Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

IE0117 | Programación bajo plataformas abiertas

I Ciclo 2022

Proyecto | Resolvedor de laberintos.

Grupo: 001

Preparado por:

Katherine Chavarría Nájera / B41841

Profesor:

Juan Carlos Coto

Índice

Introducción

En el presente proyecto se desarrolla el algoritmo de un resolvedor de laberintos en lenguaje de programación C; donde, se le introduce un archivo de texto llamado laberinto.txt que contiene el laberinto definido como una matriz de números enteros de tamaño m * n, donde m son las filas y n son las columnas. La matriz solamente puede tener como números enteros el 0, 1 y 2; donde, el 0 indica una pared, el 1 indica un camino y el 2 indica la solución.

El algoritmo está diseñado para que inicialmente registre cada uno de los bordes de la matriz en busca de puntos donde se encuentre un 1 y una vez encontrado el 1 busca si a su alrededor se encuentran más números 1 para continuar moviéndose por ese camino y a como se va desplazando la coordenada por cada 1 que pase lo reemplaza por el número 9 hasta que encuentre la solución. Si se llega a un “callejón sin salida” el algoritmo volverá a empezar en busca de otra entrada que lo guíe a la solución, pero siempre con la memoria del recorrido anterior sustituyendo por el número 9 cada 1 que recorra. Al final, una vez encontrada la solución, se imprime en pantalla el laberinto en juego con el recorrido trazado y la coordenada donde se encuentra su solución.

El código se encuentra dividido en cuatro archivos .c de código fuente y tres archivos .h de bibliotecas, adicionando el archivo .txt que debe ser introducido. El propósito de este documento es explicar la relación de estos archivos y en qué consiste cada uno para que el lector comprenda cada una de sus partes, más allá de saber cómo compilar y ejecutar el juego; como fue explicado en el repositorio.

Diseño general

El algoritmo se encuentra dividido en cuatro grandes partes dependientes entre sí y cada una de estas partes están conformadas por una función. A continuación, se explica con detalle cada parte:

1. Archivo principal.c: En el archivo llamado principal.c se encuentra la importación de las bibliotecas necesarias para el correcto funcionamiento del código, luego se encuentra la importación de los archivos dependientes y finalmente se encuentra la función principal llamada “main”. En esta función, en primer lugar, se declara la variable llamada “archivo” la cual se ha definido que sea de “tipo archivo”; a esta variable seguidamente se le asigna la función “fopen” la cual permite abrir el archivo de texto llamado laberinto.txt y por medio del comando “r” leerlo. Luego, se imprime una pequeña portada y finalmente se llama a la función llamada “lectura” a la cual se introduce el parámetro archivo. En la figura 1 se muestra dicha función “main” y como se puede observar se exponen todas las partes mencionadas anteriormente.

```
19
20  int main()
21  {
22      FILE *archivo;
23      archivo = fopen("./laberinto.txt", "r");
24      printf("\n");
25      printf("Estudiante: Katherine Chavarría Nájera.\n");
26      printf("Carné: B41841.\n");
27      printf("Proyecto: Resolvedor de laberintos.\n");
28      printf("Curso: IE0117.\n");
29      printf("\n");
30      printf("Solución:\n");
31      lectura(archivo);
32
33      return 0;
34  }
```

Figura 1. Función principal.

2. Archivo lectura.c: En el archivo lectura.c se encuentra la función llamada “lectura” que recibe un parámetro de “tipo archivo”, como se muestra en la figura 2. Seguidamente, se declara un arreglo de dos dimensiones de tipo “int” llamado “laberinto” en el cual se va a guardar el laberinto a recorrer. Luego, se llama a la función “reconocer_laberinto”, que será explicada más adelante, para poder obtener el número de filas y columnas que posee el laberinto insertado; para así, seguidamente introducir dichos parámetros en la función llamada “entrada” y por medio de esta función poder obtener las coordenadas de entrada del laberinto. Luego, estas coordenadas de entrada son utilizadas en un bucle “while” que por medio de un conjunto de “if” escanea lo que se encuentra alrededor de la coordenada y si lo que encuentra es un 1 o un 2 la coordenada se desplaza hacia ese punto y el 1 del punto anterior lo reemplaza por un 9, hasta que encuentre la solución. Si se llega a un “callejón sin salida” el algoritmo volverá a empezar en busca de otra entrada que lo guíe a la solución, pero siempre con la memoria del recorrido anterior sustituyendo por el número 9 cada 1 que recorra; esto se muestra en las figuras 3 y 4. Finalmente, en esta función se imprime el laberinto recorrido y la coordenada de salida a la cual se llegó.

```
15
16  void lectura(FILE *archivo)
17  {
18      int laberinto[100][100];
19
20      // Se llama a la función "reconocer_laberinto", la cuál reconoce y guarda el laberinto en un
21      // arreglo de dos dimensiones llamado "laberinto[][]".
22      int salidas_rl[2];
23      reconocer_laberinto(laberinto, salidas_rl, archivo);
24      int filas_main = salidas_rl[0];
25      int columnas_main = salidas_rl[1];
26
27
28
29      // Se llama a la función "entrada", la cual registra la entrada.
30      int salidas[2];
31      entrada(laberinto, salidas, filas_main, columnas_main);
32
```

