Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Estructura de Datos y Algoritmo I

Trabajo Integrador Final

Alumno:
Franco I. Vallejos Vigier

Legajo: V2952/1

Diciembre de 2020

Introducción

En este trabajo fue de suma importancia tener una idea clara y concisa de los objetivos planteados por el enunciado provisto por la cátedra.Desde un prematuro comienzo había que tener una división de objetivos, ya sea, el desarrollo de una interfaz, la correcta elección e implementación de diversas estructuras de datos que en estrecha interrelación con una retroalimentación de errores conformarían un amigable intérprete para el usuario.En esta interrelación aceitada sería primordial, antes que nada, plantear un marco metodológico el cual empezaría con un fuerte arraigo teórico orientado hacia las estructuras de datos estudiadas en la materia. Fundamentalmente, en mi proyecto, la estructura Hash y las listas simplemente enlazadas cuyo por qué de elección estará ricamente justificada en un apartado propio.
Luego de poder absorber este marco teórico, he planteado un plan de acción en el cual desde un primer momento trabajaríamos sobre una sintaxis prematura, muy arraigada a la consigna, demasiado sensible al contexto y sin retroalimentación de errores. Para luego pasar a una etapa donde optariamos por una manera eficaz tanto como eficiente de almacenar y recuperar los datos para operar con ellos.

Contents

PART I

Estructura de Datos y Dificultades

1.1 Estructura de almacenamiento troncal Hash

El primer tema a abordar en esta sección es, ¿Cuál estructura de datos adoptar que nos permita un rápido acceso y rápido ingreso de información?, obviamente contemplando el hecho de una reiterada y extensa utilización del programa. Analizando las diversas estructuras de datos dada en la materia, necesitábamos un almacenamiento que sea de rápida indexación por medio de una función que transforme nuestros datos en un número, en un costo, en un índice y así poder introducirlo en nuestra estructura de datos. De una manera sencilla y poco problemática he decidido implementar una estructura Hash a través de listas simplemente enlazada(como área de rebalse, para manejar las colisiones en nuestra tabla).Pero ahora me encontraba en la disyuntiva de encontrar una función Hash que provoque el menor número de colisiones posibles. Pero antes de todo esto había que seleccionar u optar por un largo para nuestra tabla, indagando un poco sobre la cantidad de localidades que posee nuestra provincia llegué a un estimado de 621 localidades(un tamaño un poco excesivo para nuestra tabla dado que con ese tamaño no lograríamos homogeneidad en la carga de nuestra tabla) luego de pensar y sabiendo que hay nombre de localidades iguales, me pareció óptimo tomar la mitad de ese total aproximado de localidades(311 un número primo). Ahora recapitulemos un poco y veamos lo que es una función Hash por división, recordemos que son las funciones del tipo h(k) = k mod M, donde M es el tamaño de la tabla. Analicemos un poco este número M y su importancia, si M fuera para todas las claves pares serían asignadas o indexadas a localizaciones pares, de una manera similar si M fuera impar. Una manera simple para solucionar este problema es que M sea primo y he aquí la importancia del tamaño de la tabla 311 . Ya que si M es primo nos garantiza que antes de la primera colisión todas las celdas habrán sido visitadas 1 vez. Veamos la función hash:

```
unsigned int hash(char* nombre) {
 const int constantePrima = 17;
 unsigned int hash = constantePrima;
 for(size_t i = 0; i < strlen(nombre); i++) {
   hash = hash * constantePrima + nombre[i];
 }
 return hash;
}
```

Esta función va a calcular el valor hash de nuestra localidad y luego al aplicarle la operación mod contra el tamaño primo de la tabla (311) lograríamos conseguir el índice dónde introducir nuestros datos. Podemos apreciar la simpleza de la función hash(), hecha a partir de los ejemplos impartidos en las slides de la cátedra.

Ahora la pregunta sería ¿Cómo resolver una colisión?. Dado que por la dinámica del programa hay que eliminar registros de nuestra estructura de almacenamiento Hash tanto como hay que guardarlos. Opté por un sistema de encadenamiento separado o hashing abierto, por el cual en cada localización de la tabla brindada por la función Hash se construiría una lista simplemente enlazada de registros donde su data vendría ser determinada por un puntero a

la siguiente estructura:

```
typedef struct Diario_{
  char* hora;
  char* departamento;
  char* localidad;
  int confirmados;
  int descartados;
  int enEstudio;
  int notificaciones;
} Diario;
```

Dónde es intuitivo que en la estructura desglosamos los componentes de un informe de infectados almacenando su fecha, el departamento, localidad, casos confirmados, descartados, en estudio y el total de notificaciones. Además una ventaja de implementar esta estructura en una LSE, es que dado el caso para calcular picos de contagios o casos diarios podemos ordenar la lista por el parámetro de fecha para facilitarnos la operatoria.

