Diseño y Análisis de Algoritmos

Problema 3: El Profe

Integrantes:

- Ana Karla Caballero González
- Carlos Arturo Pérez Cabrera
- Diana Laura Pérez Trujillo

1. Problema

Rodrigo es profesor de programación. En sus ratos libres, le gusta divertirse con las estadísticas de sus pobres estudiantes reprobados. Los estudiantes están separados en n grupos. Casualmente, este año, todos los estudiantes reprobaron alguno de los dos exámenes finales: P (POO) y R (Recursividad).

Esta tarde, Rodrigo decide entretenerse separando a los estudiantes suspensos en conjuntos de tamaño k que cumplan lo siguiente: En un mismo conjunto, todos los estudiantes son del mismo grupo i ($1 \leq i \leq n$) o suspendieron por el mismo examen P o R.

Conociendo el grupo y la prueba suspendida de cada estudiante, y el tamaño de los conjuntos, ayude a Rodrigo a saber cuántos conjuntos de estudiantes suspensos puede formar.

2. Interpretación Matemática

Considérese una matriz cuyas filas representen las dos posibles pruebas P y R, y la columnas los diferentes grupos de los estudiantes. En la posición i, j se encuentre un entero c, tal que c es la cantidad de estudiantes que suspendieron la prueba i del grupo j. El problema se describe entonces como encontrar cual es el mayor número de conjuntos de tamaño k que se pueden formar en la matriz, que cumplan una de las siguientes condiciones: o los k elementos tienen la misma fila o los k elementos tienen la misma columna.

3. Soluciones

3.1. Primera Solución

La primera idea que salta ante el problema es hallar todas las posibles permutaciones, y de ellas seleccionar conjuntos de tamaño k. Para ello se crea un arreglo T que contiene todos los estudiantes representados en la matriz inicial dada, de forma que en cada posición del arreglo se encuentra representado un único estudiante. Se crea una lista con todas las posibles permuatciones de dicho arreglo T, de manera que después se van seleccionando conjuntos de tamaño k. Por cada uno de los conjuntos se verifica si sus k elementos cumplen una de las dos condiciones del problema, de ser así es considerado como una distribución valida. De entre las distintas distribuciones obtenidas, se selecciona la que mayor cantidad de conjuntos tenga, así se obtiene la mayor cantidad de conjuntos de tamaño k que es posible conformar cumpliendo ambas restricciones.

3.1.1. Correctitud

Supóngase que la secuencia o distribución de los estudiantes que mayor conjunto de tamaño k contiene es una secuencia S. El algoritmo propuesto genera todas las permutaciones posibles y dentro de ellas, las porciones de tamaño k

que se pueden formar, por tanto en algún momento el algoritmo pasa por la secuencia S. Por lo que se puede decir que siempre se encontrará la solución óptima.

3.1.2. Análisis de complejidad temporal

El primer paso en el algoritmo es tomar los elementos de las filas correspondientes a los dos examenes y agregarlos a la lista T que contendrá todos los estudiantes representados en la matriz inicial, cumpliendo que en una posicón esté representado un único estudiante. Esta acción se hace en O(T), donde T es el total de estudiantes. Computar todas las posibles permutaciones tiene una complejidad temporal de O(T!), pues por cada elemento de T que se selecciona como elemento en una posición, se puede continuar con T-1 elementos restantes. Realizar las comprobaciones necesarias para determinar si una distribución cumple con las condiciones del problema es O(T), este proceso se realiza por cada una de las permutaciones, que ya se planteó que son T!, luego, este proceso sería O(T*T!). Por la regla de la suma, se concluye que la complejidad final del algoritmo es O(T*T!).

3.2. Segunda Solución

Una segunda idea resulta en formar al menos un conjunto por columna, si es que la suma de los estudiantes de dicha columna cumple que es mayor o igual a k, de otro modo ese conjunto columna no tendría sentido, y luego calcular los conjuntos por filas posibles. Se llegó a la conclusión de que exite un óptimo para el cual es necesario formar a lo sumo un conjunto por columna, y el resto formarlo por las filas.

