# *Um estudo empírico de algoritmos para o Problema da Soma de Subconjuntos*

Projeto 2º Bimestre
CT208 - Matemática da Computação


Discentes: André Francisco M. Caetano (morielo@ita.br)
Edney da S. Souza (edneyess@fab.br)
Murilo S. Bernardini (murilosb@ita.br)

Docente: Prof. Dr. Nei Yoshihiro Soma


29 de novembro de 2018

# Agenda

1. Introdução e Objetivos
2. Algoritmos implementados
3. Instâncias de testes (clássicas)
4. Instância de teste (proposta)
5. Resultados do *Benchmarking*
6. Conclusões

# 1 - Introdução e Objetivos

# Introdução e Objetivos

Sejam *n* inteiros estritamente positivos $w_1 w_2 w_3 w_4 \ldots w_n$ e sejam $x_1 x_2 x_3 x_4 \ldots x_n$ com $x_i \in 0, 1$ para todc $1 \leq i \leq n$

Pergunta: **Dado um valor inteiro $W$. Existe algum subconjunto cuja soma a seguir seja válida?**

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + \ldots + w_n x_n = W$$

# Introdução e Objetivos

Neste trabalho foram implementadas três das técnicas mais conhecidas para solucionar o problema proposto.

Cada técnica foi posteriormente executada com instâncias específicas de teste que tem como objetivo avaliar o desempenho de cada solução com diferentes entradas de dados.

# 2 - Algoritmos Implementados

# *Meet in the Middle*

Origem do método:

HOROWITZ, E.; SAHNI, S. Computing partitions with applications to the knapsack problem. Journal of the ACM (JACM), ACM, v. 21, n. 2, p. 277–292, 1974.

# Meet-in-the-Middle

- Dividir o vetor na metade.
- Calcular as somas de todos os subconjuntos da primeira parte e guardar os resultados
- Ordenar os resultados
- Calcular uma a uma a soma de subconjuntos da segunda parte, buscar a diferença a W nos resultados da primeira parte.
- Se a busca for bem sucedida, então a resposta é “SIM”, existe uma soma igual a W. Caso contrário, a resposta é “NÃO”.

# Meet-in-the-Middle

$$V_h = (2, 3, 5, 7, 11, 13)$$

$$V_h = (2, 3, 5, 7, 11, 13) \Rightarrow V_1 = (2, 3, 5), V_2 = (7, 11, 13)$$

# Meet-in-the-Middle

$$
\begin{aligned}
w'_0 &= V_1 \cdot X_0 = (2,3,5) \cdot (0,0,0) = 0 \\
w'_1 &= V_1 \cdot X_1 = (2,3,5) \cdot (0,0,1) = 5 \\
w'_2 &= V_1 \cdot X_2 = (2,3,5) \cdot (0,1,0) = 3 \\
w'_3 &= V_1 \cdot X_3 = (2,3,5) \cdot (0,1,1) = 8 \\
w'_4 &= V_1 \cdot X_4 = (2,3,5) \cdot (1,0,0) = 2 \\
w'_5 &= V_1 \cdot X_5 = (2,3,5) \cdot (1,0,1) = 7 \\
w'_6 &= V_1 \cdot X_6 = (2,3,5) \cdot (1,1,0) = 5 \\
w'_7 &= V_1 \cdot X_7 = (2,3,5) \cdot (1,1,1) = 10
\end{aligned}
$$

# Meet-in-the-Middle

$$w'_0 = 0, X_0 = (0,0,0)$$
$$w'_4 = 2, X_4 = (1,0,0)$$
$$w'_2 = 3, X_2 = (0,1,0)$$
$$w'_1 = 5, X_1 = (0,0,1)$$
$$w'_6 = 5, X_6 = (1,1,0)$$
$$w'_5 = 7, X_5 = (1,0,1)$$
$$w'_3 = 8, X_3 = (0,1,1)$$
$$w'_7 = 10, X_7 = (1,1,1)$$

# Meet-in-the-Middle

$$w''_0 = V_2 \cdot X_0 = (7, 11, 13) \cdot (0, 0, 0) = 0$$

$$L = 31 - 0 = 31 \Rightarrow \text{Não encontrado!}$$

$$w''_1 = V_2 \cdot X_1 = (7, 11, 13) \cdot (0, 0, 1) = 13$$

$$L = 31 - 13 = 18 \Rightarrow \text{Não encontrado!}$$

$$w''_2 = V_2 \cdot X_2 = (7, 11, 13) \cdot (0, 1, 0) = 11$$

$$L = 31 - 11 = 20 \Rightarrow \text{Não encontrado!}$$

$$w''_3 = V_2 \cdot X_3 = (7, 11, 13) \cdot (0, 1, 1) = 24$$

$$L = 31 - 24 = 7 \Rightarrow \text{Encontrado } w'_5!$$

# Meet-in-the-Middle

Complexidade teórica (tempo):

$$\mathrm{O}(n2^{n/2})$$

Complexidade teórica (espaço):

$$\mathrm{O}(2^{n/2})$$

# Programação Dinâmica

Origem do método:

BELLMAN, R. The theory of dynamic programming. [S.l.], 1954.

