Fundamentos de Programación y Laboratorio.

Práctica No. 3

Resolución de un laberinto

Allan Jair Escamilla Hernández
Raúl González Portillo
Daniel Logvin

El siguiente documento presenta el análisis, diseño y codificación que se desarrolló para la creación de un programa que resuelve un laberinto haciendo uso de funciones recursivas. El programa debe encontrar el camino más eficiente para llegar a la salida y desplegar el número de pasos efectuados para llegar al punto final.

I. Introducción

- **Descripción del sistema**

Dado un laberinto, se requiere un programa que permita encontrar el mejor camino que lleve del punto de entrada al punto de salida. Se entiende por el "mejor camino" a aquel que recorre el menor número de pasos. El laberinto siempre deberá de contar con una entrada, con una salida y con al menos un camino que permita llegar desde la entrada hasta la salida.

El programa será ejecutado desde la línea de comandos y, en función del argumento que se indique, podrá presentar la solución de dos formas posibles:

A. $ laberinto nombre_archivo
B. $ laberinto nombre_archivo -pasos

Si no se indica argumento, al momento de ejecutar el programa se despliega en pantalla el laberinto, se presiona la tecla <enter> y se despliega la solución óptima, indicando cuántos pasos hay entre la entrada y la salida e indicando cuántos caminos de salida se encontraron.

Si se indica el argumento —pasos, al ejecutar el programa se despliega en pantalla el laberinto y, al presionar la tecla <enter>, comienza a desplegarse el proceso de resolución para finalizar con el despliegue del laberinto con el camino óptimo marcado, el número de pasos entre la entrada y la salida e indicando cuántos caminos de salida se encontraron.

- **Alcances y limitaciones**

Al momento de ejecutar el programa se debe validar que el argumento que se pasa, si es el caso, sea correcto, así como que no se reciba más de dos argumentos. Se deberá desplegar el mensaje de error correspondiente, así como la forma adecuada de usar el programa; por ejemplo, si el usuario ejecuta: $laberinto —p el programa debe contestar: “Error, opción incorrecta. “

Uso:

- $ laberinto nombre_archivo
- $ laberinto nombre_archivo -pasos

No se considera necesario contemplar algún esquema adicional de validación. Se asume que el programador siempre usará un laberinto cuyo perímetro sea una pared, sin importar si norma una figura regular o irregular, así como que siempre habrá una entrada, una salida y al menos un camino que permita ir de la entrada a la salida.

El tamaño máximo del laberinto es de 30 columnas por 30 renglones.

- **Planeación de la solución**

Para resolver este problema, se efectuaron ciertos análisis antes de empezar a escribir el código, se hizo un pseudocódigo, un diagrama IPO y se discutió la manera en la que el problema sería resuelto entre los miembros del equipo.

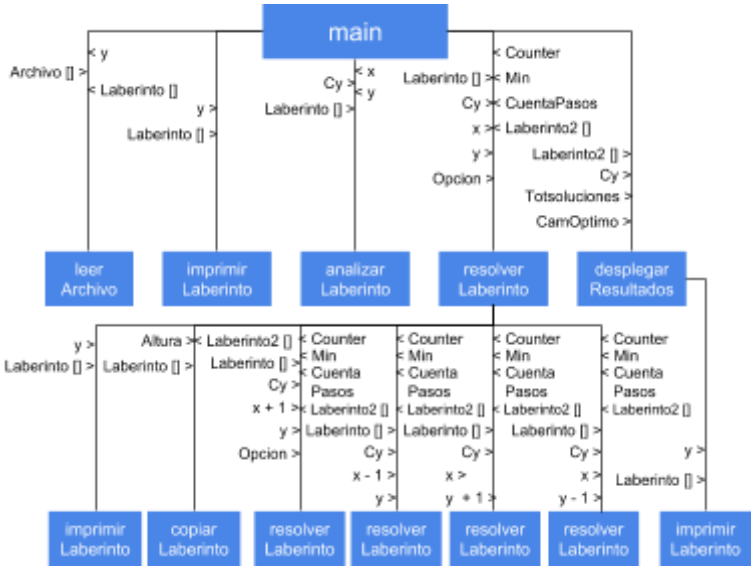
En este programa se utilizaron:

- Recursiones:
  - Dado que la resolución del laberinto es un proceso que se basa en su propia definición, utilizaremos funciones recursivas.
- Archivos:
  - Se utilizarán archivos que se abrirán por el programa, en estos se guardarán los laberintos a resolver.
- Argumentos:
  - El programa recibirá argumentos para poder comenzar su funcionamiento, en estos se indicará el nombre del archivo que contiene el laberinto a resolver, así como la indicación por parte del usuario, si es que este quiere que se despliegue en pantalla la solución óptima.

