# Editorial Tarea 1

Profesor: Andrés Abeliuk
Auxiliares: Daniel Báez
Julieta Coloma
Máximo Flores Valenzuela
Blaz Korecic
Diego Salas

## A. Quiero mis UB's!

Vamos a ver como se resuelve este interesante problema, probablemente el mas difícil de la tarea, y sin embargo, luego de entender la solución, es bastante fácil de implementar.

Vamos a resumir el camino a la solución en cuatro importantes observaciones.

- Con fines prácticos, se asume que, lo único que nos importa de los nombres de los departamentos es cual seria su posición en una lista si la ordenáramos por orden alfabético, por lo que el resto de la solución asumiremos que los nombres son tan solo números, en el ejemplo del enunciado, asumiríamos que:

$$\textbf{DIE} \rightarrow 1$$

$$\textbf{DIM} \rightarrow 2$$

$$\textbf{DCC} \rightarrow 0$$

- Si decido adoptar la estrategia de ir construyendo mi solución desde la primera hasta la ultima posición, debo priorizar la posición en la que voy por sobre las que me quedan (por como funciona el orden lexicográfico), por ende, para construir la solución correctamente, solo debo responder en cada índice a la pregunta: "cual es el mayor valor que puedo traer a esta posición", siempre considerando los cambios ya realizados a la lista durante la ejecución del algoritmo.
- Dada una lista de números, y sus respectivos amigos, puedo llevar un numero desde una posición $i$ de la lista al inicio de esta si y solo si todos los valores de la lista desde el inicio hasta $i$ son amigos del numero.

  En el ejemplo (recordemos que cambiamos los nombres por números para efectos prácticos) si quiero llevar el 100 al inicio de la lista, solo podre lograrlo si todos los números anteriores a el son amigos.

  1 2 3 4 100 5 6 7

  Si alguno de los números entre el 1 y el 4 no son amigos del 100, este nunca podrá pasarlos a todos, por lo que jamas podrá ser el primero de la fila.

- Si deseo llevar el numero mas grande posible de una lista al inicio de esta, solo me interesan los representantes de cada numero mas adelantados en la fila, ya que si esta fuese:

  $$1 \quad 2 \quad 3 \quad 4 \quad 100 \quad 5 \quad 6 \quad 100$$

  Y yo quisiera llevar un 100 al inicio de la fila, que yo pueda llevar el 100 ubicado en la ultima posición implica que yo puedo llevar el 100 que esta quinto, ya que se debe cumplir que los números anteriores a este son amigos de 100 en ambos casos.

Ya con esto, se propone el siguiente algoritmo: Por cada posición de la fila, de inicio a fin, traeremos el mayor valor que podamos, y al ubicarlo al comienzo de la fila, quedara fijo en esa posición y podremos ignorarlo en adelante, por lo que para la siguiente iteración, traeremos el mayor valor posible al nuevo inicio de la fila, la cual redujo su tamaño en 1 (ya que es como si hubiésemos sacado el valor que llevamos al inicio de esta).

Siendo la cantidad $S$ de departamentos un numero no tan grande (200 como máximo), tendremos una cola por cada departamento, la cual nos indicara si puedo llevar el numero $i$ al inicio de la fila de la siguiente forma:

Llenamos la cola con los valores de la fila original, pero iremos sacando los números que van delante de $i$ y que le privan de llegar al inicio de la fila. Sacaremos un numero cuando ya fue utilizado en la construcción de la solución, o es amigo de $i$, en ambos casos su presencia nos deja de importar.

Iremos desde la colas que representan los números mas grandes hasta los mas pequeños, si podemos agregar un numero al inicio, lo hacemos (al hacerlo en orden, sabemos que es el mas grande que posiblemente que se pueda llevar al inicio) y vamos actualizando los valores de la cola como se menciona anteriormente.

**Complejidad:** $\mathcal{O}(S \cdot N)$. Por cada iteración, vamos eliminando (y no agregando) valores de las colas. Al ser $S \cdot N = 200 \cdot 100000 = 2 \cdot 10^7$ elementos dentro de las colas en el peor de los casos, y su complejidad estar dada por esta cantidad, sabemos que la solución pasa por tiempo.

## B. Sapo y Sepo tienen hambre

Tenemos que encontrar el tamaño máximo de un subarreglo sin repeticiones.

La idea es que mantendremos nuestro subarreglo entre dos índices $l$ y $r$, e iteraremos por todos los $r$ posibles de izquierda a derecha.

Inicialmente, $l = r = 0$ y el subarreglo tiene tamaño 1. ¿Qué pasa cuando avanzamos el $r$ a $r + 1$? Hay dos posibilidades:

- $a_{r+1}$ no está en nuestro subarreglo entre $l$ y $r$: en este caso simplemente lo agregamos y el tamaño aumenta en 1.

- $a_{r+1}$ está entre nuestros números anteriores: como tendríamos una repetición, tendremos que aumentar el $l$ quitando números por la izquierda hasta que deje de estar repetido, de manera que podamos agregarlo sin generar una repetición.

Este algoritmo es correcto, pues para cada $r$ encontramos el $l$ más a la izquierda posible tal que no hayan repeticiones, y si probamos todos los $r$ y calculamos el máximo entre los tamaños, obtendremos la respuesta.