# Programação Dinâmica

- Criação da matriz n x W, completa de “False”, com a primeira coluna “True”.
- Percurso em linha, comparando cada elemento v[n] do vetor com o respectivo W.

| | 0 | 1 | 2 | 3 | 4 | 5 | $= W$ |
|---|---|---|---|---|---|---|---|
| 0 | *True* | *False* | *False* | *False* | *False* | *False* | |
| 1 | *True* | *False* | *False* | *False* | *False* | *False* | |
| 2 | *True* | *False* | *False* | *False* | *False* | *False* | |
| 3 | *True* | *False* | *False* | *False* | *False* | *False* | |
| 4 | *True* | *False* | *False* | *False* | *False* | *False* | |
| $n$ | | | | | | | |

# Programação Dinâmica

- Alterando a matriz seguindo a lógica:
  - Se v[n] > Wn então
    Matriz[ n ][ Wn ] = Matriz[ n - 1 ][ Wn ]

  - Se v[n] <= Wn então
    Matriz[ n ][ Wn ] =

    Matriz[n - 1][ Wn ]
    v ("ou")
    Matriz[n-1][Wn-v[n-1]]

# Programação Dinâmica

V = [4, 2, 1, 3] e W = 5

| | 0 | 1 | 2 | 3 | 4 | 5 | $= W$ |
|---|---|---|---|---|---|---|---|
| 0 | $True$ | $False$ | $False$ | $False$ | $False$ | $False$ | |
| 1 | $True$ | $False$ | $False$ | $False$ | $True$ | $False$ | |
| 2 | $True$ | $False$ | $True$ | $False$ | $True$ | $False$ | |
| 3 | $True$ | $True$ | $True$ | $True$ | $True$ | $True$ | |
| 4 | $True$ | $True$ | $True$ | $True$ | $True$ | $True$ | |
| $n$ | | | | | | | |

# *Backtracking*

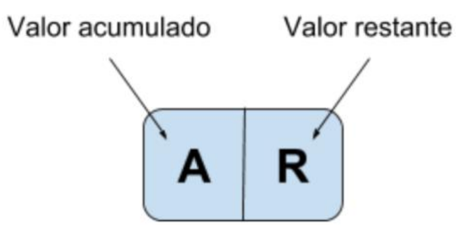
Origem do método:

ALEKHNOVICH, M. et al. Toward a model for backtracking and dynamic programming. In: IEEE. Computational Complexity, 2005. Proceedings. Twentieth Annual IEEE Conference on. [S.l.], 2005. p. 308–322.

# *Backtracking*

Algoritmo baseado na busca em profundidade em uma estrutura de dados do tipo árvore;

Cada nó da árvore possui as seguintes características:

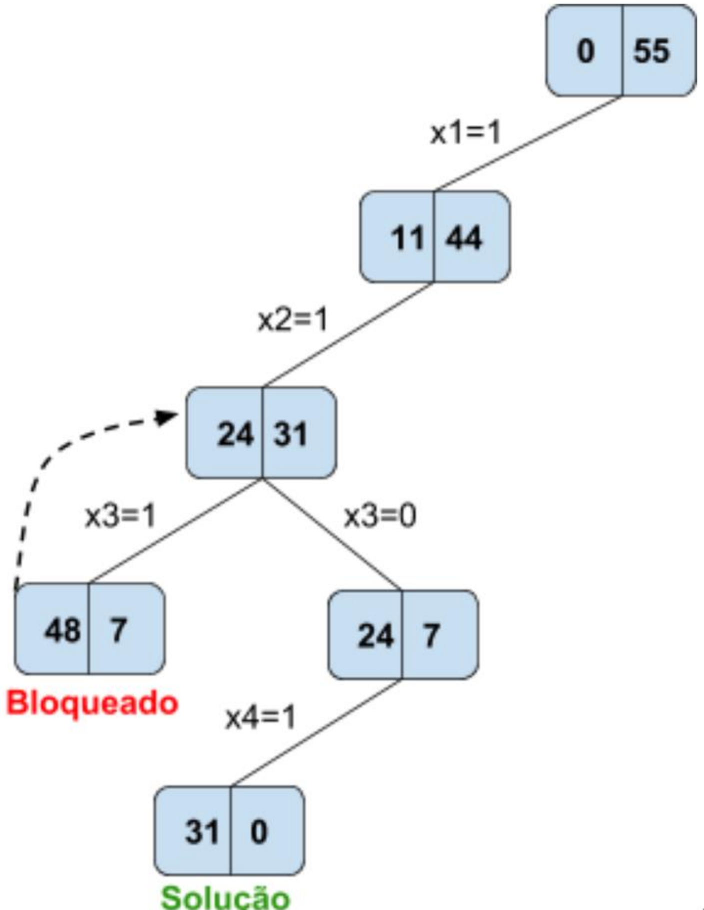



A execução do algoritmo será explicada através do seguinte exemplo:

$$n = \{11,\ 13,\ 24,\ 7\} \qquad m = 31$$

# *Backtracking*

$n = \{11,\ 13,\ 24,\ 7\}$ $m = 31$



# 3 - Instâncias de testes (clássicas)

# Instâncias de testes (clássicas)

## Distribuição Uniforme

$P(E)$ : $w_j$ *uniformemente*

*aleatório em* $[1, 10^E]$

$c = n\frac{10^E}{4}$

$n$ : 4, 8, 12, ..., 100

## Distribuição *EVEN/ODD*

$EVEN/ODD$ : $w_j$ *par e*

*uniformemente aleatório em* $[1, 10^3]$

$c = n\frac{10^3}{4}$ (*ímpar*)

$n$ : 4, 8, 12, ..., 100

# Instâncias de testes (clássicas)

**P(3)**

[81,195,210,328,496,531,656,724,855,991]
**n=10,W=2500**

**P(6)**

[90534,130627,253826,324379,480263,576825,  637120,784163,803356,988119]
**n=10,W=2500000**

**P(9)**
[64560180,139513283,258321313,387000757,437434585,587500411,640351899
,789289829 826224297,948522456]
**n=10,W=2500000000**

**EVEN/ODD**

[86,124,256,400,426,560,606,744,836 982]
**n=10,W=2501**

# Instâncias de testes (clássicas)

## Distribuição *AVIS*

$$AVIS : w_j = n(n+1) + j ,$$

$$c = \left\lfloor \frac{n-1}{2} \right\rfloor n(n+1) + \binom{n}{2}$$

$$n : 4, 8, 12, ..., 100$$

## Distribuição *TODD*

$$TODD : w_j = 2^{k+n+1} + 2^{k+j} + 1,$$

$$com\ k = \lfloor log_2 n \rfloor$$

$$c = \lfloor 0.5 \sum_{j=1}^{n} w_j \rfloor = (n+1)2^{k+n} - 2^k + \lfloor \tfrac{n}{2} \rfloor$$

$$n : 4, 5, 6, ..., 29$$

**REFERÊNCIA:** CHVÁTAL, 1980

# Instâncias de testes (clássicas)

**TODD**

[16393,16401,16417,16449,16513,16641,16897,17409,18433,20481]
**n=10**
**W=90109**

