

# PROGRAMACIÓN DINÁMICA

GRUPO: HIBISCO

Integrantes:

- ❖ Noura Lachhab Bouhmadi
- ❖ Quintín Mesa Romero
- ❖ Eduardo Rodríguez Cao
- ❖ Ramón Liria Sanchez



# ÍNDICE:

## 1. PLANTEAMIENTO DEL EJERCICIO

### 1.1 PLANTEAMIENTO DEL EJERCICIO

### 1.2 RECURRENCIA UTILIZADA EN EL ALGORITMO

### 1.3 IMPLEMENTACIÓN DEL ALGORITMO

### 1.4 RESULTADOS

### 1.5 CONCLUSIONES DEL EJERCICIO

## 2. USO DE LA PROGRAMACIÓN DINÁMICA EN LA RESOLUCIÓN DEL EJERCICIO

## 3. CONCLUSIONES

# PLANTEAMIENTO DEL EJERCICIO

Dos hermanos fueron separados al nacer y mediante un programa de televisión se han enterado que podrían ser hermanos. Ante esto, realizan un test de ADN para verificar si realmente son hermanos.

Deben encontrar el % de similitud que existe entre estos posibles hermanos. Lo haremos para 2 entradas posibles.

~~~~~  
Hermano 1 \_ abbcdefabcdxzyccd

Hermano 2 \_ abbcdeafbcdzxyccd

~~~~~  
Hermano 1 \_ 010111000100010101010010001001001001

Hermano 2 \_ 110000100100101010001010010011010100  
~~~~~

# RECURRENCIA UTILIZADA EN EL ALGORITMO

|   |       | j | 0 | 1  | 2 | 3  | 4 | 5 | 6 |            |
|---|-------|---|---|----|---|----|---|---|---|------------|
|   |       |   | 0 | 1  | 2 | 3  | 4 | 5 | 6 |            |
| i |       |   | 0 | 1  | 2 | 3  | 4 | 5 | 6 |            |
| 0 | $x_i$ |   | 0 | 0  | 0 | 0  | 0 | 0 | 0 | } cadena y |
| 1 | 0     |   | 0 | 0* | 0 | 0* | 0 | 0 | 1 |            |
| 2 | 1     |   | 0 |    |   |    |   |   |   |            |
| 3 | 1     |   | 0 |    |   |    |   |   |   |            |
| 4 | 0     |   | 0 |    |   |    |   |   |   |            |
| 5 | 1     |   | 1 |    |   |    |   |   |   |            |
| 6 | 0     |   | 0 |    |   |    |   |   |   |            |
| 7 | 0     |   | 0 |    |   |    |   |   |   |            |
| 8 | 1     |   | 0 |    |   |    |   |   |   |            |

*Cadena x*

$$I(i, j) = \begin{cases} 0 & \text{si } i = j = 0 \\ I(i - 1, j - 1) + 1 & \text{si } x_i = y_j \\ \max(I(i - 1, j), I(i, j - 1)) & \text{si } x_i \neq y_j \end{cases}$$

# IMPLEMENTACIÓN DEL ALGORITMO: TABLA

En la implementación del algoritmo que resuelve el problema de encontrar la mayor subsecuencia de ADN común a dos dadas, se ha utilizado una función auxiliar destinada al cálculo de la tabla con las longitudes de la mayor subsecuencia en la que coinciden las dos cadenas de ADN.

## EFICIENCIA TEÓRICA:

Es  $O(n^2)$ , pues tenemos cuatro bucles, tres de ellos al mismo nivel y uno anidado.