1.2 ¿LSE? Sí. ¿POR QUÉ?

En este apartado nos dedicaremos a justificar el uso de esta estructura de datos como principal precursor del motor de las operaciones. La simpleza de la lista simplemente enlazada, la facilidad para su manipulación, la posibilidad de una trazabilidad más simple de errores, seguramente a cambio de optimización y gasto de recursos terminaron inclinando la balanza hacia el por qué elegí la LSE. Teniendo en consideración que las operaciones básicas de una lista simplemente enlazada, ya sea buscar tanto como recorrer tienen una complejidad O(n) mientras que el ingreso si este se da al comienzo es O(1), no me parecía una mala base por dónde empezar a desarrollar el programa(además intentaría mantenerme en el costo operacional de cada instrucción en O(n) como meta personal). Por lo cual nuestra estructura de datos para operar en el meollo de cada operación vendría a ser definida por las siguiente estructura:

```
typedef struct ListaEnlazada_ {
  unsigned int clave;
  Diario informe;
  struct ListaEnlazada_* siguiente;
} ListaEnlazada;
```

Dónde Diario informe vendría a estar definida con la anteriormente mencionada struct _Diario (véase 2.1)

PART II

Operaciones

2.1 cargar_dataset()

Esta utilidad o instrucción del intérprete es fácilmente entendible, se toma como argumento un nombre de archivo al cual vamos a abrir y leer línea por línea, así poder desglosar su contenido separado por ',’. Con este contenido vamos a crear una estructura del tipo Diario_ para poder encapsular mejor esta información. Una vez ya toma mi información esté en la estructura previamente comentada, para introducirla en nuestra tabla vamos a hacer uso de nuestra función notificaciones_insertar() que calcula el hash de la estructura por medio de su variable "localidad", para luego aplicarle el módulo(mod) contra su capacidad y encontrar el índice adecuado para insertar la data. Cabe destacar dos aspectos en esta inserción, como se pide sobreescribir una entrada si esta ya existe antes de insertar una estructura Diario_ en la tabla Hash primero hago un eliminado de prevención para asegurarme que no existe un nodo en la tabla con los mismos datos que quiero insertar todo esto gracias a la función notificaciones_eliminar(). El segundo aspecto a compartir de esta instrucción es cuando se hacen reiterados llamados a cargar_dataset(), me pareció una buena implementación "fusionar" los datos cargados previamente con los nuevos de tal modo que si entre las nuevas entradas hay informes de localidades con fechas ya existentes reescribe este valor y si las entradas no se encuentran entre las viejas las añade.

2.2 imprimir_dataset()

Siguiendo la línea de pensamiento de la instrucción anterior, para volcar todo el contenido de una tabla Hash en un archivo es bastante sencillo. Solo hace falta recorrerla en su totalidad, no dejando lista enlazada sin visitar en su totalidad y escribir el contenido de cada nodo en el archivo .csv deseado

2.3 agregar_registro()

Esta instrucción del intérprete viene acompañada por los argumentos ordenados fecha departamento localidad confirmados descartados enEstudio. Para poder insertarlo en nuestra tabla, luego de parsear los argumentos, hago uso de la ya mencionada notificaciones_insertar() que va a calcular el índice de la LSE donde tendría que insertarse nuestro registro nuevo. Para ello mediante la función hash que actúa sobre el argumento "localidad" y luego de aplicarle mod contra 311(la capacidad de la tabla) a este valor obtenemos el índice de nuestra LSE donde tendría que insertarse(y en caso de que ya exista el registro pisarse). Luego todo se reduciría a introducir el nodo en la cabecera de esta lista enlazada para abaratar costos.

