

Estructura de Datos y Análisis de Algoritmos Informe De Laboratorio N°5

Preparado por: Carlos Alvarez Silva

Profesor:
Jacqueline Kohler Casasempere

Ayudante:
Alejandro Cisterna Villalobos

Santiago - Chile
2017

TABLA DE CONTENIDOS

TABLA DE CONTENIDOS.....	I
ÍNDICE DE FIGURAS.....	II
CAPÍTULO 1. INTRODUCCIÓN.....	1
CAPÍTULO 2. MARCO TEÓRICO	2
CAPÍTULO 3.DESCRIPCIÓN DE LA SOLUCIÓN	3
3.1. LIBRERÍAS UTILIZADAS	4
3.2. TIPOS DE DATOS ABSTRACTOS UTILIZADOS	5
3.2.1 Bool	5
3.2.2 Lista.....	5
3.2.3 ABO	¡Error! Marcador no definido.
3.3. FUNCIONALIDADES DEL PROGRAMA.....	6
3.3.1 Registro de datos	6
3.3.2 Localización de contactos	6
3.3.3 Impresión de contactos.....	7
CAPÍTULO 4. ANÁLISIS DE LOS RESULTADOS	8
CAPÍTULO 5. CONCLUSIÓN	9
REFERENCIAS.....	10
ANEXO. MANUAL DE USUARIO	11
A.1. INTRODUCCIÓN	11
A.2. COMPILACIÓN Y EJECUCIÓN	11
A.3. FUNCIONALIDADES DEL PROGRAMA.....	13
A.4. POSIBLES ERRORES	14

ÍNDICE DE FIGURAS

Figura 2.1. Ejemplo de ABO.....	2
Figura 3.1. Gráfica entre el número de datos recibidos y el tiempo de ejecución del programa medido en recursos del procesador utilizados.....	8
Figura A.1. Representación de los archivos de entrada y de salida esperados.....	11
Figura A.2 Modo de compilación y ejecución en Linux.....	12

CAPÍTULO 1. INTRODUCCIÓN

Desde hace tiempos inmemorables los seres humanos han creado acertijos y claves para diversos propósitos, siendo uno de ellos el laberinto, estructura edificada en base a murallas y pasillos, creados con el objetivo específico de confundir a cualquiera que intente atravesar a través de estos para encontrar algún elemento y/o la salida en ellas.

Bajo este contexto se tiene que Juan Estructuradedatos se encuentra saturado por la abrumadora cantidad de laberintos que debe realizar, y decide generar un registro de la amplia variedad de contactos que tiene disponible para resolver dichos acertijos.

Para el curso laboratorio de Análisis de Algoritmos y Estructuras de datos, se les pide a los estudiantes encontrar una serie de contactos organizados en una estructura variante del árbol binario ordenado, conocida bajo el nombre de árbol AVL.

Para el presente laboratorio se pide que, en base a dos archivos de texto plano, una con una serie de contactos bajo la estructura “*Nombre1 Nombre2 Apellido1 Apellido2 Fono*”, y otro con solo los nombres y apellidos, generar un nuevo archivo de texto plano con los contactos localizados y el teléfono asociado correspondiente.

En las siguientes páginas se verá en profundidad la resolución del problema, los fundamentos que dan origen al algoritmo de respuesta, y como este se ve aplicado en la solución real a la problemática presentada. Además se adjunta el manual de usuario que indica el procedimiento de preparación, compilación y ejecución requeridas para inicializar el programa, junto con los posibles errores que puedan ocurrir durante la ejecución.

CAPÍTULO 2. MARCO TEÓRICO

Para resolver el problema de búsqueda de contactos se recurre a un tipo de datos abstracto conocido como árbol AVL, una variante especial de grafo dirigido creado en 1962 por el científico en computación Ruso Georgii Adelson-Velskii y el matemático soviético Yevgeniy Landis (el nombre AVL se nombra por las iniciales de los apellidos de ambos creadores), que tiene la particularidad de que cada uno de sus nodos puede recibir solo una arista de entrada y tener entre cero a dos aristas de salida, tal y como se muestra en la figura.

