|  |
| --- |
| **Estructura de Datos y Análisis de Algoritmos**  **Informe de Laboratorio 2** |
|  |
| **CRISTÓBAL NICOLÁS MEDRANO ALVARADO** |

|  |  |
| --- | --- |
|  | Profesor: |
|  | Alejandro Cisterna |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Santiago - Chile |  |
|  | 1-2017 |  |

Tabla de Contenidos

[Tabla de Contenidos I](#_Toc481108637)

[Tabla de imágenes II](#_Toc481108638)

[CAPÍTULO 1. INTRODUCCIÓN 1](#_Toc481108639)

[CAPÍTULO 2. Descripción de la solución 2](#_Toc481108640)

[2.1 Estructura de datos utilizadas 2](#_Toc481108643)

[2.1.1 Funciones asociadas al TDA Lista. 3](#_Toc481108644)

[2.1.2 Funciones asociadas a la ejecución del programa. 6](#_Toc481108645)

[2.1.3 TECNICAS UTILIZADAS. 11](#_Toc481108646)

[2.1.4 HERRAMIENTAS UTILIZADAS. 12](#_Toc481108647)

[CAPÍTULO 3. ANALISIS DE LOS RESULTADOS 15](#_Toc481108648)

[3.1 TIEMPOS DE EJECUCIÓN 15](#_Toc481108652)

[3.1.1 OBJETIVOS 15](#_Toc481108653)

[3.2 falencias detectadas. 15](#_Toc481108654)

[3.2.1 ERROR AL ABRIR ARCHIVO 15](#_Toc481108655)

[3.2.2 WARNING AL COMPILAR EN LINUX 15](#_Toc481108656)

[3.3 Propuestas de mejora 15](#_Toc481108657)

[3.3.1 VERIFICACIÓN DE CARACTERES 15](#_Toc481108658)

[3.3.2 INGRESO DEL NOMBRE DE ARCHIVO 16](#_Toc481108659)

[CAPÍTULO 4. CONCLUSIÓN 17](#_Toc481108660)

[CAPÍTULO 5. REFERENCIAS 18](#_Toc481108661)

[CAPÍTULO 6. MANUAL DE USUARIO 19](#_Toc481108662)

[6.1 Introducción 19](#_Toc481108666)

[6.2 Como compilar y ejecutar. 19](#_Toc481108667)

[6.2.1 Compilar en Windows 19](#_Toc481108668)

[6.2.2 Compilar en Linux 21](#_Toc481108669)

[6.3 Funcionalidades del programa. 23](#_Toc481108670)

Tabla de FIGURAS

[Ilustración 1 Definicion de Estructura Laberinto en C 3](#_Toc481109404)

[Ilustración 2 Definicion de Estructura Lista en C. 3](#_Toc481109405)

[Ilustración 3 Definición de Función crearNodo en C 4](#_Toc481109406)

[Ilustración 4 Definición de Función InsertarFinal en C. 5](#_Toc481109407)

[Ilustración 5 : Definición de Función obtener en C 6](#_Toc481109408)

[Ilustración 6 : Definición de Función largo en C. 6](#_Toc481109409)

[Ilustración 7 Definición de Función leerArchivo en C. 7](#_Toc481109410)

[Ilustración 8 : Archivo de entrada 7](#_Toc481109411)

[Ilustración 9 Definición de Función guardarDatosLaberinto en C. 8](#_Toc481109412)

[Ilustración 10 Definición del procedimiento direccionarLaberinto en C. 8](#_Toc481109413)

[Ilustración 11 Definición del procedimiento guardarCaminoMin en C. 9](#_Toc481109414)

[Ilustración 12 Definición del procedimiento guardarCaminoMin en C. 10](#_Toc481109415)

[Ilustración 13 : Archivo de salida 11](#_Toc481109416)

[Ilustración 14 Método de Volver atrás (Backtracking). 12](#_Toc481109417)

[Ilustración 15 ScreenShot del programa SublimeText2. 13](#_Toc481109418)

[Ilustración 16 ScreenShot del repositorio en GitHub.com 13](#_Toc481109419)

[Ilustración 17 : Compilando en Windows parte 1. 19](#_Toc481109420)

[Ilustración 18 Compilando en Windows parte 2 20](#_Toc481109421)

[Ilustración 19 Compilando en Windows parte 3 21](#_Toc481109422)

[Ilustración 20 Compilando en Windows parte 4 21](#_Toc481109423)

[Ilustración 21 Compilando en Linux parte 1. 22](#_Toc481109424)

[Ilustración 22 Compilando en Linux parte 2. 23](#_Toc481109425)

# INTRODUCCIÓN

El laberinto es un lugar formado artificiosamente por calles y encrucijadas, para confundir a quien se adentre en él, de modo que no pueda acertar con la salida.

Resolver este tipo de problema, es todo un desafío. Es por esto que el Laboratorio 2, trata de desarrollar un algoritmo, que sea capaz de resolver un laberinto entregado a través de un archivo, buscar en él un camino mínimo y además escribir la solución en otro archivo de texto con los pasos a seguir. La implementación de la solución debe incluir listas enlazadas y ser desarrollada en el lenguaje de programación C.

Por cual, para desarrollar oportunamente la solución en este laboratorio, se especificaron diferentes objetivos.

Objetivo general: Desarrollar un programa en C, que sea capaz de resolver un laberinto con patrones pre-definidos de entrada e implemente listas enlazadas.

Objetivos específicos:

- Implementar listas enlazadas.

- Leer adecuadamente un archivo de Entrada con el laberinto.

- Obtener datos del laberinto y guardarlos en una lista de listas.

- Desarrollar algoritmo[[1]](#footnote-1) que resuelva el laberinto.

- Obtener el camino mínimo.

- Escribir correctamente un archivo de Salida con la solución del laberinto.

- Escribir correctamente los pasos a seguir de cada solución usando listas enlazadas.

Con estos objetivos definidos se espera abordar adecuadamente el problema, el cual busca principalmente que el estudiante sea capaz de implementar listas enlazadas y resolver problemas.

El cuerpo del informe está compuesto por una detallada descripción de la solución (capitulo 2), conteniendo en él, los principales algoritmos del programa, seguido por el análisis de la solución (capitulo 3), donde es posible encontrar los tiempos de ejecución y resultados de la solución, para finalizar en una conclusión (capitulo 4). Como anexo se tiene el manual de usuario (Capitulo 6), en el cual se observa como ejecutar correctamente el programa.

# Descripción de la solución

En este capítulo se hará un enfoque en los algoritmos implementados y cuál fue su uso dentro del programa desarrollado en el lenguaje de programación C.

Para abordar cada uno de los objetivos señalados previamente la solución fue dividida en una serie de sub-problemas.

1.- Lectura de un archivo de entrada.

2.- Búsqueda de la solución al laberinto.

3.- Búsqueda del camino mínimo de la solución del laberinto.

4.- Implementación de Listas Enlazadas.

5.- Escritura del archivo de salida.

Cabe destacar que cada uno de estos sub-problemas, fueron divididos en otros sub-problemas, los cuales serán detallados a medida que se explique la solución implementada.



## Estructura de datos utilizadas

Para desarrollar la solución se utilizaron dos estructuras: una estructura genérica que contiene datos del laberinto y una estructura basada en el TDA Listas Enlazadas mediante nodos. La descripción de cada una se muestra a continuación:

Laberinto:

N = Cantidad de Filas.

M = Cantidad de Colummnas

MatrizLaberinto = Matriz que alberga el laberinto inicial leído del archivo.

MatrizEntradaLlave = Matriz que alberga la solución del laberinto Entrada-Llave.

matrizLlaveSalida = Matriz que alberga la solución del laberinto Llave-Salida.

contCaminoMinimo = Contador que guarda la cantidad de camino recorrido.

contCaminoMinimoLlave = Contador que guarda el mínimo camino desde la entrada a la llave.

contCaminoMinimoSalida = Contador que guarda el mínimo camino desde la llave a la salida.

La implementación de esta estructura en C, se muestra a continuación.

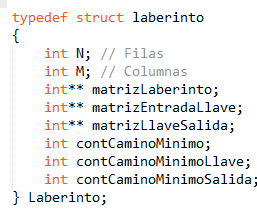


Ilustración 1 Definicion de Estructura Laberinto en C

Lista

Dato = Es el elemento que contiene el nodo.

Lista Siguiente = Es un puntero que apunta al siguiente nodo.

La implementación de esta estructura en C, se muestra a continuación.

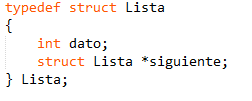


Ilustración 2 Definicion de Estructura Lista en C.

### Funciones asociadas al TDA Lista.

Basados en las definiciones anteriores y las necesidades del problema, se han utilizado las siguientes funciones:

#### Crear Nodo Lista:

Esta función crea un nodo.

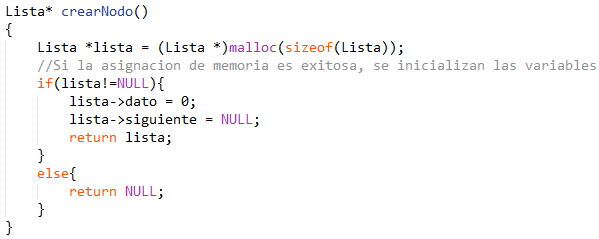


Ilustración 3 Definición de Función crearNodo en C

Este algoritmo tiene T(n) = 4 y es de orden O(1).

#### Insertar al final:

Esta función inserta un nodo de Lista en la última posición.

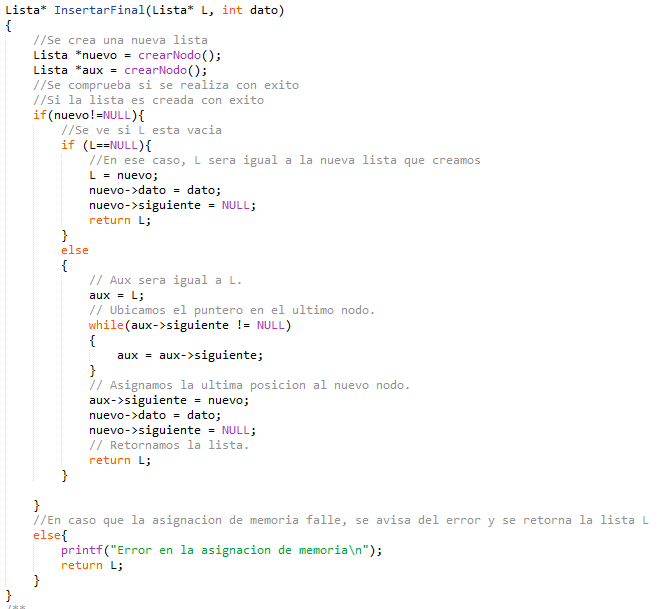


Ilustración 4 Definición de Función InsertarFinal en C.

Este algoritmo tiene T(n) = n + 12 y es de orden O(n).

#### Función obtener

Esta función obtiene un elemento de la lista enlazada según la posición.

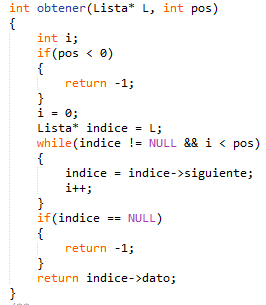


Ilustración 5 : Definición de Función obtener en C

Este algoritmo tiene T(n) = 2n + 3 y es de orden O(n).

#### Función largo

Esta función retorna el largo de la lista de nodos.

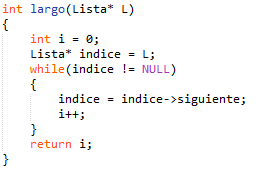


Ilustración 6 : Definición de Función largo en C.

Este algoritmo tiene T(n) = 2(n + 1) y es de orden O(n).

### Funciones asociadas a la ejecución del programa.

Basados en las definiciones anteriores y las necesidades del problema, se han utilizado las siguientes funciones:

#### Función leerArchivo.

Función que lee los datos del archivo ingresado.

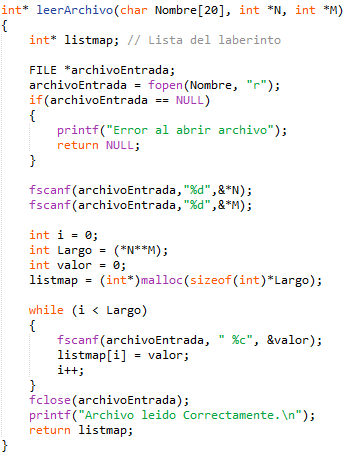


Ilustración 7 Definición de Función leerArchivo en C.

Este algoritmo tiene T(n) = 3n + 12 y es de orden O(n).

El archivo leído es de la siguiente forma:

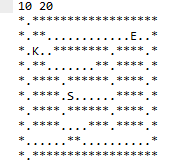


Ilustración 8 : Archivo de entrada

.

#### Función guardarDatosLaberinto

Función que guarda en la estructura laberinto, los datos del laberinto.

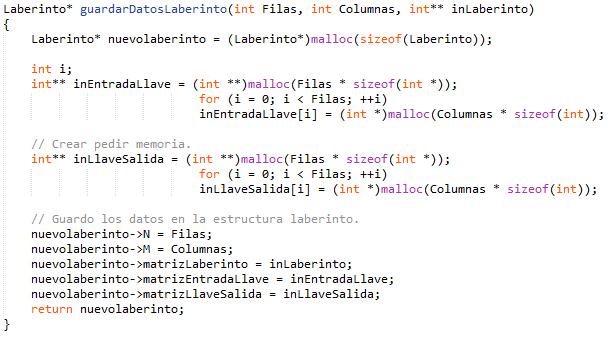


Ilustración 9 Definición de Función guardarDatosLaberinto en C.

Este algoritmo tiene T(n) = 10 y es de orden O(1).

#### Procedimiento que recorre el laberinto.

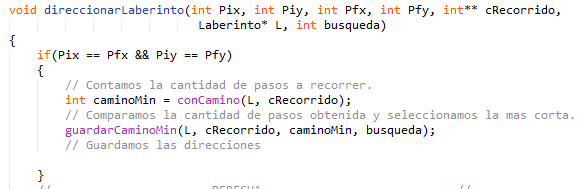


Ilustración 10 Definición del procedimiento direccionarLaberinto en C.

Este algoritmo es muy extenso y hace alusión a la técnica BackTracking la cual posteriormente es explicada en los siguientes puntos. Razón por la cual no es explicitado totalmente, de todas formas, se calcula el tiempo de ejecución.

El tiempo de ejecución depende del número de nodos generados y del tiempo requerido para cada nodo, que viene dado por el coste de las funciones.

Este algoritmo tiene T(n) = 2 + 22 + 23 + ... + 2n = 2n+1 – 2 y un orden O(2n)

#### Procedimiento Camino min

Procedimiento que guarda el camino mínimo ya encontrado por el procedimiento que recorre el laberinto(backtracking).

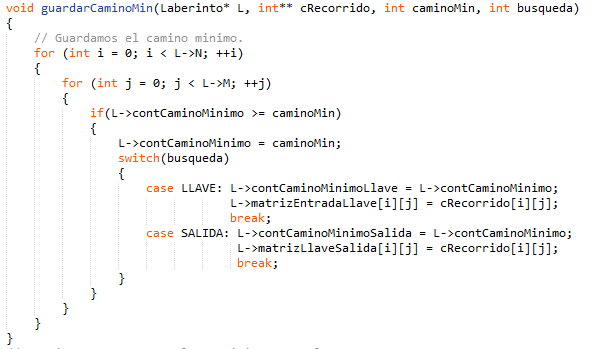


Ilustración 11 Definición del procedimiento guardarCaminoMin en C.

Este algoritmo tiene T(n) = n2 – 2 y un orden O(n2)

#### Procedimiento escribirLaberinto

Procedimiento que escribe la solución del laberinto en un archivo.

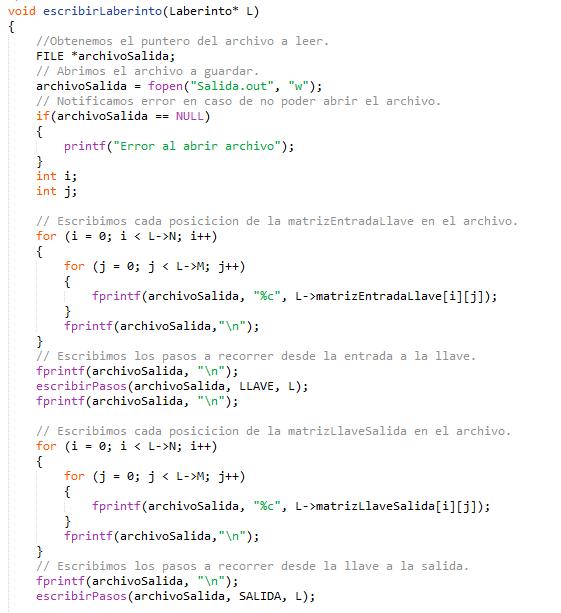


Ilustración 12 Definición del procedimiento guardarCaminoMin en C.

Este algoritmo tiene T(n) = n2 – 12 y un orden O(n2)

El archivo de salida es de la forma:

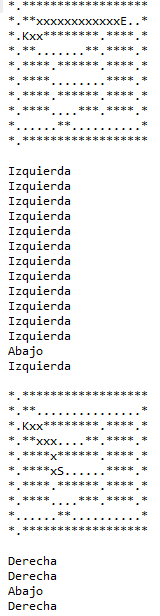


Ilustración 13 : Archivo de salida

.

### TECNICAS UTILIZADAS.

#### **BACKTRACKING**

Para obtener la solución al laberinto, se usó una técnica similar a lo que se conoce como Backtracking. Backtracking es encontrar una solución intentándolo con una de varias opciones. Si la elección es incorrecta, el computo retrocede o vuelve a comenzar desde el punto de decisión anterior y lo intenta con otra opción, bajo esta premisa, el algoritmo principal data de esta técnica la cual está determinada por posiciones en x e y, de la entrada y lo que tiene que buscar.

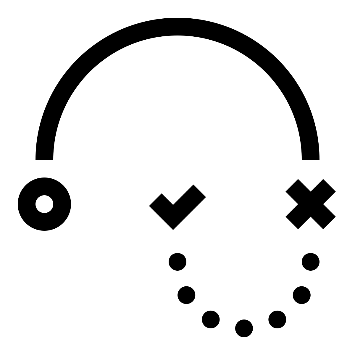


Ilustración 14 Método de Volver atrás (Backtracking).

### HERRAMIENTAS UTILIZADAS.

Las siguientes herramientas, fueron utilizadas para tener facilidad al momento de escribir código, guardar éste y recuperarlo.

#### **SUBLIME TEXT 2**

Permitiendo que el código fuese sencillo de escribir, se utilizó el software llamado Sublime Text 2. Sublime Text 2 es un editor de texto y editor de código fuente que está escrito en C++ y Python para los plugins. Desarrollado originalmente como una extensión de Vim.

La elección de este editor de texto, fue la facilidad que da para autocompletar funciones nativas y la amigable interfaz gráfica que posee, lo cual permite mantener ordenado y en vista todo el código desarrollado.

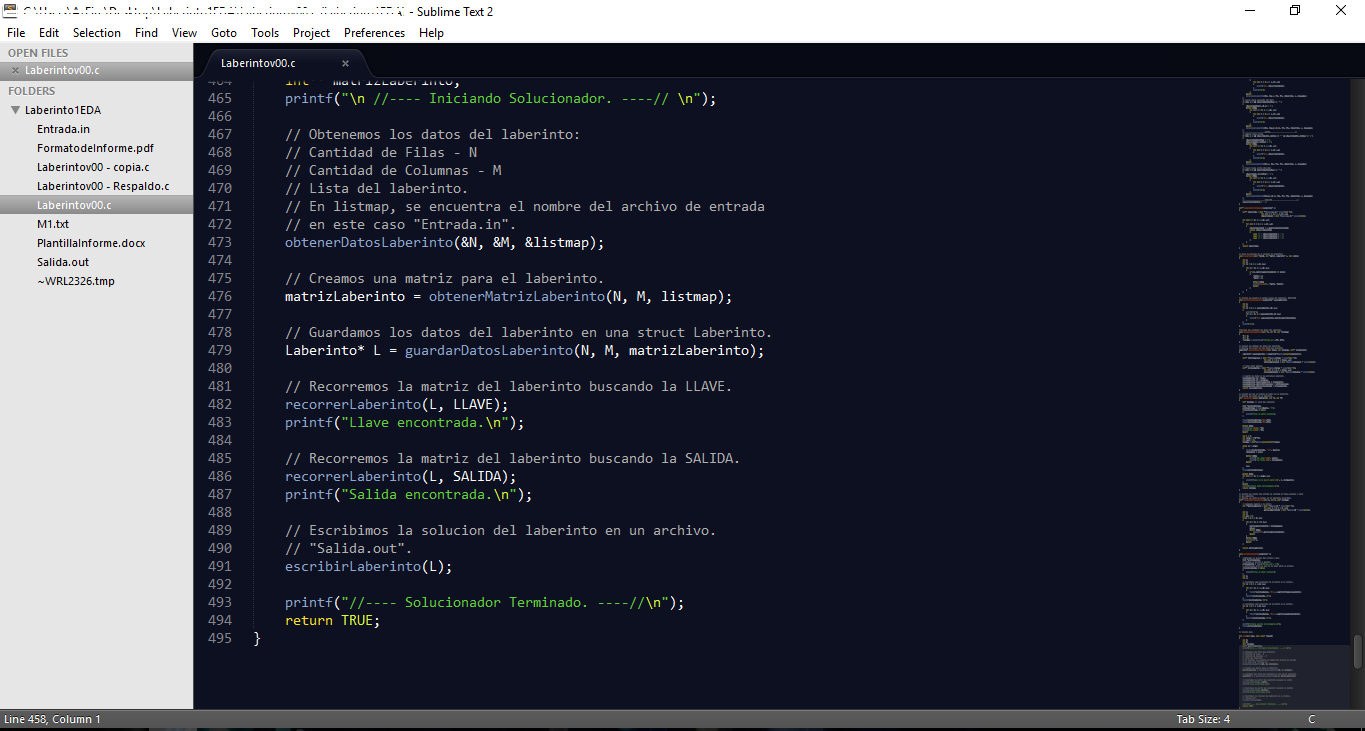


Ilustración 15 ScreenShot del programa SublimeText2.

#### **GITHUB**

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git. En esta ocasión se creó un repositorio de forma privada y se almacenó en él las distintas versiones del programa.

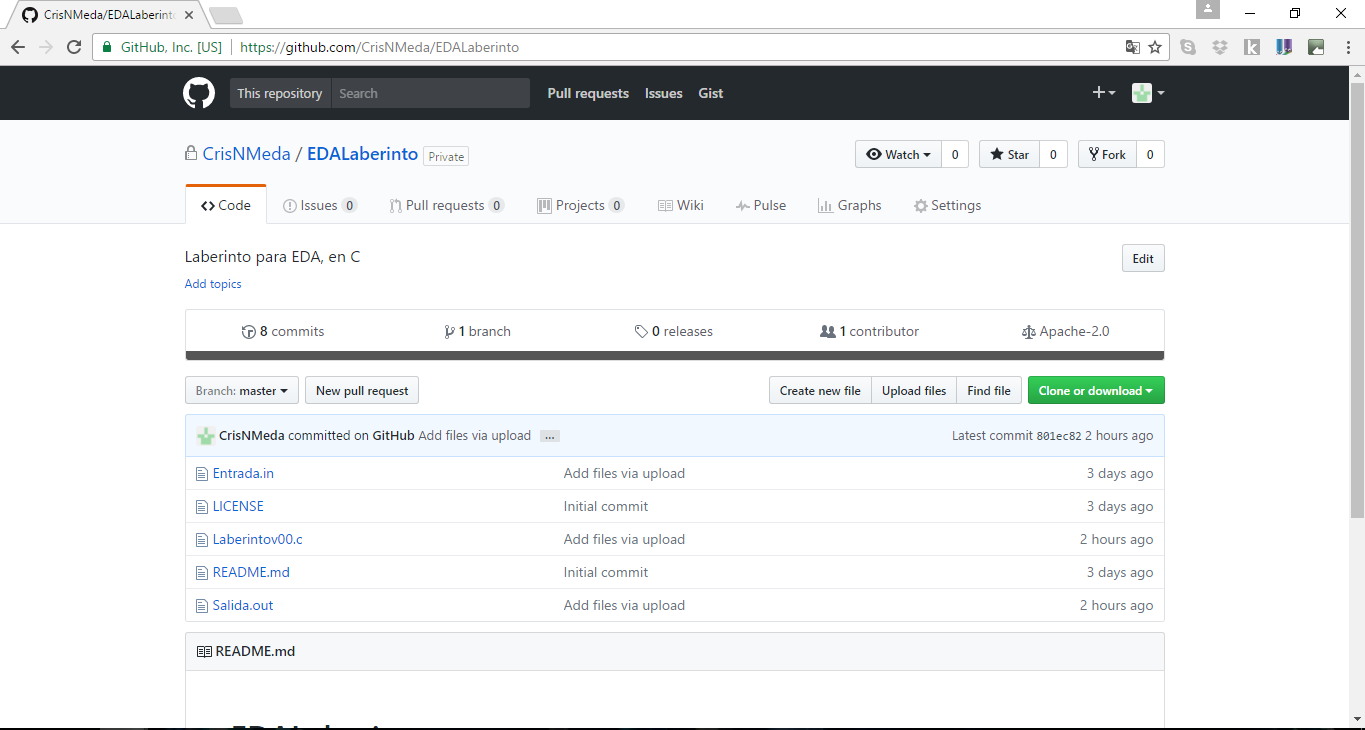


Ilustración 16 ScreenShot del repositorio en GitHub.com

#### **MINGW**

Para compilar se usó minGW (Minimalist GNU for Windows), anteriormente conocido como MinGW32, es una implementación de los compiladores GCC para la plataforma Win32, que permite migrar la capacidad de este compilador en entornos Windows.

#### **DOXIGEN**

Para comentar se usó Doxygen, Doxygen es un generador de documentación para C++, C, Java, Objective-C, Python, IDL (versiones Corba y Microsoft), VHDL y en cierta medida para PHP, C# y D.

# ANALISIS DE LOS RESULTADOS



## TIEMPOS DE EJECUCIÓN

Según los tiempos calculados, el programa tiene bajos tiempos de ejecución a excepción del algoritmo que recorre el laberinto (Backtracking), ya que su tiempo de ejecución es el más alto del programa y tiene un orden de O(2^n).

### OBJETIVOS

Los algoritmos señalados en el capítulo anterior, cumplen con los objetivos del programa en su totalidad.

## falencias detectadas.

En este apartado se detallan algunos errores que el programa posee.

### ERROR AL ABRIR ARCHIVO

Si el archivo no se encuentra disponible para abrir, el programa no se ejecuta, obligando a que sea abierto de nuevo y que exista un archivo de entrada.

### WARNING AL COMPILAR EN LINUX

Al compilar en Linux existe una advertencia sobre la declaración de un char\* que es pasado como int\*, este error solo ocurre al compilar en Linux, en Windows no tenemos Warning.

## Propuestas de mejora

### VERIFICACIÓN DE CARACTERES

Se necesita agregar una verificación en los caracteres del laberinto, con el fin de asegurar su correcta solución y no traer consigo, resultados inválidos.

### INGRESO DEL NOMBRE DE ARCHIVO

Agregando un ingreso del nombre de archivo, se genera un programa más amigable con el usuario.

# CONCLUSIÓN

La implementación de los algoritmos fue exitosa, salvo algunas excepciones que ya fueron detallas, se pudo completar todos los objetivos señalados. Obteniendo así una solución final al problema del enunciado. Se hace uso de listas enlazadas y se encuentra el camino mínimo del laberinto de una forma correcta. Los tiempos de ejecución, no son tan altos, lo cual lo hace un algoritmo bastante eficiente, generado consigo una pronta solución.

Algunas dificultades fueron los ciclos infinitos que generaba la función que se asimila al backtracking, errores de lectura, errores de escritura, errores de asignación de memoria y el uso de listas enlazadas.

Se sugiere corregir los errores principales del programa, con el fin de dar al usuario una mayor satisfacción al momento de éste querer resolver sus laberintos.

# REFERENCIAS

**Referenciar páginas web:**

GITHUB (01-04-2017). *GitHub* [www.github.com](http://www.github.com)

# MANUAL DE USUARIO



## Introducción

Este manual busca instruir al usuario en el uso del software desarrollado para dar solución al problema de resolver laberintos. En este manual, se le entrega al usuario los conocimientos necesarios para compilar, ejecutar y utilizar el software.

## Como compilar y ejecutar.

### Compilar en Windows

Para compilar en Windows se requiere tener instalado y configurado MinGW para C.

El primer paso para compilar es ir a la carpeta donde se encuentra el código fuente. Luego en un espacio en blanco, presionar shift + click derecho y darle a la opción de *abrir ventana de* *comandos* o *PowerShell* (en Windows 10) como lo muestra la siguiente figura:

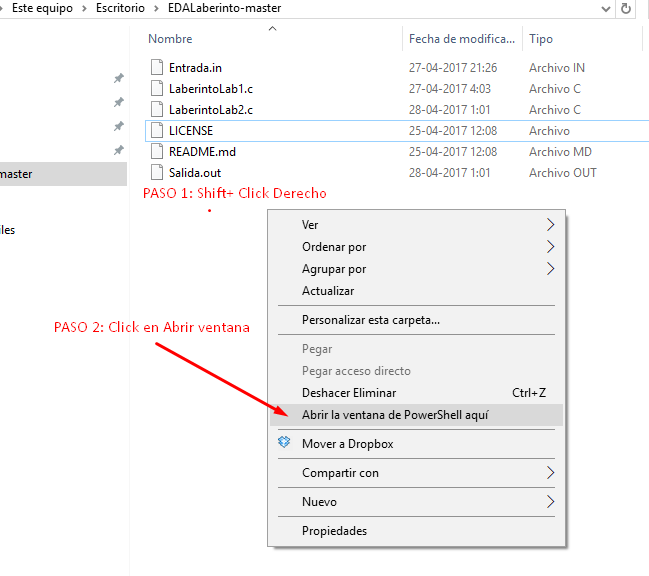


Ilustración 17 : Compilando en Windows parte 1.

Para finalizar escribimos la siguiente línea, donde LaberintoLab2.c (Es el nombre del código fuente a compilar, si es diferente, se debe escribir aquel nombre en ese campo). Y Luego presionar enter.

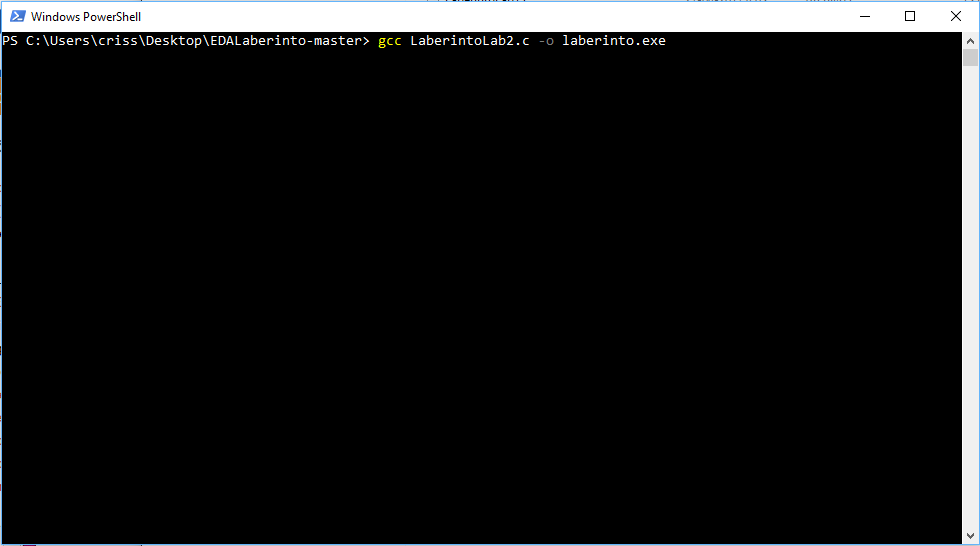


Ilustración 18 Compilando en Windows parte 2

.

Para ejecutar tenemos dos opciones, si usamos cmd o si usamos PowerShell:

Si usamos Windows con cmd:

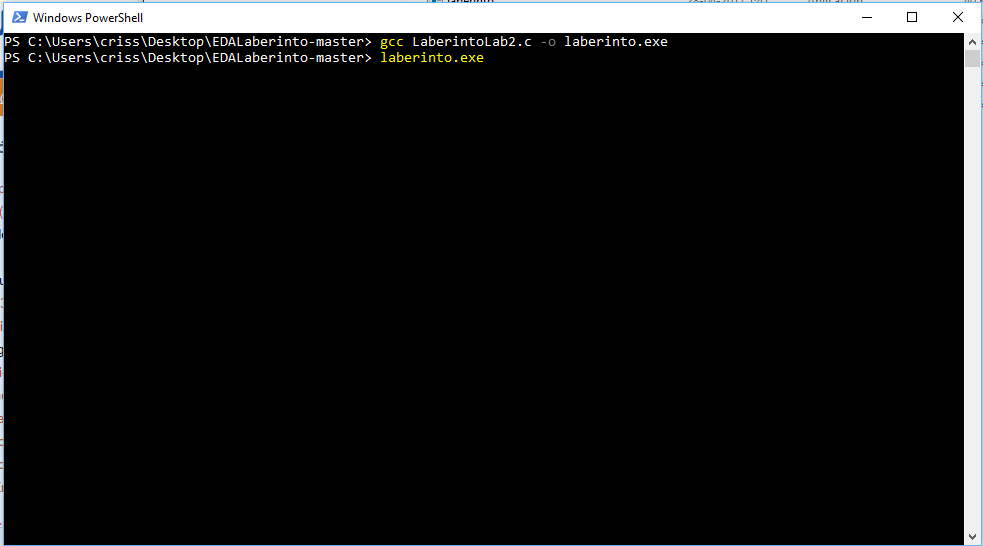


Ilustración 19 Compilando en Windows parte 3

Si usamos Windows 10 con PowerShell.

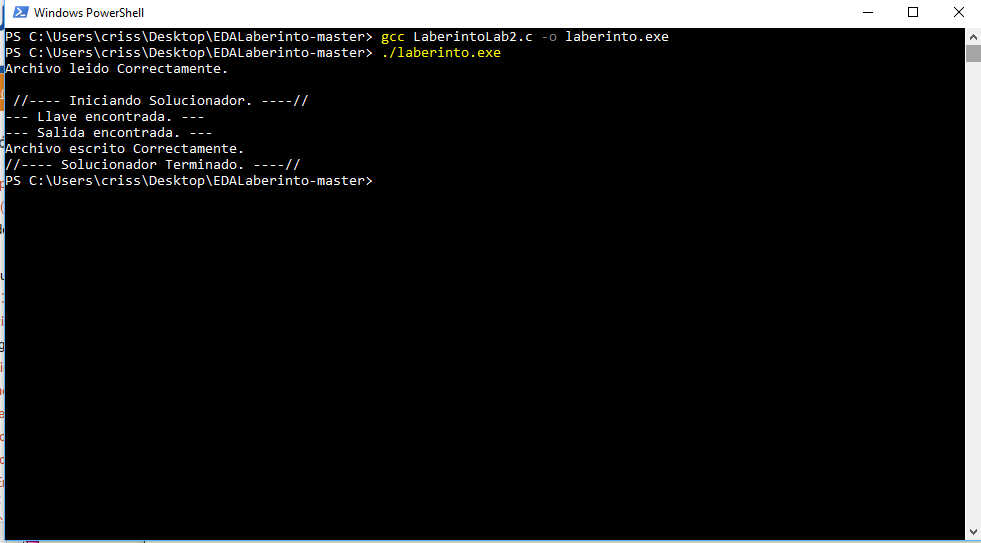


Ilustración 20 Compilando en Windows parte 4

### Compilar en Linux

Para compilar en Linux se debe tener instalado gcc. La forma de compilar es similar a la compilación en Windows usando PowerShell.

Primero nos ubicamos en la carpeta del código, luego abrimos una terminal y escribimos la línea de compilación.

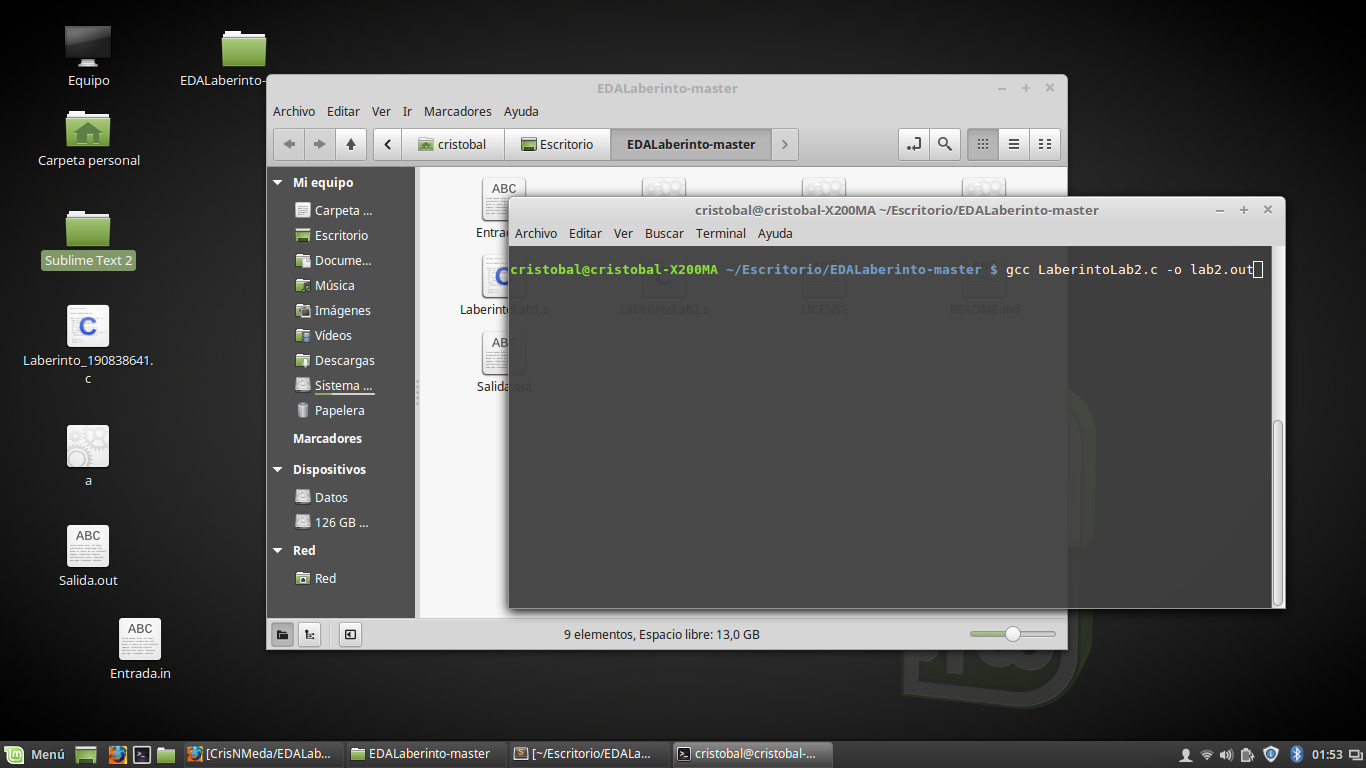


Ilustración 21 Compilando en Linux parte 1.

Ejecutando:

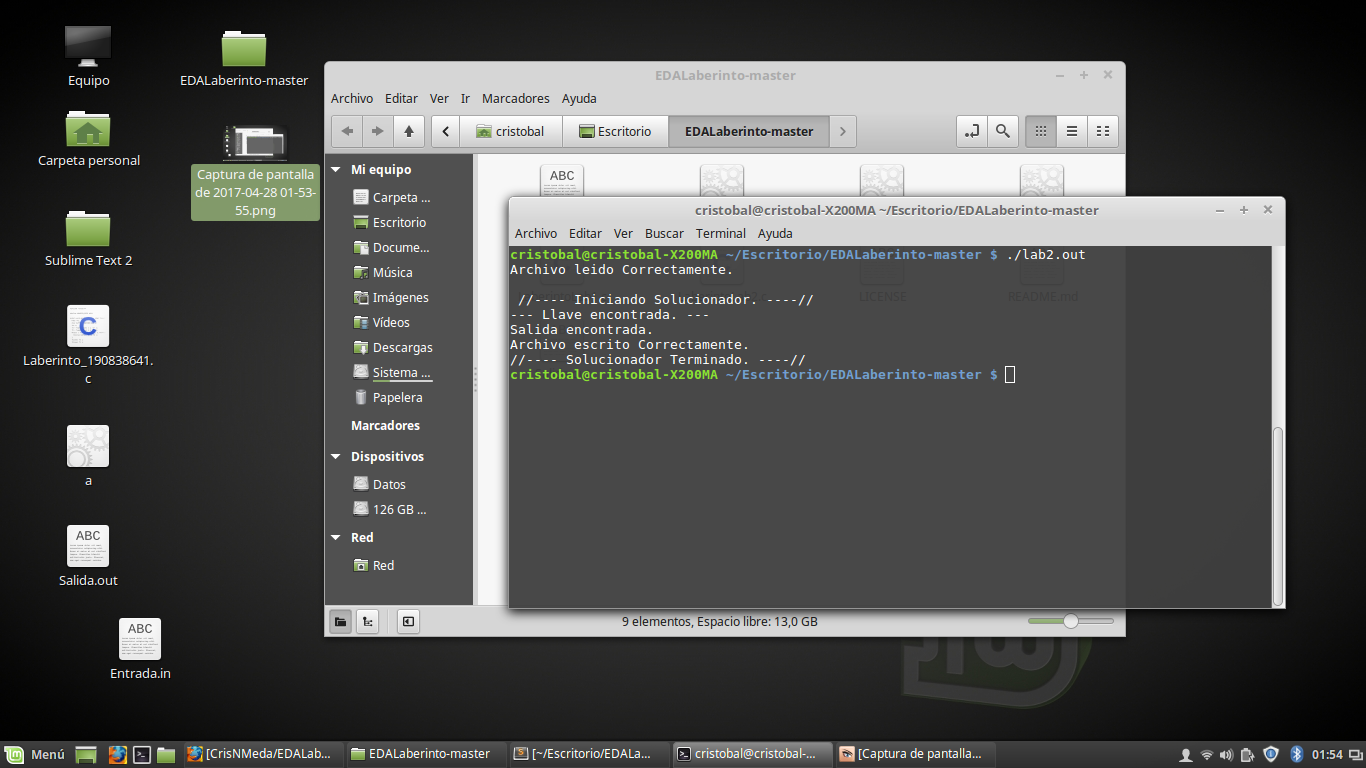


Ilustración 22 Compilando en Linux parte 2.

Funcionalidades del programa.

El programa recibe cualquier tipo de archivo llamado (archivo.in) que contenga un laberinto, como el señalado en el capitulo 2, en la lectura del archivo.

1. Entiéndase algoritmo como un conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas. [↑](#footnote-ref-1)