**AVIS**

[110,111,112,113,114,115,116,117,118,119]
**n=10**
**W=485**

# 4 - Instância de teste (proposta)

# Instância de teste (proposta)

Distribuição *EVEN/ODD* $2^k$

$$EVEN/ODD\ 2^k :\ w_j = \begin{array}{c} pares, \mathbf{todos\ pot\hat{e}ncias\ de\ 2}, \\ distribuídos\ de\ forma\ uniforme \\ no\ intervalo\ [2, 2^E] \end{array}$$

$$c = \ 2^E + 1\ (ímpar)$$

$$n:\ 4, 8, 12, \dots, 100$$

# Instância de teste (proposta)

Assim como no EVEN/ODD clássico, neste caso, como a soma de números pares sempre é um número par, o valor de W nunca será encontrado.

A soma de subconjuntos de potências de 2 já é ordenada e o algoritmo Meet-in-the-Middle usa o
QuickSort para fazer essa ordenação. O tempo do QuickSort quando o vetor está ordenado é $O(n^2)$

# Instância de teste (proposta)

EVEN/ODD $2^5$
[32,16,8,4,2 ]
**n = 5, W = 33**

EVEN/ODD $2^{10}$
[1024,512,256,128,64,32,16,8,4,2 ]
**n = 10, W = 1025**

EVEN/ODD $2^{15}$
[32768,16384,8192,4096,2048,1024,512,256,128,64,32,16,8,4,2 ] **n = 15, W = 32769**

# 5 - Resultados do *Benchmark*

# Resultados do Benchmark

Em alguns casos foi necessário limitar o tamanho do problema, tendo em vista as características de complexidade específicas a cada uma das soluções. No início de cada subseção são dados detalhes do computador utilizado em cada experimento. Como o objetivo não era comparar os algoritmos entre si, mas sim observar seus comportamentos para as instâncias de teste, foi possível executá-los em ambientes computacionais com configurações diferentes.

# *Benchmarking Meet in the Middle*

**Instâncias de teste:**

Uniforme;

EVEN/ODD;

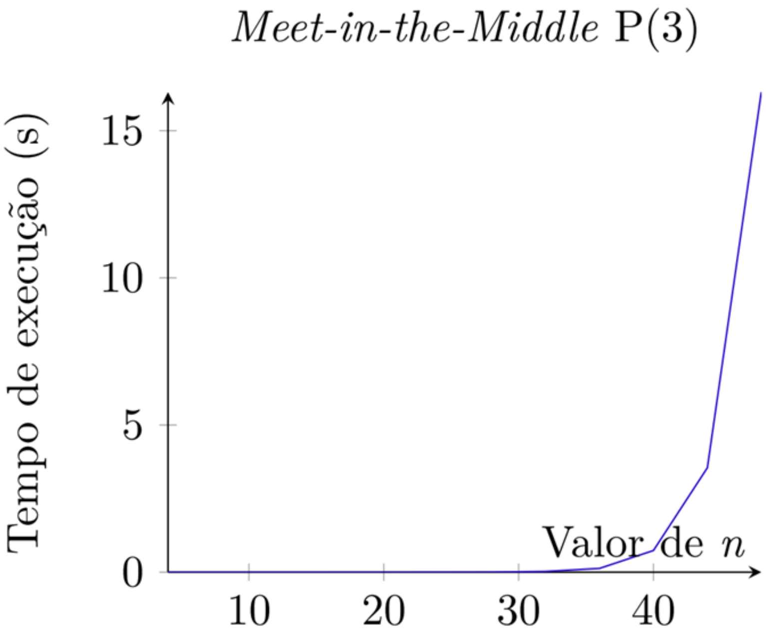
TODD;

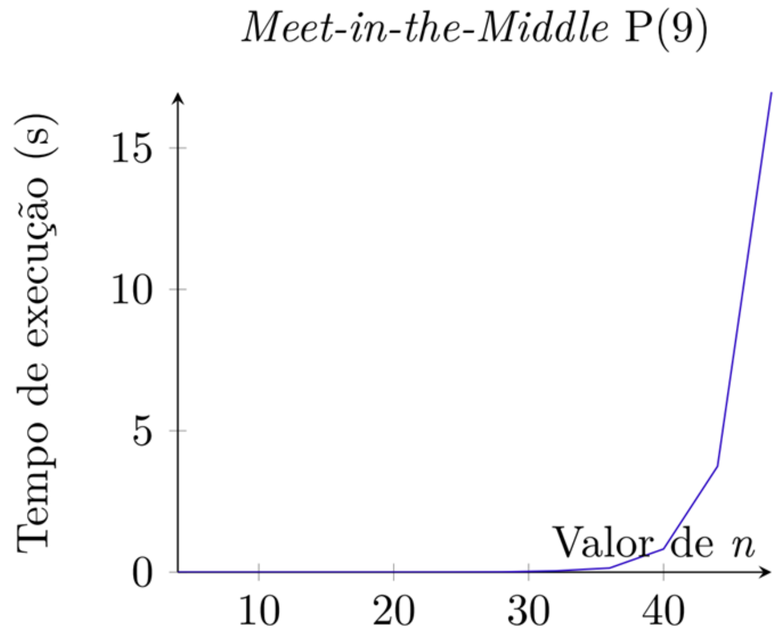
AVIS;

