Diseño y Análisis de Algoritmos. Problema 1: El profe Leandro

Jorge Soler González y Ernesto Alfonso Hernández

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

1. Definición del Problema

Leandro 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 estudaintes reprobaron alguno de los dos exámenes finales P (POO) y R (Recursividad).

Esta tarde, Leandro 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 exámen P o R.

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

2. Modelación del Problema

El problema consiste en hallar la mayor cantidad de conjuntos de tamaño k que se puedan formar, donde cada elemento de un conjunto es un estudiante y además, todos los elementos de un conjunto deben pertenecer al mismo grupo o tienen que tener la misma asignatura suspensa. Para una mejor modelación del problema, la lista inicial de estudiantes se divide en 2 listas, la primera contiene en la posicion i la cantidad de estudiantes que pertenecen al grupo $i + 1$ y suspendieron P, mientras que la segunda contiene en la posición i la cantidad de estudiantes que pertenecen al grupo $i + 1$ y suspendieron R, de esta forma armamos una matriz de $2 * n$ con los datos de entrada, donde n es la cantidad de grupos, de esta matriz sabemos que las filas indican todos los estudiantes que suspendieron la misma asignatura y las columnas los estudiantes que pertenecen al mismo grupo.

3. Primera Solución

Como primera solución se nos ocurre usar los datos de entrada tal y como están, o sea, una lista de estudiantes, que son modelados a partir de una clase "Estudiante", que contiene una propiedad grupo, y una propiedad asignatura

suspensa. A partir de esta lista, generar todas las permutaciones y por cada una dividirla mediante índices en conjuntos de tamaño k, luego contar todos los conjuntos válidos y actualizar, de ser necesario, el máximo;

3.1. Optimalidad

Sea $S = [e_1, e_2, \ldots, e_T]$ una distribución de los estudiantes de la lista de entrada, tal que la cantidad de conjuntos 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 lista de estudiantes. Sea P el conjunto que contiene a todas las permutaciones posibles. Nuestro primer acercamiento revisa a P y calcula la cantidad de conjuntos válidos en cada P_i, luego $S \in P$. Luego el algoritmo encuentra la respuesta óptima.

3.2. Complejidad Temporal

Sea E la lista de estudiantes de entrada. Computar todas las permutaciones de la lista E tiene un costo $O(|E|!)$. El cálculo de cuántos conjuntos válidos existen en una permutación de $|E|$ tiene complejidad $O(|E|)$ y este cálculo se realiza para cada permutación. Entonces por el teorema de la multiplicación calcular los conjuntos válidos que tiene cada permutación tiene complejidad $O(|E| * |E|!)$. Luego el algoritmo tiene complejidad $O(|E| * |E|!)$

3.3. Pseudocódigo

```
def Solve_bruto(students : list[Student], k : int):
    all_perm = permutations(students)
    max_groups = 0
    for perm in all_perm:
        count = 0
        for i in range(Parte_Entera_(len(perm) / k)):
            sub_list = perm[i*k : (i*k + k)]
            correct_subject_set = True
            correct_group_set = True

            for j in range(len(sub_list) - 1):
                if sub_list[j].suspended != sub_list[j+1].suspended:
                    correct_subject_set = False
                    break

            for j in range(len(sub_list) - 1):
                if sub_list[j].group != sub_list[j+1].group:
                    correct_group_set = False
                    break
```

```
            if correct_subject_set or correct_group_set:
                count += 1

        max_groups = max(max_groups, count)

    return max_groups
```

4. Intentos de Mejora

Nuestra primera propuesta de solución tiene un costo computacional elevado, para un tamaño de entrada lo suficientemente grande, el tiempo de ejecución es absurdo. Para lograr una mejora al tiempo de complejidad de la solución, hemos pasado por varias etapas y aproximaciones a la que sería la mejor solución encontrada. A partir de este momento vamos a tratar los datos de entrada como la matriz de $2 * n$ explicada en la modelación del problema.

