PRÁCTICA 3 EN PYTHON

07/12/2022

DIMITRIS GIANNAKOS KAISARIS

EDUARDO JUNOY ORTEGA

ALGORITMIA Y ESTRUCTURAS DE DATOS

GRUPO 14
LABORATORIO 02

I. EL PROBLEMA DE SELECCIÓN

1-A QuickSelect básico

1. Función split:

```
"""  Reparte los elementos de un array entre dos arrays con los elementos menores y mayores.

    Argumentos:

        t(np.ndarray): el array que queremos partir

    Devuelve:

        Una tupla con los elementos menores, el elemento t[0] y los elementos mayores.

    Autor:

        Dimitris y Eduardo

  """
def split(t: np.ndarray)-> Tuple[np.ndarray, int, np.ndarray]:

  mid = t[0]
  t_l = [u for u in t if u < mid]
  t_r = [u for u in t if u > mid]
  return (t_l, mid, t_r)
```

2. Función qsel:

```
"""  Aplica el algoritmo QuickSelect a un array dado de manera recursiva

    Argumentos:

        t(np.ndarray): el array a ordenar
        k(int): la clave del algoritmo

    Devuelve:

        el valor del elemento que ocuparía el índice k en una ordenacion de t si ese elemento existe y None si no.


    Autor:

        Dimitris y Eduardo

  """
def qsel(t: np.ndarray, k: int)-> Union[int, None]:
  if len(t) == 1 and k == 0:
```

```
        return t[0]
    elif k >= len(t) or k < 0:
        return None

    t_l, mid, t_r = split(t)
    m = len(t_l)
    if k == m:
        return mid
    elif k < m:
        return qsel(t_l, k)
    else:
        return qsel(t_r, k-m-1)
```

3. Función qsel_nr

```
""" Aplica el algoritmo QuickSelect a un array dado de manera iterativa

        Argumentos:

            t(np.ndarray): el array a ordenar
            k(int): la clave del algoritmo

        Devuelve:

            el valor del elemento que ocuparía el índice k en una ordenacion de t si ese
elemento existe y None si no.


        Autor:

            Eduardo

    """
def qsel_nr(t: np.ndarray, k: int)-> Union[int, None]:
    while len(t) > 0:
        if len(t) == 1 and k == 0:
            return t[0]

        t_l, mid, t_r = split(t)
        m = len(t_l)
        if k == m:
            return mid
        elif k < m:
            t = t_l
        else:
            t = t_r
            k = k-m-1
```

1-B QuickSelect_5

1. Función split_pivot:

""" Reparte los elementos de un array entre dos arrays con los elementos menores y mayores a partir del elemento medio.

Argumentos:

t(np.ndarray): el array que queremos partir
mid(int): el elemento medio

Devuelve:

Una tupla con los elementos menores, el elemento medio y los elementos mayores.

Autor:

Dimitris y Eduardo

"""

```
def split_pivot(t: np.ndarray, mid: int) -> Tuple[np.ndarray, int, np.ndarray]:
    t_l = [u for u in t if u < mid]
    t_r = [u for u in t if u > mid]
    return (t_l, mid, t_r)
```

2. Función pivot5:

""" Algoritmo que devuelve el "pivote 5" de acuerdo al procedimiento de mediana de medianas

Argumentos:

t(np.ndarray): el array que queremos ordenar

Devuelve:

El "pivote 5" del array

Autor:

Dimitris y Eduardo

"""

```
def pivot5(t: np.ndarray)-> int:
    if len(t) < 5:
        s = sorted(t)
        return s[int(len(t)/2)]

    i = 0
```

```
    m = []

    while i < len(t):
        res = len(t) - i
        if res >= 5:
            t5 = t[i:i+4]
            s = sorted(t5)
            p5 = s[int(len(s)/2)]
            m.append(p5)
            i += 5
        else:
            t5 = t[i:]
            s = sorted(t5)
            p5 = s[int(len(t5)/2)]
            m.append(p5)
            break

    pivot = qsel5_nr(m, int(len(m)/2))

    return pivot
```