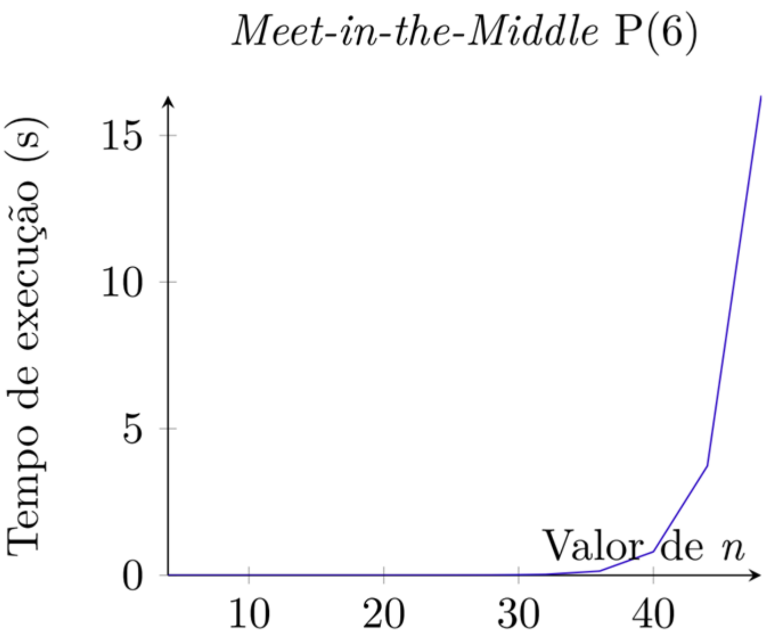
EVEN/ODD $2^K$

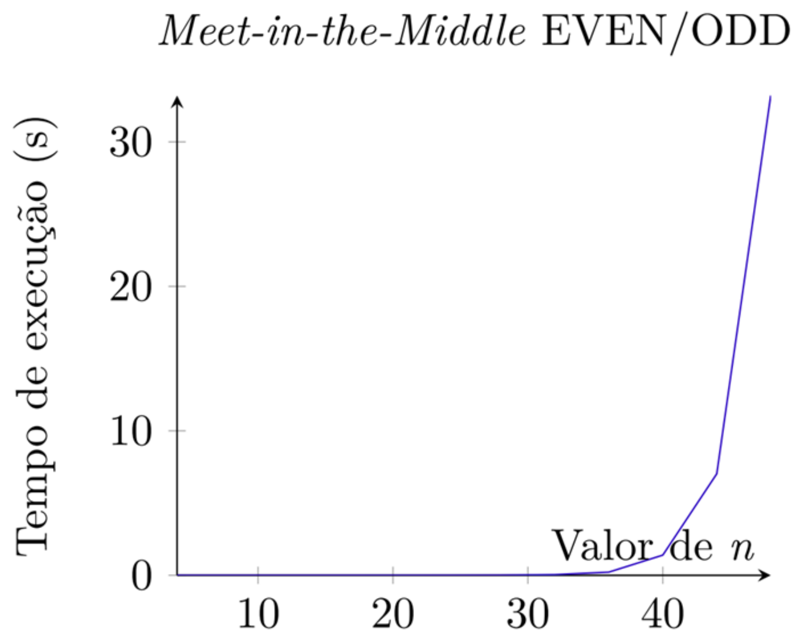
# Benchmarking Meet-in-the-Middle

| | |
|---|---|
| **CPU** | Intel Core i7 4700MQ 2.40GHz |
| **S.O.** | Debian 9 "Stretch" |
| **Memória** | 4GB DDR3 1600MHZ |