4.1. Primera aproximación

Nos damos cuenta de un resultado importante:

Sea E la lista de estudiantes de entrada, y que a partir de ella se arma la matriz M definida en la modelación del problema. Demostremos que para cualquier entrada del problema, la solución a este es mayor o igual que $\lfloor \frac{|E|}{k} \rfloor$:

A partir de la matriz de armada con los datos de entrada, sea P la primera fila de la matriz, y R la segunda. sea $SP = \sum_{i=0}^{n} P_i$ y $SR = \sum_{i=0}^{n} R_i$

Cumpliendo con las restricciones del problema, siempre es posible formar $\lfloor \frac{SP}{k} \rfloor = s_p + \lfloor \frac{SR}{k} \rfloor = s_r$ conjuntos válidos, donde cada elemento de los conjuntos pertenecen a una misma fila, quedando r_p y r_r restos de cada fila respectivamente. Luego:

$$SP = s_p * k + r_P, \;\; r_p < k$$

$$SR = s_r * k + r_r, \;\; r_r < k$$

$$T = SP + SR = s_p * k + s_r * k + r_p + r_r$$

Como $r_p, r_r < k$, entonces $r_p + r_r < 2k$. Luego, si $r_p + r_r < k$, tomando $r' = r_p + r_r$, se cumple que:

$$T = (s_p + s_r)k + r', \;\; r' < k$$

Luego $\lfloor \frac{T}{k} \rfloor = s_p + s_r = \lfloor \frac{SP}{k} \rfloor + \lfloor \frac{SR}{k} \rfloor$

Si $r_p + r_r$ no es menor que k entonces $r_p + r_r \geq k$, entonces $r_p + r_r = r' + k$ con $r' < k$. Luego se cumple que:

$$T = (s_p + s_r)k + k + r' = (s_p + s_r + 1)k + r'$$

Luego $\lfloor \frac{T}{k} \rfloor = s_p + s_r + 1 \Rightarrow \lfloor \frac{T}{k} \rfloor - 1 = s_p + s_r = \lfloor \frac{SP}{k} \rfloor + \lfloor \frac{SR}{k} \rfloor$

Para el caso en que $r_p + r_r < k$, como $\lfloor \frac{T}{k} \rfloor = s_p + s_r$ esta solución es factible y óptima.

Para el caso en que $r_p + r_r \geq k$, $s_p + s_r = \lfloor \frac{T}{k} \rfloor - 1$ 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 $P_i + R_i \geq k$, $s_p + s_r$ no se garantiza que sea una solución óptima.

La primera idea es buscar la mayor cantidad de conjuntos de tamaño k que se pueden hacer por filas, o sea, por asignatura, esto es equivalente a $\frac{\sum_{i=0}^{n} P_i}{k} + \frac{\sum_{i=0}^{n} R_i}{k}$. A partir de este punto pueden suceder varias cosas:

- Si $r_p + r_r < k \Rightarrow |E| \,\%k = r_p + r_r$ por lo tanto se lograron armar la mayor cantidad de conjuntos de tamaño k.
- Si $r_p + r_r \geq k \Rightarrow$ existe la posibilidad de armar un nuevo conjunto

Para resolver la problemática de encontrar el conjunto que podría faltar, primeramente se pensó en lo siguiente: Si se cumple el siguiente predicado: $min(r_p, P_i) + min(r_r, R_i) \geq k \Rightarrow$ parte de la columna i se puede utilizar para armar el nuevo conjunto, esta vez con el criterio de que todos sus elementos pertenecen al mismo grupo.

Demostremos lo anterior: Con esta forma de maximizar los conjuntos filas se cumple que $r_p + r_r < 2k$ ya que cada $r < k$. Luego con el resto solo será posible armar a lo sumo un conjunto más en caso que este cumpla que es mayor que k. Además si un estudiante suspendió la misma asignatura que otro es indiferente cual escoger para formar un conjunto, pués si con el otro era posible con este último también. Al escoger $min(r_p, P_i)$ se garantiza que nunca se intercambien más estudiantes pertenecientes a un grupo que los que son posibles intercambiar con el resto.(Homólogamente para R) Luego si $\sum min \geq k$ significa que el resto que quedó pudo tener al menos k estudiantes pertenecientes al mismo grupo y como anteriormente se indicó que era posible intercambiar los estudiantes que habían quedado en el resto por otros que suspendieron la misma asignatura, entonces es posible intercambiar k estudiantes del grupo i por los estudiantes que quedaron en el resto, luego se crearía el conjunto que falataba con esos k estudiantes intercambiados. Este es un conjunto válido pués todos los estudiantes pertenecen al mismo grupo. El resultado anterior es un resultado correcto, es decir si ocurre que $min(r_p, P_i) + min(r_r, R_i) \geq k \Rightarrow$ siempre será posible formar un conjunto nuevo, pero existen casos que no se cumple esto y aún así es posible crear un nuevo conjunto, veamos uno:

Para la matriz:

$$\begin{pmatrix} 6 & 6 & 6 \\ 1 & 1 & 1 \end{pmatrix}$$

Con $k = 7$, aplicando la idea anterior a esta entrada, tenemos que en principio por fila se pueden armar 2 conjuntos, y tenemos que $r_p = 4$ y $r_r = 3$, aquí pasamos a comprobar si se pude crear un nuevo conjunto, ya que $r_p + r_r \geq k$, la condición anterior nos dice que después de una pasada no se puede crear el nuevo conjunto, ya que el mínimo para cada columna i es 4 y 1, porque nunca se llega a k. Pero observemos la respuesta correcta en este caso es 3 conjuntos, por tanto nuestro algoritmo es incorrecto.

4.2. Segunda Aproximación

En este punto, nos damos cuenta de un resultado interesante: Dado una columna tal que $P_i + R_i \geq 2k \Rightarrow$ existe una solución que solamente coge un cojunto donde todos sus elemenetos pertenecen a esa columna y logra armar $\frac{P_i+R_i}{k} - 1$ conjuntos que todos sus elementos pertenecen a la misma fila.

Demostremos lo anterior:

Sea $P_i + R_i = c$.

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

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

$$P_i = p_p * k + r_p, \quad r_p < k$$

p_p, es la cantidad de conjuntos fila que se pueden formar solo con los elementos de P_i.

$$R_i = p_r * k + r_r, \quad r_r < k$$

p_r, es la cantidad de conjuntos fila que se pueden formar solo con los elementos de R_i. Por tanto:

$$c = P_i + R_i = p_p * k + p_r * k + r_p + r_r, \quad r_p < k, r_r < k$$

$$c = (p_p + p_r) * k + r_p + r_r$$

con $r_p + r_r < 2k$.

Si $r_p + r_r < k$ entonces por el algoritmo de división, la cantidad de conjuntos que se pueden formar en la columna i, $\lfloor \frac{c}{k} \rfloor = p_p + p_b$. Es decir, para este caso, con formar solo conjuntos filas con los elementos de P_i y R_i se iguala la cantidad de conjuntos columnas que se pueden formar en la columna i.

Si $k \leq r_p + r_r < 2k \Rightarrow r_p + r_r = r' + k$ con $r' < k$, luego:

$$c = (p_p + p_r) * k + r' + k$$

$$c = (p_p + p_r + 1) * k + r', \quad r' < k$$

Esto significa que, por el algoritmo de división, la cantidad de grupos que se pueden formar en la columna i, $\lfloor \frac{c}{k} \rfloor = p_p + p_r + 1$. Es decir, para este caso, con formar solo conjuntos filas con los elementos de P_i y R_i obtengo la cantidad máxima de cojuntos menos 1. Ese conjunto de menos será un conjunto columna

Ahora, partiendo de la primera aproximación. Vamos a dividir el problema general en 2 subproblemas, un primer subproblema, que son las columnas de la matriz que solamente sus integrantes se puedan usar para armar conjuntos por fila, o sea que $P_i + R_i < k$; y un segundo subproblema que tiene las restantes columnas que potecialmente, con ellas se pueden armar conjuntos por fila o por columna. Evidentemente el primer subproblema solo tiene una forma de

resolverse, haciendo los conjuntos por filas. Sea $r_{p1} y r_{r1}$ el resultado de $P_1 \,\%k$ y $R_1 \,\%k$ respectivamente, donde P_1 y R_1 son las filas correspondientes al primer subproblema.

