PROYECTO DE PROGRAMACION III

COVID 19

Aquilino Mendizábal, Octavio Clave 2008893

Rojo Justicia, Nicolás Clave 2000262

Vietto Herrera, Joaquín Clave 2006111

UCC 2021 INGENIERÍA

Introducción

Se nos planteó como objetivo realizar un visualizador de datos de los casos de COVID en Argentina. El programa será comandado por línea de comandos para realizar determinadas tareas, y será leído y procesado para brindar una información específica.

Las funciones a desarrollar son “p_casos” , “p_muertes”, “estad”, “casos_cui” y “casos_edad”. Cada una de ellas cumple con una tarea en particular, que a lo largo del informe se irá explicando.

Aplicando los conocimientos adquiridos a lo largo del semestre, se comenzó a armar el proyecto.

Funciones y sus desarrollos:

- Se comenzó por desarrollar el MAIN con el objetivo de recorrer la lectura del archivo .csv para que se lean los datos. Se tuvo que hacer un filtrado tanto de comillas como de comas, para que la información del archivo pueda ser leída de forma limpia, y sea útil para las funciones que se debían ejecutar. Además, se agregó el token “NA” en los casos donde no había ningún dato cargado en el archivo.

- Luego, se continuó con **p_casos**. Esta función tiene como tarea mostrar la cantidad de casos por provincia en orden descendente. En una sintaxis muy similar, se encuentra la función **p_muertes**, cuya tarea es mostrar la cantidad de muertes por provincia, también en orden descendente como hace la función p_casos. Se implementó el ordenamiento conocido como QUICKSORT (algoritmo que permite ordenar n elementos en un tiempo proporcional a n log n), el cual resulto muy cómodo ya que para hacer el conteo de las muertes y los casos se implementaron arreglos de 25 posiciones. Para poder hacer que quicksort ordene los nombres de las provincias con su correspondiente cantidad de muertes o casos, se debió cargar estos dos valores en una misma lista (de dos nodos) y definir el array para que acepten las listas. Dentro del quicksort se generaron modificaciones para que tenga en cuenta la cantidad de casos a la hora de ordenar, en lugar del nombre de la provincia.

  En un primer momento en las funciones pcasos y casos cui pmuertes se hacia utilización de dos funciones distintas, una con parámetros y otra sin, con el fin de reducir la cantidad de if en la función sin parámetros para hacer el código mas eficiente. Pero haciendo un balance, nos pareció mejor manejarnos con una sola función con el fin de evitar la repetición de códigos muy similares.

- Por otro lado, como se mencionó al comienzo, se encuentra la función **Casos_edad**. La cual muestra los datos de todas las personas cuya edad fuese pasada como argumento, ordenadas según el ID de cada provincia. Esta fue una de las funciones que más dificultades generó. Se implementó la estructura de árbol AVL, a la cual se le tuvieron que modificar los headers para lograr un funcionamiento óptimo para la tarea particular que se debía realizar, y luego de mucha prueba y error se pudo ejecutar la función con éxito. Hubo problemas al momento de correr la función “stoi()” (función que convierte un string a variable int) debido al filtrado de los datos del que dependía.

  En un principio se pensó en utilizar TABLA DE HASH para esta función, pero se optó finalmente por ARBOL AVL, al ser más eficiente en su utilización. Para **casos_cui**, que es la función que determina según una fecha específica la cantidad de cuidados intensivos y todos sus datos, que hubo desde dicha fecha en adelante, también se implementó ARBOL AVL ya que demostró ser la más eficiente. Para este caso, también se volvió a modificar el header de la estructura para que sea útil en lo que se necesitaba resolver.

- Para la función **estad**, cuya tarea es mostrar datos particulares como porcentajes de muertos, porcentajes de contagiados, cantidad de muertos y contagiados según rango etario, entre otras, se optó por utilizar vectores, ya que demostraron buenos resultados en eficiencia.

Adaptación del árbol AVL

En una primera instancia se comenzó por modificar el nodo del árbol, el cual en su interior debía tener una lista correspondiente a cada fila del archivo. Así se le agrego un método getClave que devolvía la posición de la lista que se debía tener en cuenta a la hora de introducir en el árbol (en el caso de caso cui era la fecha de cuidado intensivo, y en el caso de la edad era el id de la provincia). Y finalmente se introdujo el método print, encargado de imprimir la lista que contenía.

