Análisis y Diseño de Algoritmos

Divide y Vencerás

Francisco Javier Mercader Martínez
Pedro Alarcón Fuentes

En esta memoria se explicará el código utilizado para resolver el problema de encontrar, en una cadena dada, la subcadena de longitud **m** que contiene la mayor cantidad de apariciones consecutivas de un carácter específico **C**. Las dos funciones son:

1. **resolver_directo(A, m, C)**
2. **divide_y_venceras(A, m, C, l, r)**

1. Función resolver_directo(A, m, C)

```python
def resolver_directo(A, m, C):
    """
    Encuentra la subcadena óptima utilizando un método directo

    :param A: Cadena original (str)
    :param m: Longitud de la cadena (int)
    :param C: Carácter a buscar (str)
    :return: tuple: Índice de inicio de la subcadena óptima y el número máximo de
    ↪ apariciones consecutivas.
    """
    n = len(A)
    max_consecutivos = 0
    inicio_optimo = -1

    # Recorrer todas las subcadenas de longitud m
    for i in range(n - m + 1):
        subcadena = A[i:i + m]

        # Contar el número máximo de apariciones consecutivas de C en la
        ↪ subcadena
        contador_actual = 0
        max_actual = 0

        for char in subcadena:
            if char == C:
                contador_actual += 1
                if contador_actual > max_actual:
                    max_actual = contador_actual
```

```
            else:
                contador_actual = 0

        # Actualizar la mejor solución encontrada
        if max_actual > max_consecutivos:
            max_consecutivos = max_actual
            inicio_optimo = i

    return inicio_optimo, max_consecutivos
```

Descripción general

Esta función implementa un algoritmo directo que examina todas las posibles de longitud **m** en la cadena **A** y determina cuál de ellas contiene el mayor número de apariciones consecutivas del carácter **C**. Es un enfoque de fuerza bruta que garantiza encontrar la solución óptima al evaluar exhaustivamente todas las opciones posibles.

Parámetros de entrada

- **A**: Cadena original donde se buscarán las subcadenas.
- **m**: Longitud de las subcadenas a considerar.
- **C**: Carácter cuyo número de apariciones consecutivas se desea maximizar.

Proceso del algoritmo

1. **Preparación:**
   - Calculamos **n**, que es la longitud de la cadena **A**.
   - Inicializamos **max_consecutivos** en 0 para almacenar el máximo de **C** consecutivos encontrados hasta ahora.
   - Inicializamos **inicio_optimo** en -1 para guardar el índice de inicio de la mejor subcadena encontrada.
2. **Recorrido de subcadenas:**
   - Se utiliza un bucle **for** que va desde **i = 0** hasta **i = n - m**, de modo que se puedan extraer todas las subcadenas de longitud **m** sin exceder los límites de la cadena.
   - En cada iteración, se extrae la subcadena **subcadena = A[i:i + m]**.
3. **Cálculo de apariciones consecutivas:**
   - Para cada subcadena, se inicializan **contador_actual** y **max_actual** a 0.
   - Recorremos cada carácter de la subcadena:
     - Si el carácter es igual a **C**, se incrementa **contador_actual** y se actualiza **max_actual** si **contador_actual** es mayor.
     - Si el carácter no es **C**, se reinicia **contador_actual** a 0.

- Este proceso permite determinar el número máximo de apariciones consecutivas de **`C`** en la subcadena actual.

4. **Actualización de la mejor solución:**

   - Si **`max_actual`** es mayor que **`max_cosecutivos`**, se actualizan **`max_consecutivos`** con **`max_actual`** y **`inicio_optimo`** con el índice actual **`i`**.

5. **Resultado:**

   - Al finalizar el bucle, se retorna una tupla **`(inicio_optimo, max_consecutivos)`**, que indica el índice de inicio de la subcadena óptima y el número máximo de apariciones consecutivas de **`C`** en dicha subcadena.