Luego nos queda resolver el segundo subproblema, para esto nos basamos en la idea demostrada anteriormente: de cada columna se puede formar un conjunto donde todos sus elementos pertenecen a la misma columna, y el resto se arma como conjuntos por filas. Para cada una de estas columnas se formara un conjunto de tamaño k, luego sea r_i el resto que corresponde a la columna i luego de armar el conjunto. Sea $r_{P2} = \sum_{i=0}^{n} r_i$ tales que $r_i \in P2$(Homólogamente para r_{R2}). Maximicemos r_{P2}, esto significa siempre que pueda escoger un estudiante de la fila 2 por sobre la fila 1 lo escoja a él. Al maximizar r_{P2} es evidente que se minimiza r_{R2}, maximicemos también r_{R2}, por tanto tenemos, el mínimo y el máximo resto que puede quedar en cada fila.

Ahora, resolvamos por asignatura, es decir, por fila, el resto siguiente: $r_{P2} + r_{p1}$ y $r_{R2} + r_{r1}$, y utilicemos para esto el máximo resto posible a escoger en r_{P2}. El resultado de esta solución, es sumado al resultado anteriormente obtenido de calcular el primer subproblema. Sea r_R y r_P los respectivos restos luego de calcular el resultado anterior. Entonces volvemos a los casos anteriores, si $r_R + r_P < k$ no es posible formar mas conjuntos, pués por el teorema de la división es la mayor cantidad de conjuntos que se pueden formar, si $r_R + r_P \geq k \Rightarrow$ existe la posibilidad de formar otro conjunto.

Definamos el concepto de fluctuación en el problema: Una fluctuación es la acción de escoger un estudiante perteneciente a la fila 1 y porconsiguiente descartar uno de la fila 2 y viceversa.

Luego de esto definamos, la cantidad de fluctuaciones que puede tener r_R y r_P: es $cota_max(r_{P2}) - cota_min(r_P 2)$. Esto es evidente, pues es la cantidad máxima de estudiantes que puedo dejar de coger en $R2$ para cogerlos en $P2$.

Con esta misma idea se resuelve la cantidad de fluctuaciones que puede tener cada columna, y estas también se encuentran acotadas superior e inferiormente en cada casilla de la columna. Recordemos que estas columnas tienen exactamente k estudiantes y estos están distribuidos de forma que se maximice el $r_P 2$ es decir que para cada columna i la cantidad de estudiantes en p_i será mínima, pues el resto de estos escogidos es el mayor posible. Si el valor existente en la columna i original es menor que k entonces este es el valor máximo posible a esocger en su casilla, en caso contrario es k.

A partir de este momento el problema se reduce a encontrar unas fluctuaciones, en caso de que existan, que si eliminando m conjuntos columnas, se logran formar, $m + 1$ conjuntos filas entonces el resultado es $\frac{|E|}{k}$.

El enfoque algoritmico de este problema tiene la misma complejidad temporal, que la primera solución, por lo tanto descartado, pero nos brinda un nuevo enfoque al problema que es a partir que se tenga una cantidad de conjuntos x formados por filas encontrar j de ser posible, tal que $x - j \Rightarrow x + 1$ conjuntos.

5. Mejor solución encontrada

Para esta solución vamos a utilizar un conjunto de ideas que surgieron a partir de las aproximaciones anteriores.

Básandonos en la primera aproximación, recordemos que existe el siguiente conflicto: Hay casos en los que existe i tal que $P_i + R_i \geq k$, $s_p + s_r$ no se garantiza que sea una solución óptima. Se podría buscar i tal que $min(P_i, r_p) + min(R_i, r_r) \geq k$, para este caso bastaría con asignar el resto de cada fila a P_i y R_i respectivamente con ello se formaría un grupo más con elementos pertenecientes a una misma columna, sin disminuir la cantidad $s_p + s_r$ de grupos que se pueden formar, logrando crear $s_p + s_r + 1$ grupos que es un resultado óptimo.

Pero de no existir el i no se garantiza aún que $s_p + s_r$ sea el resultado deseado, aquí aparece el enfoque de la segunda aproximación, si dejando de hacer $x - j$ conjuntos de un tipo, se pueden formar $x + 1$ conjuntos.

