Programação Dinâmica

Ennio Ferreira e Victor Matheus
08/06/2019 - LAMFO /UnB

AGENDA

- INTRODUÇÃO
- CARACTERIZAÇÃO DO PROBLEMA
- TIPOS DE SOLUÇÕES
- DESEMPENHO
- PROBLEMA AVANÇADO
- REFERÊNCIAS

INTRODUÇÃO

INTRODUÇÃO

Conceito - Programação Dinâmica

A programação dinâmica é um método de desenvolvimento que busca encontrar a solução de vários subproblemas para, daí então, encontrar a solução do problema geral.

*A grande novidade dessa metodologia é que **os subresultados são armazenados em memória** pois eles são utilizados em diversos momentos dentro do cômputo da solução.*

INTRODUÇÃO

Conceito - Programação Dinâmica

DIVISÃO-E-CONQUISTA

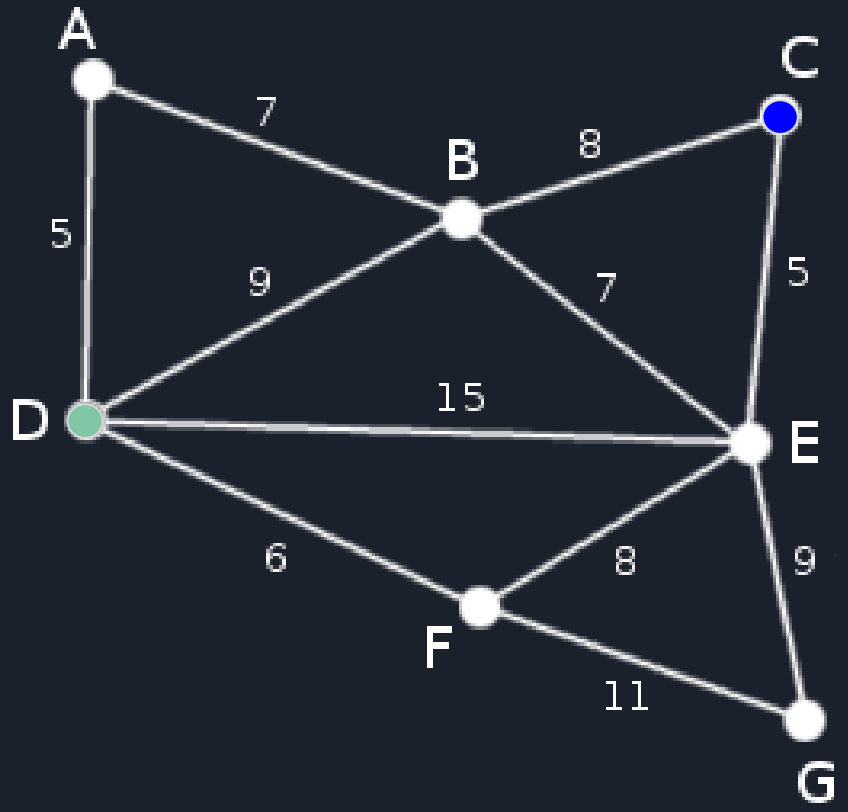
ARMAZENAMENTO

DEPENDÊNCIA DE SOLUÇÕES

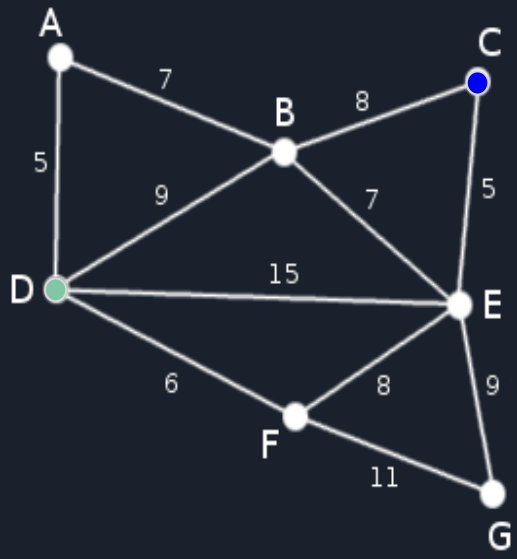
PROGRAMAÇÃO DINÂMICA

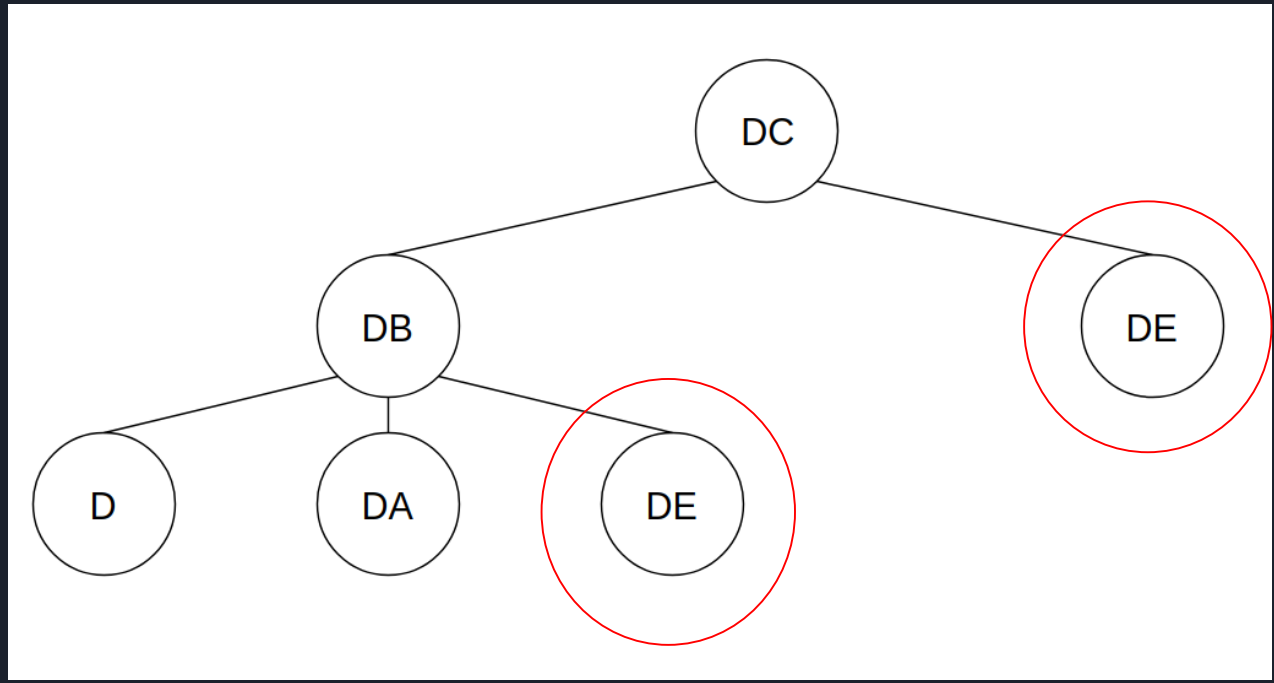
INTRODUÇÃO

Exemplo - Caminho mínimo



INTRODUÇÃO
Exemplo - Caminho mínimo





CARACTERIZAÇÃO DO PROBLEMA

CARACTERIZAÇÃO DO PROBLEMA

Exemplo - Série de Fibonacci

$$FIB(N) = \begin{cases} FIB[N-1] + FIB[N-2], & \text{se } N > 1 \\ 0, & \text{se } N = 0 \\ 1, & \text{se } N = 1 \end{cases}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Encontrar o termo N da sequência de Fibonacci é um problema que pode ser resolvido com Programação Dinâmica. Mas por quê?

CARACTERIZAÇÃO DO PROBLEMA

1º) Identificar problema como sendo de Programação Dinâmica

- **Subestrutura ótima:** a solução ótima do problema provém das soluções de subproblemas dependentes.

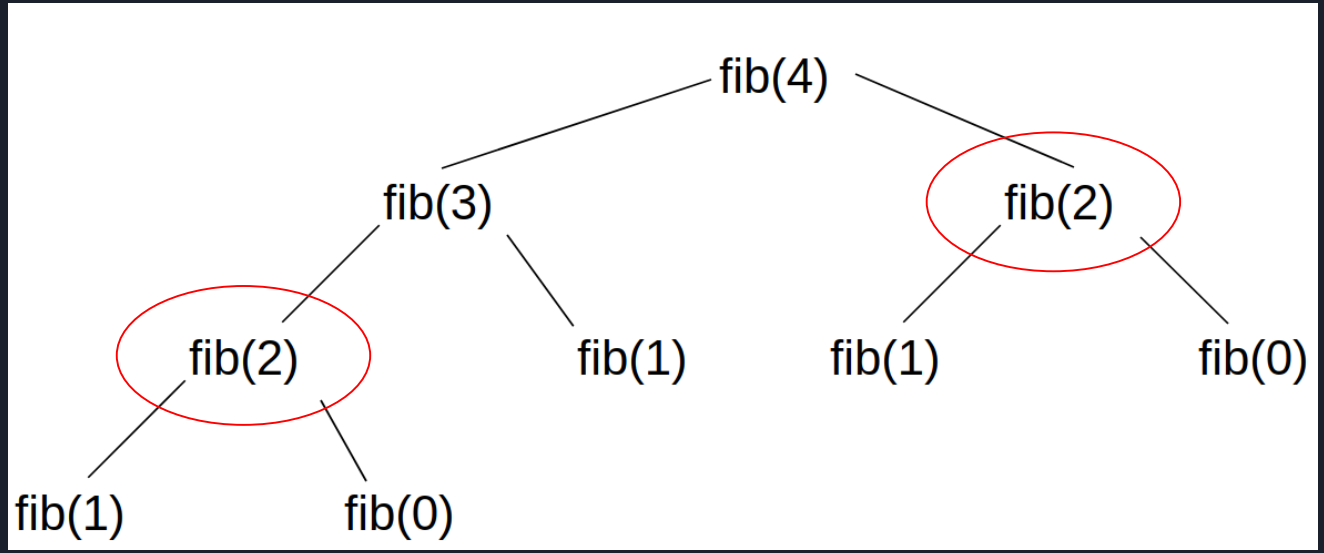
FIB(N) =

- FIB[N - 1] + FIB [N - 2], se N > 1
- 0, se N = 0
- 1, se N = 1

CARACTERIZAÇÃO DO PROBLEMA

1º) Identificar problema como sendo de Programação Dinâmica

