Resolución de Tarea 4 - Programación Dinámica (Fecha: 20 de Octubre de 2025)

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5651 - Diseño de Algoritmos I
Septiembre - Diciembre 2025
Estudiante: Junior Miguel Lara Torres (17-10303)

Tarea 4 (9 puntos)

Indice

Pregunta 1

Mi fecha de naimiento es: 8 de **Junio** de 1999 y mi dia de nacimiento es **Martes**. Verificar acá.

La tabla de la distancia de edición entre **Martes** $\rightarrow$ **Junio** es

Index	1	2	3	4	5	6
Día	M	A	R	T	E	S
Mes	J	U	N	I	O	

- En la inicialización de fila 0 y columna 0 tenemos:

0	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0

- Tabla para $i = 1$

0	1	2	3	4	5
1	1	2	3	4	5
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0

- Tabla para $i = 2$

0	1	2	3	4	5
1	1	2	3	4	5
2	2	2	3	4	5
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0

- Tabla para $i = 3$

0	1	2	3	4	5
1	1	2	3	4	5
2	2	2	3	4	5
3	3	3	3	4	5
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0

- Tabla para $i = 4$

0	1	2	3	4	5
1	1	2	3	4	5
2	2	2	3	4	5
3	3	3	3	4	5
4	4	4	4	4	5
5	0	0	0	0	0
6	0	0	0	0	0

- Tabla para $i = 5$

0	1	2	3	4	5
1	1	2	3	4	5
2	2	2	3	4	5
3	3	3	3	4	5
4	4	4	4	4	5
5	5	5	5	5	5
6	0	0	0	0	0

- Tabla para $i = 6$

0	1	2	3	4	5
1	1	2	3	4	5
2	2	2	3	4	5
3	3	3	3	4	5
4	4	4	4	4	5
5	5	5	5	5	5
6	6	6	6	6	$\boxed{6}$

Por lo tanto, a distancia de edición entre las cadenas es 6.

> [!NOTE] Dado que no existe al menos una letra en común entre los estados iniciales de las palabras notamos que la distancia es de 6 igual a la longitud de la palabra mas larga (martes). Esto básicamente simula que se debe cambiar todos los caracteres para conseguir el objetivo. Acá el código en C++ con el algoritmo respectivo para realizar pruebas: distancia_edicion.cpp.

Pregunta 2

Definimos $DP[i][j]$ como la longitud del par de subarreglos familiares más largo, donde **el primer subarreglo termina en el índice i y el segundo subarreglo termina en el índice j**, asumiendo $i < j$.

Para calcular $DP[i][j]$, consideramos la posibilidad de extender un par familiar anterior, $DP[i-1][j-1]$.

Sea $L' = DP[i-1][j-1]$.

1. **Condición de Coprimos:** Los elementos $A[i]$ y $A[j]$ deben ser coprimos (es decir, $\gcd(A[i], A[j]) = 1$).
2. **Condición de Disyunción:** El nuevo par de longitud $L = L' + 1$ debe ser disjunto. Dado que $i < j$ y la longitud L es $L' + 1$, el primer subarreglo va desde $i - L + 1$ hasta i, y el segundo va desde $j - L + 1$ hasta j. Para que sean disjuntos, el índice final del primero (i) debe ser estrictamente menor que el índice inicial del segundo ($j - L + 1$).

   $$i < j - L + 1 \quad \Rightarrow \quad L \leq j - i$$

   Si esta condición se cumple, los subarreglos son disjuntos (o adyacentes si $L = j - i$).

La **recurrencia** es:

$$DP[i][j] = \begin{cases} DP[i-1][j-1] + 1 & \text{si } \begin{cases} \gcd(A[i], A[j]) = 1 \\ \wedge \\ DP[i-1][j-1] + 1 \leq j - i \end{cases} \\ 0 & \text{en otro caso} \end{cases}$$

El valor final buscado es el máximo valor encontrado en toda la tabla DP.

Complejidad

La dependencia $DP[i][j] \leftarrow DP[i-1][j-1]+1$ es puramente diagonal. Esto significa que, al calcular la fila i, solo necesitamos los resultados de la fila anterior $i-1$.