Figura 2: Función “lectura”, parte 1.

```
// Algoritmo que encuentra la salida del laberinto por medio de un camino que inicialmente viene
// de una entrada encontrada.
int i, j;
i = salidas[0];
j = salidas[1];

while (1)
{
    //===== Condicionantes cuando el algoritmo encuentra un 2 =====//
    // Conjunto de "if" que escanea lo que se encuantra alrededor de la coordenada.

    if (laberinto[i-1][j] == 2)
    {
        laberinto[i][j] = 9;
        i = i -= 1;
        break;
    }

    if (laberinto[i][j+1] == 2)
    {
        laberinto[i][j] = 9;
        j = j += 1;
        break;
    }

    if (laberinto[i+1][j] == 2)
    {
        laberinto[i][j] = 9;
        i = i += 1;
        break;
    }

    if (laberinto[i][j-1] == 2)
    {
        laberinto[i][j] = 9;
        j = j -= 1;
        break;
    }
```

Figura 3. Función “lectura”, parte 2.

```
        //===== Condicionantes cuando el algoritmo encuentra un 1 =====//
        // Escaneo.

        if (laberinto[i-1][j] == 1)
        {
            laberinto[i][j] = 9;
            i = i -= 1;
            continue;
        }

        if (laberinto[i][j+1] == 1)
        {
            laberinto[i][j] = 9;
            j = j += 1;
            continue;
        }

        if (laberinto[i+1][j] == 1)
        {
            laberinto[i][j] = 9;
            i = i += 1;
            continue;
        }

        if (laberinto[i][j-1] == 1)
        {
            laberinto[i][j] = 9;
            j = j -= 1;
            continue;
        }

        else
        {
            entrada(laberinto, salidas, filas_main, columnas_main);
            i = salidas[0];
            j = salidas[1];
            continue;
        }


    }
```

Figura 4. Función “lectura”, parte 3.

3. Archivo reconocer.c: En el archivo reconocer.c se encuentra la función llamada “reconocer_laberinto” a la cual se le debe de introducir un arreglo de una y de dos dimensiones de tipo int y una variable de “tipo archivo”. Esta función posee un bucle “do…while” que guarda el contenido del archivo .txt importado en un arreglo de dos dimensiones de tipo “char” y convierte el arreglo de tipo “char” en uno de tipo “int”. Esto se realiza gracias al comando “fgetc” que permite leer elemento por elemento el contenido del archivo y guardarlos en el un arreglo de tipo char; sin embargo, cada vez que se guarda un elemento de tipo “char” en el arreglo ese elemento es convertido a un elemento de tipo “int” con el comando que se muestra en la línea 30 de la figura 5. Así mismo, en el bucle se cuentan cantidad de filas y el total de elementos que contiene la matriz insertada. Este bucle “do…while” dejará de reiniciarse hasta que se llegue a la totalidad del archivo. Finalmente, en la función “reconocer_laberinto” se guarda la cantidad de filas y columnas, que posee la matriz insertada, en el arreglo llamado “salidas”.

```c
// Función que importa el contenido del .txt, guarda el contenido en un arreglo de dos dimensiones
// de tipo "char" y convierte el arreglo de tipo "char" en uno de tipo "int".
void reconocer_laberinto(int laberinto[100][100], int salidas_rl[], FILE *archivo)
{
    int c;
    int i = 0;
    int j = 0;
    int cantidad_elementos = 1;
    int filas = 1;
    int columnas;

    // Bucle que guarda el contenido del .txt en un arreglo de dos dimensiones
    // de tipo "char" y convierte el arreglo de tipo "char" en uno de tipo "int".
    // De igual manera cuenta las filas y el total de elementos que contiene la matriz insertada.
    do
    {
        c = fgetc(archivo);
        if(c == '0' || c == '1' || c == '2')
        {
            laberinto[i][j] = c - '0';
            j = j += 1;
            cantidad_elementos = cantidad_elementos += 1;
        }
        if (c == '\n')
        {
            i = i += 1;
            j = 0;
            filas = filas += 1;
        }
    } while (c != EOF);

    fclose(archivo);
    columnas = cantidad_elementos/filas;

    salidas_rl[0] = filas;
    salidas_rl[1] = columnas;
}
```

Figura 5. Función “reconocer _laberinto”.

