Diferencias principales entre **LinkedList** y **ArrayList**

Una de ellas es la implementación interna de cada una. La clase **LinkedList** emplea una lista doblemente enlazada, mientras que **ArrayList** utiliza un arreglo que se redimensiona de forma automática dependiendo de si ejecutan inserciones o extracciones de datos.

Otra diferencia es que, mientras que **ArrayList** tiene un acceso directo a los elementos mediante el método *get()*, **LinkedList** requiere recorrer secuencialmente la lista hasta llegar a la posición buscada.

Se utiliza mas un ArrayList si no se va a tener gran cantidad de inserciones/extracciones durante la ejecución de un programa.

Dicho todo esto, decidimos utilizar **LinkedList** dado que el volumen de datos que pide el problema es de hasta 400000 (cuatrocientos mil). Utilizar un **ArrayList** nos hubiese perjudicado la eficiencia del algoritmo, dado que, al eliminarse un elemento, se debe de redimensionar internamente el arreglo, llegando al peor caso de tener un costo de O(n) si se quiere eliminar el primer/último elemento de dicho arreglo. Mientras que con **LinkedList**, dicho costo sería de O(1) porque solo tiene que redirigir los punteros anterior y siguiente del próximo elemento. Aunque también sabemos que dichos punteros generan un costo en tamaño para cada elemento, pero aun así nos pareció la implementación que mas encaja con nuestro algoritmo.

Calcular las pasadas

Decidimos realizar el calculo del resto para obtener la posición del OIA a ser eliminado. Este algoritmo nos permite capturar dinámicamente el OIA buscado a medida que se vayan eliminando elementos de la LinkedList. Nos garantiza nunca pasarnos por fuera de la lista de OIAS, y además arrancar el próximo calculo desde la position del OIA siguiente al que fue eliminado anteriormente.

La alternativa era un puntero con un for interno que vaya calculando de manera secuencial la posición del aventurero a ser eliminado. Esto generaría un costo mayor, dado que, dentro de dicho for, tendríamos que preguntar constantemente si el puntero se pasó de posición con respecto al arreglo (es decir, contemplar el caso de un IndexOutOfBounds). Además, esta operación nos haría recorrer la lista de manera tal que se pueden llegar a leer varias veces los mismos elementos en una sola pasada.

Escritura de los Aventureros en el Archivo

Para el grabado de los datos en el archivo de salida, decidimos ir guardándolos en el archivo a medida que se iban eliminando de la **LinkedList**. Esto nos ahorra memoria con respecto a la alternativa, que era implementar una lista que vaya guardando por orden aquellos aventureros que fueron siendo eliminados a medida que se daban las pasadas. Esto involucraba tener que recorrer nuevamente una lista tamaño N (siendo N el tamaño de la lista de OIAS) para ir grabando secuencialmente los datos en el archivo de salida.

Optamos por la solución ofrecida previamente porque es de menor costo para el procesamiento de archivos de entrada con gran cantidad de datos. Tener que recorrer dos veces una lista de 400000 elementos (siendo este el peor de los casos) no hubiese sido eficiente.