# Introducción

El problema que se busca resolver trata sobre el manejo de datos de un centro de vacunación de la comuna de Maipú en donde se requiere controlar información relevante relacionado a las personas que acuden a inocularse, específicamente, requiere conocer la información relacionada con las personas registradas que tienen su primera dosis, y, a partir de el tiempo que se debe esperar entre la primera dosis y la segunda dosis de cada vacuna, conocer la fecha exacta que le correspondería a la persona volver a inocularse con la segunda dosis. Junto con lo anterior, se busca obtener información respecto al stock necesario de cada vacuna para cada mes y un registro de las personas las cuales ya han recibido la primera y segunda dosis.

Los requisitos para el problema es entregar todos los datos ordenados por fecha de forma cronológica y ordenada por el apellido del paciente, dichos datos deben mostrarse en archivos siguiendo el formato dado por el centro de vacunación. Además, el problema se resolverá utilizando listas enlazadas y la proposición de 2 TDA´s, uno para los pacientes y otro para las vacunas

# Solución propuesta

Bien pronunciado en la introducción, para resolver este problema se utilizaron listas enlazadas y estructuras de datos, a continuación, se mostrará brevemente las estructuras de datos utilizadas:

typedef struct nodoGenerico{

char \*rut;

char \*nombre;

char \*apellido;

int age;

int day;

int month;

int year;

int id;

int day2;

int month2;

int year2;

char \*segundaDosis; // dd/mm/yyyy

struct nodoGenerico\* siguiente;

}nodo;

typedef struct listaGenerica{

nodo\* inicio;

}TDAlista;

typedef struct nodoVacunaGenerico{

int id;

char \*codigo;

char \*nombre;

int periodo;

int dosis;

struct nodoVacunaGenerico\* siguiente;

}nodoVacuna;

typedef struct listaGenericaV{

nodoVacuna\* inicio;

}TDAlistaVacuna;

TDAlista está pensado para guardar la información de los pacientes, siendo esta su rut, nombre, apellido, edad, fecha 1ra dosis, y el ID de la vacuna que se le proporcionó, el resto de variables es para almacenar la fecha de su segunda dosis.

TDAlistaVacuna corresponde a una lista la cual tiene en mente la información cada vacuna, la cual correspondería al ID de la vacuna, el código, el nombre, el periodo que hay que esperar para aplicar la segunda dosis y un contador de vacunas requeridas por mes.

Leer los archivos no requiere tanta explicación, siendo esto bastante trivial, por lo que comentare el protocolo de como se llegó a los 3 archivos de salida

actualizarFechaSegundaDosis(entrada,out,vacunas)

FILE archivo

listaDia<-crearListaVacia()

actual <- entrada->inicio

char pedacito

char separacion[2] <- “/”

char meses[12][121] <- … lista de meses

int day,month,year

while actual <> NULL do

char f<- inicializar()

char f2<- inicializar()

char f3<- inicializar()

fecha = fechaSegundaDosis() … calculo la fecha de la segunda dosis y la guardo en fecha

… se ocupa strtok() para separar la fecha y guardarla en las variables enteros day,month,year

insertarCronologicamenteDia(listaDia,actual)… ordeno a los pacientes por dia de su segunda dosis

actual<-actual->siguiente

aux<-listaDia->inicio

while aux <> NULL do

…ordeno la lista por mes  
 insertarCronologicamenteMes(out,aux)

aux<-aux->siguiente

liberarLista(listaDia)

… procedo a escribir la lista en el archive

Out = inicializarNodo()

Contador<-0

For i<-1 to 12

Contador<-0

out<-listaSalida->inicio

for j<-0 hasta largoLista(listaSalida)

if out->month2 = I then

Contador<-contador+1

Out<-out->siguiente

Out<-listaSalida->inicio

If Contador > 0 then

Write(meses[i-1],Contador)

For j<-0 to largoLista(listaSalida)

If out->month2 = I then

printNodoOut(out,vacunas,archivo)

out<-out->siguiente

close(archivo)

El pseudo código anterior muestra como a partir de la lista de vacunados1D nos deja escribir el primer archivo necesario “*listado.out*”. Este algoritmo tiene una cantidad de instrucciones de   
y una complejidad de , lo cual me deja bastante satisfecho dada la cantidad de operaciones realizadas en esta función. Ahora, mostrare el pseudo código del algoritmo para escribir el archivo “provision.out”

listadoProvision(TDAlista lista,TDAlistaVacuna vacunas)

