Tutorial: Maior Subsequência Comum

Arthur Andrade D'Olival

1 Solução do Problema

O problema da maior subsequência comum (*Longest Common Subsequence*, LCS) pode ser resolvido por meio de *programação dinâmica*. A ideia central é decompor o problema em subproblemas menores, de forma que a solução ótima final seja construída a partir das soluções ótimas desses subproblemas.

1.1 Definição do Subproblema

Sejam duas strings s_1 e s_2, de tamanhos n e m, respectivamente. Definimos:

$$dp[i][j] = \text{o comprimento da maior subsequência comum entre } s_1[1..i] \text{ e } s_2[1..j].$$

Isto é, $dp[i][j]$ representa a resposta do problema considerando apenas os primeiros i caracteres de s_1 e os primeiros j caracteres de s_2.

1.2 Função de Transição

Ao analisar um par de posições (i, j), temos duas possibilidades principais:

- Se os caracteres atuais são iguais, isto é, $s_1[i] = s_2[j]$, então podemos estender uma subsequência comum encontrada anteriormente:

$$dp[i][j] = dp[i-1][j-1] + 1.$$

- Caso contrário, não é possível usar ambos os caracteres ao mesmo tempo, então devemos escolher o melhor resultado entre ignorar um dos dois:

$$dp[i][j] = \max(dp[i-1][j],\, dp[i][j-1]).$$

Assim, a função de transição pode ser resumida como:

$$dp[i][j] = \begin{cases} dp[i-1][j-1] + 1, & \text{se } s_1[i] = s_2[j] \\ \max(dp[i-1][j],\, dp[i][j-1]), & \text{caso contrário} \end{cases}$$

1.3 Casos Base

Os casos base seguem a lógica natural do problema:

$$dp[0][j] = 0 \quad \forall j \geq 0$$
$$dp[i][0] = 0 \quad \forall i \geq 0$$

Isto representa que, se uma das strings tiver tamanho zero, nenhuma subsequência comum pode ser formada.

1.4 Recuperação da Subsequência Comum

Após o preenchimento da matriz dp, o valor $dp[n][m]$ fornece o comprimento da maior subsequência comum entre s_1 e s_2. No entanto, para obter também a subsequência em si, é necessário realizar um processo de *backtracking* sobre a matriz.

A ideia consiste em iniciar a partir da posição (n, m) da matriz e caminhar em direção à origem. Em cada passo:

- Se $s_1[i] = s_2[j]$, então este caractere faz parte da LCS. Nesse caso, retrocedemos para $(i-1, j-1)$.
- Caso contrário, avançamos para o vizinho que contém o maior valor entre $dp[i-1][j]$ e $dp[i][j-1]$, pois esse vizinho indica o caminho que manteve o comprimento máximo da subsequência.

Como a subsequência é reconstruída de trás para frente, os caracteres encontrados devem ser adicionados no início da string resultante.

O algoritmo para recuperar uma subsequência comum de comprimento máximo é o seguinte:

```
int i = n;      // ponteiro para s1
int j = m;      // ponteiro para s2
string lcs = "";  // subsequência comum sendo reconstruída

// Enquanto ainda houver caracteres a considerar
while (i > 0 && j > 0) {

    // Caso 1: os caracteres são iguais → fazem parte da LCS
    if (s1[i - 1] == s2[j - 1]) {
        lcs = s1[i - 1] + lcs;  // adiciona no início da string
        i--;
        j--;
    }
    // Caso 2: se o valor de cima é maior, movemos para cima
    else if (dp[i - 1][j] > dp[i][j - 1]) {
        i--;
    }
    // Caso 3: caso contrário, movemos para a esquerda
    else {
        j--;
    }
}
```

Ao final desse processo, a string `lcs` conterá uma maior subsequência comum entre as duas strings.