Algoritmos recursivos

emanoelim@utfpr.edu.br

Câmpus Pato Branco

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PATO BRANCO

MINISTÉRIO DA EDUCAÇÃO

BRASIL
GOVERNO FEDERAL

Algoritmos recursivos

- Dizemos que uma função é recursiva, quando dentro dela existe uma chamada para si mesma.

Algoritmos recursivos

- Vamos analisar isso através de um exemplo. Considere o cálculo do fatorial do número 5:
  5! = 5 x 4 x 3 x 2 x 1

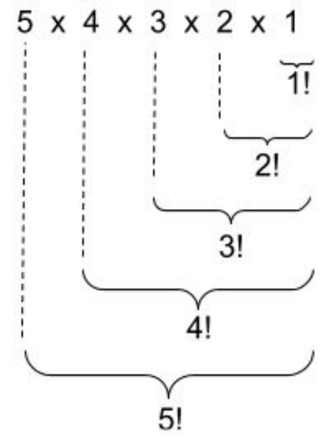
Algoritmos recursivos

- Uma possível solução em linguagem C seria a seguinte:

```c
int fatorial(int n) {
  int i;
  int fat = n;
  for(i = n - 1; i > 0; i--)
    fat *= i;
  return fat;
}
```

Algoritmos recursivos

- No entanto, podemos perceber o cálculo do fatorial da seguinte forma:



Ou seja:

5! = 5 x 4!

4! = 4 x 3!

3! = 3 x 2!

2! = 2 x 1!

1! = 1

Algoritmos recursivos

- Assim, o problema de calcular o fatorial de 5 pode ser quebrado em partes cada vez menores até encontrar a instância mais simples do problema, que é 1!
  - Resolvendo 1! é possível resolver 2!
  - Resolvendo 2! é possível resolver 3!
  - Resolvendo 3! é possível resolver 4!
  - Resolvendo 4! é possível resolver 5!

Algoritmos recursivos

- Este exemplo pode ser implementado em C da seguinte maneira:

```
int fatorial_rec(int n) {
  if(n == 1) // menor instância do problema - já retorna o
resultado
    return 1;
  else // chama a função novamente para calcular n * (n - 1)
    return n * fatorial_rec(n - 1);
}
```

Algoritmos recursivos

- Vamos pensar na pilha de execução dessa função para entender o que acontece:

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

```c
int fatorial_rec(int n) {
  if(n == 1)
    return 1;
  else
    return n * fatorial_rec(n - 1);
}
```

fatorial_rec(5)

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

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

fatorial_rec(4)
fatorial_rec(5)

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

```c
int fatorial_rec(int n) {
  if(n == 1)
    return 1;
  else
    return n * fatorial_rec(n - 1);
}
```

fatorial_rec(3)
fatorial_rec(4)
fatorial_rec(5)

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

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

fatorial_rec(2)
fatorial_rec(3)
fatorial_rec(4)
fatorial_rec(5)

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

```c
int fatorial_rec(int n) {
  if(n == 1)
    return 1;
  else
    return n * fatorial_rec(n - 1);
}
```

fatorial_rec(1)
fatorial_rec(2)
fatorial_rec(3)
fatorial_rec(4)
fatorial_rec(5)

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

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

1
fatorial_rec(2)
fatorial_rec(3)
fatorial_rec(4)
fatorial_rec(5)

1! = 1

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

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

2
fatorial_rec(3)
fatorial_rec(4)
fatorial_rec(5)

2! = 2 * 1 = 2

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

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

6
fatorial_rec(4)
fatorial_rec(5)

3! = 3 * 2 = 6

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

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

24
fatorial_rec(5)

4! = 4 * 6 = 24

Algoritmos recursivos

- Pilha de execução da função fatoririal_rec():

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

5! = 5 * 24 = 120

Algoritmos recursivos

- Em resumo:
  - Enquanto for necessário dividir o problema em problemas menores, a função continuará chamando a si mesma até chegar na menor instância do problema.
  - Quando isso acontece, a função para de dividir e começa a gerar os resultados.
  - Cada resultado é devolvido para a função que solicitou esse resultado, para que ela possa continuar seu processamento.
  - Cada resultado é agrupado com o próximo e assim sucessivamente, até formar o resultado final.
  - Sempre que um resultado retorna para a função que o chamou, ele é desempilhado da pilha de execução e os endereços ocupados por ele são liberados.

Condição de parada

- No exemplo anterior, podemos dizer que 1! é o caso base do problema fatorial.
- O caso base de um problema irá definir a condição de parada da função.
- Uma função recursiva sempre deve ter uma condição de parada atingível para que não fique em um laço infinito.
- Portanto, podemos concluir que em nosso código deverá haver um if que testa se o problema já chegou no caso base, retornando o resultado do mesmo se o resultado do teste for verdadeiro.

“Receita” para uma solução recursiva

- Considerando que para cada instância de um problema existe uma instância menor do mesmo problema, o problema pode ser resolvido de forma recursiva. Assim, podemos usar o seguinte método:
  - se o problema é suficientemente pequeno, resolver;
  - se o problema é grande, reduzir para uma versão menor do mesmo problema e aplicar o método novamente para o problema menor.

Exemplos

1) Soma dos N primeiros números inteiros

Dado um número N, deve ser feita a soma dos N primeiros números inteiros.

Por exemplo, N = 5:

$S(5) = 1 + 2 + 3 + 4 + 5 = 15$

ou :

$S(5) = 5 + 4 + 3 + 2 + 1 = 15$

Exemplos

1) Soma dos N primeiros números inteiros

Solução iterativa:

```
int soma_n(int n) {
  int i;
  int soma = n;
  for(i = n - 1; i > 0; i--)
    soma += i;
  return soma;
}
```

Exemplos

1) Soma dos N primeiros números inteiros

Pensando em uma solução recursiva, o problema poderia ser dividido da seguinte forma:

S(5) = 5 + S(4)

S(4) = 4 + S(3)

S(3) = 3 + S(2)

S(2) = 2 + S(1)

S(1) = 1

Exemplos

1) Soma dos N primeiros números inteiros

Solução recursiva:

```
int soma_n_rec(int n){
  if(n == 1) // menor instância do problema - já retorna o
resultado
    return 1;
  else // chama a função novamente para calcular n * (n - 1)
    return n + soma_n_rec(n - 1);
}
```

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```
int soma_n_rec(int n){
  if(n == 1)
    return 1;
  else
    return n + soma_n_rec(n - 1);
}
```

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```c
int soma_n_rec(int n){
  if(n == 1)
    return 1;
  else
    return n + soma_n_rec(n - 1);
}
```

soma_n_rec(4)
soma_n_rec(5)

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```
int soma_n_rec(int n){
  if(n == 1)
    return 1;
  else
    return n + soma_n_rec(n - 1);
}
```

soma_n_rec(3)
soma_n_rec(4)
soma_n_rec(5)

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```
int soma_n_rec(int n){
  if(n == 1)
    return 1;
  else
    return n + soma_n_rec(n - 1);
}
```

soma_n_rec(2)
soma_n_rec(3)
soma_n_rec(4)
soma_n_rec(5)

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```
int soma_n_rec(int n){
   if(n == 1)
      return 1;
   else
      return n + soma_n_rec(n - 1);
}
```

soma_n_rec(1)
soma_n_rec(2)
soma_n_rec(3)
soma_n_rec(4)
soma_n_rec(5)

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```
int soma_n_rec(int n){
   if(n == 1)
      return 1;
   else
      return n + soma_n_rec(n - 1);
}
```

1
soma_n_rec(2)
soma_n_rec(3)
soma_n_rec(4)
soma_n_rec(5)

S(1) = 1

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```
int soma_n_rec(int n){
   if(n == 1)
      return 1;
   else
      return n + soma_n_rec(n - 1);
}
```

2
soma_n_rec(3)
soma_n_rec(4)
soma_n_rec(5)

S(2) = 2 + 1 = 2

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```
int soma_n_rec(int n){
   if(n == 1)
      return 1;
   else
      return n + soma_n_rec(n - 1);
}
```

