Programação dinâmica I

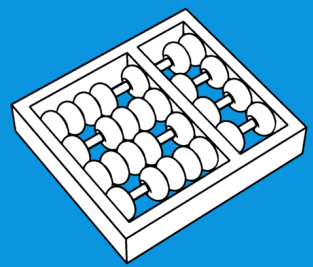
MC458 - Projeto e Análise de Algoritmos I

11/25

21

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

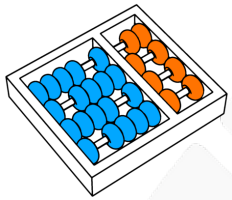
UNICAMP



“Aqueles que não lembram o passado estão condenados a repeti-lo”

George Santayana.

Introdução



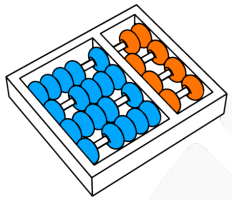
Richard Ernest Bellman (26/08/1920 – 19/03/1984)

Principais linhas de atuação:

- Matemática e teoria de controle.
- Equações diferenciais.
- Programação dinâmica e estocástica.
- Algoritmos em grafos.

Prêmios:

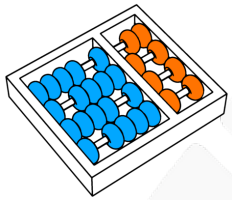
- Prêmio Dickson de Ciências (1970).
- Prêmio Norbert Wiener (1970).
- Prêmio Teoria John von Neumann (1976).
- Medalha de Honra IEEE (1979).
- Prêmio Richard E. Bellman (1984).



Problemas e subproblemas

Vamos estudar problemas com algumas propriedades:

- **Subestrutura ótima**:
  - Decompomos a instância em vários **subproblemas**.
  - A solução ótima é construída a partir de soluções ótimas de subproblemas.
- **Sobreposição de subproblemas**:
  - Uma instância é quebrada em várias instâncias menores.
  - A recursão recalcula uma mesma instância **várias vezes**.



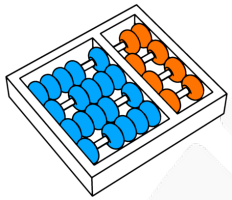
Programação dinâmica

Ideia:

- Evitar o recálculo desnecessário de subproblemas.
- Guardar as soluções de subproblemas em uma **tabela**.
- Cada entrada é uma instância distinta do subproblema.

Premissas da programação dinâmica:

- O número entradas da tabela é pequeno.
- Sabemos computar cada uma eficientemente.



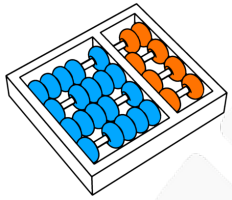
Problemas de otimização

Consideramos tipicamente **problemas de otimização**:

- Cada instância tem um conjunto de **soluções viáveis**.
- Cada uma delas tem um valor dado pela **função-objetivo**.
- Queremos alguma cujo valor é **mínimo ou máximo**.

Uma solução viável com o melhor valor é chamada de **ÓTIMA**.

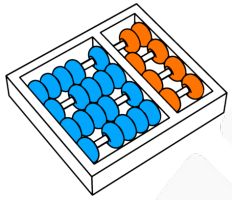
Torneio



Problema do torneio

- Considere um torneio entre duas pessoas A e B.
- Nesse jogo, uma partida nunca termina empatada.
- Vence o torneio o jogador que vencer k partidas primeiro.
- Suponha que A vence uma partida com probabilidade 0.6.
  - Outros valores também poderiam ser usados.

Problema: Qual é a probabilidade de que A vença o torneio?

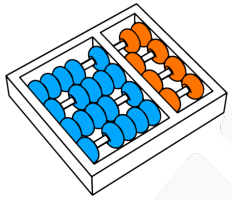


Subproblema recursivo

Seja $P(k|i,j)$ a probabilidade de A ganhar o torneio dado que:

- O número de vitórias do jogador A é i.
- O número de vitórias do jogador B é j.