```
/**
 * @brief Calcula la tabla con las longitudes de la mayor subsecuencia
 * en la que coinciden los vectores a y b
 * @param a Vector con la primera cadena
 * @param b Vector con la segunda cadena
 */
vector<vector<int>> tableSequences(const vector<char> & a, const vector<char> & b) {
    // Crea la matriz de longitud de subsecuencias
    int sizeA = a.size();
    int sizeB = b.size();
    vector<vector<int>> table(sizeA + 1, vector<int>(sizeB + 1));

    // Rellena la primera fila con 0
    for (int i = 0; i <= sizeB; i++)
        table[0][i] = 0;
    // Rellena la primera columna con 0
    for (int i = 1; i <= sizeA; i++)
        table[i][0] = 0;

    // Rellena la tabla con la longitud de las subsecuencias
    for (int i = 1; i <= sizeA; ++i)
        for (int j = 1; j <= sizeB; ++j){
            if (a[i-1] == b[j-1])
                table[i][j] = 1 + table[i-1][j-1];
            else
                table[i][j] = max(table[i-1][j], table[i][j-1]);
        }

    return table;
}
```

# IMPLEMENTACIÓN DEL ALGORITMO: SUBSECUENCIA

Se recorre la tabla desde el último elemento al primero reconstruyendo la subsecuencia a través de los elementos anteriores en la diagonal, a la izquierda o arriba.

## EFICIENCIA TEÓRICA:

Es  $O(n^2)$ , ya que la función que calcula la tabla es  $O(n^2)$  y a continuación hay un while con  $O(n)$ .

```
/**
 * @brief Busca la mayor subsecuencia en la que coinciden los vector
 * a y b
 * @param a Vector con la primera cadena
 * @param a Vector con la segunda cadena
 * @param result Vector con la subsecuencia resultante
 */
void longestSequence(const vector<char> & a, const vector<char> & b, s
tack<char> & result) {
    auto table = tableSequences(a,b);
    int i = a.size();
    int j = b.size();
    while (i != 0 && j != 0) {
        if ((table[i][j] - 1) == table[i-1][j-1] && a[i-1] == b[j-1]) {
            result.push(a[i-1]);
            i--;
            j--;
        }
        else if (table[i][j] == table[i-1][j])
            i--;
        else
            j--;
    }
}
```



# RESULTADOS

Resultado de ejecutar el programa para las dos secuencias de ADN de la primera entrada:

Introduzca las secuencias...

Primera secuencia: abbcdefabxcdxyccd

Segunda secuencia: abbcdeafbcdxyccd

Tabla de longitud de secuencias:

|    | #2 | a | b | b | c | d | e | a | f | b | c | d  | z  | x  | y  | c  | c  | d  |
|----|----|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| #1 | 0  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| a  | 0  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | 1  | 1  | 1  | 1  | 1  | 1  |
| b  | 0  | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2  | 2  | 2  | 2  | 2  | 2  | 2  |
| b  | 0  | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3  | 3  | 3  | 3  | 3  | 3  | 3  |
| c  | 0  | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4  | 4  | 4  | 4  | 4  | 4  | 4  |
| d  | 0  | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5  | 5  | 5  | 5  | 5  | 5  | 5  |
| e  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 6 | 6 | 6 | 6 | 6  | 6  | 6  | 6  | 6  | 6  | 6  |
| f  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 6 | 7 | 7 | 7 | 7  | 7  | 7  | 7  | 7  | 7  | 7  |
| a  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 7 | 7 | 7  | 7  | 7  | 7  | 7  | 7  | 7  |
| b  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 8 | 8  | 8  | 8  | 8  | 8  | 8  | 8  |
| c  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 9  | 9  | 9  | 9  | 9  | 9  | 9  |
| d  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| x  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | 10 | 11 | 11 | 11 | 11 | 11 |
| z  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | 11 | 11 | 11 | 11 | 11 | 11 |
| y  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | 11 | 11 | 12 | 12 | 12 | 12 |
| c  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | 11 | 11 | 12 | 13 | 13 | 13 |
| c  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | 11 | 11 | 12 | 13 | 14 | 14 |
| d  | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | 11 | 11 | 12 | 13 | 14 | 15 |

La subsecuencia coincidente más larga tiene longitud 15 y es: abbcdefbcdxyccd

El porcentaje de similitud entre las dos cadenas es: 88.2353 %





# CONCLUSIONES

1. En este problema la programación dinámica proporciona una solución óptima y eficiente  $O(n^2)$  frente al algoritmo de fuerza bruta  $O(2^n)$ .
2. Se gasta mucha memoria, tiene que tenerse en cuenta al aplicar el algoritmo sobre datos grandes.
3. No se repiten cálculos