Problema #1: Scooter

September 19, 2024

Contents

1 Problemática

1.1 ¿De qué trata el problema?

El campus de una universidad cuenta con n edificios, numerados desde 1 hasta n. En cada uno de estos edificios puede haber planeada una clase de matemáticas (M), o una clase de ciencia de la computación (CC), o ninguna clase (nunca hay planeadas clases de ambas materias en un mismo edificio). Además de esto, en cada edificio hay inicialmente a lo sumo un profesor, y cada profesor es experto de una de las materias, es decir, que hay profesores de M y de CC.

Como trabajador de University Express Inc., tu trabajo es transportar de manera rápida y amena a los profesores para que puedan impartir sus clases. Para esto se te ha otorgado nada más y nada menos que un scooter (una motorina vamos), en la que cabes tú y a lo sumo un pasajero.

Inicialmente serás la única persona en el scooter. Cuando llegues a uno de los edificios de la universidad puedes dejar o recoger a un profesor en dicho edificio. Para conseguir tu tarea se te ha permitido conducir a cada uno de los n edificios a lo sumo una vez, en el orden que desees.

Al final de tu recorrido, en cada edificio donde haya planeada una clase de M debe haber un profesor de M, mientras que en cada edificio con una clase de CC planeada debe haber un profesor experto en CC. Planea un itinerario de viajes con el que puedan ser impartidas con éxito todas las clases planeadas.

1.2 Entrada

La entrada del problema constará de 3 elementos:

- Un número entero n ($1 \leq n \leq 2000$) que será la cantidad de edificios de la universidad.
- Un string de longitud n conformado por los caracteres (`M`, `C`, `-`), el caracter en la posición i determina la materia de la clase que está programada en el edificio i: `M` representa una clase de M, `C` una de CC y `-` representa que no hay ninguna clase planeada en ese edificio.
- Un string de longitud n conformado otra vez por los caracteres (`M`, `C`, `-`), pero esta vez para definir los profesores que se encuentran inicialmente en cada edificio. Una `M` o `C` en la posición i representa que en el edificio i hay un profesor de M o de CC respectivamente mientras una `-` representa que dicho edificio está vacío.

1.3 Salida

Se debe imprimir en una primera línea un entero l, que representa el número de operaciones que hace el itinerario realizado. Luego se deben imprimir l líneas donde se muestren cuáles fueron las operaciones realizadas. Dichas operaciones pueden ser las siguientes:

- `DRIVE x`: Ir al edificio con el número x ($1 \leq x \leq n$).
- `PICKUP`: Recoge al profesor que se encuentra en el edificio actual.
- `DROPOFF`: Deja al pasajero que se lleve en ese momento en el edificio actual.

Para considerar correcto un itinerario debe cumplir que no se hagan dos instrucciones `DRIVE` hacia el mismo edificio, para cada instrucción `PICKUP` debe haber un profesor en el edificio y ninguno en el scooter, para cada instrucción `DROPOFF` debe llevarse un pasajero en ese momento, no se puede volver a recoger a un profesor que se acaba de dejar en un edificio y por supuesto debe cumplirse la condición de solución del problema, es decir, en cada edificio debe haber un profesor de la materia que se requiera, ya sea porque fue transportado ahí o porque ya estaba en el edificio desde un principio.

2 Solución

2.1 Idea general de la solución

El problema plantea un desafío logístico, donde debemos mover a los profesores de manera eficiente entre los edificios. La solución se basa en un enfoque Greedy, que busca resolver el problema tomando decisiones localmente óptimas en cada paso. Tenemos las variables: n (cantidad de edificios), p (array que indica los profesores en cada edificio) y c (array que indica las clases programadas en cada edificio)

Primeramente vamos a definir como **Desajuste** a un estado del problema que incumple la condición de solución del problema, es decir, un desajuste ocurre cuando el profesor presente en un edificio no coincide con la clase que se imparte en ese mismo edificio, o cuando no hay profesor en un edificio que necesita uno para una clase específica. De manera un poco más formal podemos decir que un desajuste en un edificio i ($1 \leq i \leq n$) ocurre si y solo si alguna de las siguientes condiciones se cumple:

