Programación Dinámica

ALGORÍTMICA

UNIVERSIDAD DE GRANADA

JOAQUÍN ARCILA PÉREZ
LAURA LÁZARO SORALUCE
CRISTÓBAL MERINO SÁEZ
ÁLVARO MOLINA ÁLVAREZ

ÍNDICE

INTRODUCCIÓN

Esta práctica consiste en el uso de programación dinámica para la resolución de un ejercicio que busca encontrar el porcentaje de similitud entre dos secuencias de ADN. Para ello, lo que deberemos buscar es la mayor subsecuencia en la que coinciden ambas secuencias, pudiendo estar esta no contigua en la cadena de ADN, pero siempre con sus términos ordenados.

Como hemos mencionado, este problema será resuelto mediante la técnica de programación dinámica, basada en cuatro etapas:

1.Naturaleza n-etápica del problema:
Se refiere al hecho de que el programa se puede separar en un conjunto de etapas o decisiones a tomar. En nuestro caso, comenzamos comprobando las dos subcadenas de longitud 1, y vamos aumentando el tamaño hasta llegar a comprobar ambas cadenas completas.

2.Verificación del POB:
El Principio de Optimalidad de Bellman nos dice que cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un problema también debe ser óptima respecto al subproblema que resuelve. Demostremoslo para nuestro caso:

Sean A de longitud p y B de longitud q dos cadenas. Si C, de longitud r es una subcadena común de longitud máxima, se cumple:

-Si los últimos términos de A y B coinciden, entonces este a su vez coincide con el único de C, y la subcadena formada por los términos c_1 a c_r-1 es común de longitud máxima para A y B sin el último símbolo.

-Si los últimos términos de A y B difieren y también difieren los de A y C, se cumple que C es la subcadena común de longitud máxima para A sin el último símbolo y B.

-Si los últimos términos de A y B difieren y también difieren los de B y C, el caso es análogo al anterior.

3.Planteamiento de una recurrencia:
Tenemos dos secuencias, X=(x1,x2,..,xn) e Y=(y1,y2,..,ym), definimos entonces la función a trozos, que representará la subsecuencia más larga entre los componentes de X e Y. Se podría definir como:

$$f(Xi,Yj)=\begin{cases} 0 & \text{si } i=0 \text{ o } j=0 \\ f(Xi-1,Yj-1)+1 & \text{si } xi=yj \\ \max(f(Xi,Yj-1),f(Xi-1,Yj)) & \text{si } xi!=yj \end{cases}$$

4.Calculo de la solución:
El algoritmo que hemos usado para resolver el problema, llamado LCS, lo explicaremos con más detalle en el próximo apartado.

DESARROLLO

Eficiencia teórica

```
void MatrizCalculos (string seq1, string seq2, vector<vector<int>> & matriz) {

for (int j=0; j<=seq2.size(); j++) {
 for (int i=0; i<=seq1.size(); i++) {
  if (j==0 || i==0)
   matriz[i][j] = 0;
  else if (seq1[i-1] == seq2[j-1])
   matriz[j][i] = matriz[j-1][i-1]+1;
  else
   matriz[j][i]=max(matriz[j-1][i], matriz[j][i-1]);
 }
}
}
```

Este bucle se recorre seq2.size()+1 veces, y resolviendo la suma, tenemos: a*n*m+a*(m+n)+a

Este bucle for se recorre seq1.size()+1 veces. Por lo tanto, el sumatorio nos da: an+a, siendo una constante que representa las operaciones del interior del bucle, y n el tamaño de la primera secuencia.

Resolviendo los sumatorios, obtenemos: a*n*m+a*(m+n)+a, por lo que es de orden T(n,m) ∈O(n*m), donde n es el tamaño de la primera secuencia y m el de la segunda. Si fuesen de igual longitud, tendríamos que nuestro algoritmo es de orden O(n^2).

```
string SubSecuencia1 (string seq1, string seq2, vector<vector<int>> & matriz) {

 string resultado = "";
 int indice1 = seq1.size();
 int indice2 = seq2.size();

 while (indice1 !=0 && indice2 !=0) {
  if (seq1[indice1-1]==seq2[indice2-1]) {
   resultado = seq2[indice2-1] + resultado;
   indice1--;
   indice2--;
  }
  else if (matriz[indice2][indice1]==matriz[indice2][indice1-1])
   indice1--;
  else
   indice2--;
 }

 return resultado;
}
```

Este bucle while se recorre seq1.size() + seq2.size() en el peor de los casos. Resolviendo el sumatorio, nos da a*n+a*m, siendo una constante que representa las instrucciones del interior del bucle.

