Sistema de control para una red de sondas

Vitek Nicolás, Paez Martín.
Lenguajes Formales y Autómatas
Universidad Nacional Arturo Jauretche
nicolasvitek91@hotmail.com mpaez.ok@gmail.com

Resumen- Este informe comprende la realización de un proyecto final para la materia Lenguajes Formales y Autómatas, del cuarto año de la carrera Ingeniería Informática, de la Universidad Nacional Arturo Jauretche. Se trata de un sistema de control para una red de sondas que tiene como fin notificar niveles críticos de pH, conductividad, turbidez y temperatura, asociados al nivel de contaminación en arroyos.

El sistema debía contemplar multiplicidad de estados y acciones a tomar ante las diversas transiciones entre ellos. Por lo cual, se pedía implementar un autómata finito, para nuestro caso un autómata de Mealy, y, comprender el alcance de dicha solución.

Además, debido a que la comunicación entre nodos guarda estrecha relación con la distancia entre ellos, se requería analizar el problema para estudiar la utilidad de los grafos y el algoritmo de Floyd-Wharshall aplicados al caso de estudio.

I. Introducción

Buscando cumplir con el enunciado de la guía del Trabajo Final de la materia Lenguajes Formales y Autómatas, disponible en la sección "Referencias" del presente informe, se dividieron los objetivos del proyecto en dos partes.

La primera parte de este proyecto se centra en la implementación y análisis de los resultados obtenidos a partir del algoritmo de Floyd-Wharshall para calcular distancias mínimas entre nodos.

La segunda parte presenta el diseño, implementación y simulación en el software Proteus de un autómata finito escrito en el lenguaje C y compilado para un Arduino que, en función de los valores leídos de sensores, notifica, por distintos medios, determinados acontecimientos.

II. Entorno de trabajo

A. *Primera Parte, Matriz de Distancias Mínimas.*

Si bien la guía solicitaba desarrollar la aplicación en el lenguaje C, se decidió implementarla en C++ ya que consideramos enriquecedor investigar al respecto.

Se utilizó Visual Studio Code (VSC) con las extensiones "C/C++ for Visual Studio Code", "C/C++ Extension Pack" y "C/C++ Themes".
Además, para poder compilar en Windows desde VSC se instaló el software MinGw y la extensión de VSC "Code Runner".

B. *Segunda parte, Red de Sondas.*

Para la segunda mitad se empleó el lenguaje C, en el entorno de desarrollo "Arduino IDE", configurado para trabajar con "Arduino Uno". Además, fue necesario instalar las librerías "Dallas Temperature" y "OneWire".

Por último, para armar y simular el circuito, se empleó el software Proteus con un circuito provisto por la cátedra, que luego sería adaptado a nuestras necesidades.

III. Diseño e implementación de la primer parte

La guía plantea tres problemas puntuales, cuyas respuestas se hacen visibles en la línea de comandos al ejecutar el código.

El código está divido en dos archivos: "Matrix.cpp" y "Main.cpp". Además, se cuenta con la carpeta "aserciones" que contiene archivos de texto con datos empleados para testear la aplicación. Por último, junto a los archivos de texto se disponen imágenes con los grafos asociados a cada uno.

```
0 300 600 650 850
-2 0 400 430 530
-2 -2 0 200 250
-2 -2 -2 0 150
-2 -2 -2 -2 0
```

Fig 1. Contenido del archivo data.txt, que tiene los datos de una matriz de adyacencia. Se eligió el número -2 para indicar que dos nodos no están conectados en dicha dirección.

Se planteó un grafo no dirigido, ya que el enunciado no especificaba si ciertos nodos eran exclusivamente emisores o receptores. Con lo cual, asumimos que todos los nodos podían emitir y recibir. Por consiguiente, al esperarse una matriz simétrica se simplificó el análisis a los valores sobre la diagonal.

A. *Algoritmo de Floyd-Wharshall*

En la función "main()" del archivo "Main.cpp" se confecciona la matriz de adyacencias y se ejecuta el algoritmo de Floyd-Wharshal. Para lograr su cometido se vale del archivo "Matrix.cpp", el cual tiene la lógica que permite instanciar un objeto "Matrix" que modela una matriz de distancias mínimas. Su representación interna se basa en una matriz de enteros, cuyos valores son cargados desde un archivo de texto plano.

