

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.

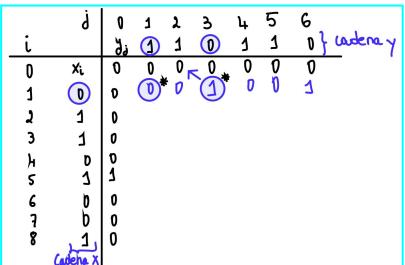
```
Hermano 1 _ abbcdefabcdxzyccd

Hermano 2 _ abbcdeafbcdzxyccd

Hermano 1 _ 0101110001000101010010001001001001

Hermano 2 _ 110000100100101010001001001001001
```

RECURRENCIA UTILIZADA EN EL ALGORITMO



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

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 coindicen los vectores a y b
 * @param a Vector con la primera cadena
 * @param a Vector con la segunda cadena
vector<vector<int>> tableSequences(const vector<char> & a, const vecto
r<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 \le sizeB; i++)
   table[0][i] = 0:
 // Rellena la primera columna con 0
  for (int i = 1; i \le sizeA; i++)
   table[i][0] = 0;
  // Rellena la tabla con la longitud de las subsecuencias
  for (int i = 1; i \le sizeA; ++i)
   for (int j = 1; j \le 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: abbcdefabcdxzyccd Segunda secuencia: abbcdeafbcdzxyccd

Tabla de longitud de secuencias:

	#2	a	Ь	Ь	C	d	e	a	f	Ь	C	d	Z	Х	у	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	
Ь	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	
Ь	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	
е	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	
Ь	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	
х	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	
у	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 %

RESULTADOS

Introduzca las secuencias...

Primera secuencia: 01011100010001010101001001001001001

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

Primera Secuencia: 0101100010001010101001001001001001 Segunda secuencia: 11000010010101001001001001001001001																																					
Tabla																																					
	#2	1	1	0	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	1	1	0	1	0	1	0	0
#1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	1 2	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	2	1	2	3	3	2	2	2	3	3	3	2	2	2	2	2	3	3	3	2	2	3	3	2	2	2	3	2	2	2	2	3
0	0	1	1	2	2	2	2	2	3	3	3	3	3	3	4	3	3	3	3	3	4	3	4	3	3	3	3	4	3	3	3	3	3	3	3	3	3
1	0	1	2	2	2	2	2	3	3	3	4	4	4	5	5	4 5	5	4	5	5	4 5	5	5	4	4 5	4	4	5	5	5	5	4	5	4	5	4	4
1	0	1	2	2	2	2	2	3	3	3	4	4	4	5	5	6	6	6	5	6	6	6	6	6	5	5	5	6	6	6	6	_	_	5	6	5	5
100		1	2	3	2	3	3		3	3	4	4	4	5	6	6	7	7	0	7	7	7	7	7	7	0	7	7	7	7	7	6	6	7	7	0	0
0	0	1	2	3	3	3	4	3	4	5	5	5	5		0	9			,		8		1	,	1	,	,				,	/	,	/	8	-	,
0	0	1	2	3	4	5	5	5	5	5	5	6	6	6	0	9	4	_'	8	9	9	8	9	8	g	g	g	8	8	8	g	9	8	9	9	g	8
1	0	1	2	3	4	5	5	6	6	6	6	6	0	0	4	8	8	8	8	9	9							10	10								10
0	0	1	2	3	4	5	6	6	7	7	7	7	7	7	8	8	9	9	8	9	10	10	10 11	10 11	10 11	10 11	10 11	11	11	10 11							
0	0	1	2	3	4	5	6	0	-	8	,	,	,	8	8	8	9	9	10	10	10		11			12					12	12	12	12			12
0	0	1	2	3	4	2	0	0	4	8	8	9	9	9	9	9	9	9	10	11	11	11	11	11	12	13	13	13	13	13	13	13	13	13	13		13
1	0	1	2	3	4	5	6	7	7	8	9	9	9	10	10	10	10	10	10	11			12				14	14	14	14	14	14	14	14	14		14
0	0	1	2	3	4	2	6	7	8	8	9	10	10	10		11		11		11							14	15	15	15	15	15	15	15	15	-	15
1	0	1	2	3	4	5	6	7	8	8	9	10						12					13							16	16	16	16	16	16		16
0	0	1	2	2	4	2	6	4	8	9	9	10			12								14	14					16	16	16	17	17				17
1	0	1	2	2	4	5	6	7	8	9	10				12		13	14	14	14	14	14	14				16	16	16	17	17	17	18	18	18		18
0	0	1	2	3	4	5	6	7	8	9	10			12		13	14	14	15	15								17	17	17	17	18	18	19	19		19
1	0	1	2	3	7	5	6	7	8	9	10				13	14	14	15		15		16	16	16	16	16	17	17	17	18	18	18	19	19	20	20	20
0	0	1	2	3	7	5	6	7	8	9		11					15		16	16		16					17	18	18		18	19	19	20	20		21
0	0	1	2	2	7	5	6	7	8	9	10			12		14	15	15	16	17	17	17	17	17	18	18	18	18	19	19	19	19	19	20	20	21	22
1	0	1	2	3	4	5	6	7	8	9		11				14	15	16	16	17	17	18	18	18	18				19	20	20	20	20	20			22
0	0	1	2	3	Z	5	6	7	8	9	10	11			14	14	15	16	17	17	18	18	19	19	19	19	19	20	20	20	20	21	21	21			22
0	0	1	2	3	4	5	6	7	8	9		11			14	14	15		17		18		19	19			20			21			21		22		10.00
0	0	1	2	3	4	5	6	7	8	9	10		12			14	15	16	17			19		19				21			21	22		22			23
1	0	1	2	3	4	5	6	7	8	9	10		12		14	15	15	16	17	18	19	20	20	20				22		22	22	22	23	23	23		23
ō	0	1	2	3	4	5	6	7	8	9	10		12			15	16	16	17		19				21				23	23	23	23	23	24	24	24	24
0	0	1	2	3	4	5	6	7	8	9	10			13	14	15	16	16	17	18	19	20		21				23	24	24	24	24	24	24	24		25
1	0	1	2	3	4	5	6	7	8	9		11				15	16	17				20		22					24	25	25	25		25			25
ō	0	1	2	3	4	5	6	7	8	9	10		12			15	16	17	18	18	19	20	21	22	23	23	23	24	24	25	25	26	26	26	26	26	26
0	0	1	2	3	4	5	6	7	8	9		11				15	16	17	18		19	20	21	22	23			24	25	25	25	26	26	27	27		27
1	0	1	2	3	4	5	6	7	8	9		11				15	16	17	18	19	19	20	21	22		24	25	25	25	26	26	26	27	27	28		28
0	0	1	2	3	4	5	6	7	8	9		11				15	16	17			20		21					26	26	26	26	27	27	28			29
ō	0	1	2	3	4	5	6	7	8	9	10		12			15	16	17		19	20	20			23				27	27	27	27	27	28	28	29	30
1	0	1	2	3	4	5	6	7	8	9													21									28	28	28			30

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