Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Proyecto Final: Resolvedor de laberintos

Programación bajo Plataformas Abiertas

Alonso Jiménez Villegas

B94125

Profesor: Juan Carlos Coto Ulate

I Ciclo

17 de Julio de 2022

Contents

Introducción

En el presente documento se presenta el informe técnico del proyecto final para el curso de Programación bajo Plataformas Abiertas, en el cual se solicita que el estudiante implemente lo aprendido a lo largo del curso. El proyecto consiste en crear un resolvedor de laberinto, en donde se debe evaluar una matriz de tamaño “m * n” para llegar a su solución.

A largo plazo, el propósito del proyecto es que el estudiante aprenda a implementar diferentes funciones para la resolución de problemas de programación. El estudiante debe estar consiente de los errores que el programa presente, para que este pueda discernir acerca de la mejor manera de su corrección. Asimismo, se espera que el estudiante se familiarice con plataformas como GitHub, para que así pueda compartir sus códigos, así como consultar en el momento de llegada alguna dificultad.

Diseño General

Se comenzó el código incluyendo las librerías de “stdio.h”, “stlib.h”, “string.h” y “stdbool.h” para incluir y utilizar diferentes funciones en el código. Luego de incluir las librerías, se creó una función llamada “mostrar” a través de un void. Lo que genera esta primera función es ir mostrando la matriz del laberinto que se quiere resolver. Se definen primero “m” y “n” que indican el tamaño de la matriz. También se definen “i” y “j” como filas y columnas.

Luego de esto, se crea una función llamada “leer”, esta función es la encargada de que el programa logre visualizar el archivo en el que se encuentra el laberinto. Esto es posible gracias a la función “fopen”, la cual llama al archivo .txt, y este se almacena en una variable llamada “texto”. Seguidamente, a través de un ciclo “while” se evalúa el archivo .txt de principio a fin, hasta que se encuentra el final del archivo. Luego de esto se utiliza un ciclo “for” para ir buscando en los bordes del laberinto en donde se encuentra un número “1” para poder empezar la resolución del laberinto. Para esto se hizo uso de un “if” para evaluar en donde se encuentra el primer “1” y en cual de las posiciones (arriba, abajo o a los lados) se encuentra el siguiente “1”, indicando el camino. Luego se genera una función “crear” para definir el tamaño de la matriz en donde se guarda el laberinto.

```
51      for (int i = 0; i < *largo; i++){
52          for (int j = 0; j < *ancho; j++){
53              if (lineas[i][j] == '1' && (j == *ancho - 1 || i == *largo - 1 || j == 0 || i == 0)){
54                  *posicionX = j;
55                  *posicionY = i;
56              }
57              break;
58          }
59          printf("%i", *posicionX);
60          printf("%i", *posicionY);
61      }
62
63  }
```

Acto seguido, se hace la función encargada de buscar el mejor camino para la resolución del laberinto brindado (llamada “resolverlaberinto”). Primero, a través de ciclos “for” y condicionales “if” se evalúa que al leer la matriz la posición no se pase del máximo valor del ancho o del largo de la matriz (guardados en punteros

llamados “*ancho” y “*largo”). Si pasa el caso de que la posición es mayor que alguno de estos dos punteros, se tiene dos “breaks” para detener el programa.

```
80      for (int i; i > *largo; i++){
81          for (int j; j > *ancho; j++){
82              if (j > *ancho){
83                  break;
84              };
85          }
86          if (i > *largo){
87              break;
88          };
89      }
```

Luego de esto se tiene un condicional “if” que define el requisito de la solución del código. Esto se da a través de lógicas “OR”, en donde se tiene que si en alguna de las siguientes posiciones (arriba, a los lados o abajo) se encuentra el número “2”, se logró llegar con éxito al final del laberinto.

```
if (matriz[i][j] == '2' || matriz[i + 1][j] == '2' || matriz[i - 1][j] == '2' || matriz[i][j + 1] == '2' || matriz[i][j - 1] == '2'){
    printf("Se encontró el camino con la solución.\n");
}
```

Si no se ha encontrado el número “2”, se sigue con el código para ir recorriendo el laberinto. Cabe destacar que se utilizó el método recursivo (función recursiva), es por lo que en cada una de las condiciones se vuelve a llamar a la función “resolverlaberinto” y se va guardando la posición a la que se movió (por ejemplo, si se movió a la derecha, el nuevo índice “i” tendrá como forma “i + 1”). La forma de resolución del laberinto va cambiando el camino de “1” por puntos para indicar que el programa ya recorrió ese camino. Similarmente, si el programa pasa por uno de los puntos anteriores, estos se reemplazan por un “:”. Si en lo que el programa recorrió el laberinto, no se encuentra el requerimiento de finalización (el número “2”), el programa le indica al usuario que no se encontró una solución posible para el laberinto.

Se tiene la función “main”, en la cual se van a llamar a todas las funciones anteriormente explicadas. Se definen “argc” y “argv”, los cuales “argc” indica al usuario que no se le entregó al programa ningún tipo de argumento, lo que quiere

decir que el usuario no brindó ningún tipo de archivo de texto en el cual se pueda encontrar el laberinto a resolver; mientras que “argv” trata de un arreglo en donde la posición “0” trata sobre el archivo “./a.out” que se genera, y la posición “1” trata de la dirección en la cual se encuentra el archivo de texto del laberinto. En el último “else” se definen varios “int” que irán dentro de cada una de las funciones.

Por último, para poder correr el código se debe ir a la terminal en donde se encuentra el código del programa. Para que el código compile, en la terminal se debe escribir “gcc ProyectoResolverLaberinto.c”. Luego de que compile, para correr el programa se debe escribir el “./a.out” y la dirección en donde se encuentra el archivo del laberinto (que, para efectos específicos, se pone ./a.out “/home/alonso/Documents/Proyecto/laberinto.txt”).

Principales Retos

1. El primer reto que se presentó fue un error de “expected identifier or ‘(‘ before for”. Este error se presentó debido a que se tenía un ciclo “for” por fuera de la función llamada “leer”, cuando se estaba definiendo los bordes del laberinto.
2. El segundo reto encontrado fue acerca de un error de “segmentation fault” en donde la posición en la que se estaba evaluando era mayor que el límite de ancho de la matriz, por lo que no se imprimía bien el laberinto y no se lograba encontrar la solución.
3. Otro reto encontrado fue aprender acerca de la función recursiva y cómo poder implementarla al llamarla en cada condición.
4. Otro reto que se presentó fue el aprender a llamar el archivo de texto para que el programa pueda leer el laberinto que se quiere resolver.
5. Por último, el quinto reto al que se enfrentó fue acerca de crear un repositorio en GitHub y poder subir los archivos del proyecto a la rama “master”.

Conclusiones

Gracias a efectuar el proyecto es que se aprendió a efectuar mejor un código en el lenguaje de “C”. Con las diferentes dificultades y retos encontrados se logró expandir más el conocimiento acerca de funciones, punteros, arreglos, entre otras cosas. Para superar las dificultades encontradas se hizo uso de “clangd” y “gdb”, que ayudaron a encontrar la localización del error en el código, así como diferentes sugerencias rápidas para corregir el código. También se aprendió a subir archivos a GitHub en las diferentes ramas (como la rama master).