6
soma_n_rec(4)
soma_n_rec(5)

S(3) = 3 + 2 = 6

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```c
int soma_n_rec(int n){
  if(n == 1)
    return 1;
  else
    return n + soma_n_rec(n - 1);
}
```

10
soma_n_rec(5)

S(4) = 4 + 6 = 10

Exemplos

1) Soma dos N primeiros números inteiros

Analisando a pilha de execução para n = 5:

```c
int soma_n_rec(int n){
  if(n == 1)
    return 1;
  else
    return n + soma_n_rec(n - 1);
}
```

15

S(5) = 5 + 10 = 15

Exemplos

2) Encontrando o maior elemento de um vetor

Solução iterativa:

```
int maior(int n, int v[]) {
    int maior = v[0], i;
    for(i = 0; i < n; i++)
        if(v[i] > maior)
            maior = v[i];
    return maior;
}
```

Exemplos

2) Encontrando o maior elemento de um vetor

A ideia básica desse algoritmo é ir comparando o atual maior com cada elemento do vetor. Essa mesma ideia pode ser implementada de forma recursiva.

Exemplos

2) Encontrando o maior elemento de um vetor

Solução recursiva:

```
int maior_rec(int n, int v[]){
    if(n == 1) // considerando somente a 1ª posição do vetor, o maior é o 1º
        return v[0];
    else {
        int maior;
        maior = maior_rec(n - 1, v);
        if(maior > v[n - 1])
            return maior;
        else
            return v[n - 1];
```

Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```
int maior_rec(int n, int v[]){
    if(n == 1)
        return v[0];
    else {
        int maior;
        maior = maior_rec(n - 1, v);
        if(maior > v[n - 1])
            return maior;
        else
            return v[n - 1];
```

maior_rec(4, v)

Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```c
int maior_rec(int n, int v[]){
    if(n == 1)
        return v[0];
    else {
        int maior;
        maior = maior_rec(n - 1, v);
        if(maior > v[n - 1])
            return maior;
        else
            return v[n - 1];
```

maior_rec(3, v)
maior_rec(4, v)

Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```
int maior_rec(int n, int v[]){
    if(n == 1)
        return v[0];
    else {
        int maior;
        maior = maior_rec(n - 1, v);
        if(maior > v[n - 1])
            return maior;
        else
            return v[n - 1];
```

maior_rec(2, v)
maior_rec(3, v)
maior_rec(4, v)

Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```
int maior_rec(int n, int v[]){
    if(n == 1)
        return v[0];
    else {
        int maior;
        maior = maior_rec(n - 1, v);
        if(maior > v[n - 1])
            return maior;
        else
            return v[n - 1];
```

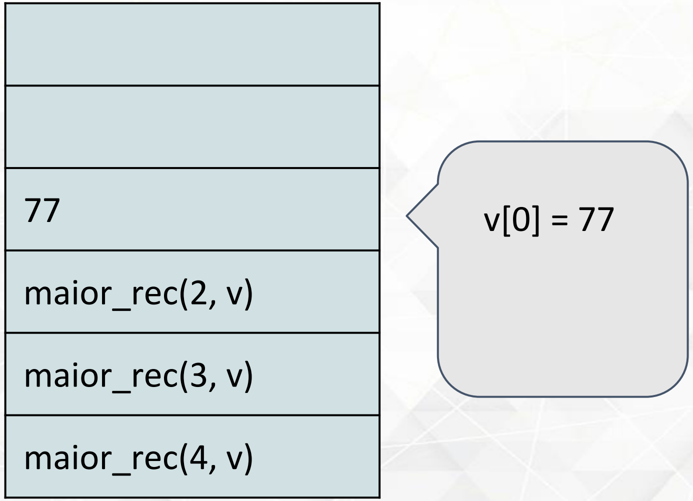
maior_rec(1, v)
maior_rec(2, v)
maior_rec(3, v)
maior_rec(4, v)

Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```c
int maior_rec(int n, int v[]){
	if(n == 1)
		return v[0];
	else {
		int maior;
		maior = maior_rec(n - 1, v);
		if(maior > v[n - 1])
			return maior;
		else
			return v[n - 1];
```



Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```c
int maior_rec(int n, int v[]){
      if(n == 1)
          return v[0];
      else {
          int maior;
          maior = maior_rec(n - 1, v);
          if(maior > v[n - 1])
             return maior;
          else
             return v[n - 1];
```

88
maior_rec(3, v)
maior_rec(4, v)

maior = 77
n = 2
v[1] = 88
77 > 88? Não
maior = 88

Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```
int maior_rec(int n, int v[]){
     if(n == 1)
         return v[0];
     else {
         int maior;
         maior = maior_rec(n - 1, v);
         if(maior > v[n - 1])
            return maior;
         else
            return v[n - 1];
```

88
maior_rec(4, v)

maior = 88
n = 3
v[2] = 66
88 > 66? Sim
maior = 88

Exemplos

2) Encontrando o maior elemento de um vetor

Analisando a pilha de execução para o vetor [77, 88, 66, 99]:

```
int maior_rec(int n, int v[]){
    if(n == 1)
        return v[0];
    else {
        int maior;
        maior = maior_rec(n - 1, v);
        if(maior > v[n - 1])
            return maior;
        else
            return v[n - 1];
```

99

maior = 88
n = 4
v[3] = 99
88 > 99? Não
maior = 99

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

A série de Fibonacci é dada por: 1, 1, 2, 3, 5, 8, 13, 21…

Seus termos são calculados conforme a equação abaixo:

$f_0 = 0, f_1 = 1$

$f_n = f_{n-1} + f_{n-2}, \qquad \text{para } n \geq 2$

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Exemplo: cálculo do oitavo termo

- Primeiro termo = 0
- Segundo termo = 1
- Terceiro termo = 1 + 0 = 1
- Quarto termo = 1 + 1 = 2
- Quinto termo = 2 + 1 = 3
- Sexto termo = 3 + 2 = 5
- Sétimo termo = 5 + 3 = 8

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Solução iterativa:

```
int fib(int x){
    int i;
    int primeiro = 0, segundo = 1, proximo;
    for(i = 0; i < x-2; i++) { // os dois primeiros são fixos (0 e 1)
        proximo = segundo + primeiro;
        primeiro = segundo;
        segundo = proximo;
    }
    return proximo;
}
```

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

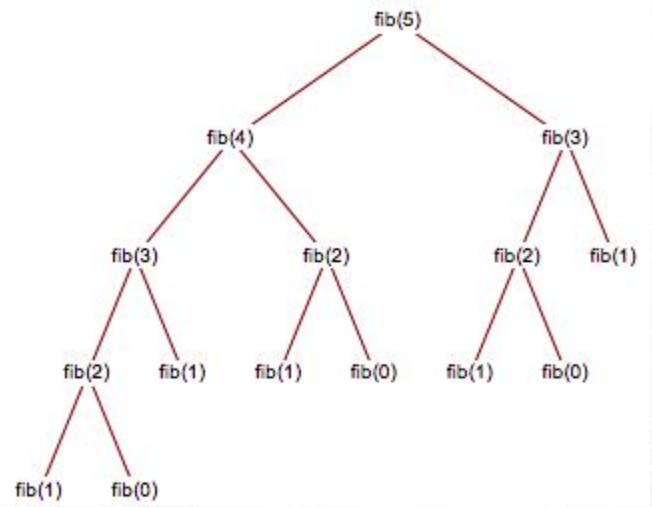
Solução recursiva:

```
int fib_rec(int x){
	if(x == 0)
		return 0;
	if(x == 1)
		return 1;
	else
		return fib_rec(x - 1) + fib_rec(x - 2);
}
```

