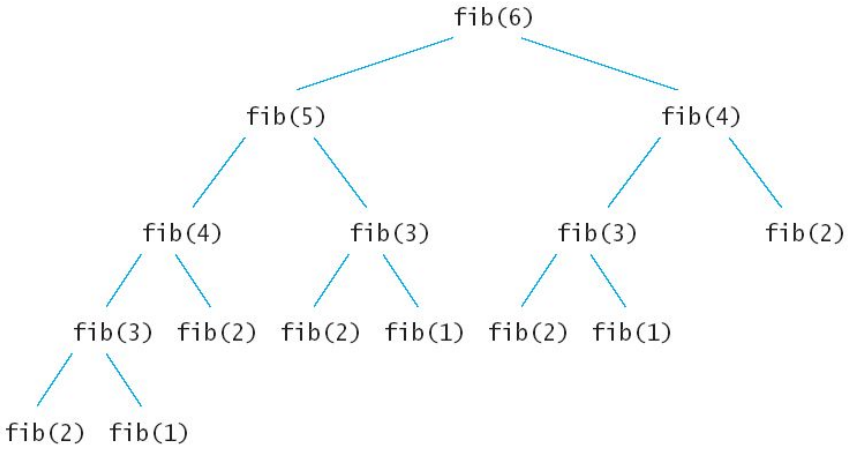
Rod cutting problem

João, Leonardo, Paulo

Problemas sobre programas recursivos

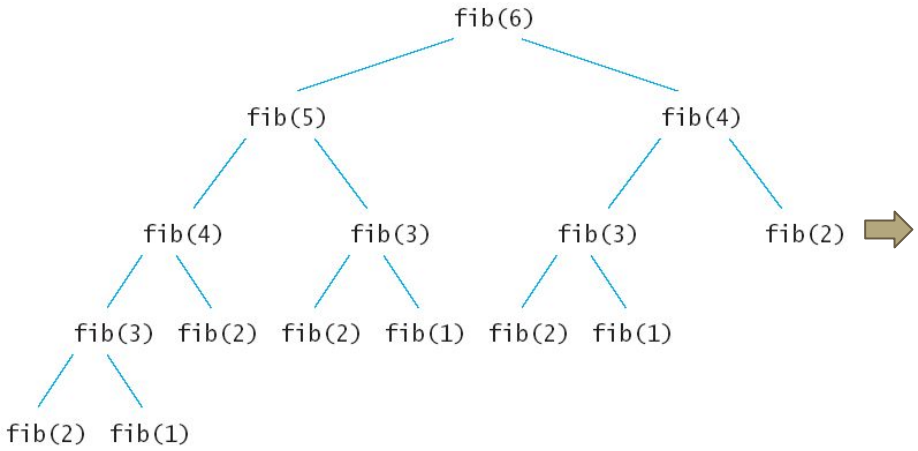
- Podem existir muitas chamadas repetidas
- Podem estourar a pilha com uma recursão muito grande
- Trocam a performance do programa pela facilidade de o programar



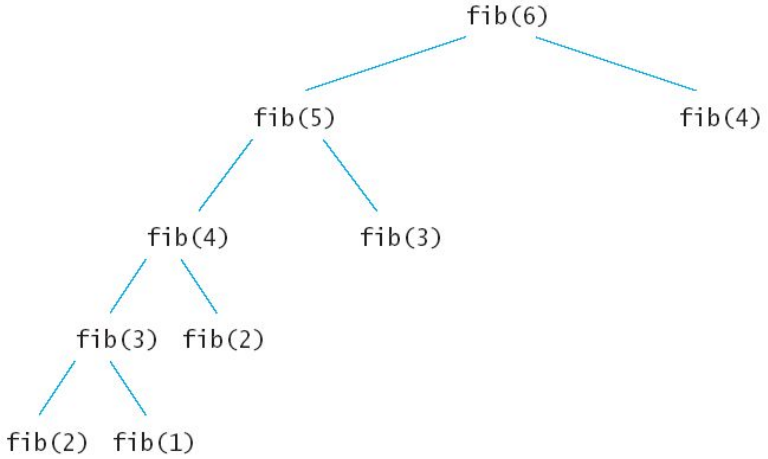
Fonte: BAS | Recursion

Programação dinâmica - O que é?

