29. Recursividade 1

DIM0320

2015.1

Sumário

Rotinas que usam outra rotinas

Recursividade

- Falta uma estrategia poderosa: **recursividade**
- Esta técnica resolve um problema inicial reduzindo-o em problemas menores **da mesma forma**
- A diferença com qualquer outro método de decomposição vem do fato que a solução do problema será feita a partir de soluções de instâncias menores do problema original.

Um esquema recursivo

Suponha que você tiver que encontrar R$ 1000000 para sua associação.

Primeiras opções

1. você conhece uma pessoa com essa quantidade de dinheiro
2. você chama 10000 amigos e peça R$ 100 a cada um deles

Uma solução recursiva

1. Delegar a 10 pessoas o problema de achar R$100000. É um problema mais fácil mas ainda difícil

Um esquema recursivo

Suponha que você tiver que encontrar R$ 1000000 para sua associação.

Primeiras opções

1. você conhece uma pessoa com essa quantidade de dinheiro
2. você chama 10000 amigos e peça R$ 100 a cada um deles

Uma solução recursiva

1. Delegar a 10 pessoas o problema de achar R$100000. É um problema mais fácil mas ainda difícil
2. As 10 pessoas usam o mesmo esquema (cada uma usa mais 10 pessoas)

Um esquema recursivo

Suponha que você tiver que encontrar R$ 1000000 para sua associação.

Primeiras opções

1. você conhece uma pessoa com essa quantidade de dinheiro
2. você chama 10000 amigos e peça R$ 100 a cada um deles

Uma solução recursiva

1. Delegar a 10 pessoas o problema de achar R$100000. É um problema mais fácil mas ainda difícil
2. As 10 pessoas usam o mesmo esquema (cada uma usa mais 10 pessoas)
3. Assim sucessivamente ...

Um esquema recursivo

Suponha que você tiver que encontrar R$ 1000000 para sua associação.

Primeiras opções

1. você conhece uma pessoa com essa quantidade de dinheiro
2. você chama 10000 amigos e peça R$ 100 a cada um deles

Uma solução recursiva

1. Delegar a 10 pessoas o problema de achar R$100000. É um problema mais fácil mas ainda difícil
2. As 10 pessoas usam o mesmo esquema (cada uma usa mais 10 pessoas)
3. Assim sucessivamente ...
4. ... até chegar ao problema (simples) de achar R$ 10 (ou 1)

Forma abstrata

```
funcao achar_dinheiro(n: inteiro): inteiro
// Contar o número de pessoas necessárias par achar R$ n
var n1: inteiro
inicio
        se n <= 10 entao retorne 1
        senao
                n1 <- n \ 10
                retorne (1 + 10 * achar_dinheiro(n1))
        fimse
fimfuncao
```

Definição no contexto de algoritmos

- Um objeto é recursivo quando é *parcialmente* definido em termos dele mesmo. Ex: Funções recursivas, estruturas recursivas, ...
- Um algoritmo é recursivo se contém ao menos uma chamada explícita (diretamente) ou implícita (indiretamente) a si mesmo.

Princípio de um algoritmo recursivo

Diminuir sucessivamente o problema em outro menor ou mais simples (da mesma forma), até que o problema reduzido possa ser resolvido de forma direta, sem recorrer a si mesmo (condição de parada).

Definir uma função recursiva

1. Escrever a declaração da função
2. Verificar a recursividade da função: deve-se saber calcular o resultado a partir de chamadas **mais simples** da mesma função
3. Prever os casos de base, que não precisam de chamada recursiva
4. Verificar que em todas as chamadas recursivas, os parâmetros (reais) são mais simples que os da função corrente : números menores, cadeias de caractere mais curtas, ...
5. Reconstituir corretamente o valor de retorno da função em função das chamadas recursivas.