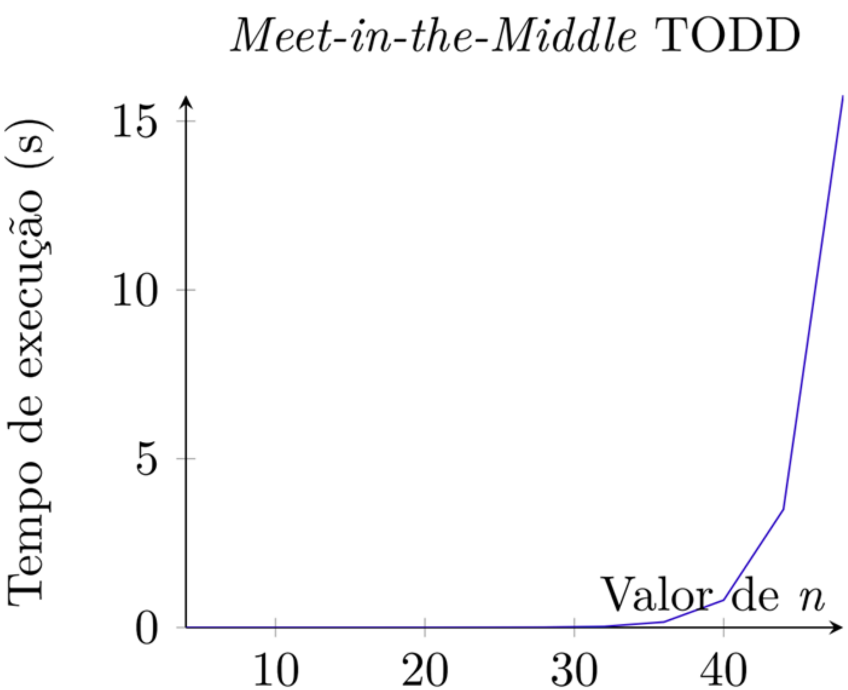

Meet-in-the-Middle P(3)
Tempo de execução (s)
Valor de n
0
5
10
15
10
20
30
40

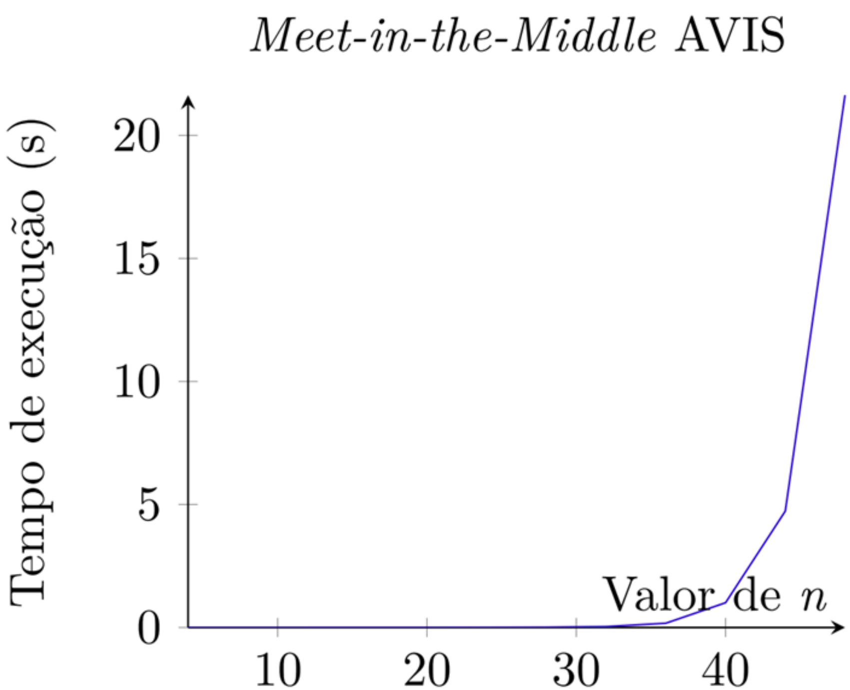

Meet-in-the-Middle P(9)
Tempo de execução (s)
Valor de n
0
5
10
15
10
20
30
40


Meet-in-the-Middle P(6)
Tempo de execução (s)
Valor de n
0
5
10
15
10
20
30
40

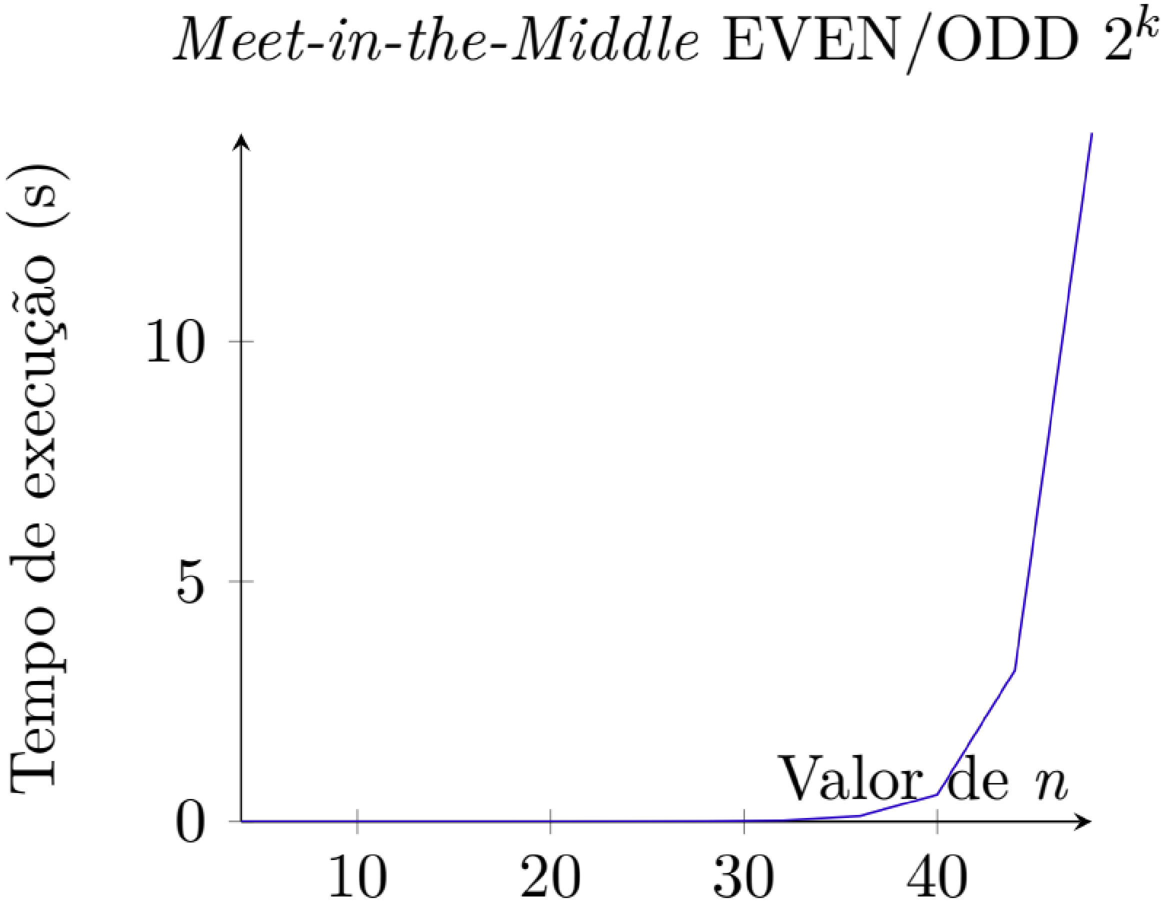

Meet-in-the-Middle EVEN/ODD
Tempo de execução (s)
Valor de n
0
10
20
30
10
20
30
40

# Benchmarking Meet-in-the-Middle



# Benchmarking Meet-in-the-Middle



# *Benchmarking Programação Dinâmica*

**Instâncias de teste:**

Uniforme;

EVEN/ODD;

TODD;

AVIS;