Sea T el total de estudiantes, de manera que: $T = p + r$ donde:

p: estudiantes que suspendieron la prueba P

r: estudiantes que suspendieron la prueba R

Se sabe que $p = x * k + rest_1$ donde x es la cantidad de conjuntos de estudiantes que se pueden formar teniendo la misma prueba P y $r = y * k + rest_2$ donde y es la cantidad de conjuntos de estudiantes que se pueden formar teniendo la misma prueba R. Entonces $x * k + rest_1 + y * k + rest_2 = T$.

$(x + y) * k + rest_1 + rest_2 = T$.

Como $rest_1, rest_2 < k$ entonces pueden suceder dos cosas: (1)$rest_1 + rest_2 < k$ o (2)$k <= rest_1 + rest_2 < 2k$.

Si ocurre (1) entonces el total de conjuntos de tamaño k de estudiantes que se pueda formar es $x + y$.

Si ocurre (2) entonces se puede formar un conjunto de tamaño k de estudiantes de la misma columna. Luego se tiene $(x + y) * k + k + rest_3 = T$ donde $rest_3 = rest_1 + rest_2 - k$, $rest_3 < k$.

Luego $(x + y + 1) * k + rest_3 = T$.

Sea c una columna con más de k estudiantes, p_i es la cantidad de estudiantes que suspendieron el examen P en el aula i y r_i es la cantidad de estudiantes que suspendieron el examen R en el aula i. $p_i, r_i > 0$ y $p_i + r_i >= k$.

Luego en el aula $c = u * k + v * k + rest_1 + rest_2$ donde u, v son conjuntos de estudiantes de la misma aula del mismo examen respectivamente.

Si $rest_1 + rest_2 > k$ entonces $(u + v + 1) * k + rest_3 = c$ por lo que se demuestra que existe una forma óptima de organizar los estudiantes en los conjuntos de manera que se pueden formar $(x + y + 1)$ conjuntos si existe al menos un aula con más de k estudiantes.

3.2.1. Correctitud

Se demostró que solo se necesita formar un conjunto por columna y los demás por fila. El algoritmo genera todas los posibles conjuntos por columnas que se pueden formar y con una distribución de conjuntos por columna se generan los posibles conjuntos por fila. De todos esos valores, el algoritmo se queda con el mayor que resulta la mayor cantidad de conjuntos de tamaño k que se pueden formar cumpliendo las restricciones del problema.

3.2.2. Complejidad Temporal

Todas las formas de seleccionar columnas es $O(2^n)$. Luego, el backtrack realizado tiene un costo $O(n*k)$. Por el algoritmo de la multiplicación, la solución planteada tiene una complejidad temporal $O(2^n + F * n * k)$, siendo F la cantidad de formas de seleccionar columnas encontrada.

3.3. Tercera Solución

Para la siguiente solución, se debe demostrar que siempre es posible formar al menos $\lfloor T/n \rfloor - 1$ conjuntos.

Sea M la matriz que representa los estudiantes, entonces:

$\lfloor \sum_{i=0}^{n} M[0, i]/k \rfloor + \sum_{i=0}^{n} \lfloor M[1, i]/k \rfloor = \lfloor T/k \rfloor$

Se tiene que la cantidad de grupos totales va a ser igual a la cantidad de grupos de la prueba P y la prueba R quedando $rest_1$ y $rest_2$ como residuales de cada

prueba, donde $rest_1, rest_2 < k$.

Anteriormente se demostró que pueden suceder dos cosas:

(1)$rest_1 + rest_2 < k$ o (2)$k \leq rest_1 + rest_2 < 2k$.

Para (1) la solución óptima es $\lfloor T/k \rfloor$.

