Facultad de Ingeniería

Escuela de Ingeniería Eléctrica

Programación Bajo Plataformas Abiertas IE-0117

Proyecto

Resuelve Laberintos

Estudiante: Brandon Andrey Alfaro Herrera

Carné: C00195

Profesor: Juan Carlos Coto Ulate

Grupo: 01

I Semestre, Año 2022

Índice

Introducción

El presente documento es un proyecto del curso de Programación Bajo Plataformas Abiertas de la Universidad de Costa Rica. El tema del proyecto está basado en la resolución de un laberinto en formato .txt con ayuda de programación en el lenguaje C.

La realización del trabajo es con el fin de hacer un programa que pueda resolver laberintos de cualquier tamaño dadas ciertas condiciones y parámetros. Así como documentar los aspectos principales de la elaboración, las dificultades encontradas al hacer la tarea y los resultados obtenidos.

La evaluación de lo hecho reportado en este documento es el propósito académico del proyecto, tomando como criterios los conocimientos adquiridos durante el curso, su buena implementación y organización.

Por otra parte, personalmente es una oportunidad de practicar los conceptos y destrezas que exige el curso, como también un manejo de la resolución de problemas en el ámbito del lenguaje en C.

Diseño, investigación, uso de teoría y utilización de herramientas computacionales fue parte de la metodología usada para este proyecto.

El proyecto consta de dos archivos de código principales, uno basado en memoria estática y otro basado en memoria dinámica.

Diseño general 1(Programa_Main)

Las partes del programa constan de funciones que hacen tareas específicas para resolver el laberinto, las cuales son:

Dimensionar la matriz

Cuenta la cantidad de filas y columnas de la matriz del archivo de texto.

Esta abre el archivo .txt para leerlo y lo cierra cuando se termina de ejecutar el código del bucle. Usando un “while” se hace un bucle que contiene a “fgetc” el cual retorna un carácter del archivo del texto (con cada iteración del while el carácter que retorna “fgetc” es el siguiente en el archivo, uno por uno). El carácter char que se devuelve se compara para identificar si es un dato relacionado con el laberinto. Se usan variables para almacenar el valor de la cantidad de filas y columnas. El argumento para detener el bucle es “feof”, esto retorna un 0 si no se ha terminado de leer el archivo.

Guardar información de la matriz

Pasa la información de la matriz del archivo .txt a la matriz de espacio de juego predefinida.

Usando parte del algoritmo de la función anterior, se comparan los caracteres que devuelve “fgetc” y se sustituyen los valores en la matriz bidimensional definida al principio del código. Dependiendo del carácter que se compara esta suma al índice una unidad(índice de columna o fila) que guía la modificación en la matriz bidimensional.

Identificar los inicios del laberinto

Identifica y cuenta las posiciones iniciales del laberinto.

A partir de bucles “for” se analizan las columnas extremas izquierda y derecha y la primera y última fila de la matriz. Si el dato que se analiza es un “1” , entonces es un inicio por lo que se guarda la información de los índices de fila y columna en la matriz “inicios”. También con una variable se guarda la cantidad de inicios y otra funciona como índice para moverse a la siguiente columna en la matriz de inicios.

Trayectoria posicional en la matriz

A partir de un inicio, recorre el laberinto posición por posición identificando los caminos de las paredes del laberinto y al llegar a un dato de valor 2, se imprime un mensaje de que se hayo la solución.

Con cuatro argumentos (posición anterior fila, posición anterior columna, posición actual fila y posición actual columna), se usan “if” que comparan una posición arriba, abajo, a la izquierda y a la derecha. Dependiendo del dato analizado en cada dirección se identifica si el valor es una pared (0), un camino(1) o una solución(2). Con ayuda de la recursividad y los cuatro argumentos, cuando se identifica un camino se siguen analizando los datos con la nueva información que se tiene de la posición del valor actual y posición del valor anterior, con esto se llega a una especie de bucle en el cual el programa termina cuando se llega a una solución o cuando se recorre todo el camino y no se encuentra ninguna solución.

Ejecución de recorrido en los inicios

Toma la información de un inicio y ejecuta la función anterior explicada.

Se usa un bucle “for” con un índice que usa como limite la cantidad de inicios. Ejecuta la función que recorre el laberinto con la información de la matriz de inicios, con cada iteración del bucle se empieza el recorrido en un inicio diferente.

Tramo de código principal

Esta función es “main” la cual ejecuta a casi todas las funciones del código anteriores.

Diseño general 2 (Programa_Main_con_Punteros)

Las partes del programa constan de funciones llamadas en el archivo principal de diferentes bibliotecas tomadas dentro de la misma dirección que el programa principal que hacen tareas específicas dentro del programa para resolver el laberinto las cuales son:

Dimensionar la matriz

Para determinar el número total de datos se ejecuta la función “dimensiona_la_matriz”, esta se encuentra en la librería “1tamano_matriz.h”.

Para realizar el conteo de datos se usó al igual que en el diseño general 1 casi el mismo código, la diferencia es que en este diseño a esta función se le pasan dos argumentos con la dirección de memoria de las variables globales del código principal para que los valores de la cantidad de filas y columnas se guarden y se puedan utilizar a lo largo del programa principal.

Espacio de memoria dinámica

Con los datos de columnas y filas obtenido en la función anterior, se ejecuta “reserva_espacio” de la biblioteca “2espacio_matriz.h”, para que el programa aparte el espacio disponible en la memoria para guardar los datos de la matriz.

“reserva_espacio” tiene tres argumentos los cuales son; la dirección de memoria del arreglo donde se almacenarán los valores de la matriz del laberinto, la cantidad de filas y la cantidad de columnas. Utilizando “malloc” y “sizeof” se define la cantidad de espacio que se necesita reservar en el arreglo.

Guardar información de la matriz

Leyendo el archivo de texto se va almacenando la información en el arreglo al cual se hizo un puntero para avanzar a la siguiente posición en el arreglo cuando se guarda dato por dato.

Abriendo el archivo para solo lectura y utilizando “feof” y “fgetc” como en el primer diseño, con ayuda del bucle “while” y condicionales con “switch”, se reemplaza el valor en el arreglo con el valor que devuelve “fgetc”. Cada vez que se reemplaza un valor en el arreglo se le suma un valor al puntero para que escriba en la siguiente posición del arreglo.

Identificar los inicios del laberinto

Esta parte del programa consta de una biblioteca que contiene tres funciones.

Primera función, cuenta la cantidad de inicios en los bordes del laberinto. Sabiendo la cantidad de filas y columnas averiguadas anteriormente se sabe la posición en la que están los bordes del laberinto. Como, por ejemplo, los primeros datos del arreglo son los datos de la fila 1 de la matriz del laberinto, o sea la fila superior de la matriz del archivo de texto, los últimos datos del arreglo son la última fila de la matriz y cada ciertos índices con patrón tienen los datos del borde izquierdo y derecho del laberinto. Para recorrer estos datos del arreglo se recurre a un puntero del arreglo. Con ayuda de bucles “for” y condicionales “if” se identifican los inicios del laberinto. Además, los argumentos de esta función son 4; un argumento requiere la dirección de memoria del arreglo para revisar los datos del laberinto, otro la cantidad de filas, uno la cantidad de columnas y el cuarto argumento es la dirección donde se almacena la cantidad de inicios del laberinto.

Segunda función, esta tiene como objetivo reservar el espacio en la memoria que se ocupara para guardar información de las posiciones de los inicios de la matriz. Como en la parte de espacio de memoria dinámica, se utiliza “malloc” y “sizeof”. Los 2 argumentos son la dirección de los arreglos en la memoria donde se reservará el espacio.

Tercera función, cuando se averigua la posición de un inicio en el arreglo, este guarda esas posiciones en un arreglo para la coordenada “x” y un arreglo para la coordenada “y”.

Trayectoria posicional en la matriz

Recorre el camino de 1’s hasta la solución.

Mediante una visualización de la matriz del laberinto se hace que esta función empiece el recorrido en un inicio y con la recursividad de la misma función se “avanza” por el laberinto hasta llegar a la solución.

Ejecución de recorrido en los inicios

Una función que inicia el recorrido del laberinto en los inicios.

Con la información de posición de los arreglos respectivos se inicia un bucle y se llama a la función “recorre_camino1” para que el programa inicie el recorrido en cada principio.

Archivo de código principal

En este archivo .c incluye las bibliotecas que contiene las partes del diseño 2 anteriores y mediante código se llaman a las funciones y le asignan los argumentos y variables para ejecutar y copilar el programa en conjunto. Nota importante: El diseño general 2 no funciono completamente.

Principales retos

1. Como leer la información del documento y guardarla en la memoria.

Se tuvo que investigar como leer la información del archivo .txt y como identificar la información contenida en este para poder guardarla en una variable o en una matriz.

2. Modificación del programa funcional de manera que se implementaran punteros para memoria dinámica.

El “Programa_Main” funciona con memoria estática predefiniendo el tamaño de la matriz a utilizar. Para hacer un programa que solucione laberintos con memoria dinámica se necesita restructurar o rehacer parte del código de manera que funcione con punteros.

3. Uso de la memoria dinámica para reservar solo memoria necesaria.

Hacer el código de manera que leyera la cantidad de datos de la matriz del archivo .txt y reservar esa cantidad de espacio para guardar la información en la matriz unidimensional que se usara.

4. Algoritmo de recorrido del laberinto.

Idear como avanzar a la siguiente posición en el que había un camino y que el programa no identificara como siguiente posición la posición anterior en la que se estuvo. En el diseño 1 se ideo más sencillo en comparación con el diseño 2 ya que en el primero se usaban posiciones en una matriz predefinida bidimensional por lo que se podía acceder a los valores con índices, pero cambio en el diseño 2 ya que se tenía contenida una matriz bidimensional en un arreglo o matriz unidimensional en el que se accedía a los valores mediante un puntero.

5. Arreglos bidimensionales con espacio indefinido dependiendo del tipo de laberinto

Un reto no comprender porque se tiene que definir el espacio a usar en una matriz bidimensional y en una unidimensional no. A raíz de esto tener que usar una matriz de una dimensión para poder almacenar una matriz de dos dimensiones, solo con el espacio de memoria requerido.

6. Aprendizaje de utilización de la herramienta de GitHub para las ramificaciones posibles en el trabajo.

Se torno difícil empezar a utilizar la herramienta GitHub, ya que no se tenía muy claro como ejecutar las funcionalidades de la herramienta.

7. Copilar el diseño 2 del programa.

No se logró hacer copilar el programa, ya que el algoritmo de recorrido del laberinto presenta problemas al ejecutarse.

Conclusiones

En general es un placer esforzarse y poder al final del trabajo resolver problemas del proyecto. Idear algoritmos para hacer una tarea especifica es sin duda un objetivo que lleva tiempo y conlleva a una mejora en la resolución de problemas cuando se está aprendiendo. Es verdaderamente fructífero tener teoría y necesitar investigar más allá de lo brindado en clase, pero en ocasiones se hace tedioso e irritante intentar encontrar la solución a un problema en concreto. Una de las lecciones aprendidas es a tener que investigar más allá de lo aprendido para tener una mejor ejecución y entendimiento de lo aprendido en clase. Practicar los conceptos y materia obtenida en clase más allá de los ejemplos básicos, ayuda a tener un mayor panorama de en qué puede ser utilizado el código de programación. Una buena pregunta personalmente es si copiar códigos ya hechos que cumplen con las tareas que se necesitan resolver es buena o mala práctica o depende de otros factores. Mientras se investigaba se llegó a una información de punteros dobles, triples, etc., se tiene curiosidad de cómo, porque y para que se utilizan esas mecánicas de código. Aparte de las funciones utilizadas en este trabajo para leer archivos, cuáles son las funciones para leer o escribir sobre otro tipo de archivos diferentes a .txt.