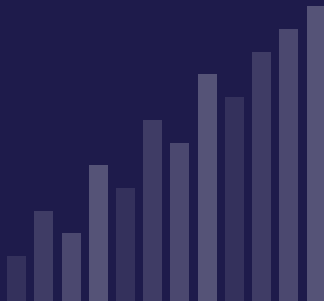

AULA 33 · BLOCO 4: GRAFOS

Programação dinâmica

Seções 15.1, 15.3 e 15.4

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Reconhecer subestrutura ótima e subproblemas sobrepostos
- ▶ Resolver o corte de hastes de cima para baixo (memoização) e de baixo para cima
- ▶ Calcular a maior subsequência comum (LCS) e reconstruí-la
- ▶ Medir 2^n chamadas contra $n(n+1)/2$



Teoria

Corte de hastes, o grafo de subproblemas, LCS



Prática em C

Corte de hastes e LCS em C



Experimento

Recursão ingênua contra memoização



Conexões

Bioinformática, diff, linguagem natural e finanças

Corte de hastes

Cortar uma haste de comprimento n em pedaços para **maximizar a receita** r_n :

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}), \quad r_0 = 0$$

O primeiro pedaço tem comprimento i ; o resto, $n - i$, é cortado da melhor forma:

subestrutura ótima. Para $n = 4$: cortar em $2 + 2$ dá $5 + 5 = 10$, melhor que vender inteira

(9).

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Figura 15.1 de Cormen et al. (2009) — preço p_i de uma haste de comprimento i .

Por que a recursão ingênua explode

- ▶ $\text{Cut-Rod}(p, n)$ chama a si mesma para $n - 1, n - 2, \dots, 0$.
- ▶ $T(n) = 1 + \sum_{j=0}^{n-1} T(j) = 2^n$ chamadas.
- ▶ São só $n + 1$ subproblemas diferentes: **sobrepostos**.
- ▶ Guardar cada resposta na primeira vez (**memoização**) ou calcular de baixo para cima: $\Theta(n^2)$.

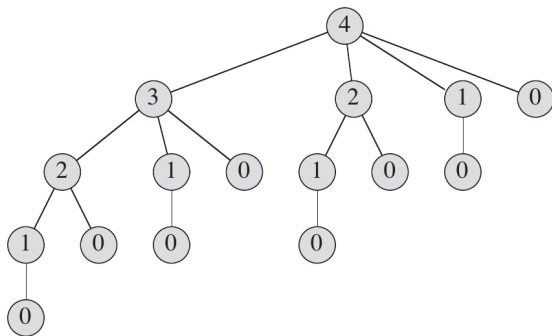


Figura 15.3 de Cormen et al. (2009) — árvore de recursão de $\text{Cut-Rod}(p, 4)$: os mesmos subproblemas aparecem muitas

O grafo de subproblemas

- ▶ Os subproblemas e suas dependências formam um **DAG**.
- ▶ De baixo para cima = resolver em ordem topológica reversa (aula 29).
- ▶ Custo = (número de vértices) $\times$ (trabalho por vértice) = soma dos graus de saída: $\Theta(n^2)$.
- ▶ **Quatro passos:** caracterizar a solução ótima, recorrência, calcular os valores, reconstruir a solução.

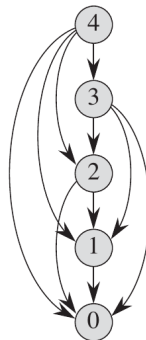


Figura 15.4 de Cormen et al. (2009) — o grafo de subproblemas para $n = 4$: uma aresta $x \rightarrow y$ indica

Maior subsequência comum (LCS)

$$c[i, j] = \begin{cases} 0 & i = 0 \text{ ou } j = 0 \\ c[i - 1, j - 1] + 1 & x_i = y_j \\ \max(c[i - 1, j], c[i, j - 1]) & x_i \neq y_j \end{cases}$$

- Tabela $(m + 1) \times (n + 1)$: $\Theta(mn)$.
- Seguindo as setas a partir de $c[m, n]$, reconstrói-se a LCS.

		j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A	
0	x_i		0	0	0	0	0	0	
1	A		0	0	0	0	1	1	
2	B		0	1	1	1	1	2	
3	C		0	1	1	2	2	2	
4	B		0	1	1	2	2	3	
5	D		0	1	2	2	2	3	
6	A		0	1	2	2	3	4	
7	B		0	1	2	2	3	4	

Figura 15.8 de Cormen et al. (2009) — tabela c e setas b para $X = ABCBDAB$ e $Y = BDCABA$: LCS = BCBA, de tamanho 4.



Prática em C

- ▶ Corte de hastes de baixo para cima com os preços da Figura 15.1
- ▶ LCS com tabela e reconstrução de trás para frente
- ▶ Testes: a LCS de ABCBDAB e BDCABA é BCBA

Corte de hastes e LCS em C (1/3)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  /* Preços da Figura 15.1: p[i] = preço de uma haste de comprimento i. */
5  int p[11] = {0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30};
6
7  int corte(int n) {                                // BOTTOM-UP-CUT-ROD:  $\Theta(n^2)$ 
8      int r[11] = {0};                               // r[j]: melhor receita para comprimento j
9      for (int j = 1; j <= n; j++) {
10         int q = -1;
11         for (int i = 1; i <= j; i++)                // primeiro pedaço de comprimento i
12             if (p[i] + r[j - i] > q) q = p[i] + r[j - i];
13         r[j] = q;
14     }
15     return r[n];
16 }
```

Corte de hastes e LCS em C (2/3)