Para (2) $\lfloor T/k \rfloor - 1$ es la solución óptima si no es posible crear un conjunto de estudiantes de la misma aula. En otro caso, no se garantiza de momento que sea una solución óptima, dado que habría que buscar el i tal que $min(M[0,1], rest_1) + min(M[1,i], rst_2) \geq k$. Con esto se formaría un conjunto más de estudiantes de la misma aula, logrando crear $x + y + 1$ conjuntos que es un resultado óptimo.

En caso de no existir dicho i el resultado sería $x + y$, por lo que para cualquier conjunto de estudiantes T y una manera de agruparlos k existe al menos $\lfloor T/k \rfloor - 1$ conjuntos de tamaño k.

Si se pudiera crear una submatriz A de tamaño m donde estarán solo las aulas que contengan al menos k estudiantes, donde $A[0,i], A[1,i] > 0$, en esta se podrán crear m conjuntos de estudiantes pertenecientes a la misma aula, por lo que si se tiene en cuenta la cantidad de conjuntos creados en A, dejando de hacer $m - 1$ conjuntos de estudiantes por pruebas, se tiene el óptimo.

$T = x + y + m - (m - 1) => T = x + y + m - m + 1 => T = x + y + 1$.

Apoyándose de lo demostrado en la segunda solución, se sabe que existe un óptimo que usa un conjunto de estudiantes del mismo grupo. Entonces si existiese A se seguiría cumpliendo que:

$P' = P - \lfloor \sum_{i=0}^{n} A[0,i] \rfloor$

$R' = R - \lfloor \sum_{i=0}^{n} A[1,i] \rfloor$

Entonces $T = |A| * k + (\lfloor P'/k \rfloor + \lfloor R'/k \rfloor) + k + P' \,\%k + R' \,\%k$.

Por el algoritmo de la división se tiene que el $\lfloor T/k \rfloor = |S| + \lfloor P'/k \rfloor + \lfloor R'/k \rfloor$

Se puede decir que $T = \lfloor T/k \rfloor * k + T \,\%k$ donde si $P' \,\%k + R' \,\%k < k$ entonces también $P' \,\%k + R' \,\%k = T \,\%k$. Por tanto, si existe la matriz solución A, $P' \,\%k + R' \,\%k = T \,\%k$.

Luego $P' \,\%k \leq T' \,\%k$

Sustituyendo P':

$(P - \sum_{i=1}^{m} A[0,i]) \,\%k \leq T \,\%k$.

$P \,\%k - \sum_{i=1}^{m} A[0,i] \,\%k \leq T \,\%k$.

$P \,\%k - T \,\%k \leq \sum_{i=1}^{m} A[0,i]$

Como $P' \,\%k > 0$ se cumple que $P \,\%k \geq \sum_{i=1}^{m} A[0,i] \,\%k$.

Uniendo ambas condiciones:

$P \,\%k - T \,\%k \leq \sum_{i=1}^{m} A[0,i] \,\%k \leq P \,\%k$

Se reduce el problema a encontrar valores en la submatriz A tal que cumplan estas condiciones. En caso de encontrarlos se puede decir que es posible armar $\lfloor T/k \rfloor$ conjuntos.

3.3.1. Correctitud

Se quiere hallar por cada aula a_i de la prueba P los posibles valores que se pueden utilizar de a_i para formar un conjunto en su aula, llamemos a estos valores a_{ij}.
Se sabe que los posibles valores que se pueden tomar son a lo sumo k es posible representarlos a través de un array z de tamaño k donde en la posición i-ésima de z se tiene un 1 si es posible que $\sum_{i=1}^{m} A[0,i] \,\%k$ tomó el valor i o un 0 si no. En realidad solo interesan los $k-1$ valores posibles dado que $\sum_{i=1}^{m} A[0,i] \,\%k = 0$ no es solución ya que se debe cumplir que $P \,\%k > T \,\%k$.
Sea z_i el array que representa los posibles valores que peude tomar $\sum_{i=1}^{m} A[0,i] \,\%k$ usando solo las primeras i columnas de la submatriz, a partir de z_i es posible calcular z_{i+1} de la siguiente manera:

1- Para todos los valores que son 1 en z_i también lo son en z_{i+1}.
2- Para todos los valores $a_{i+1,j}$ son 1 en z_{i+1} pues es posible lograrse sin usar ninguna de la anteriores columnas.
3- Para cada valor v en 1 en z_i los valores $(v + a_{i+1,j}) \,\%k$ son 1 en z_{i+1} para todo j.

