

Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Programación Bajo Plataformas Abiertas

IE0117

**Proyecto: resolvidor de laberintos en un entorno
individual**

Estudiante: Angie J ulissa Mèndez Ramìrez

B74721

Profesor: J uan Carlos Coto

II Semestre - 2022

Tabla de contenido

1. Primera Parte: Introducción	1
2. Segunda Parte: Diseño general	1
A. Creación de un ejemplo de "laberinto.txt":	1
B. Estructura para reconocer el documento.txt:	2
C. Lógica del J uego:	4
D. Muestra de soluciones y datos del laberinto:.....	4
E. Uso de Git para crear repositorio:.....	5
3. TERCER PARTE: Limitaciones presentadas.	6
4. CUARTA PARTE: Conclusiones.	7
5. QUINTA PARTE: FUENTES:	7

1. Primera Parte: Introducción

El objetivo del presente consiste en sintetizar de manera clara y breve el trabajo realizado en la creación de un resolutor de laberintos en formato matricial a través del reconocimiento de un archivo tipo “.txt” en el lenguaje de programación C. Se busca exponer las principales partes del juego, así como explicar qué conocimientos fueron aplicados en la creación del mismo, ya sea por lo visto en clase o por investigación del estudiante. La idea es que el/la estudiante pueda construir un algoritmo que lea cualquier tipo de laberinto y éste halle desde el reconocimiento de sus elementos, una única solución mediante uno o varios caminos o bien, no encuentre ninguna.

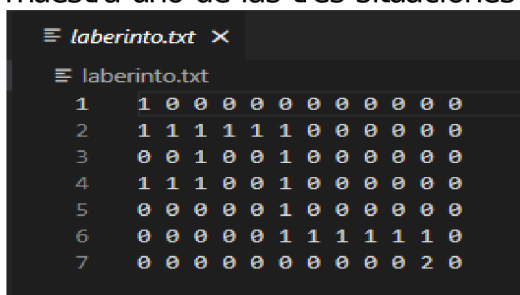
2. Segunda Parte: Diseño general

A. Creación de un ejemplo de “laberinto.txt”:

Para comprobar el funcionamiento del algoritmo por elaborar, es necesario arrancar por formar un laberinto de tamaño $m \times n$ que nos permita corroborar que:

- Mi algoritmo es capaz de leer un archivo de esta extensión.
- Mi algoritmo puede recorrer los elementos, filas y columnas de la matriz contenida en ese archivo.
- Mi algoritmo identifica a través de cierto tipo de carácter, si la estructura de mi laberinto tiene solución única o no tener ninguna.

Con esto en mente, consideramos el laberinto de la Figura 1, donde se muestra un arreglo matricial de 7 filas y 12 columnas que será evaluado posteriormente y muestra uno de las tres situaciones que puede resolver el programa:



	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	0	0
3	0	0	1	0	0	1	0	0	0	0	0	0
4	1	1	1	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0
6	0	0	0	0	0	1	1	1	1	1	1	0
7	0	0	0	0	0	0	0	0	0	0	2	0

Figura 1: Ejemplo propio de un laberinto de 84 caracteres.

B. Estructura para reconocer el documento.txt:

En el archivo llamado “recorrermatriz.h” se construye una lógica basada en una variable “c” que se ocupará de distinguir entre distintos tipos de carácter, no por grupos si no por tipo, es decirle posteriormente al programa que si es 0 es pared, si es 1 es un camino/salida y si es 2 es una solución. Luego se procede a verificar que se puede abrir el documento “.txt” a través de un puntero FILE que me guarde dicha acción en una variable condicional “fp” mediante la función “abrirdoctxt” . La encargada de ejecutar esta acción es la función “fopen”, la cual recibirá a “name” y pasará por parámetro a dicha función y abrirá el archivo en modo lectura. Se incluye el caso de que no se pueda abrir el archivo por estar vacío, con el valor de NULL. Después de esto, es necesario estructurar cómo se reconocerán los elementos de la matriz del laberinto basado en el orden de sus filas y columnas. El entero “traerFilastxt” almacena a través de “name” esa lógica donde se inicia en el valor de 1 porque al contar las filas naturalmente, el lenguaje C comienza en 0, sin embargo, para que el total de filas no dé un valor negativo, se inicializa en 1 para obtener un entero positivo, mientras que en las columnas se inicializa en 0 porque, porque si va a extraer la cantidad de columnas original del lenguaje C. Se llama a la función de abrir el archivo y tanto para filas como columnas se aplica un condicional “for” donde se guardará el valor de “c” previamente definido como una función extraída tipo “c =getc(f)”. La palabra “EOF” se aplica cuando ya el algoritmo recorre toda la matriz y necesita reiniciarse para volver a aplicar la lógica, ya que ha llegado al final del documento. El siguiente “for” posee la estructura ilustrada en la Figura 2.