1. **Hay una clase pero no un profesor adecuado:**
    - Si en el edificio i se imparte una clase de M ($c[i] = \texttt{M}$) pero no hay un profesor en dicho edificio ($p[i] = \texttt{-}$) o hay un profesor de CC ($p[i] = \texttt{C}$)
    - Si en el edificio i se imparte una clase de CC ($c[i] = \texttt{C}$) pero no hay un profesor en dicho edificio ($p[i] = \texttt{-}$) o hay un profesor de M ($p[i] = \texttt{M}$)
2. **Hay un profesor pero no se imparte la clase correcta:**
    - Si en el edificio i hay un profesor de M ($p[i] = \texttt{M}$) pero no se imparte una clase en dicho edificio ($c[i] = \texttt{-}$) o se imparte una clase de CC ($c[i] = \texttt{C}$)
    - Si en el edificio i hay un profesor de CC ($p[i] = \texttt{C}$) pero no se imparte una clase en dicho edificio ($c[i] = \texttt{-}$) o se imparte una clase de M ($c[i] = \texttt{M}$)

Por tanto podemos decir que un problema con 0 desajustes está resuelto, ya que si el problema tiene 0 desajustes significa que en cada edificio donde se imparte una clase hay un profesor de dicha materia. Más adelante veremos que hay dos tipos de desajustes que son despreciables, es decir, no es necesario eliminarlos para resolver el problema. El objetivo del algoritmo es eliminar todos estos desajustes necesarios, moviendo profesores entre edificios.

Para esto primeramente se agrupan los edificios en 6 categorias según el tipo de desajuste que presentan (nótese que en la definición de desajuste se plantean 6 condiciones que generan un desajuste), esto nos permite identificar claramente cuáles edificios tienen un problema de desajuste y qué tipo de profesores necesitamos mover.

El enfoque Greedy utilizado resuelve los desajustes uno a uno. Para esto primeramente se lleva un factor de balance que indica cuál de estos dos desbalances es mas frecuente: Profesores de M en edificios con clases de CC, y Profesores de CC en edificios con clases de M. En dependencia de qué desajuste es más frecuente el algoritmo se desarrola de una forma u otra, por supuesto hay un tercer caso que es cuando se da la igualdad en la frecuencia de estos desajustes el cual implica una tercera vía en que se desarrolla el algoritmo. Las 3 vías resuelven los distintos tipos de desajustes en diferentes órdenes, apoyándose en la información inicial que se tiene del problema. Cada una de estas vías se verá con más detalles en las próximas secciones, así como la demostración formal de la correctitud del algoritmo.

2.2 Explicación del código

Luego de leer las entradas del problema (n, c y p) se crea una lista de listas g de 6 elementos, uno para cada tipo de desajuste. La lista g queda conformada de la siguiente manera:

g	$p[i]$	$c[i]$
[0]	C	-
[1]	M	C
[2]	-	M
[3]	M	-
[4]	C	M
[5]	-	C

Table 1: Composición de la lista g con los diferentes tipos de desajuste. En la primera columna está representado el indice de g en donde se guardará el tipo de desajuste que hay en un edificio i si se cumple que los valores de $p[i]$ y $c[i]$ son los que vienen dados en el resto de la fila. Por ejemplo en $g[2]$ se van a guardar los desajustes del tipo: Edificio con clase de M programada sin profesor disponible de ningun tipo.

De ahora en adelante se hará referencia a los tipos de desajuste por números, donde el desajuste de tipo i representa el tipo de desajuste que se guarda en $g[i]$.

En la lista g se van a guardar los índices de los edificios que presenten desajustes, es decir, todos los índices de los edificio que tienen un desajuste de tipo 0 se guardarán en $g[0]$ y asi con el resto. Para rellenar g se recorren los arrays p y c y se van comprobando las condiciones de desajuste. Por supuesto los edificios donde $p[i] = c[i]$ son ignorados pues no presentan ningún desajuste.

