Análise do Problema da Subsequência Comum Mais Longa

Luiz Carlos B. Vieira[1], **Sérgio Caetano Júnio**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)

luizcarlosbv@discente.ufg.br, sergio_caetano@discente.ufg.br

Resumo. *Esse relatório tem por objetivo analisar o problema clássico da Subsequência Comum Mais Longa no que se refere à sua implementação por meio da Programação Dinâmica. Serão discutidas a subestrutura ótima e a superposição de subproblemas, características associadas ao problema inicial. Após implementado, um estudo sobre o algoritmo que retorna a maior subsequencia comum entre duas outras sequencias será efetivado em conjunto com a verificação de seu tempo de execução assintótico.*

1. Descrição do Problema

O problema da subsequência comum mais longa (LCS) se baseia em encontrar, dadas duas sequências predecessoras X e Y, uma subsequência de caracteres pertencentes tanto a X quanto a Y, seguindo algumas restrições. Seja X = ⟨x1, . . . , xm) e Z = ⟨z1, z2, . . . , zk⟩, Z é uma subsequência de X se existe uma sequência estritamente crescente ⟨i1, i2, . . . ik⟩ de índices de X tais que, para todo j = 1, 2, . . . , k, temos xij = zj. Por exemplo, Z = ⟨B,C,D,B⟩ é uma subsequência de X = ⟨A,B,C,B,D, A,B⟩ com sequência de índices correspondentes ⟨2, 3, 5, 7⟩. Dizemos que Z é uma subsequência comum de X e Y se Z é uma subsequência de X e de Y simultaneamente. A subsequência comum mais longa nada mais é que a subsequência de tamanho máximo comum entre X e Y.

2. Subestrutura Ótima

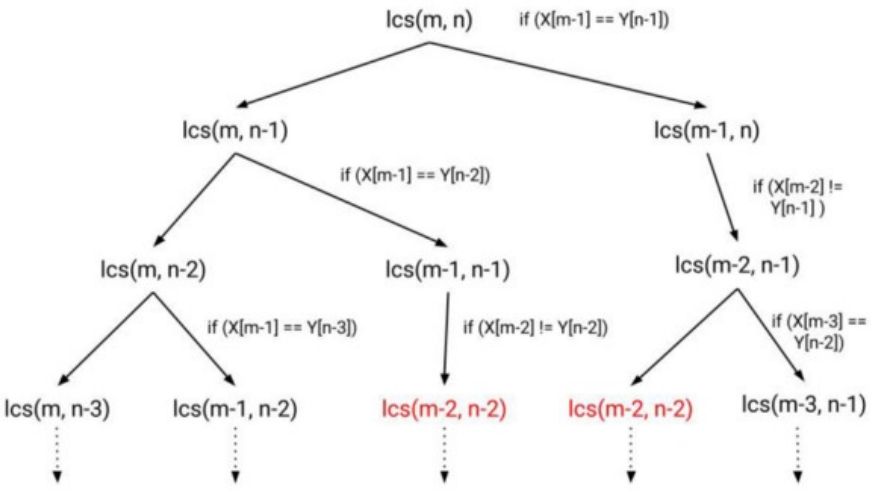

Figura 1. Árvore de recursão

Por definição, um problema apresenta uma subestrutura ótima quando, para se obter uma solução ótima, o seu universo de subproblemas apresenta soluções ótimas. O

problema da subsequencia comum mais longa pode ser resolvido ao se testar e comparar todas as subsequências possíveis de X e Y definidos na seção anterior. Essa resolução teria complexidade de tempo denotada por $O(n2^n)$, valores referentes à quantidade de subsequências de X e a sua comparação com os n valores de Y. Uma subestrutura ótima pode, porém, ser descrita para uma solução otimizada desse problema, até então inviável para entradas muito grandes.

Descrevendo o problema de forma resursiva (através da árvore de recursão representada acima), é possível observar que a maior subsequência comum entre duas sequências de tamanho m e n, representada por lcs(m,n), pode ser encontrada em termos de dois subproblemas. A solução ótima para o problema final busca encontrar a subsequência de tamanho máximo comum entre duas outras sequências. Como representa a recursão, essa subsequência de tamanho máximo pode ser descrita em termos de outras duas subsequências de tamanho máximo, representantes de soluções ótimas para aquelas etapas da recursão. O mesmo se repete até que os subproblemas triviais sejam resolvidos. Como todo problema tem solução ótima baseada em soluções ótimas de subproblemas, incluindo o problema final, existe uma subestrutura ótima para a o problema da LCS.