- Método para resolver problemas recursivos
- Aplicável em problemas onde a solução pode ser descrita por uma equação
- Usa de soluções menores para construir a próxima solução



Fonte: BAS | Recursion



Adaptado de: BAS | Recursion

Programação dinâmica - Técnicas e termos

- Equação de recorrência
  - Dá a definição do problema
  - Define que tipo de chamada será feita
  - Identifica de que maneira a chamada recursiva será feita

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n > 1. \end{cases}$$

Fonte: Token Rock | Fibonacci Numbers

Memoização

- Técnica para memorizar cálculos anteriormente realizados
- Evita a repetição de chamadas
- Contribui para a diminuição no uso da pilha de recursão
- Geralmente mais intuitivo
- Resolve o problema de maneira top-down
    - “Para dominar o mundo, dominarei primeiro meu continente. Para dominar meu continente dominarei primeiro meu país. Para dominar meu país dominarei primeiro minha cidade”

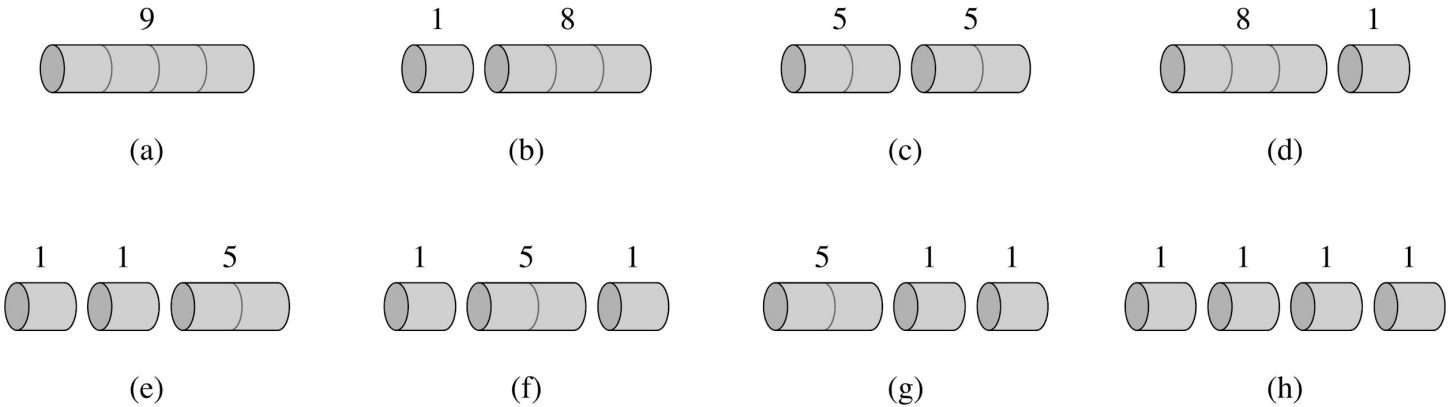
Tabulação

- Técnica para memorizar cálculos anteriormente realizados
- Evita a repetição de chamadas
- Elimina o uso de pilha e de recursão
- Pode ser menos intuitivo
- Resolve o problema de maneira bottom-up
    - “Vou dominar minha minha cidade, após isso isso meu país, após isso dominarei meu continente e então dominarei o mundo”

Rod cutting problem

- Entrada do problema
  - Barra de tamanho - N
  - Conjunto de tamanhos - p
  - Conjunto de preços - r

- Qual a melhor maneira de cortar a barra de modos que você maximize seu lucro?