Ejemplo de uso

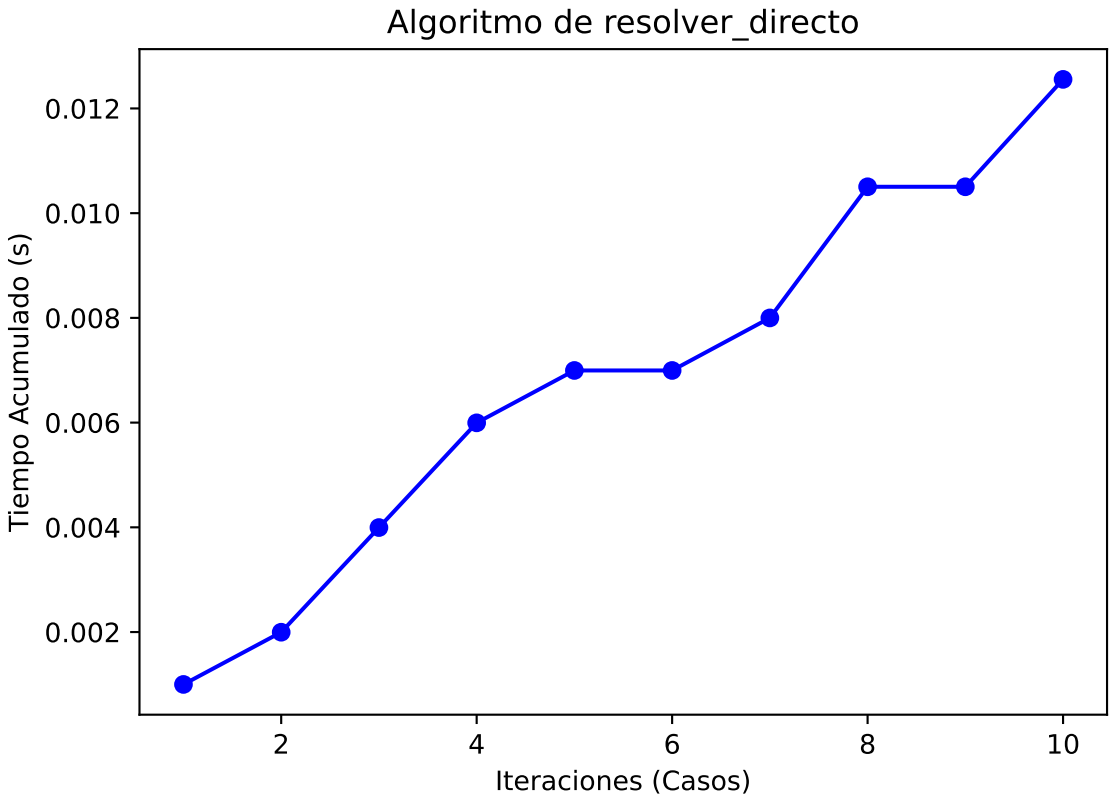
```
if __name__ == '__main__':
    # Generar una cadena de ejemplo con el alfabeto
    alfabeto = "abcdefghijklmnopqrstuvwxyz"
    C = 'c'  # Carácter a buscar
    m = 100  # Tamaño de la subcadena fijo
    num_pruebas = 10 # Número de pruebas a realizar para comprobar que el código
    ↪ funciona

    for i in range(num_pruebas):
        print(f"\n -- Prueba {i + 1} --")
        A = ''.join(random.choices(alfabeto, k=1000))  # Cadena aleatoria de
    ↪ longitud 1000
        resultado = resolver_directo(A, m, C)
        print(f"Índice de inicio: {resultado[0]} \nMáximo de apariciones
        ↪ consecutivas: {resultado[1]}")
```

```
##
##  -- Prueba 1 --
## Índice de inicio: 440
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 2 --
## Índice de inicio: 0
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 3 --
## Índice de inicio: 331
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 4 --
## Índice de inicio: 422
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 5 --
```

```
## Índice de inicio: 174
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 6 --
## Índice de inicio: 579
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 7 --
## Índice de inicio: 0
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 8 --
## Índice de inicio: 567
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 9 --
## Índice de inicio: 630
## Máximo de apariciones consecutivas: 2
##
##  -- Prueba 10 --
## Índice de inicio: 48
## Máximo de apariciones consecutivas: 2
```

Análisis de la complejidad



El algoritmo tiene una complejidad temporal lineal respecto al tamaño de la cadena