2.2.3 Calculo de una subsecuencia común más larga

Una subsecuencia es cualquier subconjunto de elementos que mantienen el orden, entonces el problema de encontrar la subsecuencia común más larga se define como:

> Dadas 2 cadenas, sean X e Y, encontrar la longitud de la cadena más larga Z, tal que los caracteres de Z aparezcan en ese mismo orden dentro de las cadenas X e Y.

Aplicaciones

El problema de la subsecuencia común más larga se aplica en bioinformática, es frecuente la necesidad de comparar el ADN de dos o más organismos una secuencia de ADN se representa como una cadena en las letras que representan cada una de las bases posible: A (adenina), G (guanina), C (citosina) y T (tiamina).

Ejemplo: **ACCGGTCGGGATGCACCTGAGAAAGCGG**

Un posible criterio de "similitud" entre secuencias de DNA es encontrar la subsecuencia común más larga de bases que aparezca en las secuencias aún en forma no consecutiva, por ejemplo, para AGCGTAG y GTCAGA la subsecuencia común más larga es GCGA no es lo mismo que la subcadena más larga, ya que se permiten otros caracteres en el medio.

Algoritmo

Dados los conjuntos X e Y de la siguiente forma

$$X = \{x_1, x_2, \dots, x_m\}$$

$$Y = \{y_1, y_2, \dots, y_n\}$$

Z es una subsecuencia de X si existe una secuencia que es estrictamente creciente en el índices de X en $i1, i2, i3, \dots ik$ tales que $X_{ij} = Z_j \; para \; 1 \leq j \leq k$

Regla

Z es una subsecuencia común de *X* si
Z es subsecuencia de *X*
Z es subsecuencia de *Y*

Para cualquier par de cadenas que tengan símbolos en común siempre habrán subsecuencia comunes de longitud máxima.

Solución por fuerza bruta

Un algoritmo de fuerza bruta para resolver LCS es enumerar todas las posibles subsecuencia de *X*, controlar si también es subsecuencia de *Y*, y recordar la más larga de ella.

Este algoritmo es de O(2m), y por lo tanto inviable para *m* grandes, sin embargo, es posible comprobar que LCS tiene una subestructura óptima.

Solución con programación dinámica

Para

$$X = \{x_1, x_2, \dots, x_m\}$$

$$Y = \{y_1, y_2, \dots, y_n\}$$

$$Z = \{z_1, z_2, \dots, z_k\}$$

Donde Z es una secuencia común de longitud máxima LCS donde $pref_i = \{x_1, x_2, \dots, x_i\}$ y $pref_0 = \{\ \}$, dadas las cadenas X de longitud m e Y de longitud n, la longitud de la secuencia común más larga estará dada por la siguiente formula recursiva:

$$SCL(X[0,\dots m], Y[0,\dots,n]) = \begin{cases} 0 & si\ m = 0\ o\ n = 0 \\ SCL(X[0,\dots,m-1], Y[0,\dots,n-1]) + 1 & si\ X[m] == Y[n] \\ max\big(SCL(X[0,\dots,m], Y[0,\dots,n-1]), SCL(X[0,\dots,m-1], Y[0,\dots,n])\big) & otro\ caso \end{cases}$$

La definición recursiva anterior expresa que la subsecuencia común más larga de dos cadenas contiene al elemento *X*[*i*], si *X*[*i*] es igual a *Y*[*j*], concatenado con la subsecuencia común más larga del resto de ambas cadenas. Para demostrar el algoritmo, se presenta el siguiente ejemplo:

Calcular la subsecuencia común más larga entre las siguientes dos cadenas de caracteres:

Entrada:

X = {ABCBDAB} cuya longitud igual a 7

Y = {BDCABA} cuya longitud igual a 6

El algoritmo construye una matriz, la cual llena con los valores de la subsecuencia común más larga para cada pareja de valores, como se muestra en la figura 2.2.3.1.

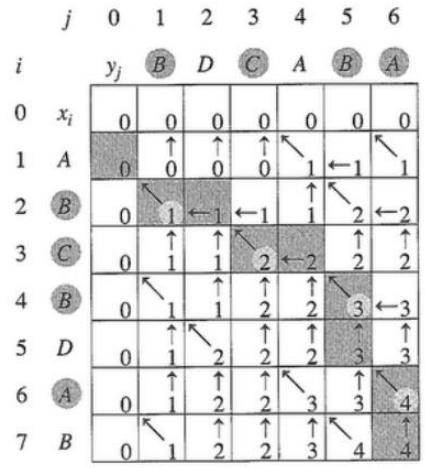

Figura 2.2.3.1 Matriz dinámica para encontrar el SCL

En el código 2.2.3.1 muestra la programación del algoritmo y la salida del ejemplo es la siguiente

```
  ======== -- -- ------------ ----- === ----- --- ------------ ========

| Problema de la SubSecuencia Comun Mas Larga Con Programacion Dinamica |

  ======== -- -- ------------ ----- === ----- --- ------------ ========


        ==> Ingrese la Cadena A: ABCBDAB

        ==> Ingrese la Cadena B: BDCABA


        TABLA DINAMICA

              (POR CUESTION DEL GRAN NUMERO DE DATOS SE OMITIO LA TABLA)

==> La Subsecuencia comun mas larga B en A:


 ==> { BCB }


==> El numero de ocurrencia es: 496
```

