BACTRACKING ESTUDIO

OBJETIVO

Obtener todas las combinaciones posibles de N vacas y quedarnos con aquellas que cumplan los requisitos de leche y espacio definidos.

En los apartados B y C se ha decidido usar una clase dedicada, Registro, a contar soluciones y guardarnos la mejor. Esto es para evitar que tengamos que explorar todas las opciones dos veces ya que lo único que cambia entre estos dos apartados es llevar una cuenta en el B, donde la iríamos devolviendo, para el C, llevaríamos otro objeto solución que representase la mejor, la cual también deberíamos devolver.

Por esto, se ha decidido crear la clase Registro, quien guarda la cuenta de las soluciones y el límite de leche, para el apartado B, la mejor solución encontrada para el apartado C. El único método de interés que tiene es contemplarSolución, con dos ifs, uno para el apartado B y otro para el C respectivamente.

COMPLEJIDAD

Se asumirá que las funciones no mencionadas tienen una complejidad constante, al igual que las operaciones básicas. En casos de bifurcaciones, tomaremos el camino más costoso.

La lectura de datos es igual que en ejercicio de voraces, es decir O(N) donde N es el número de líneas del fichero, que coincide con el número de vacas.

Las clases de solución tienen métodos de complejidad O(1) excepto el toString y la copia de la solución, que realizan tantas iteraciones como vacas seleccionadas (M) haya en el caso de la clase OrientadoSolución, por lo que tiene complejidad de O(M). Por otra parte la Solución (orientada a elementos) ejecuta tantas iteraciones como vacas disponibles haya(N), alcanzando una complejidad de O(N). Cabe destacar que $N \geq M$.

La clase registro tiene una llamada a la copia de Solución en el método contemplarSolución, por lo que su complejidad es O(N).

Tomaremos N como la longitud del array de vacas, las vacas disponibles, y $etapa \in [0;N]$.

En la clase Main, las primeras llamadas tienen el mismo coste que las llamadas recursivas, que para cada apartado es:

APARTADO A

DECISIÓN DE LA ETAPA

La etapa coincide con la profundidad de la pila de llamadas.

De las vacas que nos quedan disponibles sin haber seleccionado, ya que no se permiten repeticiones, seleccionaremos aquella que va a pasar a pertenecer a la solución. Supuesto que ya hemos decidido otras vacas que pertenecen a la solución.

Comprobación De Finalización

Tendremos una solución cuando el límite de leche se alcance, aunque esta se puede mejorar añadiendo más vacas.

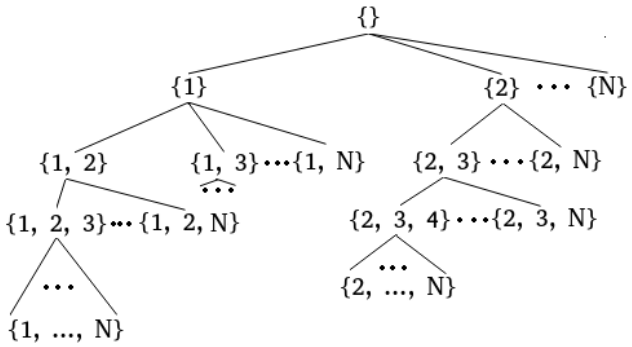
Test De Fracaso

Una vaca no pertenecerá a la solución si al añadirla superamos el límite de espacio. Tampoco la añadiremos si esta ya pertenece a la solución.

Generación

En cada etapa, podemos seleccionar cualquier vaca de las disponibles para que pase a pertenecer a la solución.

Complejidad



El código se ejecuta de esta forma, en profundidad. Representado con un árbol general, cada nodo es un posible subconjunto a coger del conjunto de N elementos original. La profundidad representa el número de elementos del subconjunto.

Se harán tantas llamadas como nodos existan en este árbol (o menos por la poda del límite del espacio). Y en cada llamada recorremos, al hacer la copia, todos los elementos seleccionados, es decir, para el nodo raíz, no recorremos, para los hijos del nodo raíz, debemos hacer una iteración en la lista, para los de nivel 2, 2 iteraciones, y para el único nodo con N elementos ({1, ..., N,}), haremos N iteraciones.

Por tanto, para contar ciclos, contaremos cuántos nodos hay de M elementos y multiplicaremos esa cifra por el número de iteraciones, M. Esto es:

$$\sum_{M=0}^{N} M\binom{N}{M} = 2^N N$$

Apartado B y C

Decisión De La Etapa

Intentamos seleccionar la vaca del vector. Decidimos si nos quedamos con la vaca de esa etapa. Supuesto que ya hemos decidido sobre las anteriores.