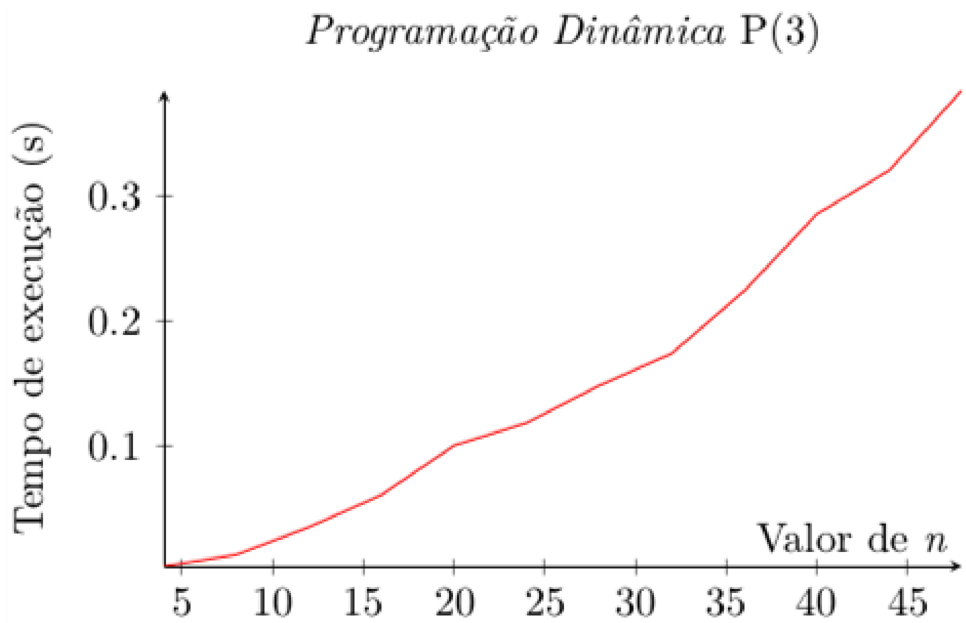
# Benchmarking Programação Dinâmica

| | |
|---|---|
| **CPU** | Intel Pentium P6200 2.13Ghz |
| **S.O.** | Ubuntu 18.04 LTS "Bionic Beaver" |
| **Memória** | 2GB DDR3 1066MHz |

# Benchmarking Programação dinâmica

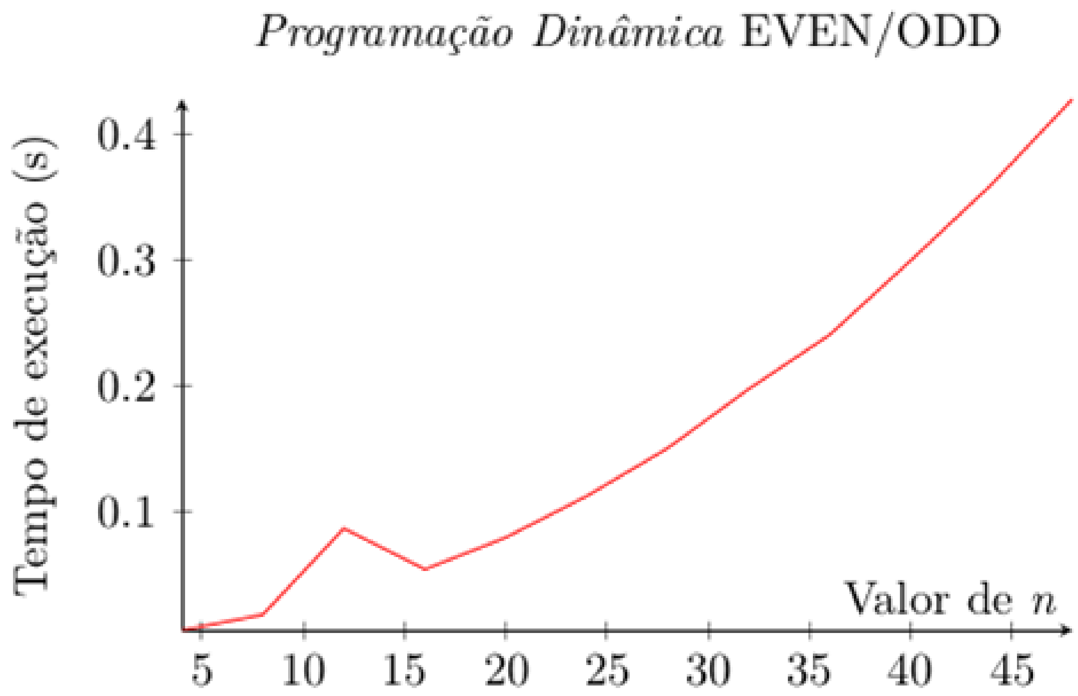
- Para n=48 temos
  W = 12000 = 250 * n

- Então temos
  O(Wn) = O(250n²)




- $P(6) = \theta(Wn) = \theta(12.000.000 * n) = \theta(250.000 * n * n) = \theta(250.000 * n^2)$
- $P(9) = \theta(Wn) = \theta(12.000.000.000 * n) = \theta(250000000 * n * n) = \theta(250.000.000 * n^2)$

# Benchmarking Programação dinâmica




- Para n=48 temos W = 12001 ≃ 250 * n
- Então temos O(Wn) ≃ O(250n²)

