Programação Dinâmica

Intro, WISP, WSSP...

Gustavo Batista

Programação Dinâmica

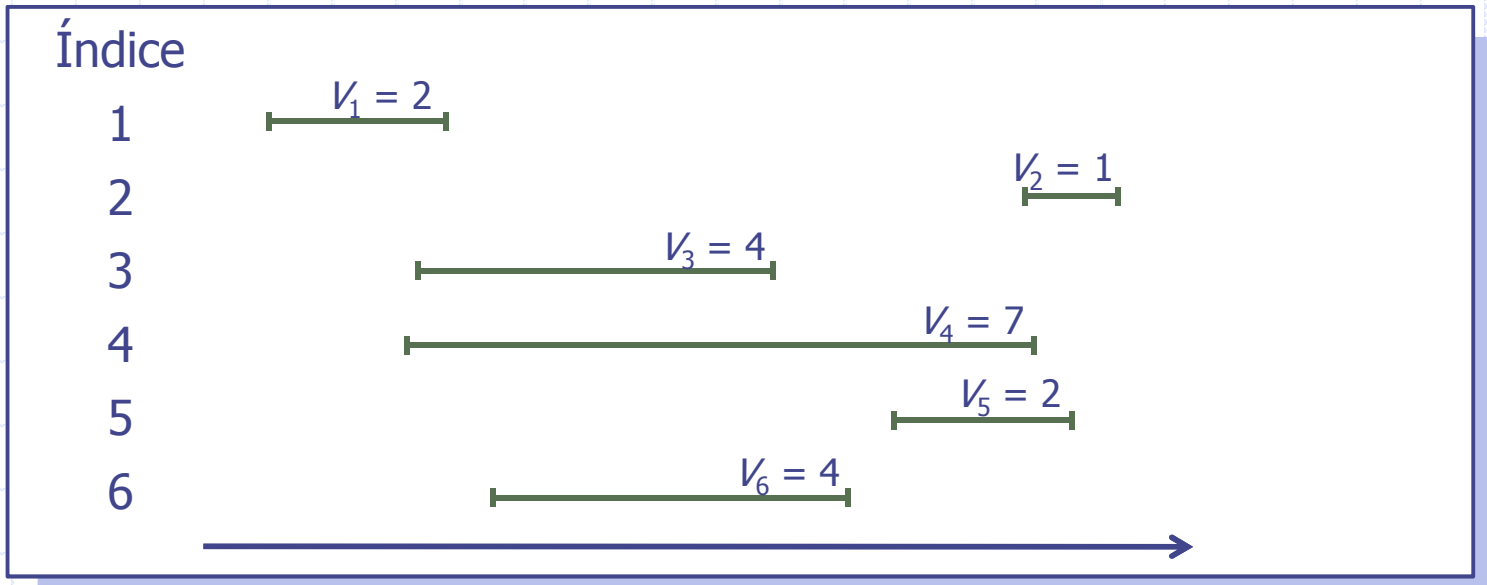
- Programação Dinâmica (PD) é uma poderosa técnica de programação matemática.
- Embora seja mais simples de discutir o que é PD após alguns exemplos práticos, a sua idéia geral é:
  - Explorar implicitamente o espaço de todas as possibilidades, decompondo o problema cuidadosamente em sub-problemas. Construir soluções corretas para problemas maiores por meio da composição dos sub-problemas.
  - É o mesmo que força-bruta? Não. DP trabalha sobre o espaço de soluções de tamanho exponencial sem ter que analisar todas as soluções possíveis explicitamente.

Programação Dinâmica

- Como é possível obter uma solução ótima sem analisar todo o espaço de soluções?
  - DP somente é aplicável a um conjunto de problemas que obedecem o princípio de optimalidade de Bellman (sub-estrutura ótima):
  - "*Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision* (Bellman, 1957)."

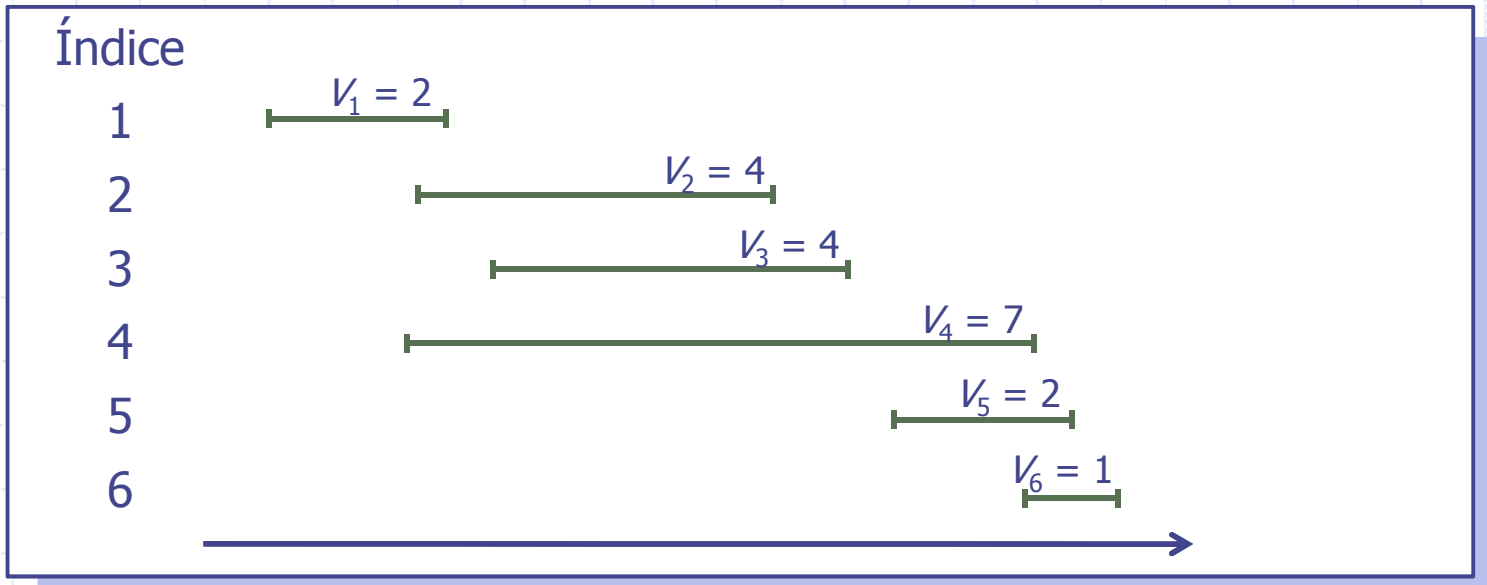
Exemplo 1 – WISP

- Problema de escalonamento de intervalos com pesos (Weighted Interval Scheduling Problem – WISP):
  - Selecionar um sub-conjunto de intervalos com a maior soma de pesos possível sem que dois intervalos estejam sobrepostos.



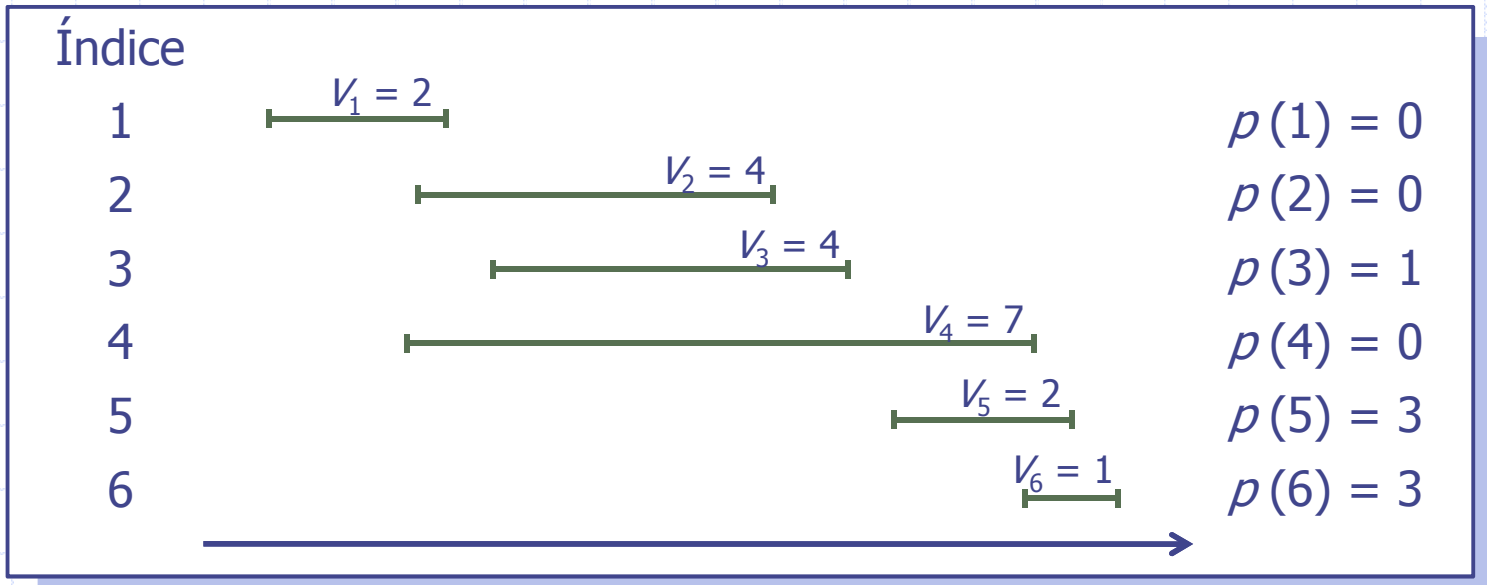
Exemplo 1 – WISP

- Vamos supor que os intervalos estão ordenados pelo tempo de término.



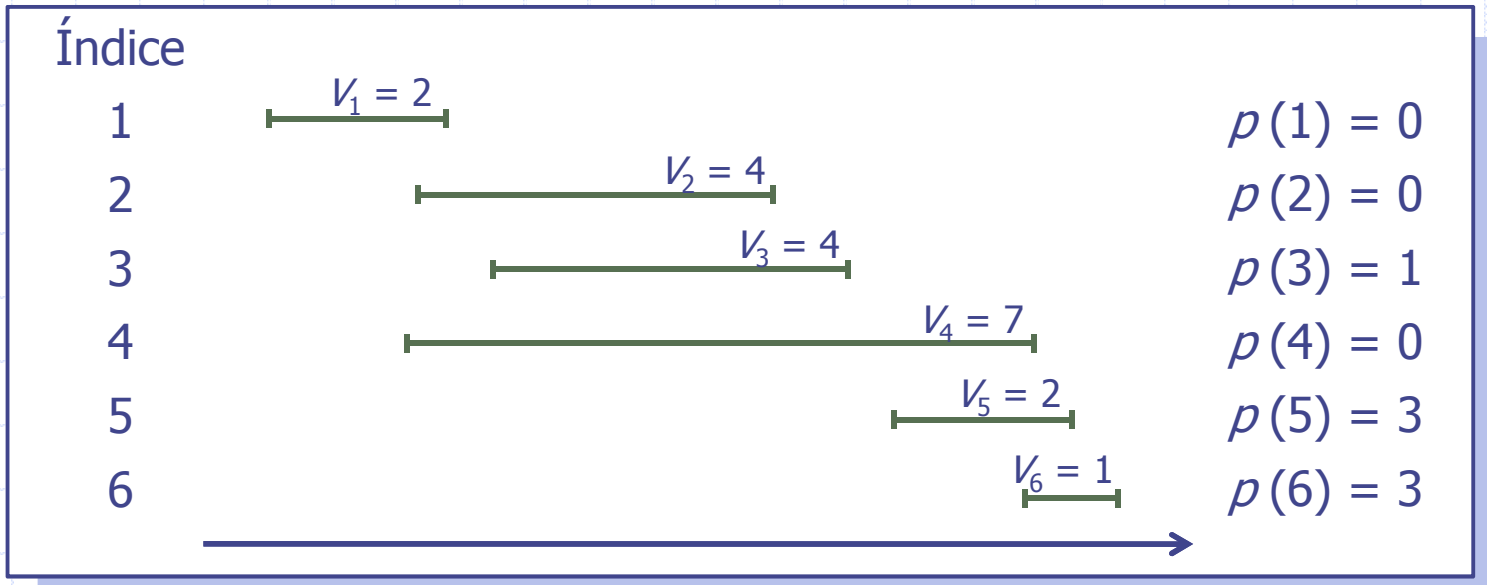
Exemplo 1 – WISP

- Vamos supor que os intervalos estão ordenados pelo tempo de término.
- E que definimos $p(j)$ como o maior índice $i < j$ tal que os intervalos i e j são disjuntos.



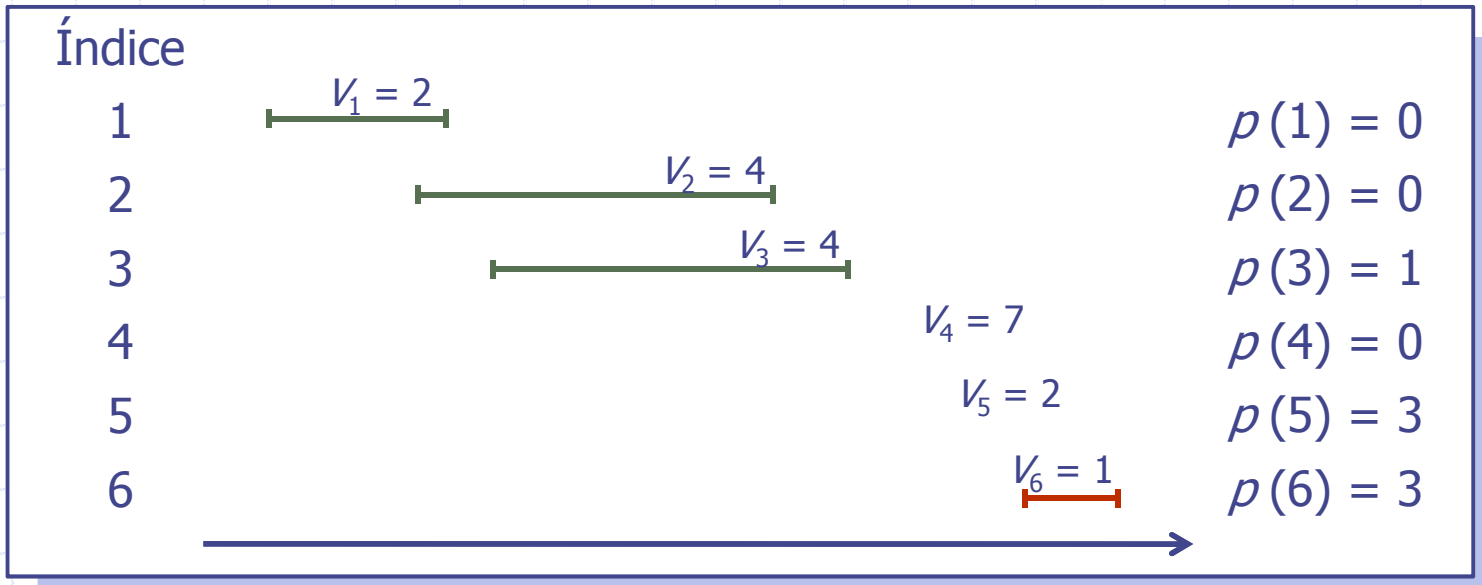
Exemplo 1 – WISP

- Um pouco mais formalmente:
  - Podemos rotular os sub-conjuntos 1,...,*n*
  - Estamos a procura de um sub-conjunto $S \subseteq \{1,...,n\}$ que maximize $\Sigma_{i \in S} v_i$



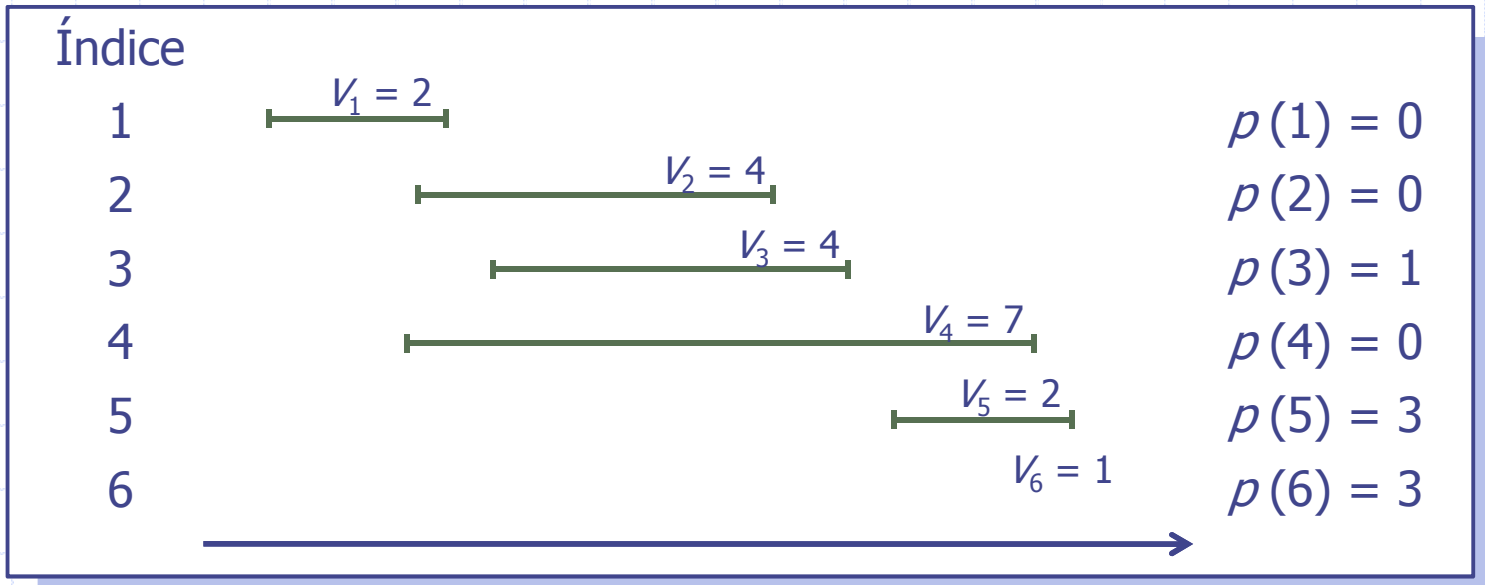
Exemplo 1 – WISP

- Uma coisa óbvia que podemos dizer sobre *S*:
  - Ou *n* (último intervalo) pertence a *S*, ou *n* não pertence a *S*
  - Se $n \in$ S, então nenhum intervalo entre $p(n)+1$ e $n-1 \in$ S
  - Ainda, *S* possui uma solução ótima para os intervalos $\{1, \ldots, p(n)\}$



Exemplo 1 – WISP

- Uma coisa óbvia que podemos dizer sobre *S*:
  - Ou *n* (último intervalo) pertence a *S*, ou *n* não pertence a *S*.
  - Se n $\notin$ S, então existe uma solução ótima com os intervalos do conjunto {1, …, *n*–1}



Exemplo 1 – WISP

- Encontrar uma solução ótima no intervalo {1, 2,..., n} envolve encontrar soluções ótimas em um intervalo menor {1, 2, ..., j }.
- Seja OPT(j) a soma ótima dos intervalos para {1, 2, ..., j }. Então:
  - Se $j \in S$, OPT(j) = v_j + OPT(p (j))
  - Se $j \notin S$, OPT(j) = OPT(j - 1)

Exemplo 1 – WISP

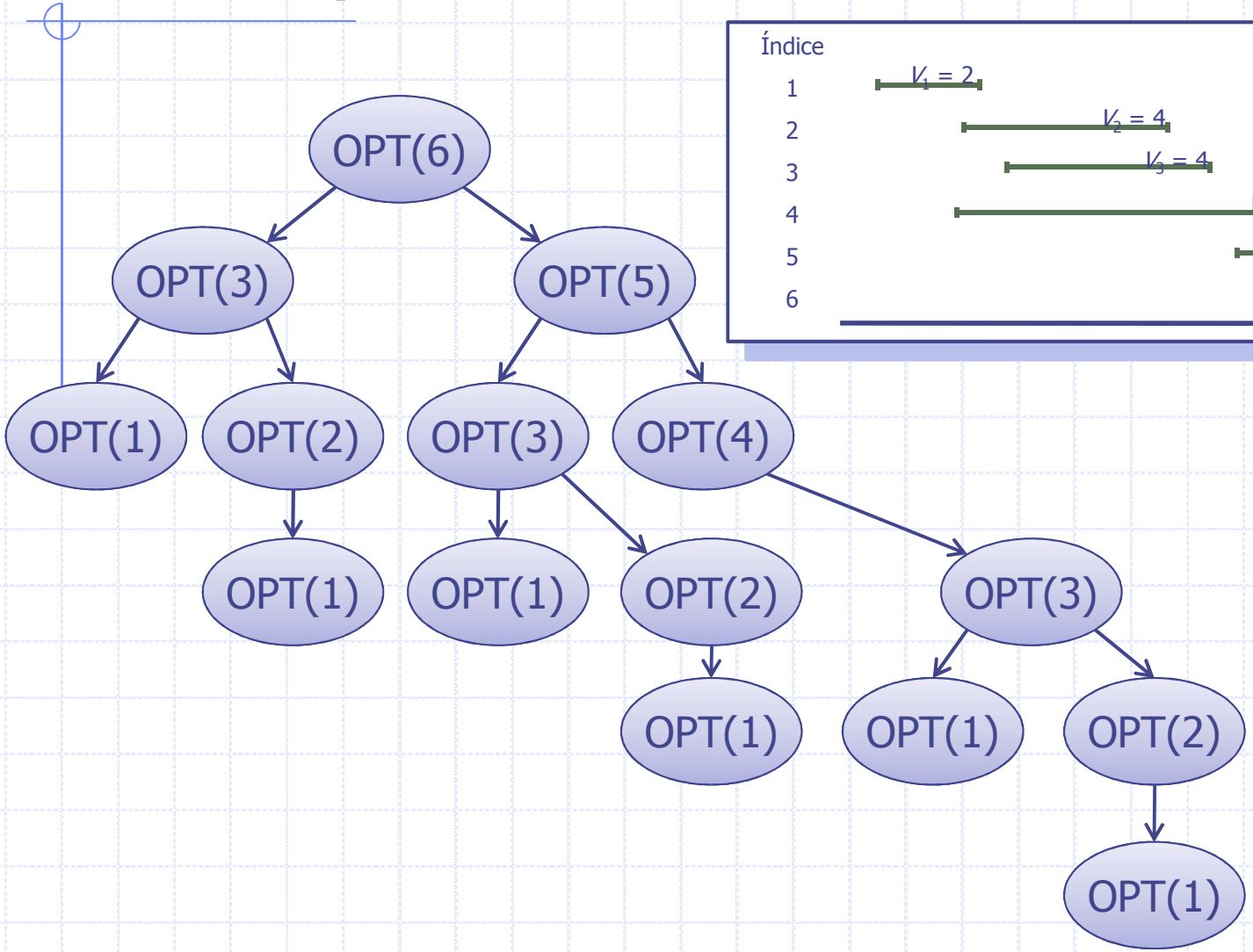
- Encontrar uma solução ótima no intervalo {1, 2,..., n} envolve encontrar soluções ótimas em um intervalo menor {1, 2, ..., j }.
- Seja OPT(j) a soma ótima dos intervalos para {1, 2, ..., j }. Então:
  - Se $j \in S$, OPT(j) = v_j + OPT(p (j))
  - Se $j \notin S$, OPT(j) = OPT(j - 1)
- Ou seja:

$$OPT(j) = \max(v_j + OPT(p(j)) , OPT(j - 1))$$

Exemplo 1 – WISP

```
Compute-Opt(j)
    if j = 0 then
        return 0
    else
        return max(v[j] + Compute-Opt(p(j)), Compute-Opt(j-1))
end
```