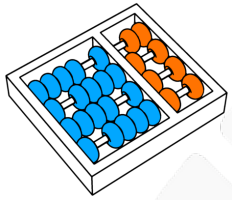
Vamos construir um algoritmo por **indução reversa** em i e em j.



Algoritmo por indução reversa

- Base:
  - Se $i = k$, então $P(k|k,j) = 1$ para todo j pois A é o vencedor.
  - Se $j = k$, então $P(k|i,k) = 0$ para todo i pois B é o vencedor.
- Hipótese de indução:
  - Sabemos calcular $P(k|i',j')$ para $i' > i$ e $j' > j$.
- Passo:
  - Considere os casos em que A ou B vence a próxima partida.
  - Por hipótese, já sabemos calcular $P(k|i+1,j)$ e $P(k|i,j+1)$.
  - Então podemos calcular a probabilidade condicional:

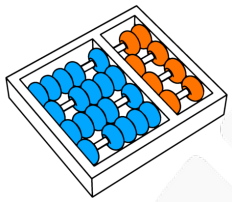
$$P(k|i,j) = 0.6 \cdot P(k|i+1,j) + 0.4 \cdot P(k|i,j+1)$$



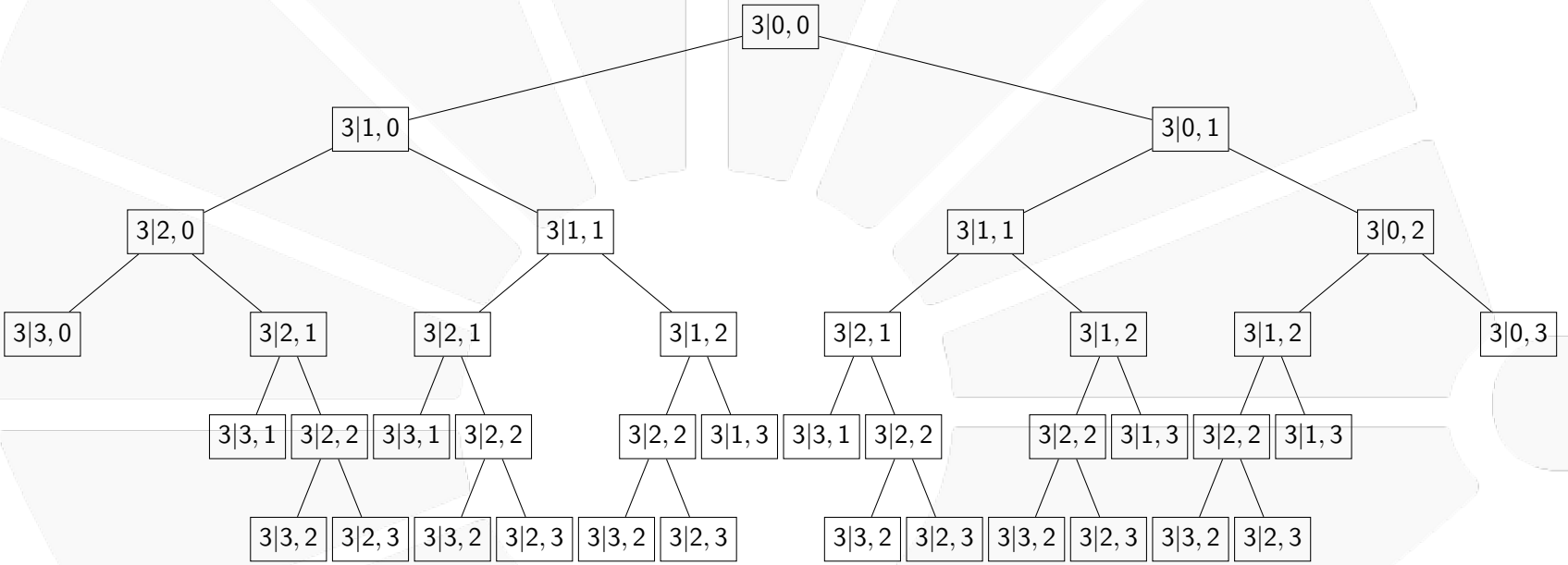
Algoritmo recursivo

Algoritmo: $\textsc{Recursivo}(k, i, j)$