De aquí podemos reducir el problema a lo siguiente:

Se tiene una submatriz de la matriz M armada con los datos de entrada donde solo están las columnas i pertenecientes a M tal que $M[0, i] + M[1, i] \geq k$. Sea $r_p = SP - s_p * k$ y $r_r = SR - s_r * k$, tal que $r_p, r_r < k$ y $r_p + r_r \geq k$ ($s_p + s_r = \lfloor \frac{T}{k} \rfloor - 1$). ¿Es posible encontrar un conjunto de columnas C de tamaño m tal que pueda formar un conjunto con elementos pertenecientes a una misma columna por cada columna en C, dejando de hacer $m - 1$ conjuntos de elementos pertenecientes a una misma fila? ($\lfloor \frac{T}{k} \rfloor = s_p + s_r + m - (m - 1)$)

Si existiese una solución C de este subproblema se debe cumplir que: Siendo:

$$SP' = SP - \sum C[0, i]$$

$$SR' = SP - \sum C[1, i]$$

Entonces $SP' \,\%k + SR' \,\%k < k$ ya que:

$$T = |C| * k + \lfloor \frac{SP'}{k} \rfloor * k + SP' \,\%k + \lfloor \frac{SR'}{k} \rfloor * k + SR' \,\%k$$

$$T = (|C| + \lfloor \frac{SP'}{k} \rfloor + \lfloor \frac{SR'}{k} \rfloor) * k + SP' \,\%k + SR' \,\%k$$

Por lo que por el algoritmo de la división: $|C| + \lfloor \frac{SP'}{k} \rfloor + \lfloor \frac{SR'}{k} \rfloor = \lfloor \frac{T}{k} \rfloor$

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

Nótese que si $SP' \,\%k + SR' \,\%k < k \Rightarrow SP' \,\%k + SR' \,\%k = T \,\%k$

Por tanto, si existe una solución C, $SP' \,\%k + SR' \,\%k = T \,\%k$.

Demostremos que $SP' \,\%k + SR' \,\%k = T \,\%k \Leftrightarrow SP' \,\%k \leq T \,\%k$:

$\Rightarrow$

Demostrando por reducción al absurdo, supongamos que $SP' \,\%k > T \,\%k \Rightarrow SP' \,\%k + SR' \,\%k > T \,\%k$, contradicción, ya que $SP' \,\%k + SR' \,\%k = T \,\%k$. Luego $SP' \,\%k \leq T \,\%k$

$\Leftarrow$

$$T = (|C| + \lfloor \frac{SP'}{k} \rfloor + \lfloor \frac{SR'}{k} \rfloor) * k + SR' \,\%k + SP' \,\%k$$

$$T \,\%k = ((|C| + \lfloor \frac{SP'}{k} \rfloor + \lfloor \frac{SR'}{k} \rfloor) * k + SR' \,\%k + SP' \,\%k) \,\%k$$

$$T \,\%k = (|C| + \lfloor \frac{SP'}{k} \rfloor + \lfloor \frac{SR'}{k} \rfloor) * k \,\%k + (SR' \,\%k + SP' \,\%k) \,\%k$$

$$T \,\%k = 0 + (SR' \,\%k + SP' \,\%k) \,\%k$$

$$T \,\%k = (SR' \,\%k + SP' \,\%k) \,\%k$$

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

Si $SP' \,\%k \leq T \,\%k \Rightarrow SR' \,\%k = T \,\%k - SP' \,\%k + x * k$. Como $SR' \,\%k < k \Rightarrow x = 0$ y $SR' \,\%k = T \,\%k - SP' \,\%k$. Luego $SP' \,\%k + SR' \,\%k = T \,\%k$

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

$$(SP - \sum C[0,i]) \,\%k \leq T \,\%k$$

$$SP \,\%k - T \,\%k \leq (\sum C[0,i]) \,\%k$$

y además como $(SP - \sum C[0,i]) \geq 0$

$$SP \,\%k \geq \sum C[0,i] \,\%k$$

entonces se cumple que:

$$SP \,\%k - T \,\%k \leq (\sum C[0,i]) \,\%k \leq SP \,\%k$$

Luego el problema se ve reducido a determinar si existe C tal que:

$$C \,\%k - T \,\%k \leq (\sum C[0,i]) \,\%k \leq SP \,\%k$$

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

Nos basamos para la solución en programación dinámica:

Utilizamos en este punto la idea de las fluctuaciones, vista en la segunda aproxumación. Por cada columna SP_i nos interesa saber las fluctuaciones que pueden tener las casillas de esa columna.