3. Solução

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

Figura 2. Algoritmo base

O código acima representa a parte fundamental do algoritmo não recursivo implementado como solução para o problema da maior subsequência comum. Além da subestrutura ótima, é possível observar, pela árvore de recursão, a superposição de problemas (ex.: repetição de lcs(m-2,n-2)). Com base nessas características, foi possível projetar um algoritmo que utiliza programação dinâmica para armazenar subproblemas já resolvidos e utilizá-los na solução de seus problemas pai, até resolver o problema inicial. As linhas do código apresentado percorrem uma matriz cujas posições guardam os tamanhos de subsequências comuns já encontradas. Segue a matriz modelo:

		Y1	Y2	Y3	Y5	Y6
	NULL	NULL	NULL	NULL	NULL	NULL
X1	NULL					
X2	NULL					
X3	NULL					
X4	NULL					

Figura 3. Matriz modelo

Sempre que um termo da sequência X é igual a um termo da sequência Y, um novo tamanho de subsequência é gravado, utilizando-se do tamanho da subsequência anterior armazenada na matriz (c[i,j] = c[i-j][j-1] + 1). Quando os termos de X e Y são diferentes, o maior tamanho entre as subsequências que já possuem aquele valor de X ou Y é reutilizado (c[i][j] = max(c[i,j-1],c[i-1,j])). É notável, entretanto, que essa rotina apenas armazena os tamanhos das maiores subsequências. Assim, uma adaptação responsável pela transformação da matriz modelo em uma matriz dinâmica de três dimensões possibilitou o armazenamento das subsequências já encontradas, ao invés do tamanho das mesmas.

```
30  //Função que cria uma matriz dinâmica tridimensional de caracteres que será utilizada, posteriormente, pela função LCS.
31  char *** criaMatrizStrings(char* sequencial, char* sequencia2, int tamanhoSequencial, int tamanhoSequencia2){
32
33      char *** matrizStrings;
34      char stringTemporaria[2]= "0";
35
36      matrizStrings = (char***)malloc((tamanhoSequencial + 2) * sizeof(char**));
37
38      for(int i = 0; i < (tamanhoSequencial + 2); i++){
39
40          matrizStrings[i] = (char**)malloc((tamanhoSequencia2 + 2) * sizeof(char*));
41      }
42
43      // TAMANHO DA SEQUÊNCIA 1 - > QUANTIDADE DE LINHAS -> i
44      // TAMANHO DA SEQUENCIA 2 - > QUANTIDADE DE COLUNAS -> j
45
46          for (int i = 0; i < (tamanhoSequencial + 2); i++){
47              for(int j = 0; j < (tamanhoSequencia2 + 2); j++){
48
49                  matrizStrings[i][j] = (char*)malloc(min(tamanhoSequencial,tamanhoSequencia2) * sizeof(char));
50
51                  if(i==0 && j>=2){
52
53                      stringTemporaria[0] = sequencia2[j-2];
54                      strncpy(matrizStrings[i][j], stringTemporaria,2);
55                  }
56
57                  if(j==0 && i>=2){
58
59                      stringTemporaria[0] = sequencial[i-2];
60                      strncpy(matrizStrings[i][j], stringTemporaria,2);
61                  }
62              }
63          }
64
65      return matrizStrings;
66  }
67
```

Figura 4. Definição da matriz dinâmica

A função responsável pela criação da matriz dinâmica recebe como parâmetros as sequências X e Y, onde será pesquisada a maior subsequência comum, e o tamanho dessas sequências. As linhas 36, 40 e 49 alocam memória para as posições da matriz de forma a possibilitar o armazenamento das subsequências de strings nas posições matrizStrings[i][j]. Além disso, a função inicializa a matriz, na primeira linha e primeira coluna, com as sequências informadas (linhas 53, 54, 59 e 60).

```
71  char* LCS (char*** matrizStrings, int tamanhoSequencia1, int tamanhoSequencia2) {
72
73      for(int i = 1; i < tamanhoSequencia1 + 2; i++) {
74          for(int j = 1; j < tamanhoSequencia2 + 2; j++) {
75              if(i == 1 || j == 1) {
76
77                  strcpy(matrizStrings[i][j], "");
78              }
79              else if(strcmp(matrizStrings[i][0],matrizStrings[0][j]) == 0){
80
81                  strcpy(matrizStrings[i][j],matrizStrings[i-1][j-1]);
82                  strcat(matrizStrings[i][j],matrizStrings[i][0]);
83              }
84              else if(strcmp(matrizStrings[i][0],matrizStrings[0][j]) != 0){
85
86                  if(strlen(matrizStrings[i][j-1])>=strlen(matrizStrings[i-1][j])){
87
88                      strcpy(matrizStrings[i][j],matrizStrings[i][j-1]);
89                  }
90                  else{
91
92                      strcpy(matrizStrings[i][j],matrizStrings[i-1][j]);
93                  }
94              }
95          }
96      }
97      return(matrizStrings[tamanhoSequencia1+1][tamanhoSequencia2 + 1]);
98  }
```