3. Función qsel5_nr:

```
""" Aplica el algoritmo QuickSelect a una tabla de 5 elementos de manera iterativa

        Argumentos:

            t(np.ndarray): el array a ordenar
            k(int): la clave del algoritmo

        Devuelve:

            el valor del elemento que ocuparía el índice k en una ordenacion de t si ese
elemento existe y None si no.



        Autor:

            Dimitris y Eduardo

    """
def qsel5_nr(t: np.ndarray, k: int)-> Union[int, None]:
    if  k >= len(t) or k < 0:
        return None

    if len(t) == 1:
        return t[0]

    m = -1
```

```
    while k != len(t):
        p = pivot5(t)

        if p is None:
            return None

        t_l, p, t_r = split_pivot(t, p)
        m = len(t_l)

        if k == m:
            return p
        if k < m:
            t = t_l
        elif k > m:
            t = t_r
            k = k-m-1
```

1-C QuickSort_5

1. Función qsort_5:

```
""" Aplica el algoritmo QuickSort a una tabla de 5 elementos utilizando las funciones anteriores

        Argumentos:

            t(np.ndarray): el array a ordenar

        Devuelve:

            La tabla ordenada

        Autor:

            Dimitris y Eduardo

    """
def qsort_5(t: np.ndarray) -> np.ndarray:
    t2 = np.zeros(len(t), dtype=int)
    for i in t:
        pos = qsel5_nr(t, i)
        t2[pos] = i
    return t2
```

I-D Cuestiones sobre QuickSelect y QuickSort

Contestar razonadamente a las siguientes cuestiones incluyendo gráficas si fuera preciso.

1. Argumentar que MergeSort ordena una tabla de 5 elementos en a lo sumo 8 comparaciones de clave.

En una tabla de 5 elementos [1,2,3,4,5], MergeSort hacer un primer partir en [1,2,3] y [4, 5]. Posteriormente, vuelve aplicar la rutina partir a estas subtablas resultando en las siguientes subtablas: [1,2], [3], [4] y [5]. Y, por último, se realiza el último partir de los primeros dos elementos: [1], [2], [3], [4] y [5]. En este procedimiento hemos realizado 4 CDC dentro de la rutina partir.
A continuación se aplica la rutina combinar de la manera inversa, donde se vuelven a realizar 4 CDC: [1,2], [3], [4] y [5] → [1,2,3] y [4, 5] → [1,2,3,4,5]
El total de CDC es 8 a lo sumo, esto también viene demostrado por la fórmula aplicable a la función partir y combinar: (“techo”)⌈$\log_2$N) + 1 = ⌈$\log_2$5) + 1 = 3 +1 = 4
4*2 (una para cada función) = 8.
Visualmente es más fácil verlo así, aunque en esta figura el ejemplo sea de una tabla de 8 elementos:

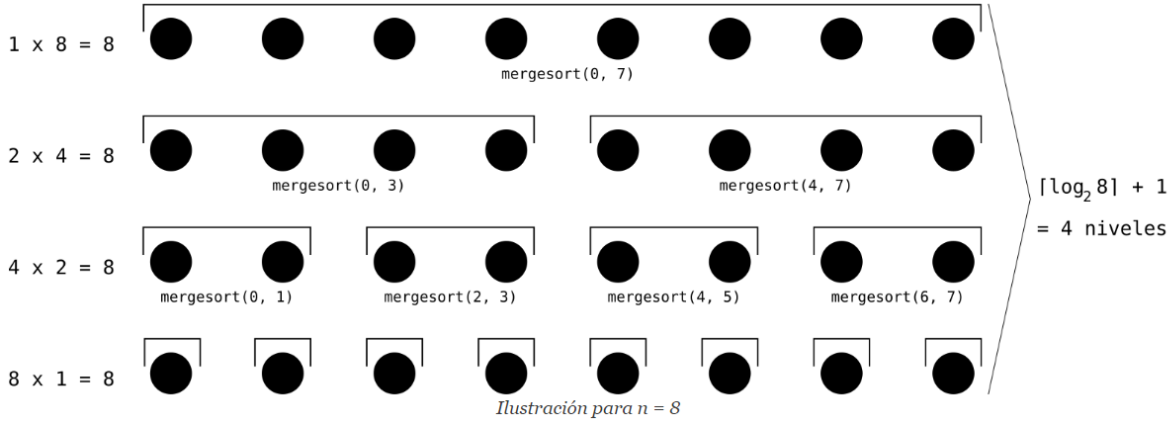

Ilustración para n = 8

2. En realidad, en qsel_5 solo queremos encontrar la mediana de una tabla de 5 elementos, pero no ordenarla. ¿Podríamos reducir así el número de comparaciones de clave necesarias?

