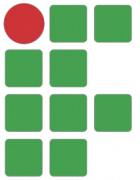
Funções



- Na linguagem C, uma função pode chamar outra função.
  - A função main() pode chamar qualquer função, seja ela da biblioteca da linguagem (como a função printf()) ou definida pelo programador (função imprime()).

```c
#include <stdio.h>

void imprime (int valor) {
    for (int i=0; i<valor; i++) {
        printf ("%d\n", i);
    }
}

int main(int argc, char **argv)
{

    int valor = 5;
    printf ("%d\n", valor);
    imprime(valor);
}
```

Encontrar mínimo / máximo

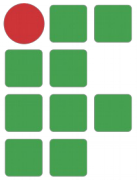


```c
#include <stdio.h>
void encontra (int v[], int tam, int *min, int *max) {
    *min = v[0]; *max = v[0];

    for (int i =0; i< tam; i++) {
        if (v[i] < *min)
            *min = v[i];
        if (v[i] > *max)
            *max = v[i];
    }
}

int main() {
    int vet[]={2,5,4,17,8,3,6,9,0,-1};
    int min, max;
    encontra(vet, 10, &min, &max);
    printf("Maior valor =%d\n", max);
    printf("Menor valor =%d\n", min);
    return 0;
}
```

Recursão

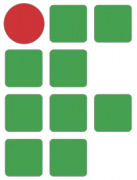
- Logo:
  - Uma função também pode chamar a si mesmo
    - **FUNÇÃO RECURSIVA**
    - Também é chamada de definição circular. Ela ocorre quando **algo é definido em termos de si mesmo.**

Função Recursiva



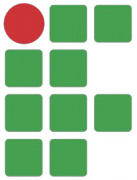
```
void func(int n) {
     printf("%d ", n);
     func(n);
     printf("* ");
}
```

Função Recursiva



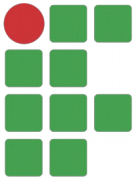
```
void func(int n) {
     printf("%d ", n);
     func(n);
     printf("* ");
}
```

Recursão



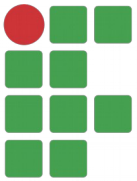
- A recursão é uma **técnica** que define um problema em termos de uma ou mais versões menores deste mesmo problema
  - Dividir para conquistar

Recursão



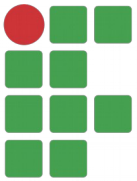
- Um exemplo clássico de função que usa recursão é o **cálculo do fatorial de um número**
  - 4! = 4 * 3 * 2 * 1
  - 3! = 3 * 2 * 1
  - 2! = 2 * 1
  - 1! = 1
  - 0! = 1

Recursão



- Um exemplo clássico de função que usa recursão é o **cálculo do fatorial de um número**
  - 4! = 4 * 3!
  - 3! = 3 * 2!
  - 2! = 2 * 1!
  - 1! = 1 * 0!
  - 0! = 1

Recursão



- Um exemplo clássico de função que usa recursão é o **cálculo do fatorial de um número**
  - 4! = 4 * 3!
  - 3! = 3 * 2!
  - 2! = 2 * 1!
  - 1! = 1 * 0!
  - 0! = 1