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

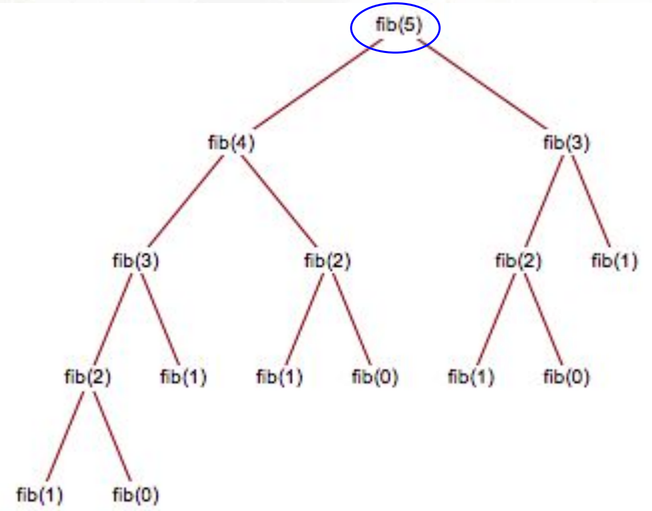
Para calcular o termo 5, as seguintes chamadas serão feitas:



Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

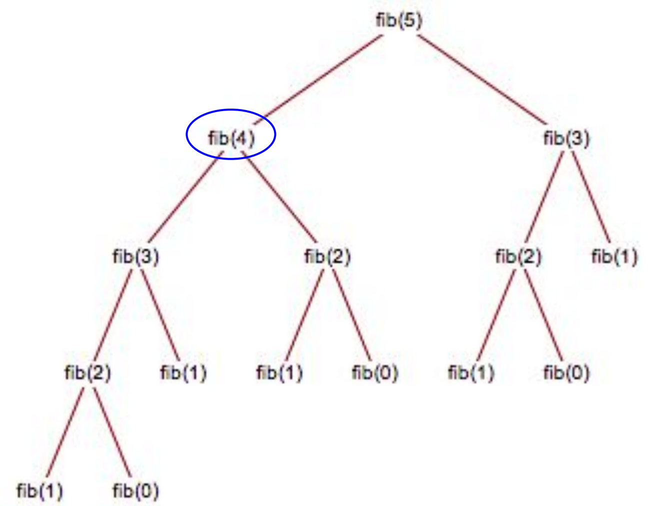
Analisando a pilha de execução para n = 5



Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

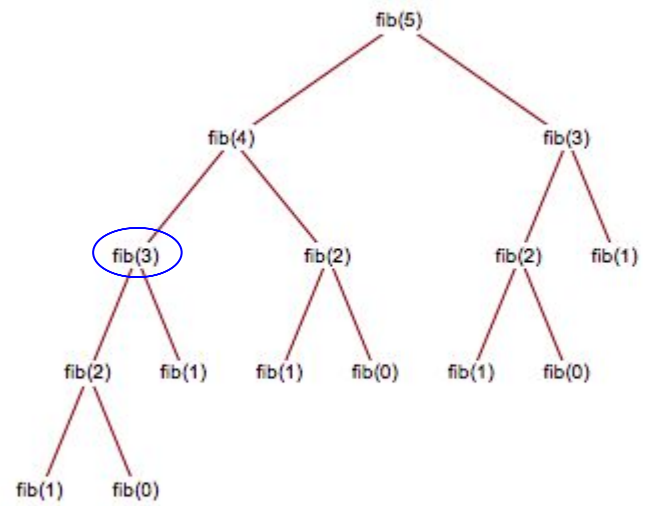


fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

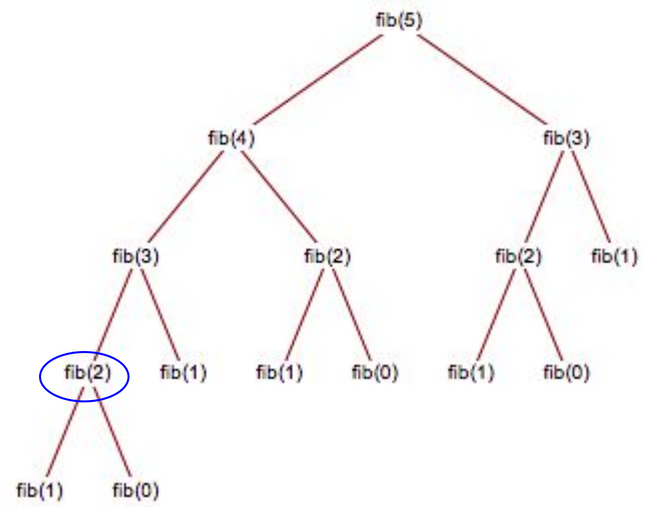


fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

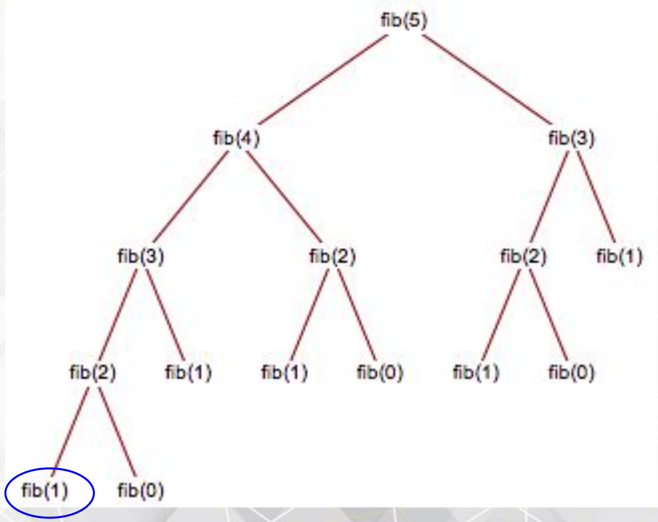


fib(2)
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

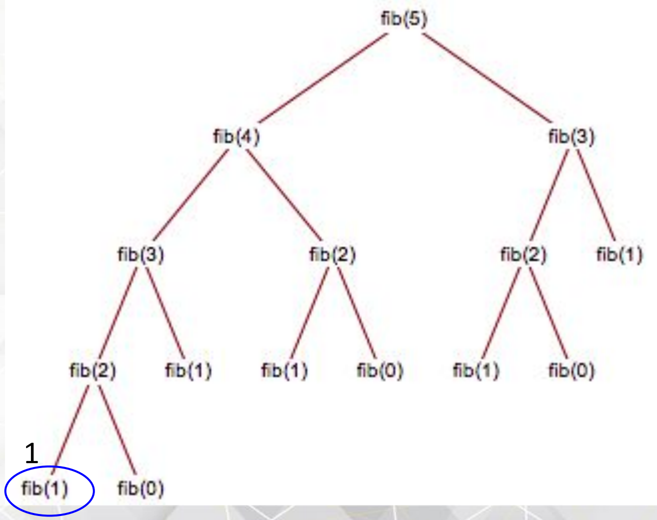


fib(1)
fib(2)
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

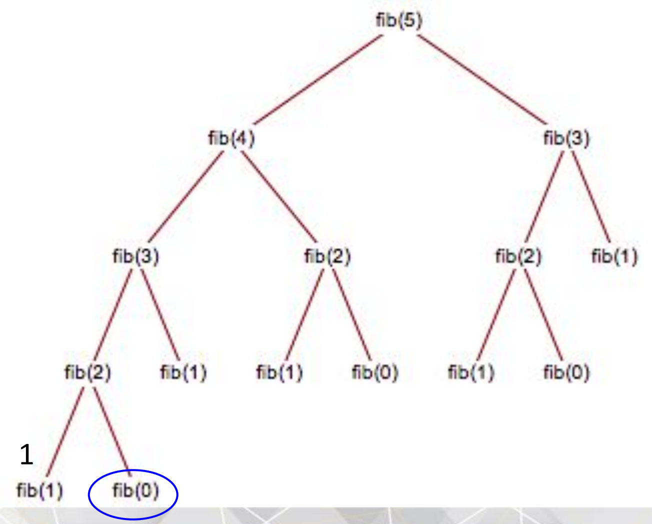


1
fib(2)
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

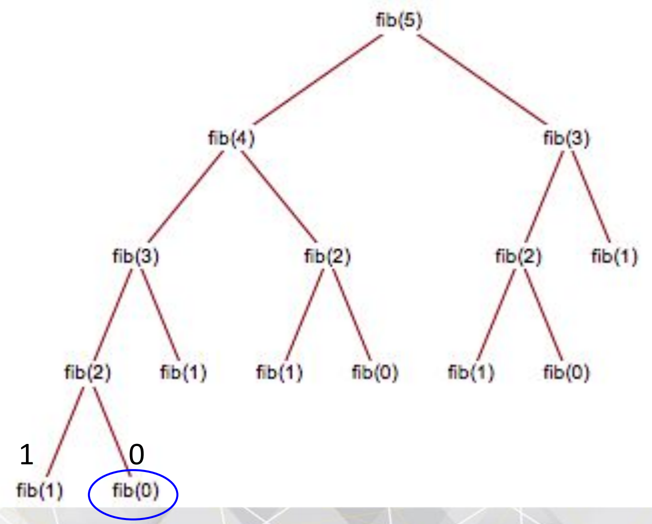


fib(0)
1
fib(2)
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

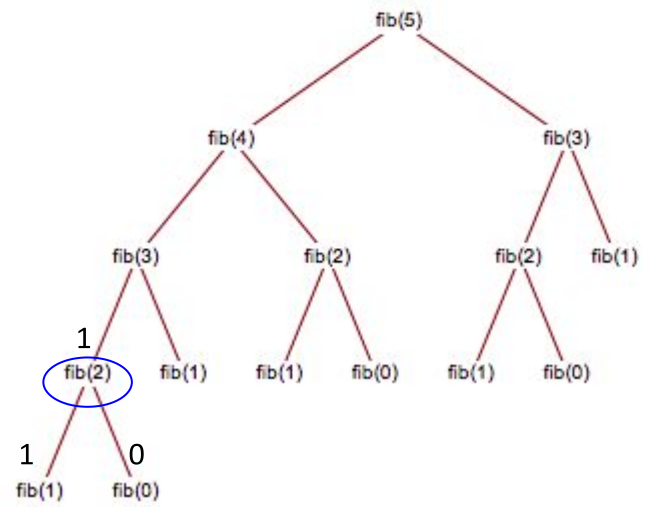


0
1
fib(2)
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

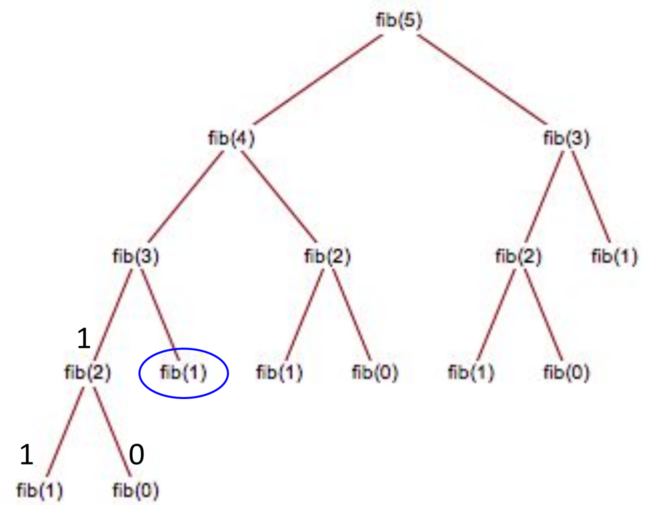


1
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

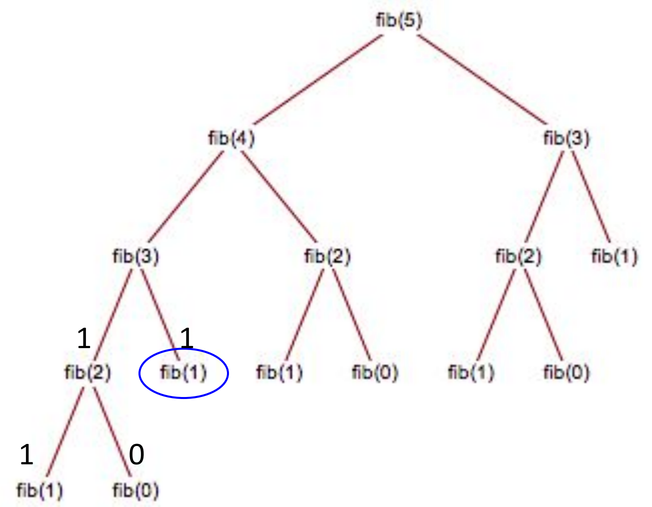


fib(1)
1
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

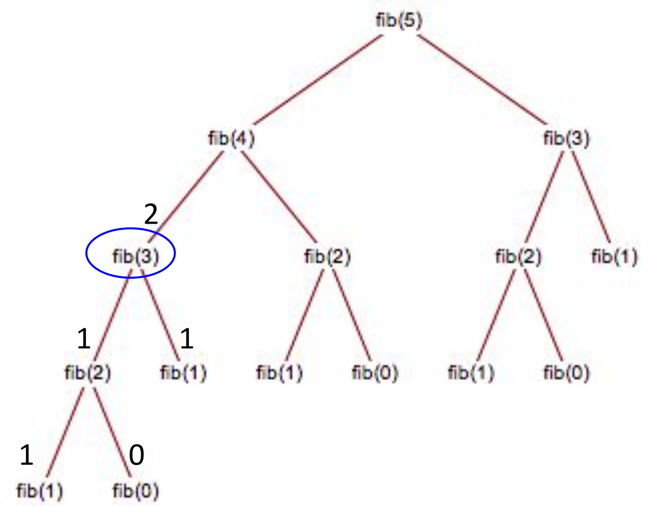


1
1
fib(3)
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

