Taller 5

Carlos Barón - Andrés Cocunubo

28-02-18

1. Descripción del problema

   Encontrar la subsecuencia común más larga creciente entre dos secuencias de símbolos, posiblemente de diferentes tamaños.

2. Formalización

   - Entradas: Dos secuencias $X_m = [x_1,x_2,...,x_m]$ y $Y_n = [y_1, y_2, ..., y_n]$, posiblemente $m \neq n$.
   - Salida : Una subsecuencia $Z = [z_1, z_2, ..., z_k]$ donde $z_j = x_{i_a} = y_{j_a}$ para la secuencia estrictamente creciente $[i_1 : j_1, i_2 : j_2, ..., i_k : j_k]$.

3. Algoritmos

Algorithm 1 Fuerza Bruta

```
procedure LCSBruteForceAux(i,j,X,Y)
    if i < 0 ∨ j < 0 then return 0
    else if X[i-1] = Y[j-1] then
        return LCSBruteForceAux(i-1, j-1, X, Y) + 1
    else
        s1 ← LCSBruteForceAux(i, j-1, X, Y)
        s2 ← LCSBruteForceAux(i-1, j, X, Y)
        if s1 > s2 then return s1
        else return s2
procedure LCSBruteForce(X,Y)
    return LCSBruteForceAux(|X|, |Y|, X, Y)
```

Algorithm 2 Memoización

```
procedure LCSM_Aux(M,i,j,X,Y)
    if M[i][j] = 0
        if i < 0 ∨ j < 0 then M[i][j] ← 0
        else if X[i] = Y[j] then M[i][j] ← LCSM_Aux(M, i-1, j-1, X, Y) + 1
        else
            s1 ← LCSM_Aux(M, i, j-1, X, Y)
            s2 ← LCSM_Aux(M, i-1, j, X, Y)
            if s1 > s2 then M[i][j] ← s1
            else
                M[i][j] ← s2
    return M[i][j]
procedure LCSM(X,Y)
    let M ∈ ℕ^{|X|x|Y|} ← [0]
    R ← LCSM_Aux(M,|X|,|Y|,X,Y)
    return R
```

Algorithm 3 Bottom-up

```
procedure LCSBU_Aux(M,m,n,X,Y)
    for i ←0 to m do
        M[i][0]← 0
    for i ←0 to n do
        M[0][i]← 0
    for i ← 1 to m do
        for j ←1 to n do
            if X[i] = Y[j] then
                M[i][j] ← M[i-1][j-1] + 1
            else
                s1 ← M[i][j-1] ∧ s2 ← M[i-1][j]
            if s1 > s2 then
                M[i][j] ← s1
            else
                M[i][j] ← s2
    return M[m-1][n-1]
procedure LCSBU(X,Y)
    let M ∈ ℕ^{|X|x|Y|} ← [0]
    let B ← ℕ^{|X|x|Y|} ← end
    R ← LCSBU_Aux(B,M,len(X),len(Y),X,Y)
    c ← LCSBT(B,M,len(X)-1,len(Y)-1,X,Y)
    return R
```

Algorithm 4 Back-Tracking

```
procedure LCSBT(B,M,i,j,X,Y)
    cad ← empty sequence
    while B[i][j] != end do
        if B[i][j] = up then
            if X[i] = Y[j] then
                cad ← cad + X[i]
            i ←i-1
        else if B[i][j] = left then
            if X[i] = Y[j] then
                cad ← cad + Y[j]
            j ←j-1
        else if B[i][j] = diagonal then
            if X[i] = Y[j] then
                cad ← cad + X[i]
            i ←i- 1∧ j ←j- 1
    cad ← cad + X[j]
    return cad
```

Algorithm 5 Programación Dinámica

```
procedure LCS(X,Y)
    let M ∈ ℕ^{|X|x|Y|} ← [0]
    let B ← ℕ^{|X|x|Y|} ← end
    R ← LCSBU_Aux(B,M,len(X),len(Y),X,Y)
    c ← LCSBT(B,M,len(X)-1,len(Y)-1,X,Y)
    return R
```

4. Invariante

   Los datos que se encuentran dentro de la tabla de memoización M son los tamaños de la subsecuencia más larga según los simbolos que se están en las secuencias X, Y recibidas por entrada, es decir, $M[i][j]$ representa el tamaño más grande hasta el simbolo que ha procesado de cada secuencia. Adicionalmente, el llenado de la tabla extra para realizar backtracking $B[i][j]$ posee la misma invariante, pero esta solo guarda un valor que indica de donde proviene.

5. Complejidad

   La complejidad por inspección del algoritmo de bottom-up es $O(n + m + mn)$, para el algoritmo de back-tracking es $O(k)$, la complejidad del algoritmo de programación dinámica por inspección de código es $O(mn)$ donde m es el tamaño de la secuencia uno y n es el tamaño de la secuencia dos.

6. Análisis

Como el algoritmo de fuerza bruta retorna solo el tamaño de la subsecuencia más larga común entre las dos secuencias iniciales, se realiza la memoización de los datos para conocer los tamaños de todas las subsecuencias que se evaluan en la recursión, donde $M[i][j]$ representa el tamaño más grande hasta el simbolo que ha procesado de cada secuencia y $B[i][j]$ indica la dirección en donde se encuentra la solución, "izquierda, derecha, arriba y diagonal".

Se elimina la recursión con el algoritmo bottom-up, utilizando ciclos para recorrer las secuencias para finalmente hacer back-tracking y obtener que simbolos de las cadenas nos interesan para formar la subsecuencia con la solución, se empieza desde la ultima posición de la matriz en donde con ayuda de la matriz $B[i][j]$ y $M[i][j]$ se obtiene la dirección y el tamaño hasta el momento, si se encuentra en una posición válida se guarda el simbolo obtenido de $Y[j]$ si es un movimiento a la izquierda y $X[i] \wedge Y[i]$ son iguales, o de $X[i]$ si es un movimiento hacía arriba y si $X[i] \wedge Y[i]$ son iguales, o de $X[i] \vee Y[i]$ si es un movimiento diagonal y $X[i] \wedge Y[i]$ son iguales.

7. Manual de uso

   Para ejecutar el programa se debe ejecutar LCS.py con python3

   Ejemplo ejecución en ubuntu:

   1. Abrir una terminal en linux en el directorio del programa
   2. Ejecutar en la terminal "python3 LCS.py"
   3. En pantalla se muestra la subsecuencia más larga común de las dos secuencias recibidas por entrada.