Exemplo primitivo: quociente da divisão inteira

Assunto

Escrever uma função recursiva que calcula o quociente da divisão euclidiana (inteira) de um inteiro positivo por um outro, sem usar \

Exemplo primitivo: quociente da divisão inteira

Assunto

Escrever uma função recursiva que calcula o quociente da divisão euclidiana (inteira) de um inteiro positivo por um outro, sem usar \

Solução

1. Declaração

```
funcao quociente (dividendo, divisor: inteiro): inteiro
```

2. Uma definição recursiva é possível: diminuir o dividendo em cada chamada recursiva até contar o número de vezes que ele contem o divisor

Exemplo primitivo: quociente da divisão inteira

Assunto

Escrever uma função recursiva que calcula o quociente da divisão euclidiana (inteira) de um inteiro positivo por um outro, sem usar \

Solução

1. Declaração

```
funcao quociente (dividendo, divisor: inteiro): inteiro
```

2. Uma definição recursiva é possível: diminuir o dividendo em cada chamada recursiva até contar o número de vezes que ele contem o divisor
3. Caso de base: dividendo $<$ divisor $\Rightarrow$ quociente $= 0$

```
se (dividendo < divisor) entao
  retorne 0
fimse
```

Exemplo primitivo: quociente da divisão inteira

Assunto

Escrever uma função recursiva que calcula o quociente da divisão euclidiana (inteira) de um inteiro positivo por um outro, sem usar $\backslash$

Solução

1. Declaração

```
funcao quociente (dividendo, divisor: inteiro): inteiro
```

2. Uma definição recursiva é possível: diminuir o dividendo em cada chamada recursiva até contar o número de vezes que ele contem o divisor
3. Caso de base: dividendo $<$ divisor $\Rightarrow$ quociente $= 0$

```
se (dividendo < divisor) entao
  retorne 0
fimse
```

4. Nos outros casos, subtraímos o divisor ao dividendo (isso fica positivo), não modificamos o divisor. Sempre termina, pois o dividendo (> 0) não pode diminuir infinitamente

Exemplo primitivo: quociente da divisão inteira

Assunto

Escrever uma função recursiva que calcula o quociente da divisão euclidiana (inteira) de um inteiro positivo por um outro, sem usar $\backslash$

Solução

1. Declaração

```
funcao quociente (dividendo, divisor: inteiro): inteiro
```

2. Uma definição recursiva é possível: diminuir o dividendo em cada chamada recursiva até contar o número de vezes que ele contem o divisor
3. Caso de base: dividendo $<$ divisor $\Rightarrow$ quociente $= 0$

```
se (dividendo < divisor) entao
  retorne 0
fimse
```

4. Nos outros casos, subtraímos o divisor ao dividendo (isso fica positivo), não modificamos o divisor. Sempre termina, pois o dividendo (> 0) não pode diminuir infinitamente
5. Em cada chamada recursiva, o divisor é contado uma vez no dividendo, o quociente deve aumentar de 1.

Solução

```
funcao quociente (dividendo, divisor: inteiro): inteiro
inicio
  se (dividendo < divisor) entao
        retorne 0
  fimse
  retorne 1 + quociente(dividendo - divisor, divisor)
fimfuncao
```

Exercício

Modificar o programa anterior para calcular o resto da divisão euclidiana.

Recursão mútua

Assunto

Escrever as duas funções recursivas abaixo:

1. `par` deve retornar `verdadeiro` se o número positivo inteiro dado for **par**, `falso` senão.
2. `impar` deve retornar `verdadeiro` se o número positivo inteiro dado for **impar**, `falso` senão.

Recursão mútua

Assunto

Escrever as duas funções recursivas abaixo:

1. `par` deve retornar `verdadeiro` se o número positivo inteiro dado for **par**, `falso` senão.
2. `impar` deve retornar `verdadeiro` se o número positivo inteiro dado for **impar**, `falso` senão.

Solução

```
funcao par (n : inteiro) : logico
inicio
        retorne (n = 0) ou (impar (n - 1))
fimfuncao

funcao impar (n:inteiro) : logico
inicio
        retorne (n > 0) e ((n = 1) ou (par (n - 1)))
fimfuncao
```

Recursividade: por que ?

- Para cada algoritmo recursivo tem um algoritmo iterativo correspondente (e reciprocamente). Pode ser difícil encontrar o correspondente.