2.4 eliminar_registro()

Siguiendo la metodología de la función que se dedica a insertar los registros, la función de eliminación es muy similar en cuanto a la operatoria de encontrar el índice de la LSE donde se encontraría este nodo mediante la función hash y la operación 'mod'. Una vez obtenido este índice recorremos la lista enlazada correspondiente al mismo en busca del registro a eliminar, si lo encontramos nos posicionamos en el nodo anterior a él y cambiamos los links al siguiente

elemento quedando así el nodo buscado desvinculado de la lista y listo para su posterior liberación de memoria.

2.5 buscar_pico()

Esta instrucción le es acompañada por los argumentos (en orden) localidad departamento. Y devuelve el pico máximo de casos diarios. Para ello calculo el índice de la LSE(de la forma ya mencionada), una vez ya obtenida la correspondiente LSE sabiendo que en esa lista se van a encontrar registros de distintas localidades y teniendo en cuenta que la definición de casos diarios es casos_totales_hoy - casos_totales_ayer. Opto por ordenar la lista por la variable fecha en orden de más antiguo a más reciente así es más fácil acarrear un puntero al "dia anterior" para esa localidad y departamento en específico, así pudiendo calcular los casos diarios fácilmente en cada iteración. Para sacar el máximo pico, creo una variable con un valor inicial nulo que se pisa si hay un caso diario que es más grande que ella, garantizando el valor máximo de esta .

2.6 casos_acumulados()

Los argumentos de esta función son fecha localidad departamento, en este orden. Lo que vamos a buscar es devolver la variable de los casos confirmados para esa localidad en la fecha dada. Para ellos calculo el valor hash de la localidad y obtengo la LSE correspondiente a esa clave. Y buscamos un nodo con los mismos valores que los argumentos en caso de encontrarlo devuelvo la cantidad de casos confirmados, de caso contrario devuelvo 0.

2.7 tiempo_duplicacion()

Los argumentos de esta función son fecha localidad departamento, en este orden. Por el ejemplo provisto en el enunciado a simple vista se puede decir que es necesario retroceder en el tiempo en búsqueda de un registro con un valor de confirmados aproximado de la mitad de los casos acumulados totales, en la localidad y en la fecha pasadas como argumentos, y fruto de esta distancia de días es el tiempo de duplicación. Para empezar y facilitar el cálculo me pareció correcto ordenar los registros de la lista por la variable fecha de lo más antiguo a lo más reciente, ya que este orden induce también un orden en los casos confirmados totales para esa localidad y departamento. Pero me surgían varias dudas...
1) Qué pasa si no existe el registro pasado como parámetro?
2) Qué pasa con el tiempo de duplicación si el registro argumento es único o el primero para su localidad y departamento?
3) Qué pasa si el registro del día anterior, al de nuestro argumento, tiene casos confirmados menor a la mitad que el de nuestro argumento?
Por ejemplo 5 ->100
4) Qué pasa si encontramos la mitad exacta?
Por ejemplo 150->160->170->300
5) Qué pasa si no encontramos la mitad exacta?
Por ejemplo 155->160->170->300
Para estas preguntas fije algunas pautas... Pero primero debemos recorrer la LSE correspondiente que guarda esos valores.

