ELIGIENDO CARRERAS

AUTOR: LAUTARO LASORSA

Solución Esperada

En la solución esperada se utiliza la estructura de datos **Trie**[1], tener en cuenta que cada nodo representa un prefijo de una o más de las palabras (en este caso cada una de las carreras). Que los elementos de los vectores sean enteros de 1 a 1.000.000.000 fuerza, en la implementación, a usar un **map** o una estructura similar en cada nodo para indicar los nodos siguientes.

La idea en este caso es que cada nodo del **Trie** contenga (en un set, por ejemplo) que carreras de Ana y de Joe (por separado) pasan por él, además de saber su profundidad (la cantidad de cuatrimestres que comparten las carreras que pasan por él).

Además, debemos tener un **set** global donde se guarden pares de la forma **{profundidad, nodo}**, y utilizando la función ***set.rbegin()** podemos acceder al último elemento de la estructura, es decir el nodo que tiene mayor profundidad.

Entonces, en cada actualización puede que a un nodo le agreguemos o le quitemos una carrera de alguno de los 2 conjuntos. Después de cada actualización, si el nodo tiene al menos una carrera en cada conjunto lo agregamos al **set** (si ya está la estructura sola va a evitar que haya duplicados) y si en alguno de los 2 no hay ninguna carrera lo borramos del **set** (si no existía previamente no habrá problema, de nuevo por cómo funciona el **set** por dentro)

Finalmente, después de cada actualización **el set estará vacío si y sólo si alguno de los 2 no sigue ninguna carrera** (ya que el vector vacío es prefijo de cualquier carrera), por lo que en este caso devolveremos que es imposible.

Sino, debemos mirar el mayor elemento del **set**, esto nos indicará cuál es la máxima profundidad posible y un nodo con el que se pueda lograr dicha profundidad. Luego, es solo ir a ese nodo y tomar una carrera del conjunto de Ana y otra del conjunto de Joe, y listo.

Así, la complejidad queda **T*log(T)**.

Soluciones Parciales

[1] https://es.wikipedia.org/wiki/Trie

ELIGIENDO CARRERAS

AUTOR: LAUTARO LASORSA

La primer subtarea se espera sea resuelta simplemente teniendo un conjunto con las carreras de Ana y otro con las de Joe (que puede ser un array, un **set**, etc.) y después de cada actualización compare todas las carreras de Ana con todas las de Joe. La complejidad de esta solución llega a ser $\mathbf{T^3}$.

Para la segunda subtarea, se espera que cada vez que se procesa una carrera se vean todos sus prefijos, y en un **map** se indica para cada prefijo si Ana / Joe gustan de ese prefijo, y guardando el conjunto de carreras mediante las cuales lo hacen. Al igual que en la solución general, cada vez que se cambia un prefijo se debe actualizar un **set** para poder acceder al óptimo rápidamente. La complejidad de esta solución es $\mathbf{T^2 * log(T)}$. Tener en cuenta además que un **map** tiene una mala constante.

Para la tercera subtarea ya es necesario armar la estructura **Trie** esperada en la solución completa, pero como en cada actualización solo se agregan carreras, la respuesta nunca decrece (es decir, la profundidad máxima después del **i-ésimo** update es igual o mayor a la que había después del update anterior), por lo tanto lo único que es necesario es actualizar un máximo con la respuesta en lugar de usar un **set** para obtener el máximo de un conjunto donde pueden agregarse o quitarse elementos. La complejidad en este caso es **T**.