Vantagens

- Compacidade, legibilidade, entendimento
- Algoritmos fáceis de serem implementados
- Ligação forte com matemática

Desvantagens

- Requerem alocação/desalocação de memória (na pilha), mas tem uma forma de recursão, chamada **recursão terminal ou de cauda**, que trata esse problema, com o apoio do compilador

Fatorial

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

Fatorial

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

```
funcao fatorial (n: inteiro): inteiro
inicio
        se n = 0 entao retorne 1
        senao retorne n * (fatorial (n - 1))
        fimse
fimfuncao
```

Sequência de Fibonacci

Definição

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

Leonardo Fibonacci(c. 1170 – c. 1250)

Sequência de Fibonacci

Definição

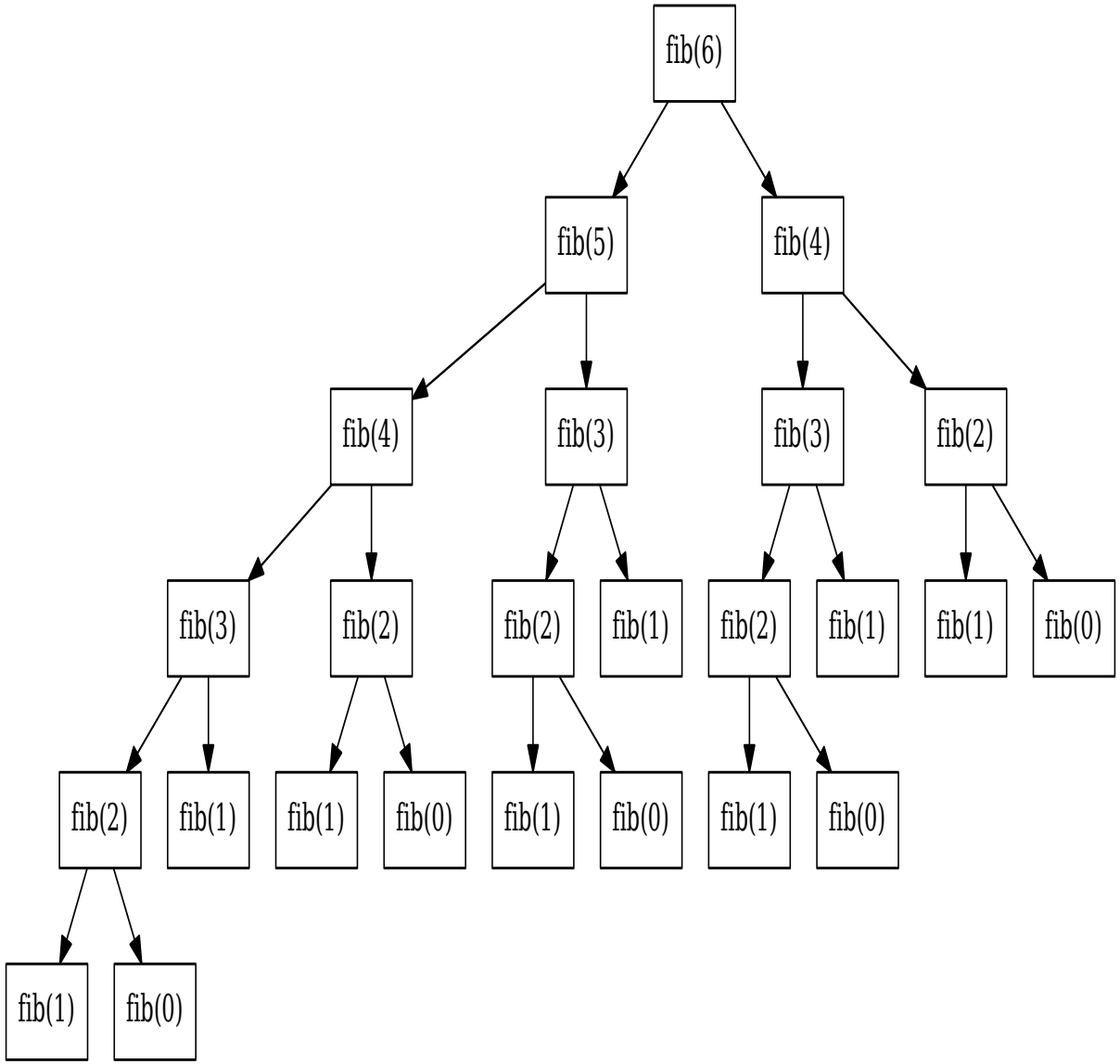
$$F(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$

Leonardo Fibonacci(c. 1170 – c. 1250)

Solução recursiva

```
funcao fib(n: inteiro): inteiro
inicio
  se n = 0 entao retorne 0
  senao
    se n = 1 entao retorne 1
    senao retorne fib(n -1) + fib(n - 2)
    fimse
  fimse
fimfuncao
```