Posteriormente en el código aparece la funcion *put* la cual es muy importante pues es la que ejecuta los movimientos en el problema, pero no lo hace de manera literal, es decir, en ningún momento en el problema se modifican los arrays c y p originales con los movimientos de los profesores. Por el contrario la función put cada vez que sea llamada va a extraer un elemento de algún $g[i]$ y esto representará que un movimiento del itinerario será conducir a i y realizar la acción que viene dada por el parámetro ty (type), si $ty = 1$ se realizará la operación `PICKUP`, si $ty = 2$ se realiza la operación `DROPOFF` y si $ty = 3$ se realiza la operación `DROPOFF` seguida de la operación `PICKUP`. *put* utiliza una lista *ans* para ir guardando las operaciones que se hagan de la forma: (i, ty) donde i es el índice de un edificio al que se va a conducir y

ty representa la o las operaciones que se van a ejecutar en dicho edificio. Por ejemplo $(1, 3)$ se traduce como `DRIVE 1`, `DROPOFF`, `PICKUP`. Luego de esto es que se evalúa que vía se usará para resolver el problema. Como funciona exactamente cada caso se puede apreciar en **Code_with_Comments.py** donde aparecen comentadas línea a línea las acciones que se realizan, aquí vamos a dar un repaso general de como funciona cada caso:

- **Caso 1:** Se lleva a cabo cuando la cantidad de desajustes de tipo 1 es mayor que la cantidad de desajustes de tipo 4. En este caso los primeros desajustes que se resuelven son los de tipo 4, para esto se van solucionando de manera alternada un desajuste de tipo 1 y uno de tipo 4 haciendo los viajes de manera que se intercambia un profesor de CC por uno de M en cada edificio con estos tipos de desajuste. Cada vez que se visita un edificio se resuleve el desajuste que haya en este, por lo que nunca es necesario viajar dos veces al mismo edificio. Como los desajustes de tipo 4 son menos estos se resolverán primero, dicho de otra forma $g[4]$ se vaciará primero. Posteriormente se resolverán los desajustes de tipo 1, 5 y 2 en ese orden (los desajustes de tipo 2 pueden quedar resueltos en distintos momentos de la ejecución).

- **Caso 2:** Se lleva a cabo cuando la cantidad de desajustes de tipo 4 es mayor que la cantidad de desajustes de tipo 1. El funcionamiento de este caso es similar solo con variaciones en las prioridades para resolver los desajustes. Por ejemplo en este caso el primer tipo de desajuste que se resuelve es el 1, siguiendo una idea análoga a la del caso anterior. Luego se resuelven los desajustes de tipo 4, 2 y 5 en ese orden (los desajustes de tipo 5 pueden quedar resueltos en distintos momentos de la ejecución).

- **Caso 3:** Este se lleva a cabo cuando la cantidad de desajustes de tipo 4 y de tipo 1 son iguales. Si dicha cantidad es igual a 0 entonces solo se resuelven los desajustes de tipo 2 y 5. En caso contrario se comprueba que existan desajustes de tipo 0, en cuyo caso se resuelven los desajustes de tipo 1 y 4 simultaneamente y siguiendo la misma idea de los casos anteriores (se resuelven a la vez porque hay la misma cantidad) empezando resolviendo un desajuste de tipo 1, es decir, el primer profesor que se va a recoger es de CC. En caso contrario, o sea, si no hay desajustes de tipo 0 se resuelven los de tipo 1 y 4 a la vez pero esta vez empezando por uno de tipo 4, es decir, el primer profesor que se recoge es de M. Una vez terminado esto se resuelven entonces (si no se ha hecho ya) los desajustes de tipo 2 y 5