FILE archivo

vacuna<-iniciarNodoV()

paciente<-iniciarNodo()

num count

char meses[12][12]<-… listade strings con meses

for i<-1 to 12

paciente<-lista->inicio

vacuna<-vacunas->inicio

count<-0

while paciente<>NULL do

if paciente->month2 = i then

vacuna<-retornarNodoVacuna(vaunas,paciente->id)

vacuna->dosis<-vacuna->dosis+1

count<-count+1

paciente<-paciente->siguiente

if count > 0 then

write(meses[i-1],count)

vacuna = vacunas->inicio

while vacuna<>NULL

write(vacuna->codigo,vacuna->dosis)

vacuna->dosis<-0;

vacuna<-vacuna->siguiente

close(archivo)

El algoritmo anterior tiene una cantidad de instrucciones de con una complejidad de , el código seguramente pueda modificarse para ser de orden pero encuentro que no era necesario dada la optimización del resto del código. Para finalizar, mostraré el pseudo para el código “*vacunacionCompleta.out*”.

listadoVacunacionCompleta(TDAlista vacunados2D,TDAlista vacunados1D,TDAlistaVacuna vacunas)

FILE archivo

listaSalida<-crearListaVacia()

actual<-iniciarNodo()

salida<-iniciarNodo()

num totalPersonas<-0

actual<-vacunados2D->inicio

while actual <> NULL do

duplicado<-iniciarNodo

if obtenerNodo(vacunados1D,actual->apellido)<> NULL then

duplicado<- obtenerNodo(vacunados1D,actual->apellido)

insertarNodoFinal(listaSalida,duplicado)

insertarNodoFinal(listaSalida,actual)

totalPersonas<-totalPersonas+1

else

insertarNodoFinal(listaSalida,actual)

totalPersonas<-totalPersonas+1

actual<-actual->siguiente

salida<-listaSalida->inicio

write(totalPersonas)

while salida<>NULL

printNodoVCompleta(salida,vacunas,archivo)

salida<-salida->siguiente

close(archivo)

El algoritmo anterior tiene una cantidad de instrucciones de con una complejidad de .

Esas funciones son las más representativas con respecto a mi código, muchas funciones de mis TDA no son difíciles de replicar, la única que me gustaría recalcar seria la serie de instrucciones de insertarCronologicamenteDia, insertarCronologicamenteMes e InsertarAlfabeticamente. Esas funciones anteriores fueron un gran salvavidas, puesto que permiten ordenar la lista de la manera que se me solicita, y la lógica de una se puede utilizar para crear distintos tipos de ordenamiento. A continuación, pondré el pseudo código de insertarAlfabeticamente a modo de idea para otro tipo de implementaciones

insertarAlfabeticamente(TDAlista lista, nodo entrada)

nuevo<-inicializarNodo()

anterior<-inicializarNodo()

actual<-inicializarNodo()

num cmp

anterior<-NULL

actual<-lista->inicio

while actual<> NULL do

cmp<- strcmp(apellido,actual->apellido)

if cmp < 0 then

break

anterior<-actual

actual<-actual->siguiente

nuevo<-crearNodo(variables)

if anterior = NULL then

nuevo->siguiente<-lista->inicio

lista->inicio<-nuevo

else

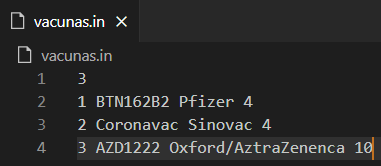
anterior->siguiente<-nuevo

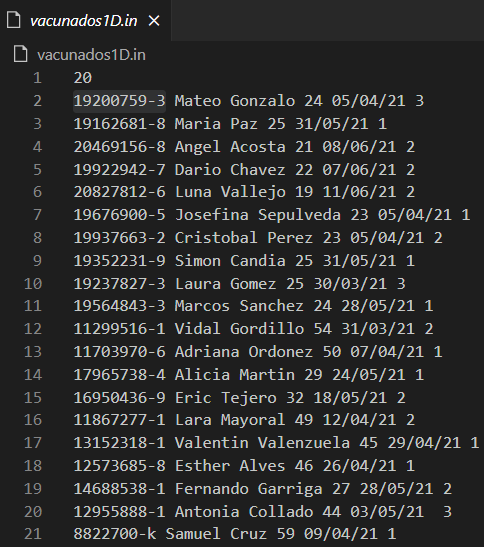
nuevo->siguiente<-actual

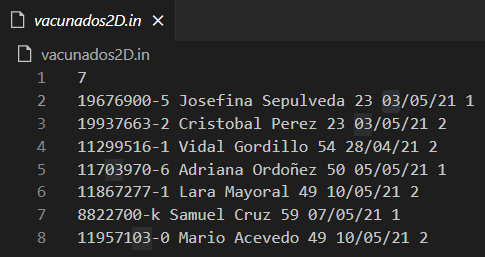
Este algoritmo permite ordenar una lista dada alfabéticamente, tiene una cantidad de instrucciones de y una complejidad de   
lo cual considero eficiente, pudo ser algo como bubble sort o quick sort, pero de esta forma no está nada mal.