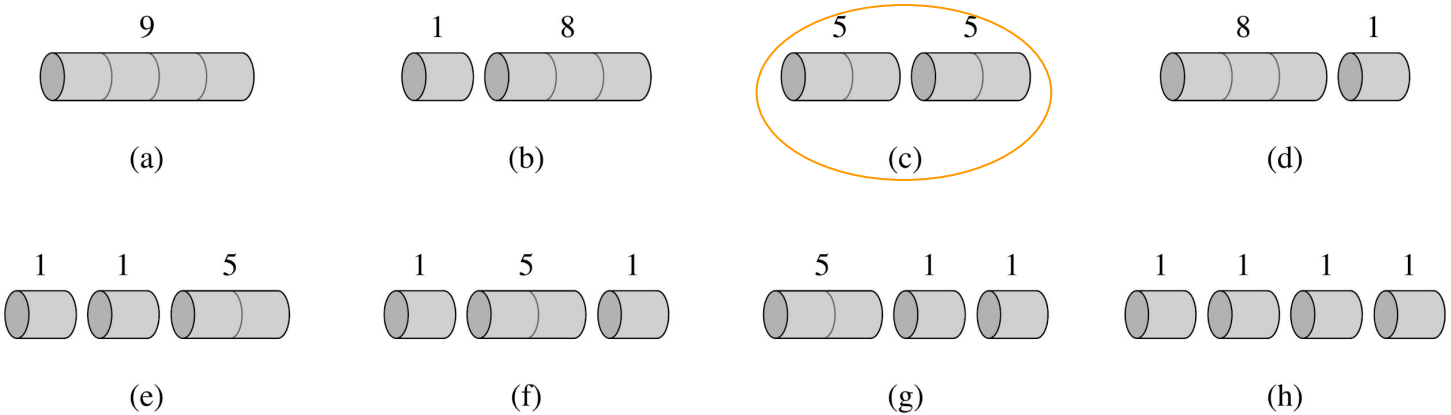


Fonte: Token Rock | Fibonacci Numbers

Rod cutting problem

- Entrada do problema
  - Barra de tamanho - N
  - Conjunto de tamanhos - p
  - Conjunto de preços - r

- Qual a melhor maneira de cortar a barra de modos que você maximize seu lucro?



Fonte: Token Rock | Fibonacci Numbers

Solução ótima

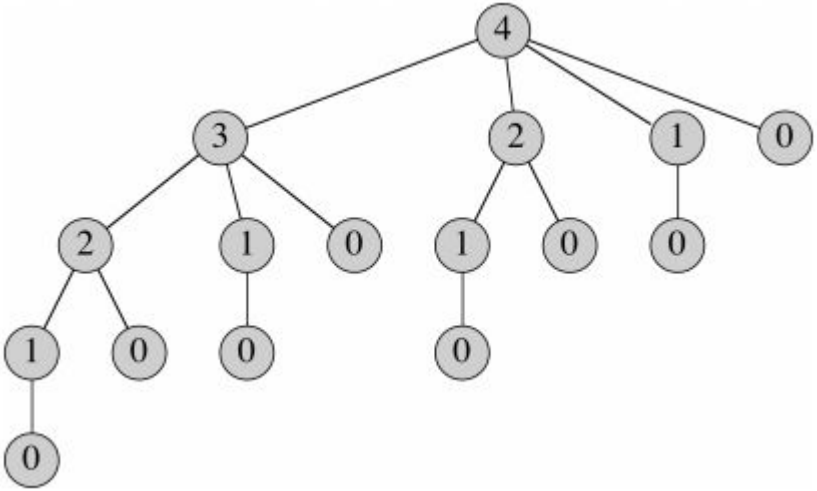
$$r_k = \max(p_i + r_{k-i}) \text{ over all } 1 \leq i \leq k$$

Fonte: Radford | Dynamic Programming - Rod Cutting

r_k : Maior lucro para barra ou corte de tamanho k
p_i : Preço por corte de tamanho i

Exemplo de solução recursiva

```
CutRod(array p, int n):
        if n = 0
                then return 0
        q := -Inf
        for i in 1 .. n loop:
                q := max(q, p(i) + CutRod(p, n-i)
        return q
```



