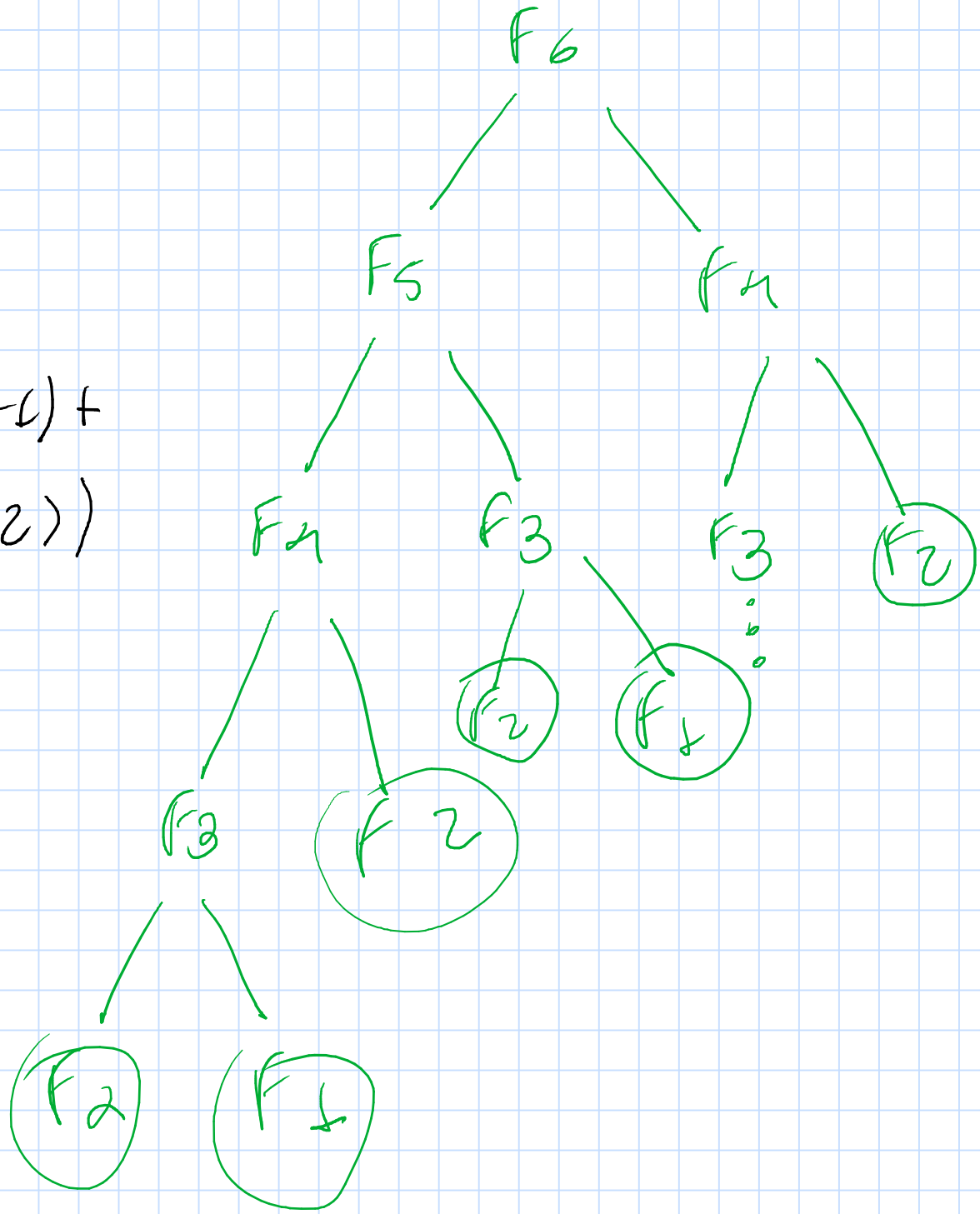
Fibonacci (n)

Se $n \leq 2$

Retorna 1

Retorna (Fibonacci(n-1) +

Fibonacci(n-2))