Luego se pasó a modificar el arbol.h, el cual al igual que el nodo, debía aceptar listas. En este header principalmente se modificó el método de put, encargado de introducir los nodos de la forma correcta, acomodándolos a la izquierda si la clave (proveniente de getClave) de la raíz es mayor o igual que la del nodo a colocar. Caso contrario, se acomoda hacia la derecha. Y una leve modificación en la condición de rotación a la derecha, para que funcione con casos en que la clave sea

igual. Además, la función inorder implementa la utilización del método print creado en el header del nodo.

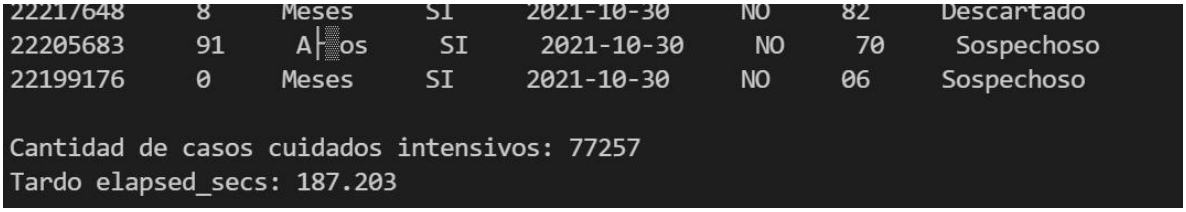
Eficiencia-testeos-pruebas

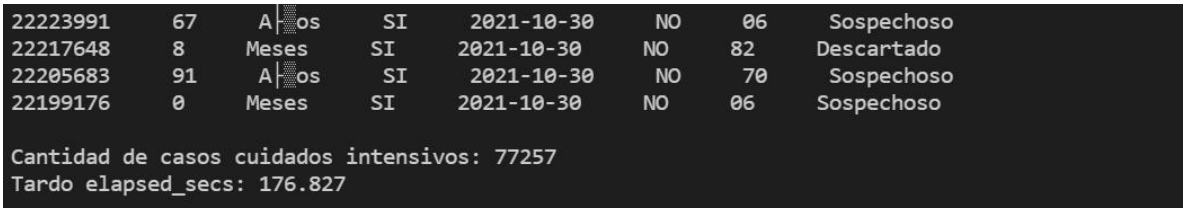
Utilizando QUICKSORT, ARBOL AVL y ARBOL BINARIO, se fueron testeando las distintas funciones para comprobar su eficiencia. A lo largo de las pruebas los tiempos fueron variando, e incluso en ocasiones, cuando los datos no eran muchos, el quicksort generaba tiempos muy similares al árbol AVL. Pero cuando el tamaño de los datos aumentaba, el árbol AVL generaba una gran eficiencia. Además, nos pareció interesante implementar la estructura de árbol AVL debido a que abre muchas más posibilidades en su implementación que si se usa el quicksort, que se limita solo a ordenamiento.

Otro aspecto que pudimos observar haciendo las pruebas, es que cuando al árbol binario se le cargaban una gran cantidad de datos, generaba problemas en la ejecución, incluso cortándola en algunos casos. Cuestión que con AVL no sucedía. Ya que este optimizaba el ordenamiento de los mismos.

Todas las funciones fueron medidas según el tiempo, con la librería <ctime>.

A continuación, se dejan algunos ejemplos de pruebas que se fueron ejecutando:





```
22217648     8     Meses     SI     2021-10-30     NO     82     Descartado
22205683     91     A├▒os     SI     2021-10-30     NO     70     Sospechoso
22199176     0     Meses     SI     2021-10-30     NO     06     Sospechoso

Cantidad de casos cuidados intensivos: 77257
Tardo elapsed_secs: 154.508
```

```
22223991     67     A├▒os     SI     2021-10-30     NO     06     Sospechoso
22199176     0     Meses     SI     2021-10-30     NO     06     Sospechoso
22205683     91     A├▒os     SI     2021-10-30     NO     70     Sospechoso
Cantidad de casos cuidados intensivos: 77257
Tardo elapsed_secs: 152.201
```

Conclusión

Como conclusión se podría destacar las diversas posibilidades e implementaciones posibles que existen para realizar una misma tarea. Además, mencionar también que los conocimientos adquiridos fueron puestos a prueba con éxito, demostrando así la utilidad y el potencial que poseen las estructuras de datos. Con este tipo de trabajos se abre la posibilidad de ejercitar la lógica de programación, aplicada a casos reales y puntuales como lo es en este caso el COVID, virus que ha cambiado la realidad del presente.