Exemplo 1 – WISP



Exemplo 1 – WISP

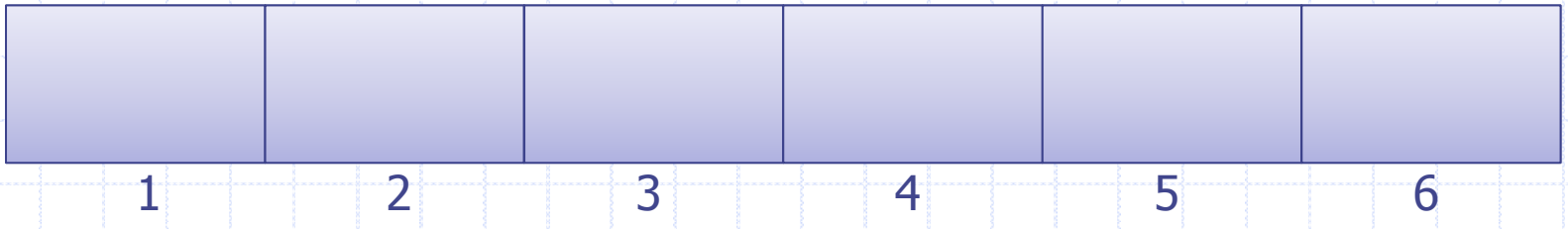
- Existe uma grande ineficiência nessa árvore de execução, por exemplo:
  - OPT(1) é calculado 6 vezes
  - OPT(2) é calculado 3 vezes, e assim por diante.
- A complexidade do procedimento *Compute-Opt* é exponencial.
  - Pode-se mostrar que o número de chamadas de Compute-Opt cresce como a série de Fibonacci
  - Que por sua vez cresce exponencialmente.

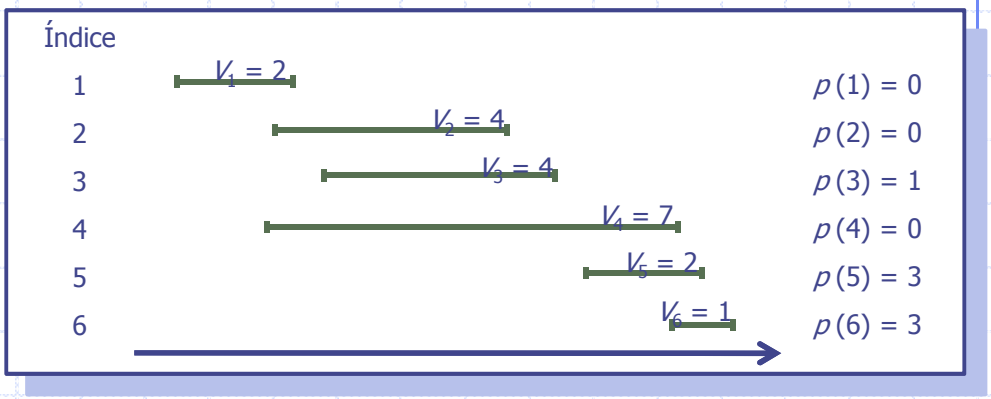
Exemplo 1 – WISP

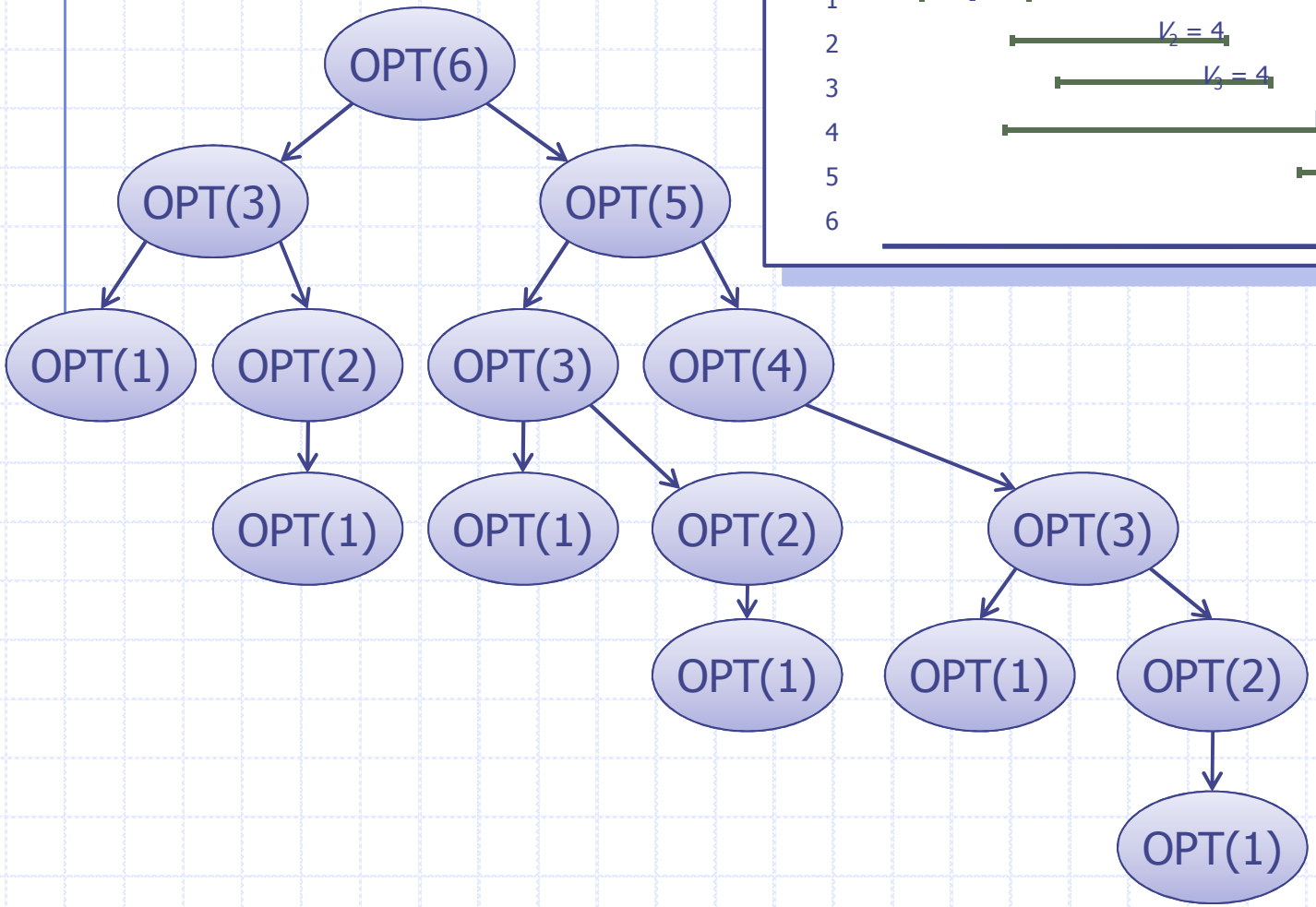
- Na verdade *Compute-Opt* somente calcula as soluções para $n + 1$ sub-problemas.
- Uma solução para esse problema é a *memoização*, que consiste em armazenar as soluções para problemas parciais em uma estrutura global.

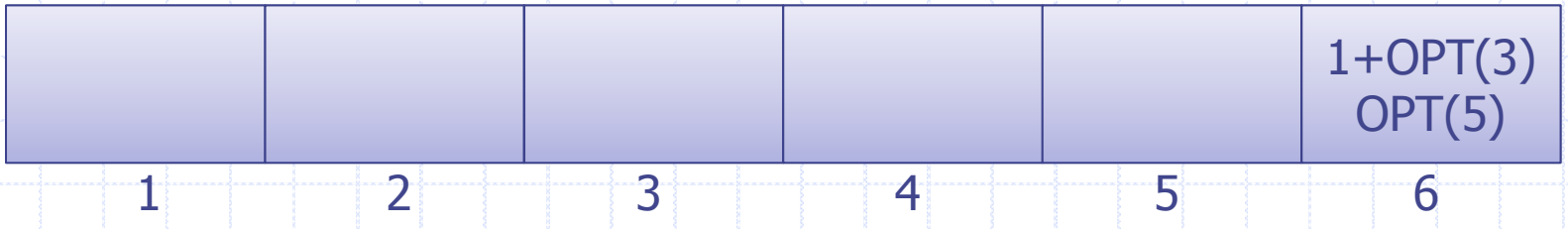
Exemplo 1 – WISP

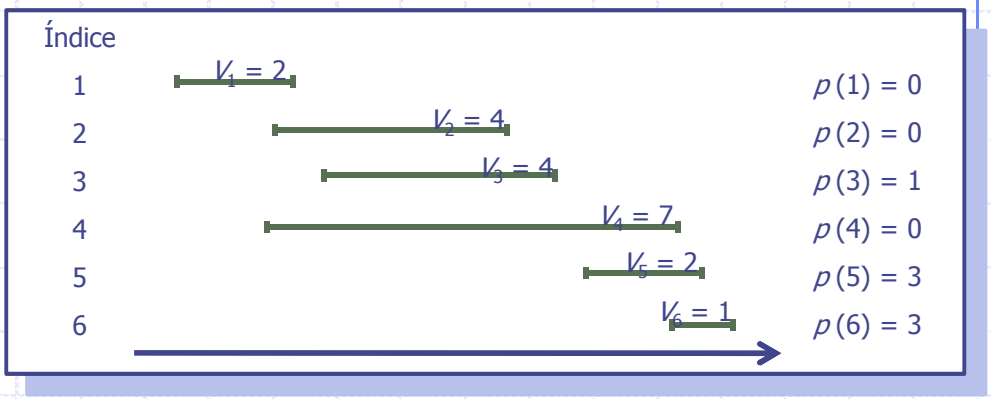
```
M-Compute-Opt(j)
    if j = 0 then
        return 0
    else if M[j] is not empty then
            return M[j]
        else
            M[j] = max(v[j] + Compute-Opt(p(j)), Compute-Opt(j-1))
            return M[j]
end
```