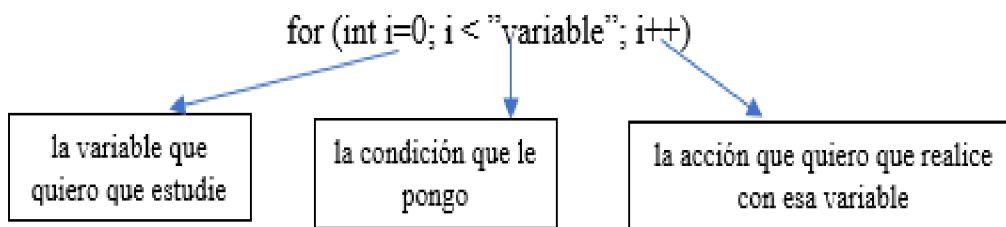


Figura 2: Estructura del condicional usado para la variable de recorrido.

Para las filas se definen una condición:

- Si el carácter que lee es un salto de línea, entonces al contador “row” va aumentar conforme se agreguen saltos de línea, es decir, se le indica que empiece en 1 y a partir de ese 1 se le va sumando uno hasta llegar al final de la matriz para que retome.

Para las columnas se definen dos condiciones:

- Si el carácter es diferente a un espacio y diferente a un salto de línea, al contador de columna se le aumenta una y
- Cada vez que el carácter que se está leyendo sea un salto de línea, se retorna ese contador a cero (es decir, empieza de nuevo).

En otras palabras, esa segunda condición es importante porque con esa se obliga a la lógica a leer los elementos que hay en UNA FILA, y por lo tanto, contar la cantidad de columnas ya que es la misma cantidad de elementos en todas las filas y cuando cambia de fila empezar de nuevo a contar, para NO contar todos los elementos de una matriz.

Ahora, para imprimir la matriz se usa “matrizprint” donde se utilizan el comando “struct” para luego que al abrir el documento se recupere el tamaño del archivo y así se defina una estructura con “stat” que guarde esa información en “sb”. Posteriormente con el puntero “file_contents” se busca construir una memoria con un número determinado de bytes a través de “malloc” el cual especifica ese valor. Basado en esto, la función “fopen” nuevamente abre el documento y designa una memoria del tamaño de archivo completo para guardar el flujo de lectura en él.

Además, la cadena de formato “%[\n]” se crea con el propósito de leer el flujo de archivos hasta encontrar el elemento de nueva línea. Por otro lado, “fscanf”, se encarga de regresar EOF cuando se alcanza el final de la matriz; por lo que, iteramos con el condicional “while” e imprimimos cada línea una por una. Hay que recordar que se ejecuta siempre y cuando dicha condición entre paréntesis evalúa como “True”.

Por último, dentro de este archivo, se tiene el lector de coordenada de solución “coordSol” que hace un recorrido de toda la matriz, filas y columnas. Esto para que

cuando encuentre la única solución, me lea la coordenada. Se aplica la misma lógica de lectura en “traerFilastxt” y “traerColumnastxt”:

- Si es diferente de espacio y diferente de un salto de línea, agregue al contador de columnas, una columna más.
- Si es igual a un salto de línea, agregue al contador de filas una columna de más.

Y se le agrega:

- Cuando encuentre el 2, se indica que se imprima el lugar dentro de la matriz donde está, en forma de (x, y) ó sea, (fila, columna).

C. Lógica del J uego:

Se trae todas las funciones usadas en “recorrermatriz.h” para ahora usar una nueva variable de recorrido “cc” y navegar mediante la función “travel” por la matriz en busca de soluciones. Se utiliza lógica booleana donde se inicia en False y si:

- El carácter encontrado no es un espacio, un salto de línea o un cero: con esto se busca encontrar “unos” para hallar un camino determinado y para encontrar “dos” y así la solución. Si se encuentra ese “2” se imprime y se cambia a un True.

Posteriormente se define algunos parámetros para formar visualmente la solución del laberinto imprimiendo los saltos de línea, cambiando los ceros por espacios e imprimiendo los espacios en sí, para ver explícitamente el camino y la solución.

Finalmente, está el contador de dos, quien busca las soluciones mediante la asignación a “cc” como un 2. Donde:

- Si es un dos, se agrega al contador un “2” más. Se toma en cuenta que, se busca que ese contador siempre dé uno, ya solo queremos tener un camino válido con una única solución. Me retorna el total de “dos”.

D. Muestra de soluciones y datos del laberinto:

Esta función principal llama al archivo “laberintoproyecto.h” (que a su vez llamaba a “recorrermatriz.h”) para acomodar las soluciones encontradas por “travel” después

de guardar en un “char” la lectura del documento “.txt” donde se imprime el número de filas y columnas encontradas anteriormente con los punteros del recorrido del laberinto elemento por elemento, se imprime la matriz en sí encontrada por “matrizprint” conteniendo al argumento “doc” y a partir de esto se definen tres situaciones ilustradas por la Figura 3:

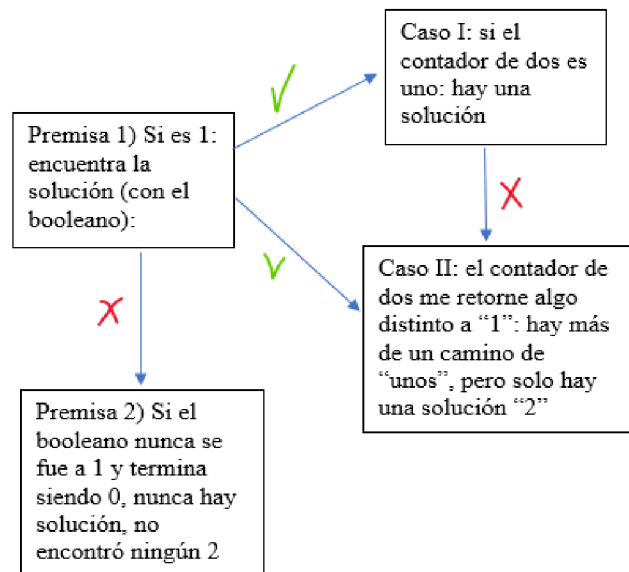


Figura 3: Situaciones de solución para el laberinto.

Para el caso específico del Caso II, en el “.txt” habrá más de un 2, pero solo hay un camino valido de “1” que van a un “2” los demás “2” están encerrados entre pared, aunque haya más caminos de “1”. La Figura 1 cumple con la Premisa 1.

E. Uso de Git para crear repositorio:

Para exponer el trabajo realizado se ha creado un repositorio de GitHub donde, se ha definido como rama “default” a “master” y posee el nombre PROYECTO-LABERINTO-B74721 y se han aplicado los siguientes de la Figura 4 comandos para añadir al mismo los archivos ejecutables de este proyecto. Se añade un “README” con instrucciones para compilar el archivo main.c desde la terminal con todos los archivos en la misma carpeta.

```
git clone git@github.com:anjuly01/PROYECTO-LABERINTO-B74721.git  
*agregando los archivos correspondientes a la carpeta*  
git add nombre_archivo.extensión  
donde "nombre_archivo.extensión" se intercambia por cada uno de los  
archivos ya sea .c, .h, .txt o .pdf  
git commit -m "añadir nombre_archivo.extensión "  
git push
```

Figura 4: Comandos para creación del repositorio GitHub.

3. TERCER PARTE: Limitaciones presentadas.

- a) Al revisar mi código hace unos días, noté que cada vez que encuentra un "2" hace el "print" de la coordenada en la matriz independientemente si es solución o no (es decir si hay unos o ceros a la par de ese "2"): el código sabe que solo hay una solución aunque hayan varias caminos de "1", pero aun así me imprime todos los "2" en coordenadas, ya que no distingue qué hay a la par de cada "2" para saber cuál corresponde a la coordenada correcta. Intenté hacer otro puntero con una lógica más complicada pero no logré que el código puede discriminar solo un dos. Igual es un detalle de impresión ya que sí reconoce una única solución.
- b) Me ha costado un poco cambiar la rama default del GitHub ya que no recordaba de las clases que se tenía que hacer desde la terminal con el comando "git checkout NOMBRE-RAMA". Posteriormente lo repasé e investigué en internet y lo logré.
- c) Tuve problemas al principio para aprender una lógica que me permitiera imprimir la matriz en el archivo "recorrermatriz.h" con estructuras y memorias, ya que no sabía que tenía que aplicar un "while" para tomar en cuenta el final del documento y reiniciar. Sin embargo, estudié las clases del curso y busqué algo de información en internet y lo conseguí hacer.

- d) Al principio, para obtener el link de mi repositorio tuve que crear una SSH ya que no me aparecía sin hacer esta configuración. Con las instrucciones de la página oficial de GitHub resolví el problema.
- e) Se me dificultó un poco descubrir cómo hacer la lógica para abrir un archivo .txt e incluir el caso "NULL" cuando se encuentra un archivo vacío ya que no había considerado esa opción. Posteriormente lo corregí.

4. CUARTA PARTE: Conclusiones.

A través de este proyecto, se ha descubierto de nivel experimental, la importancia del uso de punteros, estructuras, memorias y demás comandos a la hora de identificar los elementos de un archivo. Además, se ha aprendido técnicas para aplicar los condicionales de forma conveniente en nuestra lógica y organizar un código en varios archivos ejecutables para obtener mejor orden en el programa.

Mi única pregunta sería, basada en mi primer limitación, ¿cómo le indico a la función con otra variable qué hay a la par del carácter que estoy estudiando para definir dentro de la lógica que debe tomar ese carácter de estudio?

5. QUINTA PARTE: FUENTES:

- 1) DelfStack, 2021. *Leer archivo línea por línea usando fscanf en C*. Recuperado de: <https://www.delftstack.com/es/howto/c/fscanf-line-by-line-in-c/>
- 2) Montiel, O., 2021. *Explicación de Git branch: Cómo eliminar, mover, crear y renombrar una rama en Git*. freeCodeCamp .Recuperado de: <https://www.freecodecamp.org/espanol/news/explicacion-de-la-rama-de-gi-como-eliminar/#eliminar-una-rama>
- 3) GitHubdocs, 2022. *Conectar a GitHub con SSH*. Recuperado de: <https://docs.github.com/es/authentication/connecting-to-github-with-ssh>