David Marquez Mínguez

47319570Z

Ejercicios TEMA 4

ALGORITMIA Y COMPLEJIDAD

# EJERCICIO 3

En este problema, disponemos de un conjunto de billetes de distinto valor y se desea devolver la menor cantidad de billetes posibles en un determinado intercambio. Como cabría suponer, no disponemos de un numero ilimitado de billetes, sino que tenemos un conjunto de billetes determinados para dicho cambio.

El objetivo es diseñar un algoritmo para determinar cual es la menor cantidad de billetes a devolver, de los billetes disponibles que tenemos. Como siempre inicialmente veremos como funcionara nuestro algoritmo poniendo un ejemplo.

Inicialmente disponemos de una lista del valor de cada uno de los billetes. Además, disponemos de otra lista del mismo tamaño que nos indica la cantidad de billetes de cada valor.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Valor Billete | 1 | 2 | 3 | 4 | 5 | 6 | 7 | … |
| Cantidad Billetes | 1 | 3 | 2 | 1 | 0 | 0 | 0 | … |

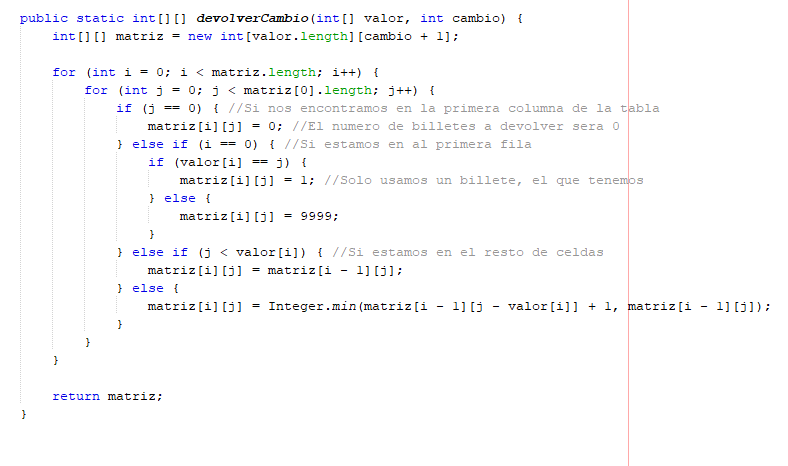
A continuación, vamos a poner un ejemplo para entender mejor el algoritmo. En la siguiente tabla, las filas representan cada uno de los billetes con su valor y la cantidad disponible y en las columnas, el cambio a devolver.

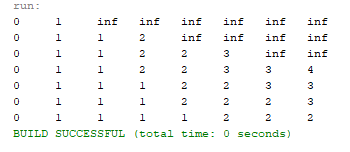
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CAMBIO | | | | | | | | | |
| BILLETES |  | 0$ | 1$ | 2$ | 3$ | 4$ | 5$ | 6$ | 7$ |
| Valor = 1 | 0 | 1 | - | - | - | - | - | - |
| Valor = 2 | 0 | 1 | 1 | 2 | - | - | - | - |
| Valor = 2 | 0 | 1 | 1 | 2 | 2 | 3 | - | - |
| Valor = 2 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 |
| Valor = 3 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 |
| Valor = 3 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 3 |
| Valor = 4 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |

La tabla se rellena de una forma muy sencilla. Pero antes de rellenarla, debemos saber lo siguiente: las celdas donde su valor es cero indican que el número de billetes a devolver es cero. Las celdas donde su valor es “-" indican que no hay un cambio disponible. En el resto de celdas se determinan el número mínimo de billetes a devolver con los tipos de billetes disponibles.

Para rellenar el resto de columnas, procedemos de la siguiente manera: si el valor del nuevo billete es menor que el cambio de esa columna, este billete no puede entrar en la solución, por lo que bastará con coger el valor de la fila anterior en esa misma columna. Por ejemplo, si el nuevo billete es de valor 4, y queremos cambiar un billetes de 3$, el nuevo billete no entra en la solución, por ello, cogemos el cambio de la fila anterior.