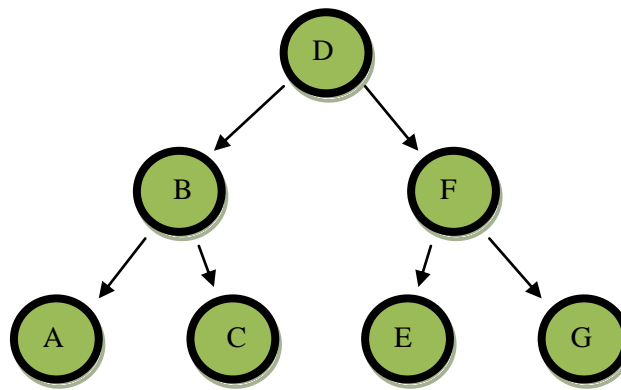


Figura 2.1. Ejemplo de AVL.

Una característica importante del árbol AVL, que lo diferencia de un ABO (árbol binario ordenado), es que los nodos, además de hallarse ordenados, siendo siempre el nodo izquierdo a un nodo menor a este, a la par que su nodo derecho es mayor al cual es apuntado, sino que además se asegura que exista una diferencia mínima entre las alturas de los sub-grafos izquierdo y derecho para cualquier nodo, característica que no era fundamental en un ABO. De este modo se asegura que la búsqueda de elementos en AVL sea más veloz en el peor escenario posible que su variante, sacrificando tiempo en el proceso de inserción de datos nuevos.

Para el caso del problema, en el que los nodos contienen palabras, concretamente nombres completos, estos se ordenan en orden alfabético, siendo el hijo izquierdo de un nodo siempre alfabéticamente menor que el hijo derecho y el nodo padre, y viceversa, tal como se muestra en la figura 2.1.

CAPÍTULO 3.DESCRIPCIÓN DE LA SOLUCIÓN

Refiriéndose al algoritmo se recurrirá enteramente al desarrollo del tipo de dato abstracto árbol AVL y sus funciones correspondientes para su construcción, modificación, recorrido y eliminación.

Se hace hincapié en este tipo de datos, dado que es la base fundamental de la solución general, debido a que su naturaleza binaria hace que, en el momento de buscar algún elemento se descartarán en cada ciclo de búsqueda siempre la mitad del número de posiciones probables en donde podrían ser localizadas coincidencias con aquello que se desea encontrar.

Luego el programa en primer lugar obtendría en base al archivo de entrada entregado, los contactos que serán ingresados a un árbol AVL para su posterior búsqueda.

Luego se realizará la búsqueda en sí, en donde se revisa en el siguiente archivo de texto cada uno de los elementos a buscar y que, por cada uno se realice una búsqueda en el árbol binario, resultando en un listado de elementos string (cadenas de caracteres) que irá en aumento en cada ciclo, siendo esta serie de strings el resultado que se mostrará finalmente en un archivo de texto entregado como resultante.

Finalmente se imprime la cadena de cadenas de caracteres que se traducen en la solución final.

3.1. LIBRERÍAS UTILIZADAS

Las librerías utilizadas en el transcurso del programa, que no fueron creadas específicamente para el laboratorio, fueron obtenidas de las librerías estándar de C (ANSI), las cuales se nombran a continuación:

stdio.h: librería que contiene funciones destinadas al manejo de los datos de entrada y de salida. Entre las funciones utilizadas en el programa se encuentran las funciones printf, scanf, fopen, fclose, entre otros.

stdlib.h: librería que contiene funciones y utilidades varias para uso general. Entre las funciones utilizadas en el programa se encuentran las funciones de manejo de memoria como malloc, calloc, realloc, free, entre otras utilidades.

string.h: librería que contiene funciones y utilidades enfocadas a la manipulación y uso de cadenas de caracteres, esenciales para el almacenamiento de palabras (como arreglos de caracteres). Se utiliza la función strcmp y la función strcat, que se encuentran disponibles en esta librería.

3.2. TIPOS DE DATOS ABSTRACTOS UTILIZADOS

3.2.1 Bool

Tipo de dato que consiste en dos constantes numéricas que C por defecto trata como operandos lógicos. Estos valores booleanos son TRUE y FALSE, los cuales numéricamente corresponden a los valores de 1 y 0 respectivamente. Como técnicamente su representación es más el de un tipo de dato común que de uno abstracto propiamente tal, su implementación como valores constantes se hallan en la librería “includes.h” dentro del programa.

3.2.2 Lista

Tipo de dato utilizado para el manejo dinámico de información. Utiliza en su implementación la dirección de un elemento tipo Nodo y un entero que indica el número de nodos en la lista. A su vez en cada nodo existe un arreglo de caracteres que representan una palabra como dato. Es utilizado para almacenar los resultados de búsqueda, los cuales pueden ser impresos en pantalla o en un archivo de texto.

3.2.3 Árbol AVL

Tipo de dato que guarda en su interior elementos del tipo “nodo Padre”; los cuales a su vez en el interior de cada uno existen cuatro elementos: dos cadenas de caracteres que contienen el nombre completo y teléfono asociado a un contacto, y dos direcciones de memoria que, pueden ser nulas o bien dirigir a otro elemento del tipo Nodo, los cuales representan a los hijos izquierdo y derecho de un nodo.

La definición de este tipo de dato y su función constructor se encuentran en los archivos “ArbolAVL.h” y “ArbolAVL.c” respectivamente.

3.3. FUNCIONALIDADES DEL PROGRAMA

3.3.1 Registro de datos

Esta funcionalidad, localizada en los archivos “RegistrarDatos.c” y “RegistrarDatos.h” recibe un elemento de dato tipo ABO, inicialmente vacío que, en base a un archivo de texto cuyo nombre ha sido indicado como parámetro de entrada, lee cada línea del archivo de texto, lo reconoce como un contacto y es ingresado en la estructura ABO. Al finalizar retorna el ABO modificado como salida.

- Entradas:
 - Un árbol binario vacío
 - El nombre del archivo sobre el cual se van a obtener los datos
- Salidas: Un árbol binario con los datos nuevos
- ❖ Tiempo de ejecución: $30 + 40n + 54n^2 + 12n^3$
- ❖ Orden de ejecución : n^3

3.3.2 Localización de contactos

Esta funcionalidad, recibe una representación tipo ABO y, como indica su nombre, busca todas las apariciones de contactos que aparecen en un archivo de texto cuyo nombre se encuentra como parámetro de entrada. Devuelve como resultado un elemento del tipo de dato Lista, con todos los elementos que se deberán imprimir más adelante.

- Entradas:
 - Un árbol binario con datos
 - El nombre del archivo sobre el cual se van a obtener los datos
 - Un arreglo de caracteres con el nombre del archivo a revisar
- Salidas: Una lista con arreglos de caracteres que representan la respuesta al problema presentado
- ❖ Tiempo de ejecución: $29 + 52n + 74n^2 + 8n^3$
- ❖ Orden de ejecución : n^3

3.3.3 Impresión de contactos

Este procedimiento recibe un elemento del tipo de dato Lista con varios elementos string (cadena de caracteres) a imprimir secuencialmente en un archivo de texto cuyo nombre se encuentra como parámetro de entrada.

- Entradas:
 - Una lista que contiene una serie de elementos string a imprimir
 - Un string con el nombre del archivo de texto sobre el cual se van a imprimir los resultados
- Salidas: Ninguna
- ❖ Tiempo de ejecución: $12 + 28n + 8n^2$
- ❖ Orden de ejecución : n^2

CAPÍTULO 4. ANÁLISIS DE LOS RESULTADOS

En base al análisis del código generado se concluye que el mismo cumple con las bases del problema inicial impuesto, vale decir, que el programa habiendo recibido una lista de contactos válidos y una lista de contactos a buscar, entrega un archivo de texto con los contactos localizados.

Considerando la implementación del algoritmo se tiene que la velocidad y orden del programa final son $T(120 + 241n + 124nn + 20nnn)$ y $\sim O(n^3)$ respectivamente. Comparativamente no se puede apreciar una variación positiva con respecto a la eficiencia entre este laboratorio y el anterior, por lo que se decide modificar los códigos de ambos para que puedan medir el tiempo en milisegundos. La grafica correspondiente se muestra a continuación. Como se puede apreciar, existe una diferencia en la velocidad entre ambos laboratorios, debido al aumento en el número de procedimientos durante la inserción de elementos nuevos.