Ahora, necesitamos alguna estructura que nos permita:

- Guardar los números de nuestro subarreglo actual.
- Consultar si un número está en la estructura.
- Insertar y eliminar números.

**Complejidad:** El multiset es capaz de realizar todas las operaciones que requerimos en $\mathcal{O}(\log n)$.

Además, los índices $l$ y $r$ pueden aumentar un máximo de $\mathcal{O}(n)$ veces cada uno, así que los ciclos que los aumentan tienen complejidad $\mathcal{O}(n)$ en total. Juntando esto con las operaciones en el multiset, nuestra solución será $\mathcal{O}(n \log n)$, logrando el límite de tiempo.

## C. Watchmen

*Este problema es una mezcla de geometría, STL y álgebra.* Sean $d_M$ y $d_E$ la distancia de Manhattan y Euclides respectivamente. Buscamos $d_M = d_E$. Desarrollemos esta condición, para $(x_i, y_i)$ y $(x_j, y_j)$ posiciones arbitrarias:

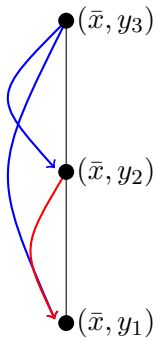
$$\begin{aligned} d_M = d_E &\iff |x_i - x_j| + |y_i - y_j| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \\ &\iff (x_i - x_j)^2 + 2|x_i - x_j||y_i - y_j| + (y_i - y_j)^2 = (x_i - x_j)^2 + (y_i - y_j)^2 \\ &\iff 2|x_i - x_j||y_i - y_j| = 0 \end{aligned}$$

Es decir, las distancias son iguales ssi $x_i = x_j \vee y_i = y_j$. Esta condición la podemos leer como "comparten alguna coordenada ($x$ ó $y$), o están en la misma posición".

Fijemos una coordenada $\bar{x}$ y supongamos que $k$ personas la comparten. Se puede demostrar que las combinaciones totales que satisfacen la condición del enunciado ($d_M = d_E$) es:

$$\mathcal{S} = \sum_{i=1}^{k-1} i = \frac{(k-1) \cdot k}{2}$$

Con la siguiente intuición (para $k = 3$, p. ej.):




Donde la persona con la mayor coordenada en $y$ (en este caso $y_3$), **siempre** cumplirá la condición con las $k-1$ de abajo, luego la segunda con mayor $y$ ($y_2$) la cumplirá con las $k-2$ de abajo, y así hasta completar la suma $\mathcal{S}$. Cualquier otra flecha que no esté en el diagrama es un caso repetido.

Dicho esto, podemos usar 2 mapas para contar cuántas veces se repite cada componente $x$ e $y$ de las coordenadas que aparecen en el input. Luego, recorremos cada mapa (explicado aquí) y a la respuesta le sumamos $\mathcal{S}$ por cada componente que esté guardada *(Obs.: No importa si la componente aparece sólo* 1 *vez (i.e., no se repite), pues en dicho caso* $\mathcal{S} = 0$*).*

Lo último que tenemos que considerar es que se pueden repetir coordenadas. Estos casos no los debemos contar en la respuesta final. Podemos guardar en otro `map` las repeticiones de cada par $(x, y)$ que aparezca en el input, y usando la misma fórmula para $\mathcal{S}$, restarlo de la respuesta final.

**Complejidad:** $\mathcal{O}(n)$, pues recorremos linealmente las $n$ coordenadas.

## D. Arreglo mínimo

La observación importante es que la prioridad es minimizar el primer elemento del arreglo $c$, luego minimizar el segundo, el tercero y así, de izquierda a derecha, porque un arreglo que tiene la primera posición menor que otro será lexicográficamente menor sin importar el resto.

Intentemos minimizar $c_1$ (indexando desde 1). Como $a_1$ está fijo, tenemos que escoger qué elemento de $b$ ponemos primero. Óptimamente, lo mejor sería escoger $b_1 = n - a_1$, de forma que $(a_1 + b_1)\%n = 0$. Si no hay un $n - a_1$ en el arreglo $b$, nuestra siguiente opción óptima sería $n - a_1 + 1$, donde $c_1$ sería 1, y así.

En general, cuando escogemos qué elemento de $b$ colocar en la $i$-ésima posición, el que minimiza $c_i$ será el primer elemento de $b$ que sea mayor o igual a $n - a_i$.

Así que colocaremos todos los elementos de $b$ en un multiset. Para crear la permutación, iteramos de izquierda a derecha y en cada iteración encontramos el primer elemento de $b$ que sea mayor o igual a $n - a_i$ usando `lower_bound` y de esa manera construimos la respuesta.

Cuidado cuando no existe un número mayor o igual a $n - a_i$. En ese caso, lower bound retorna el iterador `end` y lo que más nos conviene es simplemente elegir el menor número de los que queda (el que está en `begin`).

**Complejidad:** $\mathcal{O}(n \log n)$, ya que iteramos por las $n$ posiciones y en cada una realizamos $\mathcal{O}(\log n)$ operaciones para encontrar el número que colocaremos.