MANUAL TÉCNICO TRAYECTORIA DE UN ROBOT

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

PRESENTADO POR:

Garzón Mora Daniel Felipe - 20202020123
Juan Sebastián Garnica León - 20202020124
Valencia Gutiérrez Simón Andrés - 20202020025

PRESENTADO A:

JOSÉ DAVID ÁLVAREZ PLATA

PROGRAMACIÓN BÁSICA

BOGOTÁ D.C

26 DE FEBRERO DE 2021

TABLA DE CONTENIDOS

Contenidos

Ilustraciones

Tablas

1. INTRODUCCIÓN

Para el ingeniero de sistemas, dar solución a un problema de programación constituye un gran porcentaje de su campo laboral, y por lo tanto se podría decir que será muy común para él toparse con ejercicios de este tipo durante el desarrollo de su carrera a lo largo de su vida, por esto es necesario aprender las bases de la programación desde el primer momento que inicia su carrera universitaria. Adquirir estos conocimientos básicos en programación implica tener autodisciplina y determinación de un objetivo, para poder sobrellevar dicho curso mientras aprende lo que será el inicio de una gran competencia por ser mejor cada día. En este caso nosotros como estudiantes de primer semestre de Ingeniería de sistemas en la Universidad Distrital Francisco José de Caldas también nos vemos en el deber de tomar esta materia para ir generando los conocimientos que en un futuro necesitaremos para desarrollar cualquier trabajo relacionado con la profesión. Al ser un curso tan necesario se requiere realizar una evaluación periódicamente, en este caso la evaluación se fundamenta en la resolución de un problema llamado “trayectoria de un robot”, dicha resolución se explica en el presente documento, basada en nuestra manera de resolverlo.

2. OBJETIVOS

El objetivo de este trabajo es lograr dar solución al problema de “trayectoria de un robot” propuesto por el maestro José Plata como proyecto final del curso de Programación Básica siguiendo una serie de pasos debidamente organizados, y aplicando un gran numero de conocimientos adquiridos durante el paso por este curso. Además de esto verlo como un reto para medirnos a si mismos como alumnos y probar nuestros conocimientos. Claramente con el objetivo también de hacer una simulación de resolución de un ejercicio que se puede aplicar a cualquier campo de la vida en un futuro, ya sea educativo o laboral.

3. DESARROLLO DEL PROBLEMA

Como se dijo anteriormente para llegar a la conclusión de dicho problema fue necesario aplicar una serie de pasos debidamente ordenados, estos fueron los que nos permitieron llegara a dicha solución de manera ordenada y eficiente ya que como alguna vez el profesor nos dijo “lo primero que se hace no es echar código”. Los pasos son los siguientes

3.1 Análisis del problema

Lo primordial para resolver un problema de programación es tener claro lo que se nos pide, como lo dice la Universidad Nacional de la Plata “En esta primera etapa, se analiza el problema en su contexto del mundo real. Deben obtenerse los requerimientos del usuario. El resultado de este análisis es un modelo preciso del ambiente del problema y del objetivo a resolver. Dos componentes importantes de este modelo son los datos a utilizar y las transformaciones de los mismos que llevan al objetivo.”

En este Caso nosotros desarrollamos el problema dentro del margen de pasos nombrados por la universidad de la plata, la primera etapa consistió en leer el documento que contenía el problema y entender lo que se nos pedía, en primera instancia la lectura del documento no fue suficiente, quedaban algunos cabos sueltos que fuimos ajustando mientras releíamos y anotábamos en una hoja un esquema que nos permitía entender la entrada de datos y como se debía hacer algunas validaciones de datos. Además de esto intentábamos pensar como llegaríamos a resolver cada uno de los problemas que íbamos encontrando. Primero se hicieron pruebas de escritorio en distintas hojas, las pruebas consistían en simular la entrada de datos y los movimientos que seguiría el robot para poder generar luego una síntesis de la información que se repite en todas las pruebas (Véase ilustración 1).