4. Archivo puerta.c: En el archivo puerta.c se encuentra la función “entrada”, a la cual se le debe de introducir como parámetros un arreglo de tipo “int” de dos dimensiones, un arreglo de una dimensión de tipo int y la cantidad de filas y columnas que posea la matriz ingresada. Esta función tiene como objetivo recorrer cada borde de la matriz ingresada, por medio de cuatro bucles “for”; donde, cada bucle “for” se encarga de recorrer un borde específico y buscar si en el borde se encuentra un 1, si lo hay entonces significa que en esa posición hay una entrada y guarda la coordenada de esa posición el arreglo llamado “salidas”. En las figuras 6 y 7 se muestra uno de los bucles “for” mencionados y como se puede observar este bucle recorre la primer fila de la matriz, buscando una entrada.

```
// Función que registra los bordes del laberinto y busca una entrada.
void entrada(int laberinto[100][100], int salidas[], int filas, int columnas)
{
    int cant_unos = 0;
    int cant_dos = 0;
    int cant_ceros = 0;
    int j,i;


    // Bucle que registra la primer fila de la matriz, buscado una entrada.
    for (j = 0; j < columnas; j++)
    {
        i = 0;
        if (laberinto[i][j] == 0)
                {
                    continue;
                }


        if (laberinto[i][j] == 1)
                {
                 cant_unos = 0;
                 cant_dos = 0;


                    // Conjunto de "if" que cuenta la cantidad de unos que se encuentran alrededor
                    // de la coordenada.

                    if (laberinto[i-1][j] == 1)
                    {
                        cant_unos = cant_unos += 1;
                    }

                    if (laberinto[i][j+1] == 1)
                    {
                        cant_unos = cant_unos += 1;
                    }

                    if (laberinto[i+1][j] == 1)
                    {
                        cant_unos = cant_unos += 1;
                    }

                    if (laberinto[i][j-1] == 1)
                    {
                        cant_unos = cant_unos += 1;
```

Figura 6. Función “entrada”, parte 1.

```
62
63                  // Conjunto de "if" que cuenta la cantidad de dos que se encuentran alrededor
64                  // de la coordenada.
65
66                  if (laberinto[i-1][j] == 2)
67                  {
68                      cant_dos = cant_dos += 1;
69                  }
70
71                  if (laberinto[i][j+1] == 2)
72                  {
73                      cant_dos = cant_dos += 1;
74                  }
75
76                  if (laberinto[i+1][j] == 2)
77                  {
78                      cant_dos = cant_dos += 1;
79                  }
80
81                  if (laberinto[i][j-1] == 2)
82                  {
83                      cant_dos = cant_dos += 1;
84                  }
85
86                  if (cant_unos == 1 && cant_dos == 0)
87                  {
88                      salidas[0] = i;
89                      salidas[1] = j;
90                      break;
91                  }
92
93              }
94      }
95
```

Figura 7. Función “entrada”, parte 2.

Finalmente, los archivos .h llamados:

- lectura.h
- reconocer.h
- puerta.h

Son archivos que permiten el ingreso de funciones de las cuales se tenga dependencia, importándolas como bibliotecas para poder acceder a ellas.

Principales retos

- El primer reto que se presentó fue, cómo hacer que el contenido del archivo .txt se guardara en un arreglo de cualquier tipo.

- El segundo reto fue, cómo convertir el arreglo tipo “char” que tenía el contenido del archivo .txt en un arreglo de tipo “int” con la misma información.
- El tercer reto fue, cómo recorrer la matriz, se probaron métodos con punteros y estructuras, pero al final se decidió recorrer la matriz con condicionantes “if” y “else”.
- El cuarto reto y probablemente en el que se tardó más tiempo ha sido, cómo hacer que el algoritmo exclusivamente siga un camino a partir de una entrada.
- El quinto reto fue, qué hacer para que el algoritmo tuviera memoria; es decir, si se llegaba a un “callejón sin salida” cómo hacer para que cuando el algoritmo volviera empezar no se fuera por la misma entrada que recorrió anteriormente. Y esto fue solucionado con la sustitución de los unos por nueves en cada recorrido.

Conclusiones

El crear un programa que resuelva laberintos representó un reto interesante y agradable, el cual me permitió aumentar mis habilidades de programación e investigación. Además, con este proyecto se pudo aplicar lo visto clase y el mismo se adecuaba a que la persona pudiera crearlo con los temas que se sintiera más cómodo.

Aprendí que tener el código en un sólo archivo no precisamente es lo deseado para el lector, ya que suele ser cansado de analizar; entonces al dividir el algoritmo en funciones dependientes y ponerlas en diferentes archivos, permite que el código tenga mayor orden y sea menos pesado para el lector.