Sabemos que las comparaciones de clave necesarias en qsel_5 para encontrar la mediana son $\frac{8N}{5}$ por lo que sustituyendo el valor de N por 5, que son los elementos en los que estamos buscando la mediana, nos da un resultado de 8. Sin embargo, utilizando únicamente la función para encontrar la mediana necesitaríamos 5 comparaciones de clave, ya que se puede encontrar con un número de comparaciones de O(N).

3. ¿Que tipo de crecimiento cabría esperar en el caso peor para los tiempos de ejecución de nuestra función qsort_5? Intenta justificar tu respuesta primero experimental y luego analíticamente.

Sabemos que la función qsel_5 de su interior tiene un rendimiento no mayor a $\frac{8N}{5}$ y la función quicksort tiene un coste medio es $2Nlog(N) + O(N)$. Sustituyendo el valor de qsel_5 en el rendimiento medio de QuickSort y sabiendo que se de qsel_5 la tabla resultante está ordenada, tenemos que la fórmula de qsel_5 es $\frac{16N}{5}log(N)$.

II. PROGRAMACIÓN DINÁMICA

II-A . Distancias entre cadenas y subcadena común más larga

1. Función edit_distance:

```
""" Compara dos cadenas utilizando la menor cantidad de memoria posible calculando su distacia de edición

    Argumentos:

        str_1(str): primera cadena
        str_2(str): segunda cadena

    Devuelve:

        La distancia de edición entre las cadenas

    Autor:

        Dimitris y Eduardo

"""

def edit_distance(str_1: str, str_2: str) -> int:
    distance = np.zeros((len(str_1)+1, len(str_2)+1), dtype=int)
    distance[0, 1:] = range(1, len(str_2)+1)
    distance[1:, 0] = range(1, len(str_1)+1)

    for i in range(1, len(str_1)+1):
        for j in range(1, len(str_2)+1):
            distance[i, j] = distance[i-1, j-1] if str_1[i-1] == str_2[j-1] else 1 + min(distance[i-1, j-1], distance[i-1, j], distance[i, j-1])

    return distance[-1,-1]
```

2. Función max_subsequence_length:

```
""" Compara dos cadenas utilizando la menor cantidad de memoria posible calculando su longitud de la subsecuencia común aunque no necesariamente consecutiva

    Argumentos:

        str_1(str): primera cadena
        str_2(str): segunda cadena

    Devuelve:

        La longitud de una subsecuencia comun a las cadenas

    Autor:
```

```
        Dimitris y Eduardo

    """
def max_subsequence_length(str_1: str, str_2: str) -> int:

    matrix = np.zeros((len(str_1)+1, len(str_2)+1), dtype=int)

    for i in range(1, len(str_1)+1):
        for j in range(1, len(str_2)+1):
            if (str_1[i-1] == str_2[j-1]):
                matrix[i,j] = 1 + matrix[i-1, j-1]

            else :
                matrix[i, j] = max(matrix[i-1, j], matrix[i, j-1])

    return matrix[-1, -1]
```