Lo primero que se escribió sobre el papel fue una matriz, y dentro se le asignaban valores entre 1 y 0 para definir las minas y los espacios vacíos respectivamente, luego dentro de dicha matriz se ubicó la posición inicial con una flecha (esta posición no puede estar sobre una mina o lo que es lo mismo que en la ubicación de la matriz no puede valer 1) y la final con un círculo, luego de eso nos percatamos que al

declarar la posición inicial se tenía que ingresar la orientación del robot, y que esta orientación indicaba una flecha diferente que apuntaba de la siguiente manera según el caso:

N: hacia la parte superior de la matriz (norte)→ ∆

S: hacia la parte inferior de la matriz (sur) → ∇

E: hacia el lado derecha de la matriz (este) →⊳

O: hacia el lado izquierdo de la matriz (oeste) →⊲

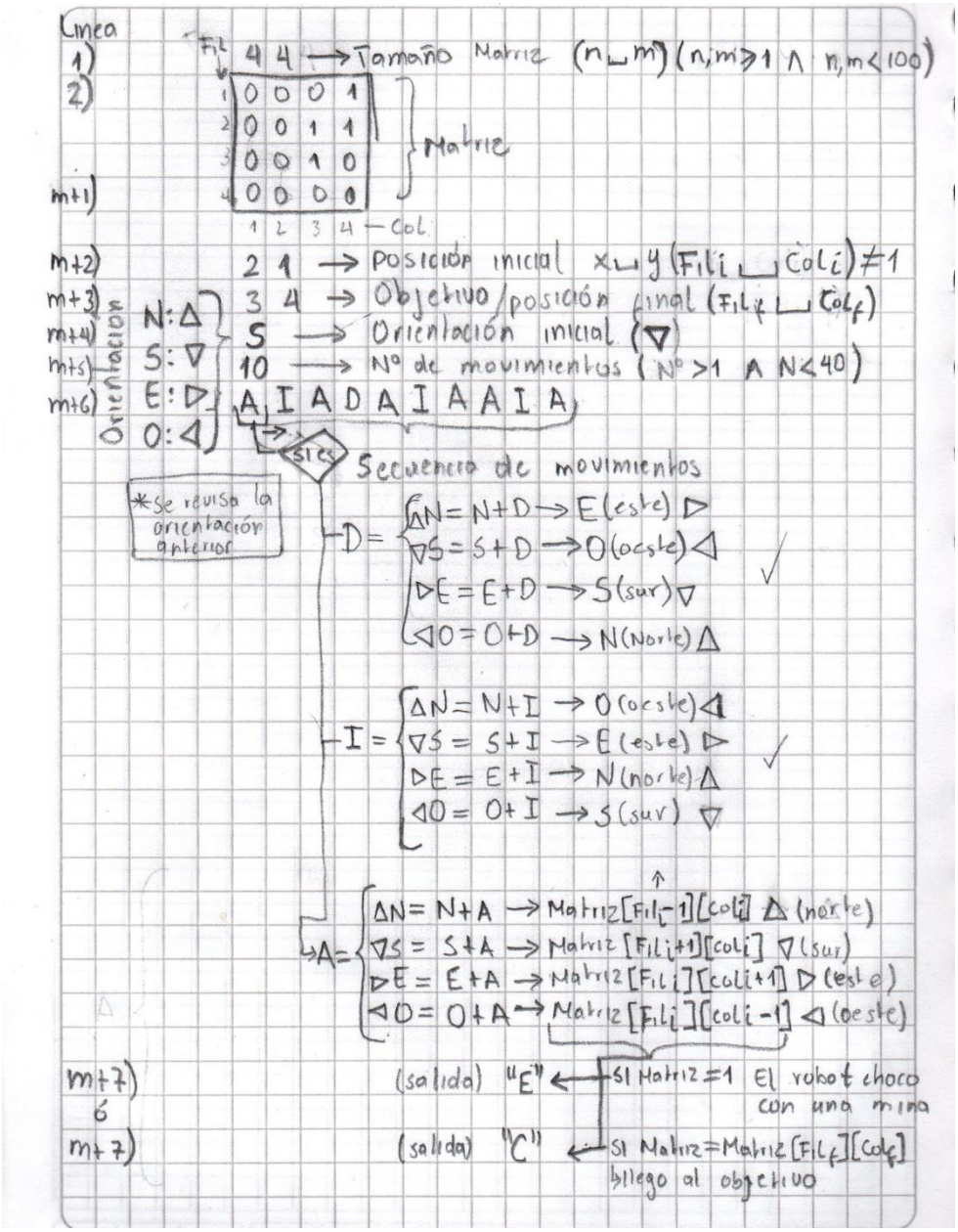

Ilustración 1:resolución del problema en escritorio

3.2 Diseño de la solución

En este caso luego de realizar distintas pruebas de escritorio procedimos a hacer el planteamiento de un algoritmo, a mano, en el cual se planteaban los pasos a seguir en un orden establecido, dicho orden fue con el que se culminó el proyecto en su versión final. A continuación, se enumerarán en orden los pasos a seguir por el programa según nuestro planteamiento:

1. Apertura de archivo
2. Lectura por líneas
3. Conversión de **char** a **int**
4. Asignación de valores en variables
5. Generar matriz[filas][columnas]
6. Llenar matriz con datos del archivo de texto
7. Leer posición inicial y final
8. Leer orientación inicial
9. Clasificar por casos la orientación
10. Leer número de pasos (entre 1 y 40)
11. Leer INDIVIDUALMENTE cada opción de movimiento de la flecha.
12. Ejecutar secuencia de comandos que modifiquen la matriz según caso.
13. Verificación de terminación (si llegó al destino o no).
14. Escribir en el archivo de texto la letra “C” en caso de llegar al destino, o escribir la letra “E” en caso contrario.
15. Cierre del archivo de texto
16. Finalización del programa

Luego de dar estos pasos de solución se procedió a generar algunas partes del seudocódigo en PSeInt ya que esta herramienta no permitía hacer algunas de las cosas planteadas en la anterior lista. (ANEXO) Además de esto también se genero un diagrama de flujo en el cual se evidencia la solución de manera gráfica, y toda la secuencia de actividades del proceso llevado a cabo. (ANEXO)

3.3 Codificación del algoritmo en lenguaje c++

A la hora de comenzar a programar en el lenguaje aprendido durante el curso (c++) nos dimos cuenta que en el análisis del problema que habíamos propuesto se ajustaba a lo que habíamos logrado aprender, por lo tanto, iniciamos declarando variables y librerías que serían necesarias para poder ejecutar el código, sin embargo, es inevitable encontrarse con errores en la sintaxis haciendo que el programa no funcione, nos encontramos principalmente con estos obstáculos:

3.3.1 ***Lectura de datos de entrada:*** No sabíamos de qué manera se haría la captura de datos, ya que para dar inicio al programa había que manejar un archivo ROBCOM.txt desde el cual se haría lectura y escritura de datos, afortunadamente este error se solucionó implementando la librería `#include <fstream>` que nos permitía por medio de la función:

```
public FILE *__cdecl fopen (const char * __restrict__ _Filename, const char * __restrict__ _Mode)
```

Abrir un archivo asignando los parámetros correspondientes, el primero era el nombre del archivo y el segundo indicando uno de los siguientes modos (véase tabla 1) , en nuestro caso escogimos el modo “r+” el cual abre archivos, los lee e ingresa datos en el mismo.

w	crea un fichero de escritura. Si ya existe lo crea de nuevo
w+	crea un fichero de lectura y escritura. Si ya existe lo crea de nuevo
a	abre o crea un fichero para añadir datos al final del mismo
a+	abre o crea un fichero para leer y añadir datos al final del mismo
r	abre un fichero de lectura
r+	abre un fichero de lectura y escritura

Tabla 1: modos de apertura de ficheros

Quedando así `FILE *archivo = fopen(nombreArchivo, "r+");` donde el nombre de archivo es pasado por medio de un apuntador del archivo dentro de los parámetros de la función `int **ejecutar(char *nombreArchivo){` . Esta función es llamada en el main en donde se le pasa el nombre del archivo que abrirá, `ejecutar("ROBCOM2.TXT");` , este archivo

tiene que estar en la misma carpeta en donde se encuentra el programa, de no ser así es necesario dictarle toda la ruta al programa para que acceda al archivo.

3.3.2 ***Identificar saltos de línea:*** Otro problema con el que nos encontramos era el cómo hacíamos para identificar cuando saltara de línea y comenzara a leer datos que posteriormente asignaría variables o generaría matrices, para poder hacer esto recurrimos de nuevo a internet donde en una página web logramos encontrar una función que hacía parte de la librería que utilizamos dicha función según la página "está diseñada para leer cadenas de caracteres. Leerá hasta n-1 caracteres o hasta que lea un cambio de línea '\n' o un final de archivo EOF. En este último caso, el carácter de cambio de línea '\n' también es leído." (Wikibooks, 2019) sí nos damos cuenta esta función es todo lo que necesitamos para cumplir con nuestro objetivo.

```
fgets(linea,150, archivo);
token = strtok(linea, " ");          //guardamos la posicion inicial del robot
posicionx = atoi(token);
token = strtok(NULL, " ");
posiciony= atoi(token);
if (posicionx<1 || posicionx>filas || posiciony<1 || posiciony>columnas ){
    cout<<"no es posible colocar estas coordenadas de incio en la matriz"<<endl;
    return 0;
}
```

Ilustración 2: implementación del fgets en el código fuente c++

3.3.3 ***No leer los espacios que separaban*** `#include <string.h>` ***los números:*** Este problema fue uno de los mas difíciles de resolver que al estar trabajando con cadenas de caracteres **char** necesitábamos eliminar dicho espacio que separaba los datos y hacer la conversión de esos datos a números enteros. Para la resolución de la primer parte de este problema se logró encontrar una función llamada *strtok* que pertenecía a la librería y en pocas palabras permitía separar las cadenas de caracteres en una serie de tokens (tutorialspoint, s.f.), lo que luego hicimos es asignarle un valor nulo a los espacios que separaban los valores y los que no estaban nulos llamarlos token. Luego para poder convertir cada token **char** a números

a enteros la misma librería nos ofrecía una función llamada **atoi**, esta nos permitió dicha acción tan solo con ingresar en los parámetros el token adquirido anteriormente (Véase *Ilustración 2*).

3.4 Explicación del código paso a paso

Luego de haber resuelto estos obstáculos comenzamos a desarrollar el código en el orden propuesto en la prueba de escritorio (véase *Ilustración 1*), iniciando desde la función **main( );** llamando al subproceso o función **ejecutar("ROBCOM.TXT");** cómo se puede observar el parámetro que se pasa es el nombre del archivo del que se van a leer los valores como se indica en el índice 3.3.1. Una vez leída la primera línea del archivo de texto se genera una matriz con los tamaños indicados por las variables $filas$ y $columnas$, esta matriz es llamada $matriz[filas][columnas]$ y se comienza llenar con los valores proporcionados en el archivo de texto, los cuales contienen valores de $\emptyset$ $en\ caso\ de\ espacio\ vacio$ y 1 $en\ caso\ de\ una\ mina$.

A continuación, se da un sato de línea con la función fgets y se guardan las coordenadas de posición inicial (x, y) del robot en las variables $posicionx$ y $posiciony$. Para la coordenada del objetivo se utiliza la misma estructura, pero se cambian las variables a $finposicionx$ y $finposiciony$, no hay que olvidar que estas variables tienen unas condiciones, dichas condiciones son las siguientes:

$$(1 \leq posicionx \leq filas \ \wedge \ 1 \leq posiciony \leq columnas)$$

```
fgets(linea,150, archivo);
token = strtok(linea, " ");         //guardamos la posicion inicial del robot
posicionx = atoi(token);
token = strtok(NULL, " ");
posiciony= atoi(token);
if (posicionx<1 || posicionx>filas || posiciony<1 || posiciony>columnas ){
    cout<<"no es posible colocar estas coordenadas de incio en la matriz"<<endl;
    return 0;
}

//cout<<"posicion en x "<<posicionx<<" posicion en y"<<posiciony<<endl;


fgets(linea,150, archivo);
token = strtok(linea, " ");
finposicionx = atoi(token);         //guardamos el destino  del robot
token = strtok(NULL, " ");
finposiciony= atoi(token);
if (finposicionx<1 || finposicionx>filas || finposiciony<1 || finposiciony>columnas ){
    cout<<"no es posible colocar estas coordenadas de posicion final en la matriz"<<endl;
    return 0;
}

//cout<<"Llegada posicion en x "<<finposicionx<<" posicion en y "<<finposiciony<<endl;
```

Ilustración 3: lectura de posición inicial y objetivo en código fuente c++

Luego continuamos verificando que el robot no inicie sobre una mina, esto mediante un $\boldsymbol{if}\ (matriz[posicionx][posicion\ y] == 1)$**,** si el robot no inicia sobre una mina continua guardando la dirección inicial (N, S, E, O) y asignándole un símbolo ASCII a cada caso, en este caso es guardado en la variable $flecha$ (este símbolo es asignado por un número según la lista de símbolos ASCII de la página web con nombre "Códigos Alt para símbolo" (fsymbols, 2020)), y además se le asigna un numero entero del 1 al 4, esto para que más adelante haga la verificación de que caso tomar para iniciar con el primer movimiento dado por el usuario, dicho número se almacena en la variable $orientacion$.

```
int orientacion;
                    //con el string orientacion sabremos a donde esta mirando el robot
    if(direccion=="N"){

        flecha=30;          //Flecha arriba
        orientacion=1;      //Orientacion 1 norte
    }
    if(direccion=="S"){

        flecha=31;          //Flecha abajo
        orientacion=2;      //Orientacion 2 sur
    }
    if(direccion=="E"){

        flecha=16;          //Flecha derecha
        orientacion=3;      //Orientacion 3 este
    }
    if(direccion=="O"){

        flecha=17;          //Flecha izquierda
        orientacion=4;      //Orientacion 4 oeste
    }
```

Ilustración 4: asignación de valores y símbolos ASCII

Luego al leer la cantidad de movimientos (se almacena en la variable $movimientos$) se introduce un $\boldsymbol{for}\ (\boldsymbol{int\ o = 1; o \leq movimientos;\ o++})\{\ldots\}$ el cual va desde la primera orden hasta el número de

movimientos indicado por el usuario, dentro de esta función, dentro de esta estructura de control ejecutara los movimientos indicados en cado de (D) derecha e (I) izquierda simplemente se actualiza la posición a la que apunta la flecha, pero en la sentencia (A) adelante se tiene que cambiar la posición desde la que inicia el robot dentro de la matriz, para este caso hay 4 opciones que dependen de la orientación a la que está apuntando en ese momento, estas opciones son las siguientes:

1. ***Opción 1:*** Que apunte al **Norte**, en esta opción lo que se hace es **restar a la posiciónx** una unidad, ya que la flecha debería *subir a la parte superior* de la matriz.
2. ***Opción 2:*** Que apunte al **Sur**, en esta opción lo que se hace es **sumar a la posiciónx** una unidad, ya que la flecha debería *bajar a la parte inferior* de la matriz.
3. ***Opción 3:*** Que apunte al **Este**, en esta opción lo que se hace es **sumar a la posicióny** una unidad, ya que la flecha debería *desplazarse hacia la parte derecha* de la matriz.
4. ***Opción 4:*** Que apunte al **Oeste**, en esta opción lo que se hace es **restar a la posicióny** una unidad, ya que la flecha debería *desplazarse hacia la parte izquierda* de la matriz.

Después de esto se hace una verificación de que el robot no se encuentre en una de las filas o columnas limites tal que al avanzar hacia adelante se estrelle contra la pared, si esto no ocurre se piensa en lo siguiente: como la única manera de que el robot llegue a su destino o se estrelle contra una mina es cuando sufre un movimiento hacia adelante entonces a verificación se debe hacer en cada opción mostrada anteriormente, la manera de hacer esta verificación es asignando desde el principio un valor a la posición de llegada, y si en caso de que la posición que se revisa en este movimiento sea igual a ese valor dado simplemente se sale del **for** (haciendo **o**=100) y hace que la variable llamada $pfinalizado = 1$, en caso de que en el movimiento que se hizo el robot se encuentre con una mina se saldrá del **for** igual que como si pasara el caso anterior y hará que $pfinalizado = 2$. Si no entra en ninguno de los anteriores casos simplemente se actualiza la posición del robot e inicia a leer el siguiente movimiento que debe

hacer, así sucesivamente hasta que llegue al objetivo, se estrelle contra una mina o una pared, o se acabe la cantidad de movimientos.