Figura 5. Função que calcula a maior subsequência comum

A função responsável pelo cálculo da LCS percorre a matriz criada executando um comportamento semelhante ao explicado para a figura 2. Entretanto, ao invés de armazenar os tamanhos das novas subsequências encontradas (linhas 81 e 82), ou o tamanho das maiores subsequências já existentes compostas por X ou Y (linhas 88 e 92), o algoritmo armazena ou atualiza as próprias subsequências, visto que a matriz matrizStrings[i][j] nas posições indicadas representa ponteiros. Dessa forma, a cada iteração, se uma nova subsequência é encontrada, esta é armazenada em termos de seus subproblemas de solução ótima. Caso contrário, a maior subsequência já existente é reaproveitada.

Ao final da execução da função char* LCS(), um ponteiro contendo a maior subsequência (última posição de linha e coluna da matriz) é retornado como resultado.

4. Tempo computacional assintótico

Em um caso geral, para uma solução de força bruta, onde são inseridas sequências de tamanho n, precisamos entender inicialmente quantas subsequências uma sequência de tamanho n possui. Se considerarmos as sequências como sendo conjuntos, pela teoria de conjuntos, o número total de subconjuntos de um conjunto de tamanho n é 2^n. Em seguida, será necessário testar quais subsequências são comuns entre as sequências inseridas, sendo executada essa busca em tempo linear, visando definir a subsequência comum de maior tamanho entre todas as sequências.

Ou seja, é possível definir que a complexidade assintótica de um algoritmo de força bruta para o problema de LCS é $O(n2^n)$.

4.1. Solução com Programação Dinâmica

Para um número constante de sequências, é possível representar o problema de forma recursiva separando-o em subproblemas compostos por dois parâmetros, m e n, que serão reduzidos em 1 a cada chamada da recursão, resultando em $(m + 1)(n + 1)$ possíveis

subproblemas a serem solucionados. Vale lembrar que esses subproblemas podem se repetir na solução recursiva.

Aplicando a programação dinâmica a essa abordagem [R.A. Wagner and M. J. Fischer 1974], isto é, armazenando a solução de cada subproblema em uma memória, é possível solucionar o problema em um tempo assintótico de $O(mn)$. Isso é observável pela implementação demonstrada na figura 5. A matriz é percorrida em cada uma de suas posições apenas uma vez, sendo, o tempo assintótico para percorre-la, $O(mn)$.

5. Teste computacional do Código

O código será testado a partir de alguns casos que contemplam a maior parte das possíveis entradas para o problema. Os casos são:

1. Duas sequências iguais.
2. Duas sequências completamente diferentes.
3. Duas sequências que possuem uma LCS que não é subcadeia dessas sequências.

5.1. Caso 01

```
Digite a primeira sequência de letras: abcdef

Digite a segunda sequência de letras: abcdef

Maior subsequência comum(LCS): abcdef

Tamanho da LCS: 6

Tempo de execução: 0,000000 ms.
Process returned 90 (0x5A)   execution time : 9.071 s
Press any key to continue.
```

Figura 6. Caso 01

5.2. Caso 02

```
Digite a primeira sequência de letras: abcdef

Digite a segunda sequência de letras: 123456

Maior subsequência comum(LCS):

Tamanho da LCS: 0

Tempo de execução: 0,000000 ms.
Process returned 84 (0x54)   execution time : 10.276 s
Press any key to continue.
```

Figura 7. Caso 02

5.3. Caso 03

```
Digite a primeira sequência de letras: 12asdqwe3

Digite a segunda sequência de letras: 1defs2ga3

Maior subsequência comum(LCS): 1de3

Tamanho da LCS: 4

Tempo de execução: 0,000000 ms.
Process returned 88 (0x58)   execution time : 16.328 s
Press any key to continue.
```

Figura 8. Caso 03

6. Caso Não-Trivial

O algoritmo que soluciona a LCS possui diversas aplicações práticas. Uma dessas que iremos exemplificar no nosso caso não-trivial é na área da biologia, para determinar qual é a subsequência comum mais longa entre duas cadeias de DNA.