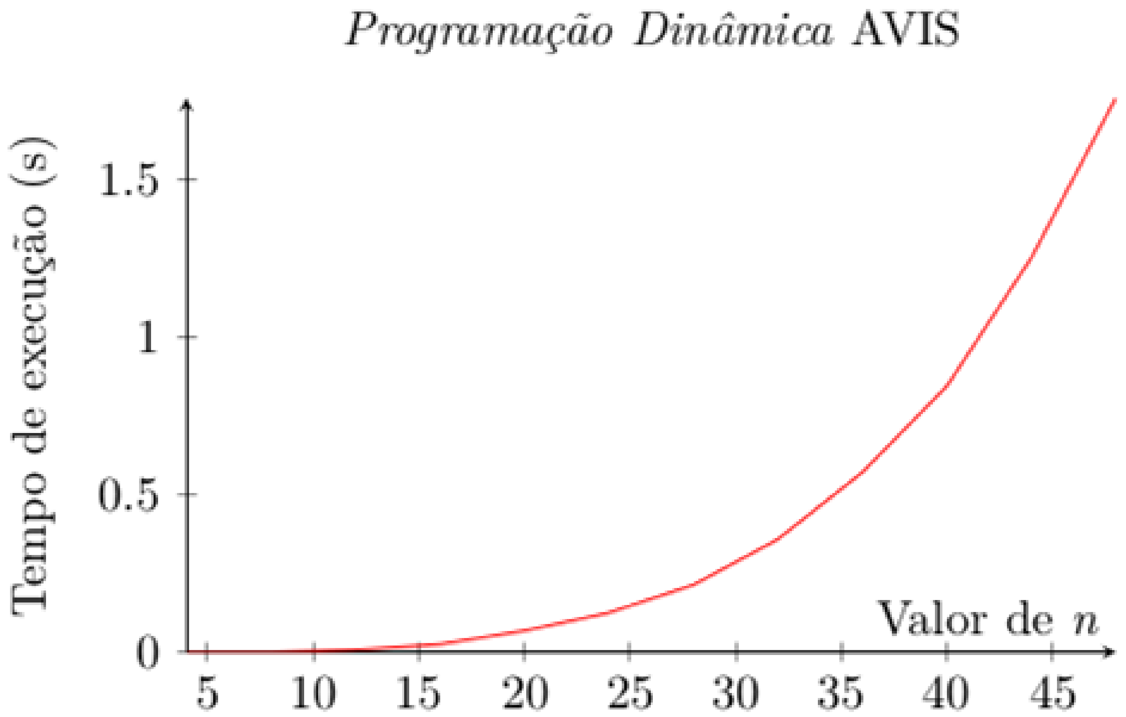
# Benchmarking Programação dinâmica

***Distribuição TODD***

- Para n=48 temos
  W = 662.029.145.223.462.016 ≃
  1.3792273858822126e+16 * n

- Então temos O(Wn) ≃
  O(1.3792273858822126e+16*n²)

# Benchmarking Programação dinâmica




- Para n=48 temos W = 55224 ≃ 1150 * n
- Então temos O(Wn) ≃ O(1150n²)

# *Benchmarking Backtracking*

**Instâncias de teste:**

Uniforme;

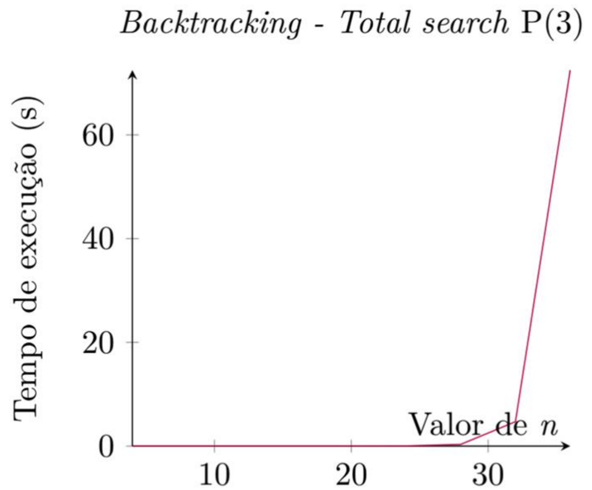
EVEN/ODD;

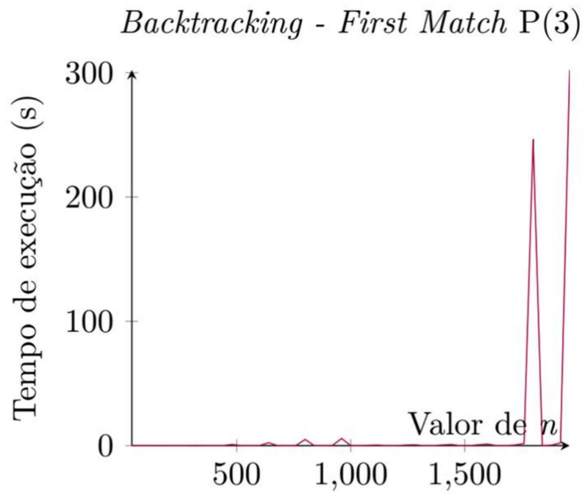
TODD;

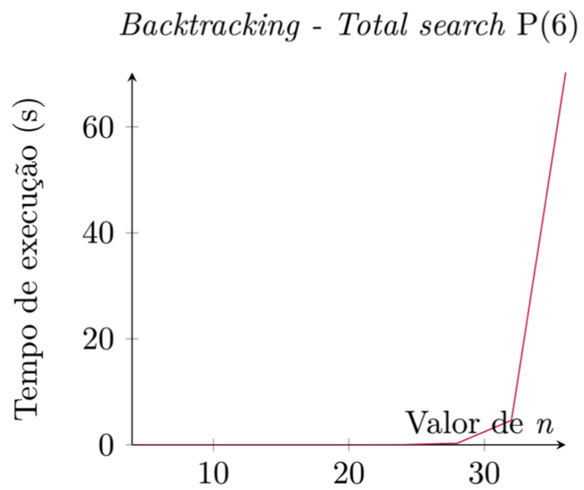
AVIS;

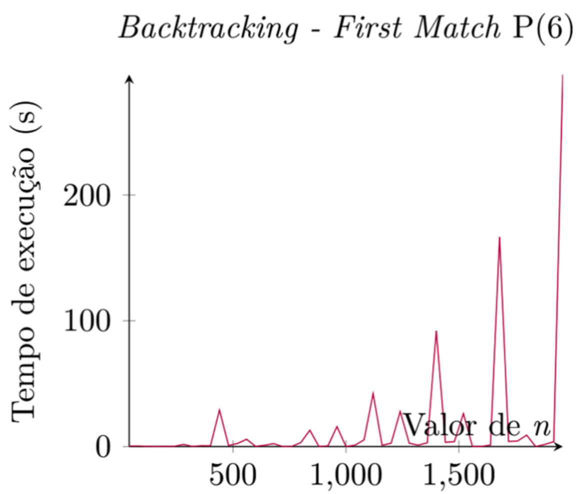
# *Benchmarking Backtracking*

| | |
|---|---|
| **CPU** | Intel Core i7 4790 3.40GHz |
| **S.O.** | MacOS 10.12.6 "Sierra" |
| **Memória** | 16GB DDR3 1600MHz |