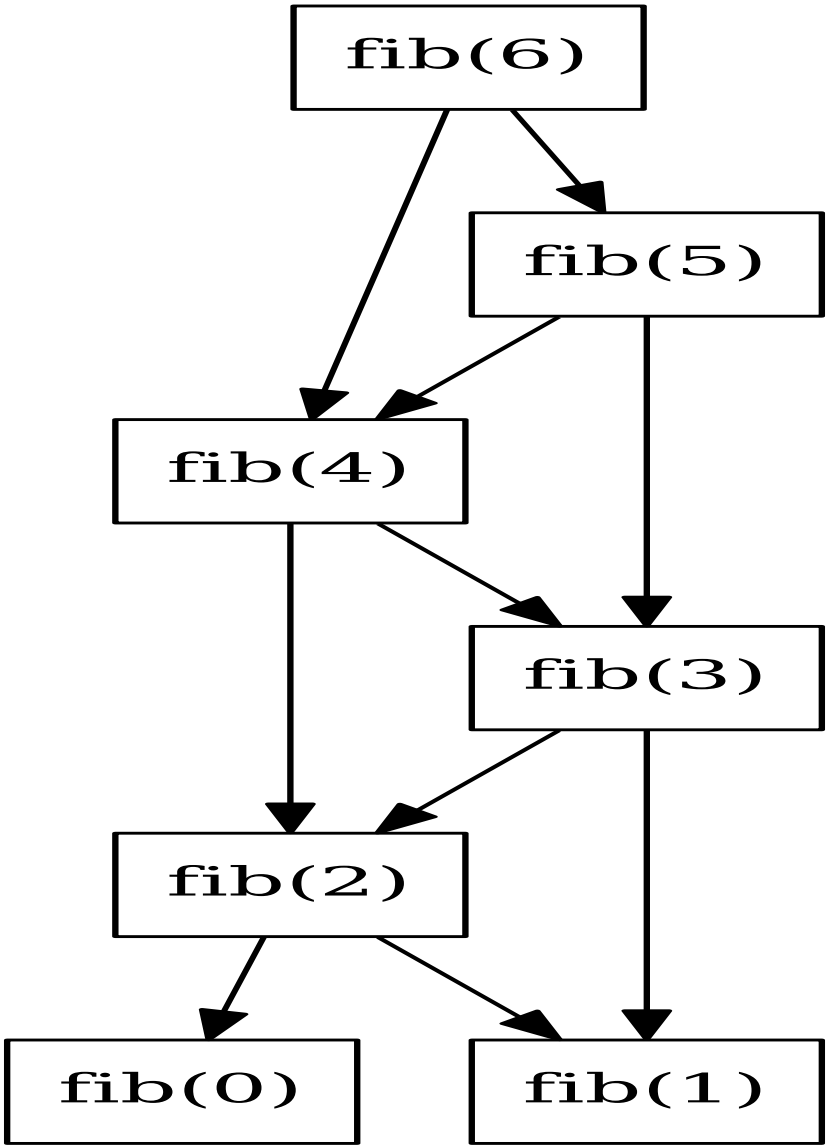
Execução: `fib(6)`



Comparação com a algoritmo iterativo

```
algoritmo "Fibonacci"
var n, i, fibn1, fibn2, tmp : inteiro
inicio
  leia(n)
  enquanto n < 0 faca
    leia(n)
  fimenquanto
  fibn1 <- 0
  fibn2 <- 1
  se n = 0 entao
    fibn2 <- fibn1
  senao
    i <- 1
    enquanto i < n faca
      tmp <- fibn1 + fibn2
      escreval(tmp)
      fibn1 <- fibn2
      fibn2 <- tmp
      i <- i + 1
    fimenquanto
  fimse
  escreval(fibn2)
fimalgoritmo
```