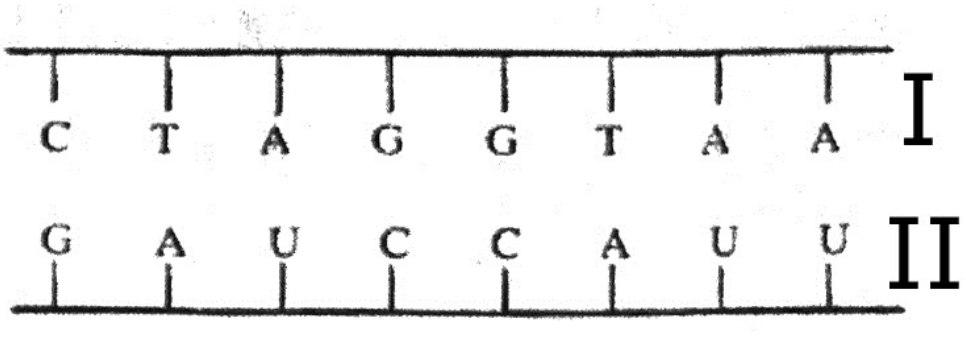

Figura 9. Cadeias de DNA

As cadeias escolhidas, como descrito na Figura 9, representam as duas sequências que serão inseridas como entrada no código desenvolvido nesse relatório.

1. CTAGGTAA
2. GAUCCAUU

O quadro a baixo apresenta todas as soluções ótimas para os subproblemas. Essas soluções foram retiradas da matrizStrings[i][j], soluções essas armazenadas durante a execução do algoritmo de programação dinâmica.

LCS(7,7): GAA
LCS(7,6): GAA
LCS(7,5): GAA
LCS(7,4): GA
LCS(7,3): GA
LCS(7,2): GA
LCS(7,1): GA
LCS(7,0): G
LCS(6,7): GA
LCS(6,6): GA
LCS(6,5): GA
LCS(6,4): GA
LCS(6,3): GA
LCS(6,2): GA
LCS(6,1): GA
LCS(6,0): G
LCS(5,7): CA
LCS(5,6): CA
LCS(5,5): CA
LCS(5,4): G
LCS(5,3): G
LCS(5,2): G
LCS(5,1): G
LCS(5,0): G

LCS(4,7): CA
LCS(4,6): CA
LCS(4,5): CA
LCS(4,4): G
LCS(4,3): G
LCS(4,2): G
LCS(4,1): G
LCS(4,0): G
LCS(3,7): CA
LCS(3,6): CA
LCS(3,5): CA
LCS(3,4): G
LCS(3,3): G
LCS(3,2): G
LCS(3,1): G
LCS(3,0): G
LCS(2,7): CA
LCS(2,6): CA
LCS(2,5): CA
LCS(2,4): A
LCS(2,3): A
LCS(2,2): A
LCS(2,1): A
LCS(2,0): NULL

LCS(1,7): C
LCS(1,6): C
LCS(1,5): C
LCS(1,4): C
LCS(1,3): C
LCS(1,2): NULL
LCS(1,1): NULL
LCS(1,0): NULL
LCS(0,7): C
LCS(0,6): C
LCS(0,5): C
LCS(0,4): C
LCS(0,3): C
LCS(0,2): NULL
LCS(0,1): NULL
LCS(0,0): NULL

Podemos exemplificar a superposição de problemas utilizando os subproblemas intermediários LCS(7,1) e LCS(3,6). Observa-se, pelo algoritmo representado na figura 2, que a solução da LCS(7,1) é composta pela solução de LCS(6,0), visto que LCS(7,1) compara as posições destacadas entre CTAGGTA**A** e G**A**UCCAUU. Como LCS(6,0) aparece em posições anteriores na matrizStrings[i][j], esse é um problema superposto já solucionado.

LCS(3,6), observando-se novamente o algoritmo na figura 2, é composto pelas soluções de LCS(3,5) e LCS(2,6), visto que compara as posições destacadas em CTA**G**GTAA e GAUCC**A**UU. Da mesma forma, como LCS(2,6) e LCS(3,5) aparecem em posições anteriores na matriz, esses são problemas superpostos já solucionados.

7. Referências

R.A. Wagner and M. J. Fischer(1974). The String-to-String Correction Problem. Journal of the ACM.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2001). Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill.

David Maier (1978). "The Complexity of Some Problems on Subsequences and Supersequences". J. ACM. ACM Press.

BAASE, S.; GELDER, A. V.(1999). Computer Algorithms: Introduction to Design and Analysis. 3rd Edition. Pearson.