Técnicas de Desenvolvimento de Algoritmos (parte 3)

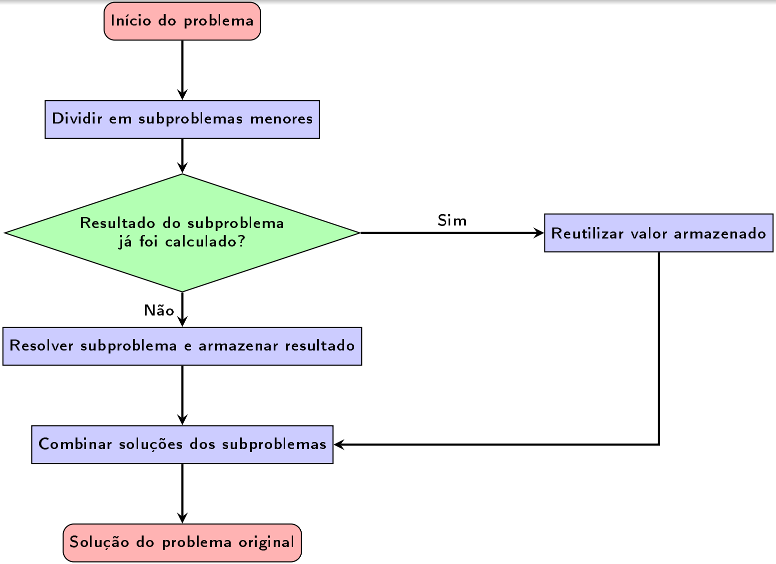
Prof. Marcelo Rosa

Algoritmos e Estrutura de Dados 2 (AE43CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

Sumário

Programação dinâmica

- Constrói a solução para um problema gerando subproblemas menores, resolvendo-os e combinando as soluções (divisão e conquista).
    - No entanto, **só resolve um subproblema se já não o tiver resolvido antes**.
- Usamos uma "tabela" (**memória extra**) para armazenar o resultado do que já foi resolvido.
- O ponto é: **não resolva o mesmo subproblema mais de uma vez!**


Início do problema
Dividir em subproblemas menores
Resultado do subproblema
já foi calculado?
Sim
Reutilizar valor armazenado
Não
Resolver subproblema e armazenar resultado
Combinar soluções dos subproblemas
Solução do problema original

Sumário

Sumário

Problema: Número de Fibonacci

Dado um número inteiro $n \geq 0$, encontrar F_n tal que

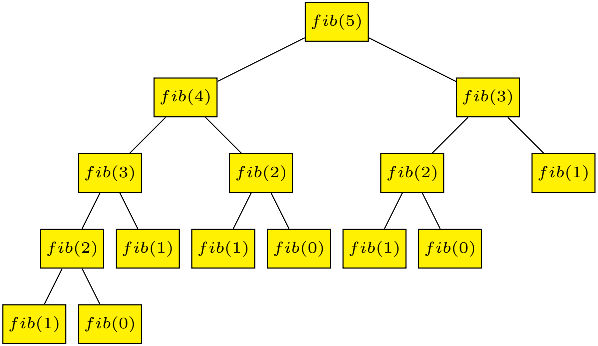
$$F_n = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F_{n-1} + F_{n-2} & \text{se } n \geq 2 \end{cases}$$

- A sequência é 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Programação dinâmica

Número de Fibonacci

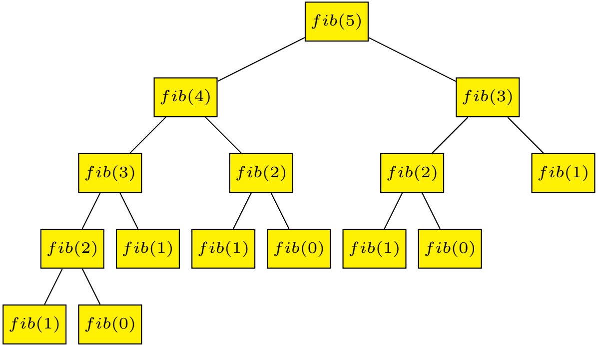
```
function fib(n)
    if n ≤ 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return fib(n − 1) + fib(n − 2)
    end if
end function
```



Programação dinâmica

Número de Fibonacci

```
function fib(n)
    if n ≤ 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return fib(n − 1) + fib(n − 2)
    end if
end function
```



Observações

- Só existem $n+1$ subproblemas a serem resolvidos: $n, n-1, n-2, \dots, 2, 1, 0$
- Cada subproblema é totalmente descrito por um valor $0 \leq i \leq n$.
    - Logo, um vetor de tamanho $n+1$ poderia armazenar esses resultados

Programação dinâmica

Número de Fibonacci: *Top-Down*

function Fib-TopDown(n)
  Criar um vetor global $F[0..n]$
  $F[0] \leftarrow 0;$
  $F[1] \leftarrow 1;$
  for $i = 2$ **até** n, incrementado **do**
    $F[i] \leftarrow -1$
  end for
  return FibRecursivo-TopDown(n)
end function