# Resultados y Análisis

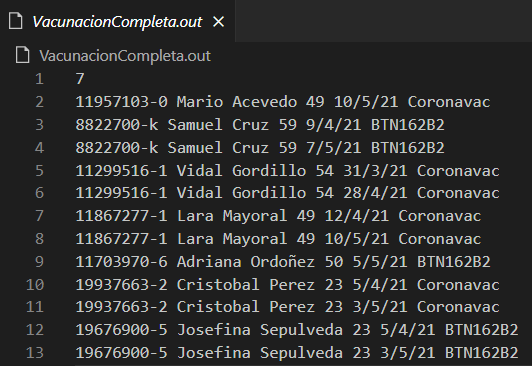
Luego de haber presentado la idea general de las funciones más importantes del programa, se expondrán los diversos resultados obtenidos, para poder ilustrar estos resultados, se utilizará los siguientes archivos de entrada como prueba:

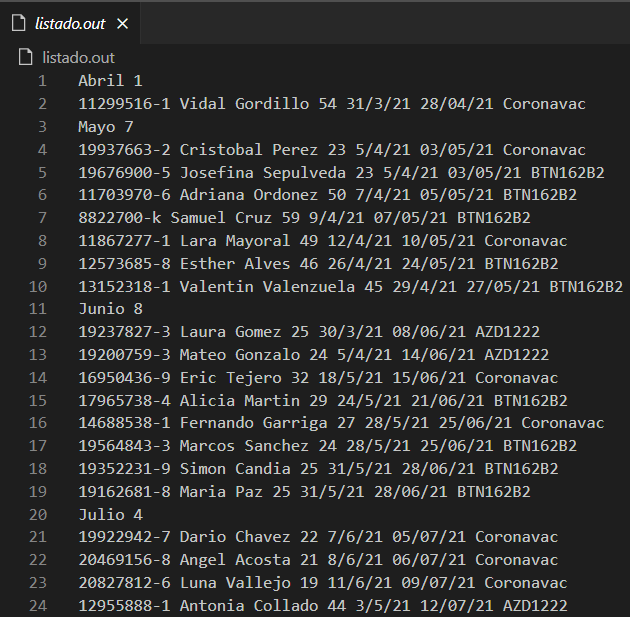
  
Figura 1: archivo “*vacunas.in”*

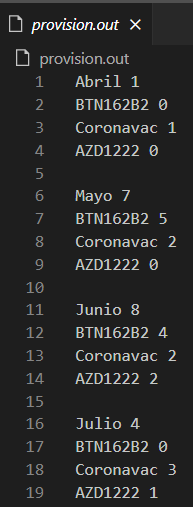
  
Figura 2: archivo “*vacunados1D.in*”

  
Figura 3: archivo *“vacunados2D.in”*

Para estos archivos de prueba, los resultados obtenidos fueron los siguientes:

**Figura 4: archivo *“vacunacionCompleta.out”*

  
Figura 5: archivo *“listado.out”*

**Archivo 6: archivo *“provision.out”*

Se puede ver que los archivos resultantes cumplen con el requerimiento del problema, dado que están separados por mes y dentro de cada mes las personas tienen un orden cronológico por día y alfabéticamente por apellido, por lo tanto, se puede decir que se cumple con los requisitos. Además, el tiempo de ejecución del programa es minúsculo, inferior a 0s, lo cual muestra una buena optimización de Código

# Conclusión

Esta experiencia de tarea 2 fue el primer acercamiento que se tuvo con el tipo de dato de listas enlazadas y considero que fue una buena forma de comenzar a utilizar este concepto, no del todo seguro si me convencen para reemplazar los clásicos arreglos, dado que hubo cosas que si fuesen en arreglos, personalmente me hubiese costado menos darme cuenta el como se pudiese hacer. Pero el desafío de acomodar ciertas funciones de los arreglos a otro tipo de dato fue bastante entretenido, fuera de la frustración de ciertas cosas que no salen bien de inmediato. Por lo mismo, estoy más que conforme con la solución propuesta, creo que mi cumple con todas las expectativas y dejé poco espacio para poder mejorar el código, quizás intentar reutilizar funciones ya escritas para no tener que reescribir funciones ya hechas, pero con otros parámetros, pero más allá de eso considero que la solución fue cumplida.