Execução compacta: `fib(6)`



Optimização : recursão terminal

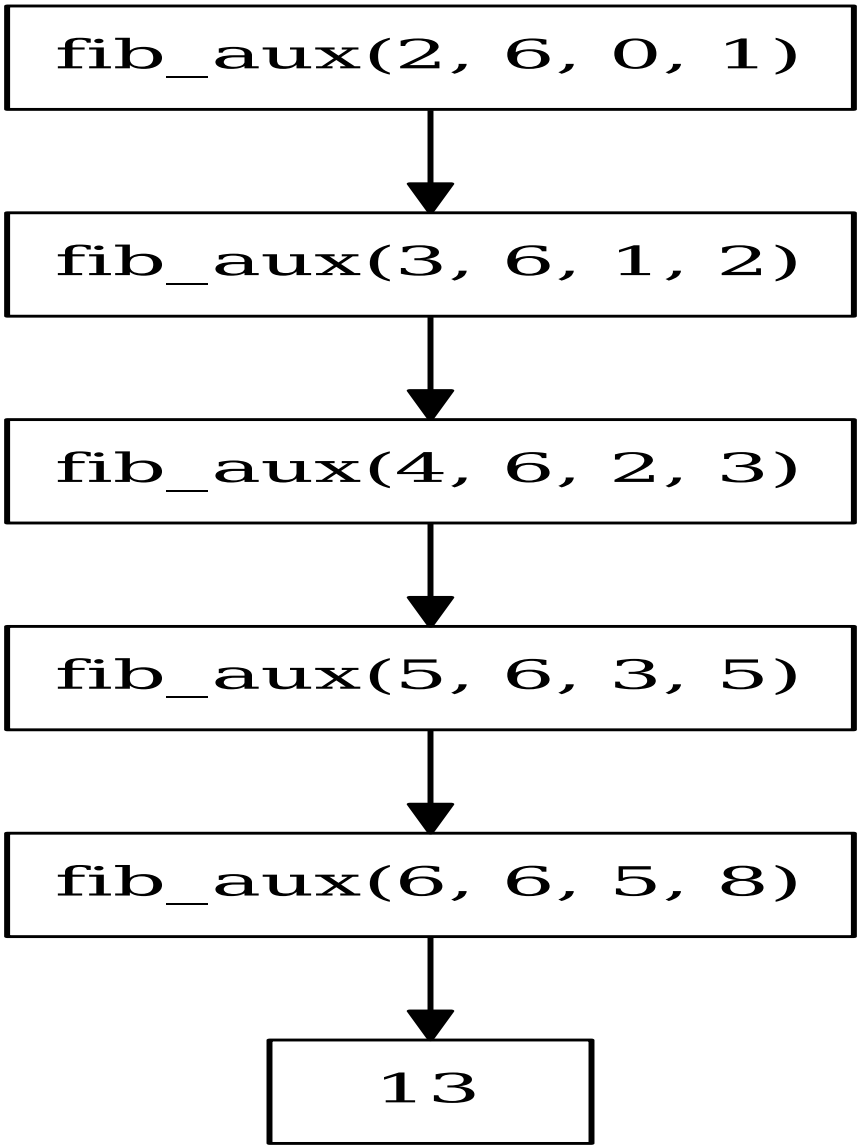
```
algoritmo "foo"
var n: inteiro

funcao fib_aux (m, n, fib1, fib2: inteiro): inteiro
inicio
  se m = n entao
    retorne fib1 + fib2
 senao
    retorne fib_aux(m + 1, n, fib2, fib1 + fib2)
  fimse
fimfuncao

funcao fib(n: inteiro): inteiro
inicio
  se n = 0 entao retorne 0
  senao se n = 1 entao retorne 1
  senao retorne (fib_aux(2, n, 0, 1))
  fimse fimse
fimfuncao

inicio
  leia(n)
  escreval(fib(n))
fimalgoritmo
```

Árvore de chamada



Resumo

Perguntas ?

http://dimap.ufrn.br/~richard/dim0320

Logaritmo inteiro

Assunto

O logaritmo inteiro de x, $x \geq 1$ — notado $elog(x)$ — é o número de vezes que é necessário dividir x por 2 para obter um número ≤ 1.

- Propor uma solução recursiva.
- Generalizar a para outras bases.

MDC

Assunto

Escrever uma função recursiva que calcula o mdc de dois inteiros usando a algoritmo de Euclides.

Potência

Assunto

Escrever uma função recursiva que calcula x^n para um real x e um inteiro n.

Collatz em Siracusa (2014.1)

Assunto

Considere a definição da função u abaixo, definida paras os inteiros positivos.

$$u(n) = \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1 \\ u(\frac{n}{2}) & \text{se } n \text{ é par} \\ u(3n+1) & \text{se } n \text{ é impar} \end{cases}$$

1. Escreva uma função **recursiva** que implemente a especificação acima.
2. Escreva a árvore de chamada de $u(12)$. Detalhe o valor dos parâmetros.