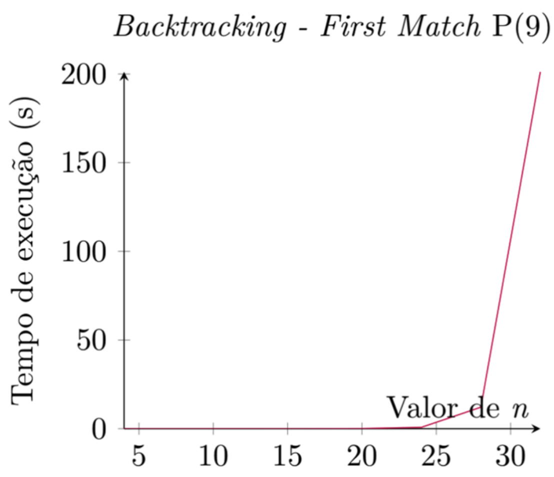


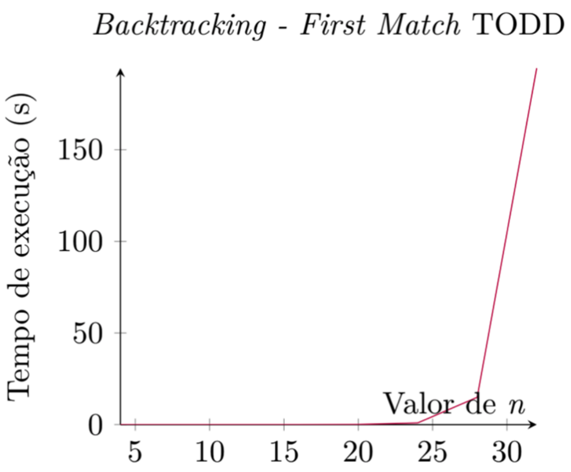



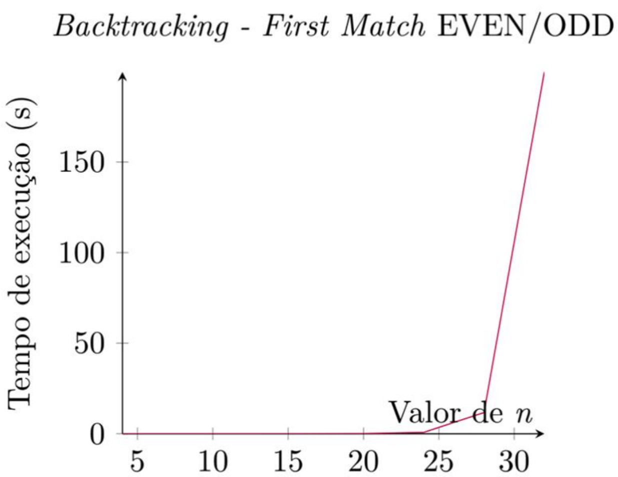



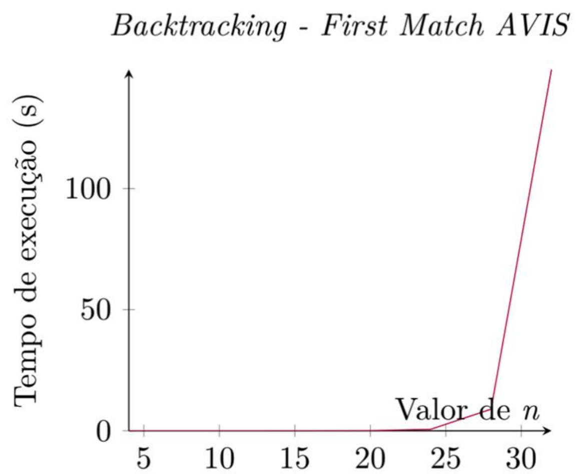



# Benchmarking Backtracking



# 6 - Conclusões

# O algoritmo **Meet-in-the-Middle:**

- Melhor que o Backtracking na maioria dos casos (justificado pela complexidade dos algoritmos)

- Recomendável quando o conjunto de dados é pequeno, até 40 ou 50 elementos.

- Melhor que a solução por Programação Dinâmica quando W é grande.

## O algoritmo com **Programação Dinâmica:**

- Melhor que os demais na maioria dos casos (justificado pela complexidade pseudopolinomial).
- Recomendável quando o conjunto de dados é grande, a partir de 40 ou 50 elementos.
- Não é recomendável quando W é grande, pois o tempo e espaço gastos dependem diretamente desse valor.

# O algoritmo **Backtracking**:

- Foi o mais afetado dos três nos *benchmarks* realizados.

- Conseguiu executar para valores de *n* bastante altos, bem mais que o Meet-in-the-Middle, com o uso de *First Match*.

- Sua utilização ou não em cada caso depende de uma avaliação mais detalhada desse algoritmo.

# Referências Bibliográficas

ALEKHNOVICH, M. et al. Toward a model for backtracking and dynamic programming. In: IEEE. Computational Complexity, 2005. Proceedings. Twentieth Annual IEEE Conference on. [S.l.], 2005. p. 308–322.

BELLMAN, R. The theory of dynamic programming. [S.l.], 1954.

CHVÁTAL, V. Hard knapsack problems. Operations Research, INFORMS, v. 28, n. 6, p. 1402–1411, 1980.

HOROWITZ, E.; SAHNI, S. Computing partitions with applications to the knapsack problem. Journal of the ACM (JACM), ACM, v. 21, n. 2, p. 277–292, 1974.