Nótese que en ningún momento se han intentado eliminar los desajustes de tipo 0 o 3, ya que estos desajustes no violan la condición de solución del ejercicio, es decir, estos son desajustes en los que hay profesores en edificios sin clases programadas y dichos edificios no se toman en cuenta a la hora de resolver el ejercicio, los que importan son los que si tienen clases programadas en los cuales debe haber un profesor de la materia. Por esta razón una vez terminada la ejecución de este algoritmo todos los $g[i]$ $(0 \leq i \leq 5)$ estarán vacíos excepto quizás $g[0]$ y $g[3]$. Esto ocurre porque, como se dijo anteriormente, cada llamado a la función *put* elimina un elemento de un $g[i]$ determinado, o lo que es lo mismo, cada llamado la función *put* reduce en 1 la cantidad total de desajustes. El algoritmo está dividido en bloques *while* que chequean si todavía existen desajustes de un tipo en específico y en todos los casos vistos anteriormente siempre se chequearon todos los tipos de desajustes excepto los despreciables para la solución (0 y 3), por lo que una vez terminado el algoritmo en *ans* se encuentra la secuencia de pasos correcta y óptima para resolver el ejercicio.

Finalmente se itera por los elementos de *ans* y se muestra la cantidad de pasos que se necesitaron para solucionar el problema seguido de las operaciones que se realizaron en orden.

2.3 Demostración formal

Para demostrar la correctitud del algoritmo debemos demostrar que devuelve un itinerario que cumpla con la condición de solución del ejercicio, y que además es óptimo. Esto implica demostrar que en cada paso se reduce el número de desajustes del problema y que se hace en un orden correcto. Como vimos anteriormente hay dos tipos de desajuste (el 0 y el 3) que no necesitan ser necesariamente eliminados para resolver el ejercicio y explicamos por que, luego la función *put* se ejecuta repetidas veces hasta que se cumple que $(len(g[1]) = len(g[2]) = len(g[4]) = len(g[5]) = 0)$, es decir, hasta que se han eliminado todos los desajustes importantes para resolver el problema, y como se dijo antes, un problema con 0 desajustes no despreciables está resuelto.

Vamos a demostrar entonces que para cada caso de ejecución se lleva a cabo una secuencia válida de pasos donde en cada uno se elimina un desajuste. Justificaremos y demostraremos además algunas deciciones que se toman en cada uno. Recordemos los tipos de desajustes (Tabla de Desajustes) pues serán un elemento bastante recurrente en la demostración.

2.3.1 Caso 1

En este caso $len(g[1]) > len(g[4])$. Lo primero que se hace es recoger a un profesor de CC en $g[0]$, demostremos que siempre es posible encontrar a un profesor de CC en $g[0]$ en este caso:

Como $len(g[1]) > len(g[4])$ hay al menos un profesor de CC que está mal asignado pero no en $g[4]$. Visto de otra forma: Sea $l_1 = len(g[1])$ y $l_2 = len(g[4])$, esto significa que en el problema hay al menos l_1 profesores de M mal asignados pero además l_1 edificios con clase de CC desajustados, por otro lado tenemos al menos l_2 profesores de CC mal asignados pero $l_1 > l_2$ por tanto deben haber al menos $l_1 - l_2$ profesores de CC disponibles para cubrir todos los l_1 edificios desajustados. Estos profesores solo pueden estar en $g[0]$ pues la otra posibilidad es que estén en edificios correctos ya, lo cual es imposible ya que si no no habría suficientes profesores de CC para resolver el ejercicio. Luego siempre es posible en este caso encontrar al menos un profesor de CC en $g[0]$.

Una vez se recoge este profesor el edificio queda vacío y sin clases, por lo que no será necesario regresar a él para solucionar el problema. Con este profesor recogido se procede a realizar el siguiente bucle:

Se va a un edificio con clase de CC que tiene un profesor de M, para dejar al profesor de CC que se está llevando y recoger el de M. Como se dejó un profesor de CC en un edificio de CC no es necesario regresar a él para resolver el ejercicio. Se lleva entonces este profesor de M a un edificio donde sea necesario y haya en este un profesor de CC, el cual se recoge. Por un razonamiento análogo al caso anterior ya no es necesario regresar este edificio. Nótese que en cada iteración de este bucle se resuelve primero un desajuste de tipo 1 y luego uno de tipo 4.