Comprobación De Finalización

Cuando hemos decidido sobre todas las vacas, si cumplimos el límite de leche, tenemos una solución.

Test De Fracaso

Se puede escoger una vaca, si queda espacio disponible. En caso contrario eliminamos la rama donde cogemos a esa vaca.

Generación

Seleccionamos la vaca o no la seleccionamos.

Optimización

Nos quedamos con aquellas soluciones que tienen una mayor producción de leche.

Complejidad

Al igual que para el apartado A, podríamos verlo como un árbol, en este caso sería uno binario con profundidad N, que coresponde a la etapa y donde en cada nodo representa la selección de una vaca habiendo decidido sobre las anteriores. En los nodos internos se no habría bucles, pero al llegar a las hojas, se haría la copia, ejecutando N iteraciones.

Buscaremos otra forma de razonarlo para no repetir el primer apartado.

Para el caso base, cuando la etapa alcanza la N, llamamos a contemplarSolución, que debe recorrer un array de N elementos. $etapa \in [0, N]$

Para el resto de los casos, se realizan 2 llamadas donde en cada una se reduce el problema en 1 unidad y todas las operaciones realizadas cuestan un tiempo constante, tenemos una ecuación de recurrencia del estilo:

$$T(etapa) = \begin{cases} N, etapa = N \\ 1 + 2\,T(etapa + 1), etapa < N \end{cases}$$

Lo cual expandido quedaría:

$$T(0) = 1 + 2\Big(1 + 2\big(1 + \ldots + 2(N)\big)\Big)$$

Si repartimos los 2 que multiplican, obtendremos:

$$1 + 2 + 4\big(1 + \ldots + 2(N)\big) = 1 + 2 + 4 + 8 + \ldots + 2^{N-1} + 2^N N$$

Aquí, podemos identificar 2 partes, la suma de las potencias de dos y el sumando con fator N. Intentaremos simplificar la suma de potencias de dos:

$$\sum_{i=0}^{N-1} 2^i = 1 + 2 + 4 + 8 + \ldots + 2^{N-1} = 2^N - 1$$

Al final, obtenemos:

$$T(0) = 2^N - 1 + 2^N N$$

El sumando de mayor peso es $2^N N$, por lo que tomaremos este como complejidad. Como vemos, la primera forma y la segunda nos dan la misma complejidad.

El main llama a la lectura de vacas, el apartado A, y se ha encontrado solución, el apartado B y C. En total, la complejidad sería:

$$O(n)+O(2^N N)+O(2^N N)$$

Que se reduciría a:

$$O(2^N N)$$

ACELERACIÓN DE LA CONVERGENCIA DEL ALGORITMO

A) Sólo buscamos una solución.

En este caso debemos llegar a un mínimo de leche. Para ello, primero se debe cumplir que la suma de la producción de todas las vacas sea mayor o igual que la cantidad deseada.

Suponiendo que esto ocurre, la forma de mejorar el tiempo sería que una vez tenemos la cantidad deseada, parar la ejecución. De la forma hecha, podemos parar la ejecución sin tener que llegar a decidir al menos una vez sobre todas las vacas, como nos obligaría el apartado B y C. Esto mejora la solución si es que la encontramos muy pronto, por ejemplo, cuando el límite de mínimo de leche sea despreciable. De todas formas, la mejora es despreciable.

B) Buscamos mostrar el número de las posibles soluciones que cumplan los límites.

La única posible aceleración es, una vez vemos que añadir una vaca hace superar el límite de espacio, cortar esa rama. Esto hará que siempre cumplamos el requisito de espacio. Esto es lo que ya se hace, que diferencia al algoritmo de uno de fuerza bruta.

C) Queremos la mejor solución.

Para este, la mejora sería que la comparación y copia de soluciones cueste lo menos posible. Lo cual se lleva a cabo haciendo que cada solución vaya calculando qué tan buena es al añadir o quitar la vaca, quitando la necesidad de hacer un bucle al final.

INTRATABILIDAD

La complejidad obtenida está entre las peores. Para el conjunto de vacas dado (30), ya tenemos un número absurdamente grande 3.2×10^{10} o 32 mil Millones de ciclos.

Cada vez que añadimos una vaca hacemos más que duplicar el tiempo. Esta forma de resolver el problema hace que no podamos tratar conjuntos de datos muy grandes ya que tardamos un tiempo exponencial en resolverlos.

AUTORES

- Noelia Díaz-Alejo Alejo

- Samuel Espejo Gil