

**ESTRUCTURA DE DATOS Y ANÁLISIS DE ALGORITMOS
INFORME Y MANUAL**

Cristian Eduardo Espinoza Silva

Profesor: Jacqueline Köhler
Alejandro Cisterna
Ayudante: Luis Migryk
Fecha de Entrega: 6 de abril de 2017

Santiago de Chile

1 - 2017

TABLA DE CONTENIDO

CAPÍTULO 1. Introducción	3
1.1 Descripción del problema.....	3
1.2 Objetivos del software	4
CAPÍTULO 2. Desarrollo.....	5
2.1 Marco teórico	5
2.2 Aspectos de implementación y alcances	6
2.3 Metodología implementada.....	6
2.4 Estructura implementada.....	7
2.5 Descripción del algoritmo	8
2.6 Análisis de resultados.....	9
CAPÍTULO 3. Conclusión	11
CAPÍTULO 4. Referencias.....	12
CAPÍTULO 5. Manual de usuario.....	13
5.1 Introducción	13
5.2 Instrucciones.....	14
5.2.1 Descarga de MinGW	14
5.2.2 Compilación	14
5.2.3 Ejecución	21
5.3 Funcionalidades.....	23
5.4 éxitos y Posibles errores	28

ÍNDICE DE FIGURAS

Figura 1: Entrada del software.	3
Figura 2: Salida entregada por el software.	4
Figura 3: Grafo, aristas y nodos.	5
Figura 4: Estructuras del software.	7
Figura 5: Ventana ejecutar en sistema operativo Windows.	14
Figura 6: Consola de Windows.	15
Figura 7: Crear carpeta y almacenar software dentro.	15
Figura 8: Buscando ruta de la carpeta creada.	16
Figura 9: Compilando software.	16
Figura 10: Creado el ejecutable del software.	17
Figura 11: Abrir consola de Linux o Terminal.	18
Figura 12: Ventana de la consola de Linux o Terminal.	18
Figura 13: Carpeta con el software dentro.	19
Figura 14: Consola de Linux con la ruta de la carpeta que contiene el software.	19
Figura 15: Compilación de software.	20
Figura 16: Carpeta con el archivo ejecutable.	20
Figura 17: Software ejecutado.	21
Figura 18: Software ejecutado.	22
Figura 19: Menú principal del software.	23
Figura 20: Cargando el archivo al software.	24
Figura 21: Mostrando el laberinto por pantalla.	24
Figura 22: Buscando camino mínimo hacia la llave.	25
Figura 23: Buscando camino minimo hacia la salida.	25
Figura 24: Cerrando el software.	26
Figura 25: Carpeta con el archivo de “salida.out” del software.	26
Figura 26: Salida final del software.	27
Figura 27: Posible error dentro de software.	28

ÍNDICE DE TABLAS

Tabla 1: “Tiempos de ejecución y orden de ejecución”	9
Tabla 2: “Tiempo y orden de ejecución del bloque principal”	9

CAPÍTULO 1. INTRODUCCIÓN

En el presente informe se ha pedido a los alumnos de la universidad de Santiago de Chile de la asignatura Estructura de datos y análisis de algoritmos que lleven a cabo un software orientado a la resolución de un “Laberinto” en particular, ya que está representado mediante grafos (no dirigidos).

Respecto a la solución única del “Laberinto” que recae en encontrar la salida, lo que conlleva a los alumnos de la asignatura crear un software que sea capaz de poder cumplir con dicho objetivo planteado anteriormente, con un paradigma de programación imperativo, además de ocupar un lenguaje de programación C.

En el transcurso del informe se dará a conocer el desarrollo y la solución implementada para dicho problema, es decir, se describirá paso a paso el desarrollo de la solución que logro acabar con cada una de las problemáticas planteadas, para proceder a analizar los resultados y finalizar con una reflexión acerca de software entregado.

1.1 DESCRIPCIÓN DEL PROBLEMA

El problema consiste en diseñar un software que sea capaz de resolver un “Laberinto”, el cual tiene una temática particular la cual recae en resolver un “Laberinto” representado un grafo no direccionado. Se leen las relaciones de cada nodo y sus respectivos pesos de un trayecto a otro, estos datos son obtenidos mediante la lectura de un archivo.in, el cual se puede apreciar en la figura 1:

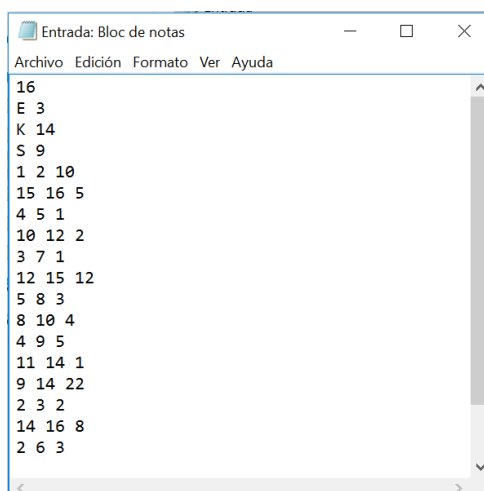


Figura 1: Entrada del software.

El jugador se encarga de manejar el menú de inicio, con el fin de poder dar una interacción con el usuario.

La salida que entrega el software es el camino mínimo que se encuentra desde la entrada hacia llave en primera instancia y luego el camino mínimo que se encuentra desde la llave hacia la salida del laberinto. Lo mencionado anteriormente se entrega en un archivo.out, teniendo una apreciación como se muestra en la figura 2:

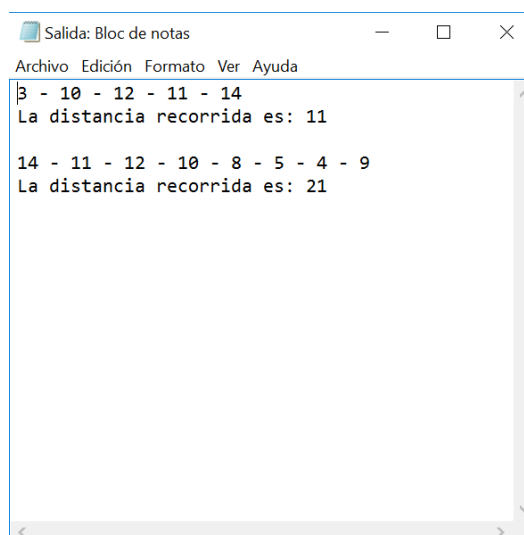


Figura 2: Salida entregada por el software.

1.2 OBJETIVOS DEL SOFTWARE

El objetivo general es realizar un software capaz de solucionar el “Laberinto”, desarrollado en lenguaje de programación C, ocupando el paradigma de programación imperativo.

Los objetivos específicos del software son implementar lo siguiente:

- Cargar las relaciones de los nodos y sus respectivos pesos para moverse de uno a otro.
- Encontrar el camino mínimo hacia la llave, luego encontrar el camino mínimo hacia la salida del “Laberinto”.
- Entregar un archivo de texto con los nodos visitados para lograr los objetivos descritos anteriormente, además de entregar el peso necesario para realizar dicha trayectoria en cada uno de los casos.

CAPÍTULO 2. DESARROLLO

2.1 MARCO TEÓRICO

- Grafo: Es un conjunto de puntos y un conjunto de líneas, cada una de las cuales une un punto con otro. Los puntos se llaman nodos o vértices de un grafo y las líneas se llaman aristas o arcos.

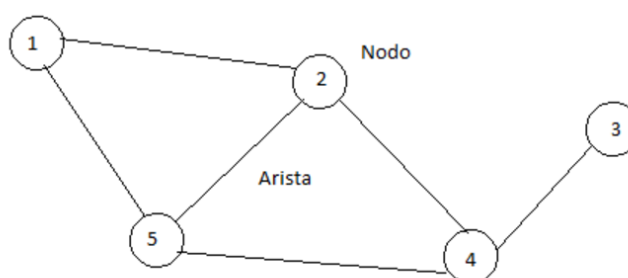


Figura 3: Grafo, aristas y nodos.

- Nodo: Es un punto de intersección, conexión o unión de varios elementos que confluyen en el mismo lugar.
- Arista: Corresponde a una relación entre dos vértices de un grafo.
- Estructuras: Representan colecciones de variables que son diferenciadas cada una de ellas por el tipo de variable y su respectivo nombre, estas pueden contener variables de distintos tipos.
Ocupan la palabra reservada “Typedef struct”, en la figura 2 se puede apreciar los struct declarados.

2.2 ASPECTOS DE IMPLEMENTACIÓN Y ALCANCES

El principal alcance que se presenta es el cumplimiento del desarrollo del software.

Se lleva a cabo el software en el lenguaje de programación C, desarrollado en el editor de texto “Sublime Text 3”, es un editor de texto encarecido, es decir, funciona como una ayuda con la sintaxis propia del lenguaje destacando estructuras, funciones, declaraciones, etc.

La estructura del software se organiza en tres archivos, los cuales son: El archivo ejecutable que contiene el “main”, un archivo “.c” que contiene las funciones que se ocupan y por último el “.h” que contiene las estructuras que se utilizan en el transcurso del software.

Se importan distintos tipos de bibliotecas, con el fin de poder controlar e solucionar problemas que se fueron descubriendo en el transcurso del software.

2.3 METODOLOGÍA IMPLEMENTADA

La metodología implementada fue división en sub-problemas. El algoritmo implementado dentro del software es *Dijkstra*, el cual se encarga de encontrar el camino mínimo dentro de un grafo no direccional.

La secuencia de pasos que sigue en algoritmo implementado son las siguientes:

- El algoritmo comienza leyendo la cantidad de nodos que contiene el grafo, para luego leer cada una de las relaciones que tiene sus grafos con su respectivo peso, con el fin de guardar en memoria cada una de ellas.
- Posteriormente este deja guardado cada una de las posiciones donde se encuentran la entrada, llave y salida del “Laberinto”.
- Luego se produce a realizar la búsqueda del camino mínimo que permite llegar desde un nodo inicial (Entrada), hacia un nodo de objetivo (llave), esta búsqueda se realiza con el algoritmo de *Dijkstra*, el cual se encarga de ir abriendo cada nodo adyacente que contenga el nodo que actualmente se está y agregarlos a una lista de nodos pendientes.
- Destacar que este proceso se realiza dentro de un ciclo while, con el cual se escoge el siguiente nodo a ser actual, el criterio que se ocupa para elegirlo es buscar el peso mínimo dentro de la lista de los nodos pendientes.

Tener en cuenta que el peso de las aristas se obtiene desde la matriz de adyacencia.

- Como se describe el algoritmo va creando el camino mínimo que se tiene desde el nodo inicia al objetivo entregado, el proceso anteriormente descrito se repite hasta que el nodo actual sea igual al nodo objetivo.
- Finalmente, se debe recalcar que el algoritmo de *Dijkstra*, es llamado en 2 oportunidades, la primera instancia para buscar el camino mínimo hacia la llave y luego para buscar el camino mínimo para la salida del “Laberinto”. Luego de terminar lo anterior se produce a escribir el archivo de salida, el cual indica el camino mínimo tomado en cada uno de los trayectos con el respectivo peso que se necesita para llegar al objetivo planteado inicialmente.

2.4 ESTRUCTURA IMPLEMENTADA

Las estructuras que se utilizan para implementar el software, son las siguientes:

```
typedef struct Nodo
{
    int origen;
    int destino;
    int peso;
}Nodo;

typedef struct Lista
{
    int largo;
    Nodo* nodo;
}Lista;

typedef struct Matriz
{
    int numeroDeNodos;
    int** adyacencia;
    Lista* listaNodos;
    int entrada;
    int llave;
    int salida;
}Matriz;
```

Figura 4: Estructuras del software.

2.5 DESCRIPCIÓN DEL ALGORITMO

A continuación, se describe cada una de las funciones empleadas dentro del software y su procedimiento.

- **createBoard(int numeroDeNodos):** Función que se encarga de crear la matriz de adyacencia, además de guardar la cantidad de nodos que tendrá el grafo.
- **crearLista():** Función que se encarga de crear una lista y ,inicializarla en datos Null.
- **crearNodo(int peso, int origen, int destino):** Función que se encarga de crear un nuevo nodo con los parámetros entregados.
- **agregarNodo(Lista* lista , Nodo* aux):** Función que se encarga de agregar un nuevo nodo a la lista.
- **cargar(char nombre[]):** Función que se encarga de cargar la matriz que se obtiene desde el archivo de texto, es una de las funciones fundamentales del algoritmo, ya que es la encargada de obtener todos lo datos para poder trabajarlos, en estos se encuentran las relaciones de cada uno de los nodos, sus relaciones entre ellos y finalmente el peso que contiene cada uno de las relaciones.
- **print(Matriz *matriz):** Función que se encarga de mostrar por pantalla la matriz de adyacencia y cada una de las relaciones que tienen los nodos con sus respectivos pesos.
- **mostrarLista(Lista* lista):** Función que se encarga de mostrar los nodos pendientes.
- **rellenarPendientes(Lista* lista,Lista* pendiente, int origen, Lista* rutaMinima):** Función que se encarga de buscar el nuevo nodo en la lista de nodos pendientes que será el nuevo nodo actual, con el criterio de verificar buscar la arista de menor valor.
- **noCerrado(Lista* lista, int numero):** Función que se encarga de verifica que el nodo no se encuentre ya en la lista de nodos pendientes.
- **buscarCamino(Matriz* matriz, int objetivo, int entrada):** Función que se encarga de realizar el proceso de iteración, hasta que se obtiene el camino mínimo hacia el objetivo que se entrega.
- **calcularPeso(Lista* lista, int peso):** Función que se encarga de calcular el peso total que se utiliza en el camino mínimo encontrado.
- **calcularRutaMinima(Lista* lista , Nodo* objetivo):** Función que se encarga de unir el camino mínimo hacia el objetivo.

- **buscarNodoMinimo(Lista* lista):** Función que se encarga de buscar el nodo minimo, dentro de la lista de los nodos pendientes.
- **borrarElemento(Lista* lista , Nodo* auxNodo):** Función que se encarga de eliminar un nodo de la lista.
- **saveTablero (Lista* rutaMinima, int pesoTotal):** Función que se encarga de crear el archivo de salida, con los dos caminos encontrados para cada uno de los objetivos, además de mostrar el peso que se necesita para realizar cada uno de las rutas.

A continuación, se deja una tabla con cada una de las funciones y sus tiempos de ejecución y ordenes de cada una de ellas:

<i>Funciones</i>	<i>T(n)</i>	<i>O(n)</i>
createBoard(int numeroDeNodos)	$n^2 + n + 9$	n^2
crearLista()	4	1
crearNodo(int peso, int origen, int destino)	10	1
agregarNodo(Lista* lista , Nodo* aux)	10	1
cargar(char nombre[])	$10n + 33$	n
print(Matriz *matriz)	$n^2 + 2n + 2$	n^2
mostrarLista(Lista* lista)	n	n
rellenarPendientes(Lista* lista, Lista* pendiente, int origen, Lista* rutaMinima)	$n^2 + 8n + 4$	n^2
noCerrado(Lista* lista, int numero)	$n + 1$	n
buscarCamino(Matriz* matriz, int objetivo, int entrada)	$n^3 + 4n^2 + 4n + 11$	n^3
calcularPeso(Lista* lista, int peso)	$n + 1$	n
calcularRutaMinima(Lista* lista , Nodo* objetivo)	$n^2 + 6n + 8$	n^2
buscarNodoMinimo(Lista* lista)	$7n + 6$	n
borrarElemento(Lista* lista , Nodo* auxNodo)	$4n + 7$	n
saveTablero (Lista* rutaMinima, int pesoTotal)	$n + 9$	n

Tabla 1: “Tiempos de ejecución y orden de ejecución”

2.6 ANÁLISIS DE RESULTADOS

Si realizamos un análisis a la función principal, el bloque principal hace el llamado a todas las funciones descritas anteriormente, quedando su tiempo de ejecución de la siguiente manera:

<i>Funciones</i>	<i>T(n)</i>	<i>O(n)</i>
Bloque principal	$2n^3 + n^2 + n + 16$	n^3

Tabla 2: “Tiempo y orden de ejecución del bloque principal”

Respecto a los anterior, tenemos que es un algoritmo poco eficiente, es decir, que su tiempo y orden de ejecución es muy alto.

Una de las falencias del software recae en lo mencionado anteriormente, sin embargo, esta falencia se puede solucionar implementado un algoritmo con un orden menor que cumpla las mismas expectativas.

Una posible mejora a la falencia detectada anteriormente, recae en disminuir los siglos que se utilizan dentro de la función principal del software la cual es: *buscarCamino(Matriz* matriz, int objetivo, int entrada)* , la cual además es la que contiene le orden mayor dentro del algoritmo.

CAPÍTULO 3. CONCLUSIÓN

En este presente “Laboratorio” se pudo llevar a ejercer la teoría que se aprendió en la catedra, pudiendo tener una visión del cómo llevar a la práctica cada uno de los elementos vistos y estudiados.

Como se ha mencionado anteriormente, se ha podido lograr la implementación en cada uno de los objetivos tanto como generales y específicos. Destacar que se llegó a la solución de cada uno de ellos mediante el algoritmo de *Dijkstra*. En esta ocasión no quedan objetivos sin cumplir, ya que el software cumple con los requisitos solicitados.

La mayor dificultad dentro del desarrollo del software fue la organización del código para obtener la solución requerida.

Para llegar a cada una de las soluciones finales se crearon varios algoritmos que no cumplían con lo que se buscaba obtener, estos han sido descartados, pero dejando una gran idea para poder llegar al algoritmo correcto, que cumplía con todas las necesidades que se buscaba.

Además, se debe destacar que los resultados del software fueron exitosos en la prueba realizada.

Finalmente, se implementó un algoritmo de $O(n^3)$, sin embargo, se debe dejar abierta la posibilidad de posibles mejoras al algoritmo. Una de las mejores que se pueden implementar es disminuir el uso de iteraciones en las funciones principales del algoritmo, además de poder optimizar cada una de las funciones que fueron implementadas.

CAPÍTULO 4. REFERENCIAS

Anomino. (Julio de 2007). *C con clase*.

Borjas, V. (2013). *Estructuras en C*.

Jacqueline, K. C. (s.f.). *Apuntes de la asignatura: Análisis de algoritmos y estructura de*.
Chile, Universidad de Santiago de Chile, Depto. De Ingeniería en Informática.
Obtenido de
http://www.udesantiagovirtual.cl/moodle2/pluginfile.php?file=%2F203862%2Fmod_resource%2Fcontent%2F1%2FApuntes%20completos%20-%201-2017.pdf

CAPÍTULO 5. MANUAL DE USUARIO

5.1 INTRODUCCIÓN

En el siguiente documento se presenta una forma adecuada, explicativa y simple de cómo utilizar el software hecho.

El software principalmente consiste en la resolución del problema entregado en el laboratorio 3, que el objetivo principal es encontrar el camino mínimo hacia la llave y luego el camino mínimo hacia la salida del “Laberinto”.

Las reglas del “Laberinto” son diferentes a las que estamos acostumbrados, ya que se podría decir que es un “Laberinto circular”, de manera que si uno sale por un lugar inferior procede a entrar por la parte superior, así mismo si uno procede a salir por los costados entrara por el costado opuesto.

Dentro del transcurso del manual se procede a explicar desde el primer paso que se debe hacer para poder utilizar el software de una manera correcta.

Se recomienda tener poseer un lenguaje técnico mayor al promedio para poder aprovechar las ventajas que entrega el software.

Por ultimo debemos destacar que el software esta implementado en un paradigma imperativo, que principalmente consta de declaraciones que se ejecutan secuencialmente.

5.2 INSTRUCCIONES

A continuación, se procede a explicar cada uno de los pasos a seguir para poder compilar el software que soluciona el “Laberinto”.

5.2.1 Descarga de MinGW

Para poder compilar el software credo se necesita descargar lo siguiente, con el fin que los comandos ingresados sean conocidos por su equipo.

Se puede obtener del siguiente link:

- <https://www.fdi.ucm.es/profesor/luis/fp/devtools/MinGW.html>

5.2.2 Compilación

5.2.2.1 Windows:

1. Debemos abrir la consola de Windows, existen dos opciones las cuales son:
 - a. Apretar Windows + R
 - b. Abrir la ventana de ejecutar y escribir “cmd”.

Una de las opciones se puede apreciar en la Figura 5, que viene a continuación:

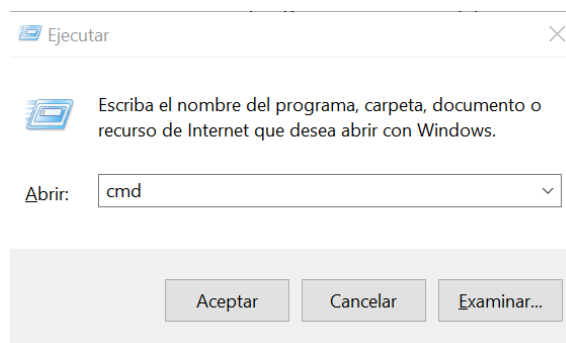


Figura 5: Ventana ejecutar en sistema operativo Windows.

2. Luego de realizar el anterior paso, se abrirá la consola como mostrará en la Figura 6:

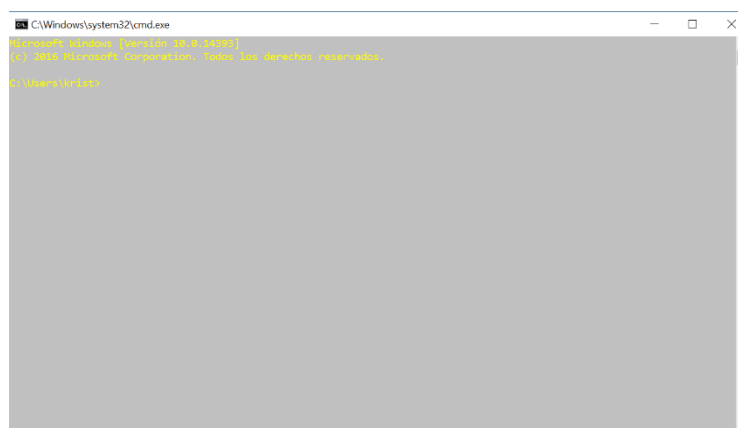


Figura 6: Consola de Windows.

3. Almacenar el software dentro de una carpeta para poder tener todo lo necesario dentro de ella, lo recomendado es crear dicha carpeta en el escritorio de su máquina, como se muestra en la Figura 7:

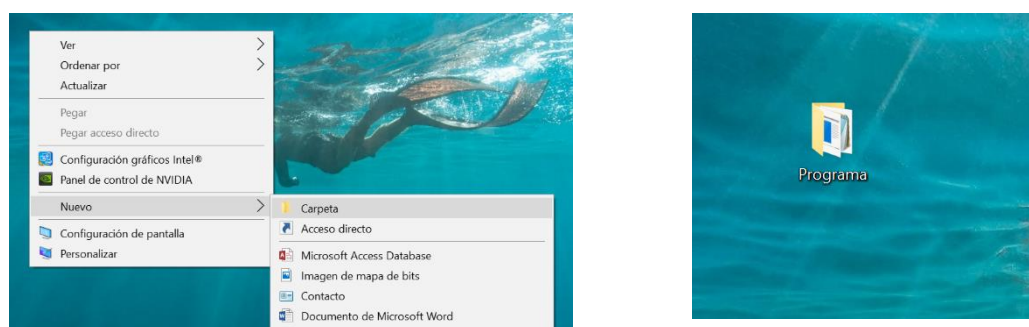
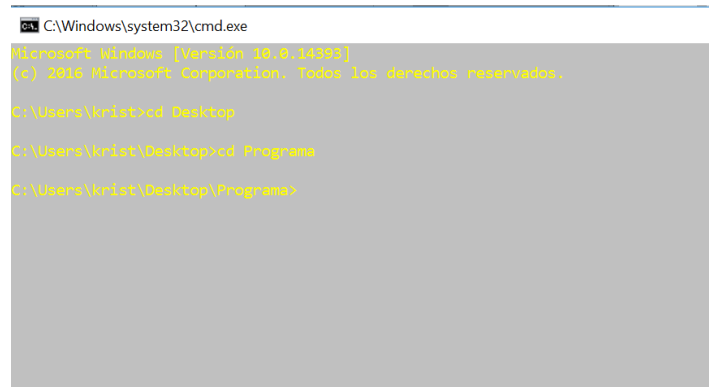


Figura 7: Crear carpeta y almacenar software dentro.

4. Luego volvemos a la consola de Windows, para poder ejecutar el software y seguir los siguientes pasos:
 - a. Primer paso, dejamos la consola de Windows en la ruta donde tenemos almacenado nuestro programa, de la manera que muestra en la Figura 8:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.14393]
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

C:\Users\kr1st>cd Desktop

C:\Users\kr1st\Desktop>cd Programa

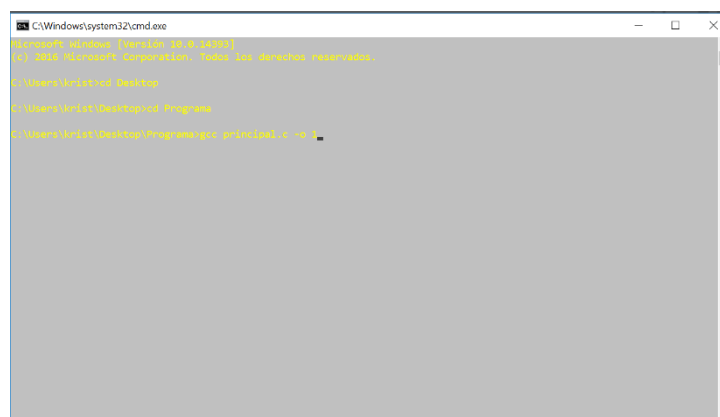
C:\Users\kr1st\Desktop\Programa>
```

Figura 8: Buscando ruta de la carpeta creada.

b. Luego comenzamos la compilación, escribiendo lo siguiente en la consola de Windows:

- gcc (Nombre del archivo).c -o (Nombre del ejecutable)
- (Nombre del ejecutable).exe

La Figura 9 entrega las instrucciones con mayor claridad:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.14393]
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

C:\Users\kr1st>cd Desktop

C:\Users\kr1st\Desktop>cd Programa

C:\Users\kr1st\Desktop\Programa>gcc principal.c -o 1_
```

Figura 9: Compilando software

- c. Luego se crea el archivo ejecutable en la carpeta donde tenemos almacenado el software, como muestra la Figura 10:

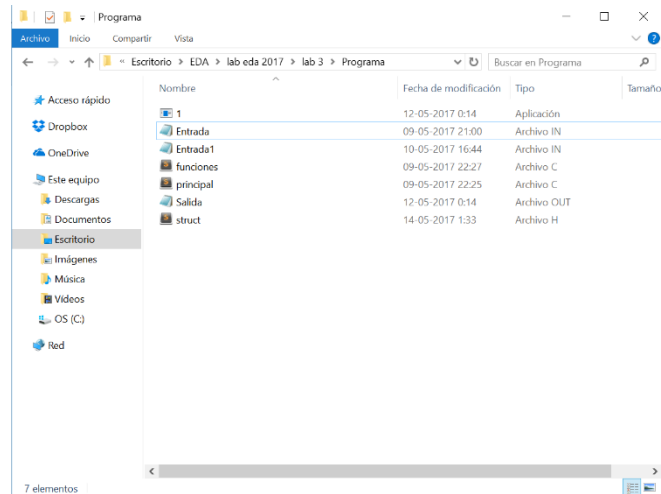


Figura 10: Creado el ejecutable del software.

5.2.2.2 Linux

1. Debemos abrir la consola de Linux o también llamada terminal, existen dos opciones las cuales son:
 - a. Apretar clic derecho y colocar la opción abrir terminal.
 - b. Buscar en los programas de Linux y colocar “Terminal”

Una de las opciones se puede apreciar en la Figura 11 que se observa abajo:

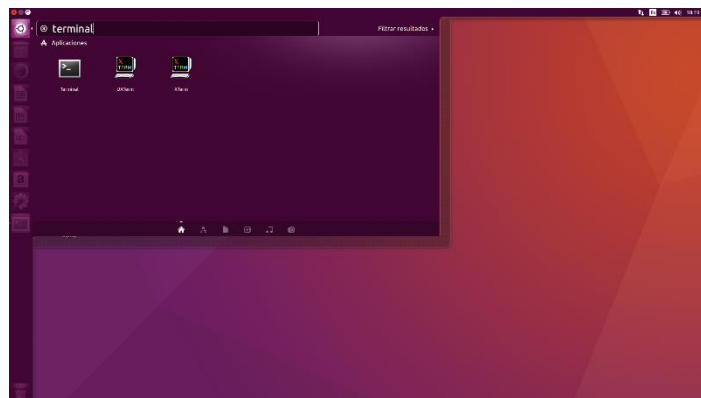


Figura 11: Abrir consola de Linux o Terminal.

2. Luego de realizar el anterior paso, se abrirá la consola como mostrará la Figura 12:

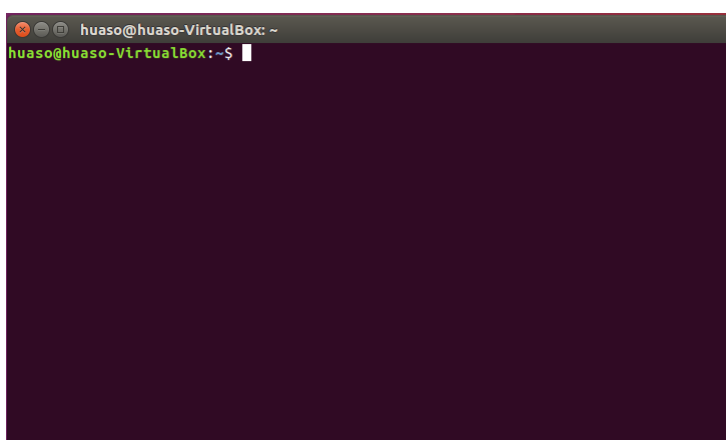


Figura 12: Ventana de la consola de Linux o Terminal.

3. Almacenar el software dentro de una carpeta para poder tener todo lo necesario dentro de ella, lo recomendado es crear dicha carpeta en el escritorio de su máquina, como se muestra en la Figura 13:

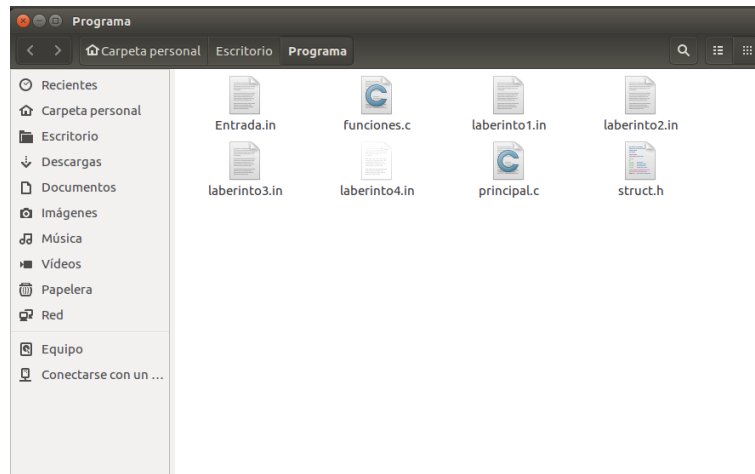


Figura 13: Carpeta con el software dentro.

4. Luego volvemos a la consola de Linux, para poder ejecutar el software como muestra en la Figura 14:
 - a. Primer paso, dejamos la consola de Linux en la ruta donde tenemos almacenado nuestro programa, de la manera que muestra la siguiente imagen:

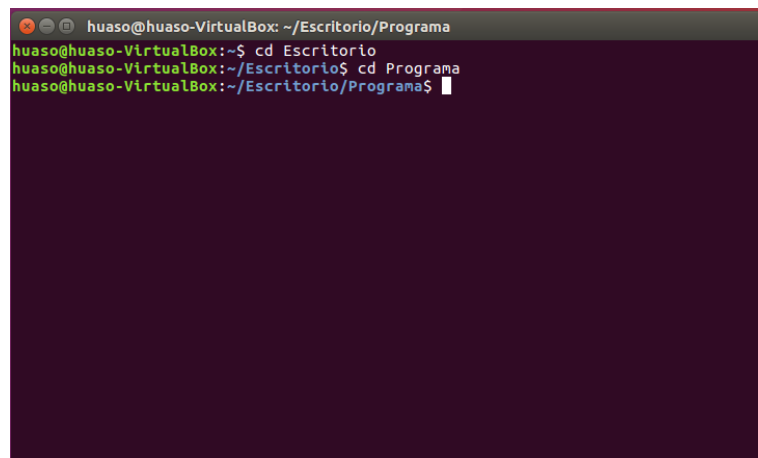
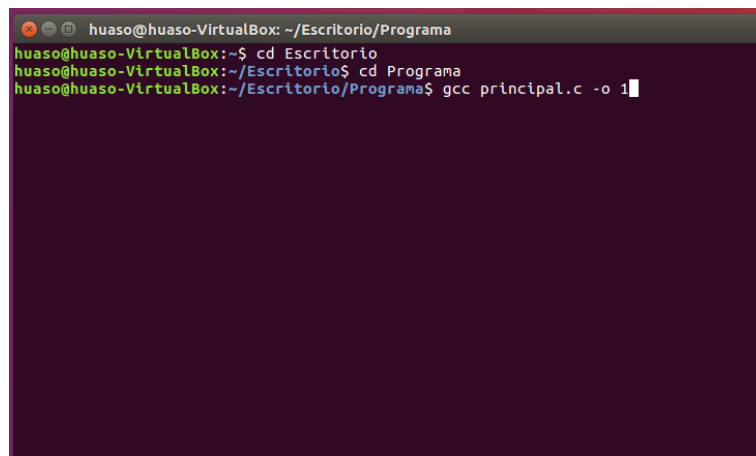


Figura 14: Consola de Linux con la ruta de la carpeta que contiene el software.

b. Luego comenzamos la compilación, escribiendo lo siguiente en la consola de Linux:

- gcc (Nombre del archivo).c -o (Nombre del ejecutable)
- ./(Nombre del ejecutable)

La Figura 15 entrega las instrucciones con mayor claridad:

A terminal window titled 'huaso@huaso-VirtualBox: ~/Escritorio/Programa' showing a sequence of commands: 'cd Escritorio', 'cd Programa', and 'gcc principal.c -o 1'. The prompt is green, and the output is white on a dark purple background.

```
huaso@huaso-VirtualBox: ~/Escritorio/Programa
huaso@huaso-VirtualBox:~$ cd Escritorio
huaso@huaso-VirtualBox:~/Escritorio$ cd Programa
huaso@huaso-VirtualBox:~/Escritorio/Programa$ gcc principal.c -o 1
```

Figura 15: Compilación de software.

c. Luego se crea el archivo ejecutable en la carpeta donde tenemos almacenado el software, como muestra Figura 16:

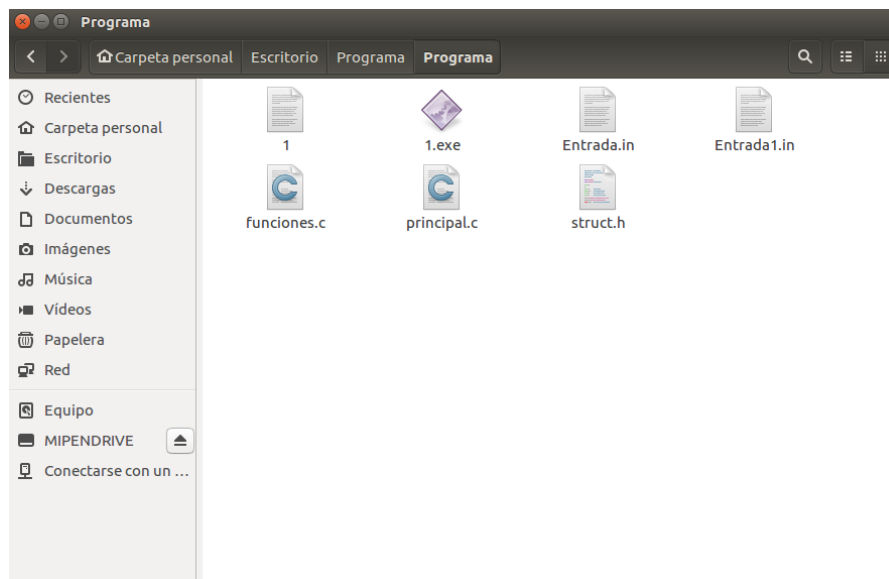


Figura 16: Carpeta con el archivo ejecutable.

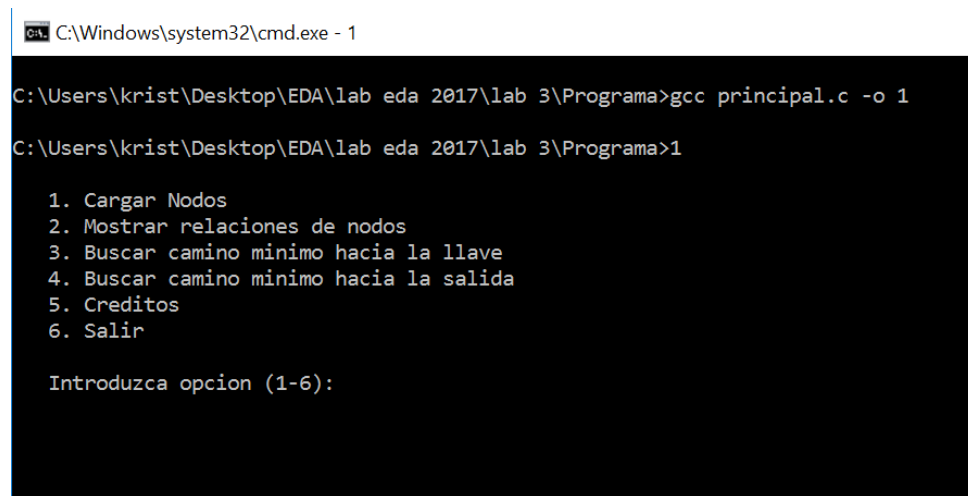
5.2.3 Ejecución

Teniendo el archivo “.exe” o “.out” en Windows y Linux respectivamente se procede a ejecutarlo cada uno en su ambiente.

Antes de llevar a cabo dicha acción se debe verificar que los caminos de los “Laberinto” estén en la misma carpeta donde se tiene almacenado dicho software, además de corroborar que los archivos donde se tiene almacenado cada uno de los caminos estén con la extensión.in.

5.2.3.1 Windows

1. Se vuelve a la consola de comandos de Windows, colocando la siguiente línea para efectuar la ejecución, en la Figura 17 se puede apreciar de mejor forma:



```

C:\Windows\system32\cmd.exe - 1

C:\Users\krist\Desktop\EDA\lab eda 2017\lab 3\Programa>gcc principal.c -o 1
C:\Users\krist\Desktop\EDA\lab eda 2017\lab 3\Programa>1

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino mínimo hacia la llave
4. Buscar camino mínimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6):

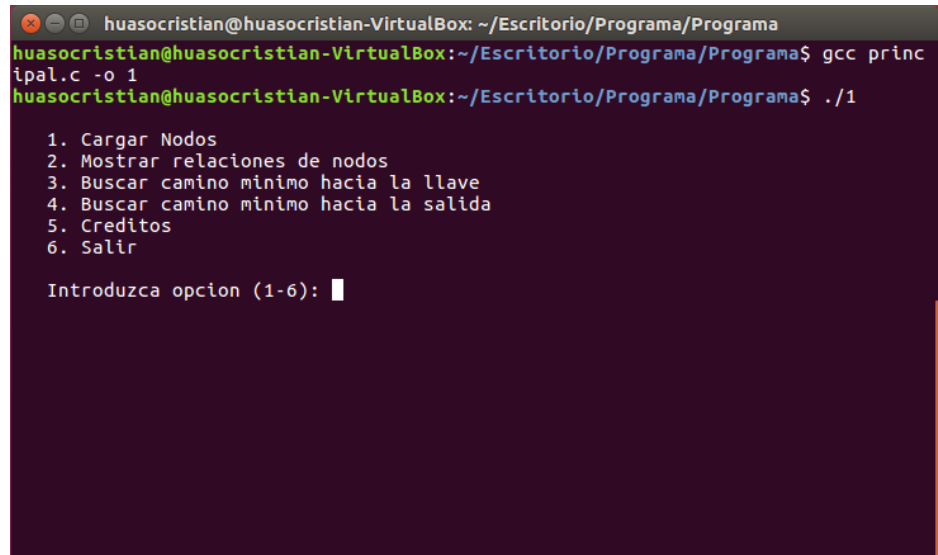
```

Figura 17: Software ejecutado.

- Al terminar el proceso anterior, se puede observar en la imagen anterior que el software está ejecutado y listo para ser manipulado.

5.2.3.2 Linux

1. Se vuelve a la consola de comandos de Linux, colocando la siguiente línea para efectuar la ejecución, en la Figura 18 se puede apreciar de mejor forma:

A screenshot of a Linux terminal window. The window title is "huasocristian@huasocristian-VirtualBox: ~/Escritorio/Programa/Programa". The terminal shows the following commands and output:

```
huasocristian@huasocristian-VirtualBox:~/Escritorio/Programa/Programa$ gcc principal.c -o 1
huasocristian@huasocristian-VirtualBox:~/Escritorio/Programa/Programa$ ./1

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino minimo hacia la llave
4. Buscar camino minimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6):
```

Figura 18: Software ejecutado.

- Al terminar el proceso anterior, se puede observar en la imagen anterior que el software está ejecutado y listo para ser manipulado.

5.3 FUNCIONALIDADES

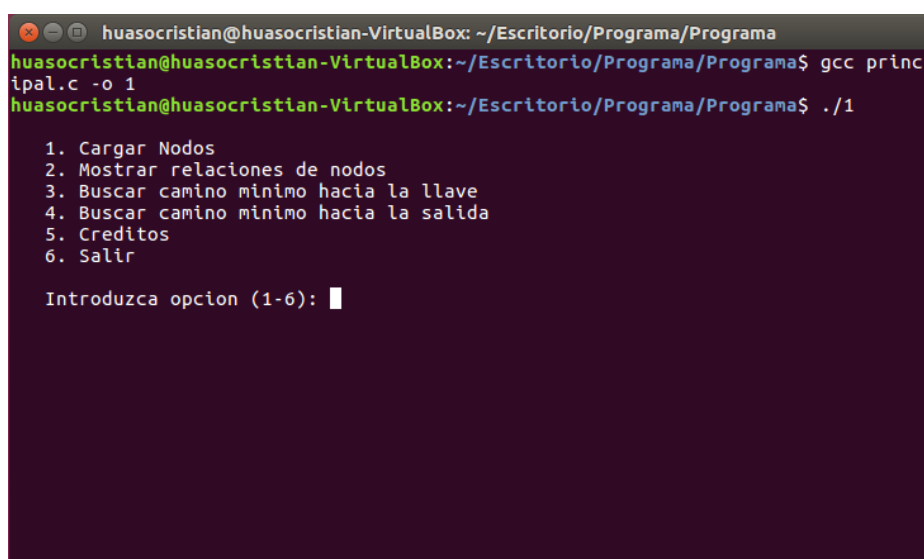
El software tiene la funcionalidad de encontrar el camino mínimo que se tiene para resolver el “Laberinto”, esto en dos oportunidades, primeramente, para encontrar el camino hacia la llave y a continuación hacia la salida del “Laberinto”.

El usuario tiene interacción con el software que recae en ingresar el nombre del “Laberinto” que se desea abrir, tener en cuenta que este archivo tendrá una entrada “.in”.

El resultado entregado por el software se verá expuesto en un archivo de salida con extensión “.out”.

A continuación, se realizará un procedimiento de los pasos a seguir para hacer uso del software de una buena manera:

- Primero debemos volver a la consola y ejecutar el software, entregando una ventana como muestra la Figura 19:



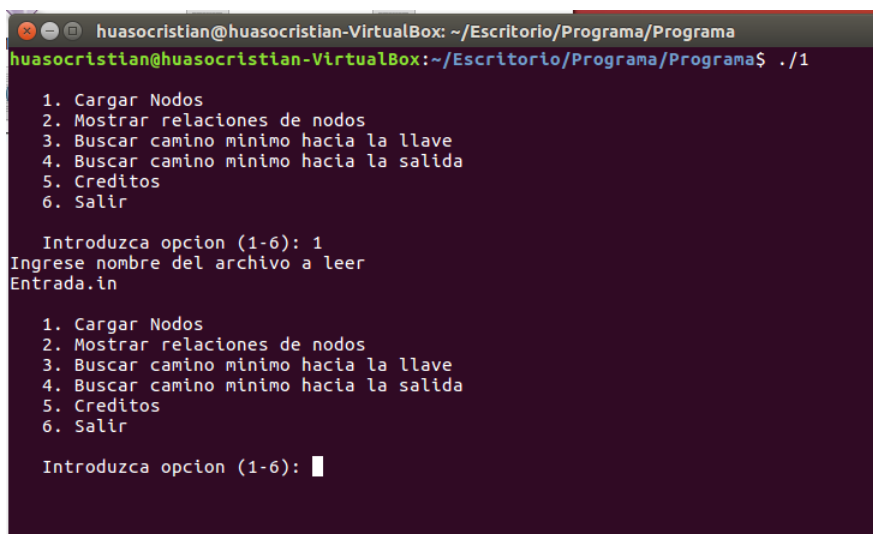
```
huasocristian@huasocristian-VirtualBox: ~/Escritorio/Programa/Programa
huasocristian@huasocristian-VirtualBox:~/Escritorio/Programa/Programa$ gcc princ
ipal.c -o 1
huasocristian@huasocristian-VirtualBox:~/Escritorio/Programa/Programa$ ./1

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino minimo hacia la llave
4. Buscar camino minimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): █
```

Figura 19: Menú principal del software.

- Luego de eso debemos seleccionar la opción de ingresar el nombre del tablero, como se muestra en la Figura 20:



```

huasocristian@huasocristian-VirtualBox: ~/Escritorio/Programa/Programa
huasocristian@huasocristian-VirtualBox:~/Escritorio/Programa/Programa$ ./1

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino minimo hacia la llave
4. Buscar camino minimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): 1
Ingrese nombre del archivo a leer
Entrada.in

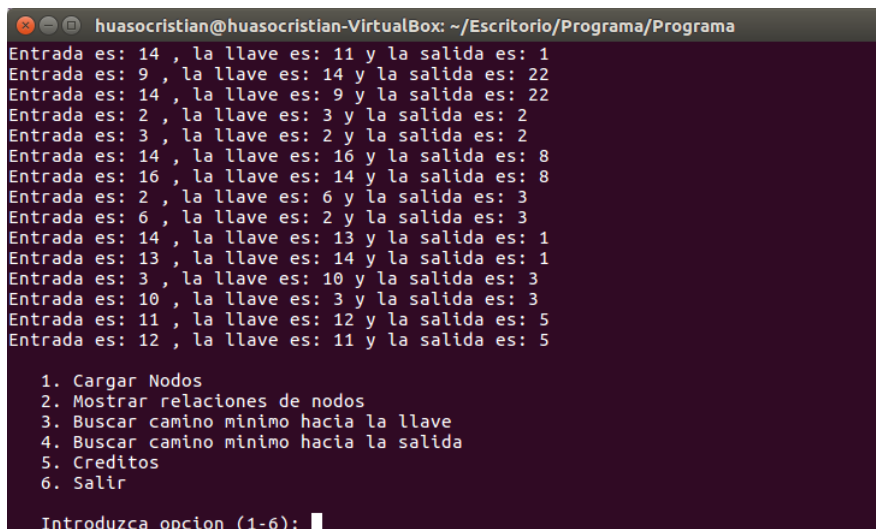
1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino minimo hacia la llave
4. Buscar camino minimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): █

```

Figura 20: Cargando el archivo al software.

- Luego de eso debemos volver a ingresar una opción, en este caso ocuparemos la opción número 2 que nos muestra el tablero leído, se debe hacer como se muestra en la Figura 21:



```

huasocristian@huasocristian-VirtualBox: ~/Escritorio/Programa/Programa
Entrada es: 14 , la llave es: 11 y la salida es: 1
Entrada es: 9 , la llave es: 14 y la salida es: 22
Entrada es: 14 , la llave es: 9 y la salida es: 22
Entrada es: 2 , la llave es: 3 y la salida es: 2
Entrada es: 3 , la llave es: 2 y la salida es: 2
Entrada es: 14 , la llave es: 16 y la salida es: 8
Entrada es: 16 , la llave es: 14 y la salida es: 8
Entrada es: 2 , la llave es: 6 y la salida es: 3
Entrada es: 6 , la llave es: 2 y la salida es: 3
Entrada es: 14 , la llave es: 13 y la salida es: 1
Entrada es: 13 , la llave es: 14 y la salida es: 1
Entrada es: 3 , la llave es: 10 y la salida es: 3
Entrada es: 10 , la llave es: 3 y la salida es: 3
Entrada es: 11 , la llave es: 12 y la salida es: 5
Entrada es: 12 , la llave es: 11 y la salida es: 5

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino minimo hacia la llave
4. Buscar camino minimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): █

```

Figura 21: Mostrando el laberinto por pantalla.

- Luego debemos realizar los pasos 3 para buscar el camino hacia la llave y en seguida colocar 4 para buscar el camino hacia la salida, tener en cuenta que, si uno revisa el archivo creado antes de terminar las dos opciones, solo se verá el camino que solicito buscar, como se aprecia en la Figura 22 y 23:

```

huasocristian@huasocristian-VirtualBox: ~/Escritorio/Programa/Programa
1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino mínimo hacia la llave
4. Buscar camino mínimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): 3
origen es: 11 , la destino es: 14 y el peso es: 1
origen es: 12 , la destino es: 11 y el peso es: 5
origen es: 10 , la destino es: 12 y el peso es: 2
origen es: 3 , la destino es: 10 y el peso es: 3
origen es: 0 , la destino es: 3 y el peso es: 0
El peso total recorrido fue de: 11
Camino mínimo hacia la Llave encontrado , archivo generado

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino mínimo hacia la llave
4. Buscar camino mínimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6):

```

Figura 22: *Buscando camino mínimo hacia la llave.*

```

huasocristian@huasocristian-VirtualBox: ~/Escritorio/Programa/Programa
4. Buscar camino mínimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): 4
origen es: 4 , la destino es: 9 y el peso es: 5
origen es: 5 , la destino es: 4 y el peso es: 1
origen es: 8 , la destino es: 5 y el peso es: 3
origen es: 10 , la destino es: 8 y el peso es: 4
origen es: 12 , la destino es: 10 y el peso es: 2
origen es: 11 , la destino es: 12 y el peso es: 5
origen es: 14 , la destino es: 11 y el peso es: 1
origen es: 0 , la destino es: 14 y el peso es: 0
El peso total recorrido fue de: 21
Camino mínimo hacia la Salida encontrado , archivo generado

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino mínimo hacia la llave
4. Buscar camino mínimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6):