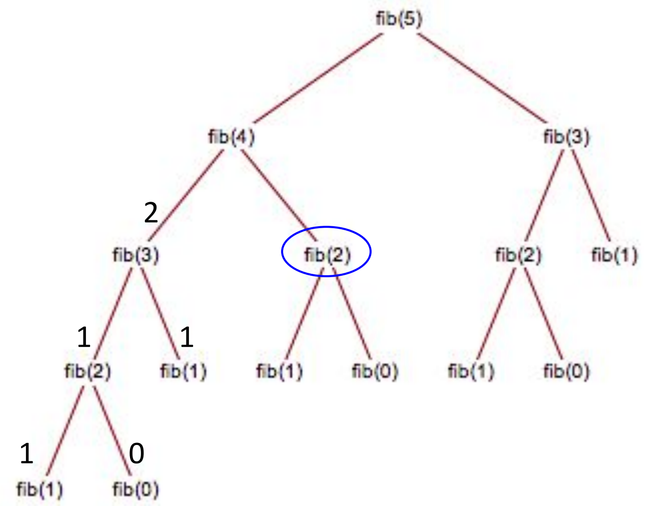


2
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

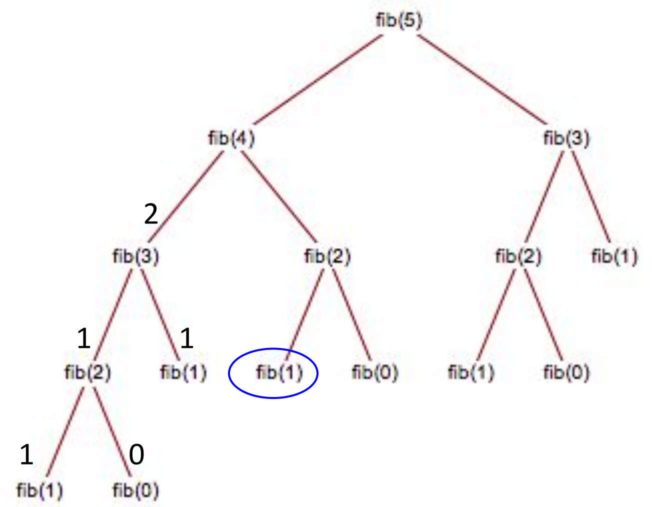


fib(2)
2
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

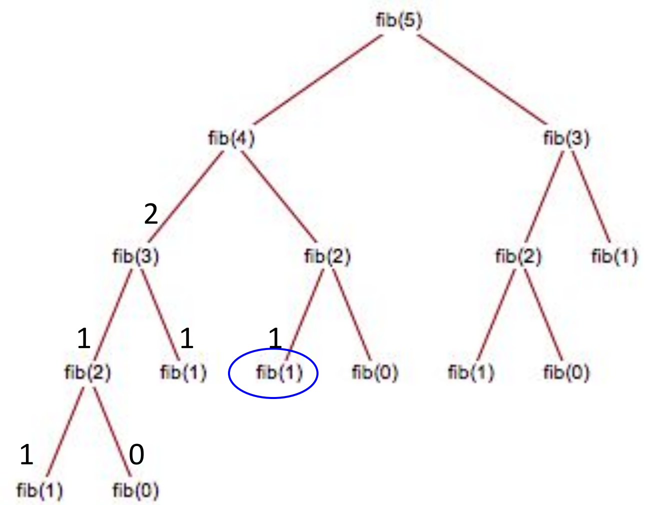


fib(1)
fib(2)
2
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

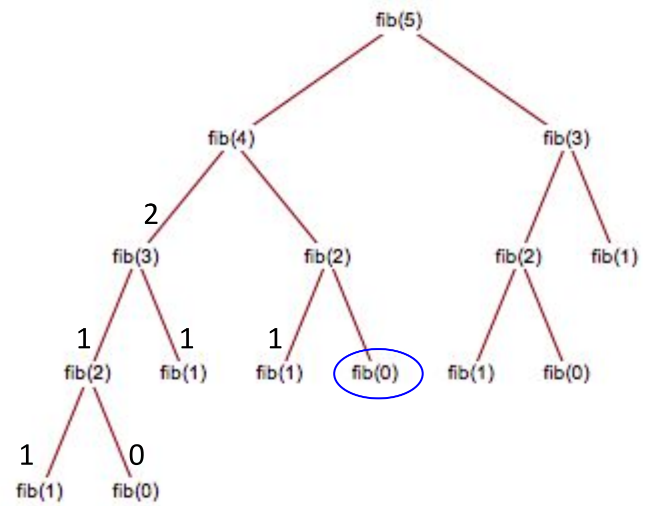


1
fib(2)
2
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

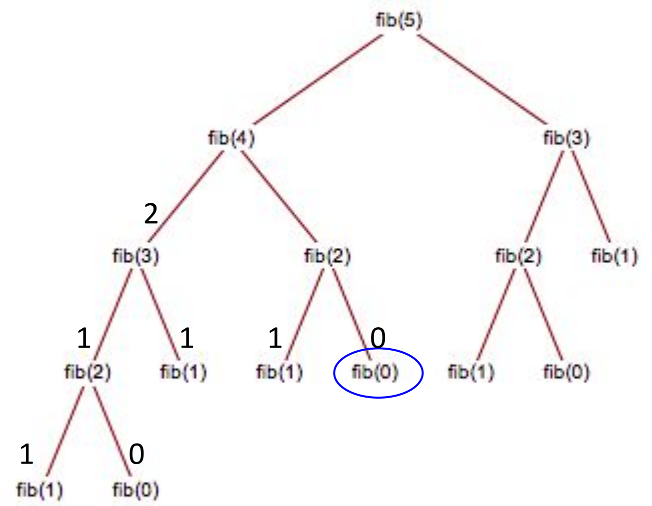


fib(0)
1
fib(2)
2
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

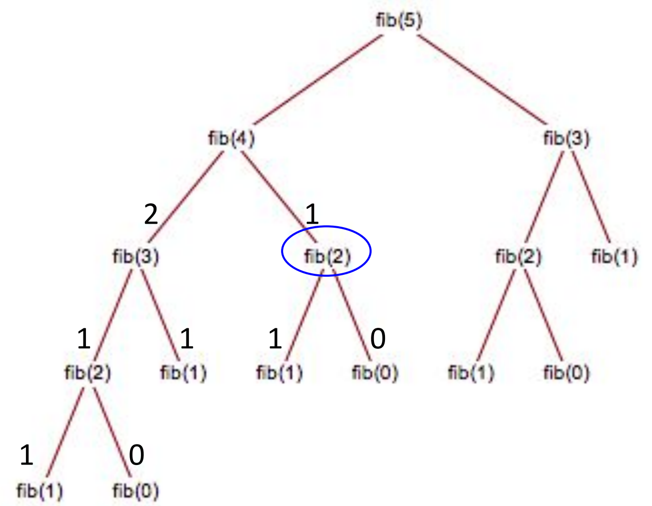


0
1
fib(2)
2
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

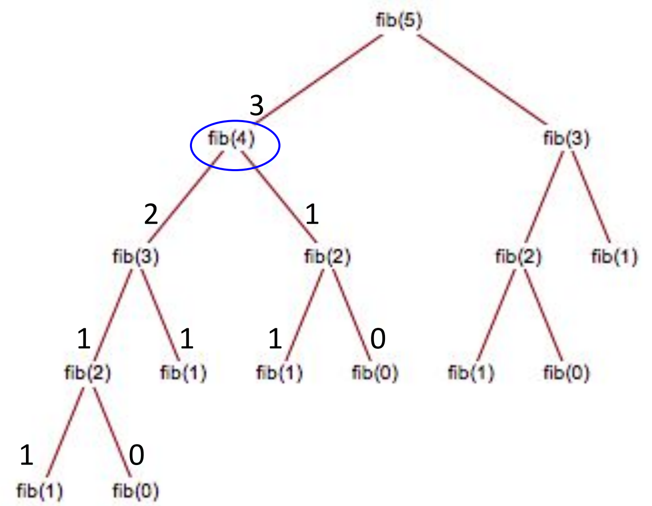


1
2
fib(4)
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

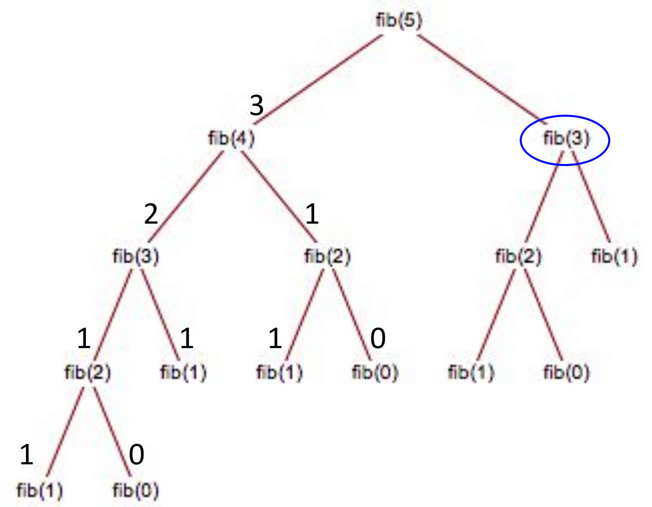


3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

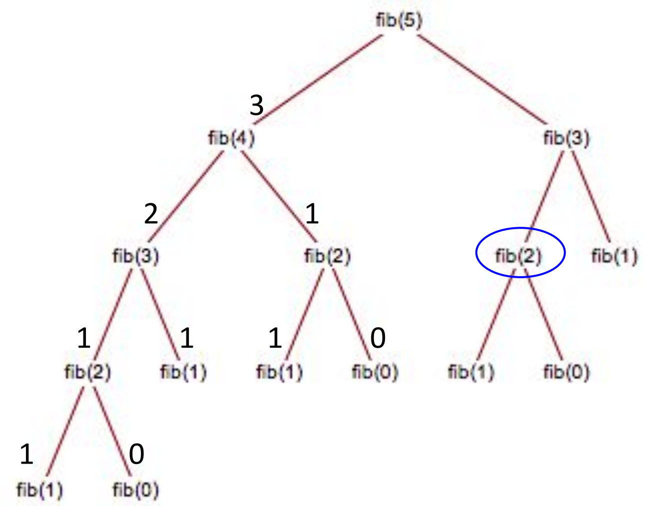


fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

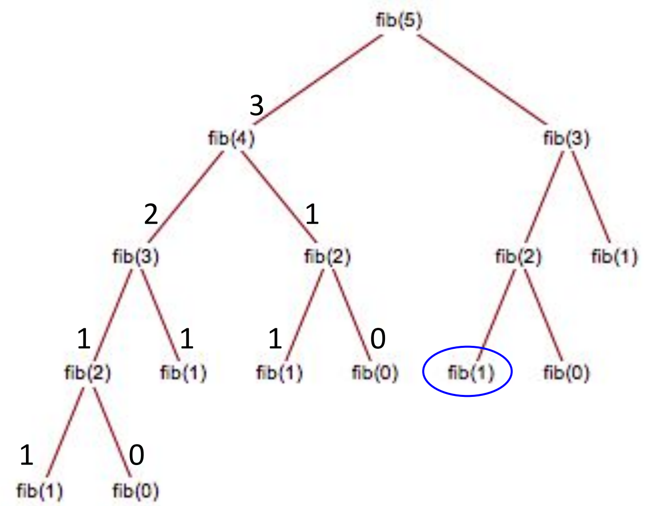


fib(2)
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

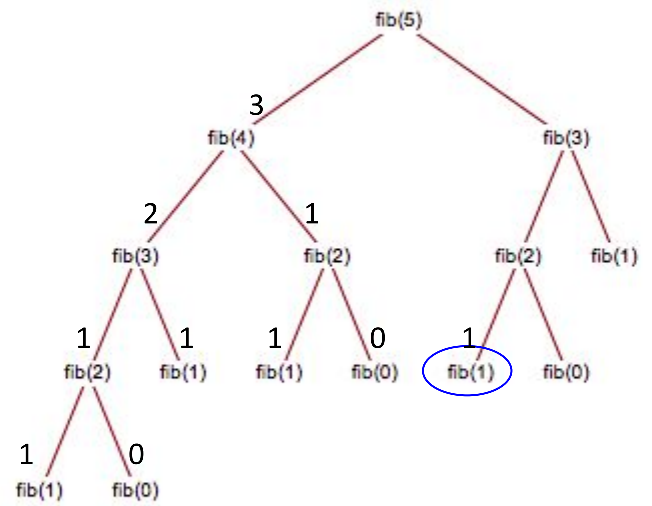