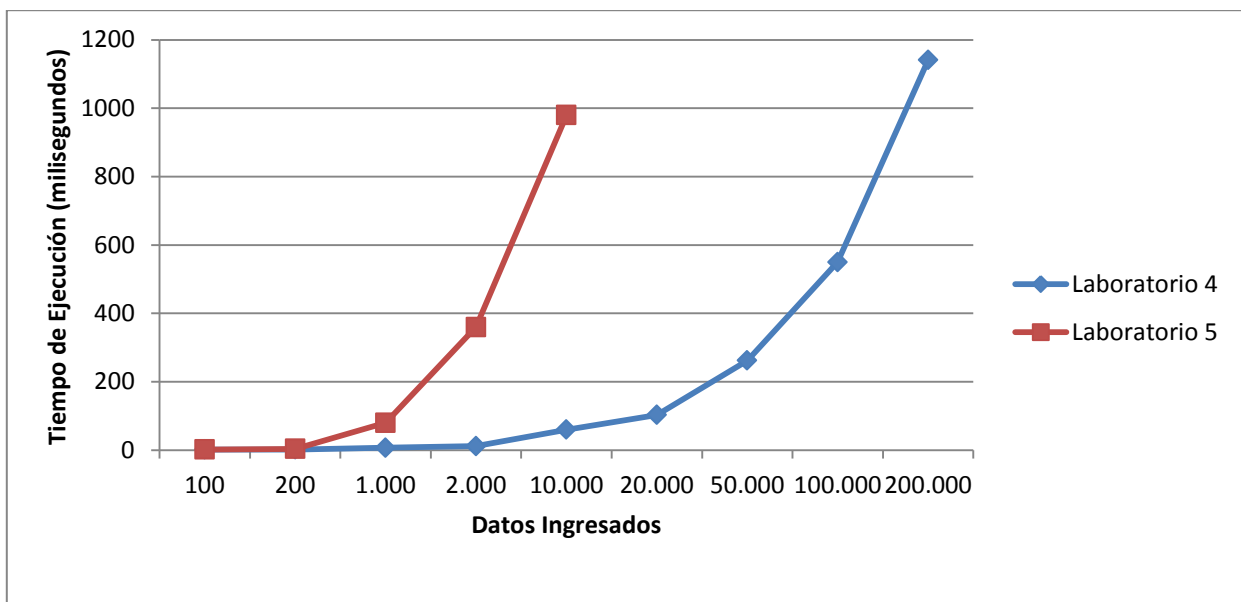


Figura 3.1. Gráfica entre el número de datos recibidos y el tiempo de ejecución del programa medido en milisegundos.

CAPÍTULO 5. CONCLUSIÓN

Para concluir se puede decir que los objetivos del laboratorio se completaron en su totalidad; el programa compila y ejecuta correctamente hasta el final, logrando, exitosamente encontrar rápidamente una serie de contactos en una lista.

En base a la experiencia del laboratorio, se puede concluir que un planteamiento correcto y el uso de datos abstractos adecuados son cruciales para el desarrollo eficiente de un programa, además que el uso de grafos y sus derivados facilitaron la búsqueda de elementos durante los ciclos de laboratorio, ya sean salidas para variados laberintos, o un contacto dentro de una base de datos.

Se puede notar que mientras ciertos planteamientos privilegian memoria, otros demuestran ser más eficientes, aun cuando se conserva la eficacia en ambos casos. Se hace necesario tener una idea concreta de cuáles serán las prioridades durante la creación de un programa, cuales secciones se utilizarán con mayor frecuencia y cuáles no, entre otras variables.

REFERENCIAS

- [1] Biblioteca ANSI C. Recuperado el 12/05/2017 de la página C con Clase. Página disponible en: <http://c.conclase.net/librerias/>
- [2] Árbol AVL. Recuperado el 20/06/2017 de la página Wikipedia. Página disponible en: https://es.wikipedia.org/wiki/%C3%81rbol_AVL

ANEXO. MANUAL DE USUARIO

A.1. INTRODUCCIÓN

El presente manual tiene como objetivo presentar el siguiente programa que localiza una serie de contactos indicados en un archivo de texto plano bajo el nombre de “Buscados.in”, desde una base de datos creada en otro archivo de texto bajo el nombre de “Telefonos.in” y que entrega el resultado en otro archivo de texto bajo el nombre de “TelefonosEncontrados.out”.