II. Análisis

- **Entradas:**
  - Nombre del archivo del laberinto **(Archivo [])**
- **Procesos:**
  - Leer un archivo de texto **(leerArchivo)**
  - Imprimir el laberinto **(imprimirLaberinto)**
  - Buscar las coordenadas X y Y de entrada **(analizarLaberinto)**
  - Avanzar un paso en el laberinto **(resolverLaberinto)**
  - Copiar la mejor solución **(copiarLaberinto)**
- **Salidas:**
  - Altura del laberinto **(y / Cy / Altura)**
  - Laberinto resuelto **(Laberinto2 [])**
  - Numero de soluciones **(Totsoluciones / Counter)**
  - Número de pasos en el camino optimo **(Min / CamOptimo)**

- **Diagrama de Entradas, Procesos y Salidas (IPO):**



III. Diseño

- **Pseudocódigo:**

```
Principal(argc | argv[]) {
 Validacion[] = "-p";
 si((argc < 2 OR argc > 3) OR
(comparar(argv[1], Validacion) == 0)){
   imprime ("Error, opcion incorrecta\n");
 }si no{
   si(argc == 3 && comparar(argv[2],
Validacion) != 0){
     imprime("Error, opcion
incorrecta\n");
   }si no{
     si (argc == 3 && comparar(argv[2],
Validacion) == 0)
       flag = 1;
     strcpy(Archivo, argv[1]);
     leerArchivo(Archivo | Laberinto, y);
     imprimirLaberinto(Laberinto, y | );
     imprime("Presione enter para
continuar... ");
     leer();
     analizarLaberinto(Laberinto, y |
Entradax, Entraday);
     resolverLaberinto(Laberinto,
Entradax, Entraday, y| contador, Min,
CuentaPasos, Laberinto2);
     desplegarResultados(Laberinto2,
contador, Min, y | );
   }
 }
 devolver 0;
}
leerArchivo ( Archivo[] |
Laberinto[30][30], y) {
 Abrir (Archivo, "r");
 i = 0;
 si (Arch == NULL) {
   imprime("Error, opción incorrecta\nEl
archivo no fue encontrado\n");
   salir(0);
 }
 mientras(!findearchivo(Arch)){
   leer(Laberinto[i], 31, Arch);
   i++;
 }
 y = i;
 cerrar(Arch);
}
imprimirLaberinto(Laberinto[30][30], y |
){
 sistema("limpiar");
 j = 0;
```

```
 desde i = 0 hasta i < y-1; i++ {
   j = 0;
   mientras (Laberinto[i][j] != '\n') {
     imprime(Laberinto[i][j]);
     j++;
   }
   imprime("\n");
 }
}
analizarLaberinto( Laberinto[30][30], cy |
x, y){
 j = 0;
 desde i = 0 hasta i < cy-1; i++ {
   j = 0;
   mientras(Laberinto[i][j] != '\n') {
     j++;
     si(Laberinto[i][j] == 'E') {
       x = i;
       y = j;
     }
   }
 }
}
resolverLaberinto(Laberinto[30][30], x, y,
Cy, Opcion|Counter, Min, CuentaPasos,
Laberinto2[30][30]){
 si (Laberinto[x][y] != 'S') {
   si(Opcion == 1){
     imprimirLaberinto(Laberinto, Cy | );
     sistema("dormir 0.1");
   }
   si(Laberinto[x+1][y] == 'S' OR
Laberinto[x][y+1] == 'S' OR
Laberinto[x-1][y] == 'S' OR
Laberinto[x][y-1] == 'S'){
     Counter++;
     if(CuentaPasos < Min OR Counter ==
1){
       Min = CuentaPasos;
       copiarLaberinto(Laberinto, Cy |
Laberinto2);
     }
   }
   si(Laberinto[x][y+1] != '*' AND
Laberinto[x][y+1] != 'S' AND
Laberinto[x][y+1] != '.' AND
Laberinto[x][y+1] != 'E') {
     Laberinto[x][y+1] = '.';
     CuentaPasos++;
     resolverLaberinto(Laberinto, x, y+1,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion | );
   }
   si (Laberinto[x-1][y] != '*' AND
Laberinto[x-1][y] != 'S' AND
Laberinto[x-1][y] != '.' AND
Laberinto[x-1][y] != 'E') {
     Laberinto[x-1][y] = '.';
     CuentaPasos++;
     resolverLaberinto(Laberinto, x-1, y,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion | );
   }
   si (Laberinto[x+1][y] != '*' AND
Laberinto[x+1][y] != 'S' AND
Laberinto[x+1][y] != '.' AND
Laberinto[x+1][y] != 'E') {
     Laberinto[x+1][y] = '.';
     CuentaPasos++;
     resolverLaberinto(Laberinto, x+1, y,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion | );
   }
   si (Laberinto[x][y-1] != '*' AND
Laberinto[x][y-1] != 'S' AND
Laberinto[x][y-1] != '.' AND
Laberinto[x][y-1] != 'E') {
     Laberinto[x][y-1] = '.';
     CuentaPasos++;
     resolverLaberinto(Laberinto, x, y-1,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion | );
   }
   si(Laberinto[x][y] != 'E'){
     Laberinto[x][y] = ' ';
     *CuentaPasos--;
   }
   si(Opcion == 1){
     imprimirLaberinto(Laberinto, Cy | );
     sistema("dormir 0.1");
   }
```

```
 }
}
copiarLaberinto( Laberinto[30][30], Altura
| Laberinto2[30][30]){
 i = 0;
 mientras (i < Altura) {
   copiar(Laberinto2[i], Laberinto[i]);
   i++;
 }
}
desplegarResultados(Laberinto2[30][30],
Totsoluciones, CamOptimo, Cy | ) {
 imprimirLaberinto(Laberinto2, Cy | );
 printf("Camino optimo: “, CamOptimo, “
pasos.\n");
 printf("Se encontraron “, Totsoluciones,
“ caminos de salida.\n");
}
```