.Si la lista llega a ser NULL, significa que no se encontró la fecha, la localidad y/o el departamento, por lo cual no existe el tiempo de duplicación. Solucionando la pregunta 1
.Nosotros vamos a parar de recorrer la lista en el caso de que se encuentre un nodo que tenga como variables los argumentos dados, por lo cual el puntero que recorre la LSE va a quedar apuntando a este nodo y acompañado de una bandera que nos diga si es el primer registro para su localidad y departamento el tiempo de duplicación sería 0. Solucionando la pregunta 2
.Como nosotros en nuestro recorrido de la LSE incrementamos el valor del contador del tiempo duplicación si encontramos un nodo para esa localidad y departamento que tenga una cantidad de confirmados mayor a la mitad de los confirmados del argumento. Dado que apodé al caso de la pregunta 3 como "pico abrupto" consideré que lo más acorde sería que para este caso el tiempo de duplicación sea 1. Solucionando la pregunta 3
.En el caso de la pregunta 4(si se encuentra exactamente la mitad) en el ejemplo dado en el enunciado no tiene en cuenta este día para el resultado final, no lo contabiliza. Solucionando la pregunta 4. Ejemplo 150->160->170->300 tiempo duplicación 3
.Dada la pregunta 5(si no se encuentra la mitad exacta) pero hay valores entre la mitad y los confirmados de mi nodo argumento. Contabilizo todos y cada uno de ellos. Solucionando la pregunta 5. Ejemplo 155->160->170->300 tiempo duplicación
Adjunto el código explicado en la siguiente página.

```c
void tiempo_duplicacion(Notificaciones* agenda, char* fecha, char* localidad,
                                char* departamento) {
//Extraigo los casos confirmados para la localidad
  int contagiosDuplicados = casos_acumulados(agenda, localidad, fecha,
                                                        departamento);
  //Calculo la mitad de los casos confirmados, para saber a que numero
  //de contagio atras en el tiempo debo llegar(en el caso de que este
  // en la tabla)
  contagiosDuplicados = contagiosDuplicados / 2;
  //Calculo el indice mediante la funcion hash
  unsigned int hashCalculado = hash(localidad);
  unsigned int indice = hashCalculado % agenda->capacidad;

  ListaEnlazada * lista = agenda->entradas[indice].listaEnlazada;
  //Variable que lleva los dias de distancia
  int contadorDuplicacion = 1;
  //Variable que nos dice si es el unico nodo en la LSE de esa localidad
  int unico = 0;
  //Como tengo que ir atras en el tiempo me parece primordial ordenar
  //la lista de mas antiguo a mas reciente en cuanto a fechas
  ListaEnlazada* nuevaLista = glist_selection_sort(lista);
  //Recorro la LSE
  for(;nuevaLista != NULL;nuevaLista = nuevaLista->siguiente){
  //Voy a contar aquella fechas anteriores con casos de contagio mayores
  //a la mitad que los contagios de mi parametro
    if (strcmp(nuevaLista->informe.departamento, departamento) == 0 &&
        strcmp(nuevaLista->informe.localidad, localidad) == 0 &&
        nuevaLista->informe.confirmados > contagiosDuplicados) {
      unico++;
  //Si encuentro la fecha de mi parametro salgo del ciclo
      if (strcmp(nuevaLista->informe.hora, fecha) == 0) {
        break;
      }
  //Incremento en 1 los dias de duplicacion
      contadorDuplicacion++;
    }
  }
  //Si la lista es nula, significa que nunca encontros la fecha parametro
  //o que junto con localidad como el departamento no se encontraron
  if (nuevaLista == NULL) {
      printf("Fecha, localidad o departamento no encontrado\n");
    }
  //Si la lista no es nula y solo hizo 1 iteracion, entonces solo
  //para esa localidad hay un solo registro
  if (nuevaLista != NULL && unico == 1) {
    contadorDuplicacion = 0;
  }
  //Sino estamos en presencia de un pico 5, 100 o el caso 50, 60, 70, 100
  printf("El tiempo de duplicacion para la fecha %s,", fecha);
  printf(" en la localidad de %s del dpto %s es de %i\n", localidad,
           departamento, contadorDuplicacion);
}
```

2.8 graficar()

Tomando en cuenta el consejo de investigar "gnuplot" para llevar a cabo las gráficas de los casos diarios y de los casos totales en un intervalo de tiempo para una localidad y departamento específico. Noté que gnuplot puede plotear archivos .dat donde la primera columna es el eje x y la segunda es el eje y. Entonces pensé en hacer dos funciones que conformen dos archivos, uno que

