ALGORITMIA

Grado en Ingeniería Informática del Software - Curso 2018-2019

GUIÓN DE LA PRÁCTICA 5

OBJETIVO:

- **Resolución de problemas con algoritmos basados en programación dinámica**

1. Tasación de Collares

Una importante joyera quiere maximizar el beneficio de los collares, brazaletes o cualquier otra joya que compra. Para ello tenemos que desarrollar un algoritmo que obtenga un valor de tasación aproximado para esas joyas.

Para ello disponemos de la siguiente información:

- Un collar (o cualquier otra pieza de joyería) se representa por una secuencia de piedras preciosas o gemas. El proceso de tasación consiste en hacer divisiones de gemas, tantas como queramos. Cada subdivisión de gemas podrá venderse por separado, obteniéndose así un beneficio.
- Una tabla con los valores de las gemas. Dado que algunas combinaciones de gemas son más bonitas y/o apreciadas, la tabla de gemas contendrá gemas individuales o secuencias de gemas.

Ejempo de tasación:

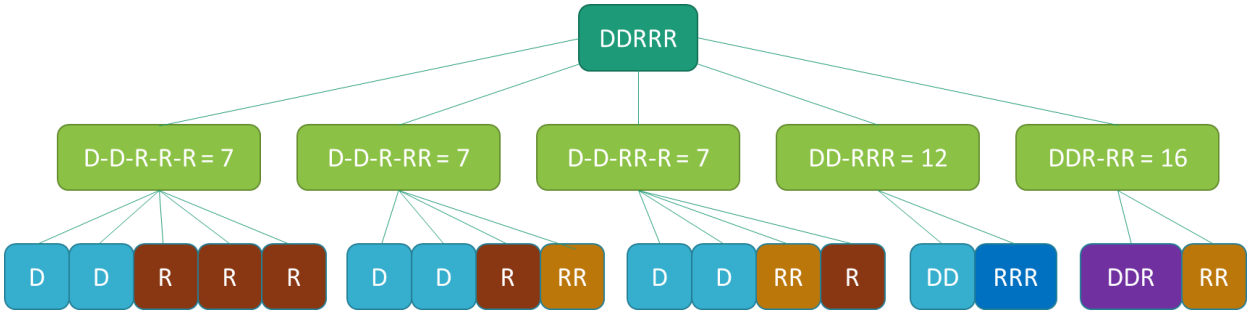
Un `Necklace` (collar en inglés) se codificará como una secuencia de letras. La misma letra representa gemas de calidad similar y por lo tanto todas tendrán el mismo valor. Otros parámetros como tipo de gema, tamaño, color o cualquier otro podría representarse con diferentes letras. Esto representaría diferentes categorías y sería necesario una nueva entrada en la tabla de valores.

Dados los siguientes datos:

- Collar: `DDRRR`
- Valores:

Secuencia	Valor
Diamante (D)	2
Rubí (R)	1
RR	3
DD	5
DDD	8
RRR	7
DDR	14

El número de subdivisiones posibles dependerá de la longitud del collar y del número de secuencias en la tabla de valores. En la figura siguiente se muestran algunas de las posibles subdivisiones, así como sus valores de este ejemplo.



Para implementar este algoritmo podríamos por ejemplo utilizar un enfoque Divide y Vencerás. El collar original se divide en dos o más problemas menores. Sin embargo, la solución obtenida podría no ser óptima. En este ejemplo si nuestro algoritmo D&V divide la cadena `DDRRR` como `DD-RRR` podría estar obviando la solución óptima `DDR-RR` de la figura anterior.
Otra alternativa podría ser utilizar Backtracking para generar todas las posibles subdivisiones de collares. En este caso la principal desventaja (que también podría suceder en un algoritmo D&V) es que algunos sub-problemas podrían repetirse, y se estarían resolviendo múltiples veces.

2. Programación Dinámica

En este caso vamos a implementar un algoritmo basado en Programación Dinámica. La principal idea es crear una tabla de caché que rellenaremos con la siguiente información:

- En columnas: Tenemos números desde 0 hasta la longitud de collar. Esto representará la gema actual que estamos teniendo en cuenta en el proceso de tasación.
- En filas: La lista de subdivisiones de gemas.
- En cada celda de la tabla estableceremos el valor de tasación del collar. Éste se obtendrá aplicando las divisiones de gemas más ventajosas posibles.

Ejemplo de ejecución: **`Necklace = RDDRR`**

Al principio del algoritmo, la tabla está vacía. Lo primero que hacemos es rellenar la columna '`0`'. Ésta representa el valor de un collar sin gemas, y por lo tanto, tendrá `valor = 0`.

	0	1	2	3	4	5
D	0					
R	0					
DD	0					
RR	0					
DDD	0					
RRR	0					
DDR	0					

En el siguiente paso se rellenará la columna '`1`'. Esta representa un collar con un solo Rubí (1).

	0	1 (R)	2 (D)	3 (D)	4 (R)	5 (R)
D	0	0				
R	0	1				
DD	0	1				
RR	0	1				
DDD	0	1				
RRR	0	1				
DDR	0	1				

En este caso, la primera regla a aplicar es '`D`'. Dado que la primera gema es una '`R`', esta sustitución no se puede aplicar (no se puede extraer un Diamante) y se actualiza la tabla con el valor '`0`'.
La siguiente regla a aplicar es la 'R'. Dado que coincide con el elemento actual, el valor de tasación se actualiza con el valor de esa subdivisión

Aunque los primeros pasos del algoritmo son sencillos, este se vuelve un poco más complicado cuando tratamos de aplicar subdivisiones de gemas más largas.

