Tarea 7
Algoritmos

Emmanuel Peto Gutiérrez
José Luis Vázquez Lázaro

4 de diciembre de 2022

Problema 2

a)

Sea $L(i, j)$ la longitud de la subsecuencia palindrómica más larga para una secuencia A en el subarreglo $A[i...j]$.

Se define $L(i, j)$ de forma recursiva de la siguiente manera:

$$L(i,j) = \begin{cases} 0 & \text{si} \quad i > j \\ 1 & \text{si} \quad i = j \\ 2 + L(i+1, j-1) & \text{si} \quad A[i] = A[j] \text{ con } i < j \\ \text{máx}(L(i+1, j), L(i, j+1)) & \text{si} \quad A[i] \neq A[j] \text{ con } i < j \end{cases}$$

Demostración por inducción.

Caso base 1: $i > j$

Se considera que un subarreglo $A[i...j]$ es vacío cuando $i > j$. Entonces la longitud de cualquier cadena será 0 si $i > j$ y se define $L(i, j) = 0$ para este caso, por lo que la definición es correcta.

Caso base 2: $i = j$

Un subarreglo $A[i...j]$ tiene longitud 1 cuando $i = j$. Cualquier cadena de longitud 1 es palíndroma y se define $L(i, j) = 1$ cuando $i = j$, por lo que en este caso la definición es correcta.

H.I. Supongamos que la definición de $L(k, l)$ es correcta cuando $i \leq k \leq l \leq j$ y $|A[k...l]| < |A[i...j]|$.

Se demostrará que $L(i, j)$ es la longitud del palíndromo más largo formado por una subsecuencia de $A[i...j]$.

Subcaso 1: $A[i] = A[j]$.

Sea p el palíndromo más largo formado por una subsecuencia de $A[i+1...j-1]$. Entonces, se puede construir un palíndromo más largo que sea subsecuencia de $A[i...j]$ simplemente concatenando el caracter $A[i]$ (o $A[j]$) en ambos lados de p: $A[i] + +p + +A[j]$.

Por H.I., $|p| = L(i+1, j-1)$, entonces $|A[i] + +p + +A[j]| = |p| + 2 = L(i+1, j-1) + 2$. Para este caso, se define $L(i,j) = L(i+1, j-1) + 2$, y por lo tanto la definición es correcta.

Subcaso 2: $A[i] \neq A[j]$.

En este caso, se sabe que no pueden ser ambas ($A[i]$ y $A[j]$) extremos del mismo palíndromo, pues son caracteres diferentes. Así que simplemente se calcula la longitud del palíndromo más largo formado con $A[i...j-1]$ y el más largo formado con $A[i+1...j]$, los cuales, por H.I. serán $L(i, j-1)$ y $L(i+1, j)$.

El palíndromo más largo formado con $A[i...j]$ será el más largo entre el formado por $A[i...j-1]$ y el formado por $A[i+1...j]$. La longitud de ese palíndromo será $\max(L(i, j-1), L(i+1, j))$ y así se define $L(i,j)$, por lo que la definición es correcta. ■

b)

Supongamos que existe una función `buildMat(n)` que construye una matriz de $n \times n$ con todas sus entradas en -1. Entonces las siguientes funciones en python calculan la longitud de la subsecuencia más larga que es palindrómica. `longMaxPal` la calcula para la subcadena de i a j y `longMaxPal2` lo calcula para la cadena completa.

```python
def longMaxPal(cadena, i, j, matriz):
    if matriz[i][j] == -1:
        if i>j:
            matriz[i][j] = 0
        elif i==j:
            matriz[i][j] = 1
        else:
            if cadena[i] == cadena[j]:
                matriz[i][j] = 2 + longMaxPal(cadena, i+1, j-1, matriz)
            else:
                v1 = longMaxPal(cadena, i+1, j, matriz)
                v2 = longMaxPal(cadena, i, j-1, matriz)
                matriz[i][j] = max(v1, v2)
    return matriz[i][j]
```

```python
def longMaxPal2(cadena):
    n = len(cadena)
    matriz = buildMat(n)
    return longMaxPal(cadena, 0, n-1, matriz)
```

La correctez es inmediata de la ecuación definida en el inciso **a)** ($L(i,j)$).

Ahora analizaremos la complejidad.

Observe que, salvo por las llamadas recursivas, se realizan un número constante de operaciones dentro de la función `longMaxPal`. La condición del if `matriz[i][j] == -1` va a ser verdadera exactamente una vez para cada par (i,j). Eso significa que se calcula cada entrada de la matriz exactamente una vez. Como es una matriz de $n \times n$ entonces se realizan $O(n^2)$ operaciones en total.

c)

El siguiente código en python calcula la longitud de la subsecuencia palindrómica más larga.

```python
def longMPIt(cadena):
    n = len(cadena)
    matriz = buildMat(n)
    for i in range(n-1, -1, -1):
        for j in range(n):
            if i>j:
                matriz[i][j] = 0
            elif i==j:
                matriz[i][j] = 1
            else:
                if cadena[i] == cadena[j]:
                    matriz[i][j] = 2 + matriz[i+1][j-1]
                else:
                    v1 = matriz[i+1][j]
                    v2 = matriz[i][j-1]
                    matriz[i][j] = max(v1, v2)
    return matriz[0][n-1]
```

La correctez es inmediata del inciso anterior.

La complejidad es $O(n^2)$ al igual que en la versión recursiva.

d)

Supongamos que la función `longMPIt` devuelve no sólo la entrada $(0, n-1)$ de `matriz` sino que devuelve la matriz completa. También supongamos que existe una función `buildMat2(n)` que construye una matriz de $n \times n$ donde todas sus entradas son cadenas vacías.

La siguiente función `construyePML(cadena)` construye la subsecuencia palindrómica más larga dada una cadena.

```python
def construyePML(cadena):
    matLongs = longMPIt(cadena)
    n = len(cadena)
    matCads = buildMat2(n)
    for i in range(n-1, -1, -1):
        for j in range(n):
            if i==j:
                matCads[i][j] = cadena[i]
            elif i<j:
                if cadena[i] == cadena[j]:
                    letra = cadena[i]
                    matCads[i][j] = letra + matCads[i+1][j-1] + letra
                else:
                    if matLongs[i+1][j] > matLongs[i][j-1]:
                        matCads[i][j] = matCads[i+1][j]
```

```
            else:
                matCads[i][j] = matCads[i][j-1]
    return matCads[0][n-1]
```