Programação Dinâmica

Problema de Otimização

São problemas que admitem vários soluções. Cada solução tem um valor, e nós queremos encontrar uma solução ótima (uma solução de valor máximo ou mínimo, a depender do problema).

- É possível que o problema admita mais de uma solução ótima.

Exemplo de Problema de Otimização

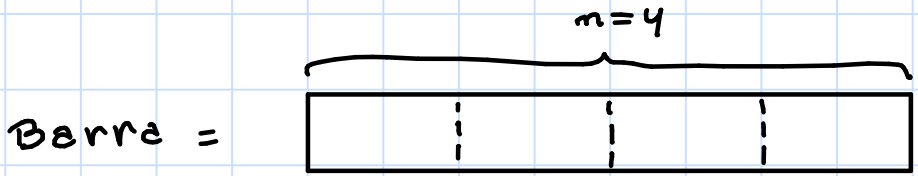
Problema do corte da Barra

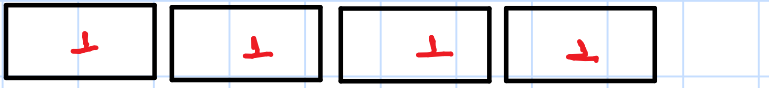
Entrada: um inteiro n representando o compimento em centímetros da barra e um vetor $p[0..n]$ de reais tal que $p[i]$ representa o preço de venda de uma barra de tamanho i

Saída: o maior valor que podemos obter ao vender pedaços da barra dada na entrada (e onde realizar os cortes).

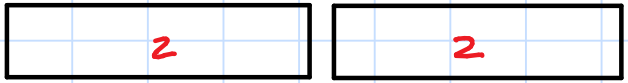
Exemplo do Problema da Barra de Corte

comprimento	1	2	3	4	5	6	7	8	9	10
preço	1	5	8	9	10	17	17	20	24	30

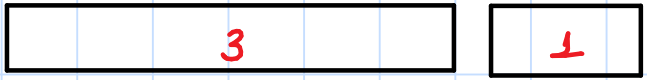




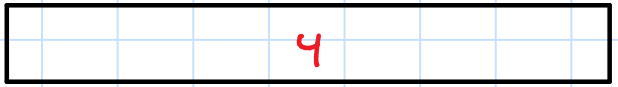
1 + 1 + 1 + 1 = 4



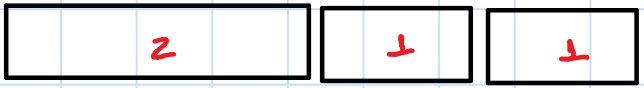
5 + 5 = 10



8 + 1 = 9

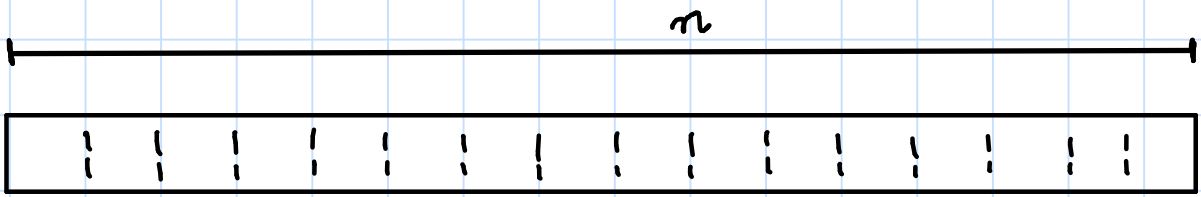


= 9



5 + 1 + 1 = 7

Solução Ótima: 10



- Existem 2^{n-1} formas de cortar a barra

→ Listar todas as soluções não é uma boa ideia

- Seja $S = \{i_1, i_2, \dots, i_k\}$ uma solução viável para o problema de Corte da Barra, onde $i_1, i_2, \dots, i_k$ são os comprimentos dos pedaços.
  - Se S é uma solução para uma barra de tamanho n, então

$$n = i_1 + i_2 + \cdots + i_k$$

- Vamos denotar o valor da solução S por $c(S)$. Assim,

$$c(S) = \sum_{i \in S} p[i]$$

- Vamos denotar por ϕ_n o custo de uma solução ótima para uma barra de tam. n.

↓ não necessariamente ótima

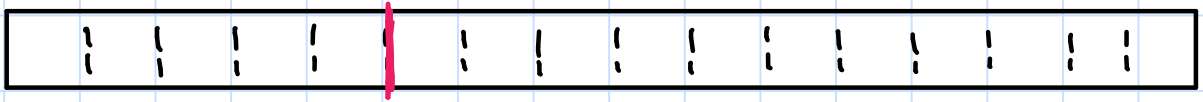
- Seja $S = \{i_1, i_2, \ldots, i_k\}$ uma solução ótima para o corte de uma barra de comprimento n

- A solução ótima não corte a barra ou conta em uma posição $1 \leq i \leq n-1$

solução ótima

$n = \underbrace{i_1 + i_2 + \cdots + i_k}$

$\varphi_n = c(S) = p[i_1] + p[i_2] + \cdots + p[i_k]$

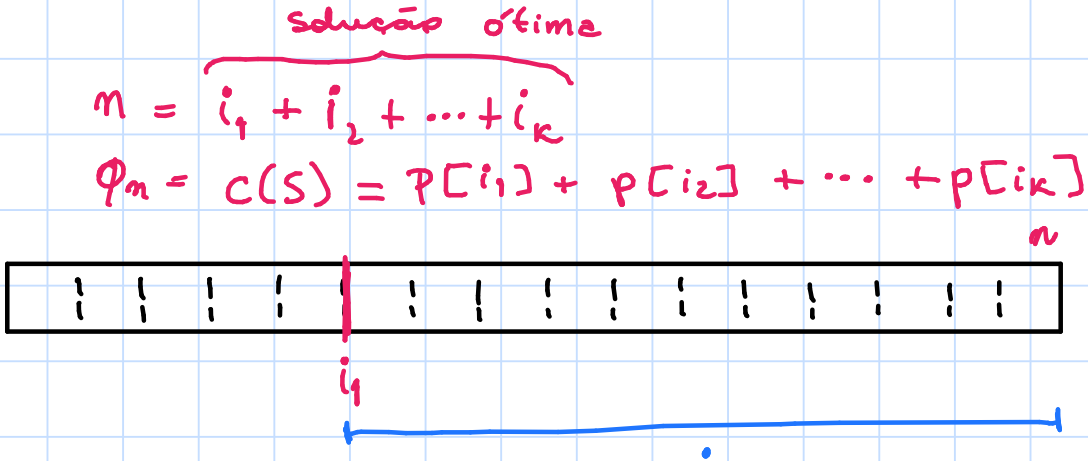


$n - i_1$

$i_2 + i_3 + \cdots + i_k \leftarrow c(S \setminus \{i_1\}) =$

φ_{n-i_1}

- A solução ótima não corta a barra ou corta em uma posição $1 \leq i \leq n-1$



- Suponha, para um absurdo, que $c(S \setminus \{i_1\}) < \phi_{n-i_1}$.
- Seja $S' = \{j_1, j_2, \cdots, j_p\}$ uma solução tal que $\phi_{n-i_1} = c(S')$
- $S^* = S' \cup \{i_1\}$ é uma solução para a barra de tamanho n tal que $c(S^*) = p[i_1] + c(S') > p[i_1] + p[i_2] + \cdots + p[i_k] = \phi_n$

- A solução para o problema do corte de Barra apresenta uma estrutura de subproblema ótimo.

  Se $S = \{i_1, i_2, \ldots, i_k\}$ e $c(S) = \phi_n$,
  então

$$c(S \setminus \{i_1\}) = \phi_{n-i_1}$$
$$c(S \setminus \{i_2\}) = \phi_{n-i_2}$$
$$\vdots$$
$$c(S \setminus \{i_k\}) = \phi_{n-i_k}$$
$$c(S \setminus \{i_1, i_2\}) = \phi_{n-i_1-i_2}$$
$$\vdots$$

- Vamos pensar em uma solução ótima S para o problema de cortar uma barra de tamanho n
- Se $|S|=1$, então não iremos fazer nenhum corte na barra
- se $|S|>1$, então vamos cortar a barra
- Seja $i \in S$
- Suponha que saibamos resolver uma instância menor do problema de corte de Barra.
  - Então podemos testar todos possíveis tamanhos j, $1 \leq j \leq n$, para acertar $j=i$.
    - se $j=n$, a barra n é cortada
  - Por hipótese, sabemos encontrar uma solução S' tal que $c(S') = \phi_{n-i} = c(S \setminus \{i\})$
  - Assim, $c(S' \cup \{i\}) = c(S) = \phi_n$
  - Como saberemos qual $j=i$?

- Como saberemos qual $j = i$?

$$\phi_n = \max_{1 \le j \le n} \left(p[j] + \phi_{n-j} \right)$$

↑ garantidamente vamos testar um $i \in S$

- Seja $S^* = \{i_1, i_2, \cdots, i_k\}$ uma solução ótima, i.e., $c(S^*) = \phi_n$
- $S' = S^* \setminus \{i_1\}$ e $c(S') = \phi_{n-i_1}$
- $c(S^*) = p[i_1] + c(S') = p[i_1] + \phi_{n-i_1}$
- Em algum momento $j = i_1$, portanto

$$\phi_n \geq \max_{1 \leq j \leq n} \left(p[j] + \phi_{n-j} \right) \geq p[i_1] + c(S') = p[i_1] + \phi_{n-i_1} = \phi_n$$

```
Conta-Barra( p[1..n], n )
1  se n = 0
2        retorna 0
3  q ← -∞
4  Para i ← 1 até n
5        q ← max( q, p[i] + Conta-Barra( p, n-i ))
6  retorna q
```

Complexidade

```
Corta-Barra( p[1..n], n )
1  se n = 0
2        retorna 0
3  q ← -∞
4  Para  i ← 1 até n
5        q ← max( q, p[i] + corta-Barra(p, n-i))
6  retorna q
```