Luego, basta con hacer 1 todos los valores de $a_{i,j}$ para z_1 y se calcula todos los z_i donde se marcarán todos los valores de $\sum_{i=0}^{m} A[0,i] \,\%k$ que son posibles dada la matriz.

Finalmente, solo se comprobaría si alguno de los posibles valores de $\sum_{i=0}^{m} A[0,i] \,\%k$ pertenece al intervalo:

$P \,\%k - T \,\%k \leq \sum_{i=1}^{m} A[0,i] \,\%k \leq P \,\%k$

3.3.2. Complejidad Temporal

Calcular los elementos pertenecientes a cada prueba es $O(n)$. El cálculo de los valores $a_{i,j}$ tiene costo $O(n * k)$, pues en cada columna n es posible que se usen k valores distintos. La creación de los z_i es $O(n * k^2)$, por cada aula es necesario verificar los k valores de $\sum_{i=1}^{m} A[0, i] \,\%k$ que se pudieron generar en z_{i-1}. Por cada uno de ellos, comprobar qué valores se pueden lograr sumando con los k posibles de los $a_{i,j}$. La verificación de pertenencia al intervalo tiene un costo $O(k)$. La complejidad final del algoritmo es $O(n + n * k + n * k^2 + k)$, es decir, $O(n * k^2)$.

3.4. Casos Prueba

Para comprobar que las propuestas fueran soluciones válidas, se implementaron un generador de casos prueba y un tester.
El código presentado implementa tres generadores de casos de prueba, cada uno diseñado para crear diferentes conjuntos de datos aleatorios que pueden ser utilizados para evaluar algoritmos o sistemas.

generator1:
El primer generador, generator1, produce un número especificado de casos de prueba en los que se generan dos listas (prueba1 y prueba2) que representan la cantidad de estudiantes disponibles para dos pruebas distintas. El número de estudiantes disponibles se inicia en 8, y se selecciona aleatoriamente cuántos estudiantes participan en cada prueba, asegurando que no se exceda el total disponible. Además, se incluye un valor k que varía entre 1 y 8.

generator2:
El segundo generador, generator2, crea casos de prueba con un número variable de elementos (entre 1 y 7). En este caso, cada elemento tiene dos valores (mi y fi) que representan cantidades aleatorias entre 0 y 20, garantizando que al menos uno de ellos sea mayor que cero. Se calcula un total acumulado de estos valores, y también se genera un valor k que varía entre 1 y 10.

generator3:
Finalmente, el tercer generador, generator3, es similar al anterior pero permite una mayor variabilidad en el número de elementos (hasta 50) y en los valores generados para mi y fi, que pueden ir de 0 a 100. Al igual que en el segundo generador, se asegura que al menos uno de los valores sea positivo. El valor k se genera aleatoriamente entre 1 y el total acumulado de los valores.

4. Organización y requerimientos

En la carpeta raíz del proyecto se encuentra una distribución para cada problema. En la carpeta *Problema*3 se encuentra el archivo *problema*3.*tex*, dentro

de ella, en la carpeta *code* se encuentran los tres archivos de código fundamentales: $Problema3.py$ que contiene el código de las tres soluciones propuestas, $Tester.py$ que contiene la consola interactiva para seleccionar la configuración del proceso de testing y $TestGenerator.py$ que contiene la lógica de la generación de grafos para los diferentes casos.

Todo fue implementado en el lenguaje de programación *Python* y con el apoyo de las siguientes bibliotecas o módulos:

- itertools
- colorama
- random