n! = n * (n - 1)! : fórmula geral

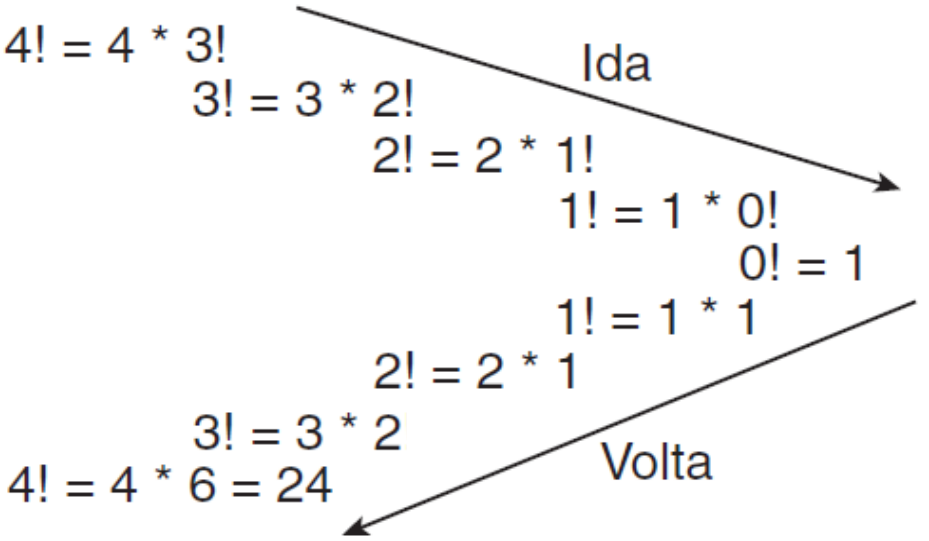
0! = 1 : caso-base

Recursão

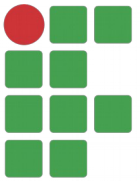
- Um exemplo clássico de função que usa recursão é o **cálculo do fatorial de um número**
  - 4! = 4 * 3!
  - 3! = 3 * 2!
  - 2! = 2 * 1!
  - 1! = 1 * 0!
  - 0! = 1

n! = n * (n - 1)! : fórmula geral

0! = 1 : caso-base



Algoritmo Fatorial



Com Recursão

```
int fatorial(int n){
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

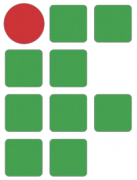
Sem Recursão

```
int fatorial (int n){
    if (n == 0)
        return 1;
    else{
        int i;
        int f = 1;
        for(i = 1; i <= n; i++)
            f = f * i;
        return f;
    }
}
```

n! = n * (n - 1)! : fórmula geral

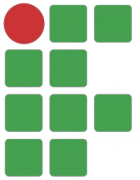
0! = 1 : caso-base

Recursão



- Em geral, algoritmos recursivos são considerados “**mais enxutos**” e “**mais elegantes**” que seus correspondentes iterativos