IV. Código fuente

```
//  Incluimos las librerias
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// PROTOTIPOS DE FUNCIONES
void leerArchivo(char Archivo[], char
Laberinto[30][30], int* y); // Funcion que
carga el archivo y lo copia a una matriz
void imprimirLaberinto(char
Laberinto[30][30], int y); // Funcion que
imprime el laberinto
void analizarLaberinto(char
Laberinto[30][30], int* x, int* y, int
cy); // Funcion que encuentra las
coordenadas del laberinto
void resolverLaberinto(char
Laberinto[30][30], int x, int y, int Cy,
int* Counter, int* Min, int* CuentaPasos,
char Laberinto2[30][30], int Opcion); //
Funcion que resuleve el laberinto
void copiarLaberinto(char
Laberinto[30][30], char
Laberinto2[30][30], int Altura); // Copia
de un laberinto a otro
void desplegarResultados(char
Laberinto2[30][30], int Totsoluciones, int
CamOptimo, int Cy); // Despliega los
resultados
// FIN DE PROTOTIPOS DE FUNCIONES


// FUNCION PRINCIPAL
int main(int argc, char *argv[]) {
 // Declaracion de variables
 char Archivo[50], Validacion[] = "-p"; //
Nombre del archivo y variable para validar
la ejecucion del programa
 char Laberinto[30][30]; // Arreglo
bidimensional que almacena el laberinto
 char Laberinto2[30][30]; // Arreglo que
almacena el laberinto con la solucion
optima
 int y; // Altura del laberinto
 int Entradax = 0, Entraday = 0; //
Coordenadas de la entrada
 int contador = 0; // Variable para la
cantidad de salidas
 int Min = 0; // Variable que almacenará
el camino minimo
 int CuentaPasos = 0; // Variable que
contará los pasos
 int flag = 0; // Variable para determinar
si se imprimen los pasos o no
 // Fin de declarcion de variables
 if((argc < 2 || argc > 3) ||
(strcmp(argv[1], Validacion) == 0)){ //
Validamos la cantidad de parametros y
formato
   printf("Error, opcion incorrecta\n");
 }else{
   if(argc == 3 && strcmp(argv[2],
Validacion) != 0){ // Validamos que el
tercer parametro no sea distinto de -p
     printf("Error, opcion incorrecta\n");
   }else{
     if (argc == 3 && strcmp(argv[2],
Validacion) == 0) // En caso de que el
usuario quiera los pasos, asignamos 1 a la
variable flag
       flag = 1;
```

```
      strcpy(Archivo, argv[1]); // Copiamos
a la variable Archivo el argumento 1 que
se paso por terminal
      leerArchivo(Archivo, Laberinto, &y);
// Leemos el archivo
      imprimirLaberinto(Laberinto, y); //
Imprimimos el laberinto
      printf("Presione enter para
continuar... ");
      getchar(); // Pausa para visualizar
el laberinto
      analizarLaberinto(Laberinto,
&Entradax, &Entraday, y); // Encontramos
las coordenadas de la entrada
      resolverLaberinto(Laberinto,
Entradax, Entraday, y, &contador, &Min,
&CuentaPasos, Laberinto2, flag); //
Resolvemos el laberinto
      desplegarResultados(Laberinto2,
contador, Min, y); // Desplegamos los
resultados
    }
  }
  return 0;
}
// FIN DE FUNCION PRINCIPAL

// DESARROLLO DE LAS FUNCIONES