```

Figura 23: *Buscando camino minimo hacia la salida.*

- Finalmente, para salir del software se selecciona la opción número 6, como muestra la Figura 24:

```

huasocristian@huasocristian-VirtualBox: ~/Escritorio/Programa/Programa
origen es: 0 , la destino es: 14 y el peso es: 0
El peso total recorrido fue de: 21
Camino minimo hacia la Salida encontrado , archivo generado

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino minimo hacia la llave
4. Buscar camino minimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): 5
* Autor: CRISTIAN EDUARDO ESPINOZA SILVA
* Universidad santiago de chile

1. Cargar Nodos
2. Mostrar relaciones de nodos
3. Buscar camino minimo hacia la llave
4. Buscar camino minimo hacia la salida
5. Creditos
6. Salir

Introduzca opcion (1-6): 6
huasocristian@huasocristian-VirtualBox:~/Escritorio/Programa/Programa$

```

Figura 24: Cerrando el software.

- Luego el software se cierra y si nos dirigimos a la carpeta donde tenemos almacenado el programa no aparece el archivo con la salida, como se puede observar en la Figura 25:

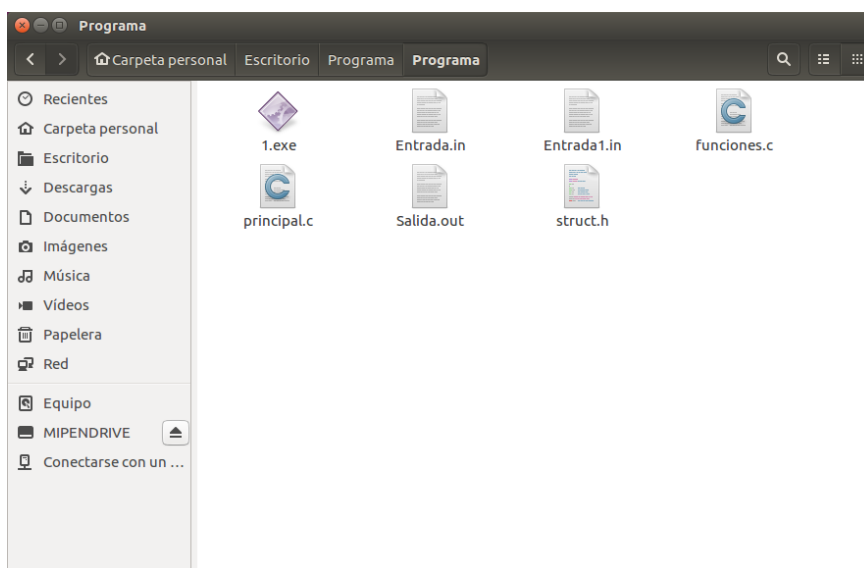


Figura 25: Carpeta con el archivo de “salida.out” del software

- Luego abrimos el archivo para verificar la solución final del “Laberinto” que entrega el software, como muestra la Figura 26:

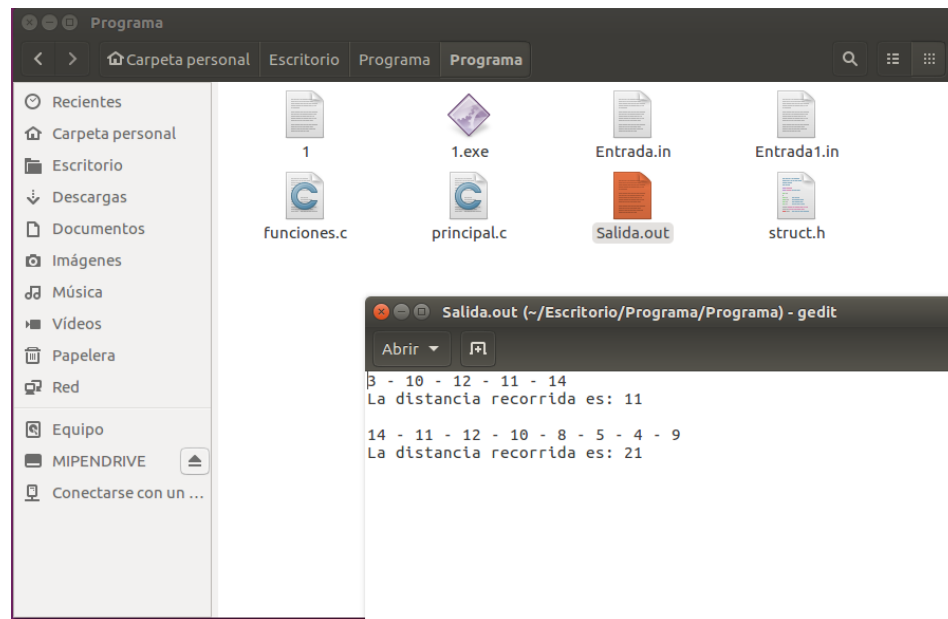


Figura 26: Salida final del software.

5.4 EXITOS Y POSIBLES ERRORES

En ejecución la aplicación paso todas las pruebas que se le realizaron, tener en cuenta que fue sometida alrededor a distintos “Laberintos” teniendo siempre un resultado exitoso.

Algunos de los errores posibles que puede hacer que la aplicación son los siguientes:

- Cuando el “Laberinto” no cuenta con una salida, entrada o llave dentro de su mapa.
- Cuando el “Laberinto” no tiene un camino posible para poder llegar a uno de los objetivos mencionados en el punto anterior.
- Cuando el usuario modifica alguna línea del código planteado.
- Cuando las entradas no son las que el software espera.
- Cuando el disco se queda sin memoria.

En la Figura 27, se puede observar como el software deja de funcionar cuando se encuentra con uno de los errores mencionados:

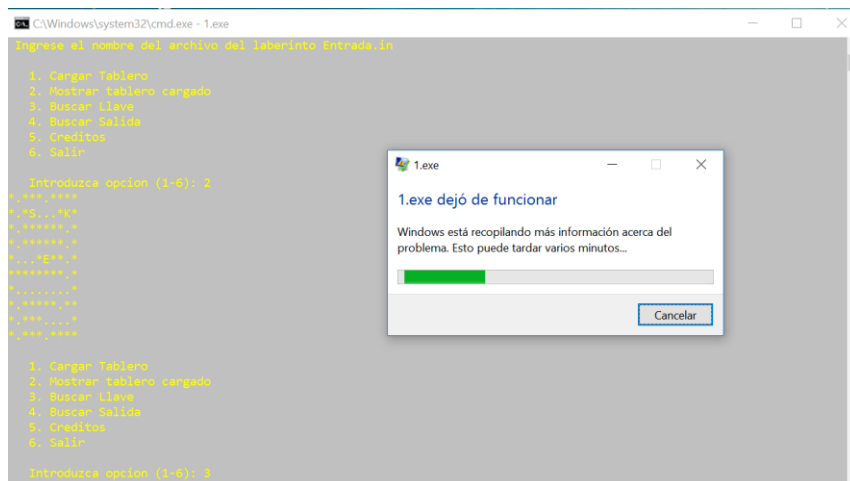


Figura 27: Posible error dentro de software.