Por el contrario, Si el valor del billete es mayor o igual que el cambio de esa columna, tendremos que mirar ahora si nos conviene añadirlo a la solución o no. Para ello, restamos el valor del nuevo billete al valor del cambio en esa columna. Miramos ahora la columna cuyo valor coincida con ese resultado, en la fila superior a la actual. Si ese valor más uno es menor que el valor de la celda situada en la fila superior de nuestra propia columna entonces mejora nuestra solución y utilizamos este en nuestra celda. Si no, utilizamos el de la celda superior.

A continuación, procedemos a implementar el algoritmo en este caso en Java. He decidido implementar este algoritmo en Java puesto que es mucho más fácil trabajar con matrices que en Python.

A continuación, se muestra su salida por pantalla:

Como se puede observar, utilizando el ejemplo anterior obtenemos la salida esperada. Las celdas donde se muestra un “inf” son las celdas en las que no existe cambio disponible.

Una vez que ya conocemos el numero de billetes mínimos para devolver el cambio, ahora solo nos queda conocer cuáles son esos billetes, es decir, cual es el valor de los mismos. ¿Cómo vamos a saber que billetes hemos empleado una vez terminado el algoritmo?

Para conocer dichos billetes vamos a hacer un proceso de backtracking en el algoritmo, comenzando en la casilla en la que se resuelve nuestro problema. La forma de retroceder y, por tanto, saber los billetes usado se basa en el uso de los valores de los billetes. Vamos a trabajar sobre el mismo ejemplo que antes para indicar como volveríamos atrás para conocer los billetes utilizados.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CAMBIO | | | | | | | | | |
| BILLETES |  | 0$ | 1$ | 2$ | 3$ | 4$ | 5$ | 6$ | 7$ |
| Valor = 1 | 0 | 1 | - | - | - | - | - | - |
| Valor = 2 | 0 | 1 | 1 | 2 | - | - | - | - |
| Valor = 2 | 0 | 1 | 1 | 2 | 2 | 3 | - | - |
| Valor = 2 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 |
| Valor = 3 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 |
| Valor = 3 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 3 |
| Valor = 4 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |

Vamos a imaginar que deseamos dar un cambio de un billete de 7$ empleando los billetes disponibles. Como se ha indicado anteriormente en la tabla, emplearíamos dos billetes ¿pero que valor tienen dichos billetes? Para conocer dicho valor hacemos lo siguiente. Nos situamos en la última celda de todas, y comparamos su valor con el valor de la casilla superior. Como el valor es menor(2 > 3), sabemos que se ha utilizado el billete de valor 4, por ello:

*7 – 4 = 3*

Ahora nos situamos en la celda de valor = 3. En este celda, vemos que el valor de la celda superior es 1, es decir, el mismo valor, por lo tanto, sabemos que el valor de esta celda no forma parte de la solución. Por ello pasamos a la fila superior sin retroceder ninguna columna y sin contar ese billete.

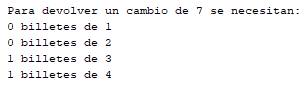
Tenemos ahora otro billete de valor 3, comparamos el valor de su celda con la fila superior y vemos que es menor(1 > 2), por lo tanto, sabemos que dicho billete forma parte de la solución, por ello:

*3 – 3 = 0*

Una vez que ya no queda más billetes por cambiar, sabemos que la solución esta compuesta por un billete de 4$ y otro billete de 3$.

A continuación, voy a implementar dicho algoritmo en Java. Como he dicho antes en esta ocasión se emplea Java pues es más cómodo trabajar con matrices. En este caso quiero trabaja con una tabla como la mostrada en el ejemplo, por ello me es más cómodo trabajar en Java que en Python.

En dicho algoritmo primero deberemos comprobar si es posible dar un cambio exacto. En caso de que se pueda, calcula de forma recursiva el numero de billetes de cada valor siguiendo el algoritmo que se ha descrito antes.