Este bucle se realiza hasta que se han resuelto todos los desajustes de tipo 4 que, como son menos que los de tipo 1 no habrá problemas. Al final del bucle nos quedamos con un profesor de CC en el scooter el cual es llevado a otro edificio con desajuste 1, como los desajustes de tipo 1 eran más que los de tipo 4 podemos garantizar que al término del bucle anterior queda al menos un desajuste de este tipo. El profesor de M de este edificio al que acabamos de ir es recogido solo si hay lugar al que llevarlo, es decir, si hay desajustes de tipo 2 (porque ya sabemos que de tipo 4 no hay) en cuyo caso lo recogemos y llevamos, reduciendo en uno los desajustes de tipo 2. Si no existieran desajustes de este tipo no es necesario recogerlo, ya que no existen en el problema más edificios que necesiten profesores de M, por lo que lo dejamos junto con el profesor de CC que acabamos de dejar. Esto no incumple la condición de solución del ejercicio pues hay un profesor de CC

en un edificio de CC, sin importar que haya otros en el mismo edificio.

Terminado esto se procede a resolver todos los desajustes de tipo 1 que resten (puede no restar ninguno) recogiendo profesores de edificios que no tienen clases, por la demostración hecha al principio de este caso se puede deducir que a estas alturas habrá un profesor de CC en un edificio sin clases por cada desajuste de tipo 1 que quede. Es fácil ver que todos los edificios que hemos visitado ya no están desajustados, por lo que no será necesario volver a ninguno de ellos para resolver el ejercicio.

Finalmente se resuelven los desajustes que pueden quedar (2 y 5) de una manera sencilla. Estos son desajustes en edificios donde no hay profesores por lo que solo debemos recoger profesores de edificios que no tienen clases y llevarlos a los lugares correspondientes. Como a estas alturas ya no quedan desajustes de tipo 1 ni 4 es fácil ver que para cada desajuste restante habrá un profesor en un edificio vacío, ya que no pueden estar en otro lugar y tienen que existir sino el problema no tendría solución.

Luego queda demostrada la correctitud del caso 1.

2.3.2 Caso 2 (análogo)

Este caso funciona de una manera totalmente análoga al caso 1. Esta vez $len(g[1]) < len(g[4])$ por lo que el profesor inicial se recoge de $g[3]$ y no de $g[0]$ pero la demostración es la análoga a la que se realizó en el caso 1. Toda la ejecución ocurre de manera similar tomando en cuenta que esta vez los desajustes de tipo 1 se resuelven primero, el profesor al final del bucle inicial no es de CC sino de M, por lo que se deja (si es posible) en un desajuste de tipo 4. El profesor que se considera si recoger o no es de CC y no de M, por lo que se valora si se puede llevar a un desajuste de tipo 5 y no 2 y el resto de la ejecución ocurre de manera similar.

Por lo tanto de la correctitud del caso 1 puede deducirse la correctitud de este caso, al ser casos análogos.

2.3.3 Caso 3

En este caso $len(g[1]) = len(g[4])$. Primero se chequea si la cantidad de desajustes de este tipo es 0, en cuyo caso solo es necesario resolver desajustes de tipo 2 y 5, lo cual se hace de manera similar y en las mismas condiciones que en los casos anteriores. En caso contrario se procede de la siguiente manera:

Si hay desajustes de tipo 0 entonces están dadas condiciones similares al caso 1, por lo que se procede de la misma forma, con la única diferencia de

que como $len(g[1]) = len(g[4])$, al término del bucle inicial nos quedamos con un profesor en el Scooter al que no necesariamente podamos ubicar, por lo que si es posible se ubica (en un desajuste de tipo 5 al ser un profesor de CC), y si no no se recoge directamente en el último paso del bucle, como vimos anteriormente esto no afectará a la resolución del ejercicio.