- **Complejidad Temporal**
  - **Doble Bucle:** El algoritmo utiliza dos bucles (uno para el índice final del primer subarreglo, i, y otro para el índice final del segundo, j). Estos bucles anidados recorren aproximadamente $n(n-1)/2$ pares de índices, lo que resulta en una complejidad de $O(n^2)$.
  - **Operaciones Constantes:** Dentro del bucle, la operación crucial (verificar la condición de coprimos mediante el Máximo Común Divisor y actualizar el valor de DP) se realiza en tiempo constante, $O(1)$, según las condiciones del problema.
- **Complejidad Espacial**

  Dado que la dependencia es local (solo de la diagonal anterior), podemos aplicar la técnica de "ahorro de espacio". En lugar de un arreglo 2D de $O(n^2)$, utilizaremos un arreglo 1D auxiliar de tamaño $O(n)$ para almacenar solo los valores de la fila anterior, reduciendo la memoria adicional a $O(n)$.

Implementación en C++

La solución se implementa en C++. Dada la restricción de $O(1)$ para las operaciones aritméticas, se incluye una implementación simple del Máximo Común Divisor (GCD) basada en el algoritmo de Euclides.

El archivo funcional se encuentra en family_array.cpp

```cpp
int calculate_gcd(int a, int b) {
    while (b) {
        a %= b;
        swap(a, b);
    }
    return a;
}

int longest_familiar_subarrays(const vector<int> &A, int n) {
    if (n < 2) return 0;

    // Usamos A...A[n]. El vector A indexado a 0 debe tener tamaño n+1.
    // Creamos un vector interno que representa A[1..n].
    // Creamos una copia A_1_based para facilitar la lógica de A[i] donde i >= 1.
    vector<int> A_1_based(n + 1);
```

```cpp
for (int i = 1; i-1 < n; ++i) {
    A_1_based[i] = A[i - 1];
}

// DP[j] almacenará el valor de DP[i-1][j] (de la fila anterior).
// DP_row[j] es la longitud del par familiar más largo que termina en (i-1, j).
// Inicialmente, todos son 0. Usamos tamaño n+1 para indexar 1 a n.
// Esto cumple con la restricción de memoria adicional O(n).
vector<int> DP_row(n + 1, 0);
int max_length = 0;

// Iteramos 'i', el índice de terminación del primer subarreglo. (i = 1 a n)
for (int i = 1; i <= n; ++i) {
    // Variable temporal para almacenar DP[i-1][j-1] (dependencia diagonal).
    // Se inicializa a 0, que es DP[i-1][i-1].
    int prev_diag_val = 0;

    // Iteramos 'j', el índice de terminación del segundo subarreglo.
    // (j = i+1 a n) Forzamos j > i para asegurar que los
    // pares (i, j) sean únicos y en orden.
    for (int j = i + 1; j <= n; ++j) {
        // 1. Guardamos el valor actual de DP_row[j] (que es DP[i-1][j]).
        //    Este valor se convertirá en la dependencia diagonal
        //    (DP[i-1][j]) para la próxima iteración de j (j+1) en la fila i
        //    (i.e., j_new = j+1, i_new=i+1).
        int current_diag_val = DP_row[j];
        int new_length = 0;

        // 2. Condición de Coprimidad: Chequeamos si A[i] y A[j] son coprimos.
        if (calculate_gcd(A_1_based[i], A_1_based[j]) == 1) {
            // L' es la longitud del par familiar que terminaba en (i-1, j-1).
            int L_prime = prev_diag_val;
            int potential_length = L_prime + 1;

            // 3. Condición de Disyunción: El nuevo par de longitud L debe
            //    ser disjunto. El subarreglo termina en i. Si la longitud
            //    es L, comienza en i - L + 1. Para que los subarreglos
            //    sean disjuntos, L debe ser a lo sumo la distancia entre
            //    ellos: L <= j - i
            if (potential_length <= j - i) {
                new_length = potential_length;
            }
        }

        // 4. Actualizamos el máximo global.
        max_length = max(max_length, new_length);
```

```
            // 5. Almacenamos el nuevo valor DP[i][j] en el arreglo DP_row[j]
            //    para la siguiente fila.
            DP_row[j] = new_length;

            // 6. Actualizamos prev_diag_val para la próxima iteración del
            // bucle j. DP[i-1][j] se convierte en DP[i-1][(j+1)-1].
            prev_diag_val = current_diag_val;
        }
    }
    return max_length;
}
```