Aplicando el mismo ejemplo que antes obtenemos el siguiente resultado:

Como podemos observar, el resultado del programa coincide con lo que habíamos previsto, para devolver el cambio de un billete de 7$ necesitamos un billete de 3$ y otro billetes de 4$.

# EJERCICIO 7

En este problema, se pide trabajar con secuencias de bits, es decir, secuencias cuyos elementos pueden tomar los valores de 0 o 1. Una posible secuencia seria la siguiente:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nombre: A | 1 | 0 | 1 | 1 | 0 | 0 | 1 |

Como podemos observar la longitud de dicha secuencia, vamos a llamarla n, es 7, es decir:

*n = 7*

A partir de una secuencia como la mostrada anteriormente, podemos definir subsecuencias de la misma. Una subsecuencia de una secuencia dada es una secuencia donde se pueden eliminar elementos de la secuencia “padre”, pero nunca alterar su orden. Por ejemplo, algunas subsecuencias de la secuencia A, serían las siguientes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre: A1 | 1 | 0 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre: A2 | 0 | 1 | 1 | 1 |

…

Por otro lado, dadas dos secuencias A y B, podemos determinar una subsecuencia común a ambas. Suponemos que disponemos de las siguientes secuencias:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nombre: A | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nombre: B | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

Una subsecuencia común a ambas seria la siguiente:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre: AB1 | 1 | 1 | 0 | 1 |

El objetivo del problema es crear un algoritmo que nos determine las subsecuencias de dos secuencias dadas, empleando para ello algoritmos de programación dinámica.

Para buscar subsecuencias comunes creare una matriz, donde una de las secuencias compondrá las filas y la otra las columnas. A partir de ahí se rellenarán las celdas de la matriz con el número de subsecuencias de ambas. A continuación, se creará la matriz y se ira explicando cómo se rellenan.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | SUBSECUENCIA A | | | | | | | | |
| SUBSECUENCIA B |  | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
| 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |
| 0 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 |
| 0 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 |
| 1 | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 5 |
| 0 | 1 | 2 | 3 | 4 | 4 | 5 | 5 | 6 |
| 0 | 1 | 2 | 3 | 4 | 4 | 5 | 5 | 6 |
| 1 | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 6 |

Vamos a ir fila por filas comprobando las posibles secuencias de cada elemento. En la primera fila, como solo disponemos de un elemento, el máximo número de subconjuntos será 1.

Para la segunda fila, ahora no solo disponemos de un elemento, sino de dos(10). Procederemos igual que antes, pasando por las columnas buscando coincidencia de valores. Para asegurarnos de que hemos entendido bien el algoritmo, vamos a hacerlo poco a poco.

Estamos en la segunda fila y tenemos dos elementos 1 y 0. Ahora vamos pasando las columnas a comparamos valores.

|  |  |
| --- | --- |
| 0 | |
| 1 | 0 |

Si eliminamos el 1, podemos formar un subconjunto común, formado por ceros. Ahora pasamos a la segundo columna.