Código 2.2.3.1 Algoritmo SCL

```
/**
 * @author sdelaot
 */
public class AlgoritmoLCS {
    private final int MAX_X = 500;
    private final int MAX_Y = 500;
    private char [][] b;
    private int [][] c;
    public AlgoritmoLCS() {
        b = new char[MAX_Y][MAX_X];
        c = new  int[MAX_Y][MAX_X];
    }
    void ejecutarAlgoritmoLCS(char X[], char Y[]) {
        int m,n,i,j;
        m=X.length-1;
        n=Y.length-1;
        for(i=1; i<=m; i++)
            c[i][0]=0;
        for(j=0; j<=n; j++)
            c[0][j]=0;
        for(i=1; i<=m; i++) {
            for(j=1; j<=n; j++) {
                if (X[i]==Y[j]) {
                    c[i][j]=c[i-1][j-1]+1;
```

```
            b[i][j]='/';
            }
         else {
            if (c[i-1][j]>=c[i][j-1]) {
               c[i][j]=c[i-1][j];
               b[i][j]='|';
               }
            else {
               c[i][j]=c[i][j-1];
               b[i][j]='-';
            }
         }
      }
   }
}
void llenarVector( char M[],String cadena) {
   int i;
   char a;
   a= cadena.charAt(0);//getchar();
   M[1]=a;// Apartir de aqui se empesara a llenar Los Vectores
   i=2;
   while(i<cadena.length()) {
      a=cadena.charAt(i-1);
      M[i]=a;
      i++;
      }
   //M[i-1]='\0';
}
/**
 * Imprime cada Ocurrencia del Vector B en A
 */
void imprimirLCS(char X[], int i, int j) {
   if (i==0 || j==0) {
      return;
      }
   if (b[i][j]=='/') {
      imprimirLCS(X,i-1,j-1);
      System.out.print(X[i]);
      }
   else
   if (b[i][j]=='|') {
       imprimirLCS(X,i-1,j);
      }
   else { // "-"
      imprimirLCS(X,i,j-1);
   }
}
void ejecutarAlgoritmo( String cadenaA, String cadenaB ) {
   char A[] = new char[MAX_X];
   char B[] = new char[MAX_X];
```

```
        int tamA,tamB;

        llenarVector( A, cadenaA );
        llenarVector( B, cadenaB );
        tamA = A.length;
        tamB = B.length;
        ejecutarAlgoritmoLCS( A, B );
        System.out.println();
        System.out.println("\tTABLA DINAMICA");
        System.out.println();
        for( int i=0; i<=tamA-1; i++) {
           System.out.print("\t");
           for( int j=0; j<=tamB; j++) {
              if( j+1==tamB+1 ) {
                 System.out.println();
                 }
              else {
                 System.out.print( " " + c[i][j] );
                 }
              }
           }
        System.out.println();
        System.out.println(
              " ==> La Subsecuencia comun mas larga B en A: \n" );
        System.out.print( " ==> { " );
        imprimirLCS( A, tamA-1, tamB-1 );
        System.out.println( " }" );
        System.out.println();
        System.out.print(" ==> El numero de ocurrencia es: ");
        System.out.println( c[tamA-1][tamB-1] + "\n\n" );
    }
}
import java.util.Scanner;
/**
 * @author sdelaot
 */
public class ProbadorLCS {
    public static void main(String[] args) {
        Scanner teclado = new Scanner( System.in );
        AlgoritmoLCS aLcs = new AlgoritmoLCS();
        System.out.println( "\n" );
        System.out.println("   ======== -- -- ------------ ----- === ----- --- ------------ ========  ");
        System.out.println(" | Problema de la SubSecuencia Comun Mas Larga Con Programacion Dinamica |");
        System.out.println("   ======== -- -- ------------ ----- === ----- --- ------------ ========  ");
        System.out.println();
        System.out.print("\t==> Ingrese la Cadena A: ");
        String cadenaA = teclado.nextLine();
        System.out.print("\t==> Ingrese la Cadena B: ");
        String cadenaB = teclado.nextLine();
        aLcs.ejecutarAlgoritmo(cadenaA,cadenaB);
```

```
    }
}
```

Tarea

Llevar al límite el algoritmo LCS empleando algún archivo de la siguiente página web

https://ftp.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG002_NA24385_son/PacBio_CCS_10kb/

Esta referencia tiene secuencias genómicas, reportar los cambios realizados al código para el soporte máximo, o cambios de atributos para que el algoritmo soporte la máxima carga de trabajo (**NOTA**: MAX_X, MAX_Y, b[][], c[][])