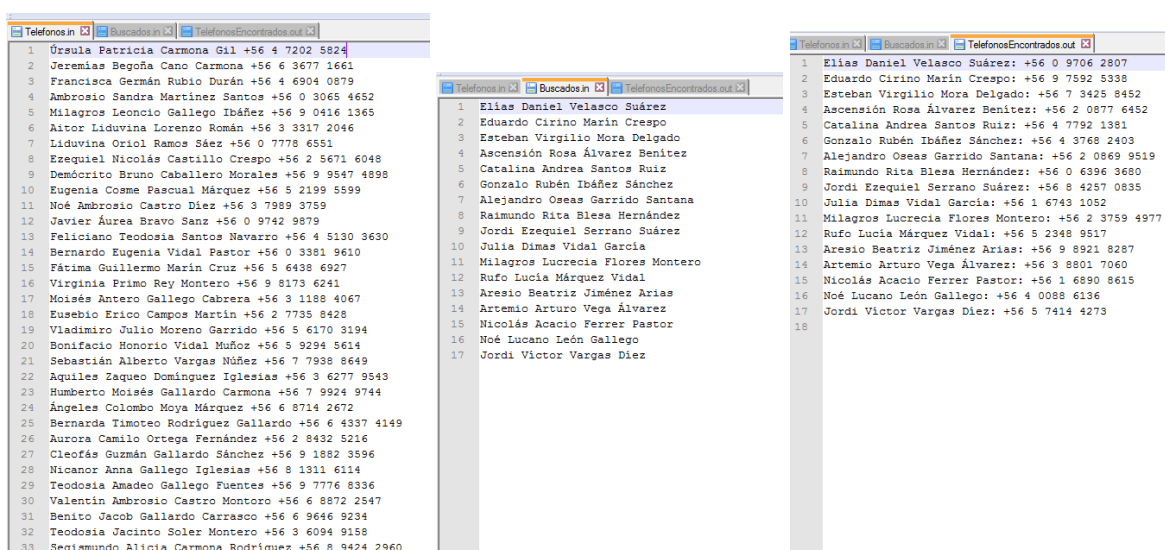


Figura A.1. Representación de los archivos de entrada y de salida esperados

A.2. COMPILACIÓN Y EJECUCIÓN

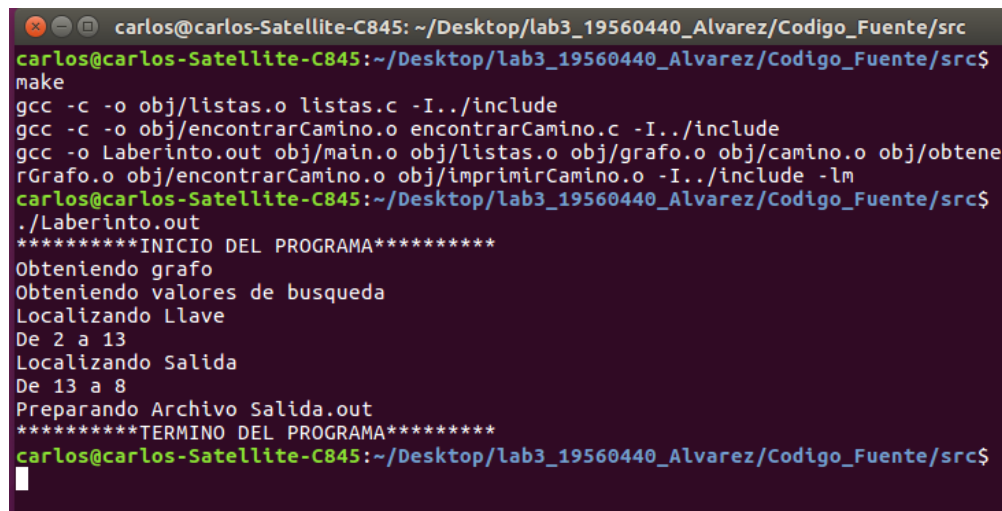
Para poder modificar y compilar el código fuente se requiere de lo siguiente:

- MinGW (Disponible en la página: <https://sourceforge.net/projects/mingw/files/>)
- Notepad++ (Disponible en la página: <https://notepad-plus-plus.org/>)

Para compilar el programa en Windows primero se debe abrir el bloque de comandos del sistema y acceder a la ubicación del archivo “Laberinto.c”. Para esto se utiliza el comando “cd” seguido de la dirección en sistema de la carpeta que contiene el código fuente,

separados por un espacio (ej: “cd C:\User\Desktop\Laboratorio\Codigo_Fuente”). En el directorio del código fuente se ejecuta el siguiente comando: “make” el cual le indica al programa MinGW que compile el archivo de nombre “Laberinto.c” y genere un archivo ejecutable.

En el caso de que se quiera compilar en Linux, se debe acceder desde la terminal del sistema operativo del mismo modo que en el caso del bloque de comando de Windows (haciendo uso de “cd”). Para compilar se debe modificar la extensión del ejecutable a “.out” dentro del archivo Makefile ubicado en la carpeta “src”, luego se ejecuta el comando “make” en la terminal. Finalmente para ejecutar se debe escribir “.” seguido del nombre del ejecutable de esta forma: “.\nombre_ejecutable.out”, tal y como se muestra en la siguiente figura:



```

carlos@carlos-Satellite-C845: ~/Desktop/lab3_19560440_Alvarez/Codigo_Fuente/src
carlos@carlos-Satellite-C845:~/Desktop/lab3_19560440_Alvarez/Codigo_Fuente/src$
make
gcc -c -o obj/listas.o listas.c -I../include
gcc -c -o obj/encontrarCamino.o encontrarCamino.c -I../include
gcc -o Laberinto.out obj/main.o obj/listas.o obj/grafos.o obj/camino.o obj/obtene
rGrafo.o obj/encontrarCamino.o obj/imprimirCamino.o -I../include -lm
carlos@carlos-Satellite-C845:~/Desktop/lab3_19560440_Alvarez/Codigo_Fuente/src$
./Laberinto.out
*****INICIO DEL PROGRAMA*****
Obteniendo grafo
Obteniendo valores de busqueda
Localizando Llave
De 2 a 13
Localizando Salida
De 13 a 8
Preparando Archivo Salida.out
*****TERMINO DEL PROGRAMA*****
carlos@carlos-Satellite-C845:~/Desktop/lab3_19560440_Alvarez/Codigo_Fuente/src$

```

Figura A.3 Modo de compilación y ejecución en Linux

Nota: es importante revisar que, en el interior del archivo Makefile se encuentre la extensión correspondiente a un archivo ejecutable según el sistema operativo sobre el cual se compila, de otro modo no se podrá ejecutar el archivo más tarde. Nótese que la extensión de un ejecutable para Windows es “.exe” y para Linux es “.out”.

A.3. FUNCIONALIDADES DEL PROGRAMA

Las funcionalidades del programa se pueden separar en tres grandes fases:

- **REGISTRO DE DATOS:**

Esta funcionalidad, localizada en los archivos “RegistrarDatos.c” y “RegistrarDatos.h” recibe un elemento de dato tipo Árbol AVL, inicialmente vacío que, en base a un archivo de texto cuyo nombre ha sido indicado como parámetro de entrada, lee cada línea del archivo de texto, lo reconoce como un contacto y es ingresado en la estructura árbol AVL. Al finalizar retorna el árbol AVL modificado como salida.

- **LOCALIZACIÓN DE CONTACTOS:**

Esta funcionalidad, recibe una representación tipo Árbol AVL y, como indica su nombre, busca todas las apariciones de contactos que aparecen en un archivo de texto cuyo nombre se encuentra como parámetro de entrada. Devuelve como resultado un elemento del tipo de dato Lista, con todos los elementos que se deberán imprimir más adelante.

- **IMPRESIÓN DE CONTACTOS:**

Este procedimiento recibe un elemento del tipo de dato Lista con varios elementos string (cadena de caracteres) a imprimir secuencialmente en un archivo de texto cuyo nombre se encuentra como parámetro de entrada.

Nota: Cada una de las funcionalidades anteriormente descritas se activa de forma secuencial inmediatamente después de iniciado el programa.

A.4. POSIBLES ERRORES

- El programa no está preparado para proseguir en caso de falta de memoria, lo que vale decir que en caso de que el programa se ejecute en un entorno demasiado atareado en términos de procesamiento y memoria, hace probable que el programa se detenga abruptamente.
- Dado que el programa recibe un archivo de texto con características específicas, el no cumplimiento de las mismas dará origen a resultados impredecibles, sin embargo, y para casos puntuales, el programa indicará falencias en el archivo de entrada, siendo estos casos la no lectura del archivo, ya sea por corrupción o inexistencia.
- Otro tipo error se da cuando no se compila el código nuevo, en ese caso se debe ejecutar el comando “make clean” en el terminal o bloque de comandos correspondiente para eliminar los archivos de extensión “.o” creados; luego se debe revisar que la extensión del ejecutable indicado en el Makefile es el indicado para el sistema operativo en cuestión para modificarlo según corresponda; finalmente compilar de forma normal.
- Si al compilar en el bloque de comandos, este indica la ausencia de librería “gcc”, ocurrió por la no instalación del programa MinGW en el caso de Windows, por tanto se resuelve con la pronta instalación del mismo.
- En caso de alguna falla ajena a lo indicado anteriormente, por favor contactar con el especialista en computación más cercano.