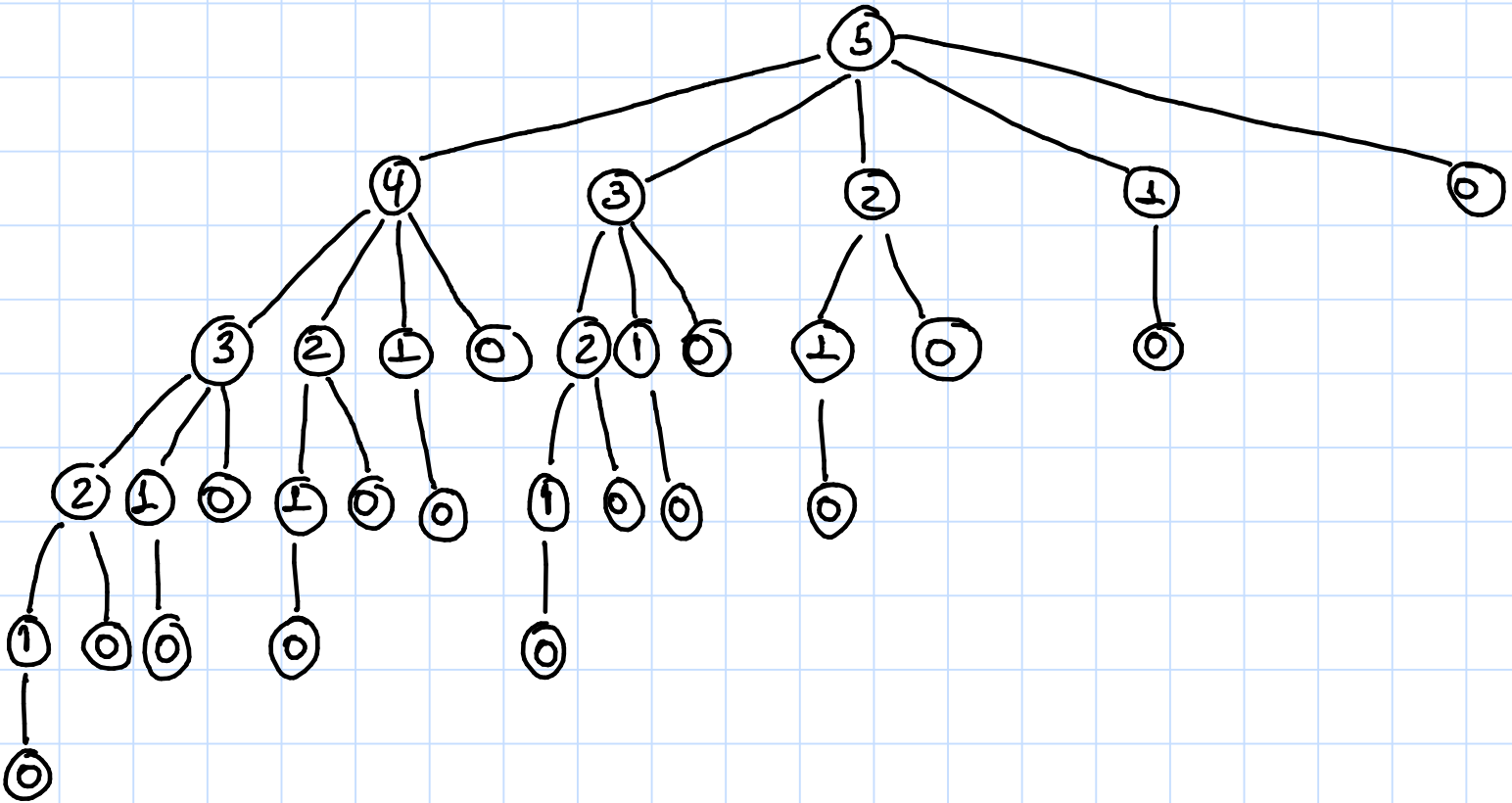
$$T(n) = 1 + \sum_{i=0}^{n-1} T(i) = 2^n$$

2^n ← exponencial no tam. da entrada

Exercício: indução

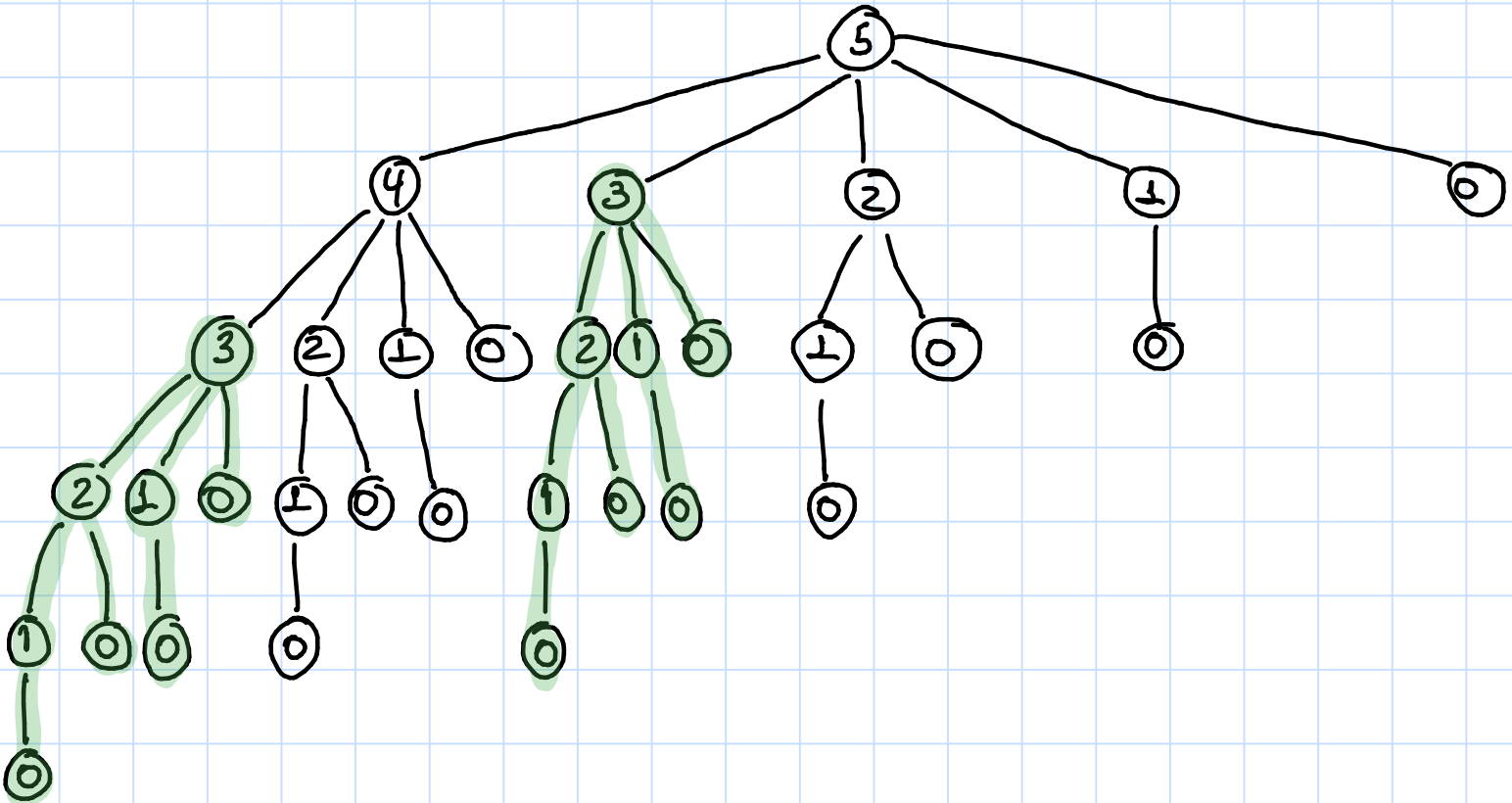
Por que esse algoritmo é Ruim?

A árvore de recursão para n = 5



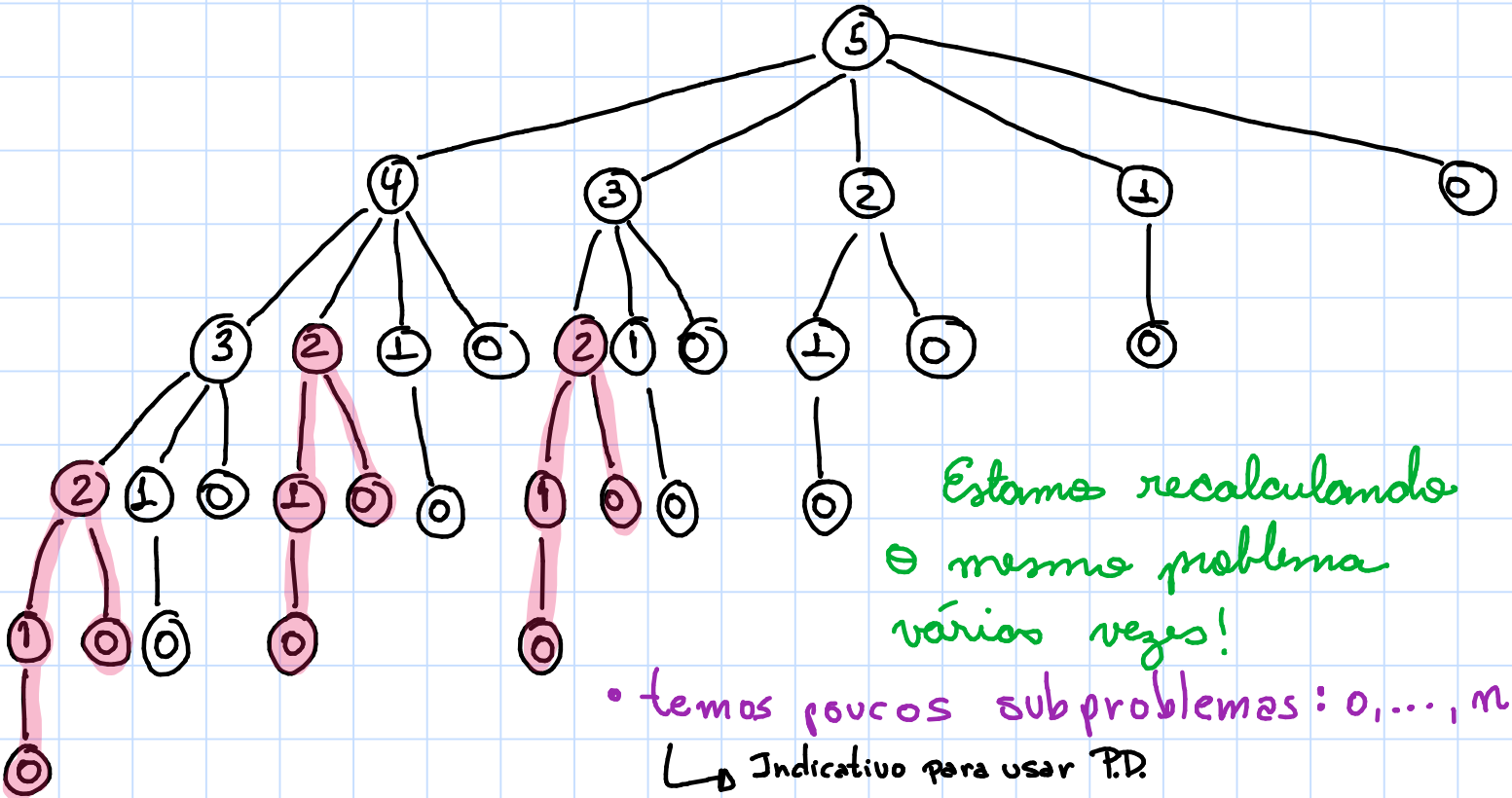
Por que esse algoritmo é Ruim?

A árvore de recursão para n = 5



Por que esse algoritmo é Ruim?

A árvore de recursão para n = 5



Estamos recalculando o mesmo problema várias vezes!

- temos poucos subproblemas: $0, \ldots, n$
  - ↳ Indicativo para usar P.D.