3. Funciones max_index y max_common_subsequence:

```
""" Encuentra el valor máximo y el índice máximo de una lista

        Argumentos:

            values(List): Los diferentes valores de una lista dada

        Devuelve:

            Una tupla con el valor máximo y el índice máximo

        Autor:

            Dimitris y Eduardo
"""
def max_index (values: List)->tuple:
    max_value = max(values)
    max_index = values.index(max_value)
    return max_value, max_index

""" Compara dos cadenas utilizando la menor cantidad de memoria posible creando una
subcadena común aunque no necesariamente consecutiva

        Argumentos:

            str_1(str): primera cadena
            str_2(str): segunda cadena

        Devuelve:
```

```
        La subcadena común a las cadenas

    Autor:

        Dimitris y Eduardo
"""
def max_common_subsequence(str_1: str, str_2: str) -> str:
    lenght = np.zeros((len(str_1)+1, len(str_2)+1), dtype=int)
    mmatrix = np.empty((len(str_1)+1, len(str_2)+1), dtype=str)

    for i in range(1, len(str_1)+1):
        for j in range(1, len(str_2)+1):
            if str_1[i-1] == str_2[j-1]:
                lenght[i, j] = 1 + lenght[i-1, j-1]
                mmatrix[i,j] = 'D'
            else:
                lenght[i,j] = max(lenght[i-1, j], lenght[i, j-1])
                ind = max_index((lenght[i-1, j], lenght[i, j-1]))
                mmatrix[i,j] = 'U' if ind else 'L'
    print(mmatrix)
    print(lenght)

    esp = ""
    while mmatrix[i, j] != '':
        if mmatrix[i, j] == 'D':
            i, j = i-1, j-1
            esp = str_1[i] + esp
        elif mmatrix[i, j] == 'U':
            j = j-1
        elif mmatrix[i, j] == 'L':
            i = i-1
        print(i, j)
    return esp
```

II-B Multiplicación óptima de matrices

1. Función min_mult_matrix:

```
def min_mult_matrix(l_dims: List[int]) -> int:
    matrix = np.zeros((len(l_dims), len(l_dims)), dtype=int)

    for dif in range(1, len(l_dims)):
        for ini in range(len(l_dims)-dif-1):
            end = ini + dif
            cost = -1
            for j in range(ini, end):
                if cost == -1:
                    cost = matrix[ini][j] + matrix[j+1][end] + \
                        l_dims[ini] * l_dims[j+1] * l_dims[end+1]
```

```
            else:
                n_matrix = min([cost, matrix[ini][j] + matrix[j+1]
                        [end] + l_dims[ini] * l_dims[j+1] * l_dims[end+1]])
            matrix[ini][end] = cost
    return matrix
```

II-C. Cuestiones sobre las funciones de programación dinámica

1. El problema de encontrar la máxima subsecuencia común (no consecutiva) se confunde a veces con el de encontrar la máxima subcadena común consecutiva. Véase, por ejemplo, la entrada Longest common substring problem en Wikipedia. Describir un algoritmo de programación dinámica para encontrar esta subcadena común máxima consecutiva entre dos cadenas S y T, y aplicarlo “a mano” para encontrar la subcadena consecutiva común más larga entre las cadenas bahamas y bananas.

```
"""  Determina la cadena común y más larga de dos cadenas dadas.

       Argumentos:

          str_1(str): primera cadena
          str_2(str): segunda cadena

       Devuelve:

           La cadena o cadenas comunes más largas

       Autor:

           Eduardo
"""

def LongestCommonString(str_1: str, str_2: str) -> set:
    #asignaciones
    l1 = len(str_1)
    l2 = len(str_2)
    matrix = [[0]*(l2+1) for x in range(l1+1)]
    longest = 0
    lcs_set = set()

    for i in range(l1):
        for j in range(l2):
            if str_1[i] == str_2[j]:
                #incrementa el contador en la posición diagonalmente inferior
                c = matrix[i][j] + 1
                matrix[i+1][j+1] = c
                if c > longest:
                        #si el contador es mayor que longest (tamaño de la cadena más larga)
actualizamos el valor de longest
```

```
                lcs_set = set()
                longest = c
                #añadimos la subcadena resultante en el conjunto
                lcs_set.add(str_1[i-c+1:i+1])
            elif c == longest:
                #si el contador es igual que longest únicamente añadimos la subcadena
                lcs_set.add(str_1[i-c+1:i+1])
    #print(matrix)
    return lcs_set
```

2. Sabemos que encontrar el mínimo número de multiplicaciones numéricas necesarias para multiplicar una lista de N matrices tiene un coste O(N3) pero ahora queremos estimar dicho número v(N) de manera más precisa. Estimar en detalle suficiente dicho número mediante una expresión v(N) = f(N) + O(g(N)), con f y g funciones tales que |v(N) − f(N)| = O(g(N)).

De estas dos fórmulas podemos deducir que:

$V(N) \geq O(g(N)) \geq 0$

$V(N) \geq f(N)$

Y, por tanto, que:

$V(N) = O(f(N)) + O(g(N))$

Las cuestiones y el presente documento han sido realizados por Eduardo.