|  |  |
| --- | --- |
| 0 | 1 |
| 1 | 0 |

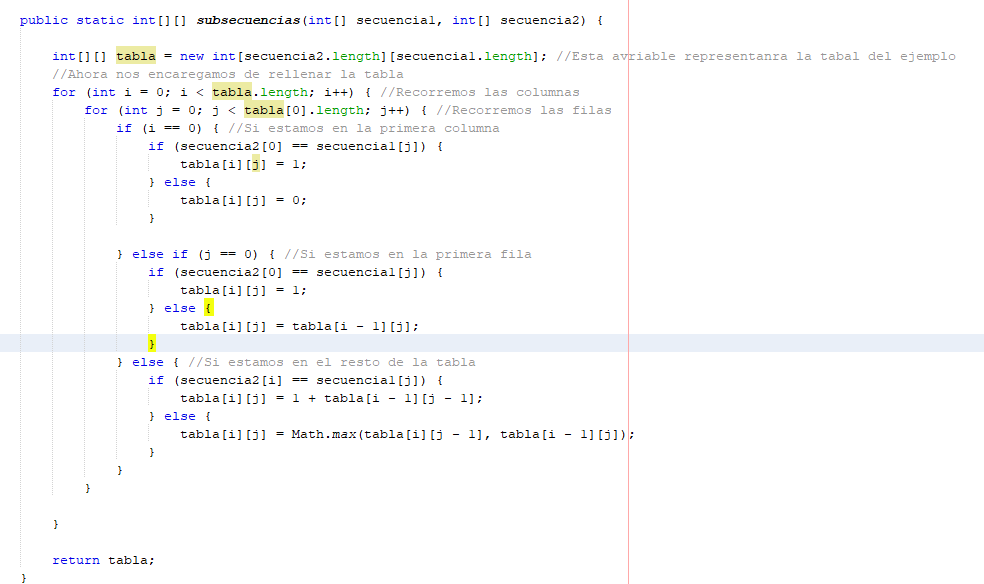
En este caso, solo podemos formar un subconjunto, el formado por ceros, o el formado por unos, pero no ambos. Ahora pasamos a la tercera columna.

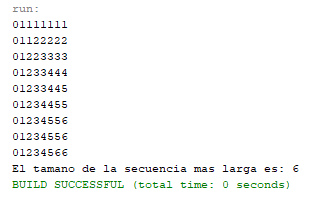
|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | | 1 |
| 1 | | 0 | |

En este caso ocurre como antes, solo podemos formar un subconjunto. Ahora pasamos a la cuarta columna.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 1 | 0 |
| 1 | | 0 | |

En este caso, si podemos formar un subconjunto y dos elementos, pues eliminando los dos primeros, podemos crear un subconjunto común a ambos. Una vez hemos encontrados un subconjunto de tamaño dos, paramos pues no podremos encontrar subconjuntos de mayor tamaño. En el resto de la tabla se trabaja de la misma forma.

Una vez terminada la matriz, como podemos ver la subsecuencia de mayor tamaño es de seis elementos. A continuación, voy a implementar dicho algoritmo nuevamente en Java. El objetivo es representar dicha tabla en una matriz de enteros y determinar así la subsecuencia más larga. En otras palabras, voy a plasmar lo que se ha hecho en el ejemplo, en Java.

Una vez implementado el algoritmo, voy a ejecutarlo poniendo los mismos datos que en el ejemplo de antes.

Como se puede comprobar, el algoritmo devuelve la tabla que hemos visto en el ejemplo y el tamaño de la cadena mas grande. Como se predijo, la cadena de mayor tamaño es de 6 elementos.

Ahora que ya sabemos la secuencia de mayor tamaño, debemos obtener una secuencia de dicho tamaño. Para resolver este problema, voy a hacer lo siguiente.

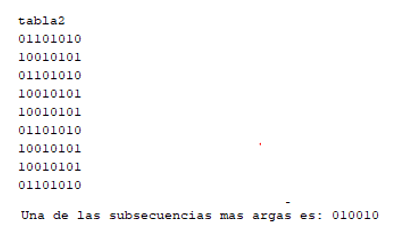
Voy a construir una tabla con todas las coincidencias de los elementos, es decir, es las cedas donde los elementos tengas el mismo valor, voy a colocar un 1, en las celdas donde el valor no sea el mismo colocare un 0. Por tanto, al tabla quedaría así:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | SUBSECUENCIA A | | | | | | | | |
| SUBSECUENCIA B |  | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

El objetivo ahora es buscar el mayor numero de unos seguidos, sabiendo que cuando encontramos un uno, debemos pasar a la siguiente fila y columna. Si encontramos un 0 pasamos a la siguiente columna simplemente. Las celdas marcadas en azul son las que me dan la secuencia más larga.

A continuación, se va a mostrar el algoritmo en Java, además, se dan los valores del ejemplo para observar que da el mismo resultado.



A continuación, se muestra la salida por pantalla. En primer lugar, se muestra la tabla que se ha descrito anteriormente y en segundo lugar una de las subsecuencias de mayor tamaño. Como podemos observar, coincide con la salida esperada.