Programação Dinâmica

Ideia:

- Para evitar recalcular a solução de cada subproblema, armazenamos o resultado em uma tabela
  - Sacrificamos memória para economizar processamento

Duas Abordagens

- De cima para baixo / memoization / TOP-DOWN
- De baixo para cima / BOTTOM-UP

De cima para baixo

IDeia :

- Colocamos uma "cache" (tabela) no algoritmo recursivo

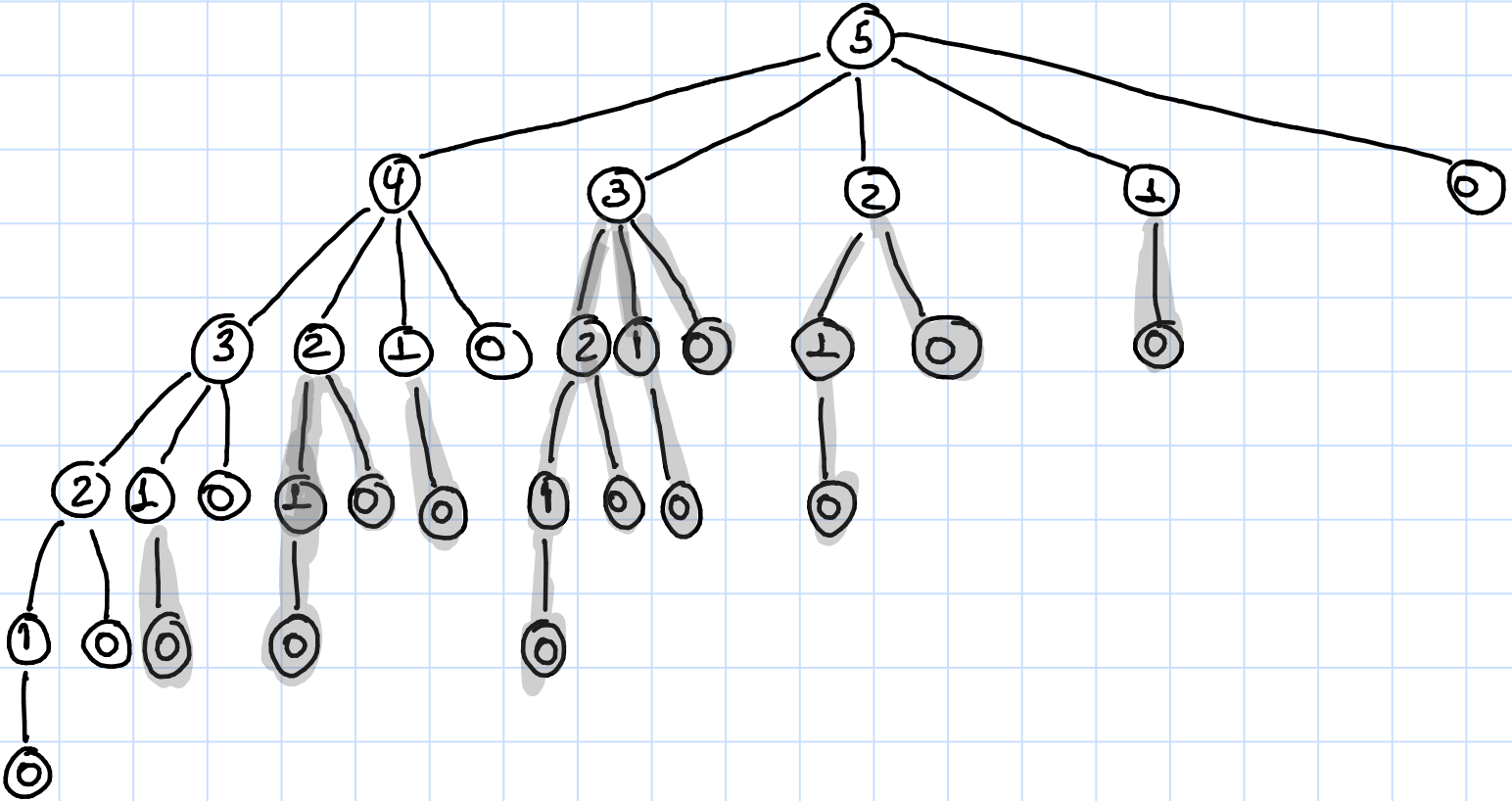
```
Memoized-Corte-Barras(p[1..n], n)
1  Seja r[0..n] um vetor
2  para i ← 0 até n
3          r[i] ← -∞
4  retorna memoized_corte-Barras-AUX(p, r, n)

Memoized-Corte-Barras-AUX(p[1..n], r[0..n], n)
1 Se r[n] > -∞
2       retorna r(n)
3 Se n = 0
4       q ← 0
5 Senão
6       q ← -∞
7       Para i ← 1 até n
8             q ← max(q, p[i] + corta-Barra-AUX(p, r, n-i))
9 r[n] ← q
10 retorna q
```

Por que esse algoritmo é ~~ruim~~?

Ficou Bom

A árvore de recursão para n = 5



Complexidade

- O laço da linhas 2-3 de memoized-corte-Barras executa n vezes e o seu corpo leva $\Theta(1)$. Portanto esse trecho leva $\Theta(n)$.

- O restante do seu tempo é dado pelo tempo de Memoized-corte-Barras-Aux

Memoized-corte-Barras-Aux:

- Por causa da tabela, cada subproblema é resolvido apenas uma vez

Complexidade

Memoized-corte-Barras-Aux:

- Por causa da tabela, cada subproblema é resolvido apenas uma vez
  - temos n subproblemas
- Um subproblema que já foi resolvido, ou quando $n=0$, encerra a chamada imediatamente, pagando $\Theta(1)$
- Quando um subproblema de tamanho n é resolvido pela primeira vez, ele executa o laço das linhas 7-8, gastando $\Theta(n)$
  - Estamos contabilizando apenas a chamada e não o tempo gasto no subproblema.
- Portanto, o tempo gasto é $\sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$

Complexidade

- Assim, o tempo total de

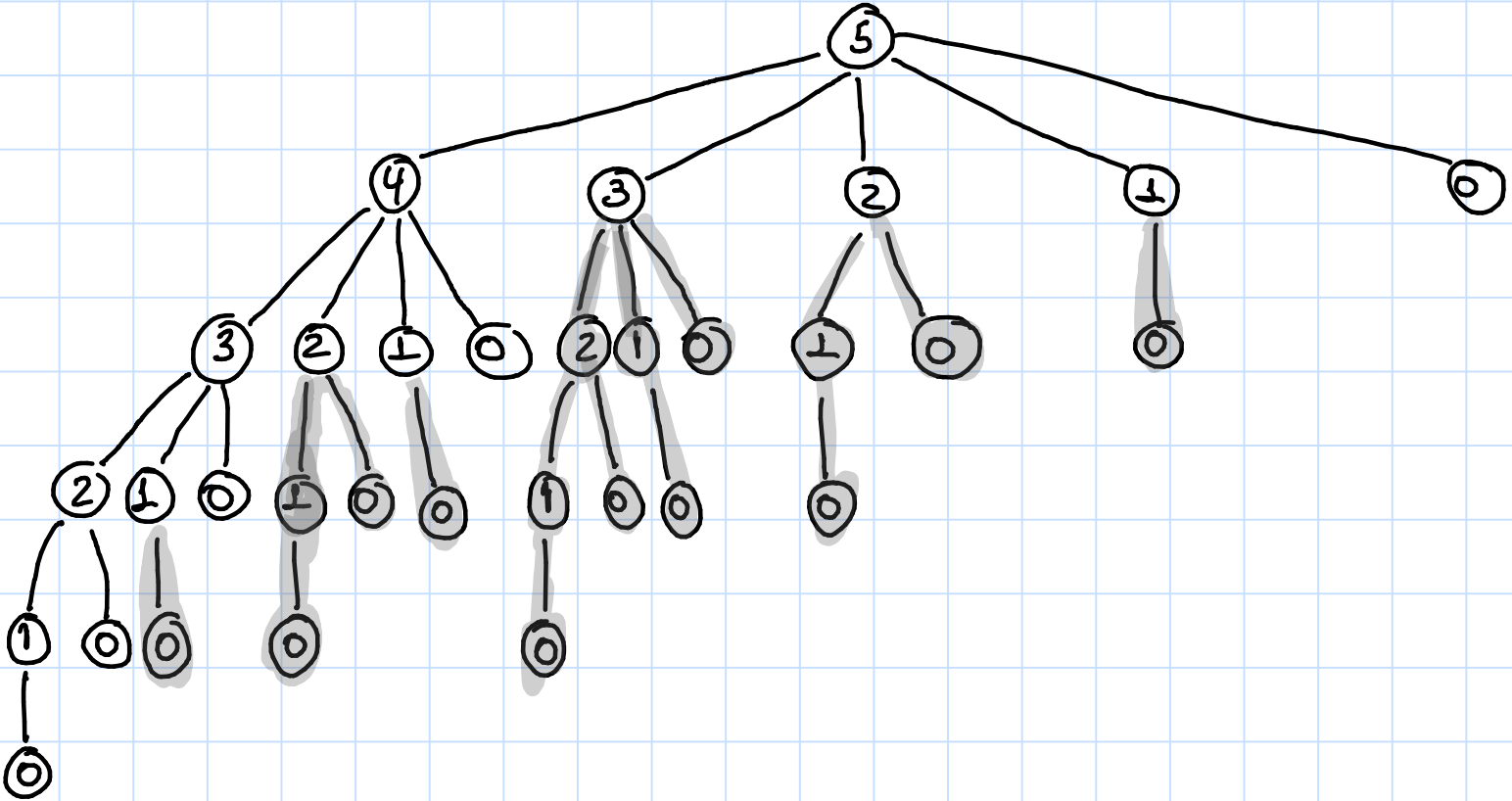
  Memoized-Corte-Barras(p[1..n], n)

  é $\Theta(n) + \Theta(n^2) = \Theta(n^2)$

Por que esse algoritmo é ~~ruim~~?