- **Sobreposição de soluções:** a solução ótima passa pela resolução de subproblemas que aparecem duas ou mais vezes.



CARACTERIZAÇÃO DO PROBLEMA

2º) Definir o valor da solução ótima recursivamente

Para o caso da série de Fibonacci, esta etapa é bastante simples tendo em vista que já na explicação do problema é possível notar que a própria função que encontra a solução do termo N é demandada novamente para descoberta dos termos N-1 e N-2.

```
def fibonacci(n):
    if n == 1 or n == 0:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

TIPOS DE SOLUÇÕES

TIPOS DE SOLUÇÕES

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

- **Top-down (*memoization*):** parte-se da solução geral ótima que se deseja encontrar e, então, analisa-se quais subproblemas são necessários resolver até que se chegue em um subproblema com resolução trivial.

  fib(N) → fib(N-1) → fib(N-2) → … → fib(2) → fib(1) → fib(0)

  Os resultados vão sendo armazenados em uma tabela e são reutilizados se o mesmo subproblema aparecer novamente.

TIPOS DE SOLUÇÕES

3°) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

```
def fibonacciTopDown(n, table = {}):
    if n == 1 or n == 0:
        return n
    try:
        return table[n]
    except:
        table[n] = fibonacciTopDown(n-1) + fibonacciTopDown(n-2)
        return table[n]
```

Solução Trivial

Busca na tabela

Armazena resultado

TIPOS DE SOLUÇÕES

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

- **Bottom-up (*tabulation*):** parte-se da solução trivial e desenvolvem-se os cálculos até encontrar a solução ótima do problema.

  fib(0) → fib(1) → fib(2) → … → fib(N-2) → fib(N-1) → fib(N)

  Os resultados também vão sendo armazenados em uma tabela para serem reutilizados.

TIPOS DE SOLUÇÕES

3°) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