$T(n) = T(n-1) + T(n-2) + 1$

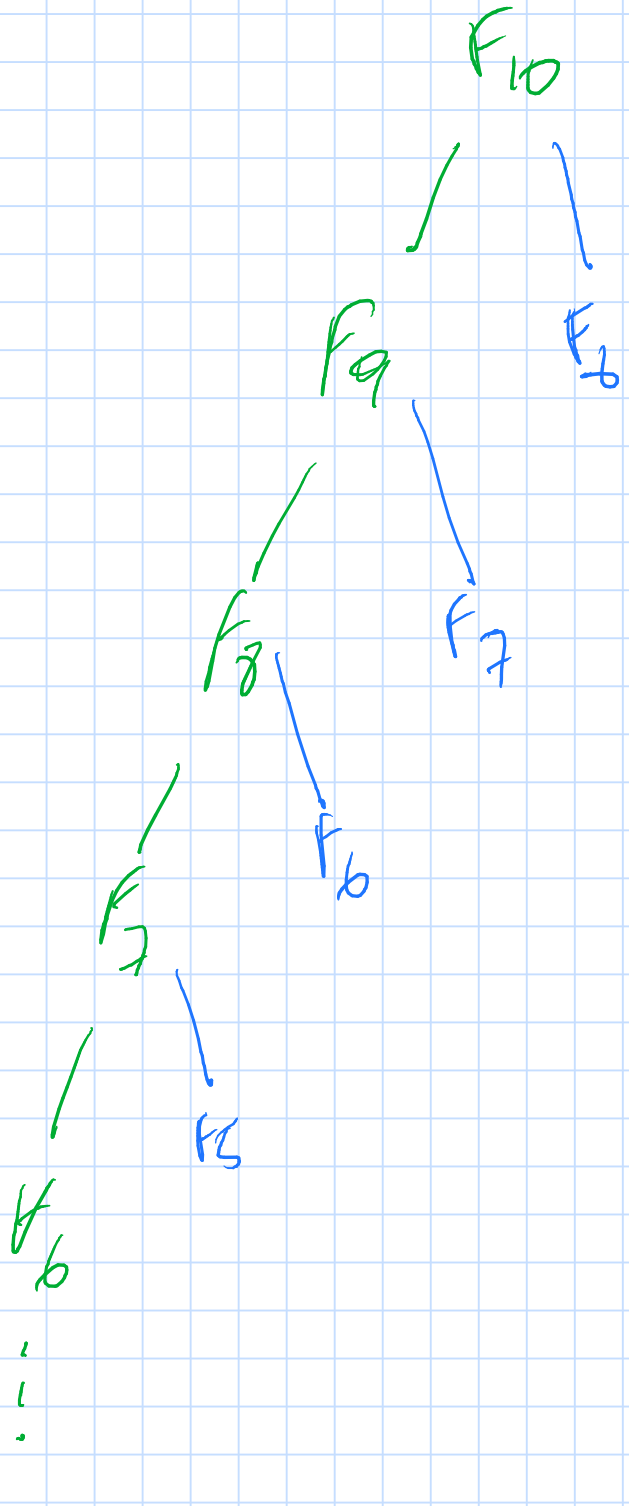
$O(2^n)$

```
Fibonacci_iterativo(n)
    Se n ≤ 2
        Retorna 1
    fib1 ← 1
    fib2 ← 1
    Para i ← 3 até n
        fibi ← fib2 + fib1
        fib1 ← fib2
        fib2 ← fibi
```

$O(n)$

```
Fibonacci_memo (n)
  Se n ≤ 2
     Retorna 1
  Cria vetor auxiliar A[1..n]
  A[1] ← 1
  A[2] ← 1
  Para i ← 3 até n
       A[i] ← 0
  Retorna Fibonacci_rec(n)
```

```
Fibonacci_rec (n)
  Se A[n] = 0
       A[n] ← Fibonacci_rec(n-1) +
              Fibonacci_rec(n-2)
  Retorna A[n].
```



Programação Dinâmica

↳ Tabelas!

↳ Problemas de otimização

↳ Sub estrutura ótima:

- A solução ótima é obtida a partir de soluções ótimas de subproblemas

Prob. otimização:

- Possui muitas soluções
- Cada solução tem um valor?

* Queremos encontrar solução com valor máximo ou mínimo

≠ Divisão e conquista

Em P.D.:

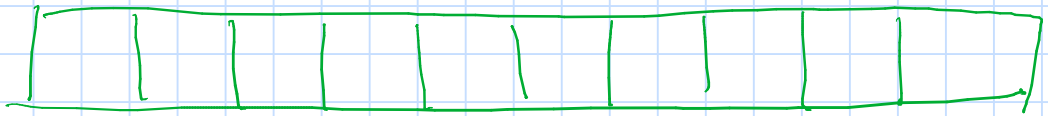
- Subproblemas se sobrescrevem
- Subproblemas não idênticos ao original

(sem trabalho extra p/ combinar)

Problema do corte de hastes

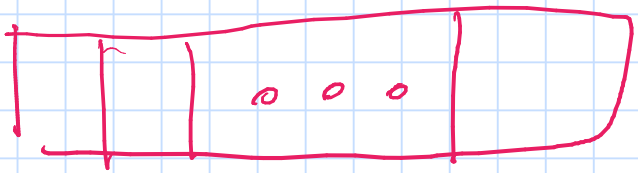
Entrada: Haste de tamanho n (metros ...) e tabela de preços p_i $(1 \le i \le n)$ para um pedaço de haste de tamanho i.