Ficou Bom

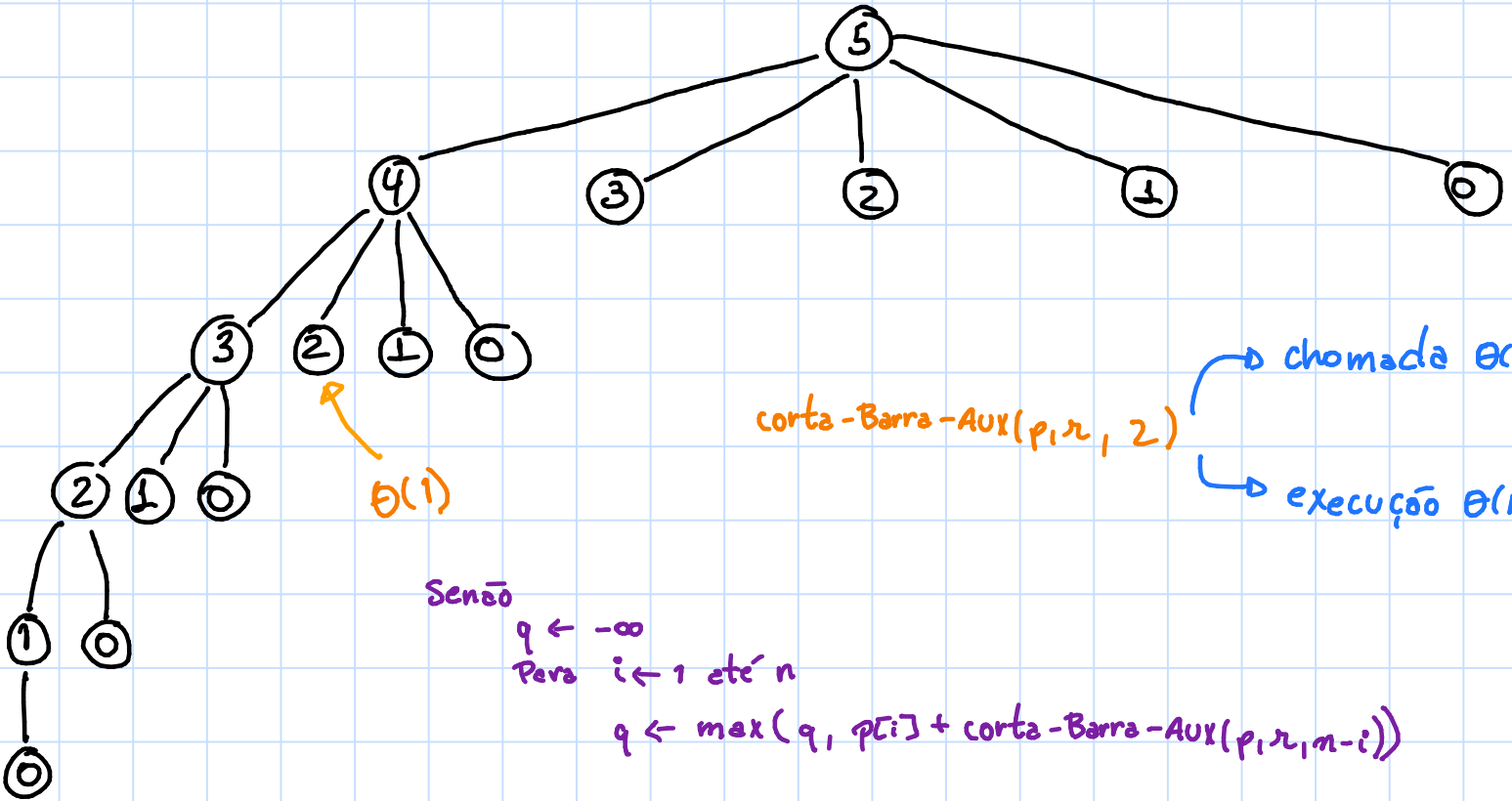
A árvore de recursão para n = 5



Por que esse algoritmo é ~~ruim~~?

Ficou Bom

A árvore de recursão para n = 5



corta-Barra-Aux(p, r, 2) → chamada Θ(1)

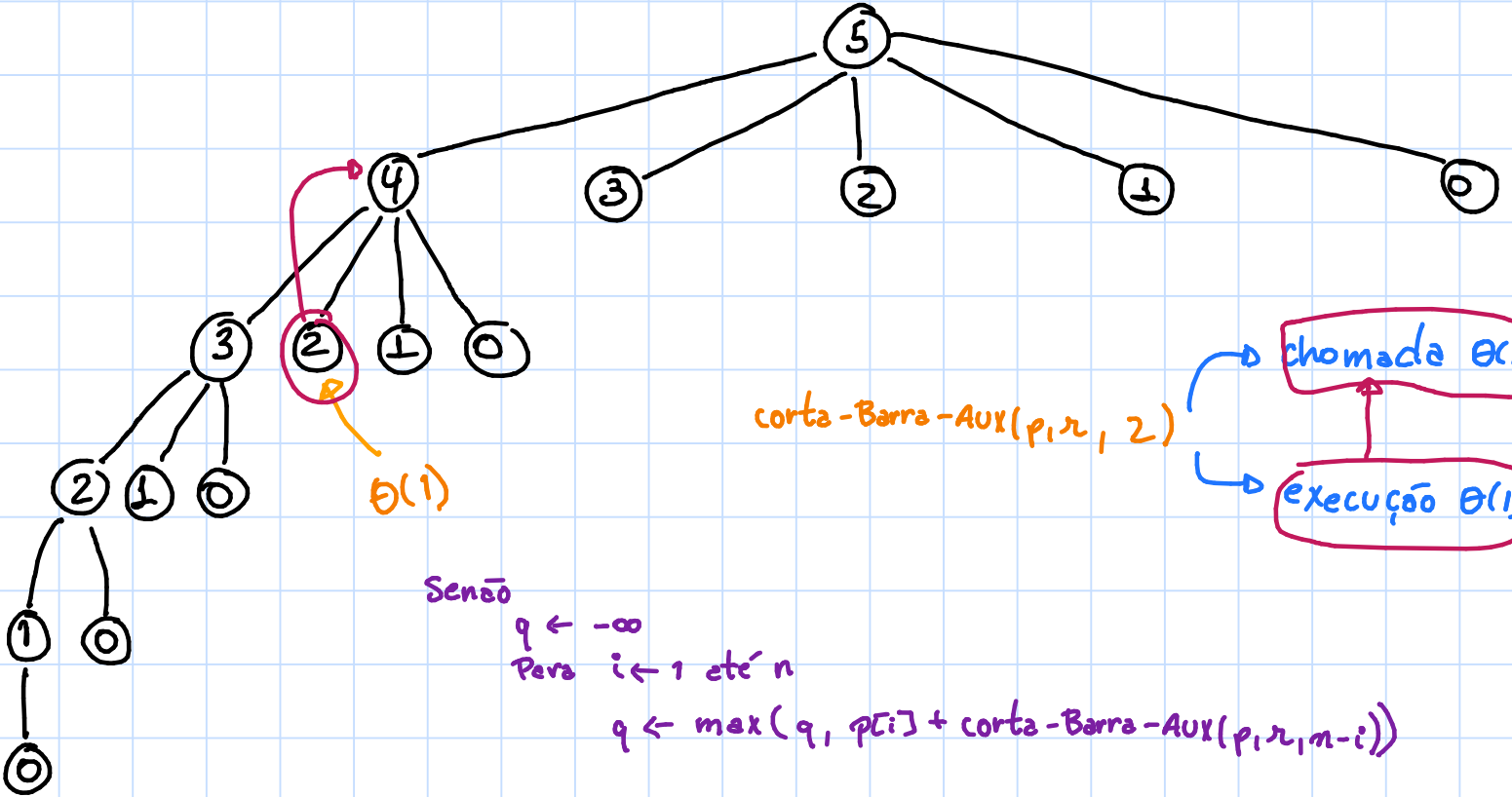
→ execução Θ(1)

```
Senão
    q ← -∞
    Para i ← 1 até n
        q ← max(q, p[i] + corta-Barra-Aux(p, r, n-i))
```

Por que esse algoritmo é ~~ruim~~?

Ficou Bom

A árvore de recursão para n = 5

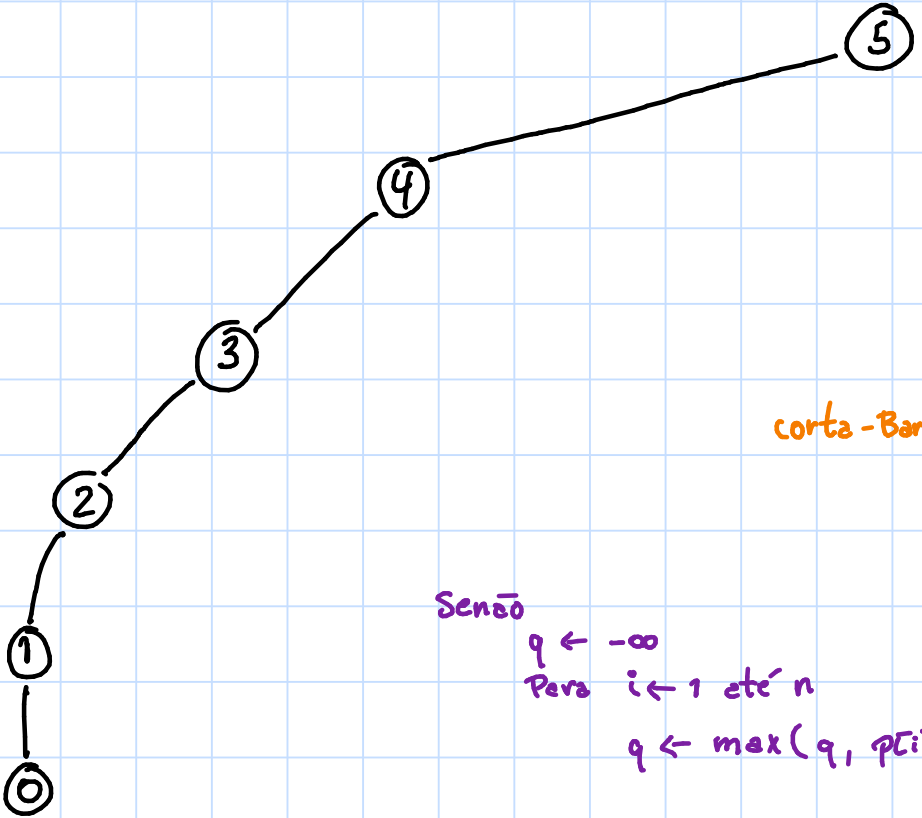



```
Senão
    q ← -∞
    Para i ← 1 até n
        q ← max(q, p[i] + corta-Barra-Aux(p, r, n-i))
```

Por que esse algoritmo é ~~ruim~~?

Ficou Bom

A árvore de recursão para n = 5



corta-Barra-Aux(p, r, 2)

chamada Θ(1)

execução Θ(1)

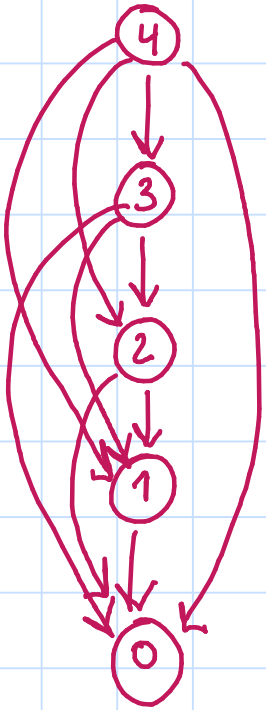
```
Senão
    q ← -∞
    Para i ← 1 até n
        q ← max(q, p[i] + corta-Barra-Aux(p, r, n-i))
```

De Baixo para Cima

Ideia

- Resolver os subproblemas iterativamente
  - Precisamos resolver os subproblemas seguindo uma ordem, do menor para o maior, de forma que quando computarmos a solução de um subproblema todos os subsubproblemas já tenham sido calculados

Corte de Barra de Baixo para Cima



grafo de dependência de subproblemas

Um subproblema de tamanho n precisa das soluções dos subproblemas

$0, 1, \ldots, n-1$

Podemos resolver os problemas na seguinte ordem

$0, 1, 2, \ldots, n$

```
Bottom-UP-Conta-Barra (p, n)

    Seja r[0..n] um vetor
    r[0] ← 0
    para j ← 1 até n
        q ← -∞
        para i ← 1 até j
            q ← max ( q , p[i] + r[ j - i] )
        r[j] ← q
    retorna r[n]
```

Complexidade

Bottom-UP-Conta-Barra (p, n)

```
1  Seja r[0..n] um vetor
2  r[0] ← 0                                   ] Θ(1)
3  para j ← 1 até n
4       q ← -∞                                ] Θ(n)
5       para i ← 1 até j                      ] Θ(j)
6            q ← max(q, p[i] + r[j - i])      ]
7       r[j] ← q                              ] Θ(n)
8  retorna r[n]                               ] Θ(1)
```