	0	1 (R)	2 (D)	3 (D)	4 (R)	5 (R)
D	0	0	3	5		
R	0	1	3	5		
DD	0	1	3	6		
RR	0	1	3			
DDD	0	1	3			
RRR	0	1	3			
DDR	0	1	3			

Esta tabla muestra el estado de la tabla cuando se analiza el tercer símbolo de collar. En primer lugar, la regla '`D`' se aplica. Esto se puede entender fácilmente como extraer otro Diamante, lo que a su vez incrementa el valor del collar de 3 a 5. Sin embargo, nos hemos dado cuenta de que la secuencia '`DD`' tiene un valor de 5. En este caso, el valor de tasación final se debe calcular como el máximo entre, el valor de tasación del collar sin considerar las dos últimas gemas (cuadro verde oscuro) más el valor de la división de dos diamantes, y el valor de tasación máximo obtenido hasta ahora (celda roja). En este caso `max(5, 1 + 5) = 6`, por lo que necesitamos actualizar la tabla apropiadamente (celda amarilla).

Finalmente, para el collar `RDDRR` obtendríamos la siguiente tabla:

	0	1 (R)	2 (D)	3 (D)	4 (R)	5 (R)
D	0	0	3	5	6	15
R	0	1	3	5	7	16
DD	0	1	3	6	7	16
RR	0	1	3	6	7	16
DDD	0	1	3	6	7	16
RRR	0	1	3	6	7	16
DDR	0	1	3	6	15	16

Reconstrucción de la solución

Una vez se ha rellenado toda la tabla, es posible obtener que subdivisiones de gemas se han realizado, para así reconstruir la solución obtenida. Este algoritmo es similar al utilizado en el algoritmo de la mochila y consiste en los siguientes pasos:

1. Comenzando en la celda de la última fila y última columna.
2. Recorrer las celdas en sentido contrario a como fueron calculadas:
    a. Tomando la última columna, desde la última fila hasta la primera.
    b. Cuando se termine una columna, comenzar por el último elemento de la columna anterior.
3. Buscar celdas contiguas (según el recorrido descrito) donde hay un cambio del valor de tasación calculado.
4. Añadir al conjunto solución la subdivisión correspondiente a la celda con mayor valor de tasación
5. Desplazarse tantas columnas a la izquierda como la longitud de la subdivisión aplicada.
6. Continuar el recorrido desde el último elemento de la nueva columna (paso 3 y siguientes).

3. TRABAJO A REALIZAR

Tarea 1

Escribe un programa `Necklace` para resolver este problema utilizando Programación Dinámica. Se recomienda utilizar el código proporcionado, aunque no es obligatorio. Contiene las siguientes clases auxiliares:

- `DynamicProgrammingCache`: Es una clase abstracta que define la interfaz básica para una tabla bidimensional para problemas de programación dinámica. Está diseñada para usarse con una colección de etiquetas y una colección de pesos, pero puedes adaptarla a tus necesidades si lo consideras necesario.
- `KnapsackCache`: En ella se implementa el problema de la Mochila 0/1 resuelto con programación dinámica. Esta clase hereda de `DynamicProgrammingCache` e implementa los métodos necesarios para funcionar correctamente.

- `KnapsackDemo`: Contiene un pequeño programa de ejemplo resolviendo el problema de la mochila 0/1. Puedes utilizarlo para ver cómo funciona nuestra implementación de la mochila 0/1.
- `NecklaceFactory`: En esta clase hay algunos métodos de utilidad que puedes utilizar para probar tu algoritmo. Alguno de esos métodos no está implementado. Puedes utilizarla, extenderla, modificarla, o simplemente ignorarla.
- `out.txt`: Contiene el resultado de dos ejecuciones de ejemplo. Puedes utilizarlo para verificar el funcionamiento de tu algoritmo.

`Necklace` debe incluir un método para obtener un valor de tasacción para un collar (método `appraisal`) y otro para obtener la lista de subdivisiones aplicadas (método `getBestElements`). Además, debes incluir una clase de prueba o JUnit para verificar el funcionamiento.
Analiza (y documenta) las complejidades temporales de ambos métodos.

Tarea 2

Construye la clase `NecklaceTiempos` para completar una tabla con mediciones de tiempos. Debes implementar un mecanismo aleatorio para generar collares y tablas con subdivisiones y sus valores. Utiliza N como la longitud del collar. En este caso no tienes que obtener las subdivisiones aplicadas.

N	*tiempo*
10000	…..
20000	…..
40000	…..
80000	…..
160000	…..
320000	…..
640000	…..
1280000	…..
…..	…..
Hasta "heap overflow"	

En el PDF debes incluir la tabla anterior junto con una breve explicación de las complejidades (metodos `appraisal` and `getBestElements`).

*Si utiliza **Eclipse**, se creará el proyecto **prac05_Pd<UOpropio>** con todas las clases necesarias.*
*Si utiliza **JDK**, cree los paquetes necesarios para esta práctica.*
*Las clases necesarias se crearán dentro del paquete **algdnipropio.p5***

Se entregará:

- *El proyecto Eclipse con los ficheros fuente de las clases, que se hayan programado.*
- *Un documento PDF con una pequeña explicación de los algoritmos utilizados en ambos programas y su complejidad.*

Se habilitará una tarea en el campus virtual para realizar la entrega. El plazo de entrega será hasta 1 día antes de la próxima sesión de prácticas.