sean los casos acumulados de una localidad en un lapso de tiempo y el segundo que sean los casos diarios para esa misma localidad en el mismo periodo de tiempo. Pero ahora el problema no radica en cómo hacer dichas funciones, que al fin y al cabo son derivadas de casos_acumulados() y buscar_pico(), sino en cómo trabajar con el intervalo de tiempo ya que se puede solapar, puede existir un solo extremo, no puede existir ningún extremos o pueden ser iguales. Igual que en la función tiempo_duplicacion fijé pautas...pero primero definamos intervalo inválido dado que el lapso de tiempo dado es [fechaInicio,fechaFin], el intervalo es válido si fechaInicio más antigua o igual a fechaFin e inválido si fechaInicio es más reciente que fechaFin.
1)Si es un intervalo válido, calculo los casos confirmados y los diarios para cada nodo en el intervalo
1')Si es un intervalo válido, pero solamente se encuentra la fechaInicio en la lista y la fechaFin está excedida o fuera de rango. Calculo los casos confirmados y los diarios para cada nodo desde la fechaInicial hasta la fecha más reciente
1")Si es un intervalo válido, pero solamente se encuentra la fechaFinal en la lista y la fechaInicio no se encuentra o está fuera de rango. No tengo referencia al inicio del intervalo por lo que no realizo operaciones
2)Si es un intervalo inválido, en el cual fechaInicio es más reciente que fechaFinal, no opero
Con esta metodología de trabajo puedo conformar mis dos archivos previamente mencionados para graficarlos con gnuplot. Adjunto código explicado de ploteo

```
void plot_casos_acumulados() {
  //Invoco a gnuplot
  FILE *gnuplotPipe = popen("gnuplot", "w");
  //Seteo el nombre de la grafica
  fprintf(gnuplotPipe, "set title \"Contagios Acumulados\" font \",20\"\n");
  fprintf(gnuplotPipe, "set key left box\n");
  fprintf(gnuplotPipe, "set samples 50\n");
  fprintf(gnuplotPipe, "set style data points\n");
  //Seteo el nombre de los ejes, en caso son los contagios confirmados(eje Y)
      y
  //el tiempo (eje X) expresado en dias posteriores a la fecha de inicio
  fprintf(gnuplotPipe, "set xlabel \"Dias siguientes a la fecha de inicio\"\n"
      );
  fprintf(gnuplotPipe, "set ylabel \"Contagios\"\n");
  //Configuro el rango a graficar
  fprintf(gnuplotPipe, "set xrange [ 0 : * ] noreverse writeback\n");
  fprintf(gnuplotPipe, "set yrange [ 0 : * ] noreverse writeback\n");
  //Ploteo el archivo en cuestion
  fprintf(gnuplotPipe, "plot [0:*] 'AcumuladosAGraficar.dat'\n");
  fflush(gnuplotPipe);
  sleep( 5 );
  printf("Para cerrar la grafica, aprete enter...\n");
  getchar();
  //Cierro la grafica
  fprintf(gnuplotPipe, "quit\n");
  pclose(gnuplotPipe);
}
```

PART III

Complementos

3.1 Makefile, compilación y ejecución

La compilación se lleva a cabo con el comando make, detallando que el archivo a compilar es el ./main creando el ejecutable.

3.2 Uso

En cuanto al uso del programa, se recomienda introducir los argumentos de las instrucciones en este orden
cargar dataset notificaciones localidad.csv
imprimir dataset salida.csv
agregar registro fecha departamento localidad confirmados descartados enEstudio
eliminar registro fecha localidad departamento
buscar pico localidad departamento
casos acumulados fecha localidad departamento
tiempo duplicacion fecha localidad departamento
graficar fechaInicio fechaFin localidad departamento
salir
Mi intérprete carece de soporte de ñ y tildes en input por parte del usuario en el teclado. Sería un punto a fortalecer en versiones posteriores.

3.3 Bibliografía y material de consulta

. Apuntes teórico de la asignatura
. El lenguaje de programación C, 2da edición. Kernighan & Ritchie

3.4 Curiosidades

3.4.1 Encarpetado

Sinceramente, me hubiese gustado darle un encarpetado más prolijo al proyecto dándole mayor organización. Desde ya sería la principal tarea pensando en un aspecto a mejorar, acompañado de un parser un poco más robusto

3.4.2 if()...

En tres ocasiones tuve que usar estructuras if() vacías cuando nos referimos a la captura de datos por teclado ya sea por la función fgets() o scanf(). Dado que cuando compilaba con el make, este tiraba el siguiente error:

```
Ignoring return value of ‘scanf’, declared with attribute warn_unused_result
Ignoring return value of ‘fgets’, declared with attribute warn_unused_result
```

Dado que intente poner el cast (void) delante de las funciones, cosa que no funcionó. Intenté colocarlas en estructuras if() seguidas de un ";" y el error persistía en el make, no me quedó otra opción que poner llaves.

```
if (fgets(alias, LIMITE, stdin)){}
char* operacion = calloc(LIMITE, sizeof(char));
int instruccion = 0;
if (strlen(alias) == LIMITE - 1) {
  printf("Overflow de entrada...\n");
```

```
    if (scanf("%*[^\n]")){}
    if (scanf("%*c")){}
    free (alias);
    free (operacion);
    continue;
    }
```

Desconozco el por qué del origen de este error dado que a mis compañeros no les ocurrió. Pero buscando en internet, un poco, este error es frecuente.