fib(1)
fib(2)
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

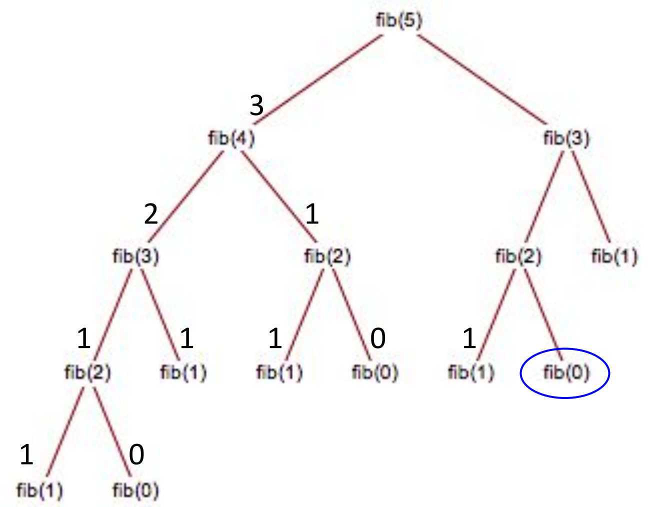


1
fib(2)
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

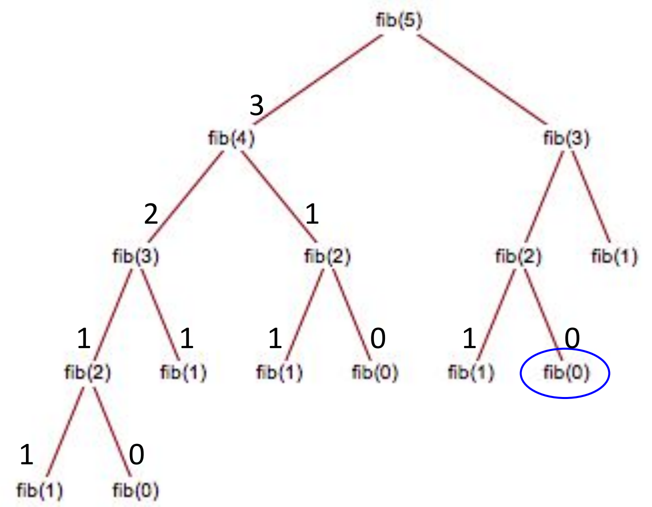


fib(0)
1
fib(2)
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

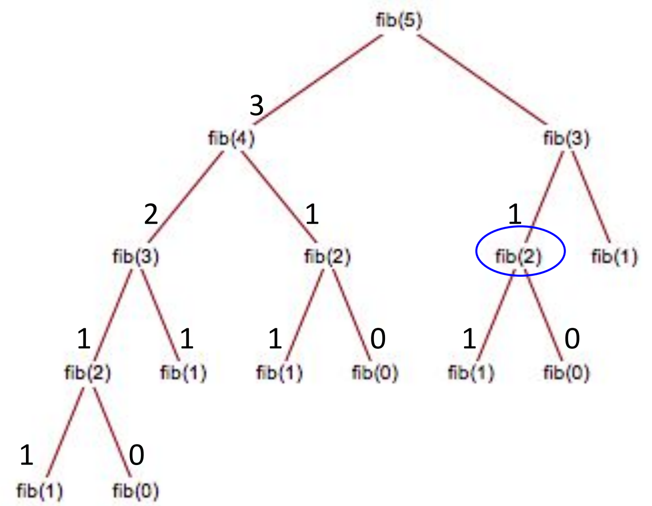


0
1
fib(2)
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

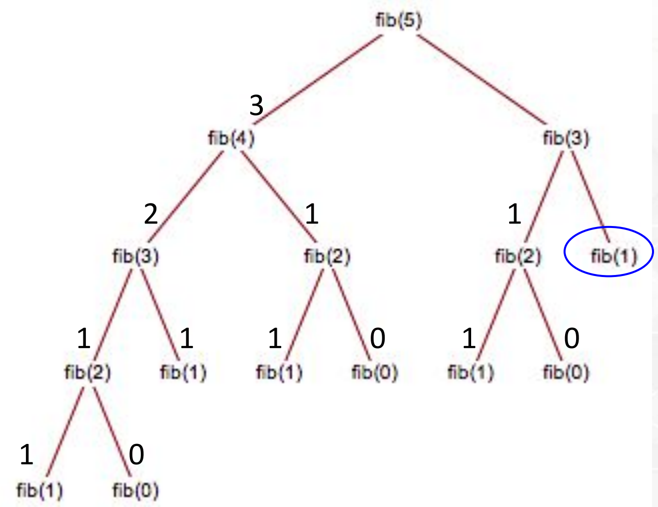


1
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

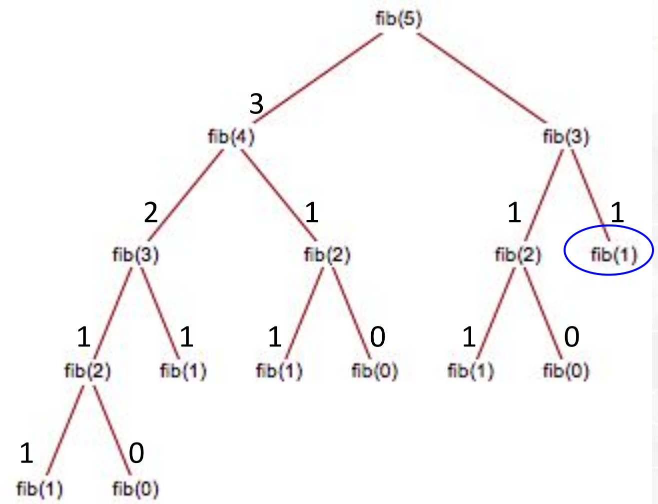


fib(1)
1
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

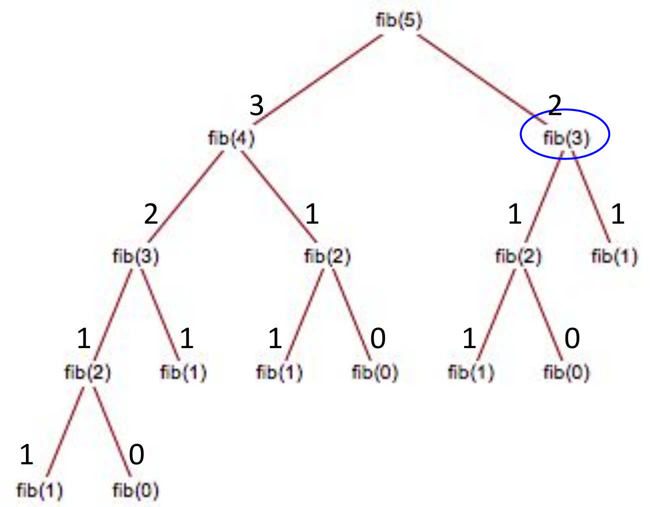


1
1
fib(3)
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5

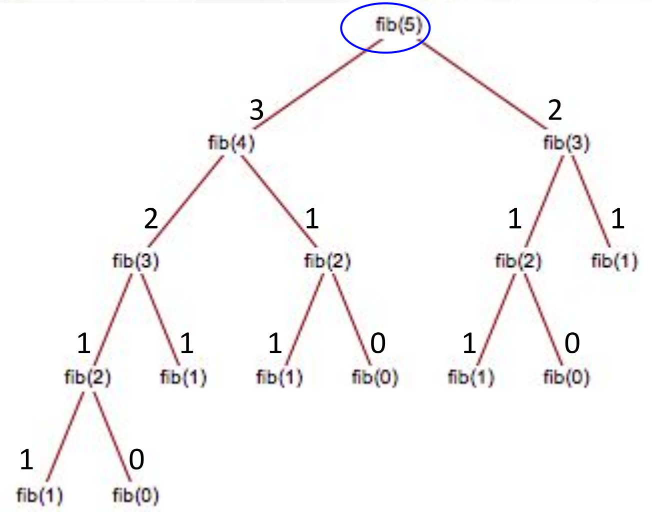


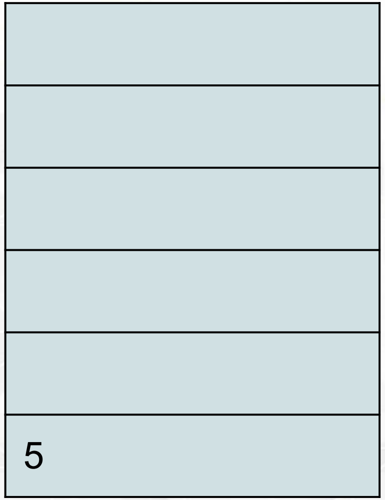
2
3
fib(5)

Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Analisando a pilha de execução para n = 5



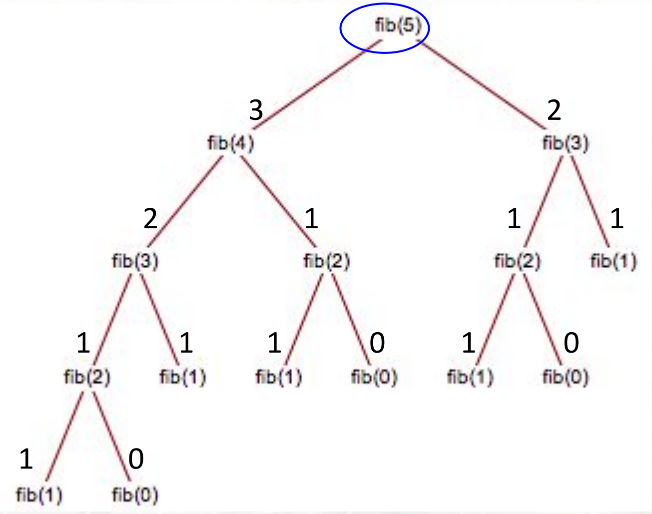


Exemplos

3) Calcular o n-ésimo termo da série de Fibonacci

Mesmo sem uma análise matemática apurada, é possível ver que esta função é extremamente ineficiente, porque um mesmo valor é recalculado várias vezes:

- fib_rec(3) vai ser calculado 2x;
- fib_rec(2) vai ser calculado 3x;
- fib_rec(1) vai ser calculado 5x;
- fib_rec(0) vai ser calculado 3x:



Exemplos

4) Imprimir uma lista encadeada de trás para a frente

Vimos que como a lista encadeada apontava somente para o próximo, era possível imprimir a lista da esquerda para a direita. Com uma função recursiva, conseguimos fazer a impressão da direita para a esquerda.

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

```
void imprime_rec(Celula *aux) {
  if (aux == NULL)
     return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

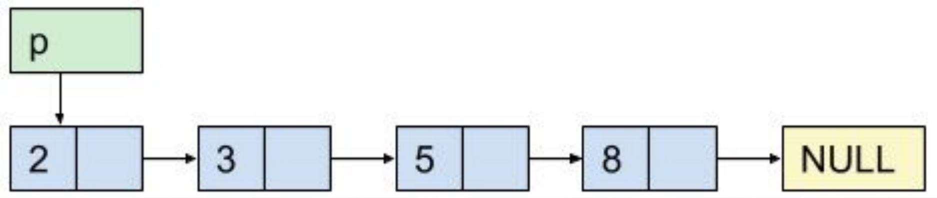
Chamamos a função passando a primeira célula da lista:

```
Celula *p;
p = primeira(l);
imprime_rec(p);
```

Exemplos

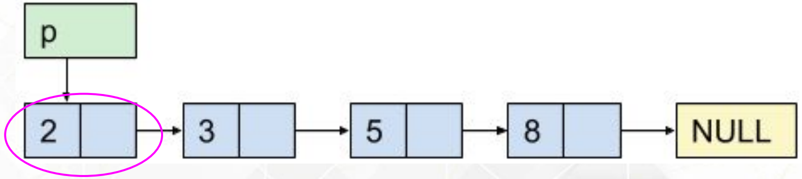
4) Imprimir uma lista encadeada de trás para a frente

Analisando a pilha de execução para a lista:



Exemplos

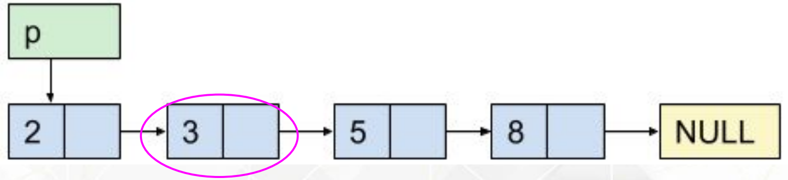
4) Imprimir uma lista encadeada de trás para a frente

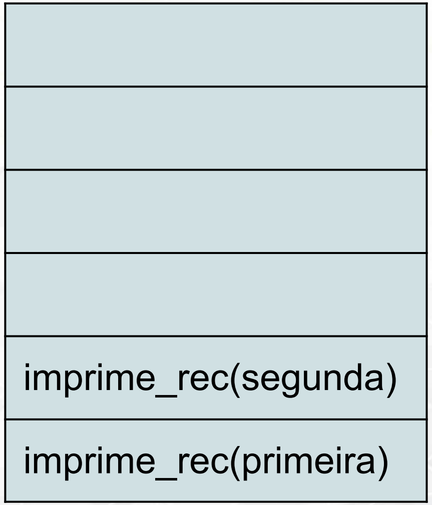



```
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

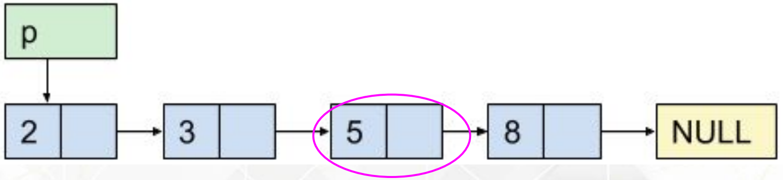





```c
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

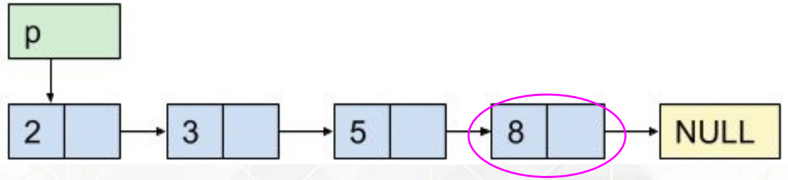



```
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

imprime_rec(terceira)
imprime_rec(segunda)
imprime_rec(primeira)

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

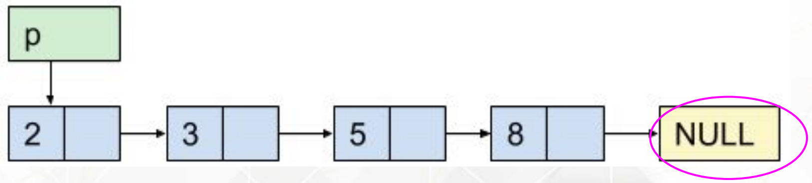



```c
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

imprime_rec(quarta)
imprime_rec(terceira)
imprime_rec(segunda)
imprime_rec(primeira)

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

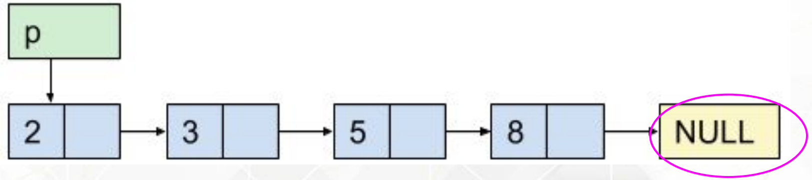


imprime_rec(null)
imprime_rec(quarta)
imprime_rec(terceira)
imprime_rec(segunda)
imprime_rec(primeira)

```
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

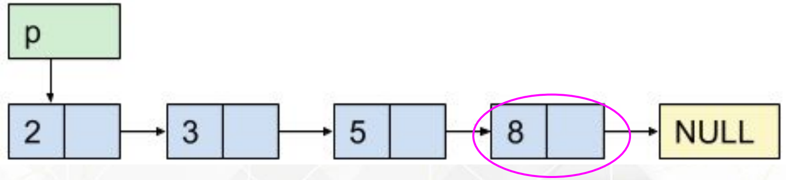



```c
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

imprime_rec(quarta)
imprime_rec(terceira)
imprime_rec(segunda)
imprime_rec(primeira)

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

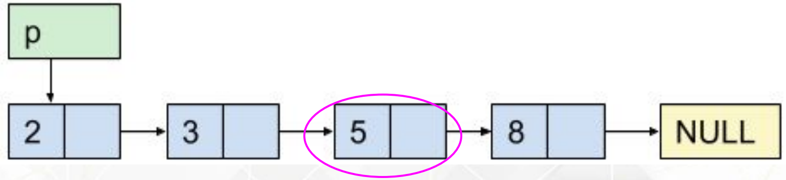


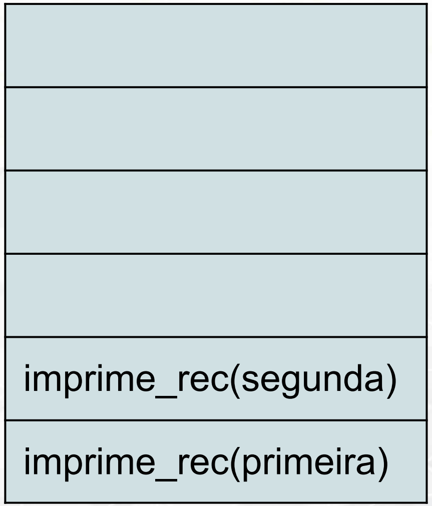

```c
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

imprime_rec(terceira)
imprime_rec(segunda)
imprime_rec(primeira)

Exemplos

4) Imprimir uma lista encadeada de trás para a frente

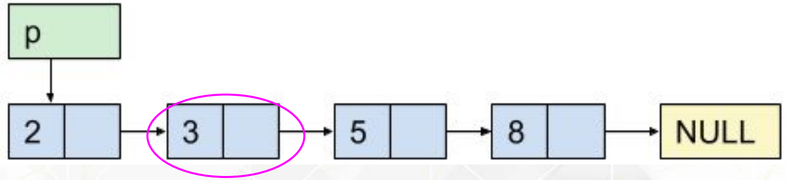





```c
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

Exemplos

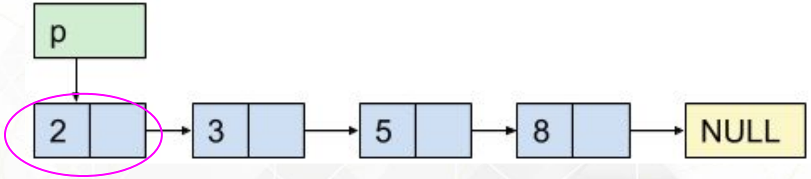
4) Imprimir uma lista encadeada de trás para a frente

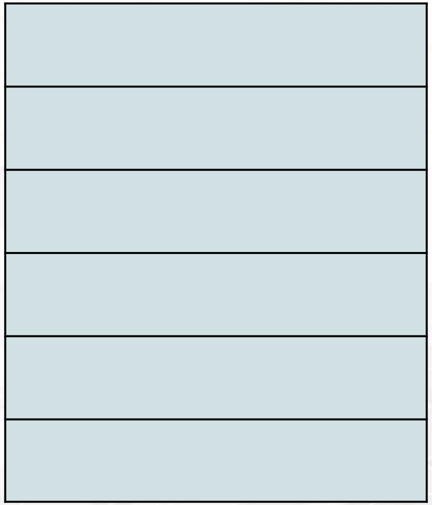



```c
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

Exemplos

4) Imprimir uma lista encadeada de trás para a frente





```
void imprime_rec(Celula *aux) {
  if (aux == NULL)
      return;

  imprime_rec(aux->prox);
  printf("chave = %d\n", aux->item.chave);
}
```

Vantagens x desvantagens

- Vantagens:
  - código mais limpo, elegante;
- Desvantagens:
  - geralmente tem maior consumo de memória;
  - por mais limpo que o código fique, às vezes não compensa em termos de eficiência (como no caso do Fibonacci recursivo, por exemplo);

Exercício

Escrever uma função que leia uma base *b* e um expoente *e*, retornando *b^e*. Não utilize funções nativas da linguagem. Tente resolver o problema primeiramente usando uma função iterativa e então usando uma função recursiva.

Links

- Explicação e vídeo mostrando a pilha de execução da função fatorial recursiva: https://www.embarcados.com.br/recursividade/
- Aplicação interativa da função fatorial recursiva: https://www.cs.usfca.edu/~galles/visualization/RecFact.html
- Gif funcionamento do Fibonacci recursivo: https://giphy.com/gifs/cincia-da-computao-3oGRFGP0Lpo7Rtsveg
- Vídeo da pilha de execução do Fibonacci recursivo: https://youtu.be/dxyYP3BSdcQ