Resolviendo los sumatorios, obtenemos: a*n+a*m por lo que es de orden T(n,m) ∈O(n+m), donde n es el tamaño de la primera secuencia y m el de la segunda.

Sumando la eficiencia de ambas funciones utilizadas, vemos que el orden de eficiencia del algoritmo es O(n*m).

DESARROLLO

Matriz

		a	b	b	c	d	e	f	a	b	c	d	x	z	y	c	c	d
	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
a	0	↖1	↑2	↑3	↑4	↑5	↑6	↑6	↖7	←7	←7	←7	←7	←7	←7	←7	←7	←7
f	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
z	0	↑1	↑2	↑3	↑4	↑5	↑6	↑7	↑7	↑8	↑9	↑10	↑10	↖11	←11	←11	←11	←11
x	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

Para hallar los valores de la matriz de cálculo creamos, en primer lugar, una matriz de tamaño (seq1.length() + 1) x (seq2.length() + 1). La fila y la columna restante, que no guardan ningún término de ninguna de las secuencias, son la primera fila y columna de la matriz y se inicializan ambas a 0. A partir de éstas, calcularemos los valores restantes, recorriendo la matriz de la forma clásica con dos bucles anidados.
Distinguimos dos casos. Por un lado, si el valor de ambas secuencias coincide, la casilla valdrá igual a la casilla situada arriba a la izquierda, más uno. En caso contrario, el valor de la casilla será el máximo entre el valor de la casilla de arriba y el de la casilla a la izquierda.

Para conseguir la subsecuencia más larga, definimos las variables indice1 e indice2 con las que recorreremos ambas cadenas, empezando por el final hasta la posición cero.
Ahora si seq1 en indice1 tiene el mismo valor que seq2 en indice2, metemos al principio de la cadena resultado dicho valor.
En caso contrario, analizamos la matriz y vemos que se pueden dar dos situaciones. La primera en la que matriz[indice1][indice2-1] sea igual a matriz[indice1][indice2], en cuyo caso nos movemos hacia esa casilla, decrementando el indice2, y comparando ahora la misma posición que antes de la primera secuencia, con la casilla decrementada de la segunda secuencia. La segunda situación que se puede dar, es que matriz[indice1-1][indice2] sea igual a matriz[indice1][indice2], en cuyo caso hacemos lo mismo que antes, pero desplazándonos hacia arriba en lugar de hacia la izquierda.
Hemos querido dar un par de subsecuencias posibles para cada ejemplo, por lo que, para la segunda subsecuencia resultante, lo que hacemos es decrementar los índices al contrario que como acabamos de describir.

DESARROLLO

Secuencias resultantes

Primer ejemplo:

Secuencias originales: abbcdefabcdxzyccd
abbcdeafbcdzxyccd

Longitud de la subsecuencia más larga: 15

Subsecuencia 1: abbcdefbcdxyccd

Subsecuencia 2: abbcdeabcdzyccd

Porcentaje de parecido: 88.2353%

Segundo ejemplo:

Secuencias originales: 0101110001000101010100100010010010011
1100001001001010100010100100110101000

Longitud de la subsecuencia más larga: 30

Subsecuencia 1: 100001000101010001000100100100

Subsecuencia 2: 110000001010101001000100100100

Porcentaje de parecido: 83.3333%

CONCLUSIÓN

Para terminar con el trabajo, vamos a describir una breve conclusión.

De la eficiencia teórica de nuestro algoritmo, vemos que es preferible usar programación dinámica, pues hemos conseguido una solución más eficiente que la que se conseguiría con el algoritmo de fuerza bruta, que sería de orden O(2^n), pues habría que comprobar todas las combinaciones posibles.

Además, conseguimos siempre soluciones óptimas, como hemos comprobado, cosa que no se cumple siempre con otro tipo de algoritmos como los Greedy.

BIBLIOGRAFÍA

https://m.cplusplus.com/reference/vector/vector/vector/