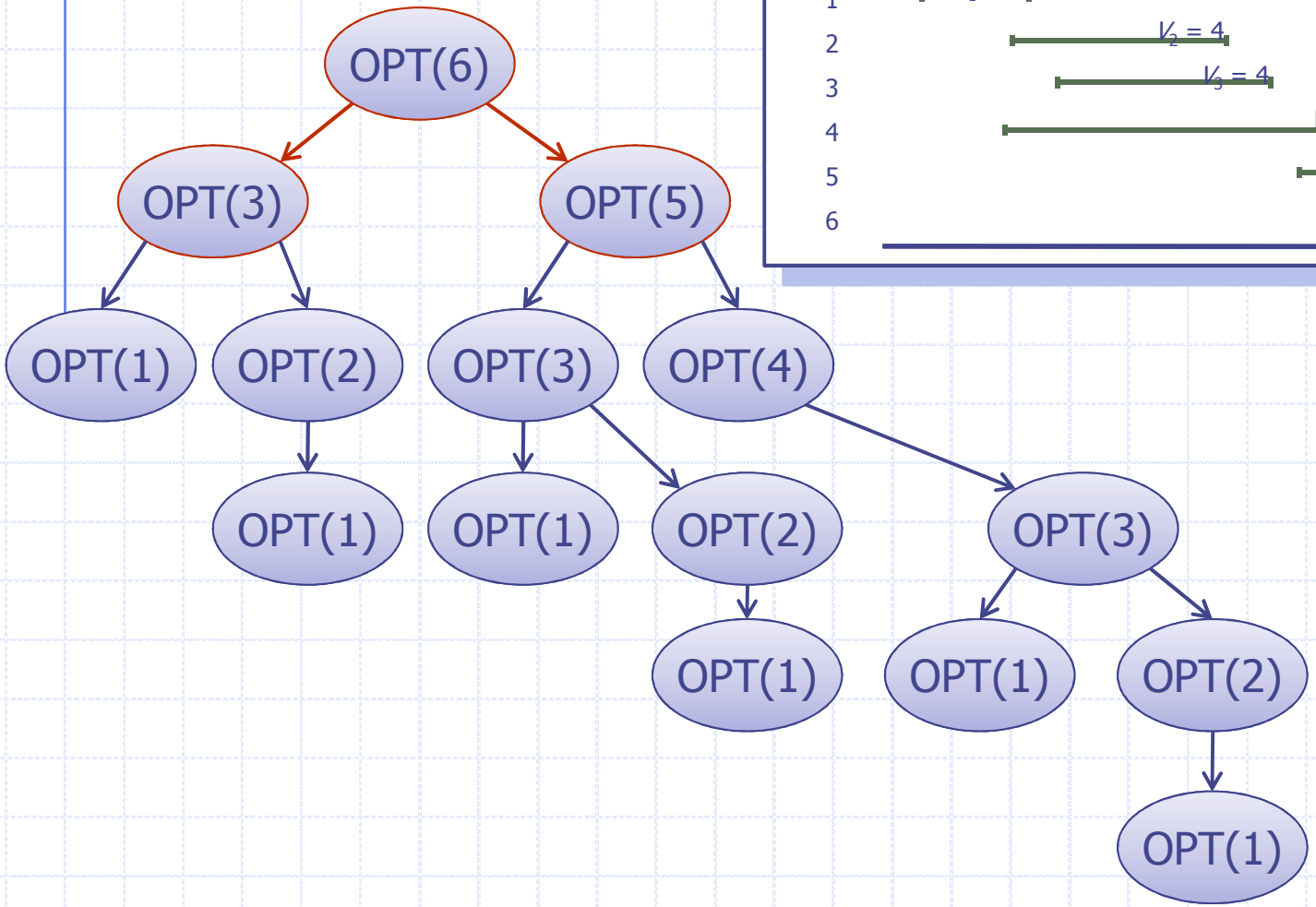

1
2
3
4
5
6

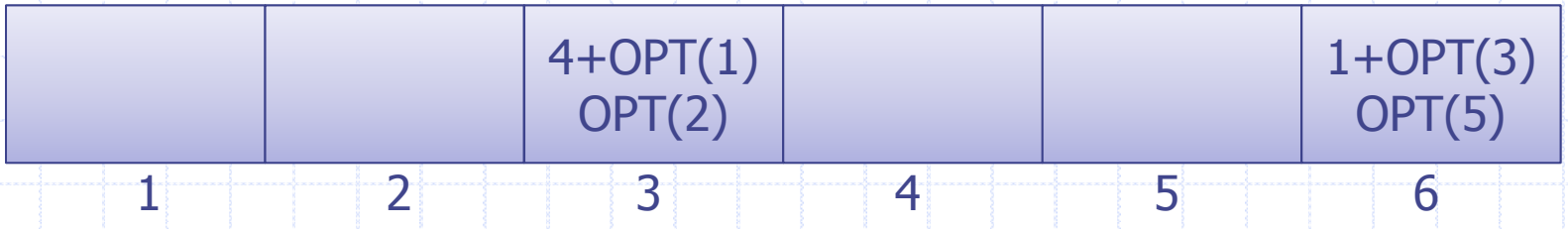

Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

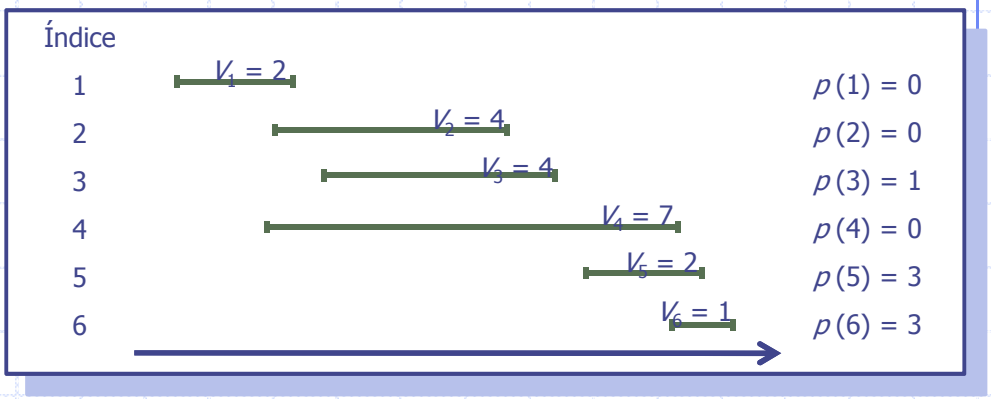

OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

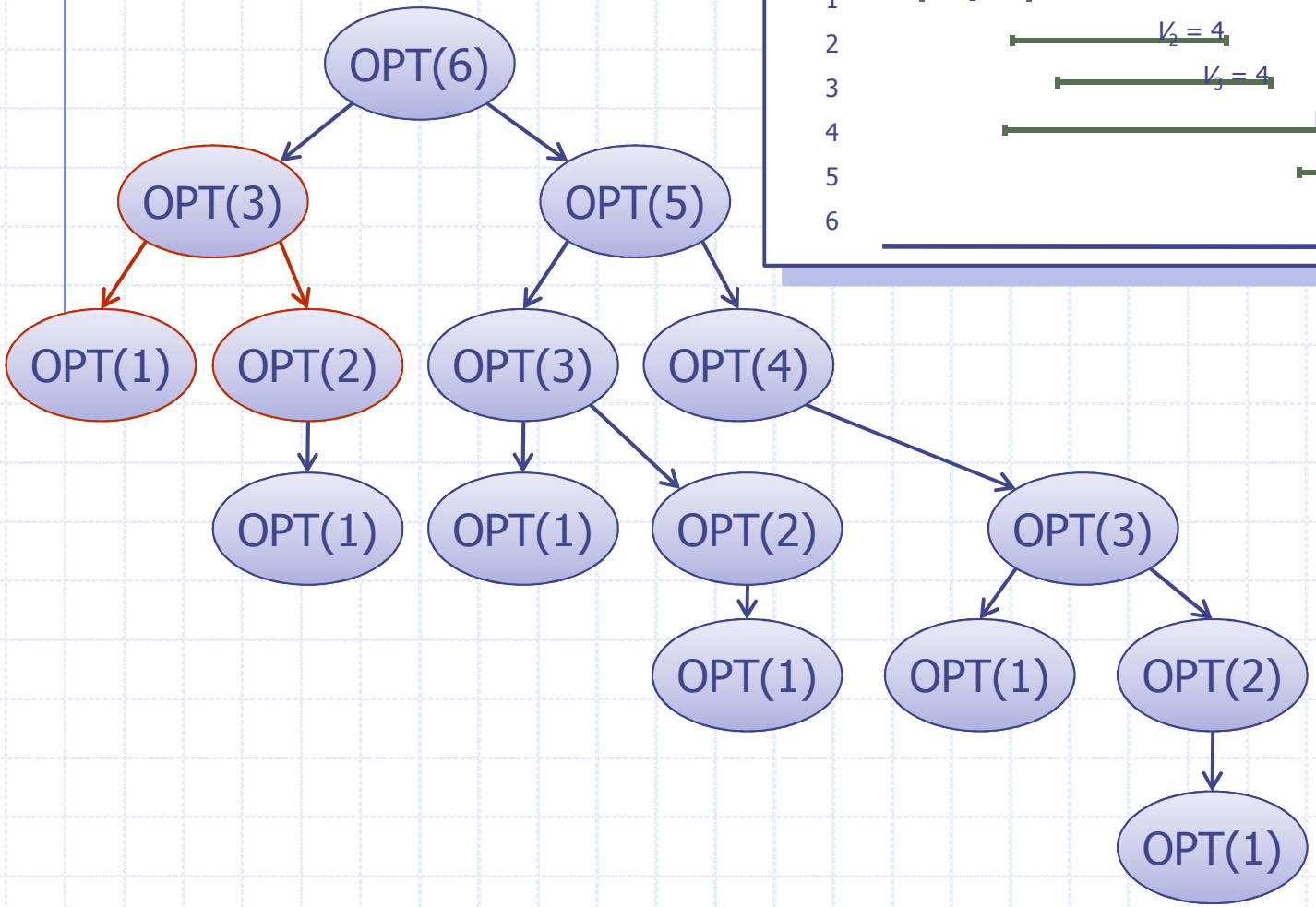

1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6

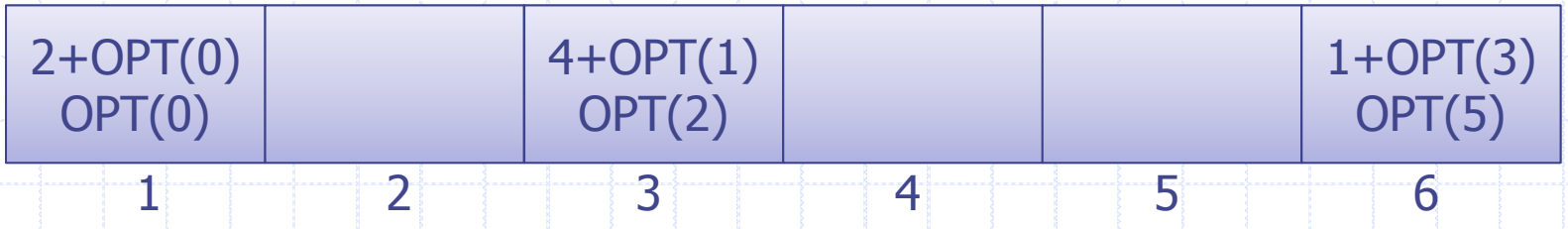

Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

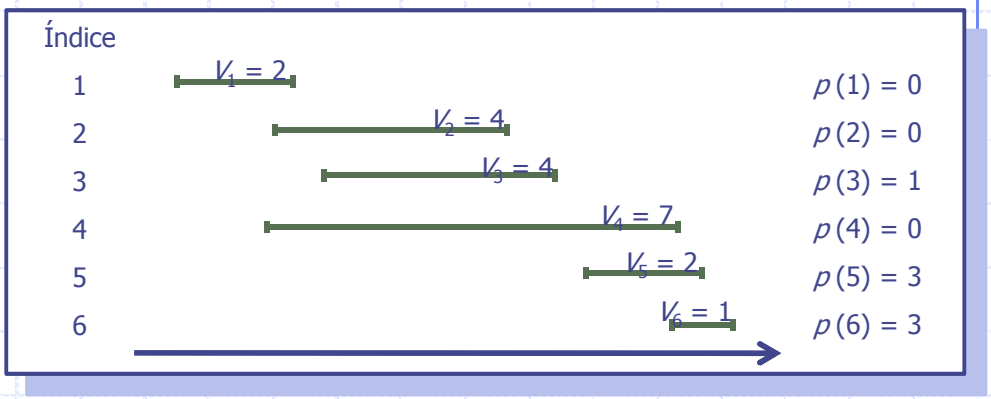

OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

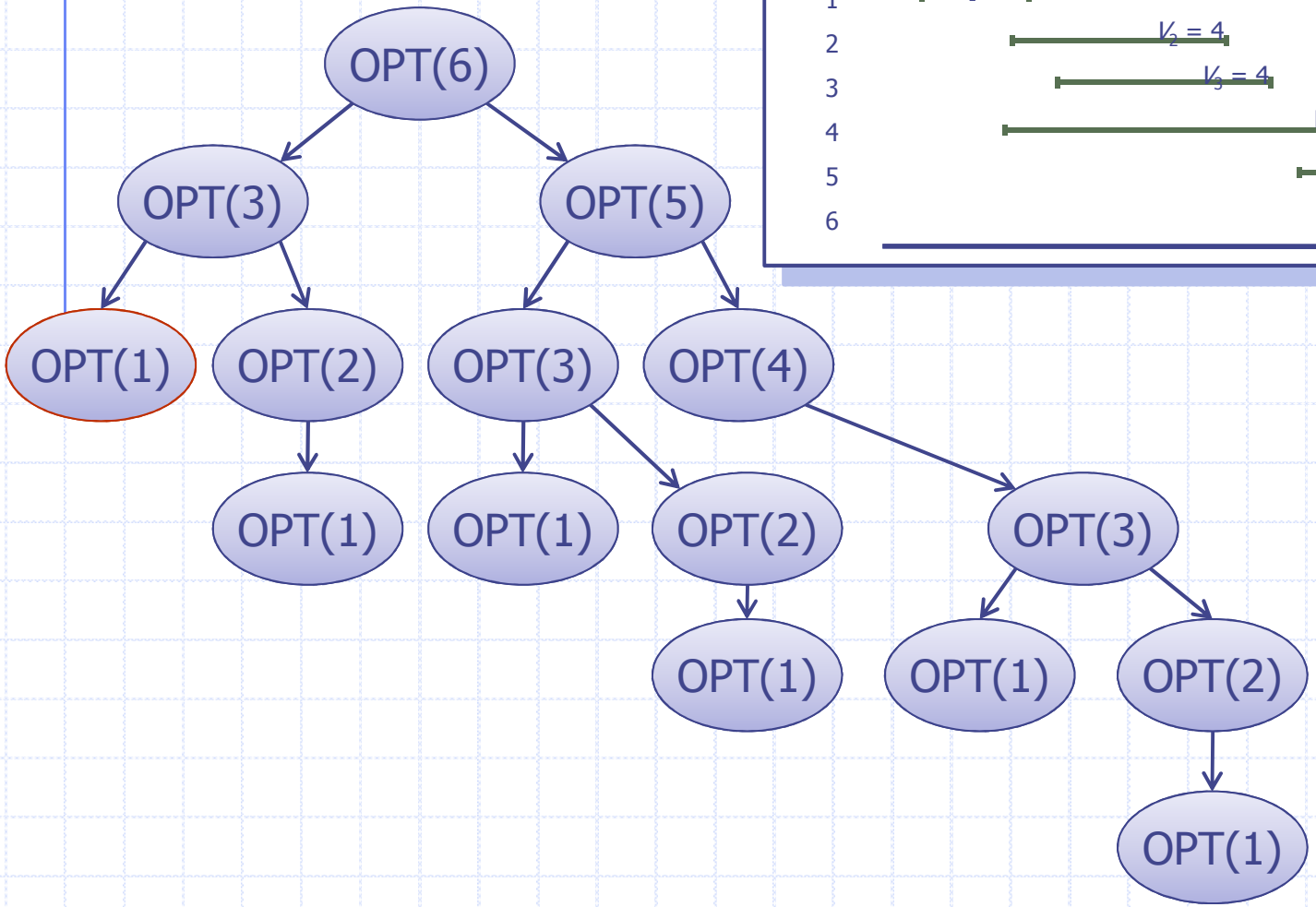

4+OPT(1)
OPT(2)
1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6

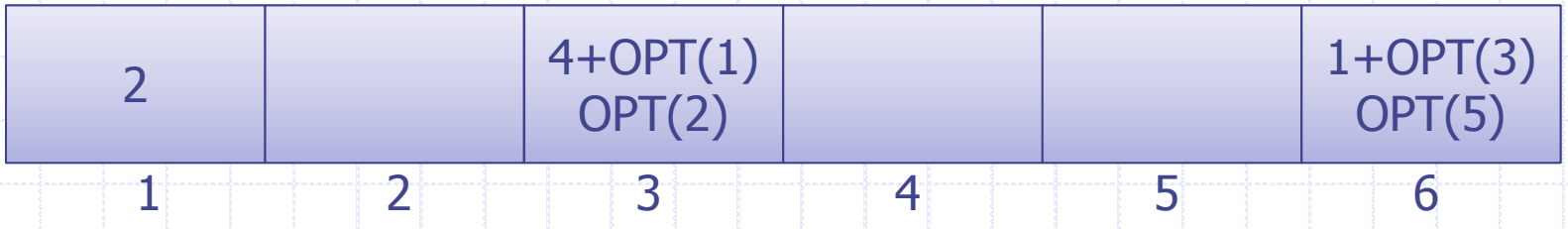

Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

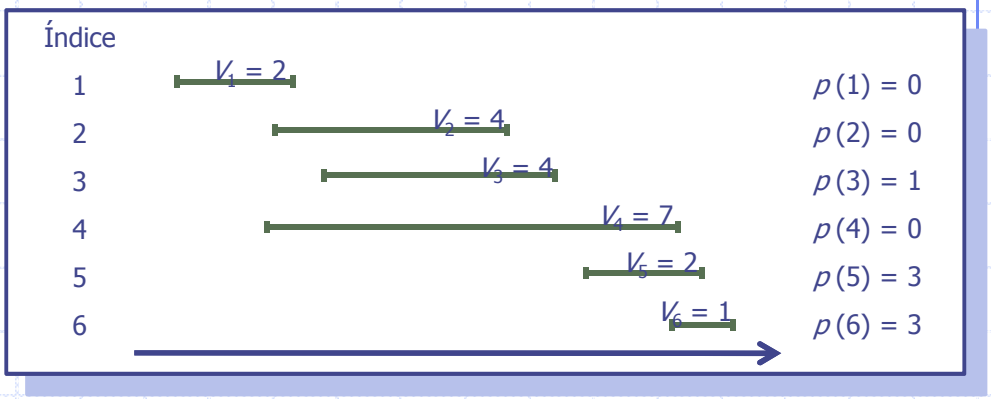

OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

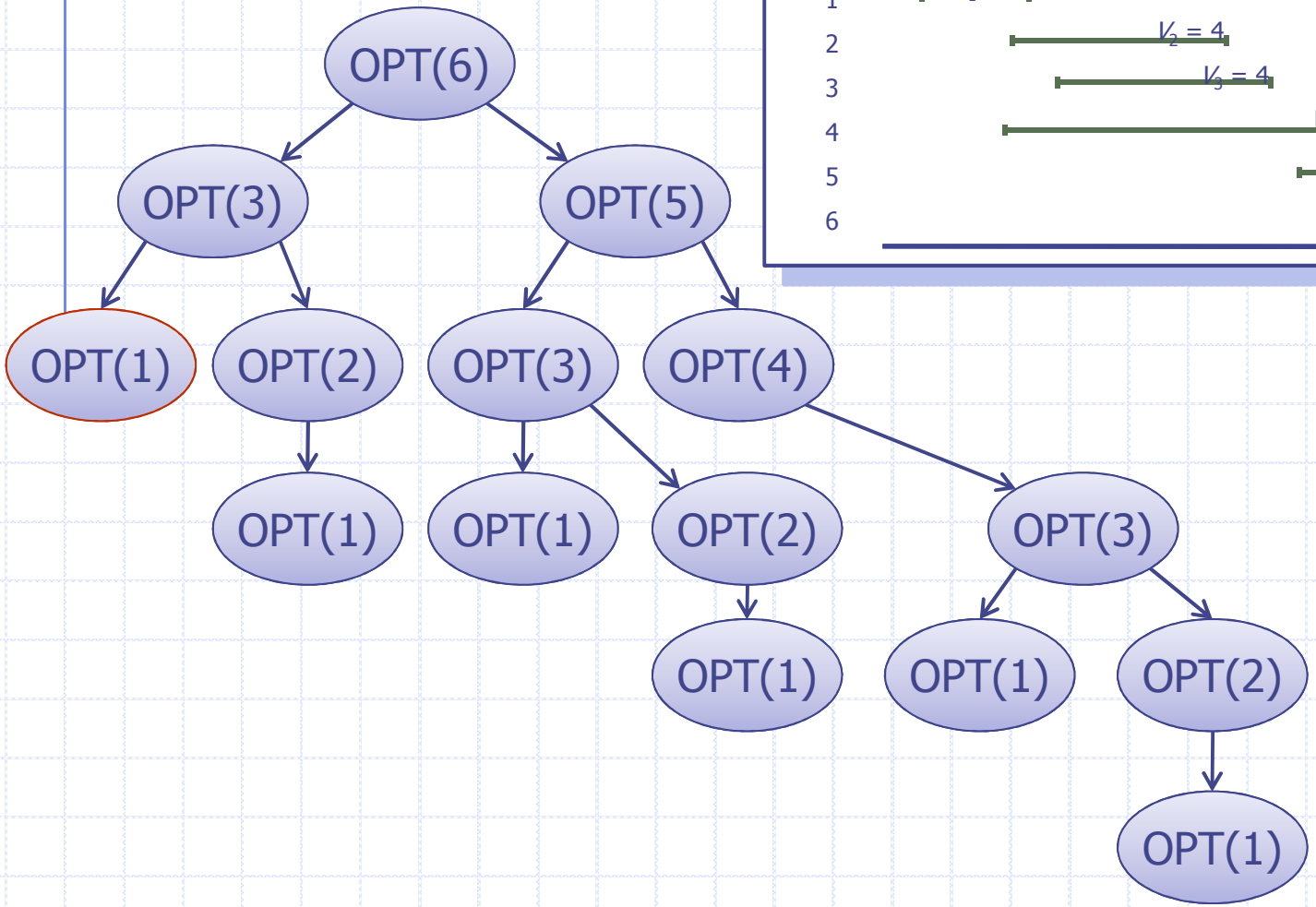

2+OPT(0)
OPT(0)
4+OPT(1)
OPT(2)
1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6

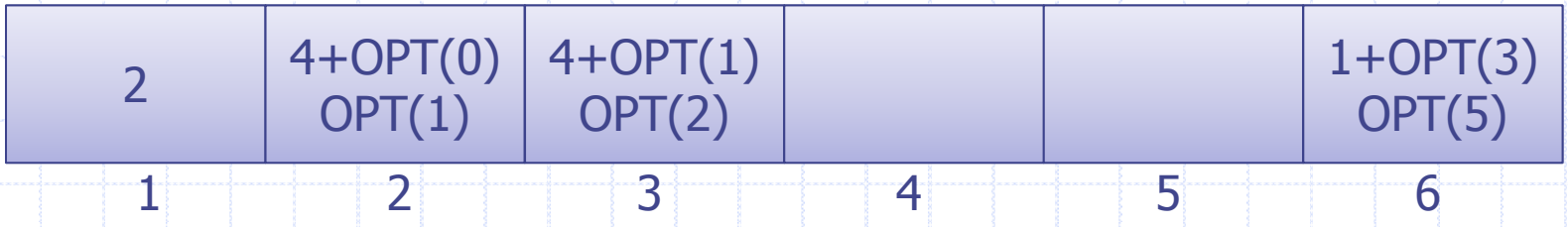

Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

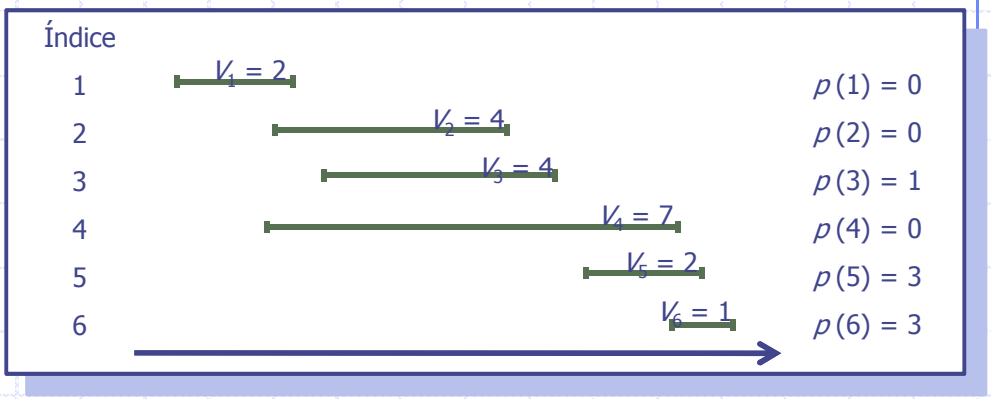

OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

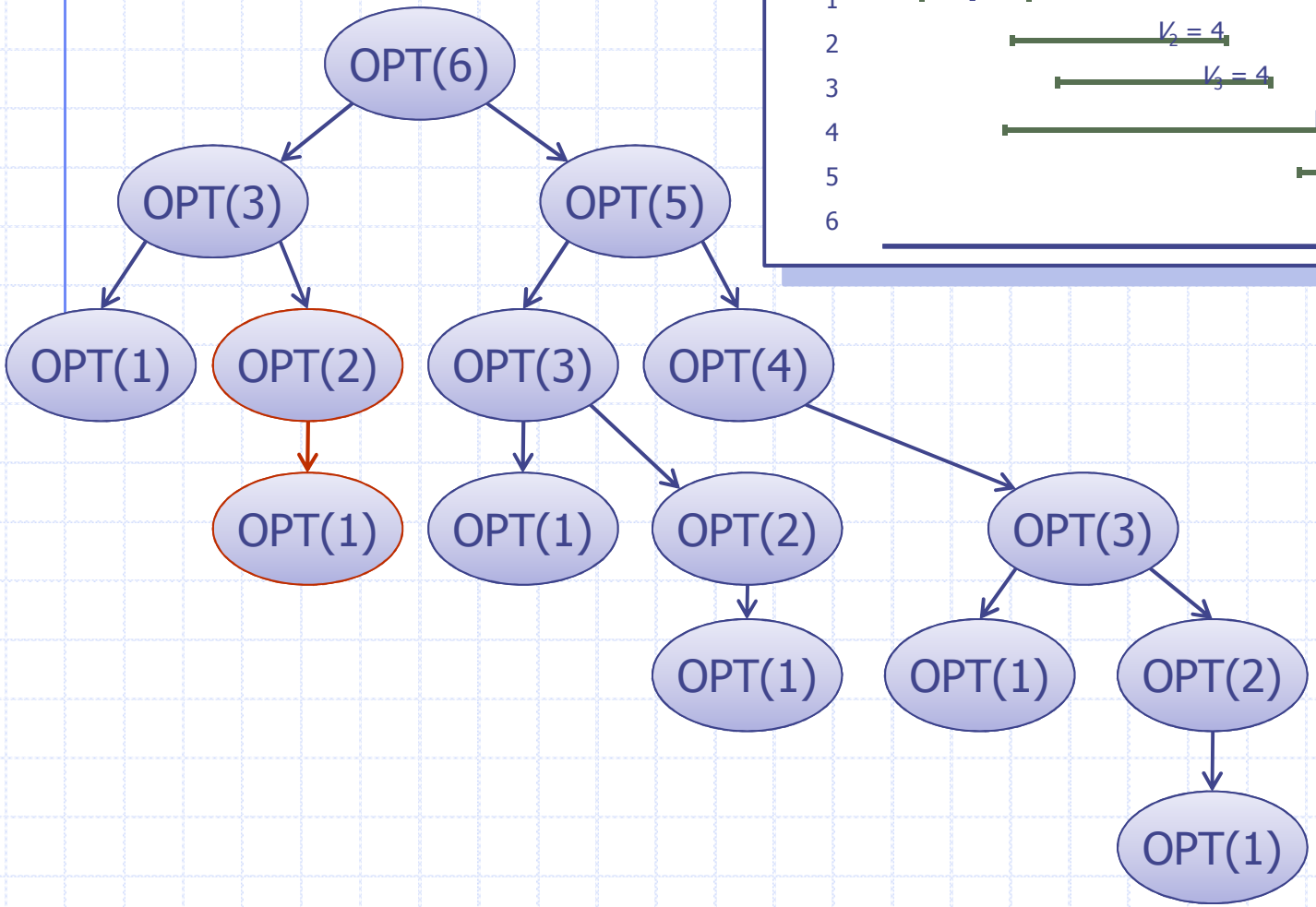

2
4+OPT(1)
OPT(2)
1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6

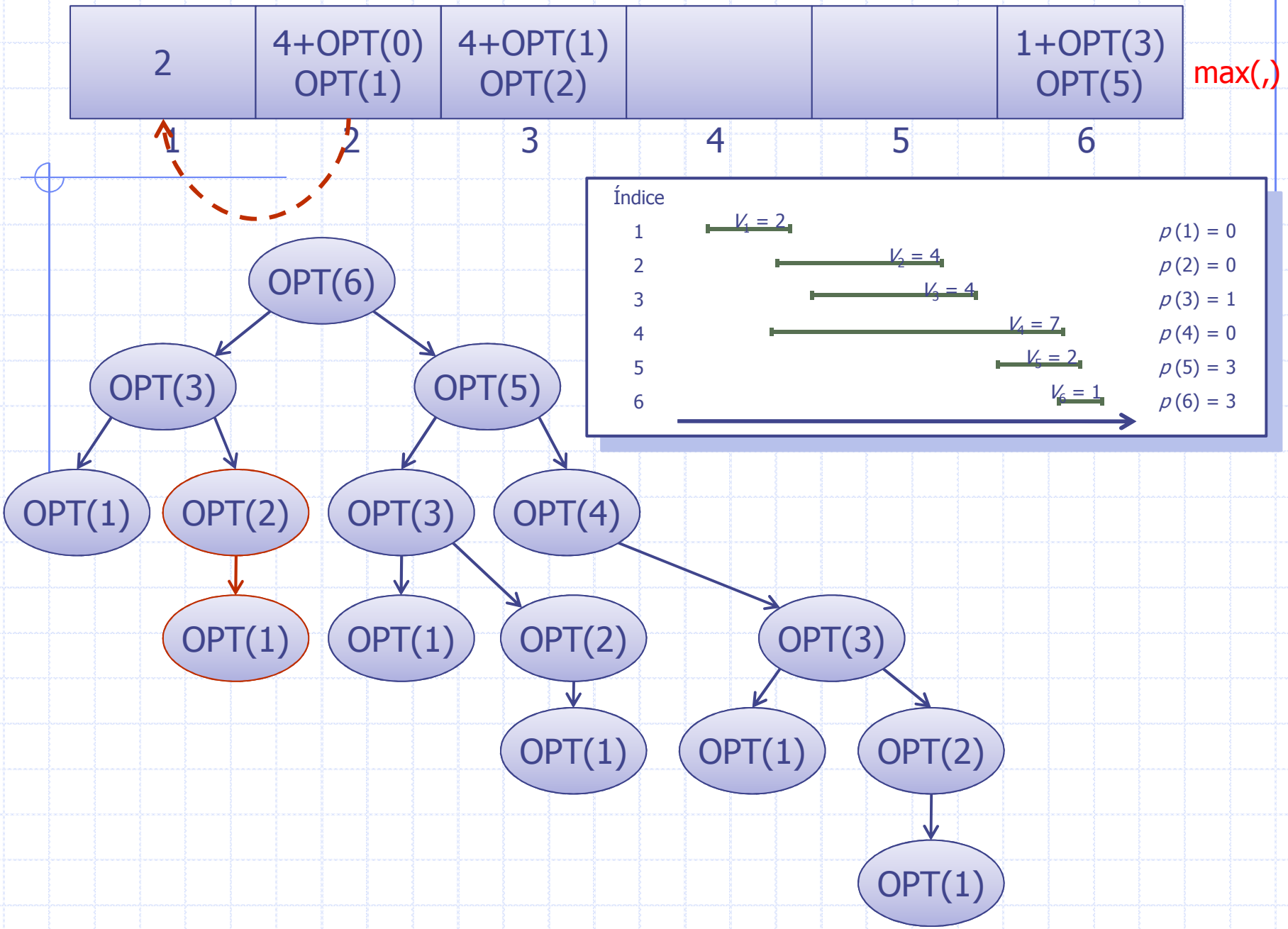

Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$


OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)


2
4+OPT(0)
OPT(1)
4+OPT(1)
OPT(2)
1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6

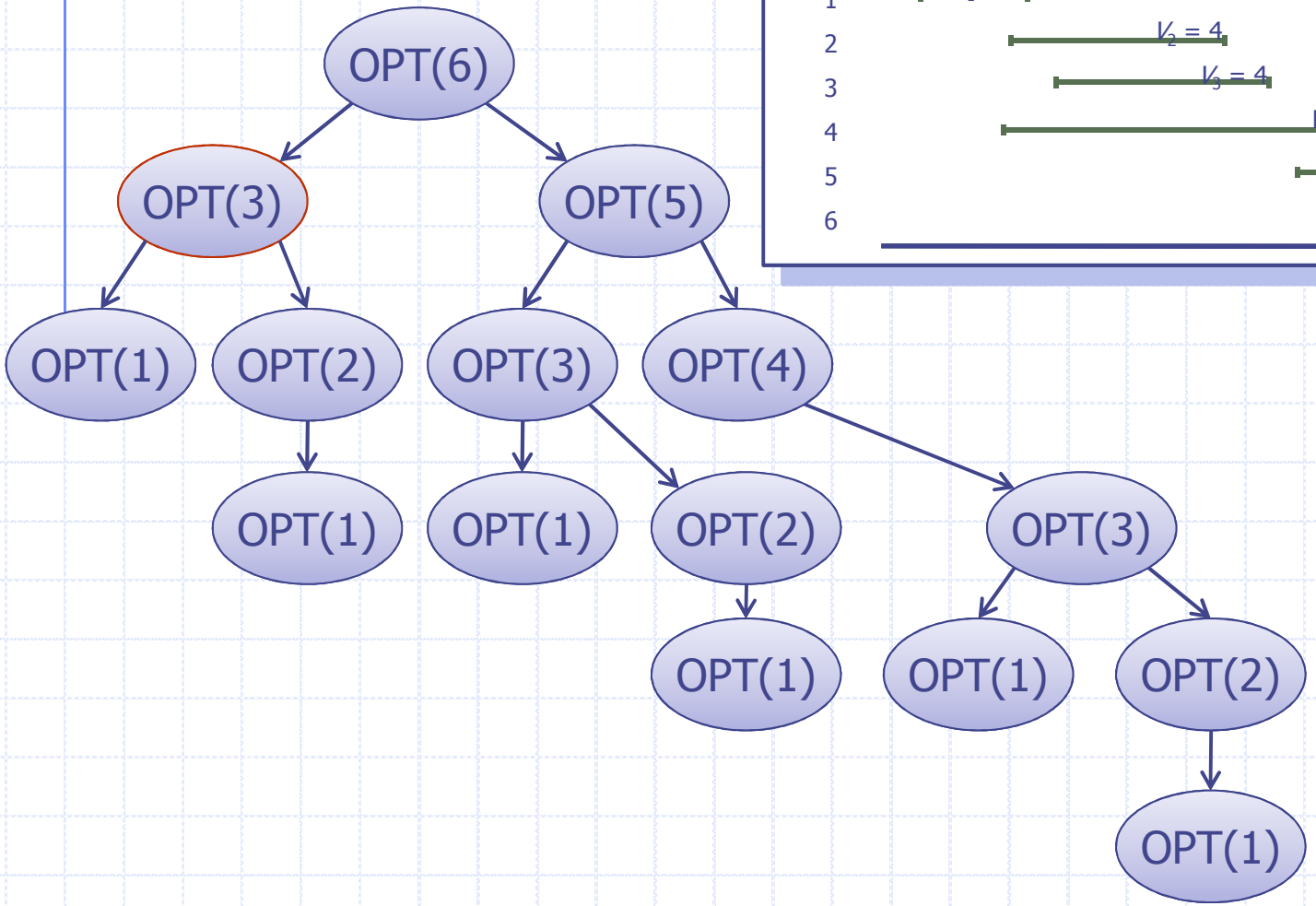

Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

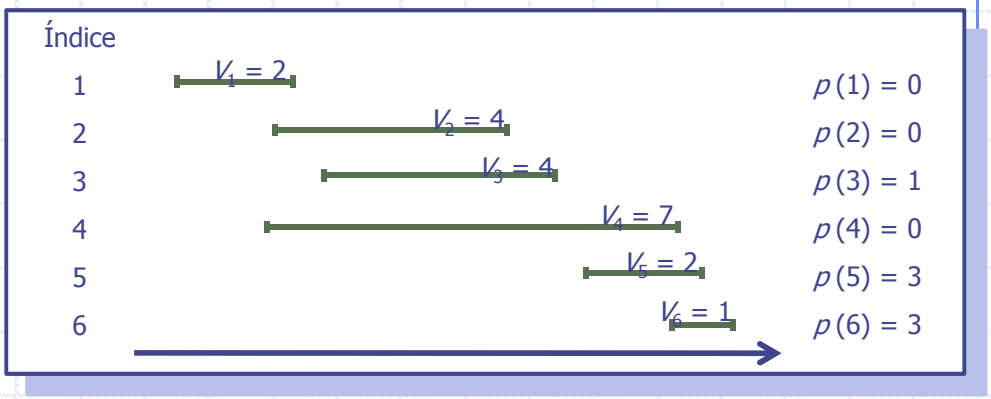

OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

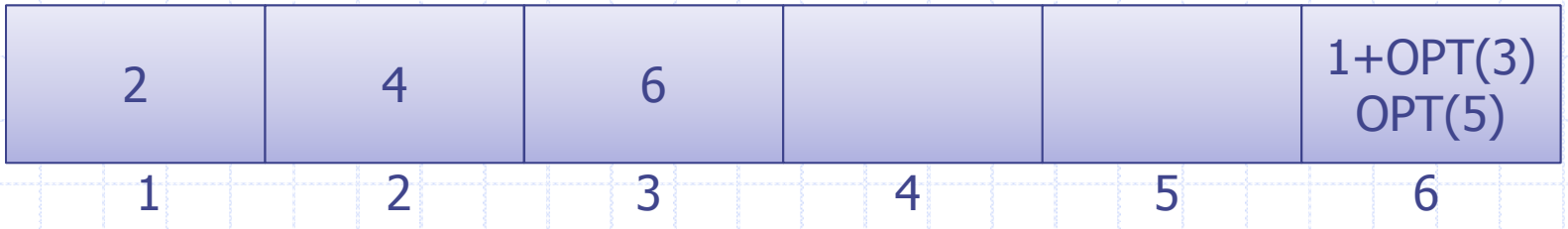

2
4+OPT(0)
OPT(1)
4+OPT(1)
OPT(2)
1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6
Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$
OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

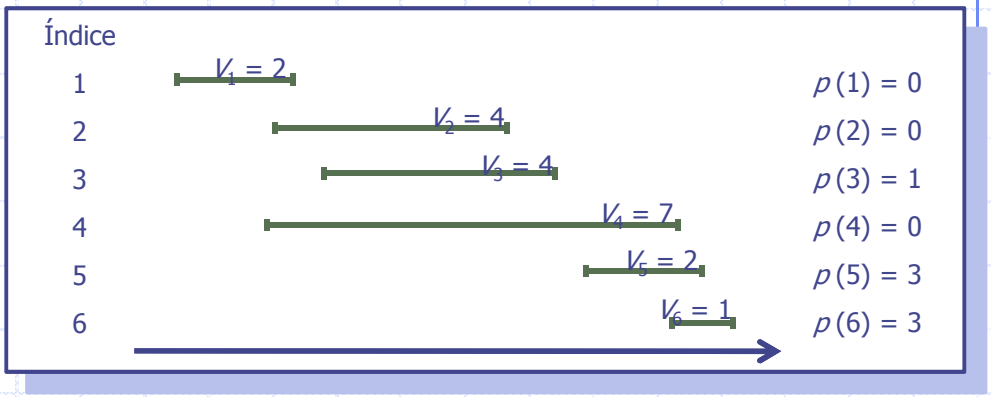
2	4	4+OPT(1) OPT(2)			1+OPT(3) OPT(5)
1	2	3	4	5	6

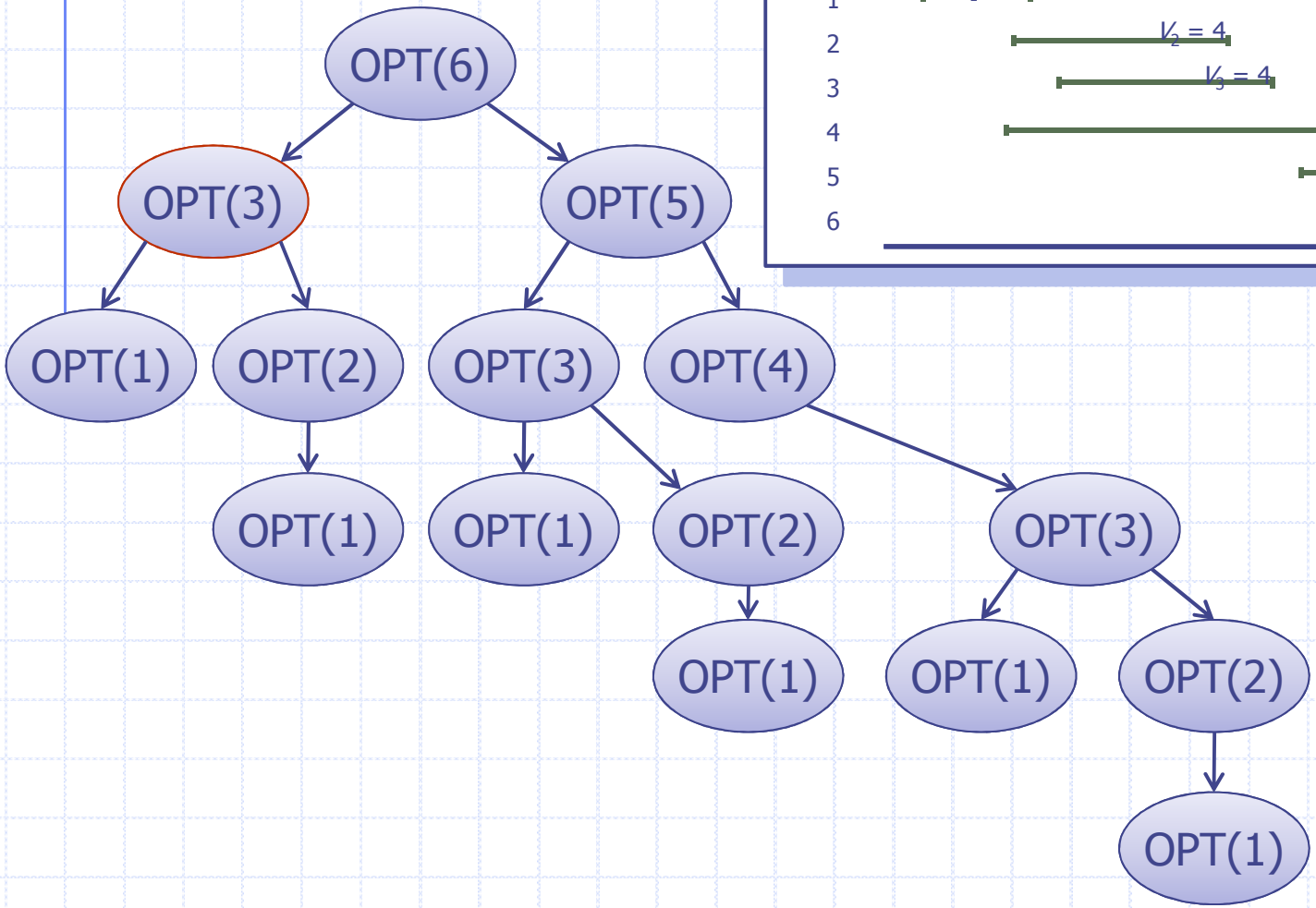
max(,)

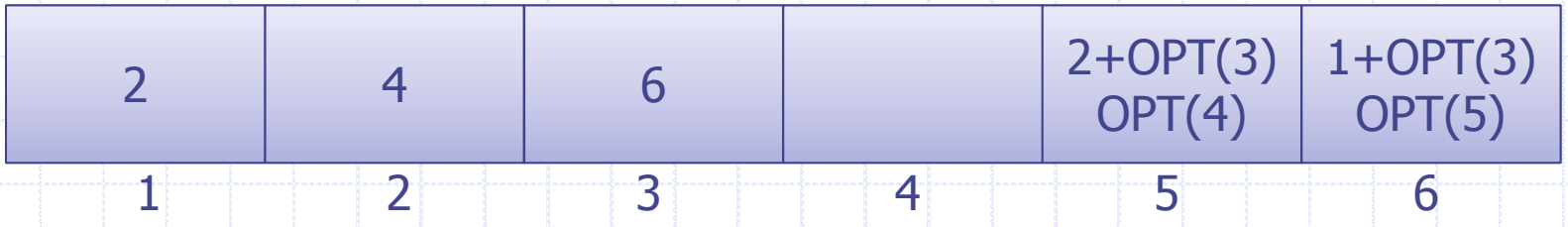


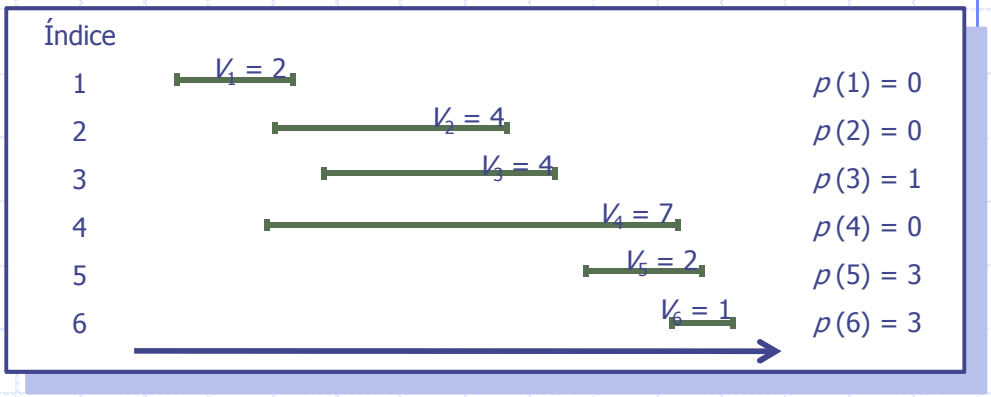


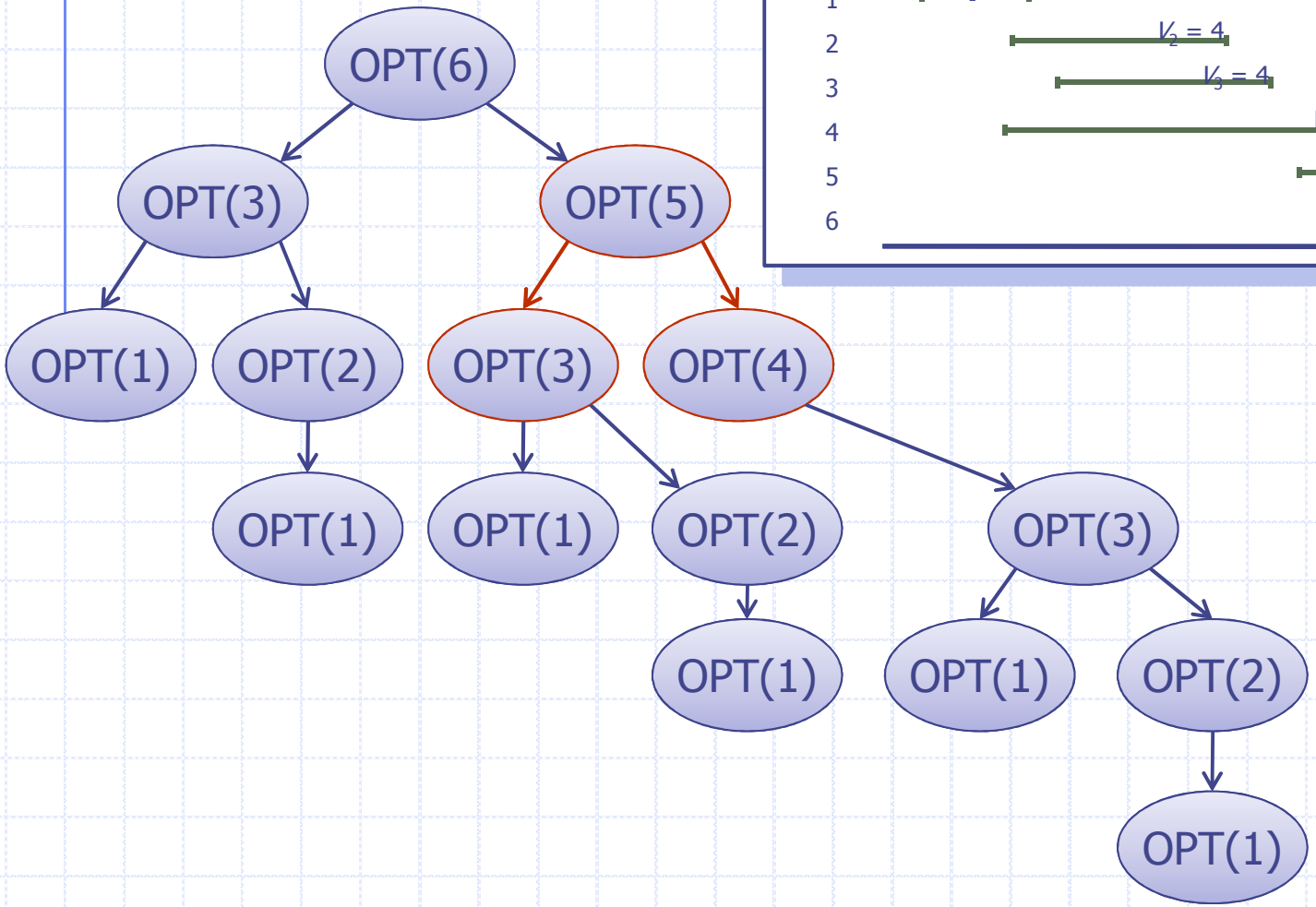

2
4
6
1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6


Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$


OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

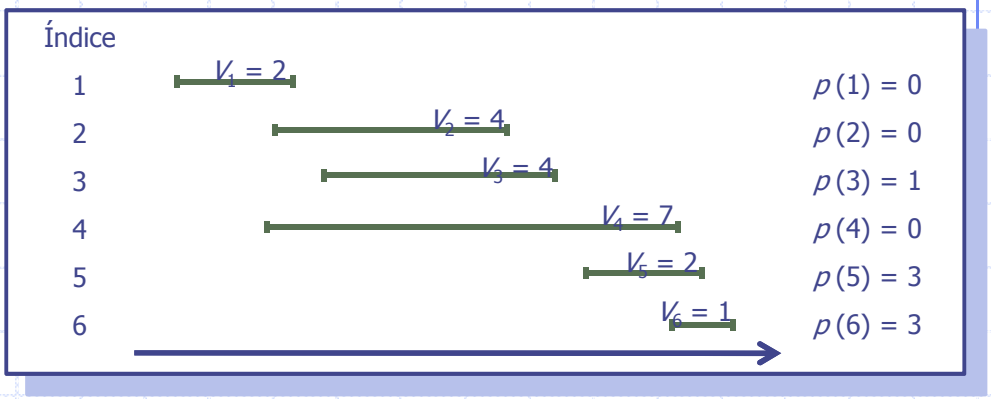

2
4
6
2+OPT(3)
OPT(4)
1+OPT(3)
OPT(5)
max(,)
1
2
3
4
5
6

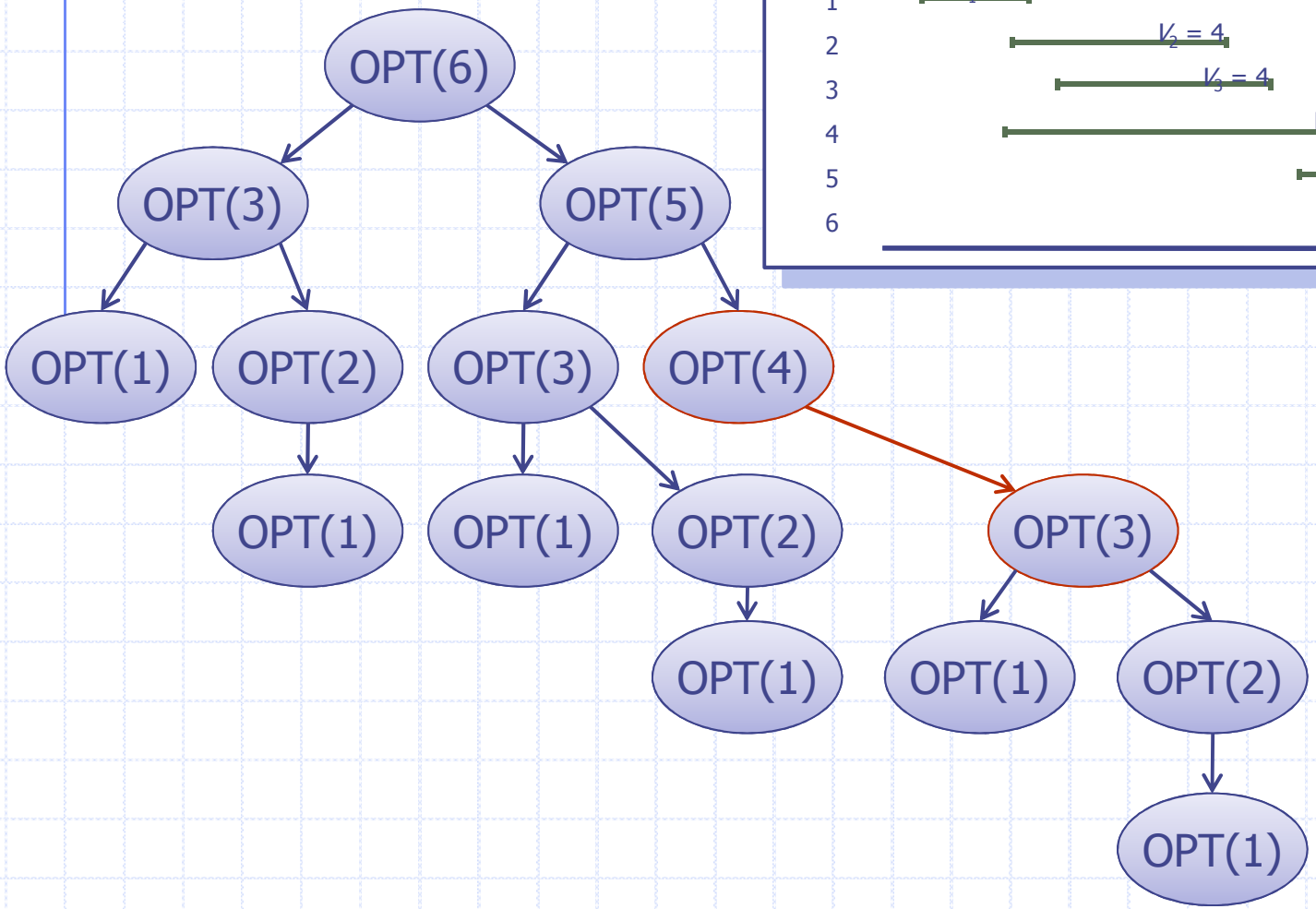

Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

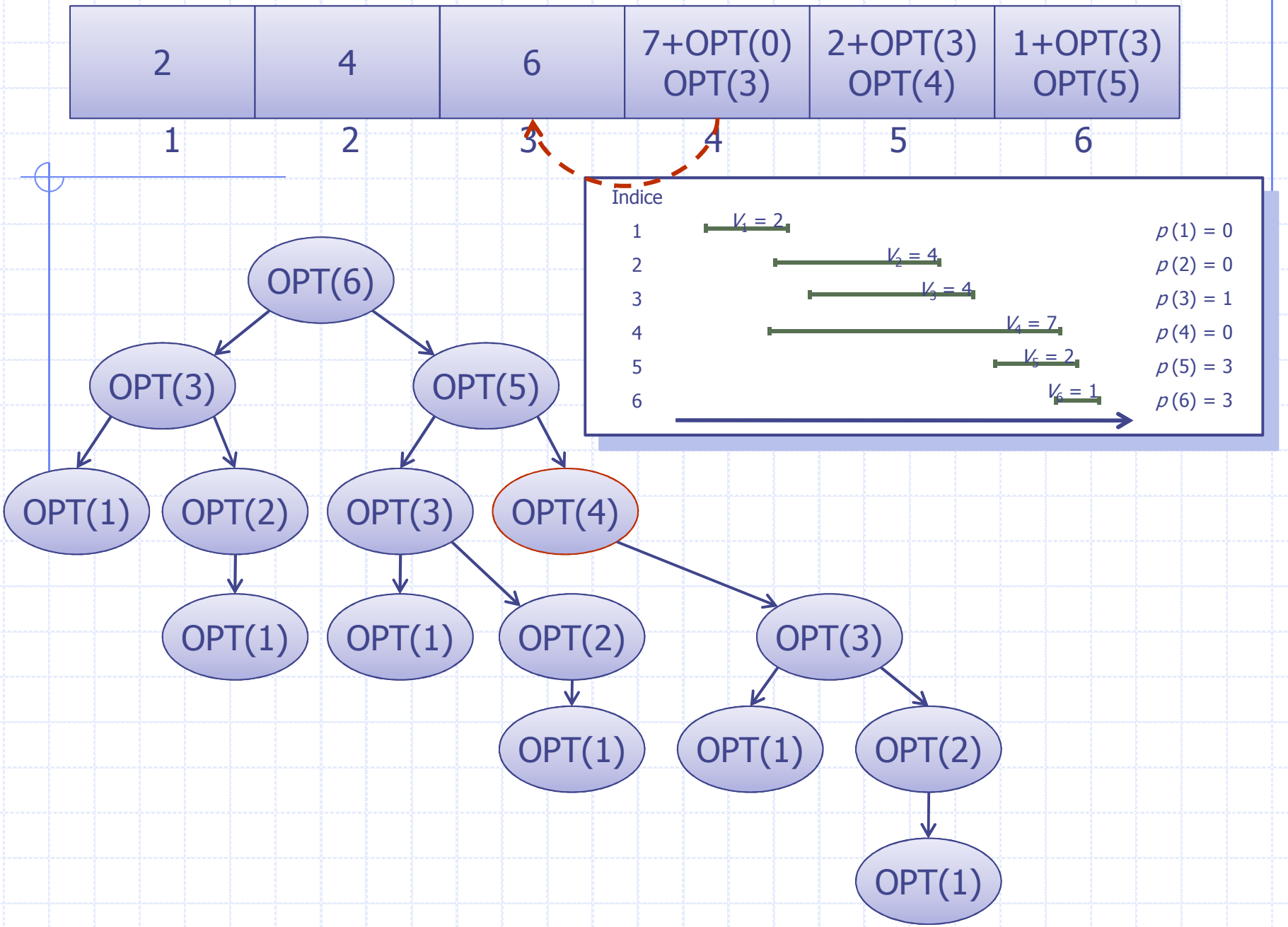

OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

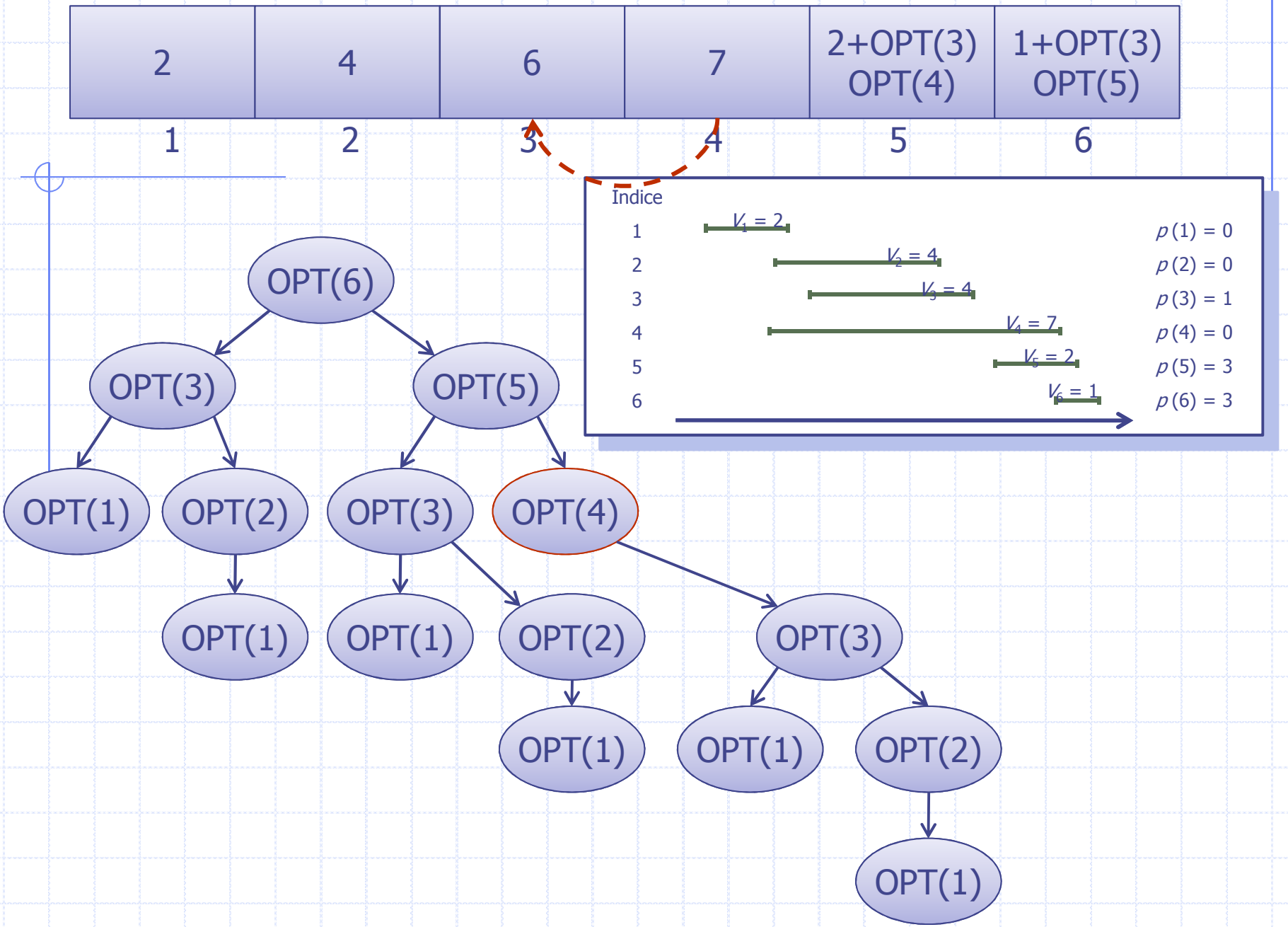
2	4	6	7+OPT(0) OPT(3)	2+OPT(3) OPT(4)	1+OPT(3) OPT(5)
1	2	3	4	5	6

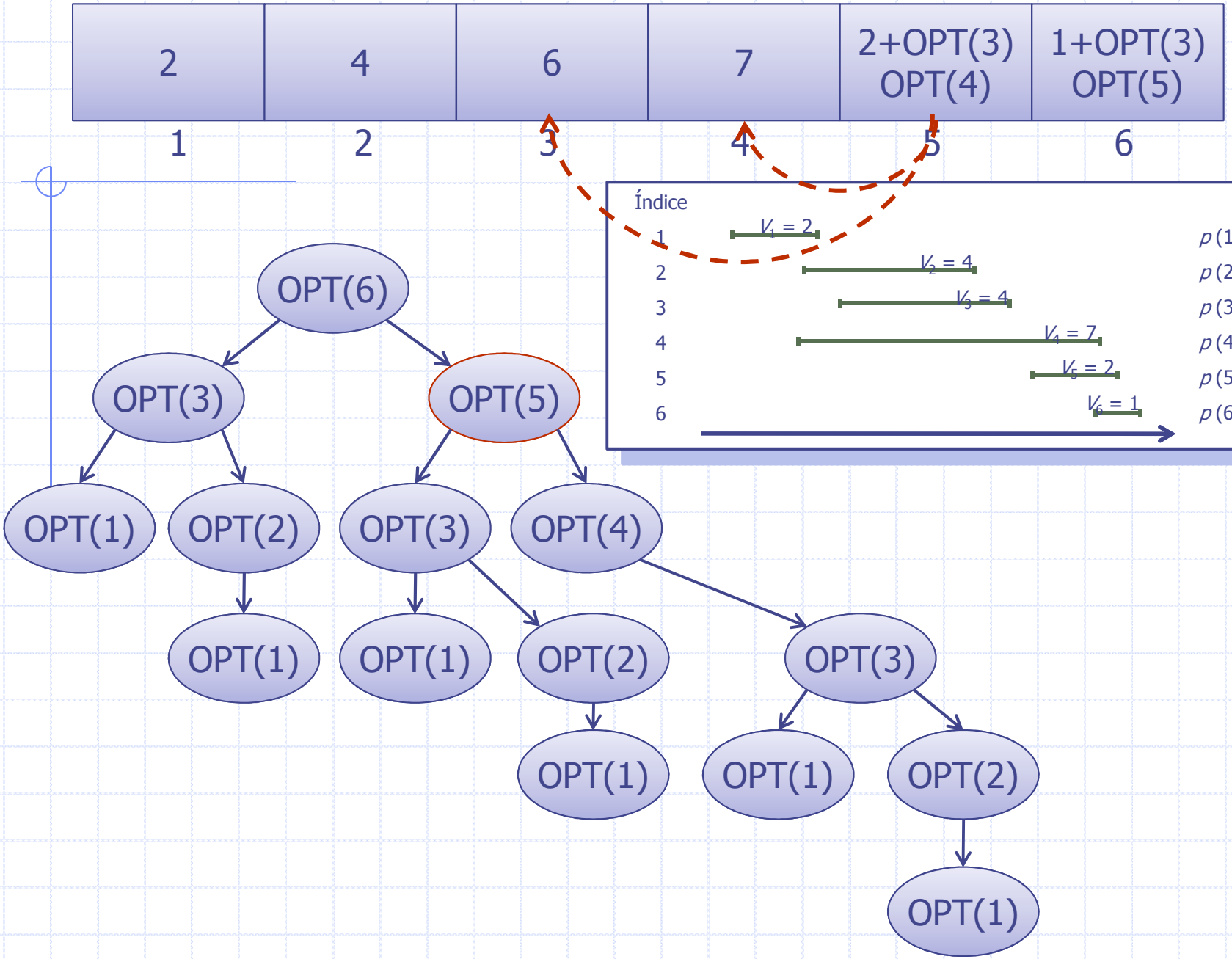
max(,)

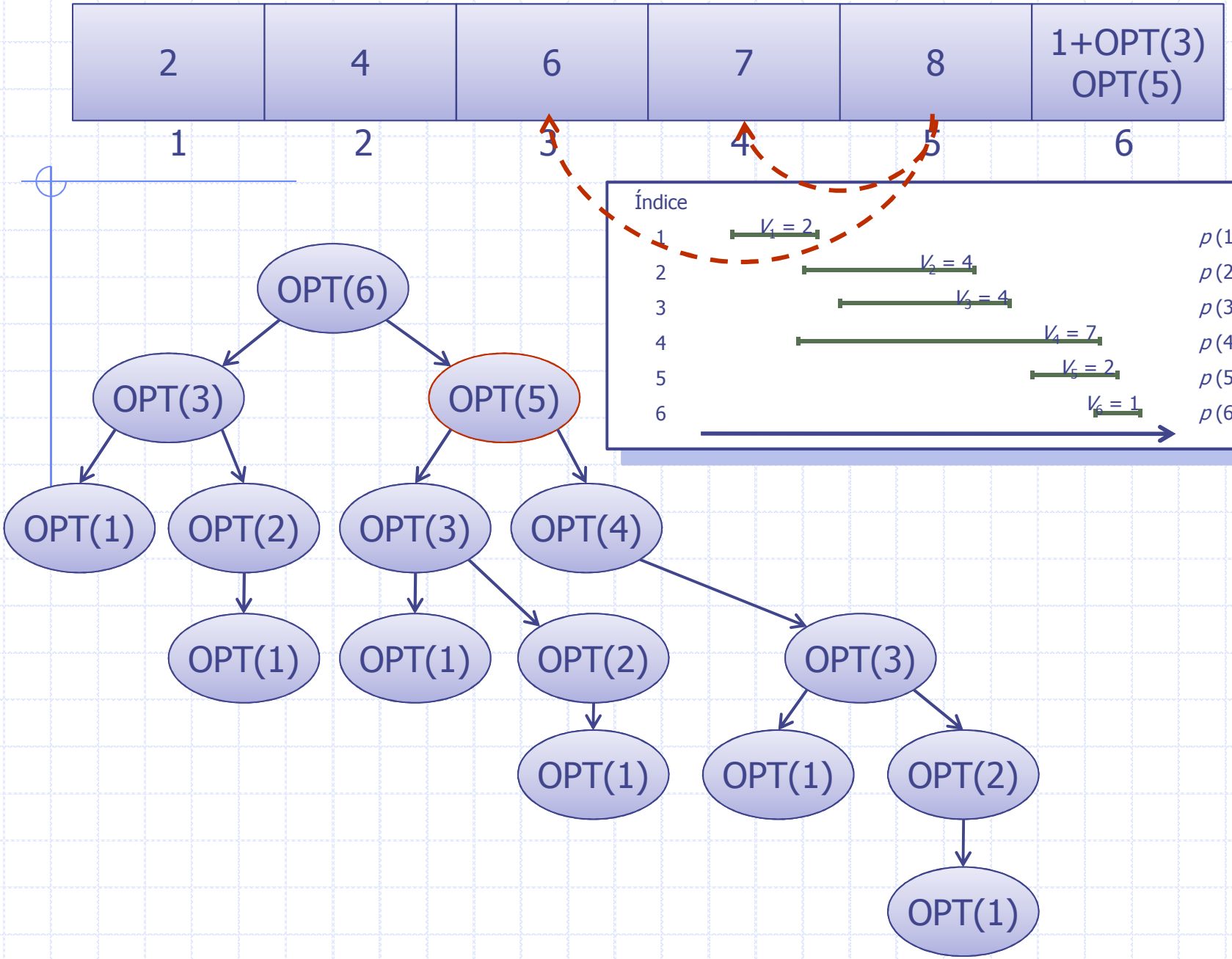


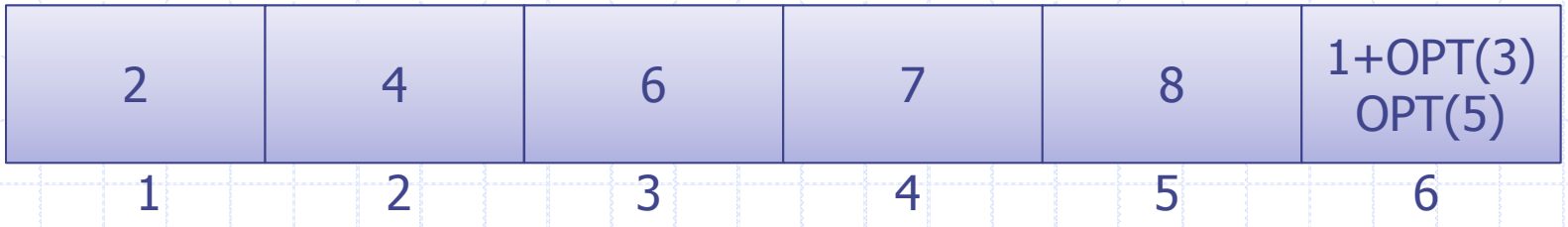


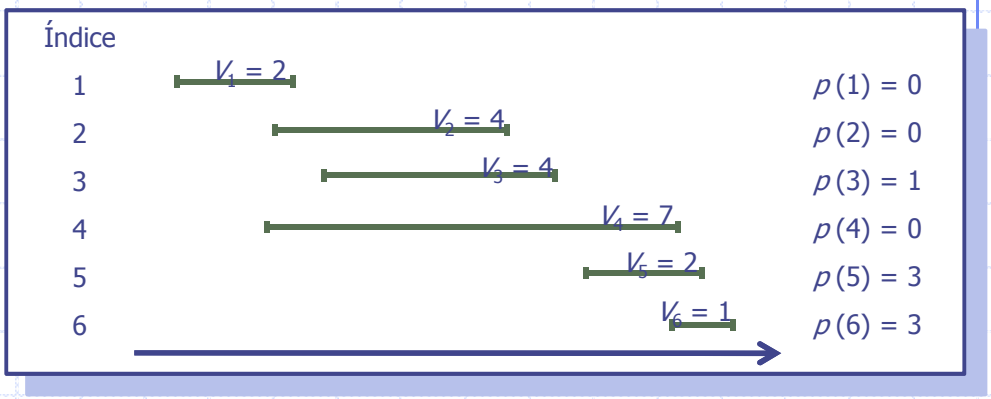

2
4
6
7+OPT(0)
OPT(3)
2+OPT(3)
OPT(4)
1+OPT(3)
OPT(5)
1
2
3
4
5
6
Indice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$
OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

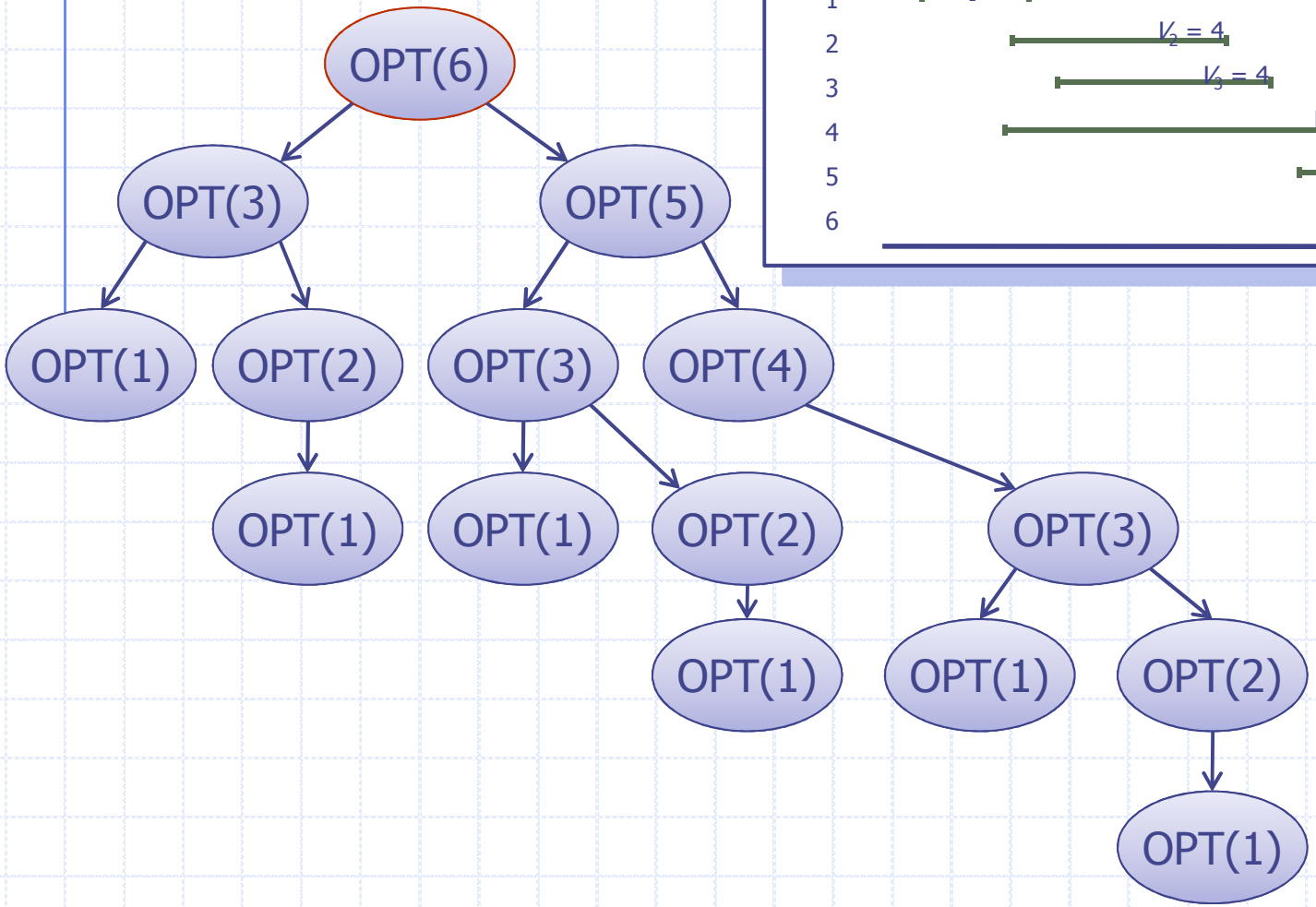

2
4
6
7
2+OPT(3)
OPT(4)
1+OPT(3)
OPT(5)
1
2
3
4
5
6
Indice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$
OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

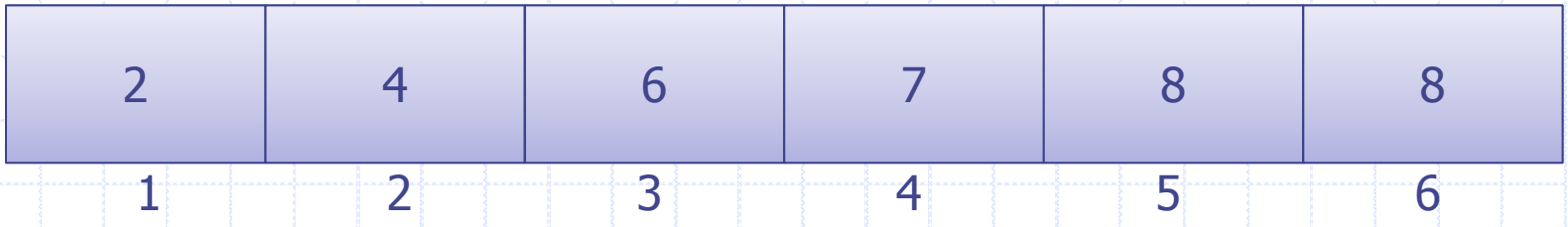

2
4
6
7
2+OPT(3)
OPT(4)
1+OPT(3)
OPT(5)
1
2
3
4
5
6
Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$
OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

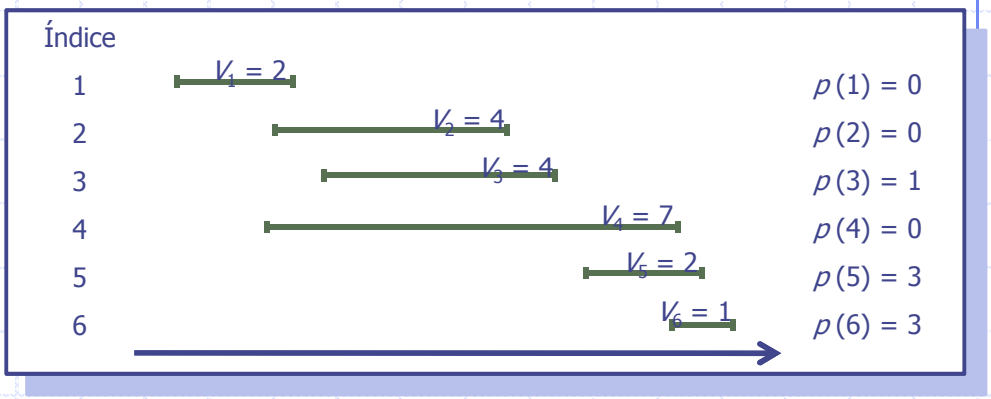

2
4
6
7
8
1+OPT(3)
OPT(5)
1
2
3
4
5
6
Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$
OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

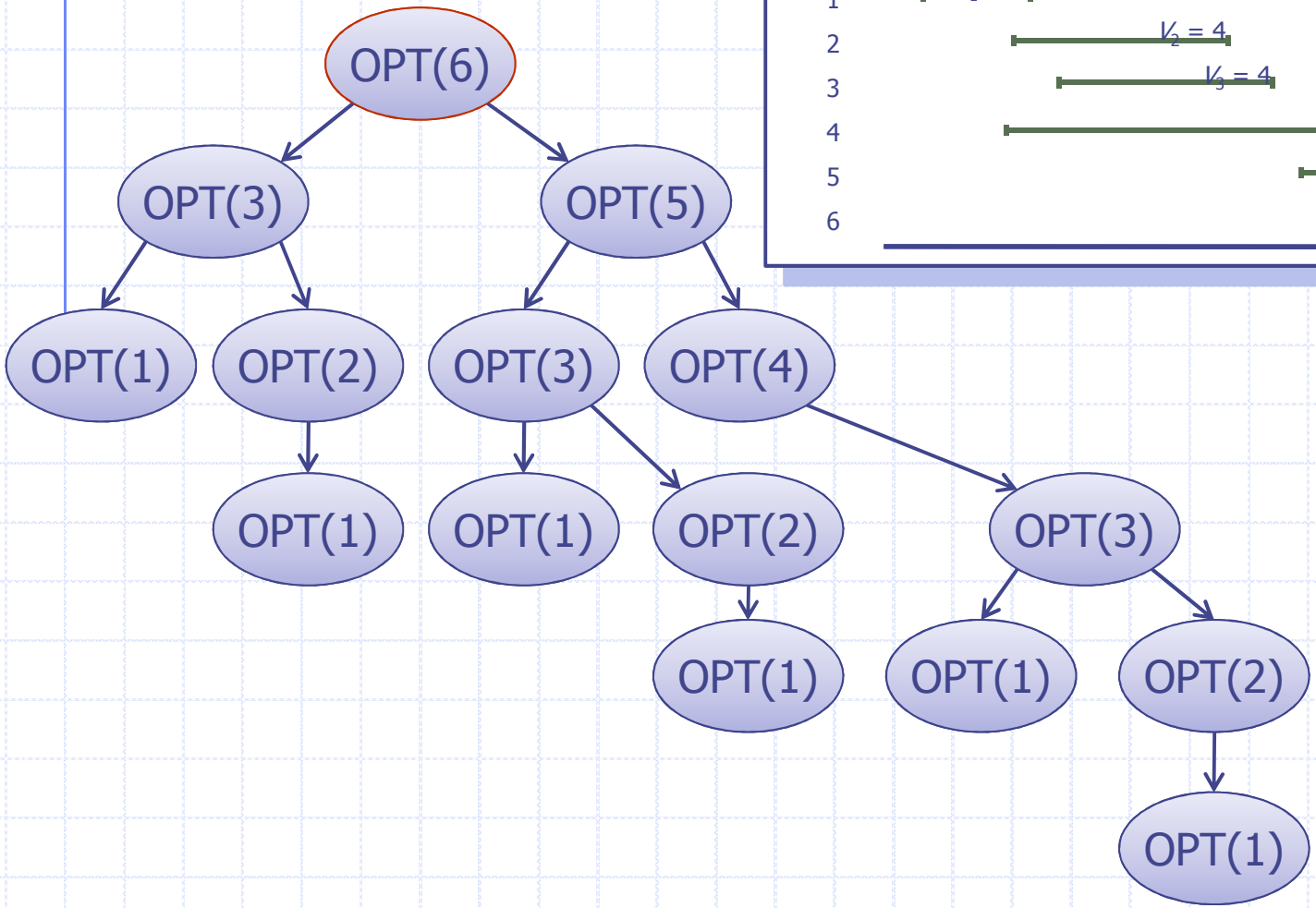

2
4
6
7
8
1+OPT(3)
OPT(5)
1
2
3
4
5
6


Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$


OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)


2
4
6
7
8
8
1
2
3
4
5
6


Índice
1
2
3
4
5
6
$V_1 = 2$
$V_2 = 4$
$V_3 = 4$
$V_4 = 7$
$V_5 = 2$
$V_6 = 1$
$p(1) = 0$
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$


OPT(6)
OPT(3)
OPT(5)
OPT(1)
OPT(2)
OPT(3)
OPT(4)
OPT(1)
OPT(1)
OPT(2)
OPT(3)
OPT(1)
OPT(1)
OPT(2)
OPT(1)

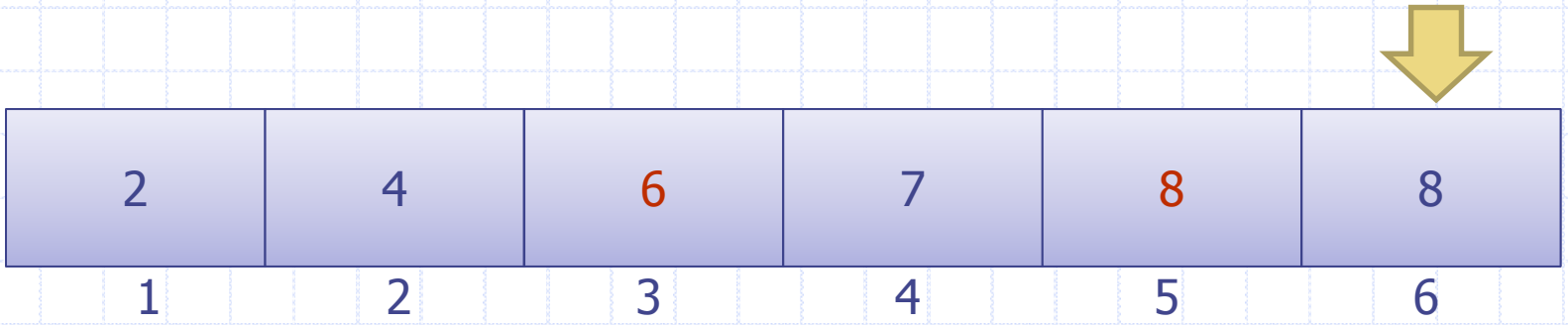
Exemplo 1 – WISP

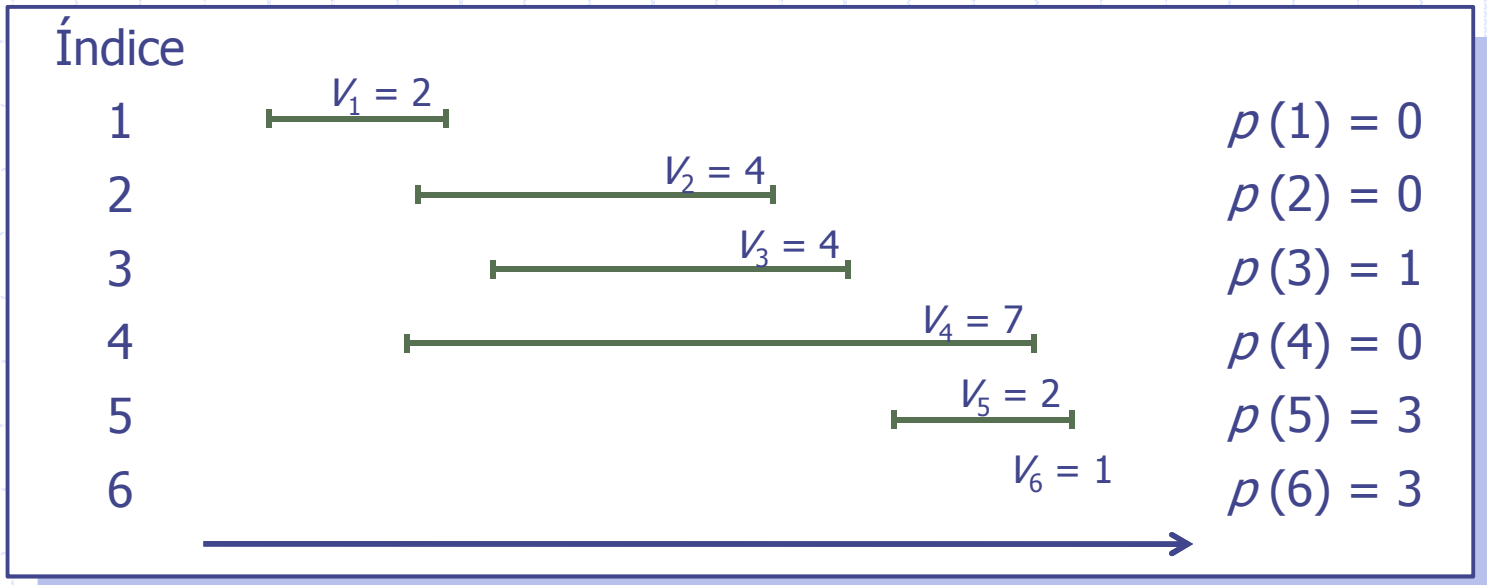
- A solução com memoização é eficiente e requer apenas $O(n)$ passos desde que os intervalos estejam ordenados.
- O vetor M auxilia não somente no cálculo do valor da solução, como também pode ser utilizado para encontrar os intervalos que compõem a solução.
- Um intervalo j pertence a solução se e somente se

$$v_j + \text{OPT}(p(j)) \geq \text{OPT}(j-1)$$

$v_j + \mathrm{OPT}(p(j)) \geq \mathrm{OPT}(j-1)$

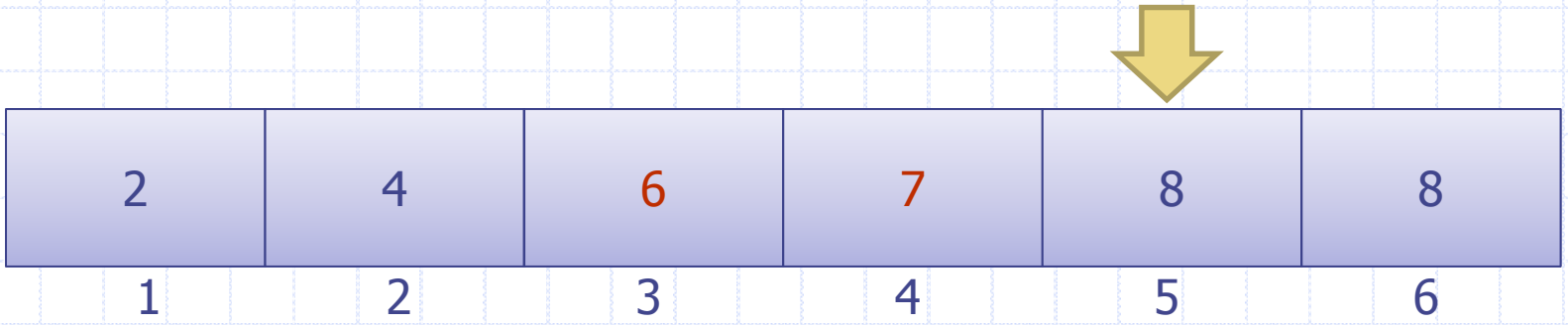
Exemplo 1 – WISP

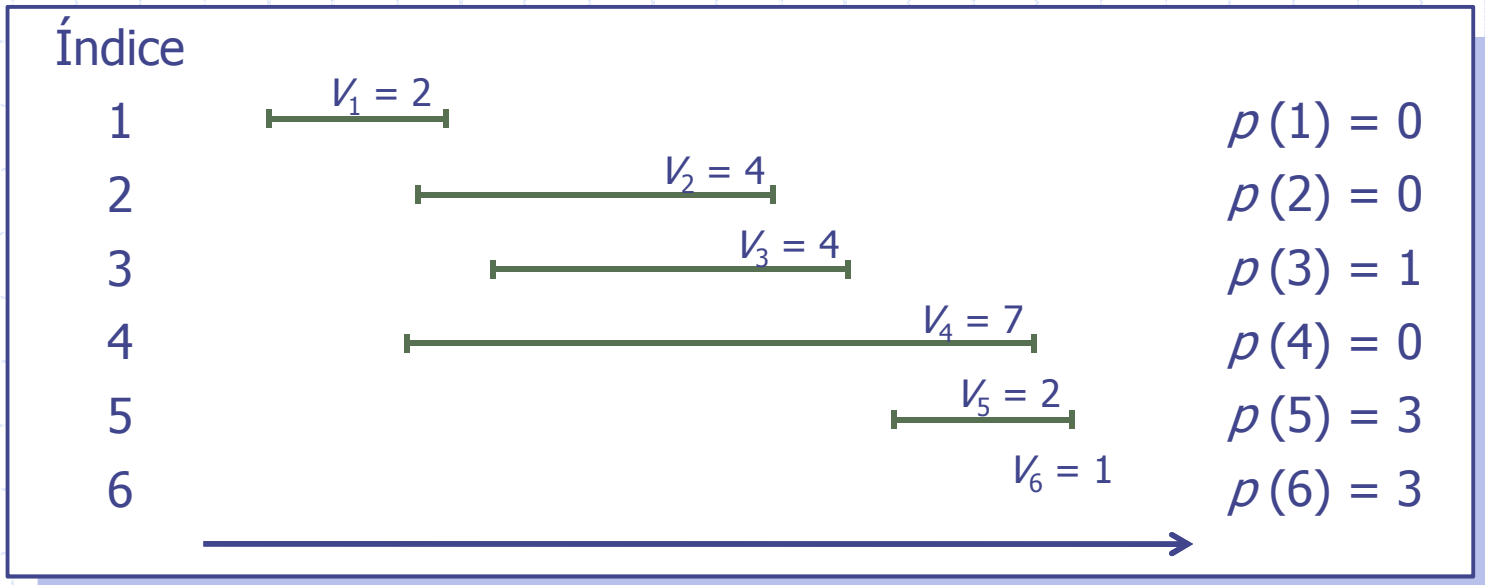




$v_j + \mathrm{OPT}(p(j)) \geq \mathrm{OPT}(j-1)$

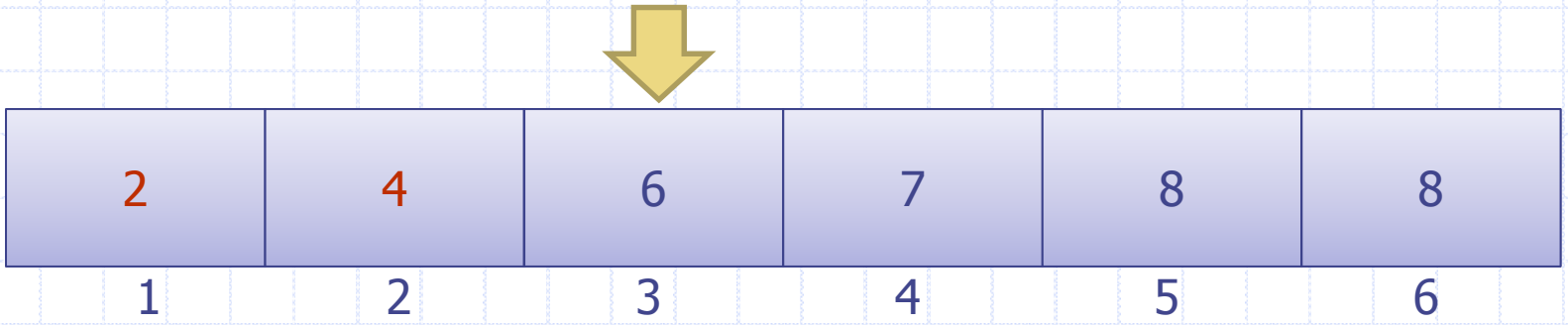
Exemplo 1 – WISP

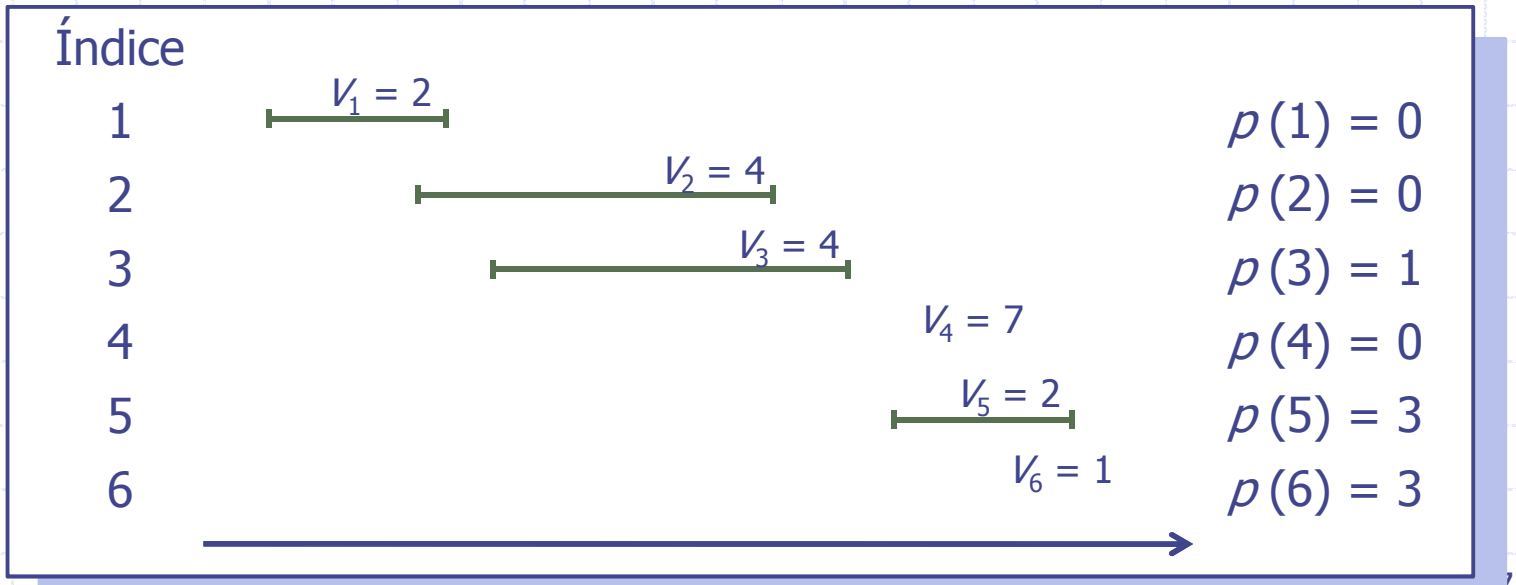




$v_j + \mathrm{OPT}(p(j)) \geq \mathrm{OPT}(j-1)$

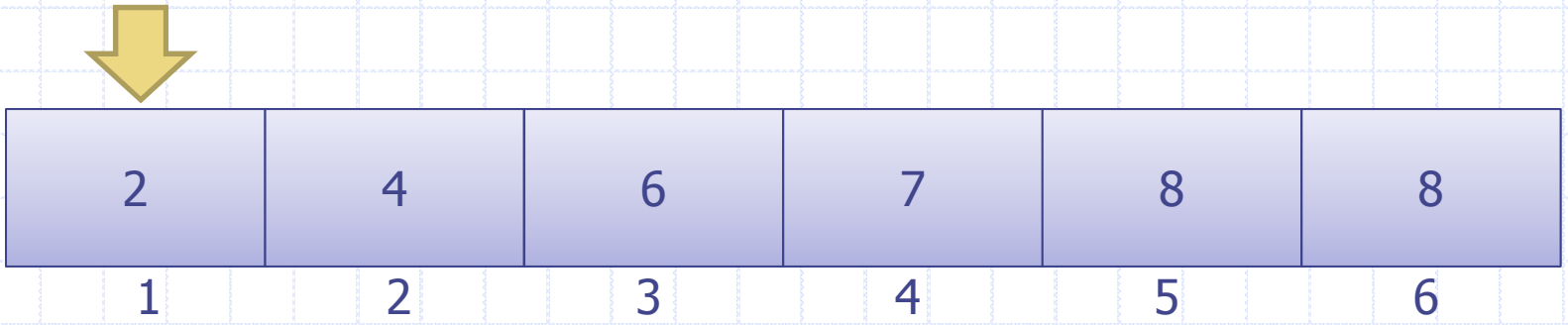
Exemplo 1 – WISP

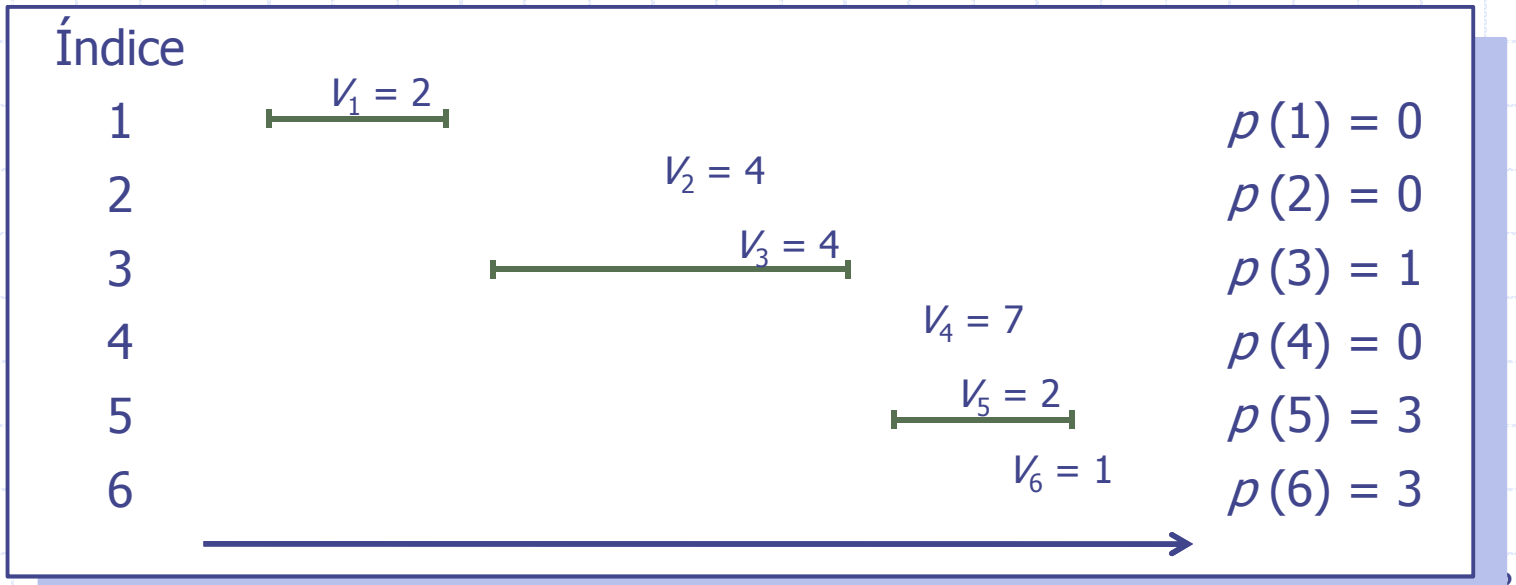




$v_j + \mathrm{OPT}(p(j)) \geq \mathrm{OPT}(j-1)$

Exemplo 1 – WISP

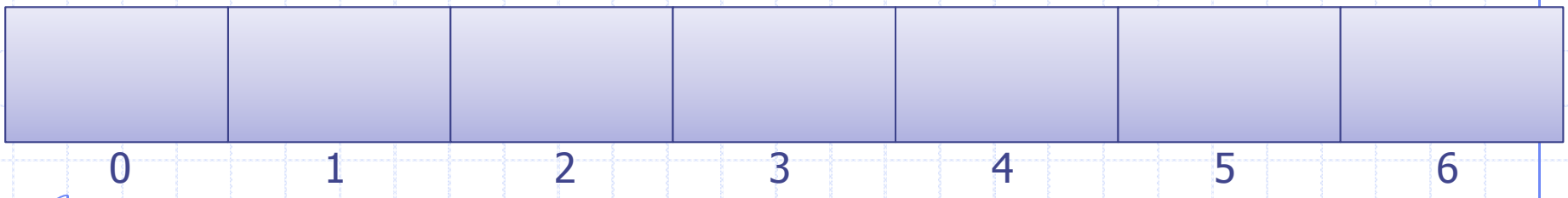




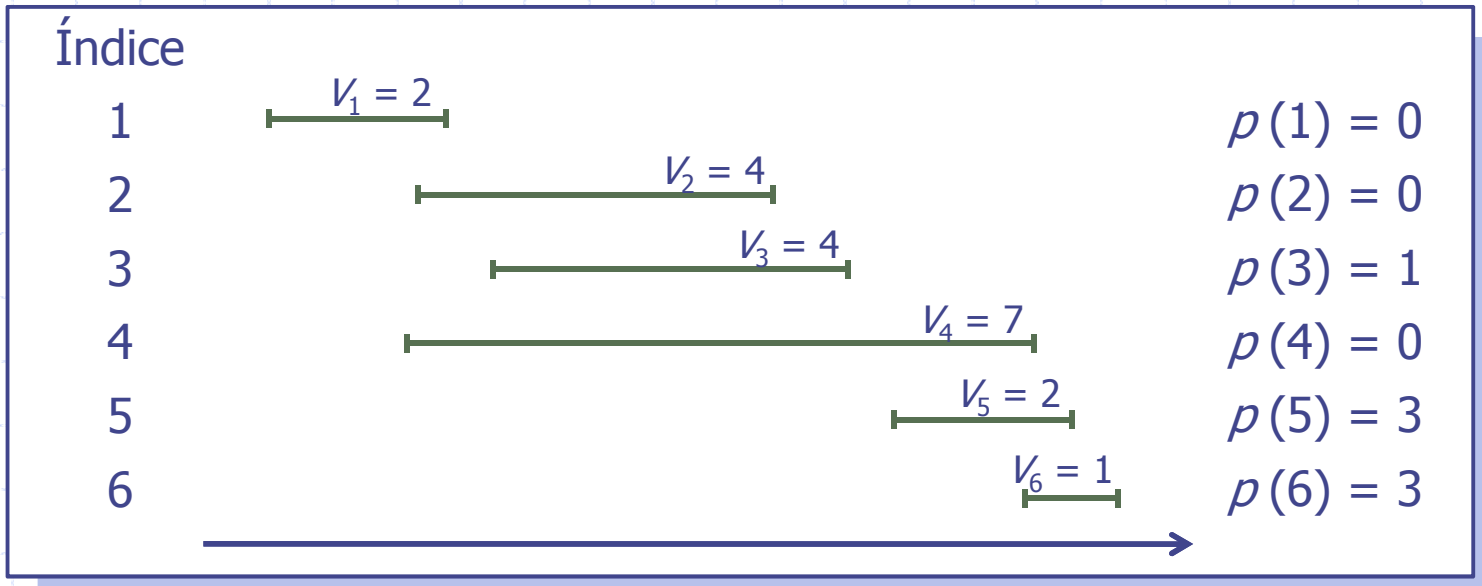
Exemplo 1 – WISP

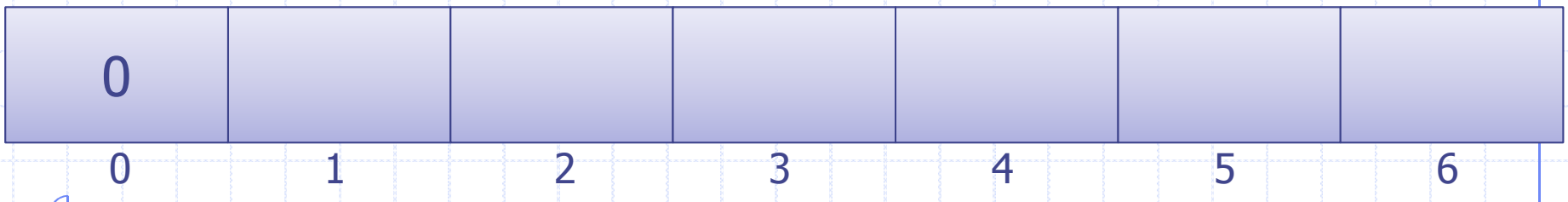
- Entretanto, Programação Dinâmica é caracterizada por:
  - Memoização
  - Interação sobre sub-problemas
- Pode-se substituir a recursão por uma iteração, essa é a forma mais simples e eficiente de PD

```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```

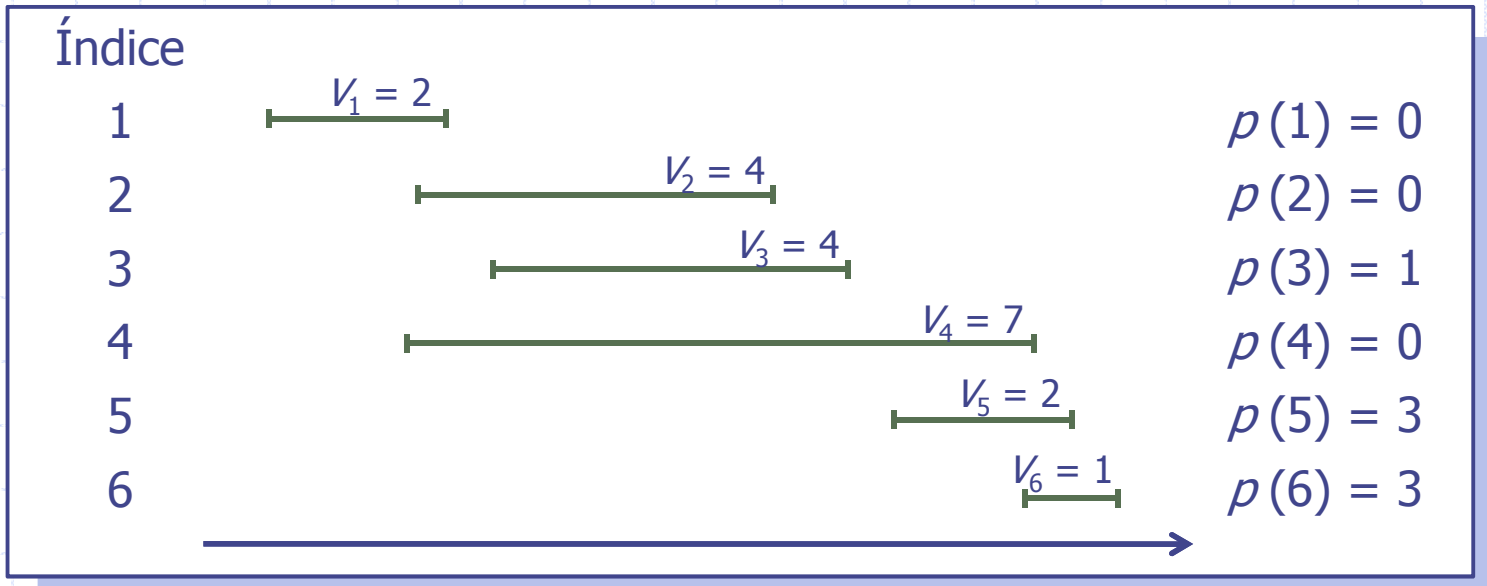


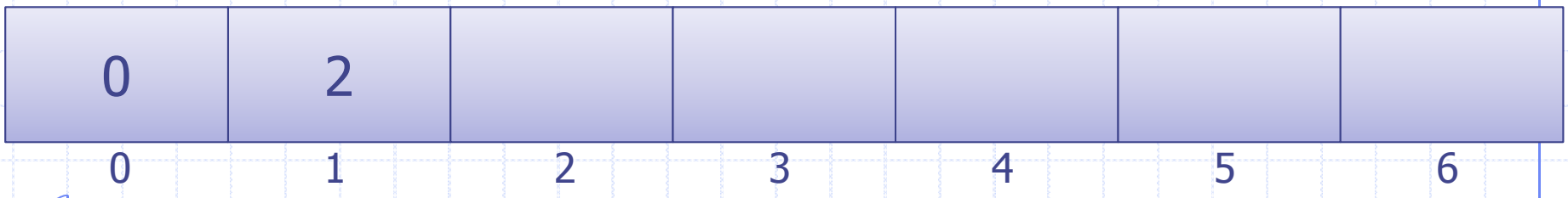

```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



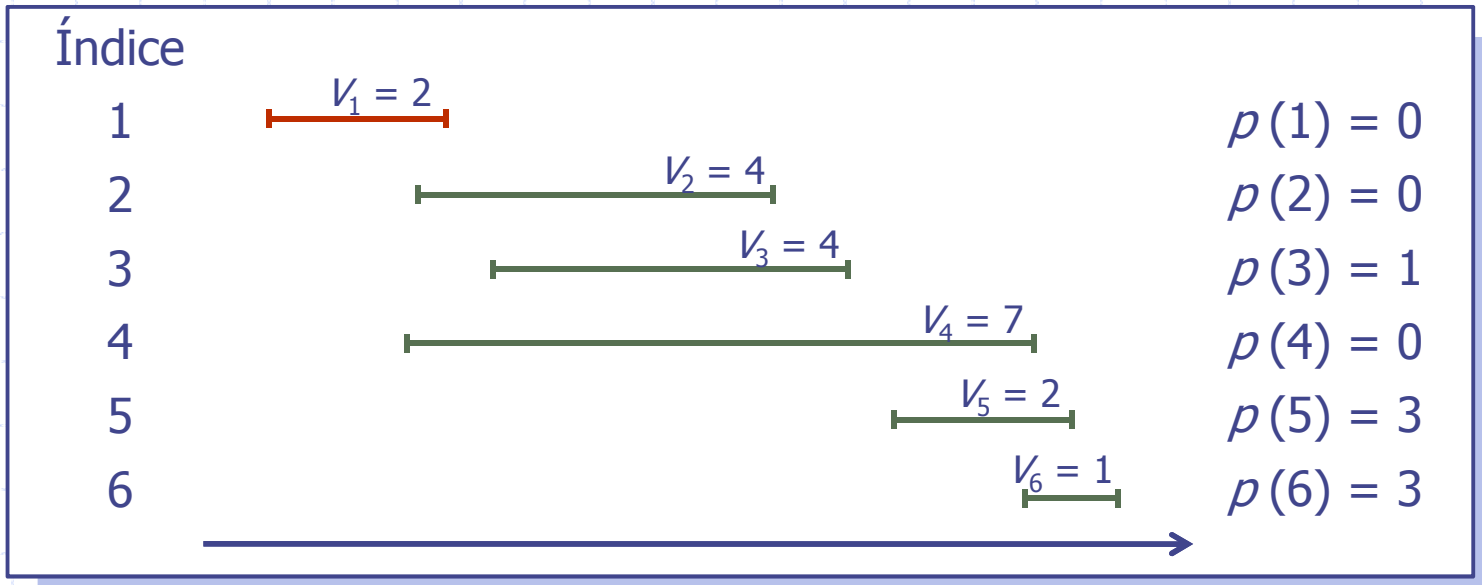


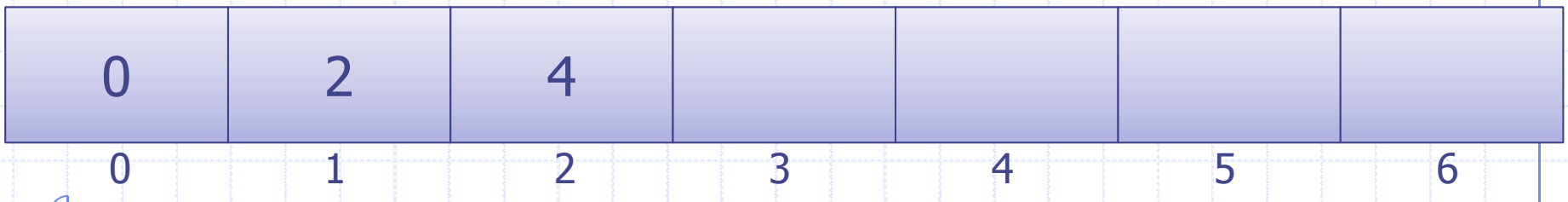

```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



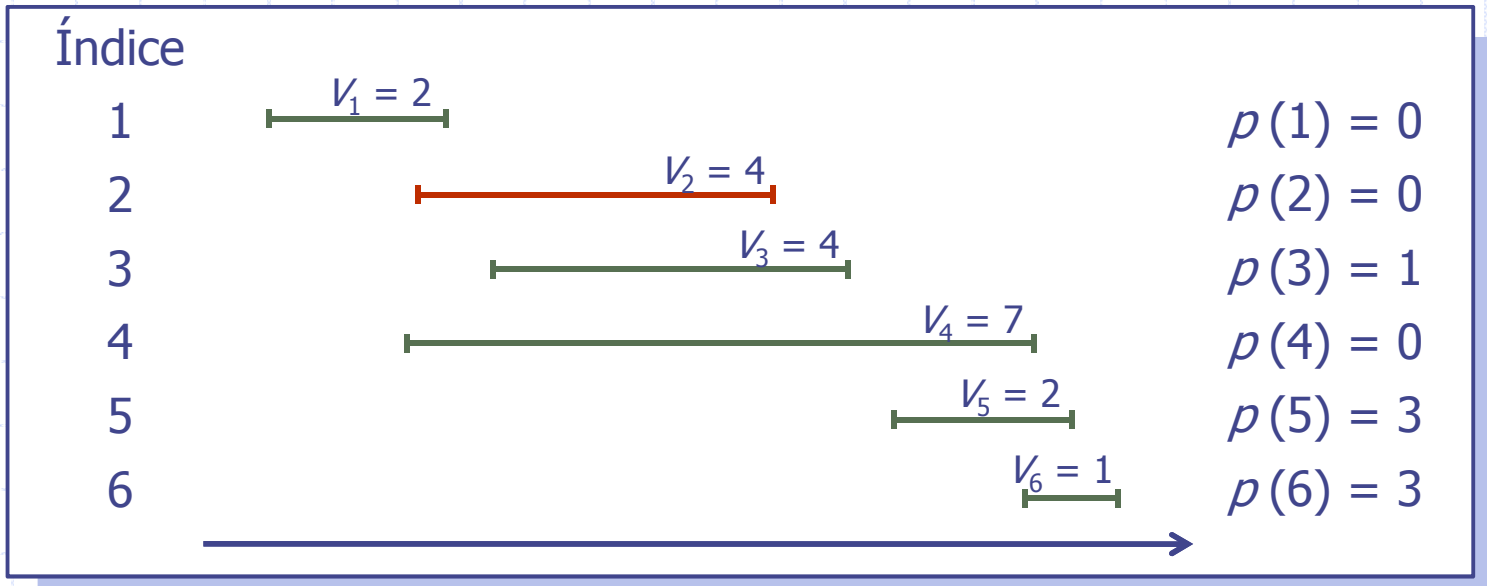



```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



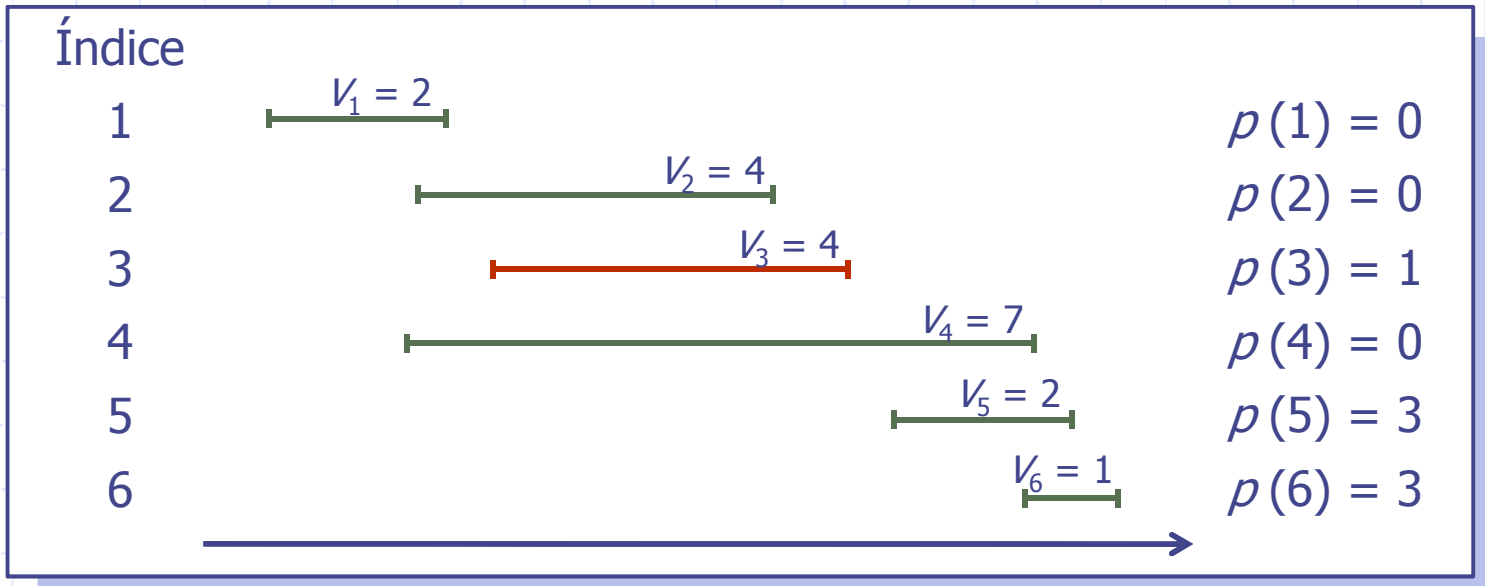



```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



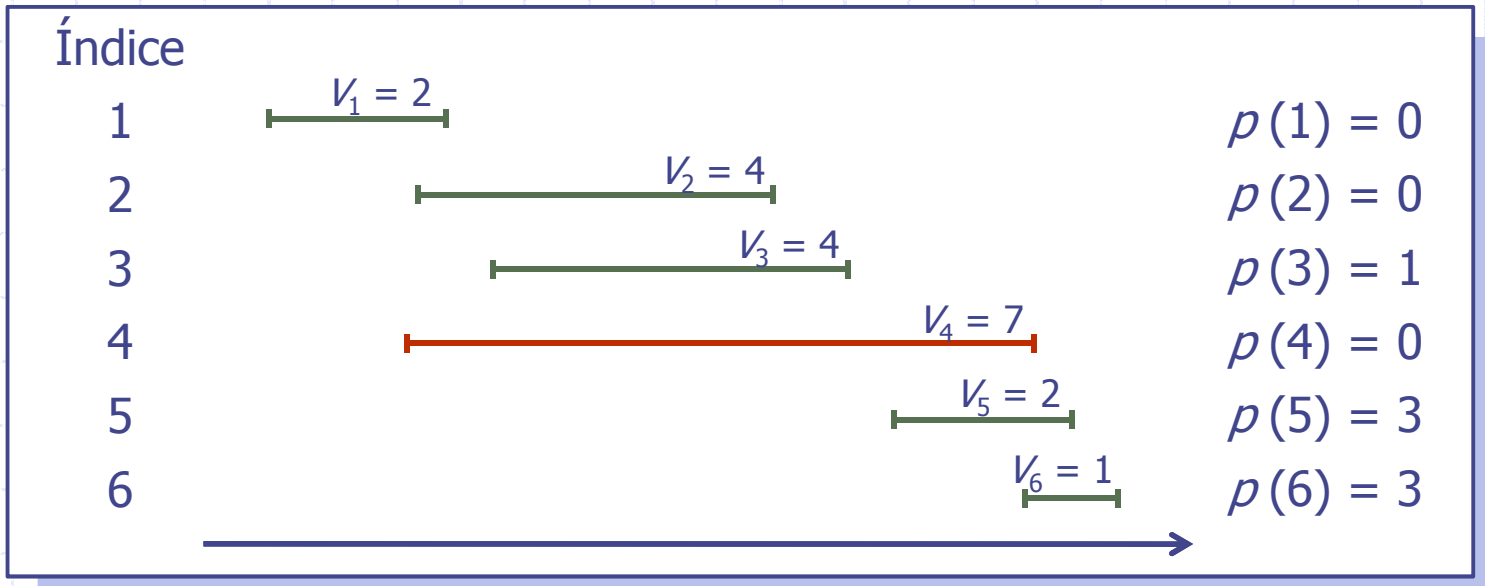
0	2	4	6			
0	1	2	3	4	5	6

```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



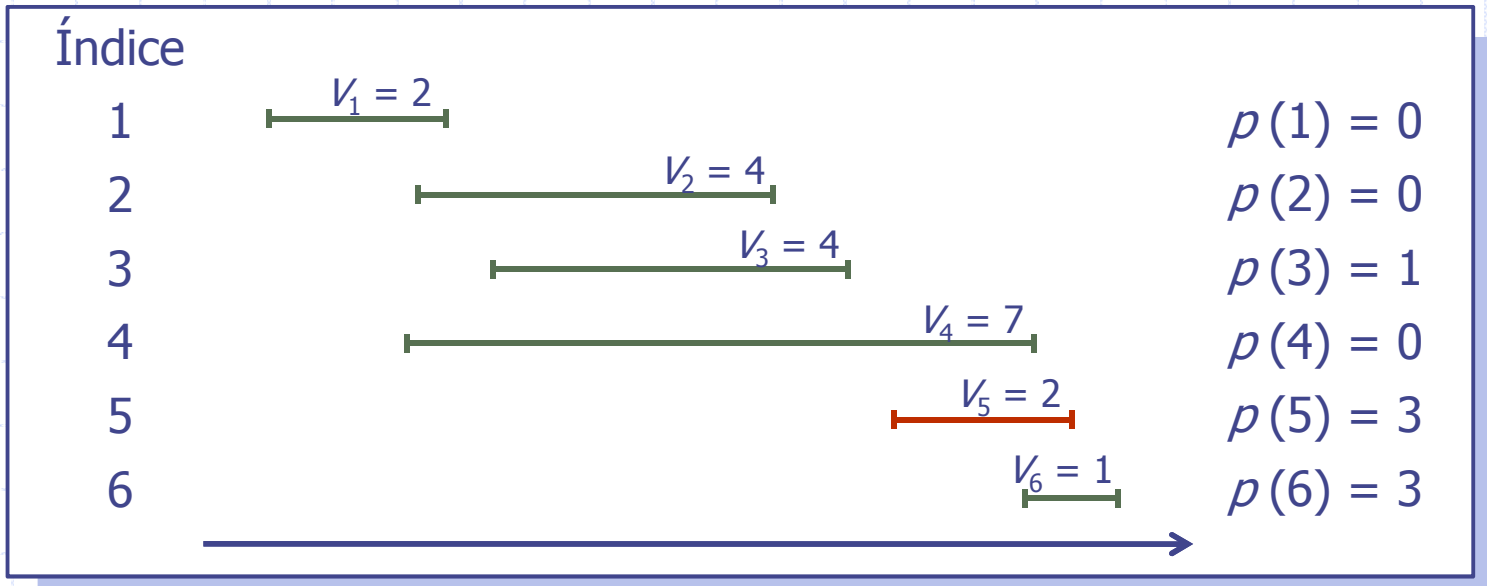
0	2	4	6	7		
0	1	2	3	4	5	6

```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



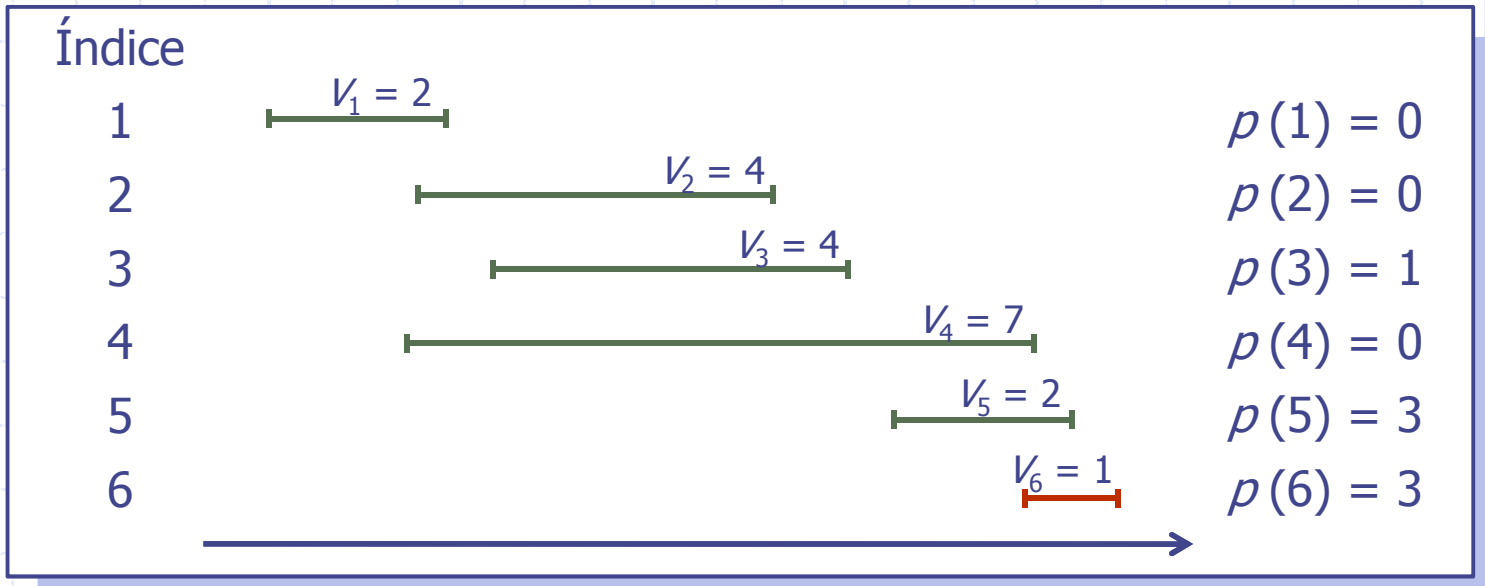
0	2	4	6	7	8	
0	1	2	3	4	5	6

```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



0	2	4	6	7	8	8
0	1	2	3	4	5	6

```
Iterative-Compute-Opt
    M[0] = 0
    for j = 1, 2, ..., n
        M[j] = max(v[j] + M[p(j)], M[j - 1])
end
```



Observações

- Tanto a solução recursiva com memoização quanto a solução iterativa são $O(n)$.
- De uma forma geral, o principal passo para resolução do problema é encontrar a relação de recorrência
- Dada essa relação, escolher pela solução recursiva com memoização ou a solução iterativa é basicamente uma escolha de abordagem *top-down* (recursiva) ou *bottom-up* (iterativa).
- A solução iterativa é preferida, por ser simples e não requerer o esforço computacional da recursão

Exemplo 2 - WSSP

- Um segundo problema que pode ser resolvido com PD é chamado de Soma de Subconjuntos com Pesos (Weighted Subset Sum Problem – WSSP).

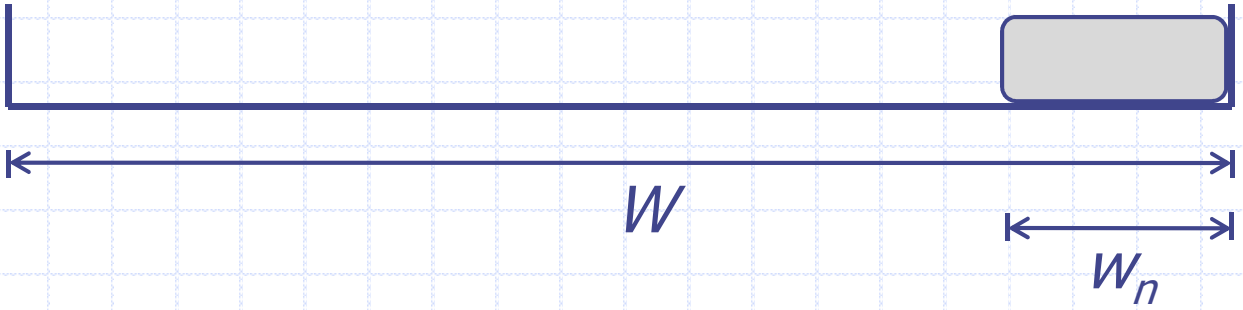
- É uma variação do problema da mochila. Dados:
  - Um conjunto de itens {1, …, n}, cada um com um peso w_i
  - E uma capacidade W
  - Deseja-se encontrar um subconjunto S, tal que $\Sigma_{i \in S} w_i$ é máxima, mas inferior a W.
- Existe alguma solução *greedy* para esse problema?

Exemplo 2 - WSSP

- Utilizando o mesmo raciocínio para o problema anterior:
  - Se $n \notin S$, então OPT(n) = OPT(n - 1)
  - Se $n \in$ S, então ...

Exemplo 2 - WSSP

- Utilizando o mesmo raciocínio para o problema anterior:
  - Se $n \notin S$, então OPT(n) = OPT(n - 1)
  - Se $n \in$ S, então ...
- O problema aqui é que aceitar a requisição n significa ter menos espaço disponível.



Exemplo 2 - WSSP

- Portanto:
  - Se $n \notin S$, então OPT(n) = OPT($n-1$, W)
  - Se $n \in$ S, então OPT(n) = OPT($n-1$, W - w_n)
- Uma observação importante é que um determinado item pode ser muito grande, ou seja $W < w_n$, nesse caso o item deve ser ignorado.
- Tem-se a seguinte recorrência:

Se $w < w_i$ então OPT(i, w) = OPT(i - 1, w), senão
OPT(i, w) = max(OPT($i-1$, w), w_i + OPT($i-1$, $w-w_i$))

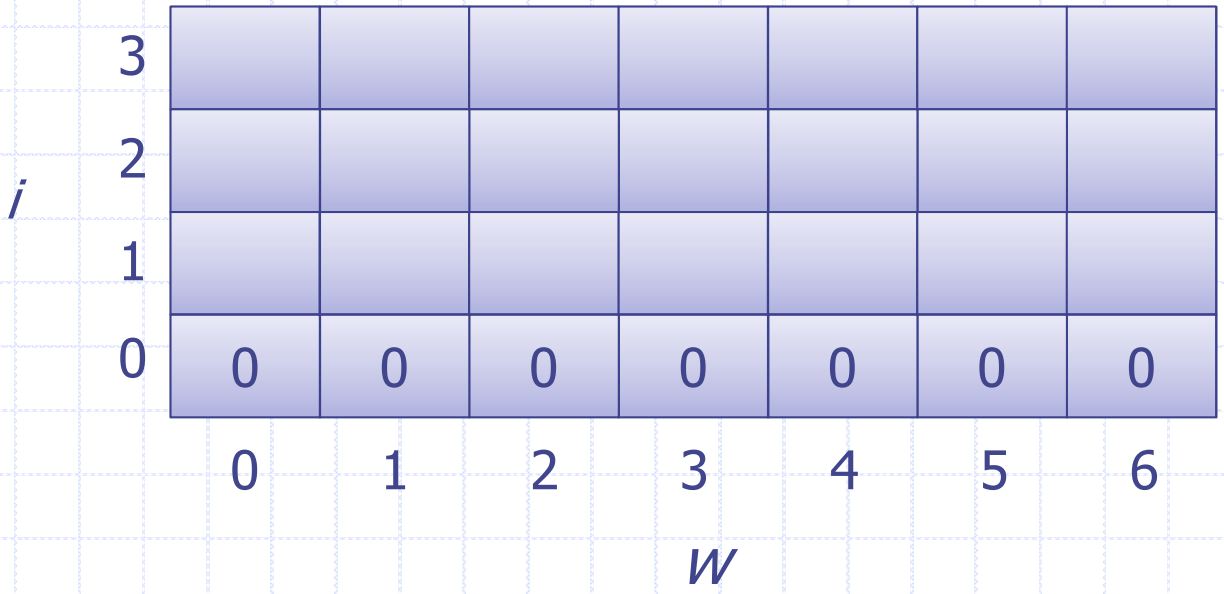
Exemplo 2 - WSSP

```
Subset-sum(n, W)
    Array M[0..n, 0..W]
    Initialize M[0, w] = 0 for each w = 0, 1, ..., W
    for i = 1, 2, ..., n
        for w = 0, ..., W
            if W < w[i] then
                M[i, W] = M[i - 1, W]
            else
                M[i, W] = max(M[i - 1, W], w[i] + M[i - 1, W - w[i]])
    return M[n, W]
end
```

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.



Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i \ w	0	1	2	3	4	5	6
3							
2							
1	0						
0	0	0	0	0	0	0	0

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i \ w	0	1	2	3	4	5	6
3							
2							
1	0	0					
0	0	0	0	0	0	0	0

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i	0	1	2	3	4	5	6
3							
2							
1	0	0	2				
0	0	0	0	0	0	0	0

w

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i	0	1	2	3	4	5	6
3							
2							
1	0	0	2	2			
0	0	0	0	0	0	0	0

w

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3							
2							
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3							
2	0	0	2				
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3							
2	0	0	2	2			
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3							
2	0	0	2	2	4		
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3							
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3	0	0	2	3			
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3	0	0	2	3	4		
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Ex

```
for i = 1, 2, ..., n
    for w = 0, ..., W
        if w < w[i] then
            M[i, w] = M[i - 1, w]
        else
            M[i, w] = max(M[i - 1, w], w[i] + M[i - 1, w - w[i]])
```

- Suponha:
  - $W = 6$ e
  - $w_1 = 2$, $w_2 = 2$ e $w_3 = 3$.

i							
3	0	0	2	3	4	5	5
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6
				w			

Exemplo 2 - WSSP

- O algoritmo calcula corretamente a resposta para o problema WSSP em tempo $O(nW)$
  - O algoritmo é chamado de pseudo-polinomial
- Dada a tabela M, pode-se encontrar o conjunto S em tempo $O(n)$

Exercícios

- O problema da mochila (*knapsack problem*) é similar ao WSSP, entretanto:
  - Além do peso w_i não-negativo
  - Existe um valor v_i
  - O objetivo é encontrar a solução com valor máximo de $\Sigma_{i \in S} v_i$, com restrição de $\Sigma_{i \in S} w_i < W$
- Formule uma equação de recorrência e um algoritmo para calcular a resposta para esse problema de forma eficiente. Calcule a complexidade da solução.