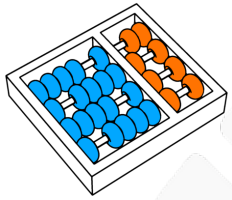
```
1 se i = k
2     devolva 1
3 se j = k
4     devolva 0
5 devolva
    0.6 · Recursivo(k, i + 1, j) + 0.4 · Recursivo(k, i, j + 1)
```



Esboço da árvore de recursão para $k = 3$

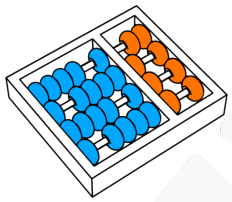


- A menor profundidade de uma folha é k.
- Esta árvore tem pelo menos $\sum_{p=0}^{k} 2^p = 2^{k+1} - 1$ nós.
- O algoritmo recursivo é $\Omega(2^k)$.

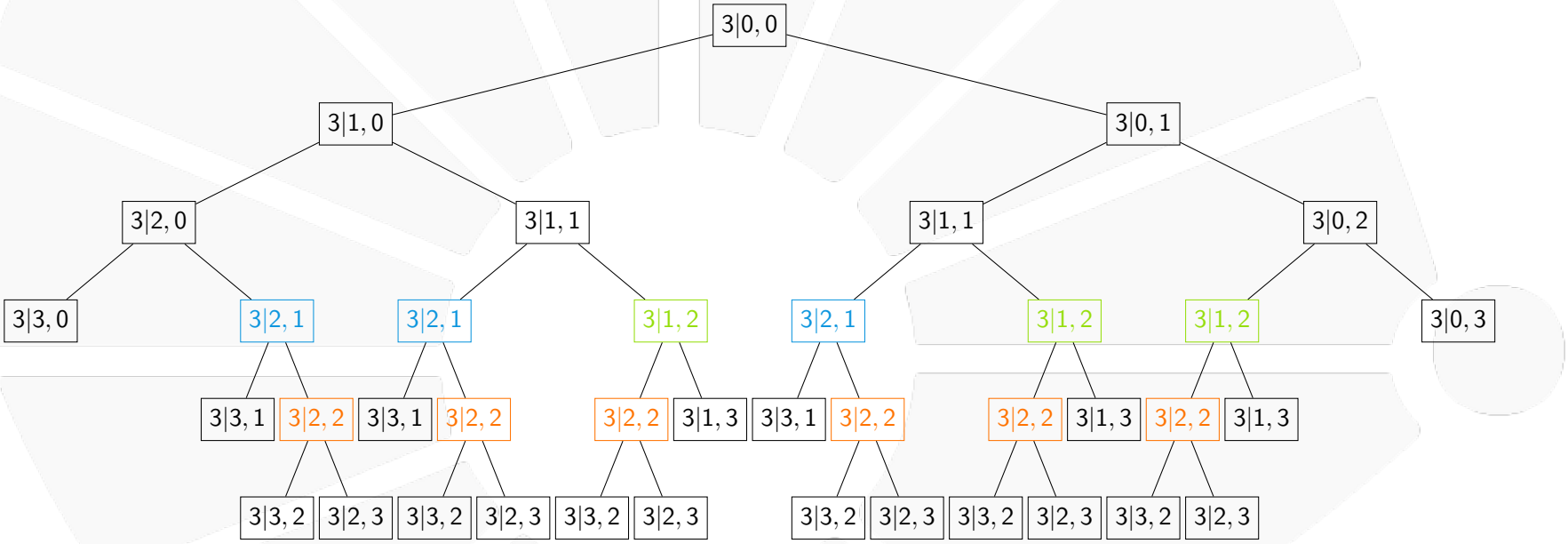


Número de subproblemas

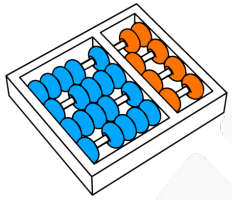
- Temos um algoritmo exponencial, que tem pouca utilidade.
- Mas os parâmetros de entrada i e j variam apenas 0 até k.
- O número de possibilidades é $(k+1)(k+1)$.
- Ou seja, há apenas $O(k^2)$ instâncias do subproblema.