void leerArchivo (char Archivo[], char
Laberinto[30][30], int* y) { // Funcion
que lee el archivo
  FILE* Arch = fopen(Archivo, "r"); //
Abrimos el archivo
  int i = 0; // Inicializamos a i
  if (Arch == NULL) { // Verificamos que no
haya habido ningun problema al momento de
abrir el archivo
    printf("Error, opción incorrecta\nEl
archivo no fue encontrado\n");
    exit(0);
  }
  // Copiamos el contenido del archivo a un
arreglo bidimensional
  while(!feof(Arch)){ // Mientras que no
sea el final del archivo, leemos el
laberinto
    fgets(Laberinto[i], 31, Arch);
    i++;
  }
  *y = i; // Obtenemos la altura del
laberinto
  fclose(Arch); // Cerramos el archivo
}
void imprimirLaberinto(char
Laberinto[30][30], int y){ // Funcion que
imprime en pantalla el laberinto
  system("clear"); // Limpiamos pantalla
  int j = 0; // Inicializamos j
  for(int i = 0; i < y-1; i++){ //
Establecemos dos ciclos para imprimir el
laberinto
    j = 0;
    while (Laberinto[i][j] != '\n') {
      printf("%c", Laberinto[i][j]);
      j++;
    }
    printf("\n");
  }
}
void analizarLaberinto(char
Laberinto[30][30], int* x, int* y, int
cy){ // Funcion que obtiene las
coordenadas de la entrada
  int j = 0;
  for(int i = 0; i < cy-1; i++){ //
Recorremos la altura
    j = 0;
    while (Laberinto[i][j] != '\n') { //
Recorremos el ancho
      j++;
      if (Laberinto[i][j] == 'E') { //
Obtenemos las coordenadas de la Entrada
        *x = i;
        *y = j;
      }
    }
  }
}
void resolverLaberinto(char
Laberinto[30][30], int x, int y, int Cy,
int* Counter, int* Min, int* CuentaPasos,
```

```
char Laberinto2[30][30], int Opcion){ //
Funcion que resuelve el laberinto
 if (Laberinto[x][y] != 'S') { //
Verificamos que no nos encontremos en la
salida
   if(Opcion == 1){
     imprimirLaberinto(Laberinto, Cy); //
Imprimimos el laberinto
     system("sleep 0.1"); // Hacemos una
pausa de 5 milisegundos
   }
   if(Laberinto[x+1][y] == 'S' ||
Laberinto[x][y+1] == 'S' ||
Laberinto[x-1][y] == 'S' ||
Laberinto[x][y-1] == 'S'){ // Contamos las
veces que encontramos la salida
     (*Counter)++;
     if(*CuentaPasos < *Min || *Counter ==
1){
       *Min = *CuentaPasos;
       copiarLaberinto(Laberinto,
Laberinto2, Cy);
     }
   }
   if (Laberinto[x][y+1] != '*' &&
Laberinto[x][y+1] != 'S' &&
Laberinto[x][y+1] != '.' &&
Laberinto[x][y+1] != 'E') { // Condicion
para que se mueva a la derecha
     Laberinto[x][y+1] = '.';
     (*CuentaPasos)++;
     resolverLaberinto(Laberinto, x, y+1,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion);
   }
   if (Laberinto[x-1][y] != '*' &&
Laberinto[x-1][y] != 'S' &&
Laberinto[x-1][y] != '.' &&
Laberinto[x-1][y] != 'E') { // Condicion
para que se mueva hacia arriba
     Laberinto[x-1][y] = '.';
     (*CuentaPasos)++;
     resolverLaberinto(Laberinto, x-1, y,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion);
   }
   if (Laberinto[x+1][y] != '*' &&
