ESTUDIO VORACES

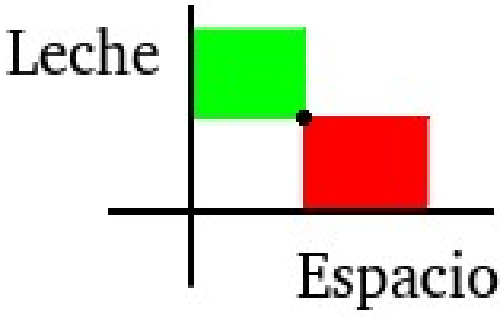
OPTIMALIDAD DE LA SOLUCIÓN

En ambos casos, independientemente de a la respuesta que alcancemos, estamos sujetos a los problemas que trae el problema de la mochila donde no se puede fraccionar, ya que al ser un algoritmo voraz, si cumplimos las restricciones, tomaremos el elemento, independientemente de si ello nos lleva a la mejor solución o no. Es decir, para asegurar la solución óptima, tendríamos que ver todas las combinaciones para poder saber si existe una solución mejor.

APARTADO A

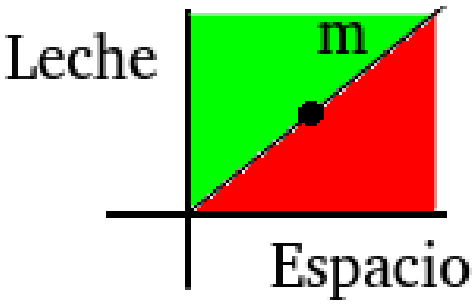
El objetivo es maximizar la producción de leche teniendo como limitante el espacio. Es decir, queremos aquellas vacas que más leche den y menos espacio ocupen. Una primera idea sería seguir este criterio estrictamente.

Para verlo gráficamente, podríamos representar 2 ejes, Espacio y Leche donde cada punto sería una vaca:



Dado el dibujo, nos quedaríamos con aquellas que están arriba a la izquierda, en la zona verde. El problema de esto es que tenemos 2 zonas, abajo izquierda y arriba derecha, las blancas, donde no sabríamos cuál sería mejor, estas vacas tendrían el mismo valor.

Para resolver el empate, podemos usar otra idea, calcular un ratio beneficio frente a coste. El beneficio sería la producción de leche y el coste, el espacio. La división quedaría:

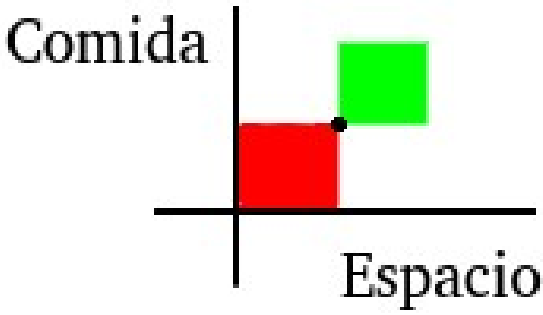


Donde m es la pendiente de la recta entre el origen y la vaca. Como vemos, esto sí divide el plano en 2 regiones, además coincidentes con la primera división. Al dividir sólo en 2 áreas es más óptimo que el primer caso. En este caso, las vacas con el mismo valor son aquellas que están en la misma recta.

La pendiente de la recta es dependiente de cada vaca o valor asignado a esa vaca, se priorizan aquellas de mayor valor, es decir, las de mayor pendiente.

APARTADO B

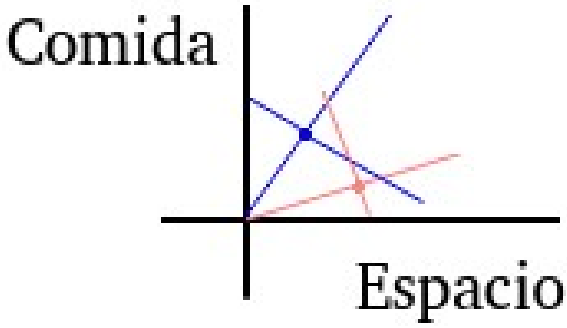
En este caso buscamos minimizar el consumo de comida con el limitante del espacio. Siguiendo la representación gráfica, ahora podemos representar las vacas en los ejes Comida y Espacio, donde si seguimos el criterio de elección dado, veremos un problema como en el primer caso:



Como vemos, ahora las áreas están al revés por lo que si hacemos el ratio veremos que nos equivocamos, ahora no tenemos un beneficio, todo son gastos. Para arreglar la situación tenemos 2 posibilidades:

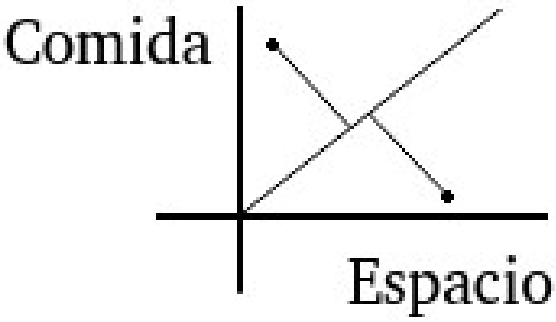
1. Buscar una recta.

   Al igual que en el primer caso, podríamos buscar una recta que nos divida el espacio como en el primer apartado. Una posibilidad sería tomar la perpendicular a la recta del primer caso y tomaríamos aquellas que estén a la arriba a la derecha, pero esto trae un problema:



   Dado este caso, si tomamos como referencia la azul, vemos que ella piensa que tiene preferencia ya que la rosa está por debajo a la izquierda de su perpendicular. Pero si tomamos como referencia la rosa vemos que ocurre lo mismo, hay ambigüedad.

   Tendríamos que encontrar una recta que nos sirva para todas las vacas. Esto podría hacerse mediante una media de todas las vacas, pero esto trae otros problemas:

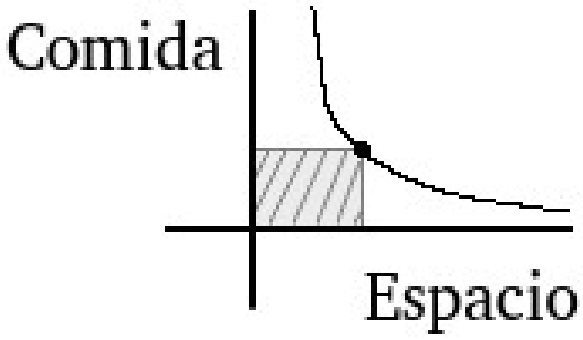


Cuanto más dispersas sean las vacas, peor será la aproximación que usemos.

No parece que podamos encontrar una recta que nos sirva. De todas formas, cabe mencionar que esta idea se podría interpretar como una suma pesada, donde el resultado es el coste, y los sumandos son pares Parámetro × Coste del Parámetro, y los posibles pármetros son espacio y coste. La relación entre los costes de los parámetros sería la pendiente de la recta que pasa por el origen. Estaríamos intentando darle un mismo valor a ambos parámetros.

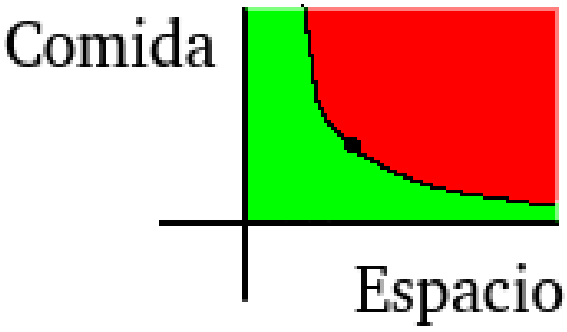
2. Buscar otra solución.

   Tenemos que solucionar el desempate visto en la primera solución propuesta. La idea que exploraremos es usar áreas, calculadas como Comida × Espacio, de esta forma obtenemos:



   Con este parámetro resolvemos el empate. Aquellas vacas que tengan menor área se priorizarán.

   La línea representada divide el espacio al igual que antes de esta forma:



   Al igual que antes, aquellas vacas que coincidan en la línea tienen el mismo coste. Se escoge las que estén por debajo a la izquierda de la línea.

   La línea depende de cada vaca o coste asignado a esa vaca, se priorizan aquellas de menor coste, es decir, las de menor área.

CÁLCULO DE COMPLEJIDAD

Bajo la suposición de que todos los métodos y operaciones que no se razonen tienen un coste constante.

Todos los métodos de la carpeta vaca tienen una complejidad constante O(1) ya que no cuentan con bucles.

El main cuenta con diversas partes: la llamada a la lectura de datos y las simulaciones.

La lectura del fichero de datos (Fichero.java) tiene un bucle que se repite tantas veces como líneas tenga el fichero. Complejidad O(n), donde n es el número de líneas del fichero, que si no hay errores, coincide con el número de vacas disponibles.

Las simulaciones están representadas por un bucle for, tantas iteraciones como simulaciones queramos hacer, en este caso 2, una por cada apartado y atendiendo a un criterio de ordenación diferente. Para cada simulación se hace:

1. Ordenación de las vacas, la llamada a quickSortC, este algoritmo tiene una complejidad de $O(n \times \log(n))$ en el caso promedio.
2. Ejecución de run, que tiene un bucle while que como mucho hará tantas iteraciones como vacas disponibles haya. Complejidad es O(n), donde n es el número de vacas disponibles.
3. Muestra de resultados, el toString de la clase Solución, cuenta con un bucle que recorre todas las diferentes vacas seleccionadas. La complejidad es O(m), donde m es el número de vacas seleccionadas, que en el peor de los casos es igual que el número de vacas dispoibles, en otro caso, es menor.

Por tanto, en total se quedaría:

$$O(n) + 2\ (O(n \times \log(n)) + O(n) + O(\min(n, m)))$$

Que se reduciría a:

$$O(n \times \log(n))$$

Esta complejidad viene por el algoritmo de ordenación.

AUTORES

- Noelia Díaz-Alejo Alejo
- Samuel Espejo Gil