Número de subproblemas



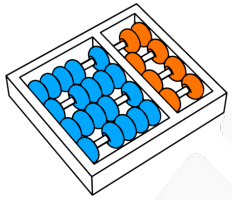
- Vários subproblemas aparecem repetidas vezes.
- Alguns deles estão com a mesma cor na árvore acima.



Duas estratégias

1. Top-down:
   - Resolvemos os subproblemas novos recursivamente.
   - Guardamos os resultados em uma estrutura de dados.
   - É chamada de programação dinâmica com memorização ou simplesmente de **memorização**.

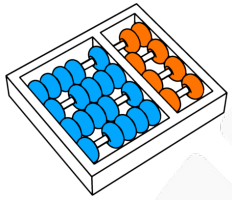
2. Bottom-up:
   - Criamos uma tabela para guardar os resultados.
   - Preenchemos as entradas para os casos básicos.
   - Preenchemos as demais, das menores para as maiores.
   - É chamada tipicamente de **programação dinâmica**.



Exemplo da estratégia bottom-up

	0	1	2	3
0				
1				
2				
3				

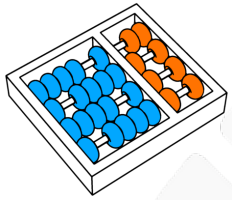
- Criamos uma matriz M de tamanho $(k+1) \times (k+1)$.
- Cada posição $M[i,j]$ guarda o valor de $P(k|i,j)$.
- O índice da linha representa i e da coluna j.
- Por exemplo, para $k = 3$ temos uma matriz 4×4.



Casos básicos

	0	1	2	3
0				0
1				0
2				0
3	1	1	1	-

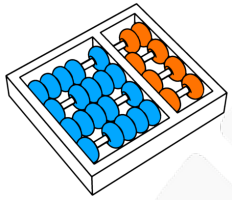
- Inicializamos as entradas para os casos básicos.
- Fazemos $M[k, j] = 1$ e $M[i, k] = 0$.



Passo

	0	1	2	3
0	0.68	0.48	0.22	0
1	0.82	0.65	0.36	0
2	0.94	0.84	0.6	0
3	1	1	1	-

- Uma entrada vale $M[i, j] = 0.6 \cdot M[i + 1, j] + 0.4 \cdot M[i, j + 1]$.
- Cada uma das entradas depende dos vizinhos na próxima linha $M[i + 1, j]$ e na próxima coluna $M[i, j + 1]$.
- Preenchemos a partir das últimas linha e coluna.
- A resposta está em $M[0, 0]$.



Algoritmo com programação dinâmica

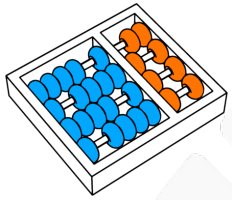
- Passamos a probabilidade de A ganhar como parâmetro g.

Algoritmo: $\mathrm{PD}(k, g)$

```
1 para i ← 0 até k − 1
2     M[k, i] ← 1
3     M[i, k] ← 0
4 para i ← k − 1 até 0
5     para j ← k − 1 até 0
6         M[i, j] ← g · M[i + 1, j] + (1 − g) · M[i, j + 1]
7 devolva M[0, 0]
```

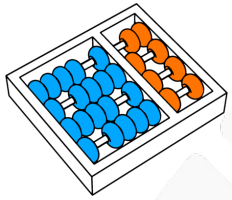
- A complexidade de tempo do algoritmo é $O(k^2)$.