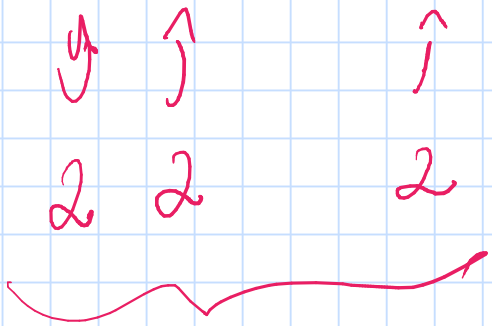
Pergunta: Qual a melhor forma de cortar a haste, de forma a obter o maior lucro possível?



tam. i	1	2	3	4	5	6	7	8	9	10
preço p_i	1	5	8	9	10	17	17	20	24	30

Quantas formas de cortar a haste?





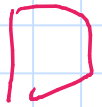
Pensando nos subproblemas...

- barra de tamanho 4

tam. i	1	2	3	4
preço p_i	1	5	8	9

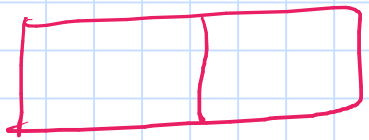
r_i = melhor solução de tamanho i

$i = 1$



$r_1 = 1$

$i = 2$

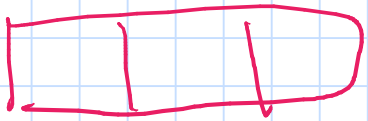


5

2

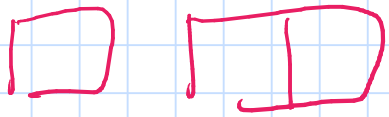
$r_2 = 5$

$i = 3$



8

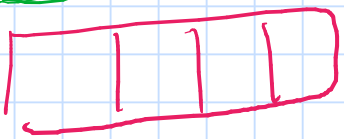
$r_3 = 8$



$1 + 5 = 6$

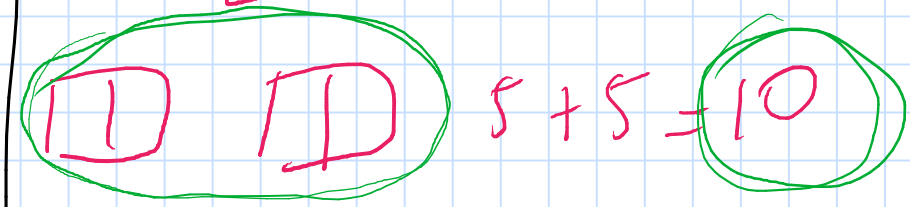
$= 6$

$i = 4$



$= 9$

$1 + 8 = 9$



$5 + 5 = 10$

Recorrência que relaciona os problemas:

$r_n = \max(p_n, r_1 + r_{n-1},$

$r_2 + r_{n-2}, r_3 + r_{n-3}, r_4 + r_{n-4}$

$\cdots r_{n-2} + r_2, r_{n-1} + r_1)$

↳ solução ótima de tamanho n

Outra forma de fazer a recorrência:

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

```
CA_rec(p, n)
  Se n = 0
     Retorna 0
  q ← -∞
  Para i ← 1 até n
     q ← max(q, p[i] + CA_rec(p, n-1))
  Retorna q
```

Programação dinâmica:
Abordagem "bottom-up"

```
Corte_de_Hastes_iter(p, n)
    Crie vetor r[0..n] e s[0..n]
    r[0] ← 0
    Para j ← 1 até n
        q ← -∞
        Para i ← 1 até j
            Se q < p[i] + r[j-i]
                q ← p[i] + r[j-i]
                s[j] ← i
        r[j] ← q
    Retorna r[n] e s[n]
```

$O(n^2)$

Passos para a construção de algoritmos de programação dinâmica:

1. Definição dos subproblemas
2. Obtenção da recorrência que relaciona os problemas
3. Reconhecimento e resolução dos casos base.

```
Corte_de_Hastes_Memoizado (p, n)
  Criar r[0..n]
  Para i ← 0 até n
      r[i] ← -∞
  Retorna CHM (p, n, r)
```

```
CHM (p, n, r)
  Se r[n] ≥ 0
      Retorna r[n]
  Se n = 0
      q ← 0
  Senão q ← -∞
      Para i ← 1 até n
          q ← max(q, p[i] + CHM (p, n-i, r))
  r[n] ← q
  Retorna q
```