- Porém, algoritmos recursivos, geralmente, utilizam mais memória para resolver um problema

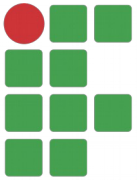
Recursão



- Muito cuidado ao escrever funções recursivas:
  - **Critério de parada**: determina quando a função deverá parar de chamar a si mesma.
  - O **parâmetro da chamada recursiva** deve ser sempre modificado, de forma que a recursão chegue a um término.

```
void func(int n) {
    printf("%d ", n);
    if (n > 0) { //condição de parada
        func(n-1); //variação do parâmetro
        printf("* ");
    }
}
```

Recursão



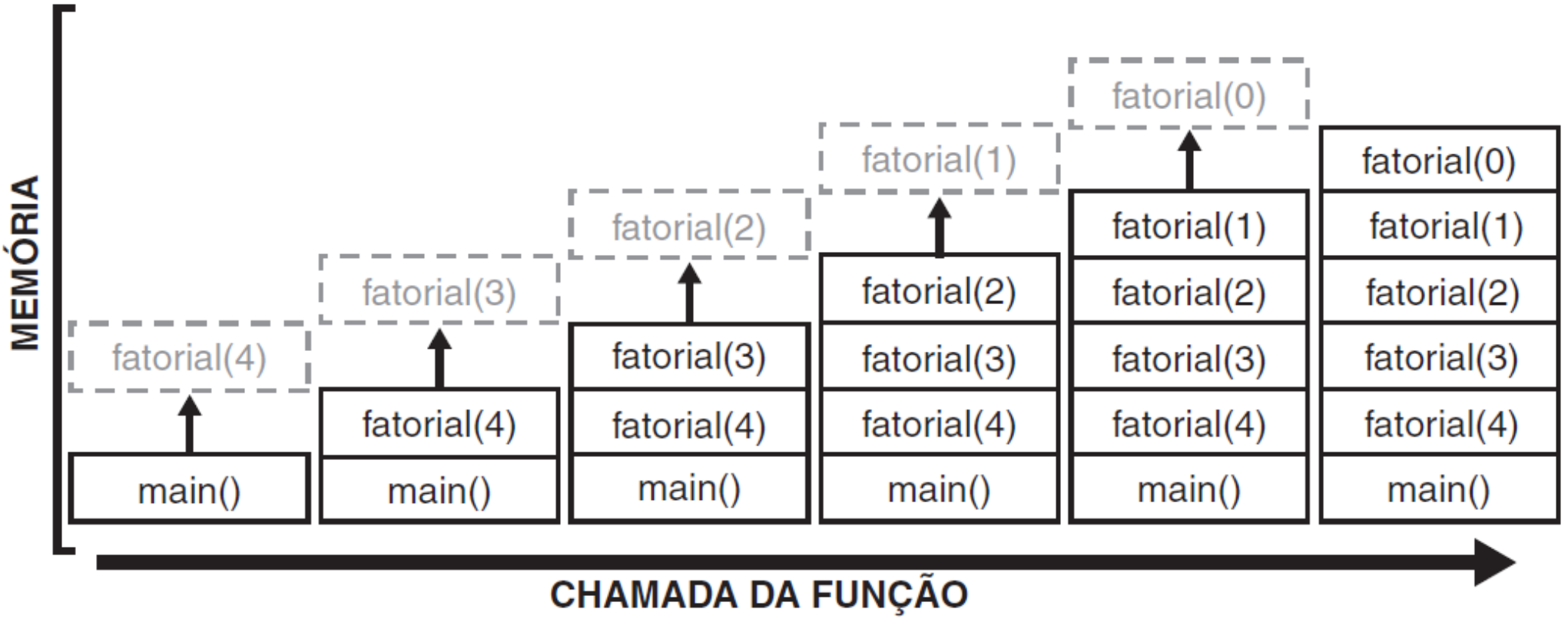
- Muito cuidado ao escrever funções recursivas:
  - **Critério de parada**: determina quando a função deverá parar de chamar a si mesma.
  - O **parâmetro da chamada recursiva** deve ser sempre modificado, de forma que a recursão chegue a um término.

```
int fatorial (int n){
    if (n == 0)//critério de parada
        return 1;
    else /*parâmetro de fatorial sempre muda*/
        return n*fatorial(n-1);
}
```

Como funciona a recursão?

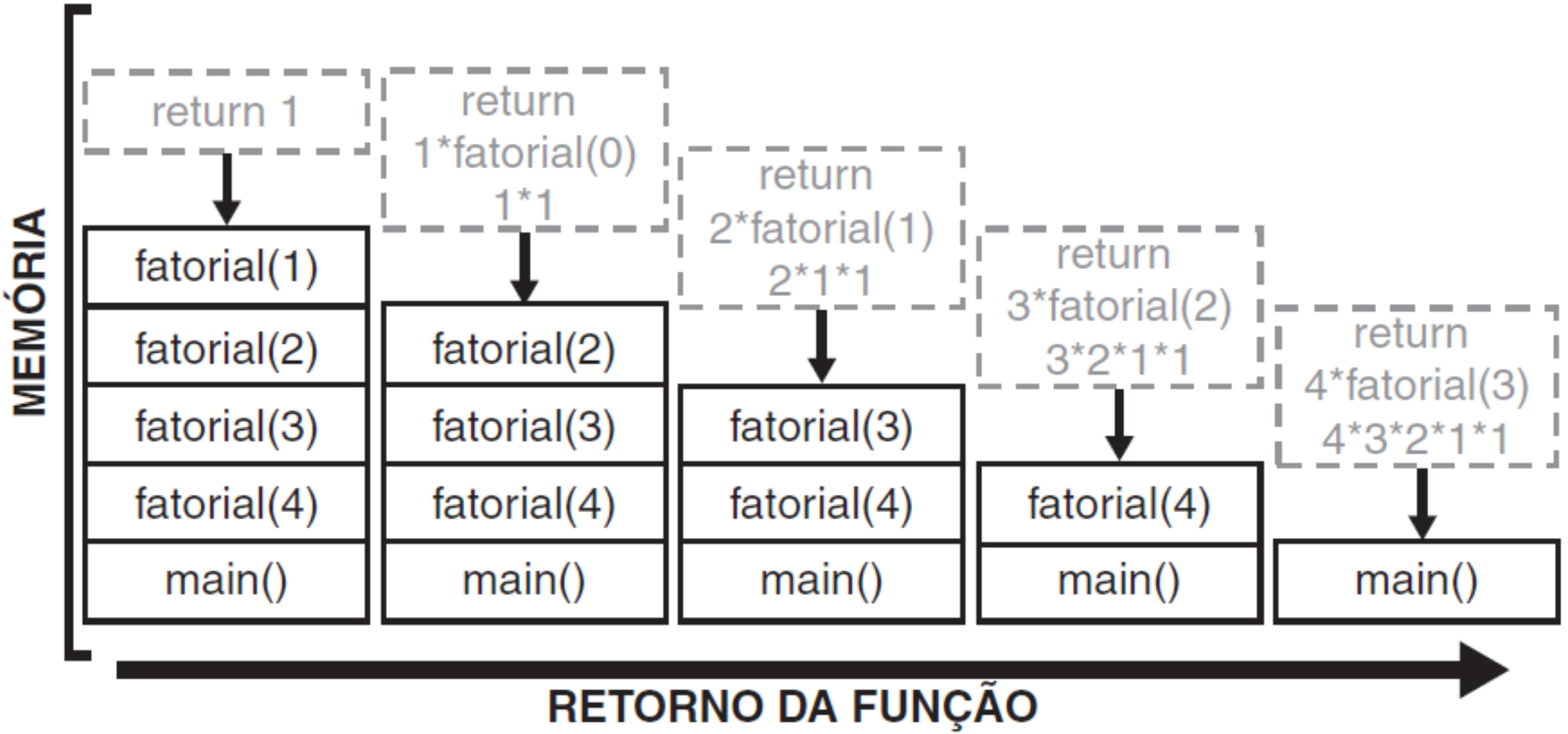
- O que acontece na chamada da função fatorial com um valor como n = 4?

```
int x = fatorial(4);
```



Como funciona a recursão?

- Uma vez que **chegamos ao caso-base**, é hora de fazer o caminho de volta da recursão.



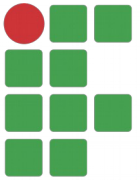
Sequência Fibonacci

- Essa sequência é um clássico da recursão
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
- A sequência de Fibonacci é definida como uma função recursiva utilizando a fórmula a seguir

$$F(n) = \begin{cases} 0, & \text{se n} = 0 \\ 1, & \text{se n} = 1 \\ F(n-1) + F(n-2), & \text{outros casos} \end{cases}$$

- Podemos escrever seu algoritmo assim:

Sequência Fibonacci



Sem Recursão

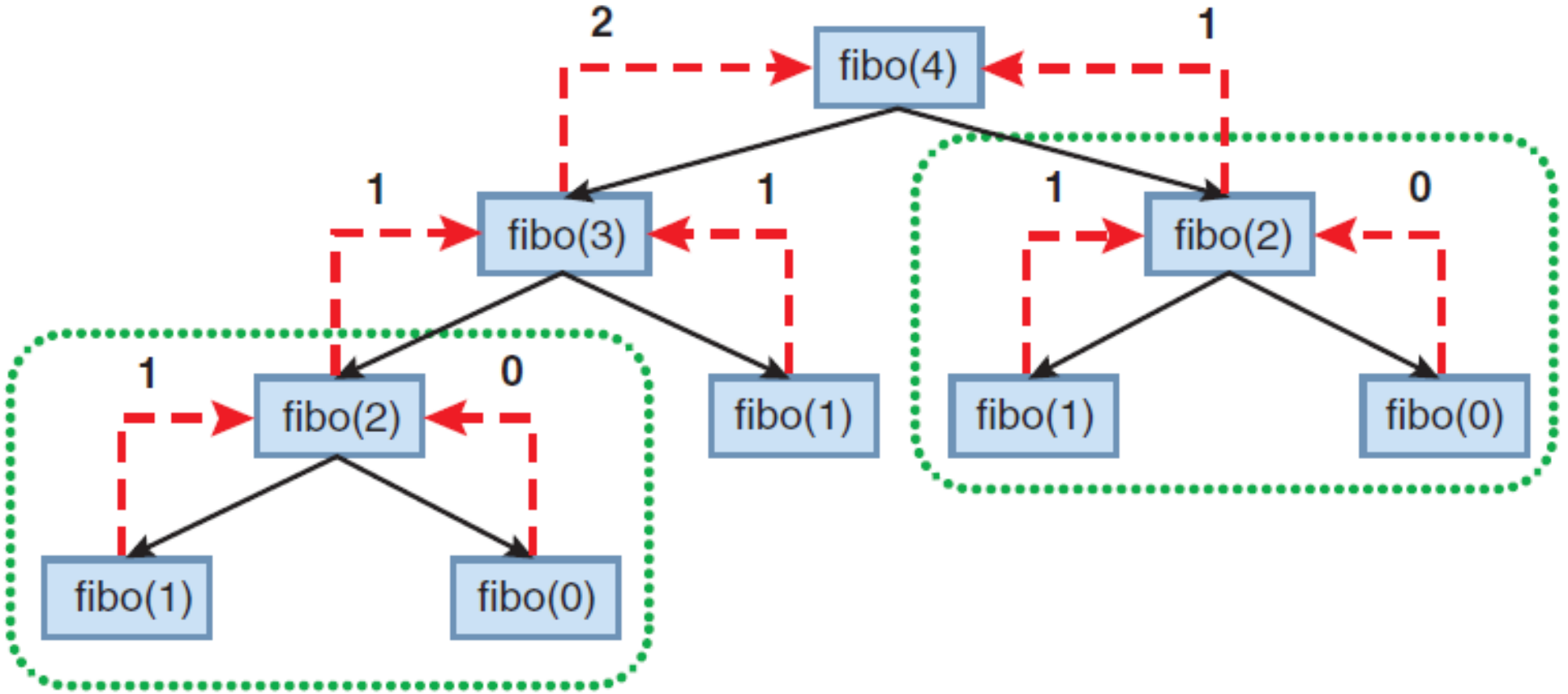
```
int fibo(int n){
    int i, t, c, a = 0, b = 1;
    for(i = 0; i < n; i++){
        c = a + b;
        a = b;
        b = c;
    }
    return a;
}
```

Com Recursão