Complexidade

- As linhas 2, 8 levam $\Theta(1)$
- As linhas 3, 4, 7 gastam tempo $\Theta(n)$
- As linhas 5-6 levam tempo $\Theta(j)$. Ao longo de toda a execução do programa, temos que essas linhas gastam

$$\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$$

$$\sum_{i=1}^{n} \Theta(i) \leq C \sum_{i=1}^{n} i = C\left[n(n+1)/2\right] = O(n^2)$$

$$\geq C' \sum_{i=1}^{n} i = C'\left[n(n+1)/2\right] = \Omega(n^2)$$

Exemplo

Comprimento	1	2	3	4	5	6	7	8	9	10
preço	1	5	8	9	10	17	17	20	24	30

$n = 4$

Reconstruindo a Solução

- Até agora nossos algoritmos só calculam o valor da solução ótima
  - É fácil recuperar a solução: vemos usar uma tabela adicional para memorizar nossas decisões

```
Bottom-UP-Conta-Barra-Sol(p, n)

    Sejam r[0..n] e s[1..n] vetores
    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
```

Exibindo a Solução

```
Print-Corte ( p, n )
    r, s ← Bottom-up-corte-Barra-sol (p, n)
    Enquanto  n > 0
        print  s[n]
        n ← n - s[n]
```

Exemplo

Comprimento	1	2	3	4	5	6	7	8	9	10
preço	1	5	8	9	10	17	17	20	24	30

i	0	1	2	3	4	5	6
$r[i]$	0	1	5	8	10	13	17
$s[i]$	0	1	2	3	2	2	6

Multiplicação de Matrizes

Multiplicação de Matrizes

$$A_{n \times m} \times B_{m \ell} = C_{n \ell}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ \vdots & & \ddots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1\ell} \\ b_{21} & b_{22} & b_{23} & \cdots & b_{2\ell} \\ \vdots & & \ddots & & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \cdots & b_{m\ell} \end{bmatrix} =$$

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & \cdots & c_{1\ell} \\ c_{21} & c_{22} & c_{23} & \cdots & c_{2\ell} \\ \vdots & & \ddots & & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \cdots & c_{n\ell} \end{bmatrix}$$

onde

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \times b_{kj}$$

Multiplicação de Matrizes

Precisam ser iguais

$$A_{n,m} \times B_{m,l} = C_{n,l}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ \vdots & & \ddots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1l} \\ b_{21} & b_{22} & b_{23} & \cdots & b_{2l} \\ \vdots & & \ddots & & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \cdots & b_{ml} \end{bmatrix} =$$

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & \cdots & c_{1l} \\ c_{21} & c_{22} & c_{23} & \cdots & c_{2l} \\ \vdots & & \ddots & & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \cdots & c_{nl} \end{bmatrix}$$

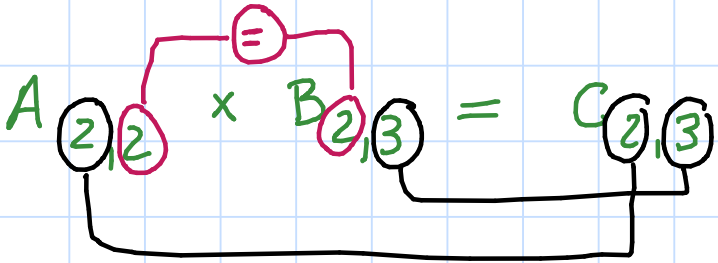
onde

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \times b_{kj}$$

Exemplo

$$A = \begin{bmatrix} 4 & -2 \\ 0 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 2 & -1 \end{bmatrix}$$

$$A \times B = AB = \begin{bmatrix} -2 & 4 & 2 \\ 9 & 6 & -3 \end{bmatrix}$$



```
Multiplicação-Matrizes (A, B)
    Seja C uma matriz A.linhas x B.colunas
    Para i ← 1 até A.linhas
        Para j ← 1 até B.colunas
            C[i, j] ← 0
            Para k ← 0 ate A.colonas
                C[i, j] ← C[i, j] + A[i, k] x B[k, j]
    Devolva C
```

Complexidade

```
Multiplicação-Matrizes(A, B)
    Seja C uma matriz A.linhas x B.colunas
    Para i ← 1 até A.linhas
        Para j ← 1 até B.colunas
            C[i, j] ← 0
            Para k ← 0 até A.colonas
                C[i, j] ← C[i, j] + A[i, k] x B[k, j]
    Devolva C
```

Suponha que $A_{n,m}$ e $B_{m,e}$. Então $O(nem)$

Note que nem é o número de multiplicações

Multiplicação de Matrizes

A multiplicação de matrizes é uma operação associativa

Sejam $A_{n,m}$, $B_{m,\ell}$ e $C_{\ell,r}$ matrizes. Então

$$(\underbrace{A_{n,m} \times B_{m,\ell}}_{AB_{n\ell}}) \times C_{\ell,r} = A_{n,m} \times (\underbrace{B_{m,\ell} \times C_{\ell,r}}_{BC_{m,r}})$$

Multiplicação de Matrizes

Dada uma sequência $\langle A_1, A_2, \ldots, A_n \rangle$ de n matrizes para serem multiplicadas, desejamos calcular o produto

$$A_1 \times A_2 \times \cdots \times A_n$$

Vamos sempre assumir que as matrizes possuem a dimensão correta

Como a multiplicação é uma operação associativa, podemos "parentizar" essa expressão de várias formas

Ex: $\langle A_1, A_2, A_3, A_4 \rangle$

$((A_1 A_2) A_3) A_4)$

$(A_1 (A_2 (A_3 A_4)))$

$((A_1 (A_2 A_3)) A_4)$

$((A_1 A_2) (A_3 A_4))$

Multiplicação de Matrizes

A forma como fazemos a parentização importa!

Ex: $\langle A, B, C \rangle$, onde $A_{10,100}$, $B_{100,5}$ e $C_{5,50}$

$((A\,B)\,C)$

- $10 \cdot 100 \cdot 5 = 5000$ multiplicações para gerar $AB_{10,5}$
- $10 \cdot 5 \cdot 50 = 2500$ multiplicações para gerar $ABC_{10,50}$
- Total: 7500 multiplicações

$(A(BC))$

- $100 \cdot 5 \cdot 50 = 25000$ multiplicações para gerar $BC_{100,50}$
- $10 \cdot 100 \cdot 50 = 50000$ multiplicações para gerar $ABC_{10,50}$
- Total = 75000 multiplicações

Multiplicação de Matrizes

Dada uma sequência $\langle A_1, A_2, \ldots, A_n \rangle$ de matrizes (com dimensões compatíveis para multiplicação) dizemos que a expressão de produto dessa sequência foi <u>totalmente parentizada</u> se ela é uma única matriz ou o produto de duas sequências totalmente parentizadas cercada por parênteses.

Ex $(A_1(A_2(A_3A_4)))$ $((A_1A_2)(A_3A_4))$

$(A_1((A_2A_3)A_4))$ $(((A_1A_2)A_3)A_4)$

Problema da multiplicação de matrizes

Entrada: uma sequência $\langle A_1, A_2, \ldots, A_n \rangle$ de n matrizes, onde para $i = 1, 2, \ldots, n$, a matriz A_i tem dimensões $p_{i-1} \times p_i$.

Saída: o produto $A_1 A_2 \cdots A_n$ totalmente parentizado de forma a minimizar o número de operações.

Será que dá para fazer na força Bruta?

Seja $P(n)$ o número de formas de parentizar uma sequência de n matrizes

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{i=1}^{n-1} P(i) \cdot P(n-i) & \text{se } n \geq 2 \end{cases}$$

$A_1 | A_2\ A_3\ A_4\ A_5 \cdots A_n$

$A_1\ A_2 | A_3\ A_4\ A_5 \cdots A_n$

$A_1\ A_2\ A_3 | A_4\ A_5 \cdots A_n$

$\vdots$

$A_1\ A_2\ A_3\ A_4\ A_5 \cdots | A_n$

Será que dá para fazer na força Bruta?

Seja $P(n)$ o número de formas de parentizar uma sequência de n matrizes

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{i=1}^{n-1} P(i) \cdot P(n-i) & \text{se } n \geq 2 \end{cases}$$

$$P(n) = \Omega(2^n)$$

Base $n = 1$

$P(1) = 1 \geq c \cdot 2^1$

$\frac{1}{2} \geq c$

Passo $n > 1$

next slide

Passo $n > 1$

$$P(n) = \sum_{i=1}^{n-1} P(i) \cdot P(n-i) = \sum_{i=1}^{n-2} P(i) \cdot P(n-i) + P(n-1)P(n-(n-1))$$

$$= P(n-1) + P(n-1) \cdot P(1)$$

$$= 2\,P(n-1)$$

$$\geq 2 \cdot c \cdot 2^{n-1} \quad \text{D por h. i.}$$

$$= c \cdot 2^n$$

Portanto, para $c = 1/2$ e $n_0 = 1$, temos que $P(n) = \Omega(2^n)$

Será que dá para fazer na força Bruta?

Seja $P(n)$ o número de formas de parentizar uma sequência de n matrizes

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{i=1}^{n-1} P(i) \cdot P(n-i) & \text{se } n \geq 2 \end{cases}$$

$$P(n) = \Omega(2^n)$$

↑

$\bar{n}$ vai rolar o teste de todas as possibilidades.

Notação

Dada uma sequência de matrizes $\langle A_1, A_2, \ldots, A_n \rangle$, denotamos por $A_{i \ldots j}$, com $i \leq j$, a matriz resultante do produto $A_i A_{i+1} \cdots A_{j-1} A_j$

Vamos denotar por $\psi_{i \ldots j}$ o número mínimo de operações para o produto $A_i A_{i+1} \cdots A_{j-1} A_j$

Dada uma parentização ϕ de uma sequência de matrizes, vamos denotar por $c(\phi)$ o número de operações resultante dessa parentização

Esse Problema tem Subestrutura Ótima?

$$\langle \overset{1}{A_{30,35}} \mid \overset{2}{B_{35,15}} \mid \overset{3}{C_{15,5}} \mid \overset{4}{D_{5,10}} \mid \overset{5}{E_{10,20}} \mid \overset{6}{F_{20,25}} \mid \overset{7}{H_{25,18}} \rangle$$

$$c(\phi) = \psi_{1..7} = 16325$$

$$\phi = \Big(\big(A_{30,35} \; \big(B_{35,15} \;\; C_{15,5} \big) \big) \; \big(\big(\big(D_{5,10} \;\; E_{10,20} \big) \; F_{20,25} \big) \;\; H_{25,18} \big) \Big)$$

$M_{30,5}$ $\qquad$ $N_{5,18}$

\# operações para fazer $M_{30,5} \cdot N_{5,18}$: $30 \times 5 \times 18 = 2700$

$$\phi = \Big(\big(A_{30,35} \; \big(B_{35,15} \; C_{15,5} \big) \big) \; \big(\big(\big(D_{5,10} \; E_{10,20} \big) \; F_{20,25} \big) \; H_{25,18} \big) \Big)$$

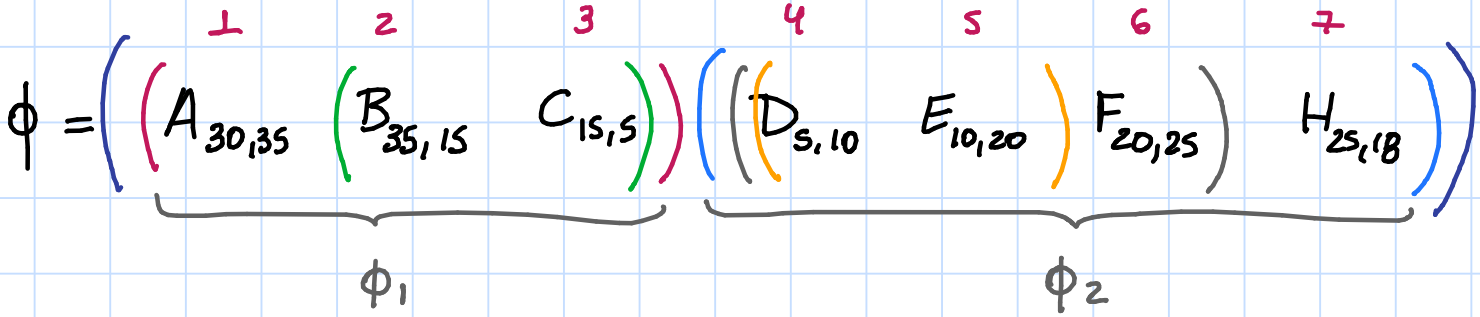
$M_{30,5}$ $\qquad$ $N_{5,18}$

\# operações para fazer $M_{30,5} \cdot N_{5,18}$: $30 \times 5 \times 18 = 2700$

$$\phi = \Big(\big(\; \big) \; \big(\; \big) \Big)$$

$M_{30,5}$ $\qquad$ $N_{5,18}$

não importa como façamos a parentização de $\langle A, B, C \rangle$ e $\langle D, E, F, H \rangle$, sempre faremos 2700 operações.



$$c(\phi_1) = 7875$$

$$c(\phi_2) = 5750$$

$$\psi_{1..7} = c(\phi) = c(\phi_1) + c(\phi_2) + 2700$$

$$= 7875 + 5750 + 2700$$

$$= 16325$$

$$\phi = \Big(\big(A_{30,35} \; \big(B_{35,15} \;\; C_{15,5} \big) \big) \Big(\big(\big(D_{5,10} \;\; E_{10,20} \big) \; F_{20,25} \big) \;\; H_{25,18} \big) \Big)$$

1 2 3 4 5 6 7

ϕ_1 ϕ_2

$$c(\phi_1) = 7875 = \psi_{1..3}$$
$$c(\phi_2) = 5750 = \psi_{4..7}$$

$$\psi_{1..7} = c(\phi) = c(\phi_1) + c(\phi_2) + 2700$$
$$= 7875 + 5750 + 2700$$
$$= 16325$$

Suponha para uma contradição que $c(\phi_1) > c(\gamma) = \psi_{1..3}$

$$\phi = \Big(\underbrace{\big(A_{30,35} \; \big(B_{35,15} \; C_{15,5}\big)\big)}_{\phi_1} \; \underbrace{\big(\big(D_{5,10} \; E_{10,20}\big) \; F_{20,25}\big) \; H_{25,18}}_{\phi_2} \Big)$$

(1 2 3 4 5 6 7)

$$\beta = \Big(\underbrace{\big(\big(A_{30,35} \; B_{35,15}\big) \; C_{15,5}\big)}_{\cancel{\phi_1}\ \gamma} \; \underbrace{\big(\big(D_{5,10} \; E_{10,20}\big) \; F_{20,25}\big) \; H_{25,18}}_{\phi_2} \Big)$$

(1 2 3 4 5 6 7)

$$\psi_{1..7} = c(\phi) = c(\phi_1) + c(\phi_2) + 2700$$

$$> c(\gamma) + c(\phi_2) + 2700 \geq \psi_{1..7}$$

Esse Problema tem Subestrutura Ótima?

- Seja $S = \langle A_1, A_2, \ldots, A_m \rangle$ e seja $\phi = (\phi_1 \ \phi_2)$ uma parentização total ótima, onde ϕ_1 e ϕ_2 são parentizações das sequências $\langle A_1, A_2, \ldots, A_K \rangle$ e $\langle A_{K+1}, \ldots, A_n \rangle$.

$$c(\phi) = \psi_{1..n}$$

- Note que $c(\phi_1) = \psi_{1..K}$ e $c(\phi_2) = \psi_{K+1..n}$
- Suponha para uma contradição que $c(\phi_1) > \psi_{1..K}$
- Seja γ uma parentização da sequência $\langle A_1, A_2, \ldots, A_K \rangle$ tal que $c(\gamma) = \psi_{1..K} < c(\phi_1)$
- Note que os resultados das parentizações γ e ϕ_1 são matrizes com a mesma dimensão.

- Seja $\phi^* = (\gamma\ \phi_2)$ e note que ϕ^* é uma parentização para $\langle A_1, A_2, \ldots, A_n \rangle$

- Seja ℓ o número de operações necessárias para multiplicar as matrizes $A_{1..k}$ e $A_{k+1..n}$

- Assim $c(\phi^*) \geq \psi_{1..n} = c(\phi) = c(\phi_1) + c(\phi_2) + \ell$
$> c(\gamma) + c(\phi_2) + \ell$
$= c(\phi^*)$.

tem substrutura ótima, então, talvez, uma arbordagem recursiva funcione!

Buscando um Algoritmo

- Seja $S = \langle A_1, A_2, \ldots, A_m \rangle$ uma sequência de matrizes
- Seja $p[0..m]$ um vetor de números inteiros tal que a matriz A_i tem dimensão $p[i-1] \times p[i]$

$$\overset{1}{\langle} A_{30,35} \mid \overset{2}{B_{35,15}} \mid \overset{3}{C_{15,5}} \mid \overset{4}{D_{5,10}} \mid \overset{5}{E_{10,20}} \mid \overset{6}{F_{20,25}} \mid \overset{7}{H_{25,18}} \rangle$$

	0	1	2	3	4	5	6	7
p	30	35	15	5	10	20	25	18

Buscando um Algoritmo

- Seja $S = \langle A_1, A_2, \ldots, A_n \rangle$ uma sequência de matrizes
- Se $n = 1$, então $\psi_{1..1} = 0$
- Suponha que $n > 1$ e que saibamos resolver sequências de tam. menor que n
- Seja ϕ uma parentização ótima de S. Então $\phi = (\phi_1, \phi_2)$, onde ϕ_1 deve ser a parentização de $\langle A_1, A_2, \ldots, A_k \rangle$ e ϕ_2 a parentização de $\langle A_{k+1}, \ldots, A_n \rangle$, para algum k.

Buscando um Algoritmo

- Seja ϕ uma parentização ótima de s. Então $\phi = (\phi_1, \phi_2)$, onde ϕ_1 deve ser a parentização de $\langle A_1, A_2, \ldots, A_k \rangle$ e ϕ_2 a parentização de $\langle A_{k+1}, \ldots, A_n \rangle$, para algum k.
  - Não sabemos quem é k
  - sabemos computar o custo $\psi_{1..k} = c(\phi_1)$
    $\psi_{k+1..n} = c(\phi_2)$
  - Podemos testar todos os valores de j, $1 \leq j < n$, para tentar acertar o valor de k
    - Como $k \in \{1, .., n-1\}$, em algum momento faremos $j = k$.

- Podemos testar todos os valores de j, $1 \le j < n$, para tentar acertar o valor de k
  - Como $k \in \{1, .., n-1\}$, em algum momento faremos $j = k$.
    - Como saberemos qndo $j = k$?

$$\psi_{1..n} = c(\phi) = c(\phi_1) + c(\phi_2) + p[0]\,p[k]\,p[n]$$

$$= \psi_{1..k} + \psi_{k+1..n} + p[0]\,p[k]\,p[n]$$

$$\le \boxed{\psi_{1..j} + \psi_{j+1..n}} + p[0]\,p[j]\,p[n]$$

Sabemos calcular

k está aqui!

para qualquer $1 \le j < n$

$$\psi_{1..n} = c(\phi) = c(\phi_1) + c(\phi_2) + p[0]\,p[k]\,p[n]$$

$$= \psi_{1..k} + \psi_{k+1..n} + p[0]\,p[k]\,p[n]$$

$$\leq \boxed{\psi_{1..j} + \psi_{j+1..n}} + p[0]\,p[j]\,p[n]$$

Sabemos calcular

k está aqui!

para qualquer $1 \leq j < n$

$$\psi_{1..n} = \min_{1 \leq j < n} \left(\psi_{1..j} + \psi_{j+1..n} + p[0]\,p[j]\,p[n] \right)$$

```
Matriz-par ( p )
    Se |p| = 2          ← dimensão de 1 matriz
        retorna 0
    Senão
        q ← ∞
        para j ← 1 até n-1
            p1 ← p[1..j]
            p2 ← p[j+1..n]
            q ← min ( q , matriz-par ( p1 ) +
                         matriz-par ( p2 ) +
                         p[0] × p[j] × p[n] )
        Retorna q
```

desnecessário

```
Matriz-par ( p[0..m], r, s)
    Se   r = s
         retorna 0
    senão
         q ← ∞
         para j ← r até s-1
              q ← min ( q, matriz-par ( p, r, j ) +
                           matriz-par ( p, j+1, s ) +
                           p[r-1] × p[j] × p[s] )
         Retorna q
```

Nosso subproblema é ligeiramente diferente.

Dada uma sequência de matrizes $\langle A_1, A_2, \ldots, A_n \rangle$ e inteiros r e s, $1 \le r \le s \le n$, queremos o menor # de operações para o produto $A_r A_{r+1} \cdots A_s$

matriz-par$(p, r, s) = \psi_{r \ldots s}$

Complexidade

$n = s - r + 1$

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

Hipótese $T(n) \geq \frac{1}{2} \cdot 2^n$

Base $n = 1$

$T(1) = 1 = \frac{1}{2} \cdot 2^1$

Passo

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

$$\geq 1 + \sum_{j=1}^{n-1} 2^j = 1 + 2^n - 1 = 2^n \geq \frac{2^n}{2}.$$

Quantos subproblemas temos?

$$\langle \underset{1}{A_{30,35}}, \underset{2}{B_{35,15}}, \underset{3}{C_{15,5}}, \underset{4}{D_{5,10}}, \underset{5}{E_{10,20}}, \underset{6}{F_{20,25}}, \underset{7}{H_{25,18}} \rangle$$

$$\psi_{1..1}, \psi_{1..2}, \psi_{1..3}, \cdots, \psi_{1..7}$$
$$\psi_{2..2}, \psi_{2..3}, \cdots, \psi_{2..7}$$
$$\psi_{3..3}, \cdots, \psi_{3..7}$$
$$\vdots$$

Seja $\langle A_1, A_2, \cdots, A_n \rangle$ uma sequência de matrizes

subproblemas: $n + \binom{n}{2}$

n: # subproblemas de índices iguais

$\binom{n}{2}$: # subproblema de índices $\neq$

Quantos subproblemas temos?

