Practica 3: Quick Select. Programación Dinámica

Algoritmos y Estructura de Datos Avanzada 2022-2023

Pareja: 11

José Manuel García Giráldez

Alejandro Monterrubio Navarro

Índice

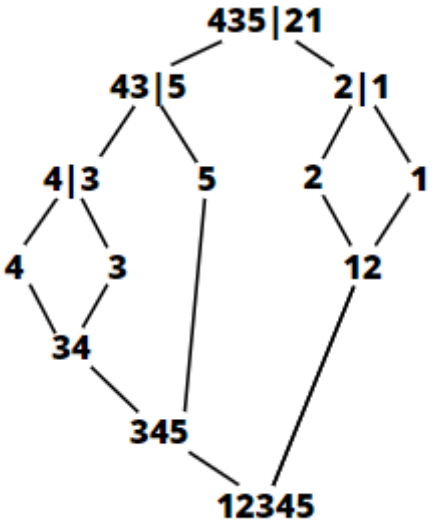
Cuestiones 1D:

Pregunta 1:

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

El proceso de MergeSort consiste en dividir la tabla de datos en subtablas hasta que cada subtabla tenga un solo elemento, y luego combinar las subtablas de manera ordenada.

Que el máximo número de comparaciones clave sea 8 se debe a que, para cada subtabla obtenida producto de la división se necesitan N-1 comparaciones. Veamos esto con un ejemplo:



Por lo tanto, se tiene que para 4 y 3 se realiza una comparación (34, N =2), para 34 con 5 se realizan 2(345, N=3), para 2 con 1 se realiza 1(12, N=1), para 345 con 12 se realizan 4(12345, N=3). De esta forma obtenemos un total de 8 comparaciones

Pregunta 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?

No, es imposible hacer menos de 5 comparaciones para encontrar la mediana de una tabla de 5 elementos con el algoritmo Quickselect. Esto se debe a que Quickselect se basa en la técnica de selección por pivote para encontrar la mediana de una tabla. Esto implica dividir la tabla en dos partes y comparar el pivote con cada elemento de la tabla para determinar en qué parte de la tabla se encuentra cada elemento. Para encontrar la mediana de una tabla de 5 elementos, es necesario hacer al menos 5 comparaciones para asegurarse de que se haya cubierto toda la tabla.

Pregunta 3:

¿Qué 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.

La complejidad del tiempo de ejecución del algoritmo QuickSort depende en gran medida de cómo se seleccione el pivote. Si el pivote se elige de manera aleatoria, la complejidad del tiempo de ejecución será de O(n log n) en el caso promedio y de O(n^2) en el peor caso. Sin embargo, si se utiliza una técnica de selección del pivote más sofisticada, como la "mediana de medianas de 5 elementos", es posible reducir la probabilidad de que el peor caso se produzca y, por lo tanto, obtener una complejidad del tiempo de ejecución de O(n log n) incluso en el peor de los casos, pues al emplear la mediana para obtener el pivote eliminamos la posibilidad de que una de las subtablas quedara vacía lo cual sería el peor caso para Quicksort.

Para justificar esto experimentalmente, podrías implementar la función qsort_5 y medir el tiempo de ejecución para distintos tamaños de la tabla t. Si observas que el tiempo de ejecución aumenta de manera lineal con el tamaño de la tabla, entonces esto sugiere que la complejidad del tiempo de ejecución es de O(n log n) incluso en el peor caso.

Analíticamente, podemos justificar la complejidad del tiempo de ejecución de O(n log n) en el peor caso de la siguiente manera: cuando se utiliza la técnica de "mediana de medianas de 5 elementos" para seleccionar el pivote, se garantiza que el pivote se encuentra en el rango del 25% inferior o superior de la tabla. Esto significa que, en el peor caso, el tamaño de la tabla se reduce a al menos el 75% en cada iteración del algoritmo. Si asumimos que el tamaño de la tabla se reduce a la mitad en cada iteración (lo que es el peor caso posible), entonces podemos demostrar que la complejidad del tiempo de ejecución es de O(n log n) utilizando un análisis de la recurrencia.

Cuestiones II-C:

Pregunta 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.

El enfoque consiste en construir una tabla de longitudes máximas de subcadenas comunes consecutivas para todos los pares de subcadenas S[i:] y T[j:], donde i y j son índices en S y T, respectivamente.