Laberinto[x+1][y] != 'S' &&
Laberinto[x+1][y] != '.' &&
Laberinto[x+1][y] != 'E') { // Condicion
para que se mueva hacia abajo
     Laberinto[x+1][y] = '.';
     (*CuentaPasos)++;
     resolverLaberinto(Laberinto, x+1, y,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion);
   }
   if (Laberinto[x][y-1] != '*' &&
Laberinto[x][y-1] != 'S' &&
Laberinto[x][y-1] != '.' &&
Laberinto[x][y-1] != 'E') { // Condicion
para que se mueva a la izquierda
     Laberinto[x][y-1] = '.';
     (*CuentaPasos)++;
     resolverLaberinto(Laberinto, x, y-1,
Cy, Counter, Min, CuentaPasos, Laberinto2,
Opcion);
   }
   if(Laberinto[x][y] != 'E'){ //
Verificamos que no nos encontremos en la
salida para comenzar a retroceder
     Laberinto[x][y] = ' '; // Limpiamos
la casilla
     (*CuentaPasos)--;
   }
   if(Opcion == 1){
     imprimirLaberinto(Laberinto, Cy); //
Imprimimos el laberinto
     system("sleep 0.1"); // Pausamos
durante 5 milisegundos
   }
 }
} // funcion que resuelve el laberinto
void copiarLaberinto(char
Laberinto[30][30], char
Laberinto2[30][30], int Altura){ //
Funcion que copia de una matriz a otra
 int i = 0; // Copiamos el laberinto a una
nueva matriz
 while (i < Altura) {
   strcpy(Laberinto2[i], Laberinto[i]);
   i++;
```

```
 }
}
void desplegarResultados(char
Laberinto2[30][30], int Totsoluciones, int
CamOptimo, int Cy) { // Funcion que
despliega los resultados obtenidos
 imprimirLaberinto(Laberinto2, Cy); //
Imprime laberinto
 printf("Camino optimo: %d pasos.\n",
CamOptimo);
 printf("Se encontraron %d caminos de
salida.\n", Totsoluciones);
}
// FIN DE DESARROLLO DE FUNCIONES
```

V. Conclusión

El programa se pudo desarrollar gracias a la planeación y el análisis del problema que se propuso; el análisis del problema fue el paso principal para poder concluir el algoritmo e interpretarlo en código para que funcione de manera correcta.

Después del análisis comenzó la fase de codificación, donde varias líneas del código fueron comentadas para poder continuar su desarrollo sin tener que analizar linea por linea e interpretar el funcionamiento de cada una de las mismas. Finalmente en la fase de pruebas, el código tuvo que ser modificado en ciertas ocasiones para eliminar errores en la compilación y lógica del programa.

Al terminar de programar el laberinto, se logró desarrollar un mayor entendimiento sobre cómo funciona la recursividad a nivel conceptual y práctico.

Además de lo anterior, en la práctica aprendió más sobre el trabajo en equipo, ya que se usó de Github para controlar las diferentes versiones del programa, lo que permitió mantener un ambiente de trabajo controlado, en el que todos los participantes aportaron al código fuente desde sus propios equipos. Al final, se consiguió que el funcionamiento del programa fuera como se especificó.

VI. Referencias

Sznajdleder P.. (2012). Algoritmos a fondo con implementaciones en C y Java. México: Alfaomega.