Seja $\langle A_1, A_2, \ldots, A_n \rangle$ uma sequência de matrizes

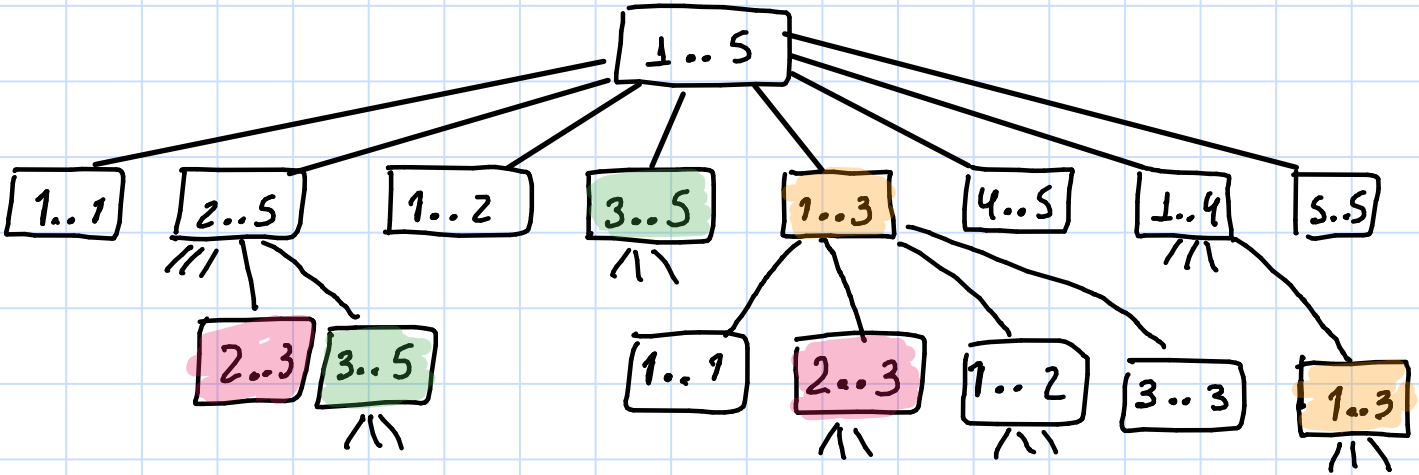
subproblemas: $n + \binom{n}{2}$

n: # subproblemas de índices iguais

$\binom{n}{2}$: # subproblema de índices $\neq$

$$n + \binom{n}{2} = n + \frac{n(n+1)}{2} = n^2 + \frac{3}{2}n = \Theta(n^2)$$

Temos sobreposição de Problemas?



Bottom-UP: ordem para atacar os sub prob.

$$\psi_{1..n} = \min_{1 \le j < n} \left(\psi_{1..j} + \psi_{j+1..n} + p[0]\, p[j]\, p[n] \right)$$

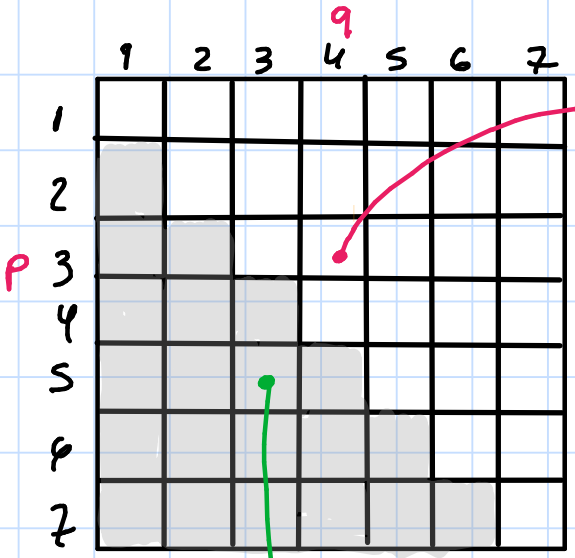
$$\psi_{r..s} = \min_{r \le j < s} \left(\psi_{r..j} + \psi_{j+1..s} + p[r-1]\, p[j]\, p[s] \right)$$

Bottom-UP: ordem para atacar os sub prob.

$$\psi_{r..s} = \min_{r \le j < s} \left(\underbrace{\psi_{r..j}}_{m[r][j]} + \underbrace{\psi_{j+1..s}}_{m[j+1][s]} + p[r-1]\, p[j]\, p[s] \right)$$

Bottom-UP: Ordem para atacar os sub prob.

Obs: a parte cinza da matriz não será usada!



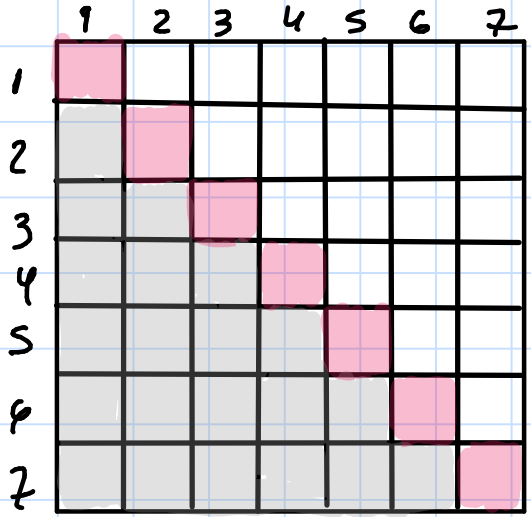
$m[p][q] = \psi_{p..q}$

$p > q \Rightarrow A_p, A_{p+1}, \ldots A_q$ é $\emptyset$

Bottom-UP: Ordem para atacar os sub prob.

Tamanho dos subproblemas: $n = s - r + 1$

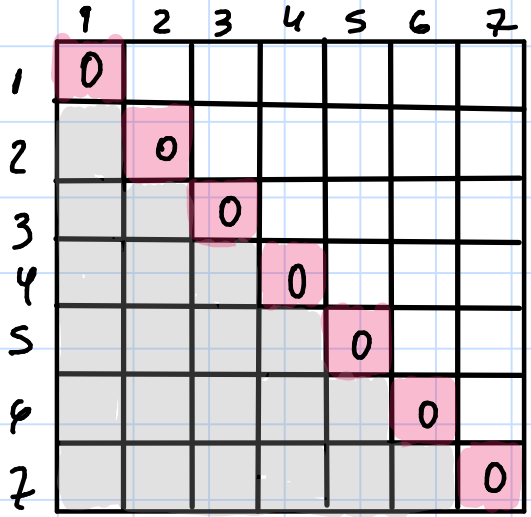
- $n = 1 \Rightarrow 1 = s - r + 1 \Leftrightarrow r = s$



Bottom-UP: Ordem para atacar os sub prob.

Tamanho dos subproblemas: $n = s - r + 1$

- $n = 1 \Rightarrow 1 = s - r + 1 \Leftrightarrow r = s$



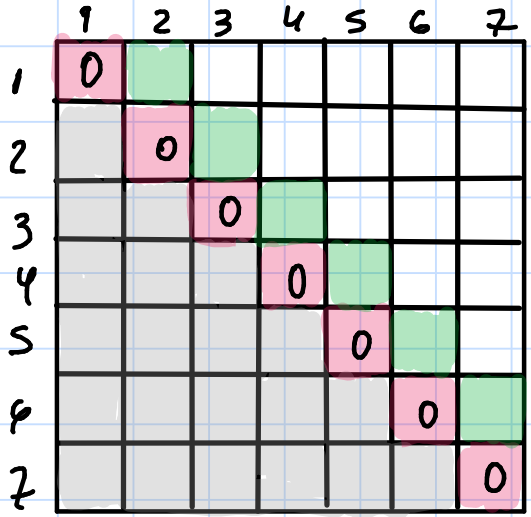
$m[i][i] = 0$

$\forall\ 1 \leq i \leq n$

Bottom-UP: Ordem para atacar os sub prob.

Tamanho dos subproblemas: $m = s - r + 1$

- $m = 2 \Rightarrow 2 = s - r + 1 \Leftrightarrow 1 = s - r$



Bottom-UP: Ordem para atacar os sub prob.

$$\langle \underset{1}{A_{30,35}} , \underset{2}{B_{35,15}} , \underset{3}{C_{15,5}} , \underset{4}{D_{5,10}} , \underset{5}{E_{10,20}} , \underset{6}{F_{20,25}} , \underset{7}{H_{25,18}} \rangle$$

$$\psi_{r..s} = \min_{r \le j < s} \left(\psi_{r..j} + \psi_{j+1..s} + p[r-1]\, p[j]\, p[s] \right)$$

	1	2	3	4	5	6	7
1	0	15750					
2		0					
3			0				
4				0			
5					0		
6						0	
7							0

$$\psi_{12} = \min_{1 \le j < 2} \left(\psi_{1..1} + \psi_{2..2} + 30 \cdot 35 \cdot 15 \right)$$

$$= \min_{1 \le j < 2} \left(0 + 0 + 15750 \right)$$

$$= 15750$$

Bottom-UP: Ordem para atacar os sub prob.

$$\langle \underset{1}{A_{30,35}} , \underset{2}{B_{35,15}} , \underset{3}{C_{15,5}} , \underset{4}{D_{5,10}} , \underset{5}{E_{10,20}} , \underset{6}{F_{20,25}} , \underset{7}{H_{25,18}} \rangle$$

$$\psi_{r..s} = \min_{r \le j < s} \left(\psi_{r..j} + \psi_{j+1..s} + p[r-1]\, p[j]\, p[s] \right)$$

	1	2	3	4	5	6	7
1	0	15750					
2		0	2625				
3			0				
4				0			
5					0		
6						0	
7							0

$$\psi_{23} = \min_{2 \le j < 3} \left(\psi_{2..2} + \psi_{3..3} + 35 \times 15 \times 5 \right)$$

$$= \min_{1 \le j < 2} \left(0 + 0 + 2625 \right)$$

$$= 2625$$

Bottom-UP: Ordem para atacar os sub prob.

$$\langle \underset{1}{A_{30,35}} , \underset{2}{B_{35,15}} , \underset{3}{C_{15,5}} , \underset{4}{D_{5,10}} , \underset{5}{E_{10,20}} , \underset{6}{F_{20,25}} , \underset{7}{H_{25,18}} \rangle$$

$$\psi_{r..s} = \min_{r \le j < s} \left(\psi_{r..j} + \psi_{j+1..s} + p[r-1]\, p[j]\, p[s] \right)$$

	1	2	3	4	5	6	7
1	0	15750					
2		0	2625				
3			0	750			
4				0	1000		
5					0	5000	
6						0	9000
7							0

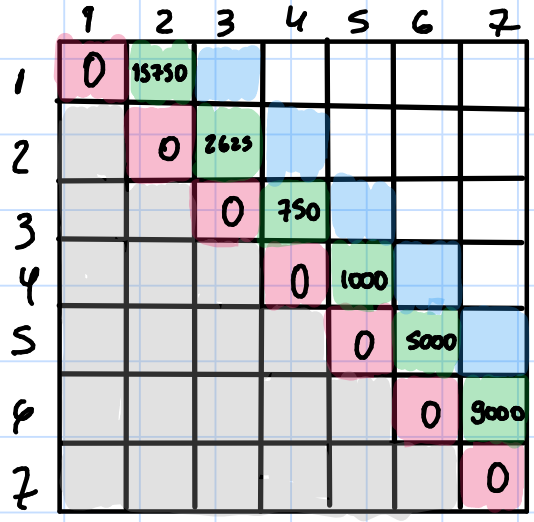
$$\psi_{23} = \min_{2 \le j < 3} \left(\psi_{2..2} + \psi_{3..3} + 35 \times 15 \times 5 \right)$$

$$= \min_{1 \le j < 2} \left(0 + 0 + 2625 \right)$$

$$= 2625$$

Bottom-UP: Ordem para atacar os sub prob.

- $m = 3 \Rightarrow 3 = s - r + 1 \Leftrightarrow 2 = s - r$



Bottom-UP: Ordem para atacar os sub prob.