```
17 0
18
19 /* LCS-LENGTH: c[i][j] = tamanho da LCS de X[1..i] e Y[1..j].  $\Theta(m n)$ . */
20 int c[101][101];
21
22 void lcs(const char *X, const char *Y, char *saida) {
23     int m = strlen(X), n = strlen(Y);
24     for (int i = 0; i <= m; i++) c[i][0] = 0;
25     for (int j = 0; j <= n; j++) c[0][j] = 0;
26     for (int i = 1; i <= m; i++)
27         for (int j = 1; j <= n; j++)
28             if (X[i - 1] == Y[j - 1]) c[i][j] = c[i - 1][j - 1] + 1; // estende
29             else c[i][j] = c[i - 1][j] >= c[i][j - 1] ? c[i - 1][j] : c[i][j - 1];
30     int k = c[m][n]; // PRINT-LCS, de trás para frente
31     saida[k] = '\0';
```

Corte de hastes e LCS em C (3/3)

```
31 00
32     for (int i = m, j = n; i > 0 && j > 0;)
33         if (X[i - 1] == Y[j - 1]) { saida[--k] = X[i - 1]; i--; j--; }
34         else if (c[i - 1][j] >= c[i][j - 1]) i--;
35         else j--;
36     }
37
38 int main(void) {
39     int n;
40     char X[101], Y[101], z[101];
41     if (scanf("%d %100s %100s", &n, X, Y) != 3 || n < 1 || n > 10) return 1;
42     printf("receita %d\n", corte(n));
43     lcs(X, Y, z);
44     printf("LCS de tamanho %d: %s\n", (int)strlen(z), z);
45     return 0;
46 }
```



Experimento

- ▶ Recursão ingênua: 2^n chamadas
- ▶ Memoizada: $n(n + 1)/2 + 1$ chamadas
- ▶ As duas versões têm de dar a mesma receita

🧪 EXPERIMENTO

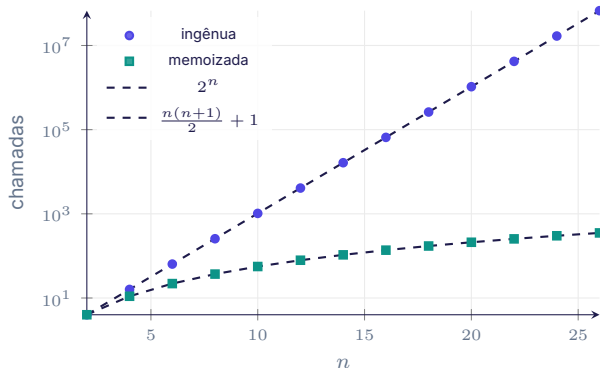
2^n **contra** $n(n + 1)/2$

Hipótese. A recursão ingênua faz $T(n) = 1 + \sum_{j < n} T(j) = 2^n$ chamadas: **exponencial**. A versão memoizada resolve cada subproblema uma vez; as chamadas que encontram a resposta pronta somam $n(n + 1)/2$, mais a chamada inicial: **quadrática**.

Método

1. Preços sintéticos para comprimentos 1 a 40
2. n de 2 a 26
3. Contador global de chamadas
4. Comparação das receitas

Chamadas recursivas



n	medido	razão
2,000	4,000	1,000
6,000	64,0	1,000
10,0	1.024	1,000
14,0	16.384	1,000
18,0	262.144	1,000
26,0	67.108.864	1,000

razão = medido $\div$ 2^n

✓ desvio máx. 0,0%

Para $n = 26$: 67 milhões de chamadas contra 352. Com $n = 50$, a ingênua levaria anos.

EXPERIMENTO

O que o experimento mostrou

As contagens são **exatamente** 2^n e $n(n + 1)/2 + 1$. Guardar as respostas dos subproblemas transforma um algoritmo exponencial em quadrático, com o mesmo resultado. A programação dinâmica troca **memória por tempo**: uma tabela de $n + 1$ posições economiza bilhões de chamadas.

➤ Reproduza: `gcc -O2 aula33_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Bioinformática
- ▶ Controle de versão
- ▶ Linguagem natural
- ▶ Finanças e IA

Onde isso aparece na computação

Bioinformática

Alinhar sequências de DNA e proteínas (Needleman–Wunsch, Smith–Waterman) é uma LCS com pesos: a base do BLAST e da comparação de genomas.

Controle de versão

diff e git diff calculam a menor sequência de edições entre dois arquivos, uma variação da LCS.

Linguagem natural

Corretores ortográficos (distância de edição), reconhecimento de voz (Viterbi) e análise sintática (CYK) são programação dinâmica.

Finanças e IA

Precificação de opções em árvores binomiais e aprendizado por reforço (equação de Bellman) resolvem subproblemas de baixo para cima.

Para casa

Exercícios

1. (15.1-2) Mostre que a estratégia gulosa “maior densidade p_i/i primeiro não é ótima no corte de hastes.
2. (15.1-3) Cada corte custa c . Modifique o algoritmo.
3. (15.4-1) Ache a LCS de $\langle 1, 0, 0, 1, 0, 1, 0, 1 \rangle$ e $\langle 0, 1, 0, 1, 1, 0, 1, 1, 0 \rangle$.
4. (15.4-5) Ache a maior subsequência crescente de n números em $O(n^2)$.
5. Implemente a distância de edição entre duas palavras e use-a para sugerir correções ortográficas.

Leitura

Cormen et al., seções 15.1, 15.3 e 15.4.

Código

josedihego.net/estruturas