Luego de esto se imprime una matriz de **char** indicando los movimientos hechos y se procede a emitir la salida de datos en el archivo de texto. Por medio de la función `fputs( "E", archivo);` , contenida en la librería `#include <fstream>` , que recibe los siguientes argumentos:

```
public int __cdecl fputs (const char * __restrict__ _Str, FILE * __restrict__ _File)
```

"La función **fputs( );** escribe una cadena en un fichero. la ejecución de la misma no añade el carácter de retorno de línea ni el carácter nulo final. El valor de retorno es un número no negativo o EOF en caso de error. Los parámetros de entrada son la cadena a escribir y un puntero a la estructura FILE del fichero donde se realizará la escritura." (Wikibooks, 2019). Tal como es indicado en el documento guía en caso de que el robot llegue al objetivo ($pfinalizado = 2$) se debe escribir la letra "C" (mediante la función **fputs ("C", archivo);**) después de dar un salto de línea con **fgets.** Algo muy similar ocurre en caso de que el robot choque con una mina ($pfinalizado = 1$) , en este caso se escribe el carácter "E" en el fichero. Con la misma función: **fputs ("E", archivo);.**

Finalmente se cierra el archivo de texto mediante la función **fclose (archivo);.** "Esta función cierra el fichero, cuyo puntero le indicamos como parámetro. Si el fichero se cierra con éxito devuelve 0" (Delgado, 2020). En este punto se vuelve a la función **main( )** y se pausa el sistema para por ultimo retornar el valor 0 y dar por terminado la ejecución del programa.

4. CONCLUSIÓN

Es posible concluir que el algoritmo y programa planteados logran dar solución al problema propuesto por el maestro, se evidencia también una solución fundamentada en los conocimientos adquiridos a lo largo del curso de Programación Básica el cual recordemos que es de vital importancia para un ingeniero de sistemas, ya que en este trabajo se demuestra manejo de distintos temas y como la unión de estos junto

con la elaboración de una estrategia de resolución ordenada y concreta logran el objetivo propuesto. En ambos casos tanto en el aspecto académico como personal. Mas que un trabajo fue un reto que se ha logrado superar.

Además de esto es notoria la necesidad de este curso para iniciar en el mundo de la programación e ir adquiriendo experiencia. También es necesario hacer notar la investigación llevada acabo para lograr el objetivo ya que como se dijo al principio en muchas ocasiones también depende del espíritu autodidacta y responsable de cada individuo, en este caso fue necesario apoyarse en distintas librerías e investigar sobre ellas, una muestra del trabajo autónomo y la intensa búsqueda de información que nos permita darle solución al problema. Luego de hacer diferentes pruebas con el código y ver que funciona en todos los casos nos damos cuenta de que el camino tomado fue el correcto.

5. ANEXOS

1. Partes de pseudocódigo en PSeInt.
2. Diagrama de flujo.

6. BIBLIOGRAFÍA

Cplusplus. (s.f.). Obtenido de C library: http://www.cplusplus.com/reference/clibrary/

Delgado, H. (21 de 11 de 2020). Obtenido de Ficheros - Abrir, cerrar, leer y agregar archivos de texto: https://disenowebakus.net/ficheros.php

fsymbols. (2020). Obtenido de ☺♥♪ Codigos Alt + Lista Todos Los Simbolos Del Teclado: https://fsymbols.com/es/teclado/windows/alt-codes/lista/atajos/

stack overflow. (2018). Obtenido de Write to a file using fputs in C: https://stackoverflow.com/questions/46152077/write-to-a-file-using-fputs-in-c/46152654

tutorialspoint. (s.f.). Obtenido de C library function - strtok(): https://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm

Universidad Nacional de la Plata. (2014). Obtenido de Resolución de problemas: http://weblidi.info.unlp.edu.ar/catedras/ingreso/Material2014/IAI/Cap1.pdf

Wikibooks. (4 de Noviembre de 2019). Obtenido de Programación en C/Manejo de archivos: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Manejo_de_archivos