- $n = 3 \Rightarrow 3 = s - r + 1 \Leftrightarrow 2 = s - r$

	1	2	3	4	5	6	7
1	0	15750	7875				
2		0	2625				
3			0	750			
4				0	1000		
5					0	5000	
6						0	9000
7							0

$$\psi_{13} = \min_{1 \le j < 3} \left(\psi_{1..1} + \psi_{2..3} + 30 \times 35 \times 5, \; \psi_{1..2} + \psi_{3..3} + 30 \times 15 \times 5 \right)$$

$$= \min_{1 \le j < 3} \left(0 + 2625 + 5250, \; 15750 + 750 + 2250 \right)$$

$$= \min (7875, 18750) = 7875$$

$$\langle \overset{1}{A_{30,35}}, \overset{2}{B_{35,15}}, \overset{3}{C_{15,5}}, \overset{4}{D_{5,10}}, \overset{5}{E_{10,20}}, \overset{6}{F_{20,25}}, \overset{7}{H_{25,18}} \rangle$$

$$\psi_{r..s} = \min_{r \le j < s} \left(\psi_{r..j} + \psi_{j+1..s} + p[r-1]\, p[j]\, p[s] \right)$$

Bottom-UP: Ordem para atacar os sub prob.

- $n = 3 \Rightarrow 3 = s - r + 1 \Leftrightarrow 2 = s - r$

	1	2	3	4	5	6	7
1	0	15750	7875				
2		0	2625	4375			
3			0	750	2500		
4				0	1000	3500	
5					0	5000	9500
6						0	9000
7							0

$$\psi_{13} = \min_{1 \le j < 3} \left(\psi_{1..1} + \psi_{2..3} + 30 \times 35 \times 5,\ \psi_{1..2} + \psi_{3..3} + 30 \times 15 \times 5 \right)$$

$$= \min_{1 \le j < 3} \left(0 + 2625 + 5250,\ 15750 + 750 + 2250 \right)$$

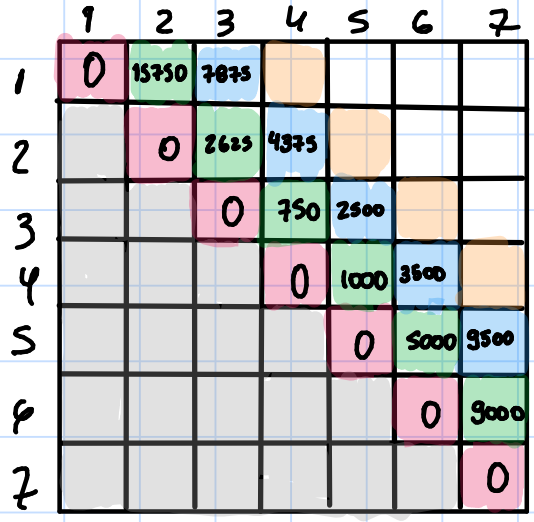
$$= \min (7875, 18750) = 7875$$

$$\langle \overset{1}{A_{30,35}}, \overset{2}{B_{35,15}}, \overset{3}{C_{15,5}}, \overset{4}{D_{5,10}}, \overset{5}{E_{10,20}}, \overset{6}{F_{20,25}}, \overset{7}{H_{25,18}} \rangle$$

$$\psi_{r..s} = \min_{r \le j < s} \left(\psi_{r..j} + \psi_{j+1..s} + p[r-1]\, p[j]\, p[s] \right)$$

Bottom-UP: Ordem para atacar os sub prob.

- $m = 4 \Rightarrow 4 = s - r + 1 \Leftrightarrow 3 = s - r$



Bottom-UP: Ordem para atacar os sub prob.

- $m = 4 \Rightarrow 4 = s - r + 1 \Leftrightarrow 3 = s - r$

	1	2	3	4	5	6	7
1	0	15750	7875	9375			
2		0	2625	4375	7125		
3			0	750	2500	5375	
4				0	1000	3500	5750
5					0	5000	9500
6						0	9000
7							0

Bottom-UP: Ordem para atacar os sub prob.

	1	2	3	4	5	6	7
1	0	15750	7875	9375	11875	15125	16325
2		0	2625	4375	7125	10500	11525
3			0	750	2500	5375	5750
4				0	1000	3500	5750
5					0	5000	9500
6						0	9000
7							0

Armazenando a solução ótima

$$\psi_{2..5} = m[2][5] = \min_{2 \le j < 5} \begin{cases} 0 + 2500 + 35 \times 15 \times 20 = 13000 & (j=2) \\ 2625 + 1000 + 35 \times 5 \times 20 = 7125 & (j=3) \\ 4375 + 5000 + 35 \times 10 \times 20 = 16375 & (j=4) \end{cases}$$

$$\psi_{2..j} + \psi_{j+1..5} + p[i-1]p[j]p[5]$$

	1	2	3	4	5	6	7
1	0	15750	7875	9375	11875	15125	16325
2		0	2625	4375	7125	10500	11525
3			0	750	2500	5375	5750
4				0	1000	3500	5750
5					0	5000	9500
6						0	9000
7							0

Uma solução ótima para a sequência $\langle A_2, A_3, A_4, A_5 \rangle$ pode ser obtida através das parentizações ótimas das sequências $\langle A_2, A_3 \rangle$ e $\langle A_4, A_5 \rangle$

Armazenando a solução ótima

$$\psi_{2..5} = m[2][5] = \min_{2 \le j < 5} \begin{cases} 0 + 2500 + 35 \times 15 \times 20 = 13000 & (j=2) \\ 2625 + 1000 + 35 \times 5 \times 20 = 7125 & (j=3) \\ 4375 + 5000 + 35 \times 10 \times 20 = 16375 & (j=4) \end{cases}$$

$$\psi_{2..j} + \psi_{j+1..5} + p[a-1]p[j]p[5]$$

	1	2	3	4	5	6	7
1	0	15750	7875	9375	11875	15125	16325
2		0	2625	4375	7125	10500	11525
3			0	750	2500	5375	5750
4				0	1000	3500	5750
5					0	5000	9500
6						0	9000
7							0

Uma solução ótima para a sequência $\langle A_2, A_3, A_4, A_5 \rangle$ pode ser obtida através das parentizações ótimas das sequências $\langle A_2, A_3 \rangle$ e $\langle A_4, A_5 \rangle$

Podemos armazenar essa informação

$sol[2][5] = 3$

Armazenando a Solução ótima

- Usaremos uma tabela $sol[1..n, 1..n]$ para armazenar a solução
  - $sol[i][j] = k$ representa que a solução ótima do algoritmo é uma parentização $\phi = (\phi_1\ \phi_2)$ para a sequência $\langle A_i, A_{i+1}, \cdots, A_j \rangle$ tal que ϕ_1 e ϕ_2 são parentizações das sequências $\langle A_i, A_{i+1}, \cdots, A_k \rangle$ e $\langle A_{k+1}, A_{k+2}, \cdots, A_j \rangle$, respectivamente.

```
Matrix-Chain-Order (p[0..n])
1  n ← p.comprimento - 1
2  Sejam m[1..n, 1..n] e sol[1..n, 1..n] matrizes
3  Para i ← 1 até n
4       m[i, i] ← 0
5  Para l ← 2 até n          ▷ l é o tamanho do subproblema
6       Para r ← 1 até n-l+1
7            s ← r + l - 1
8            m[r, s] ← ∞
9            Para j ← r até s - 1
10                q ← m[r, j] + m[j+1, s] + p[r-1] p[j] p[s]
11                se q < m[r, s]
12                     m[r, s] ← q
13                     sol[r, s] ← j
14 Retorna m[1, n], sol
```

Complexidade

- As linhas 1 e 14 tomam tempo $\Theta(1)$
- O laço da linha 3 executa n vezes, realizando um trabalho constante em cada uma delas. Assim o algoritmo gasta $\Theta(n)$ com as linha 3-4

- O laço da linha 2 executa $\Theta(n)$ vezes. O laço da linha 6 gasta $\Theta(n-l+1) = O(n)$. Assim as linhas 7-8 gastam $O(n^2)$ para executar.

- O número de iterações do laço da linha 9 é $\Theta(s-r) = O(n)$. Assim, ao longo da execução do algoritmo as linhas 10-13 gastam $\Theta(n) \cdot O(n)\, O(n) = O(n^3)$

Complexidade

- Portanto, o tempo de execução do algoritmo é $O(n^3)$.

- Agora, vamos argumentar que o tempo de execução é $\Omega(n^3)$, concluindo, portanto, que o algoritmo é $\Theta(n^3)$.

- Note que o tempo de uma execução do trecho de linhas 10-13 leva tempo $\Theta(1)$.
  - Note o tempo gasto com esse trecho é

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=i}^{i+l-2} \Theta(1) \quad \text{vezes}$$

  - Assim, o tempo gasto com esse trecho é

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=i}^{i+l-2} \Theta(1) \geq \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=i}^{i-l+2} D$$

$$= \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} (l-1)D = \sum_{l=2}^{n} (n-l)(l-1)D$$

$$= D\sum_{l=2}^{n} (n-l)(l-1) = D\sum_{l=1}^{n} (n-l)(l-1)$$

fica 0

$$= D\sum_{l=1}^{n} [nl - n - l^2 + l] = D\left[n\sum_{l=1}^{n} l - \sum_{l=1}^{n} n - \sum_{l=1}^{n} l^2 + \sum_{l=1}^{n} l\right]$$

$$= D\left[\sum_{l=1}^{n} l - \sum_{l=1}^{n} n - \sum_{l=1}^{n} l^2 + \sum_{l=1}^{n} l\right] =$$

$$= D\left[n\,\frac{1}{2}\,n\cdot(n+1) - n^2 - \frac{n(n+1)(2n+1)}{6} + \frac{1}{2}\,n\,(n+1)\right]$$

$$= D\left[\frac{3n^2(n+1)}{6} - \frac{6n^2}{6} - \frac{n(n+1)(2n+1)}{6} + \frac{3n(n+1)}{6}\right]$$

$$= \frac{D}{6}\left[3n^3 + 3n^2 - 6n^2 - 2n^3 - 3n^2 - n + 3n^2 + 3n\right]$$

$$= \frac{D}{6}\left[n^3 - 3n^2 + 2n\right]$$

$$\geq \frac{D}{6}\left[n^3 - 3n^2 + 2n\right]$$

$$\geq \frac{D}{6}\left[\frac{1}{2}n^3 + \frac{1}{2}n^3 - 3n^2 + 2n\right]$$

$$= \frac{D}{6}\left[\frac{1}{2}n^3 + 0\right]$$

$$= \frac{D}{12}n^3$$

$$\geq c\, n^3$$

Para $c \leq \frac{D}{12}$ e $n_0 = 9$, temos que o tempo gasto é $\Omega(n^3)$

$n \geq 9$

$$\frac{1}{2}n^3 - 3n^2 + 2n \geq 0$$

$$\Downarrow$$

$$n\left(\frac{1}{2}n^2 - 3n + 2\right) \geq 0$$

$$\Updownarrow$$

$$\frac{1}{2}n^2 - 3n + 2 \geq 0$$

$$y = \frac{-(-3) \pm \sqrt{(-3)^2 - 4 \cdot \frac{1}{2} \cdot 2}}{2 \cdot \frac{1}{2}}$$

$$y = \frac{6 \pm \sqrt{5}}{1}$$

$$\sqrt{4} \leq \sqrt{5} \leq \sqrt{9}$$

$$2 \qquad\qquad 3$$

Podemos escolher $n \geq 9$

Construindo a Solução Ótima

```
Imprime-Sol ( sol, r, s )
    se  r = s
        Imprime "A_r"
    senão
        Imprime "("
        Imprime-sol ( sol, r, sol[r,s] )
        Imprime-sol ( sol, sol[r,s]+1, s )
        Imprime ")"
```