```
int fiboR(int n){
    if (n == 0 || n == 1)
        return n;
    else
        return fiboR(n-1) + fiboR(n-2);
}
```

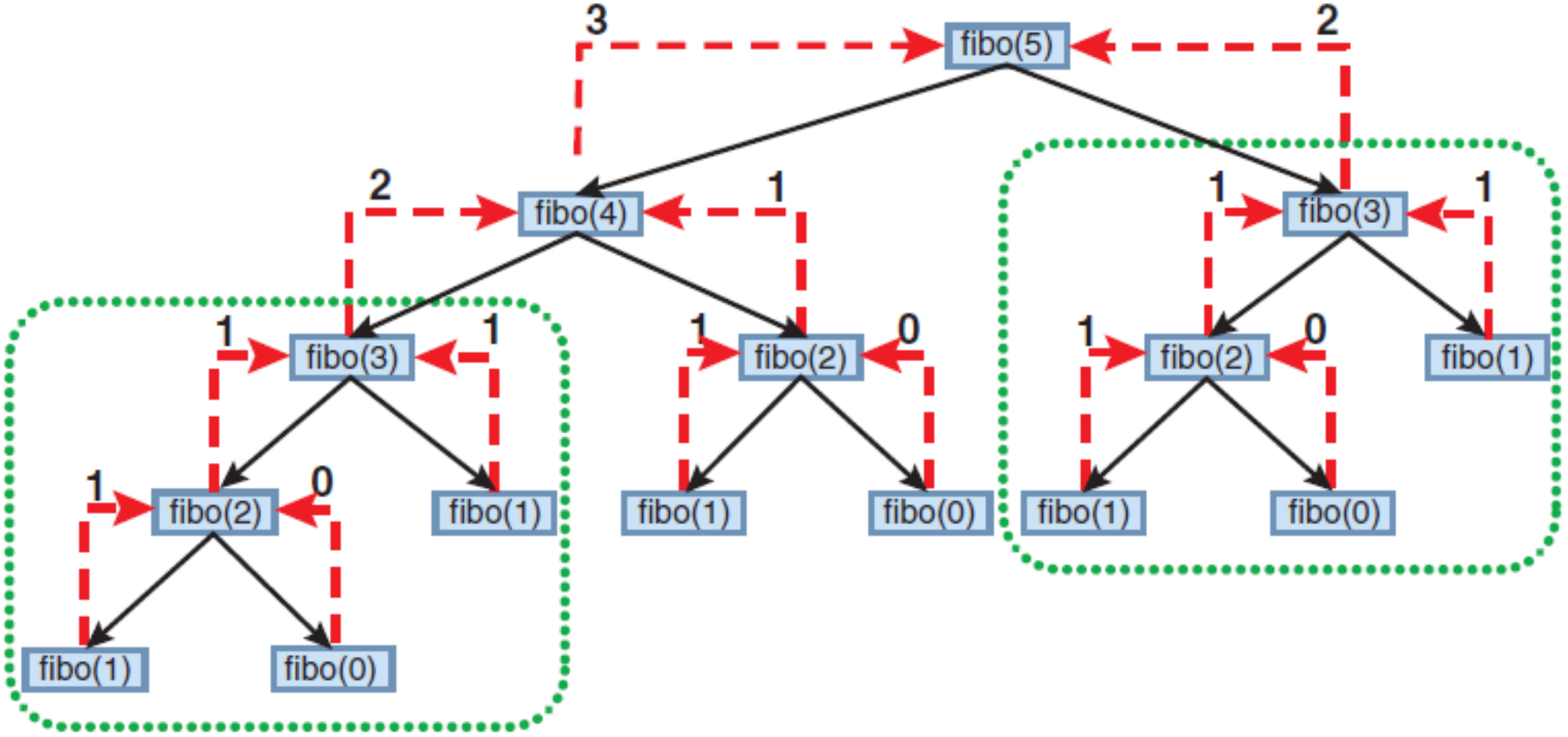
Sequência Fibonacci

- O código recursivo é mais elegante, mas é menos eficiente!

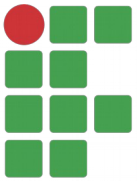


Sequência Fibonacci

- Aumentando para fibo(5)



Exercícios

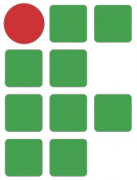


- Soma dos N primeiros números
  - Soma(5) = 5 + 4 + 3 + 2 + 1
  - Soma(4) = 4 + 3 + 2 + 1
  - Soma(3) = 3 + 2 + 1

    ....
  - Soma(5) = 5 + soma(4)
  - Soma(4) = 4 + soma(3)
  - Soma(3) = 3 + soma(2)

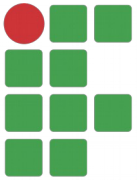
    ....

Exercícios



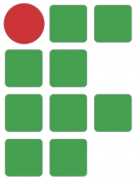
- Calcule uma função recursiva que receba 2 números (x e n) e calcule x^n
  - Caso base: $x^0 = 1$
  - Caso geral: $x^n = x * x^{n-1}$

Exercícios



- Usando recursividade, calcule a soma de todos os valores de um array de inteiros
  - Caso base → tamanho do array = 0; soma = 0
  - Caso geral → v[n-1] + soma(v, n-1)

Exercícios



- Faça o teste de mesa para o seguinte algoritmo, nos casos abaixo:
  - F1(0)
  - F1(1)
  - F1(5)

```
int f1(int n) {
if (n == 0)
    return (1);
else
    return(n * f1(n-1));
}
```

Exercícios

- Faça o teste de mesa para o seguinte algoritmo, nos casos abaixo:
  - F2(0)
  - F2(1)
  - F2(5)

```
int f2(int n) {
    if (n == 0)
        return (1);
    if (n == 1)
        return (1);
    else
        return(f2(n-1)+ 2*f2(n-2));
}
```

Exercícios

- Faça o teste de mesa para o seguinte algoritmo, nos casos abaixo:
  - F2(0)
  - F2(1)
  - F2(5)

```
void f3(int n) {
    if (n == 0)
        printf("Zero ");
    else {
        printf("%d ",n);
        f3(n-1);

    }
}
```

Exercícios

- Faça o teste de mesa para o seguinte algoritmo, nos casos abaixo:
  - F2(0)
  - F2(1)
  - F2(5)

```
void f4(int n) {
    if (n == 0)
        printf("Zero ");
    else {
        f4(n-1);
        printf("%d ",n);
    }
}
```