Corte



Corte de barras de aço

- Uma empresa corta longas **barras de aço** em pedaços menores para revenda.
- Realizar um corte tem um custo insignificante.
- Uma barra tem tamanho inteiro n e podemos cortá-la em qualquer posição $1 \leq i \leq n-1$ ou revender a barra inteira.
- Um pedaço de tamanho $1 \leq i \leq n$ tem preço de revenda p_i.



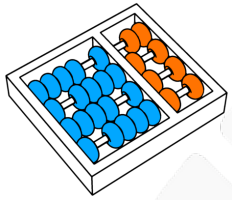
Exemplo de barra de aço

Exemplo onde $n = 4$:

tamanho	1	2	3	4
preço	1	5	8	9

Algumas formas de cortar:

- 4 itens de tamanho 1, **preço total 4**.
- 2 itens de tamanho 2, **preço total 10**.
- 2 itens de tamanho 1 e 1 item de tamanho 2, **preço total 7**.



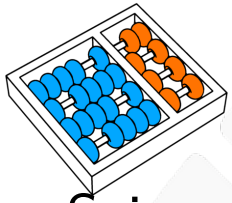
Problema do corte de barra de aço

Problema

Entrada: *Um inteiro n e um vetor com os preços de revenda p.*

Solução: *Quais cortes realizar na barra.*

Objetivo: ***MAXIMIZAR*** *preço total de revenda dos pedaços.*



Projeto por indução

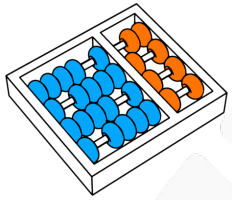
Seja r_n o preço total ótimo para uma barra de tamanho n.

- Base:
  - Se $n = 0$ então $r_0 = 0$.
  - Se $n = 1$ então $r_n = p_1$.
- Hipótese de indução:
  - Sabemos calcular r_k para $k < n$.
- Passo:
  - Seja i o tamanho do primeiro pedaço da solução ótima.
  - A melhor forma de cortar o resto da barra vale r_{n-i}, daí:

$$r_n = p_i + r_{n-i}$$

  - Como não conhecemos o tamanho do primeiro pedaço:

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



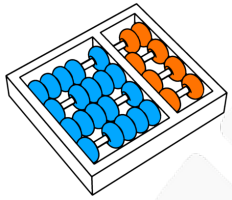
Algoritmo recursivo

Algoritmo: RECURSIVO(p, n)

```
1 se n = 0
2     devolva 0
3 s ← 0
4 para i ← 1 até n
5     t ← p_i + RECURSIVO(p, n − i)
6     se t > s
7         s ← t
8 devolva s
```

A complexidade de tempo do algoritmo é dada por:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 0 \\ \sum_{j=0}^{n-1} T(j) + \Theta(1) & \text{se } n > 0 \end{cases}$$



Tempo do algoritmo recursivo

Subtraindo $T(n-1)$ de $T(n)$ obtemos:

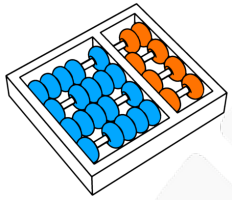
$$T(n) - T(n-1) = \left(\sum_{j=0}^{n-1} T(j) + c\right) - \left(\sum_{j=0}^{n-2} T(j) + c\right) = T(n-1)$$

Ou seja:

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

Resolvendo esta última recorrência, obtemos:

$$T(n) = \Theta(2^n)$$



Algoritmo de PD

- Criamos vetor $r[0 \ldots n]$ para usar programação dinâmica.
- Preenchemos do caso base para os casos maiores.

Algoritmo: $\mathrm{PD}(p, n, r)$

```
1  r[0] ← 0
2  para n' ← 1 até n
3      s ← 0
4      para i ← 1 até n'
5          t ← p_i + r[n' − i]
6          se t > s
7              s ← t
8      r[n'] ← s
```

- O algoritmo tem complexidade de tempo $\Theta(n^2)$.

Programação dinâmica I

MC458 - Projeto e Análise de Algoritmos I

11/25

21

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP