Diseño y Análisis de Algoritmos. Problema 1: El Zoológico

Jesús Santos Capote y Kenny Villalobos Morales

Facultad de Matemática y Computación, Universidad de La Habana, La Habana, Cuba

1. Definición del Problema

En un zoológico un poco especial, se separa a los animales en dos hábitats diferentes de manera general. El hábitat para la reproducción solo acepta animales de la misma especie, mientras que el hábitat para la maduración admite animales de distintas especies, pero no de géneros distintos. El zoológico está pasando por una remodelación, ya que va a recibir n especies distintas de animales, cada una con un número 'a' de hembras y 'b' de machos, que puede ser distinto entre cada especie. En la remodelación se está pensando en construir salas de exhibición de los animales. Cada una se construirá para ser un hábitat de reproducción o de maduración, y cada sala podrá soportar un máximo de k animales, lo cual es igual para todas las salas. Como el zoológico necesita ser rentable y cada sala se cobra por separado, se quiere construir el máximo número posible de salas que cumplan con los requerimientos planteados y que, además, estén llenas de animales para el disfrute de los visitantes. ¿Cuántas salas deberá construir el zoológico?

2. Modelación del Problema

Se tiene una matriz de dimensión $2 * n$, donde cada casilla contiene números enteros. El problema consiste en hallar la cantidad máxima de grupos de tamaño k que se pueden formar, donde cada elemento de un grupo es una unidad perteneciente a una casilla de la matriz. Además, todos los elementos de un grupo deben pertenecer a casillas de una misma fila o de una misma columna de la la matriz.

3. Primera Aproximación

Como primera solución se propone aplanar la matriz de entrada para obtener una lista de T elementos, siendo T la suma de todos los elementos de la matriz, donde en cada posición hay un animal. Los animales son modelados como una clase *Animal* que tiene dos atributos: *gender*, fila a la que pertenece en la matriz (género), y *specie*, columna a la que pertenece en la matriz (especie). Luego generar todas las permutaciones de la matriz aplanada y por cada una dividirla mediante índices en porciones de tamaño k, contar cuantas de estas porciones son grupos válidos y actualizar, de ser necesario, el máximo;

3.1. Optimalidad

Sea $S = [a_1, a_2, \ldots, a_T]$ una distribución de los animales de la matriz aplanada, tal que la cantidad de porciones de tamaño k que se pueden formar a partir de ella, que cumplen las restricciones del problema, sea máxima. Esta distribución S es una permutación de los elementos de la matriz aplanada. El algoritmo propuesto revisa todas las permutaciones de la matriz aplanada y calcula la cantidad de grupos factibles, por tanto, revisa a S. Luego el algoritmo encuentra la respuesta óptima.

3.2. Complejidad Temporal

Aplanar la matriz tiene costo $O(T)$ donde T es la cantidad de animales. Computar todas las permutaciones de una lista de tamaño T tiene un costo $O(T!)$. La comprobación de cuántos grupos factibles contiene una permutación de T elementos tiene complejidad $O(T)$ y esta comprobación se realiza para cada permutación. Luego por el teorema de la multiplicación contar cuantos grupos factibles tiene cada permutación tiene complejidad $O(T*T!)$. Luego el algoritmo tiene complejidad $O(T+T*T!+T)$ y por el teorema de la suma, la complejidad final es $O(T*T!)$.

4. Segunda Aproximación

Demostremos que existe un óptimo que solo usa un grupo columna a lo sumo por columna:

Sea a_i y b_i los elementos de la columna i. Sea $a_i + b_i = c$.

$$c = p_c * k + r_c, \;\; r_c < k$$

p_c, es la cantidad de grupos columna que se pueden formar en la columna i.

$$a_i = p_a * k + r_a, \;\; r_a < k$$

p_a, es la cantidad de grupos fila que se pueden formar solo con los elementos de a_i.

$$b_i = p_b * k + r_b, \;\; r_b < k$$

p_b, es la cantidad de grupos fila que se pueden formar solo con los elementos de b_i. Por tanto:

$$c = a_i + b_i = p_a * k + p_b * k + r_a + r_b, \;\; r_a < k, r_b < k$$

$$c = (p_a + p_b) * k + r_a + r_b$$

con $r_a + r_b < 2k$.

Si $r_a + r_b < k$ entonces por el algoritmo de división, la cantidad de grupos que se pueden formar en la columna i, $\lfloor \frac{c}{k} \rfloor$, es igual a la suma de p_a (cantidad de grupos que se pueden formar solo con los elementos de a_i) mas p_b (cantidad de grupos que se pueden formar solo con los elementos de b_i). Es decir, para este caso, con formar solo grupos filas con los elementos de a_i y b_i se iguala la cantidad de grupos columnas que se pueden formar en la columna i.

Si $k \leq r_a + r_b < 2k$ entonces $r_a + r_b = r' + k$ con $r' < k$, luego:

$$c = (p_a + p_b) * k + r' + k$$

$$c = (p_a + p_b + 1) * k + r', \;\; r' < k$$

Esto significa que, por el algoritmo de división, la cantidad de grupos que se pueden formar en la columna i, $\lfloor \frac{c}{k} \rfloor$, es igual a la suma de p_a (cantidad de grupos que se pueden formar solo con los elementos de a_i) mas p_b (cantidad de grupos que se pueden formar solo con los elementos de b_i) mas uno. Es decir, la cantdad de grupos p_c que se pueden formar en una columna es igual a la cantidad $p_a + p_b + 1$ a lo sumo, con formar solo grupo filas con los elementos de a_i y b_i obtengo la cantidad máxima de grupos menos uno.

Por lo dicho anteriormente existe un óptimo que tiene a lo sumo un grupo con elementos pertenecientes a una misma columna i para cada i.

4.1. Explicación del Algoritmo

Partiendo de que solo se necesita tomar un grupo columna por columna a lo sumo y tomar el resto como grupos fila, el algoritmo para cada forma de seleccionar columnas prueba todas las formas de crear un grupo columna en cada columna seleccionada. Luego de tener un conjunto de columnas seleccionadas y una forma de crear grupos columna fijada se calcula la cantidad de grupos filas que se pueden formar y se suma a la cantidad de grupos columnas que se crearon. El retorno del algoritmo sera el mayor número computado de esta forma, que corresponde al número máximo de grupos que se pueden crear cumpliendo las restricciones del problema.

4.2. Complejidad Temporal

Computar todas las formas de seleccioanar columnas es $O(2^n)$, siendo n la cantidad de columnas. Sea C la cantidad de formas computadas. Todas las formas de crear grupos columna por cada columna son computadas utilizando un backtrack, cuya expresión de $T(n)$ es la siguiente:

$$T(n-1) = T(n-1) + O(k)$$

Por el método del árbol, el backtrack tiene complejidad $O(nk)$. Luego por cada forma de seleccionar columnas se ejecuta el backtrack, que por el teorema de la multiplicación este computo tiene complejdad $O(Cnk)$. Luego el algoritmo tiene complejidad temporal $O(2^n + Cnk)$

5. Tercera Aproximación

Sea $M = \sum_{i=0}^{n} matriz[0, i]$, $F = \sum_{i=0}^{n} matriz[1, i]$ y $T = M + F$.

Sabemos que la cantidad máxima de grupos que se pueden formar es $\lfloor \frac{T}{k} \rfloor$. Para toda posible entrada de datos del problema, la solución es mayor o igual que $\lfloor \frac{T}{k} \rfloor - 1$. Demostremos esto:

Cumpliendo con las restricciones del problema, siempre es posible formar $\lfloor \frac{M}{k} \rfloor = p_1 + \lfloor \frac{F}{k} \rfloor = p_2$ grupos factibles, donde cada unidad de los grupos pertenece a una misma fila, quedando r_1 y r_2 elementos residuales en cada fila respectivamente. Luego:

$$M = p_1 * k + r_1, \;\; r_1 < k$$

$$F = p_2 * k + r_2, \;\; r_2 < k$$

$$T = M + F = p_1 * k + p_2 * k + r_1 + r_2$$

Como $r_1, r_2 < k$, entonces $r_1 + r_2 < 2k$. Luego, si $r_1 + r_2 < k$, tomando $r_3 = r_1 + r_2$, se cumple que:

$$T = (p_1 + p_2)k + r_3, \;\; r_3 < k$$

Luego $\lfloor \frac{T}{k} \rfloor = p_1 + p_2 = \lfloor \frac{M}{k} \rfloor + \lfloor \frac{F}{k} \rfloor$

Si $r_1 + r_2$ no es menor que k entonces $r_1 + r_2 \geq k$, entonces $r_1 + r_2 = r_3 + k$ con $r_3 < k$. Luego se cumple que:

$$T = (p_1 + p_2)k + k + r_3 = (p_1 + p_2 + 1)k + r_3$$

Luego $\lfloor \frac{T}{k} \rfloor = p_1 + p_2 + 1$ entonces $\lfloor \frac{T}{k} \rfloor - 1 = p_1 + p_2 = \lfloor \frac{M}{k} \rfloor + \lfloor \frac{F}{k} \rfloor$

Para el caso en que $r_1 + r_2 < k$, como $\lfloor \frac{T}{k} \rfloor = p_1 + p_2$ esta solución es factible y óptima.

Para el caso en que $r_1 + r_2 \geq k$, $p_1 + p_2 = \lfloor \frac{T}{k} \rfloor - 1$ es la solución óptima si no es posible crear grupos con elementos pertenecientes a una misma columna.

En otro caso, en el que existe i tal que $a_i + b_i \geq k$, $p_1 + p_2$ no se garantiza, de momento, que sea una solución óptima. Se podría buscar i tal que $min(a_i, r_1) + min(b_i, r_2) \geq k$, para este caso bastaría con asignar el resto de cada fila a a_i y b_i respectivamente con ello se formaría un grupo más con elementos pertenecientes a una misma columna, sin disminuir la cantidad $p_1 + p_2$ de grupos que se pueden formar, logrando crear $p_1 + p_2 + 1$ grupos que es un resultado óptimo.

Pero de no existir dicho i no se garantiza aún que $p_1 + p_2$ sea un resultado óptimo, cabe la posibilidad de que dejando de crear m grupos con elementos pertenecientes a una misma fila se pudiesen formar $m + 1$ grupos con elementos pertenecientes a una misma columna. Nótese que para las columnas i tal que $a_i + b_i < k$ los elementos de a_i y b_i solo pueden ser utilizados para formar grupos de elementos pertenecientes a una misma fila.

Esto se puede ver como el siguiente subproblema:

Se tiene una submatriz de la matriz A de entrada donde solo están las columnas i pertenecientes a A tal que $A[0,i] + A[1,i] \geq k$. Sea $r_1 = M - p_1 * k$ y $r_2 = F - p_2 * k$, tal que $r_1, r_2 < k$ y $r_1 + r_2 \geq k$ ($p_1 + p_2 = \lfloor \frac{T}{k} \rfloor - 1$). ¿Es posible encontrar un conjunto de columnas S de tamaño m tal que pueda formar un grupo con elementos pertenecientes a una misma columna por cada columna en S, dejando de hacer $m-1$ grupos de elementos pertenecientes a una misma fila? ($\lfloor \frac{T}{k} \rfloor = p_1 + p_2 + m - (m-1)$)

Anteriormente se demostró que existe un óptimo que solo usa un grupo columna a lo sumo por columna. Si existiese una solución S de este subproblema se debe cumplir que:

Siendo:

$$M' = M - \sum S[0,i]$$

$$F' = F - \sum S[1,i]$$

Entonces $M' \,\%k + F' \,\%k < k$ pues:

$$T = |S| * k + \lfloor \frac{M'}{k} \rfloor * k + M' \,\%k + \lfloor \frac{F'}{k} \rfloor * k + F' \,\%k$$

$$T = (|S| + \lfloor \frac{M'}{k} \rfloor + \lfloor \frac{F'}{k} \rfloor) * k + M' \,\%k + F' \,\%k$$

Por lo que por el algoritmo de la división: $|S| + \lfloor \frac{M'}{k} \rfloor + \lfloor \frac{F'}{k} \rfloor = \lfloor \frac{T}{k} \rfloor$

$$T = \lfloor \frac{T}{k} \rfloor * k + T \,\%k$$

Nótese que si $M' \,\%k + F' \,\%k < k$, entonces, además, $M' \,\%k + F' \,\%k = T \,\%k$

Por tanto, si existe una solución S, $M' \,\%k + F' \,\%k = T \,\%k$.

Demostremos que $M' \,\%k + F' \,\%k = T \,\%k \Leftrightarrow M' \,\%k \leq T \,\%k$:

$\Rightarrow$

Supongamos con el fin de llegar a un absurdo que $M' \,\%k > T \,\%k$, entonces $M' \,\%k + F' \,\%k > T \,\%k$, contradicción, pues $M' \,\%k + F' \,\%k = T \,\%k$. Luego $M' \,\%k \leq T \,\%k$

$\Leftarrow$

$$T = (|S| + \lfloor \frac{M'}{k} \rfloor + \lfloor \frac{F'}{k} \rfloor) * k + F' \,\%k + M' \,\%k$$

$$T \,\%k = ((|S| + \lfloor \frac{M'}{k} \rfloor + \lfloor \frac{F'}{k} \rfloor) * k + F' \,\%k + M' \,\%k) \,\%k$$

$$T \,\%k = (|S| + \lfloor \frac{M'}{k} \rfloor + \lfloor \frac{F'}{k} \rfloor) * k \,\%k + (F' \,\%k + M' \,\%k) \,\%k$$

$$T \,\%k = 0 + (F' \,\%k + M' \,\%k) \,\%k$$

$$T \,\%k = (F' \,\%k + M' \,\%k) \,\%k$$

Luego T y $(F'\,\%k + M'\,\%k)$ son congruentes módulo k.

Si $M'\,\%k \leq T\,\%k$ entonces $F'\,\%k = T\,\%k - M'\,\%k + x * k$. Como $F'\,\%k < k$, entonces $x = 0$ y $F'\,\%k = T\,\%k - M'\,\%k$. Luego $M'\,\%k + F'\,\%k = T\,\%k$

Por tanto si existiese una solución S se debe cumplir que:

$$(M - \sum S[0,i])\,\%k \leq T\,\%k$$

$$M\,\%k - T\,\%k \leq (\sum S[0,i])\,\%k$$

y además como $(M - \sum S[0,i]) \geq 0$

$$M\,\%k \geq \sum S[0,i]\,\%k$$

entonces se cumple que:

$$M\,\%k - T\,\%k \leq (\sum S[0,i])\,\%k \leq M\,\%k$$

En este punto el problema se ve reducido a determinar si existe S tal que:

$$M\,\%k - T\,\%k \leq (\sum S[0,i])\,\%k \leq M\,\%k$$

. Por tanto, si se pudiese calcular los valores de $(\sum S[0,i])\,\%k$ que son posibles dada la matriz, se pudiese saber si es posible armar $\lfloor \frac{T}{k} \rfloor$ grupos.

5.1. Uso de Programación Dinámica

Inicialmente nos interesa saber por cada columna m_i de M, los posibles valores que se pueden utilizar de m_i para formar un grupo en su columna. Llamemos a estos valores $m_{i,j}$. Como sabemos que los posibles valores que puedo tomar $(\sum S[0,i])\,\%k$ son a lo sumo k es posible representar estos valores a traves de un array z de tamaño k. Donde la posición i-ésima de z tiene un 1 si es posible que $(\sum S[0,i])\,\%k$ tomo el valor i y un 0 si no (en realidad solo nos interesan $k-1$ posibles valores que puede tomar $(\sum S[0,i])\,\%k$, pues que $(\sum S[0,i])\,\%k = 0$ nunca es solución, ya que $M\,\%k > T\,\%k$).

Sea z_i el array que representa los posibles valores que puede tomar $(\sum S[0,i])\,\%k$ usando solo las primeras i columnas de la submatriz, a partir de z_i es posible calcular z_{i+1} de la siguiente manera:

• Todos los valores que son 1 en z_i también los son en z_{i+1}, pues si era posible hacer dicho valor sin considerar la columna $i+1$, simplemente se puede lograr sin usarla.

• Todos los valores $m_{i+1,j}$ son 1 en z_{i+1}, puesto que estos son posible lograrse sin utilizar ninguna de las anteriores i columnas.

• Por cada valor x en 1 en z_i, los valores $(x + m_{i+1,j})\,\%k$ son 1 en z_{i+1} para todo j.

Para z_1 basta con hacer 1 todos los valores $m_{1,j}$. Calculando los z_i hasta el último de ellos, en este estarán marcados en 1 todos los valores de $(\sum S[0,i])\,\%k$ que son posibles dada la matriz.

Luego basta comprobar alguno de los posibles valores de $(\sum S[0,i])\,\%k$ pertenece al intervalo

$$M\,\%k - T\,\%k \leq (\sum S[0,i])\,\%k \leq M\,\%k$$

.

5.2. Complejidad Temporal

Calcular la cantidad de elementos pertenecientes a cada fila tiene un costo $O(n)$. Calcular los valores $m_{i,j}$ tiene un costo $O(n*k)$, pues para cada columna n es posible que se usen a lo sumo k posibles valores distintos. Crear los arrays z_i tiene un costo $O(n*k^2)$, pues por cada columna z_i de las n columnas, es necesario verificar los posibles k valores de $(\sum S[0,i])\,\%k$ que se pudieron generar en z_{i-1} y por cada uno de ellos comprobar que valores se pueden lograr sumando con los k posibles valores de los $m_{i,j}$. Luego verificar si alguno de los valores posibles de $(\sum S[0,i])\,\%k$ pertenece al intervalo $[M\,\%k - T\,\%k, M\,\%k]$ tiene un costo $O(k)$. Luego el algoritmo tiene complejidad $O(n + n*k + n*k^2 + k)$ y por el teorema de la suma, la complejidad final es $O(n*k^2)$.

6. Generadores de Casos Prueba

Se implementaron tres generadores de casos prueba. Su funcionamiento consiste en generar unos n, k aleatoriamente, para luego generar los dos arrays de entrada. Sea a_i y b_i los elementos de los dos arrays de entrada en la posición i respectivamente. Para generar ambos arrays, se generan de forma aleatoria a_i y b_i para cada i, tal que $a_i + b_i > 0$. Lo que diferencia a los tres generadores son los rangos de generación para cada variable que se genere aleatoriamente.

6.1. generator1

Este generador está pensado para las pruebas que se realicen con el algoritmo propuesto en la primera aproximación. Debido a la alta complejidad de este primer algoritmo, los casos que crea generator1, son relativamente pequeños.

6.2. generator2

Este generador está pensado para las pruebas que se realicen con los algoritmos propuestos en la segunda y tercera aproximación. Genera casos más grandes que el primer generador.

6.3. generator3

Este generador está pensado para las pruebas que se realicen con el algoritmo de la tercera aproximación. Los casos que este generador crea son bastante grandes, tanto que provar los algoritmos de las primeras aproximaciones es impracticable.

7. Tester

El tester es una aplicación de consola interactiva. Donde el usuario debe escoger dos algoritmos para comparar sus resultados, un generador de casos prueba y una cantidad de casos prueba. Para cada caso prueba el tester compara la solución de los dos algoritmos seleccionados. En caso de que coincidan imprime un mensaje en color verde, en caso de que no coincidan imprime un mensaje en rojo.

Se aconseja realizar las pruebas de los algoritmos con generator1 debido a la rapidez con que los algoritmos propuestos resuelven los casos.

Se pueden comparar los resultados de los algoritmos de la segunda y tercera aproximación con generator2, aunque la prueba será más lenta debido a la alta complejidad temporal de la segunda aproximación.

El generador generator3 está diseñado para probar la rapidez del algoritmo de la tercera aproximación, no para la comparación con los restantes algoritmos.

7.1. Instrucciones

Abrir una consola en la carpeta donde se encuentra el archivo tester.py. Ejecutar el comando python tester.py.

Si se quisiera no comparar dos algoritmos, sino probar la rapidez contra un generador determinado, seleccione el algoritmo dos veces.