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

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

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

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

Para c_1 basta con hacer 1 todos los valores $SP_{1,j}$. Calculando los c_i hasta el último de ellos, en este estarán marcados en 1 todos los valores de $(\sum C[0,i])\,\%k$ que son posibles dada la matriz.

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

$$SP\,\%k - T\,\%k \leq (\sum C[0,i])\,\%k \leq SP\,\%k$$

.

Complejidad temporal Armar la matriz a partir de los datos de entrada tiene un costo de $O(|E|)$ Calcular la cantidad de elementos pertenecientes a cada fila tiene un costo $O(n)$. Calcular los valores $SP_{i,j}$ 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 c_i tiene un costo $O(n * k^2)$, pues por cada columna c_i de las n columnas, es necesario verificar los posibles k valores de $(\sum C[0,i])\,\%k$ que se pudieron generar en c_{i-1} y por cada uno de ellos comprobar que valores se pueden lograr sumando con los k posibles valores de los $SP_{i,j}$. Luego verificar si alguno de los valores posibles de $(\sum C[0,i])\,\%k$ pertenece al intervalo $[SP\,\%k - T\,\%k, SP\,\%k]$ tiene un costo $O(k)$. Luego el algoritmo tiene complejidad $O(|E| + n + n * k + n * k^2 + k)$ y por el teorema de la suma, la complejidad final es $O(|E| + n * k^2)$.

5.1. Pseudocódigo

```
def Solve(students: list[Student], k : int):
    matrix = BuildMatrix(students)
    amount_student_P = 0
    amount_student_R = 0
    for i in range(len(matrix[0])):
        amount_student_P += matrix[0][i]
        amount_student_R += matrix[1][i]

    amount_sets_P, rest_P = amount_student_P // k, amount_student_P % k
    amount_sets_R, rest_R = amount_student_R // k, amount_student_R % k

    if rest_P + rest_R < k:
        return amount_sets_P + amount_sets_R

    columns_to_v = []
    all_fluctuations = []
```

```
for i in range(len(matrix[0])):
        if matrix[0][i] + matrix[1][i] >= k:
            columns_to_v.append([matrix[0][i], matrix[1][i]])

if len(columns_to_v) == 0:
    return amount_sets_P + amount_sets_R

for i in range(len(columns_to_v)):
    fluctuations_i = []
    for i in range(max(0, k - columns_to_v[i][1]), min(k, k - max(0, k - columns_to
        fluctuations_i.append(i)
    all_fluctuations.append(fluctuations_i)

list_of_posible_rest = [[0 for i in range(0,k)] for i in range(len(columns_to_v))]

for i in range(len(all_fluctuations)):
    for j in range(len(all_fluctuations[i])):
        if all_fluctuations[i][j] == 0 or all_fluctuations[i][j] == k:
            continue
        list_of_posible_rest[i][all_fluctuations[i][j]] = 1

for i in range(1,len(columns_to_v)):
    for j in range(0,k):
        if list_of_posible_rest[i-1][j] == 1:
            list_of_posible_rest[i][j] = 1
            for z in range(0,k):
                if list_of_posible_rest[i][z] == 1:
                    new_rest = (z + j + 1) % k
                    if list_of_posible_rest[i][new_rest] == 0:
                        list_of_posible_rest[i][new_rest] = 1

for i in range(0,k):
    if list_of_posible_rest[len(list_of_posible_rest)-1][i] == 1:
        if i >= (amount_student_P % k - len(students) % k + k) % k and i <= amount_
            return amount_sets_P + amount_sets_R + 1

return amount_sets_P + amount_sets_R
```