Fonte: Computer Science & Engineering 423/823 Design and Analysis of Algorithms

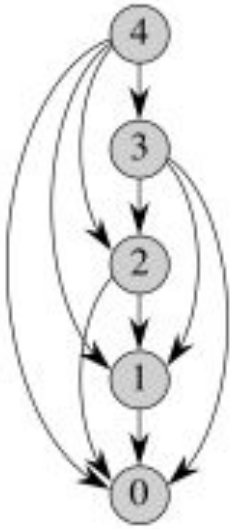
Combinações possíveis

Tamanho de corte	Valor
4	9
1, 3	1 + 8 = 9
2, 2	5 + 5 = 10
3, 1	8 + 1 = 9
1, 1, 2	1 + 1 + 5 = 7
1, 2, 1	1 + 5 + 1 = 7
2, 1, 1	5 + 1 + 1 = 7
1, 1, 1, 1	1 + 1 + 1 + 1 = 4

Exemplo de solução usando PD - Memoization

```
set = [0]

CutRodMemo(array p, int n):
        if n = 0
                then return 0
        if n < length set
                then return set(n)
        q := -Inf
        for i in 1 .. n loop:
                q := max(q, p(i) + CutRodMemo(p, n-i)
        set(n) = q
        return q
```



Fonte: Computer Science & Engineering 423/823 Design and Analysis of Algorithms

Exemplo de solução usando PD - Tabulação

```
CutRodTabulation( array p , int n ): # Bottom Up Solution

    r : array(0..n) # Lucros ótimos por barra de tamanho 0..n
    r(0) := 0

    for j in 1..n loop:
        q := -Inf
        for i in 1..j loop:
            q := max( q , p(i) + r(j-i) )
        end loop
        r(j) := q
    end loop

    return r(n)
```

Exemplo de solução usando PD

i	r_i	Máximo valor
0	r_0	0
1	r_1	$p_1 + r_0$
2	r_2	$p_1 + r_1$, $p_2 + r_0$
3	r_3	$p_1 + r_2$, $p_2 + r_1$, $p_3 + r_0$
4	r_4	$p_1 + r_3$, $p_2 + r_2$, $p_3 + r_1$, $p_4 + r_0$
...	...	

Exemplo

N = 4

Encontrar o maior lucro ao cortar uma barra de tamanho 4.

- $r_0 = 0$
- Encontrar os valores para r_i, $i < 4$.
- $r_4 = \max(p_1 + r_3, p_2 + r_2, p_3 + r_1, p_4 + r_0)$

  $= \max(1 + 8, 5 + 5, 8 + 1, 9 + 0)$

  $= \max(9, 10, 9, 9)$

  $= 10$

Complexidade computacional

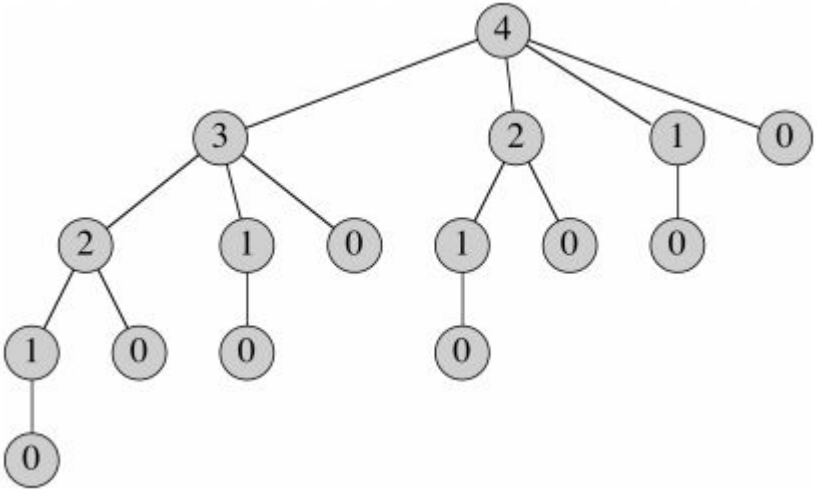
- Solução recursiva: O(2^N)
  - solução ótima
- Solução DP: O(N^2)
  - solução ótima