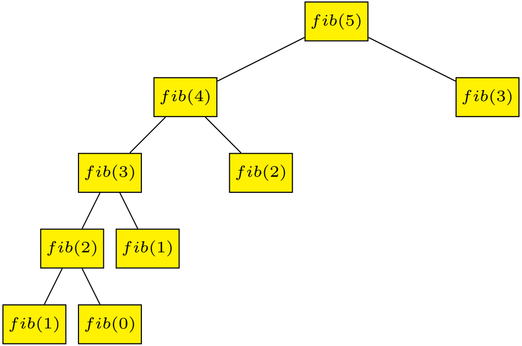
function FibRecursivo-TopDown(n)
  if $F[n] = -1$ **then**
    $F[n] \leftarrow$ FibRecursivo-TopDown($n-1$) + FibRecursivo-TopDown($n-2$)
  end if
  return $F[n]$
end function

Programação dinâmica

Número de Fibonacci: *Top-Down*

function Fib-TopDown(n)
    Criar um vetor global $F[0..n]$
    $F[0] \leftarrow 0;$
    $F[1] \leftarrow 1;$
    for $i = 2$ **até** n, incrementado **do**
        $F[i] \leftarrow -1$
    end for
    return FibRecursivo-TopDown(n)
end function

function FibRecursivo-TopDown(n)
    if $F[n] = -1$ **then**
        $F[n] \leftarrow$ FibRecursivo-TopDown($n-1$) + FibRecursivo-TopDown($n-2$)
    end if
    return $F[n]$
end function

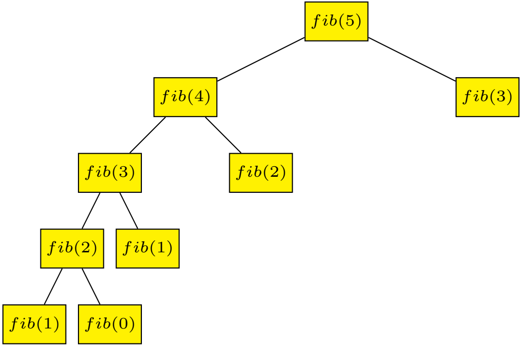


Programação dinâmica

Número de Fibonacci: *Top-Down*

```
function Fib-TopDown(n)
    Criar um vetor global F[0..n]
    F[0] ← 0;
    F[1] ← 1;
    for i = 2 até n, incrementado do
        F[i] ← −1
    end for
    return FibRecursivo-TopDown(n)
end function
```

```
function FibRecursivo-TopDown(n)
    if F[n] = −1 then
        F[n] ← FibRecursivo-TopDown(n − 1) + FibRecursivo-TopDown(n − 2)
    end if
    return F[n]
end function
```



Perceba que cada um dos subproblemas é resolvido somente uma vez durante a execução de `FibRecursivo-TopDown`; todas as operações realizadas levam tempo constante; e que existem n subproblemas (calcular $F_1, F_2, \ldots, F_n$). Assim, o tempo de execução de `Fib-TopDown` é claramente $O(n)$. Isso também pode ser observado pela árvore de recursão.

Programação dinâmica

Número de Fibonacci: *Bottom-Up*

function Fib-BottomUp(n)
    if $n < 0$ **then**
        return 0
    end if
    Seja $F[0..n]$ um vetor de tamanho n
    $F[0] \leftarrow 0;$
    $F[1] \leftarrow 1;$
    for $i = 2$ **até** n, incrementado **do**
        $F[i] \leftarrow F[i-1] + F[i-2]$
    end for
    return $F[n]$
end function

Programação dinâmica

Número de Fibonacci: *Bottom-Up*

function Fib-BottomUp(n)
    if $n < 0$ **then**
        return 0
    end if
    Seja $F[0..n]$ um vetor de tamanho n
    $F[0] \leftarrow 0$;
    $F[1] \leftarrow 1$;
    for $i = 2$ **até** n, incrementado **do**
        $F[i] \leftarrow F[i-1] + F[i-2]$
    end for
    return $F[n]$
end function

- $O(n)$

Top-down

- Forma recursiva natural
- Só faz recursão caso o problema não tenha sido resolvido e salva o que for resolvido na tabela.
    - Em geral tem 2 funções uma para iniciar a “tabela" e outra pra resolver os subproblemas

Top-down

- Forma recursiva natural
- Só faz recursão caso o problema não tenha sido resolvido e salva o que for resolvido na tabela.
    - Em geral tem 2 funções uma para iniciar a “tabela" e outra pra resolver os subproblemas

Bottom-up

- Forma iterativa
- Resolve os subproblemas do menor para o maior, salvando na “tabela"
- Ao resolver um subproblema temos certeza que os menores já foram resolvidos

Top-down

- Forma recursiva natural
- Só faz recursão caso o problema não tenha sido resolvido e salva o que for resolvido na tabela.
  - Em geral tem 2 funções uma para iniciar a “tabela" e outra pra resolver os subproblemas

Bottom-up

- Forma iterativa
- Resolve os subproblemas do menor para o maior, salvando na “tabela"
- Ao resolver um subproblema temos certeza que os menores já foram resolvidos

- Em geral têm o mesmo tempo assintótico: tempo para resolver um subproblema $\times$ quantidade de subproblemas