Si no hay desajustes de tipo 0 entonces tienen que existir obligatoriamente desajustes de tipo 3, demostremos esto:

Lo que vamos a demostrar es que si existen desajustes de tipo 1 y 4 tiene que haber al menos un desajuste de tipo 0 o 3. Supongamos lo contrario y vamos a decir que no existen desajustes de tipo 0 ni 3 dado que si existen de tipo 1 y 4. Si se diera este caso entonces los únicos lugares en donde podemos recoger profesores son edificios con clases asignadas, por tanto si recogemos a un profesor de un edificio i este quedará vacío pero con una clase programada, por lo que estará desajustado, pero como no se puede realizar un comando `DRIVE` dos veces al mismo edificio si recogemos un profesor aquí no podremos regresar y el edificio permanecerá desajustado, lo que significaría que el problema no tiene solución. Luego por reducción al absurdo queda demostrado que si existen desajustes de tipo 1 y 4 entonces tambien debe haber al menos 1 de tipo 0 o 3. De esta demostración puede deducirse además que es necesario resolver desajustes de tipo 0 o 3 (aún siendo despreciables) para la resolución del resto.

Siguiendo con el algoritmo, como hay desajustes de tipo 3 se tienen condiciones similares al caso 2 y se procede entonces de una manera acorde a lo que se vió anteriormente. Finalmente una vez eliminados todos los desajustes de tipo 1 y 4 se procede igualmente a eliminar los desajustes de tipo 2 y 5 como mismo se explicó al inicio de la demostración de este caso.

Luego queda demostrada la correctitud de este caso.

2.3.4 Cerrando la demostración

Como hemos podido observar el algoritmo resuelve el problema de manera óptima y correcta. Cumple con las restricciones del problema dejando a los profesores correctos en los edificios necesarios al final de la ejecución, no realiza operaciones `DRIVE` dos veces al mismo edificio ya que cada vez que sale de un edificio este ya no está desajustado, y siempre realiza las operaciones de `PICKUP` y `DROPOFF` en el orden correcto por la manera en que se arma la respuesta con la función *put*. Además la respuesta proporcionada es la óptima ya que el algoritmo reduce en cada paso el número de desajustes en 1, y por la estructura del problema no es posible resolver dos desajustes en un solo paso (por ejemplo no se puede transportar a más de un pasajero

en el scooter). No se realizan pasos adicionales que no contribuyan a corregir un desajuste. Por ejemplo, el algoritmo no mueve a profesores entre edificios que ya están correctamente emparejados. Para cada desajuste se necesita una operación `PICKUP` y una `DROPOFF`, el algoritmo hace exactamente una vez estas operaciones por cada desajuste por lo que no puede haber otro algoritmo que resuelva el problema en una menor cantidad de pasos. Además se puede garantizar que el algoritmo siempre termina pues la cantidad inicial de desajustes es finita y el algoritmo la va reduciendo progresivamente.

2.4 Compejidad temporal y espacial

Al principio se recorren los arrays p y c una sola vez para rellenar g, esto cuesta $O(n)$. Por otro lado el número máximo de desajustes posible es de n (no pueden haber más desajustes que edificios), y el algoritmo resuelve un desajuste por paso y no repite edificio, por lo que el costo del algoritmo Greedy es también $O(n)$. Luego la complejidad temporal total del algoritmo es de $O(n)$, siendo este el costo de todas las operaciones que este realiza.

Los arrays de entrada ocupan $O(n)$ en memoria. La lista g que almacena los edificios según su desajustes, es una lista de listas pero cada edificio puede pertenecer a lo sumo a solo una de ellas (no puede haber un edificio con dos desajustes), por lo tanto la suma de las longitudes de cada $g[i]$ es también $O(n)$. La lista que almacena la respuesta ans aumenta en uno su longitud por cada llamado a la función put la cual se llama una vez por cada desajuste, como el numero de desajustes es a los sumo n la longitud de ans es tambien $O(n)$. Por tanto la complejidad espacial total del algoritmo es $O(n)$, al ser esta la complejidad espacial de todas las estructuras de datos que se utilizaron.