Podemos ver el pseudocódigo en la página de wikipedia:

```
function LCSubstr(S[1..r], T[1..n])
    L := array(1..r, 1..n)
    z := 0
    ret := {}

    for i := 1..r
        for j := 1..n
            if S[i] = T[j]
                if i = 1 or j = 1
                    L[i, j] := 1
                else
                    L[i, j] := L[i - 1, j - 1] + 1
                if L[i, j] > z
                    z := L[i, j]
                    ret := {S[i - z + 1..i]}
                else if L[i, j] = z
                    ret := ret ∪ {S[i - z + 1..i]}
            else
                L[i, j] := 0
    return ret
```

Construimos el código en base a esto en nuestra práctica:

```
if str_1 is None or str_2 is None:
    return None

if len(str_1) == 0 or len(str_2) == 0:
    return ""

# se obtienen las longitudes de las cadenas
len_1 = len(str_1)
len_2 = len(str_2)

# se inicializa la matriz
matrix = np.zeros((len_1+1, len_2+1))

# variable auxiliar para saber la maxima distancia
max = 0
ret = []
for i in range(1, len_1+1):
    for j in range(1, len_2+1):
        if str_1[i - 1] == str_2[j - 1]:
            if i == 1 or j == 1:
                matrix[i][j] = 1
            else:
                matrix[i][j] = matrix[i - 1][j - 1] + 1
            if matrix[i][j] > max:
                max = matrix[i][j]
                ret = [str_1[i - int(max): i]]
            elif matrix[i][j] == max:
                ret.append(str_1[i - int(max): i])
        else:
            matrix[i][j] = 0

# print(matrix)
return ret
```

La tabla se construiría de la siguiente manera:

Primero se inicializa a 0 la tabla.Se recorren las cadenas de forma que cando se encuentra una coincidencia entre dos caracteres, al encontrarse se suma 1 a la posición anterior en diagonal y se almacena en la posición actual, si el valor de la posición actual es mayor al anterior valor de max, se actualiza el valor máximo y se toma como subcadena la que componga ese valor. Si el valor obtenido es igual al máximo entonces añadimos el carácter a la subcadena.

Ya que solo se recorren las subcadenas una vez es más eficiente que tener que comprobar todas las posibles subcadenas y después comprobar sus longitudes.

Para aplicar este algoritmo "a mano" para encontrar la subcadena común máxima consecutiva entre las cadenas "bahamas" y "bananas", podemos seguir los siguientes pasos:

1. Inicializamos la tabla a 0

	b	a	n	a	n	a	s
b	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0

2. Se recorre y se va calculando los valores conforme al algoritmo, sumando 1 más la diagonal anterior en aquellas que presenten igual caracteres, para mejor visualización lo haremos por filas:

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0

Se puede ver que ya se colocó el primer valor mayor a 0, pues b = b, el resto de valores permanecen valiendo 0.

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	2	0	1	0	1	0
h	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

s	0	0	0	0	0	0	0

Como se puede observar hemos actualizado 3 valores, el valor 2 se debe a que se le suma 1 a la diagonal, cuyo valor era 1.

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	2	0	1	0	1	0
h	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0

Aquí no se hizo nada pues no hay caracteres iguales. Realizamos lo mencionado anteriormente para completar la tabla, las casillas en verde son aquellas modificadas en el paso actual y en amarillo si se usó una casilla anterior para calcular el valor:

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	2	0	1	0	1	0
h	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0
m	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	2	0	1	0	1	0
h	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0
m	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	2	0	1	0	1	0
h	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0

m	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0
s	0	0	0	0	0	0	0

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	2	0	1	0	1	0
h	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0
m	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0
s	0	0	0	0	0	0	2

De esta manera se obtiene finalmente la siguiente tabla:

	b	a	n	a	n	a	s
b	1	0	0	0	0	0	0
a	0	2	0	1	0	1	0
h	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0
m	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0
s	0	0	0	0	0	0	2

Podemos ver en rojo los valores máximos obtenidos, ambos valores son iguales y por tanto ambas son las posibles subcadenas consecutivas de mayor longitud, si recorremos en diagonal hacia atrás (camino en amarillo) vemos que las posibles subcadenas son “ba” y “as”.

Pregunta 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)).

Para calcular esto podemos tomar que el coste de la multiplicación es igual a

1*(n-1) + 2*(n-2) + …. + (N-1)*(N-(N-1)

Podemos resolver esto con sumatorios de la siguiente forma:

$$\sum_{i=1}^{N} \left(i \cdot (n-i)\right) = \sum_{i=1}^{N} \left(i \cdot n - i \cdot i\right)$$

Que se resuelve como:

$$\sum_{i=1}^{N} (i \cdot n - i \cdot i) = i \cdot \sum_{i=1}^{N} i - \sum_{i=1}^{N} \left(i^2\right)$$

Si resolvemos esto nos queda finalmente:

$$\frac{\left(N^3 - N\right)}{6} = \frac{N^3}{6} - \frac{N}{6}$$

Y finalmente: V(N)=

$$\lfloor f(N) + O\big(g(N)\big) = \frac{N^3}{6} + O(N)$$