Solução recursiva - Complexidade

- Sendo T(n) o número de chamadas a função CutRod
- T(0) = 1

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) = 2^n$$

```
Recursive Solution
(10, [2, 2])
Total Calls:  16
Calls:  [8, 4, 2, 1, 1]
```

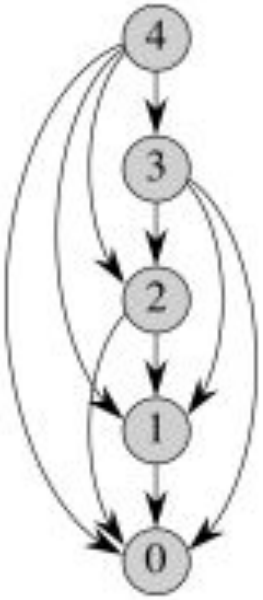


Fonte: Computer Science & Engineering 423/823 Design and Analysis of Algorithms

Solução usando PD - Complexidade

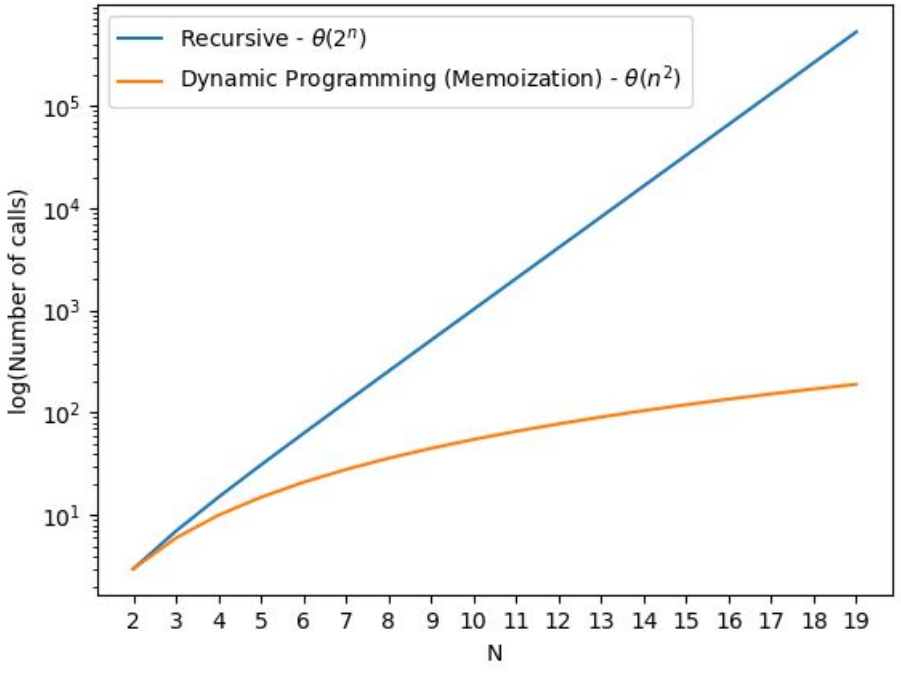
```
Dynamic Solution
(10, [2, 2])
Total Calls:  11
Calls:  [4, 3, 2, 1, 1]
```

$$1 + 2 + 3 + 4 + \cdots + n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \frac{n^2 + n}{2}$$



Fonte: Computer Science & Engineering 423/823 Design and Analysis of Algorithms

Comparação da Complexidade



Exposição do código

Referências

- Radford | Dynamic Programming - Rod Cutting
- Geeks for Geeks | Cutting a Rod
- educative.io | The rod cutting problem
- superwits | Dynamic Programming Technique
- BAS | Recursion
- OpenGenus | Rod cutting problem
- Geeks for Geeks | Tabulation Vs. Memoization