```
def fibonacciBottomUp(n, table = {}):
    table[0] = 0
    table[1] = 1
    for cont in range(2, n + 1):
        table[cont] = table[cont - 1] +  table[cont - 2]
    return table[n]
```

Solução Trivial

Busca na tabela e armazena

DESEMPENHO

DESEMPENHO

Top-down x Bottom-up

TOP-DOWN x BOTTOM-UP

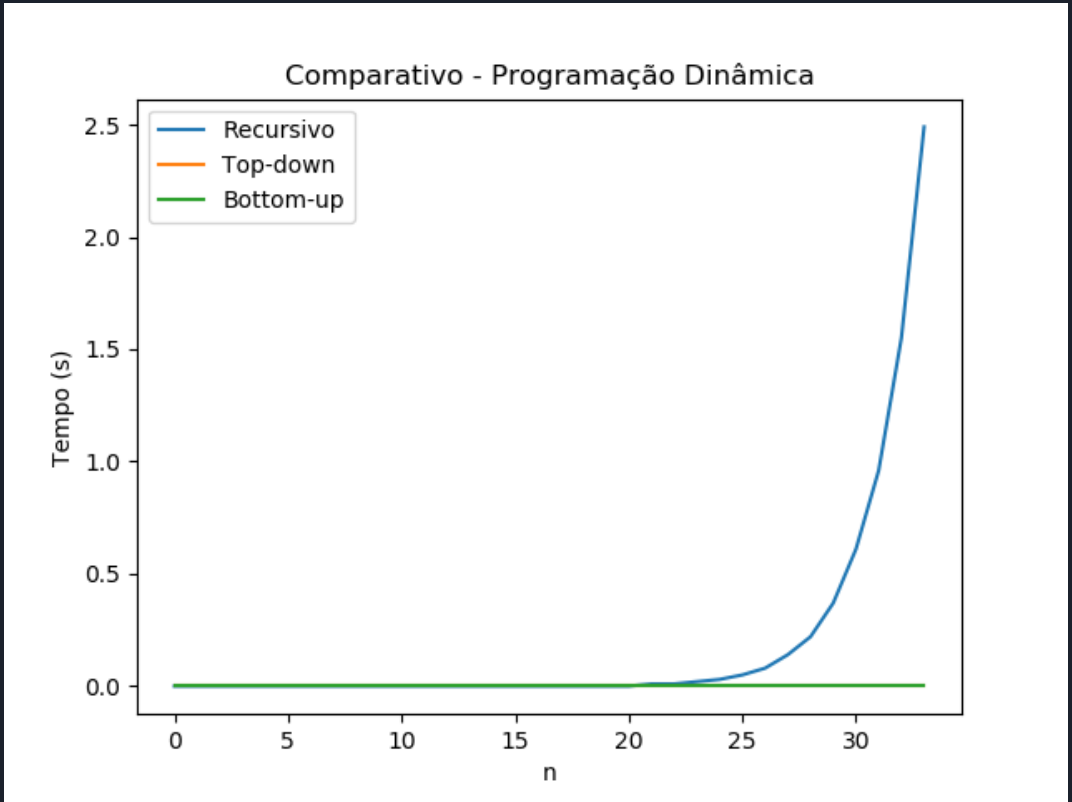
Quesitos	Top-down	Bottom-up
Especificação do problema	É mais fácil de se pensar no problema	Formular o problema pode ser complexo
Código	Codificação mais simples e com regras mais simples	Torna-se inviável quando há muitas condições
Resolução dos Subproblemas	Resolve apenas os subproblemas que são necessários	Resolve todos os subproblemas do problema original
Parâmetros na Tabela	A tabela é preenchida apenas com as soluções necessárias	Como parte da solução mais básica, preenche a tabela com soluções que podem ser desnecessárias

DESEMPENHO
Código

```
import time
for n in range(35, 40):
    start = time.time()
    result = fibonacci(n)
    finish = time.time()
    print("================================================================================")
    print("Fibonacci(", n, ")")
    print("Resultado - Fibonacci 1 (Original): ", result)
    print("Tempo Total de Execução - Fibonacci 1: ", round(finish - start, 2), "segundos")
    start = time.time()
    result = fibonacciTopDown(n)
    finish = time.time()
    print("Resultado - Fibonacci 2 (Top-Down): ", result)
    print("Tempo Total de Execução - Fibonacci 2: ", round(finish - start, 20), "segundos")
    start = time.time()
    result = fibonacciBottomUp(n)
    finish = time.time()
    print("Resultado - Fibonacci 3 (Bottom-Up): ", result)
    print("Tempo Total de Execução - Fibonacci 3: ", round(finish - start, 20), "segundos")
    print("================================================================================")
```

DESEMPENHO

Recursivo x Top-down x Bottom-up



PROBLEMA AVANÇADO

PROBLEMA AVANÇADO

Multiplicação de Matrizes - Especificação do problema

Por ser uma operação associativa, a multiplicação de matrizes em sequência pode gerar diferentes custos computacionais.

$$\mathbf{A_1 \times A_2 \times A_3}$$

4 x 10 10 x 3 3 x 12

- $(A_1 \times A_2) \times A_3 = 4 * 10 * 3 + 4 * 3 * 12 = 264$ operações
- $A_1 \times (A_2 \times A_3) = 10 * 3 * 12 + 4 * 10 * 12 = 840$ operações

PROBLEMA AVANÇADO

Multiplicação de Matrizes

1º) Identificar problema como sendo de Programação Dinâmica

Subestrutura Ótima: vamos considerar uma cadeia de matrizes representadas por $A_{i...j}$ (A_i x A_{i+1} x … x A_j) e que a solução ótima divide as matrizes na posição k conforme abaixo:

$$\mathbf{(A_{i...k})x\ (A_{k+1...j})}$$

Perceba que a solução ótima do problema geral depende da solução ótima de $\mathbf{A_{i...k}}$ e $\mathbf{A_{k+1...j}}$.

PROBLEMA AVANÇADO

Multiplicação de Matrizes

1º) Identificar problema como sendo de Programação Dinâmica

Sobreposição : vamos considerar uma sequência de multiplicação de 4 matrizes

- **k = 1**

$$(A_1) \times (A_2 \times A_3 \times A_4)$$

1..1 **2..4**

- **k = 2**

$$(A_1 \times A_2) \times (A_3 \times A_4)$$

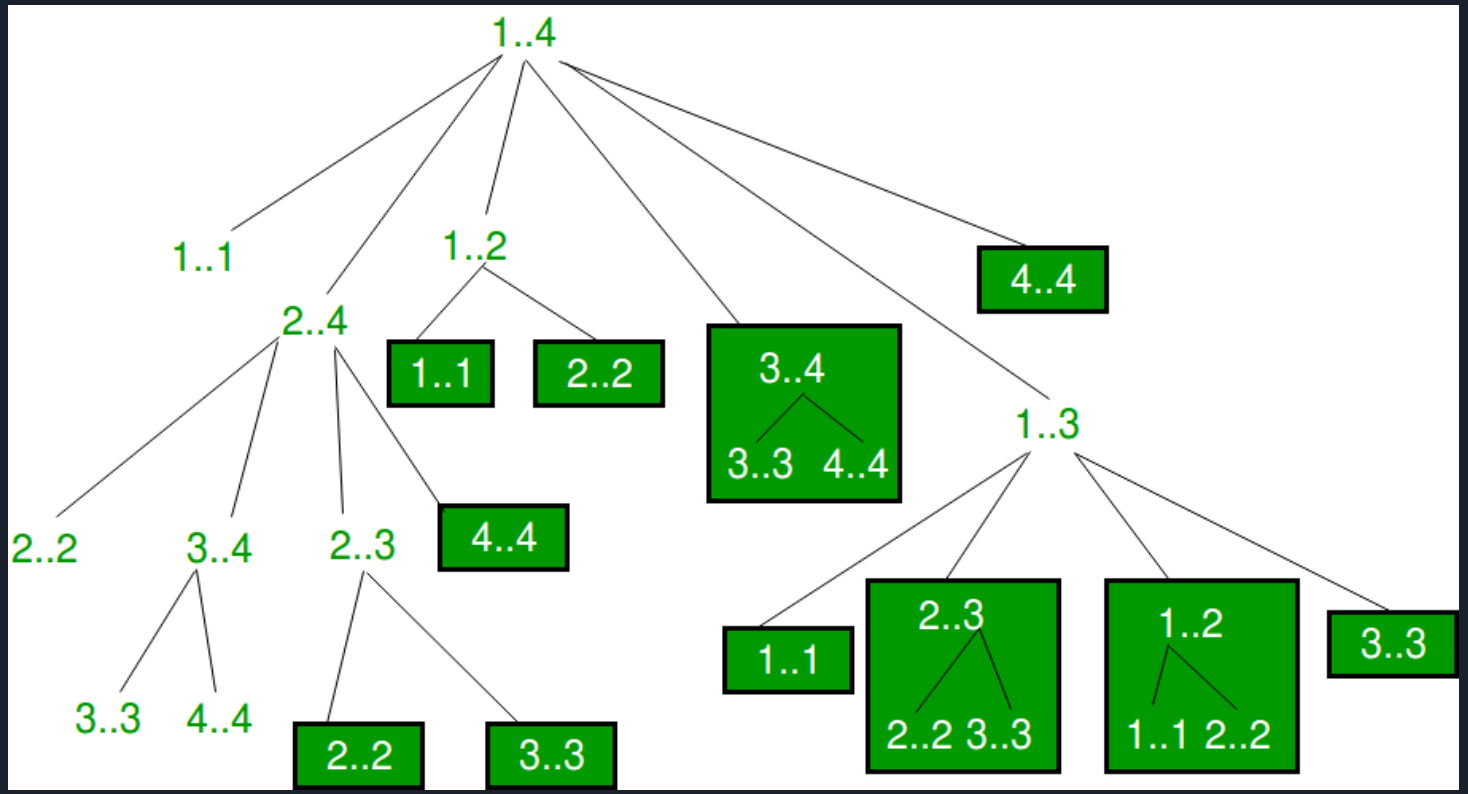
1..2 **3..4**

.
.
.

PROBLEMA AVANÇADO

Multiplicação de Matrizes

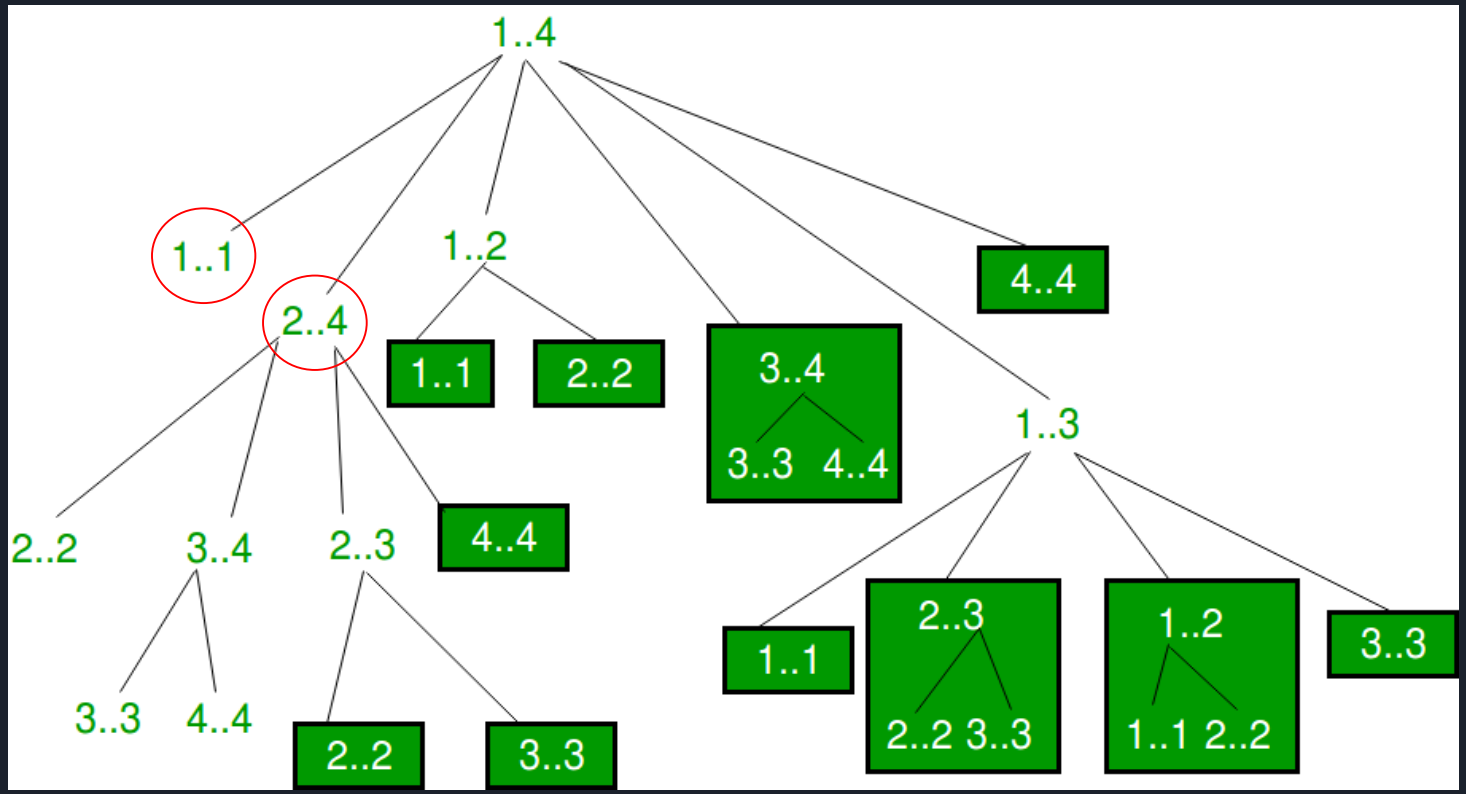
$(A_1 \times A_2 \times A_3 \times A_4)$



PROBLEMA AVANÇADO

Multiplicação de Matrizes

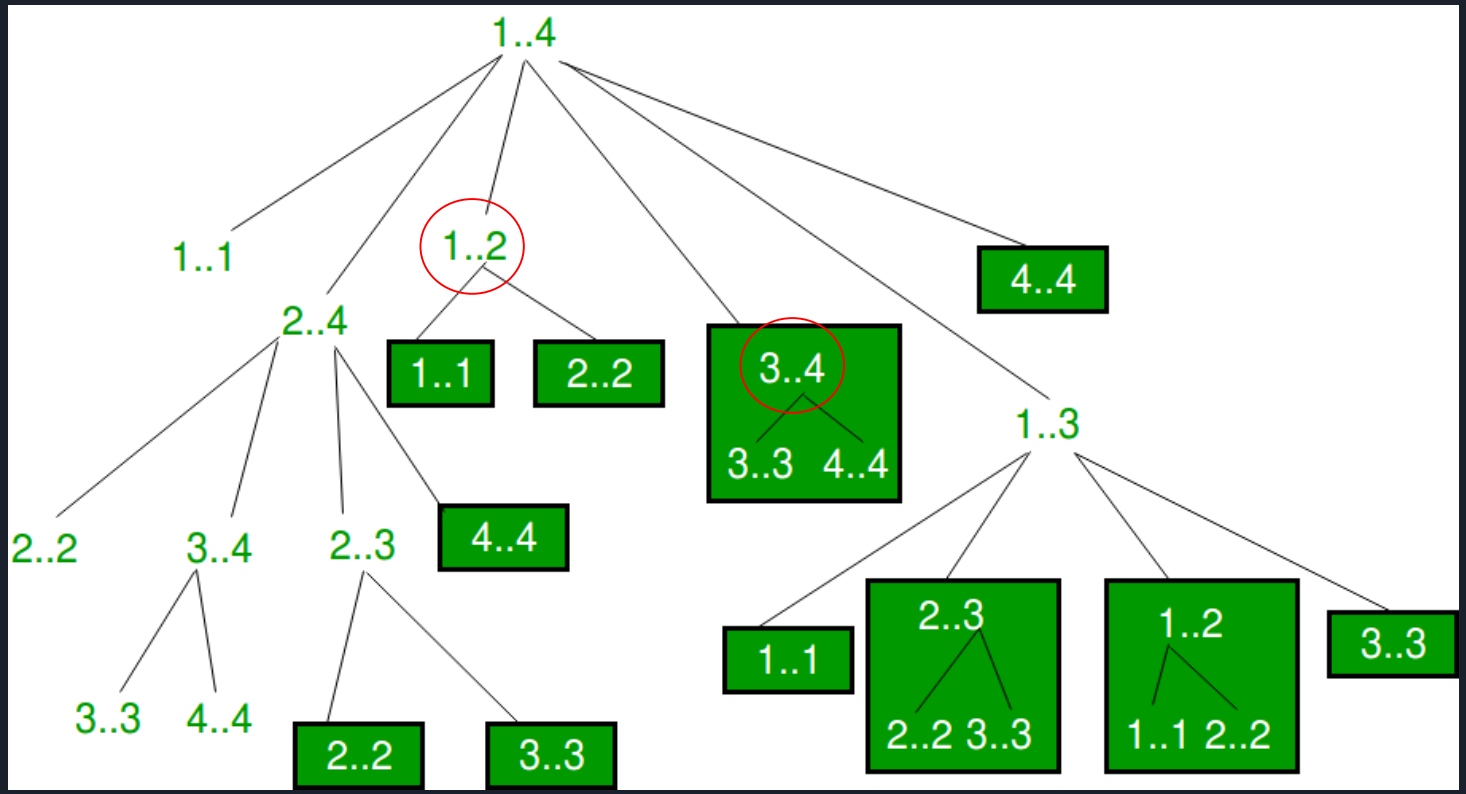
$(A_1) \times (A_2 \times A_3 \times A_4)$



PROBLEMA AVANÇADO

Multiplicação de Matrizes

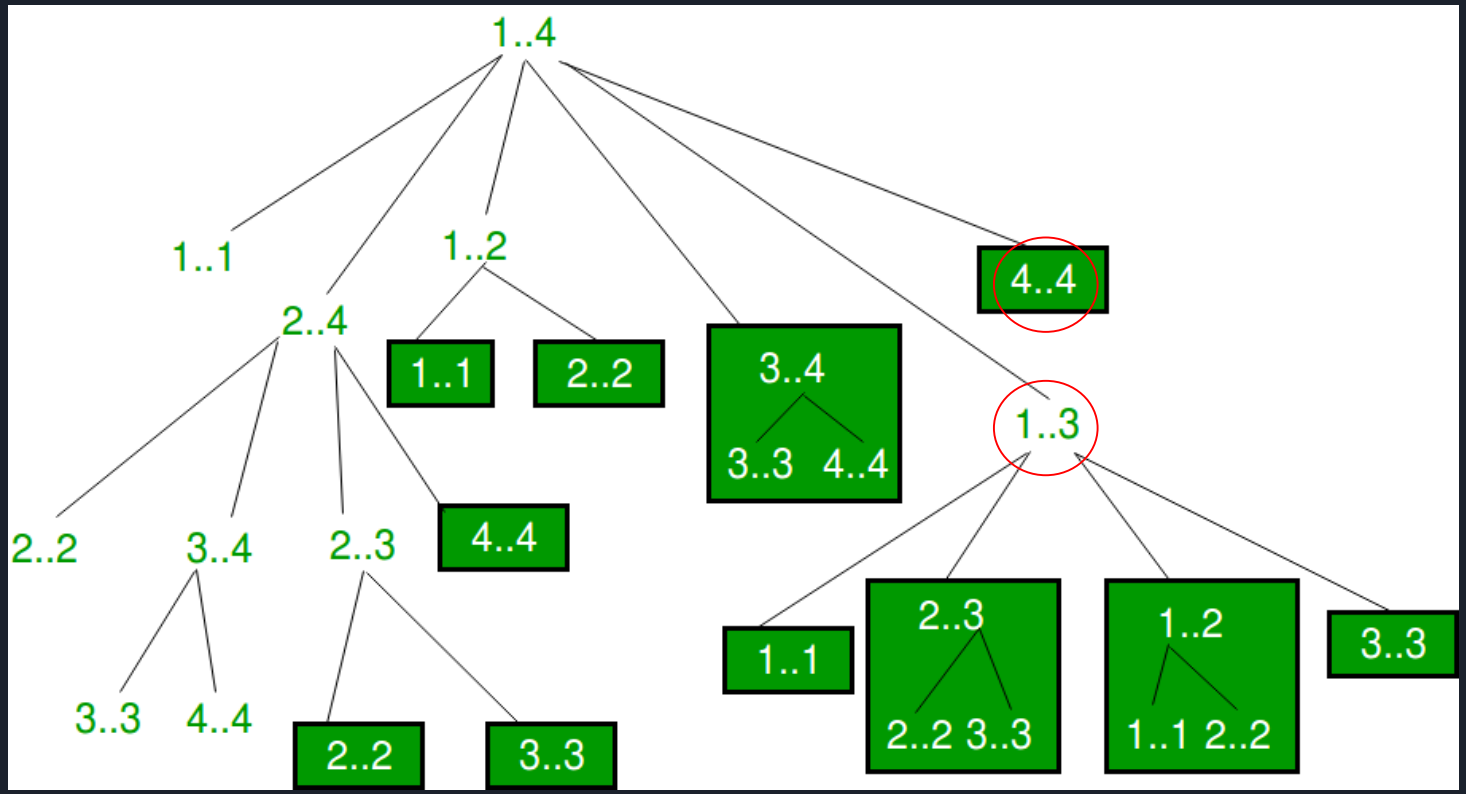
$(A_1 \times A_2) \times (A_3 \times A_4)$



PROBLEMA AVANÇADO

Multiplicação de Matrizes

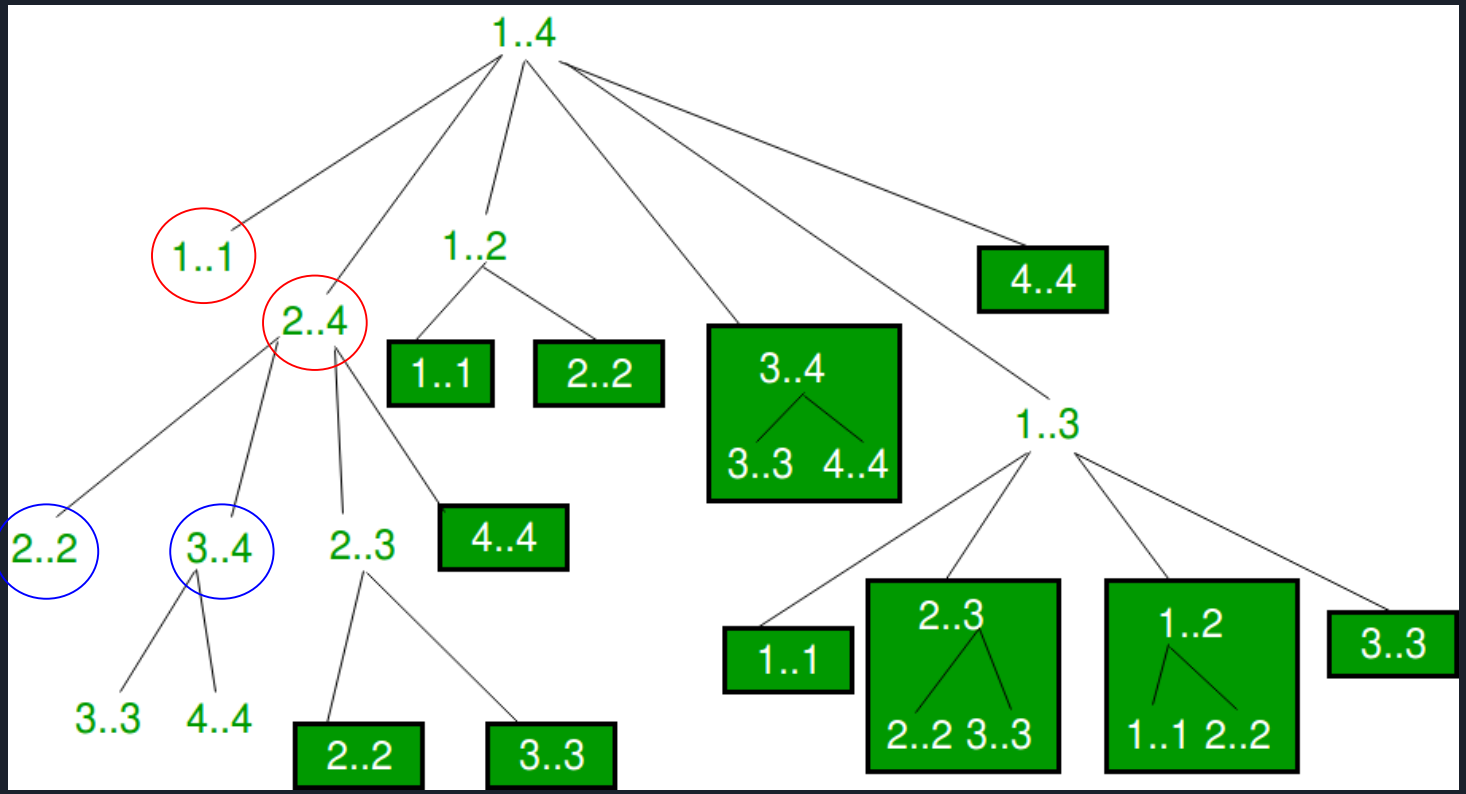
$(A_1 \times A_2 \times A_3) \times (A_4)$



PROBLEMA AVANÇADO

Multiplicação de Matrizes

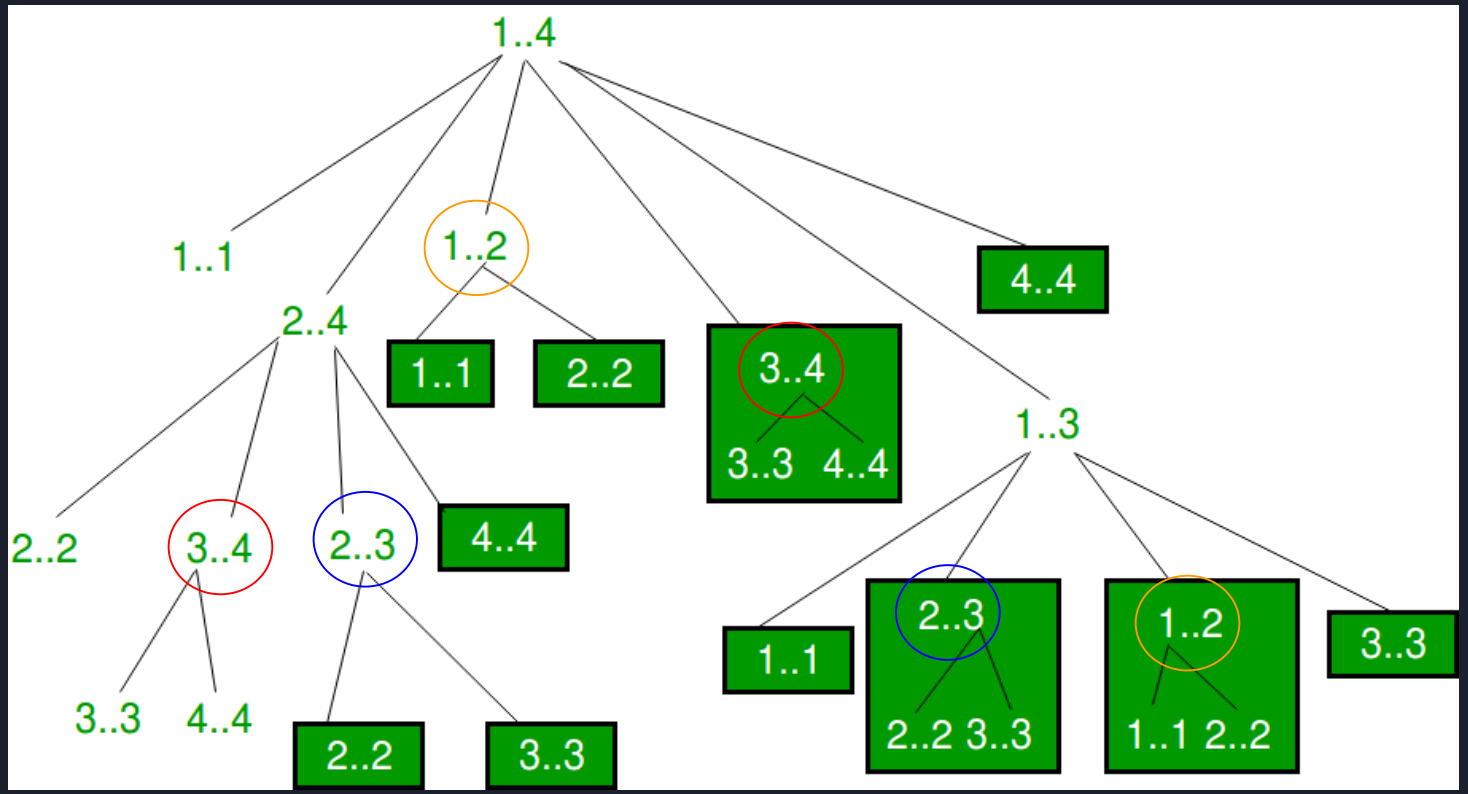
$(A_1\ ) \times ((\ A_2\ ) \times (\ A_3\ \times\ A_4)$



PROBLEMA AVANÇADO

Multiplicação de Matrizes

$(A_1 \times A_2 \times A_3 \times A_4)$



PROBLEMA AVANÇADO

Multiplicação de Matrizes

2º) Definir o valor da solução ótima recursivamente

Como visto anteriormente, podemos dividir a solução geral em subproblemas pela divisão da matrizes na k-ésima posição de tal forma que gere o menor número de multiplicações.

$$\underset{4 \times 10}{(A_{i \ldots k})} \times \underset{10 \times 3}{(A_{k+1 \ldots j})}$$

Por não sabermos que valor de k será, esse parâmetro assume valores no intervalo de $i \leq k < j$. Definindo M[i,j] como sendo o custo para se realizar a multiplicação das matrizes i até j, temos que para a divisão anterior

PROBLEMA AVANÇADO

Multiplicação de Matrizes

2º) Definir o valor da solução ótima recursivamente

$$M[i, j] = M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j]$$

Dessa forma, no exemplo anterior teríamos

$$M[i, j] = 200 + 300 + 4*10*3 = 620 \text{ operações}$$

Assim, define-se a fórmula recursiva final será:

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \mathbf{min}_{i \leq k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1$	x	A_2	x	A_3	x	$A_4)$
4 x 10		**10 x 3**		**3 x 12**		**12 x 20**
p0 p1		**p1 p2**		**p2 p3**		**p3 p4**

Para este exemplo, vamos escolher resolvê-lo pela abordagem bottom-up.

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \times A_2 \times A_3 \times A_4)$

A_1: 4 x 10, p0 p1 | A_2: 10 x 3, p1 p2 | A_3: 3 x 12, p2 p3 | A_4: 12 x 20, p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0			
2	x	0		
3	x	x	0	
4	x	x	x	0

$$M[1, 2] = M[1, 1] + M[2, 2] + p[0]p[1]p[2]$$
$$= 0 + 0 + 4 * 10 * 3 = 120$$

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \times A_2 \times A_3 \times A_4)$

A_1: 4 x 10 — p0 p1

A_2: 10 x 3 — p1 p2

A_3: 3 x 12 — p2 p3

A_4: 12 x 20 — p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0	120		
2	x	0		
3	x	x	0	
4	x	x	x	0

$$M[1, 2] = M[1, 1] + M[2, 2] + p[0]p[1]p[2]$$
$$= 0 + 0 + 4 * 10 * 3 = 120$$

$$M[2, 3] = M[2, 2] + M[3, 3] + p[1]p[2]p[3]$$
$$= 0 + 0 + 10 * 3 * 12 = 360$$

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \times A_2 \times A_3 \times A_4)$

A_1: 4 x 10, p0 p1 — A_2: 10 x 3, p1 p2 — A_3: 3 x 12, p2 p3 — A_4: 12 x 20, p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0	120		
2	x	0	360	
3	x	x	0	
4	x	x	x	0

$$M[1, 2] = M[1, 1] + M[2, 2] + p[0]p[1]p[2]$$
$$= 0 + 0 + 4 * 10 * 3 = 120$$

$$M[2, 3] = M[2, 2] + M[3, 3] + p[1]p[2]p[3]$$
$$= 0 + 0 + 10 * 3 * 12 = 360$$

$$M[3, 4] = M[3, 3] + M[4, 4] + p[2]p[3]p[4]$$
$$= 0 + 0 + 3 * 12 * 20 = 720$$

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \times A_2 \times A_3 \times A_4)$

A_1: 4 x 10 — p0 p1

A_2: 10 x 3 — p1 p2

A_3: 3 x 12 — p2 p3

A_4: 12 x 20 — p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0	120		
2	x	0	360	
3	x	x	0	720
4	x	x	x	0

- k = 1

$M[1, 3] = M[1, 1] + M[2, 3] + p[0]p[1]p[3]$
$= 0 + 360 + 4 * 10 * 12 = 840$

- k = 2

$M[1, 3] = M[1, 2] + M[3, 3] + p[0]p[2]p[3]$
$= 120 + 0 + 4 * 3 * 12 = 264$

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \times A_2 \times A_3 \times A_4)$

A_1: 4 x 10, p0 p1 | A_2: 10 x 3, p1 p2 | A_3: 3 x 12, p2 p3 | A_4: 12 x 20, p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0	120	264	
2	x	0	360	
3	x	x	0	720
4	x	x	x	0

- k = 2

$$M[2, 4] = M[2, 2] + M[3, 4] + p[1]p[2]p[4]$$
$$= 0 + 720 + 10 * 3 * 20 = 1320$$

- k = 3

$$M[2, 4] = M[2, 3] + M[4, 4] + p[1]p[3]p[4]$$
$$= 360 + 0 + 10 * 12 * 20 = 2760$$

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \times A_2 \times A_3 \times A_4)$

A_1: 4 x 10, p0 p1 | A_2: 10 x 3, p1 p2 | A_3: 3 x 12, p2 p3 | A_4: 12 x 20, p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0	120	264	
2	x	0	360	1320
3	x	x	0	720
4	x	x	x	0

- k = 1

  M[1, 4] = M[1, 1] + M[2, 4] + p[0]p[1]p[4]
  = 0 + 1320 + 4 * 10 * 20 = 2120

- k = 2

  M[1, 4] = M[1, 2] + M[3, 4] + p[0]p[2]p[4]
  = 120 + 720 + 4 * 3 * 20= 1080

- k = 3

  M[1, 4] = M[1, 3] + M[4, 4] + p[0]p[3]p[4]
  = 264 + 0+ 4 * 12 * 20= 1224

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \times A_2 \times A_3 \times A_4)$

A_1: 4 x 10, p0 p1 | A_2: 10 x 3, p1 p2 | A_3: 3 x 12, p2 p3 | A_4: 12 x 20, p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0	120	264	1080
2	x	0	360	1320
3	x	x	0	720
4	x	x	x	0

- k = 1

  M[1, 4] = M[1, 1] + M[2, 4] + p[0]p[1]p[4]
  = 0 + 1320 + 4 * 10 * 20 = 2120

- k = 2

  M[1, 4] = M[1, 2] + M[3, 4] + p[0]p[2]p[4]
  = 120 + 720 + 4 * 3 * 20= 1080

- k = 3

  M[1, 4] = M[1, 3] + M[4, 4] + p[0]p[3]p[4]
  = 264 + 0+ 4 * 12 * 20= 1224

PROBLEMA AVANÇADO

Multiplicação de Matrizes

3º) Calcular valor da solução ótima da forma “bottom-up” ou top-down”

$(A_1 \quad x \quad A_2 \quad x \quad A_3 \quad x \quad A_4)$

4 x 10 | 10 x 3 | 3 x 12 | 12 x 20

p0 p1 | p1 p2 | p2 p3 | p3 p4

$$M[i, j] = \begin{cases} 0, \text{ se } i = j \\ \min_{i \le k < j} \{ M[i, k] + M[k + 1, j] + p[i-1]p[k]p[j] \} \end{cases}$$

i \ j	1	2	3	4
1	0	120	264	1080
2	x	0	360	1320
3	x	x	0	720
4	x	x	x	0

Como saber ao final quais matrizes multiplicar primeiro?

- k = 2

$$M[1, 4] = M[1, 2] + M[3, 4] + p[0]p[2]p[4]$$
$$= 120 + 720 + 4 * 3 * 20 = 1080$$

$(A_1 \quad x \quad A_2 \quad)x(\quad A_3 \quad x \quad A_4)$

M[1,2] M[3,4]

REFERÊNCIAS

REFERÊNCIAS

- CORMEN, Thomas H. et al. Introduction to algorithms. MIT press, 2009.
- Geeks for Geeks: https://www.geeksforgeeks.org/dynamic-programming/#concepts
- Introduction to Computational Thinking and Data Science (MIT Course): https://www.edx.org/course/introduction-to-computational-thinking-and-data-science-2
- Matrix Chain Multiplication - Dynamic Programming: https://www.youtube.com/watch?v=GMzVeWpyTN0

Obrigado!