Contemplando al alcance del proyecto, se decidió agregar el algoritmo de Floyd-Wharshall como un método de instancia de la clase Matrix. Aunque, también hubiera sido válido implementarlo en una clase distinta o función, separando así las responsabilidades.

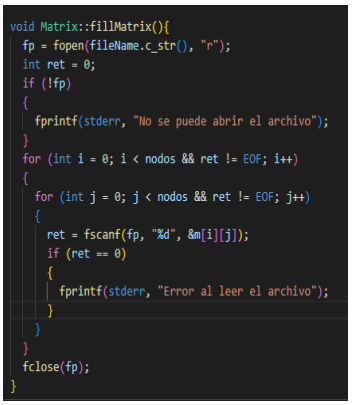

Tabla 1. Fig. 2. El método fillMatrix() de la clase Matrix toma los valores para la matriz de adyacencia de un archivo de texto plano y los guarda en una matriz de enteros.

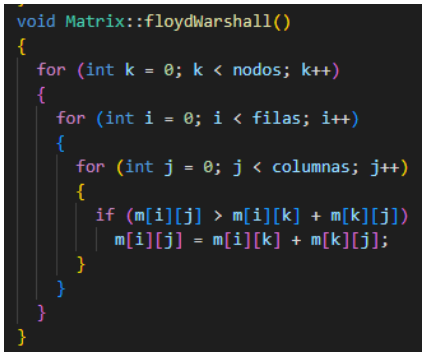

Fig. 3. El método floydWharshall() de la clase Matrix aplica el algoritmo del mismo nombre a la matriz guardada en el atributo “m”, la sobrescribe.

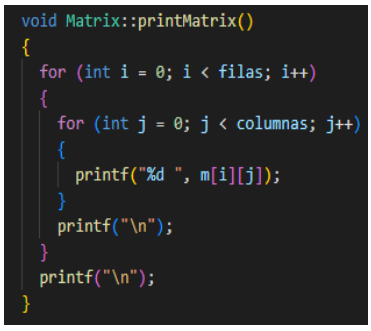

Fig. 4. El método printMatrix() de la clase Matrix toma la matriz del atributo “m” y la imprime por consola.

B. *Limitaciones del algoritmo de Floyd-Wharshall y análisis de un grafo*

En principio, no se usó el algoritmo de Floyd-Wharshall para determinar si se pueden realizar caminos con varias conexiones, todas menores a 500 metros, para transmitir información entre cualquier par de nodos, como solicita la guía del trabajo final.

Por dar un ejemplo, si tenemos que el camino mínimo esta formado por dos aristas de peso 400m cada una, esa seria una opción válida, sin embargo, la matriz devuelta por el algoritmo indicaría 800m, con lo cual, solo con esa información no se puede determinar si alguna de las dos aristas tiene un peso superior a 500m.

Para solucionar el problema confeccionamos un grafo y tras un análisis pudimos determinar que efectivamente, si es posible interconectar cualquier par de nodos respetando dicha condición.

A pesar de que el algoritmo no es suficiente, de todos modos, es una herramienta útil, ya que, si la matriz de salida indica un camino mínimo menor a 500m, podemos inferir que todas las aristas en él también tienen un peso inferior a 500m, con lo cual solo resta analizar aquellos pares de nodos que tienen asociado en la matriz un valor mayor a dicho umbral.

C. *Aserciones*

Se ejecutó una serie de aserciones para corroborar que el algoritmo funcionara de manera correcta. El código se encuentra en el archivo Main.cpp, en el cual se llama, desde la función “main()”, tres veces a la función “test(int)”, cada vez con parámetros diferentes.

La función “test(int)” lleva a cabo las aserciones del algoritmo, para lo cual emplea la información disponible en dos archivos. Por un lado, el primero de ellos contiene los datos de la matriz a la cual se le aplicará el algoritmo de Floyd-Wharshall; y, por otro lado, el segundo archivo contiene los datos esperados a la salida del algoritmo, si estos coinciden el test se considerará aprobado.

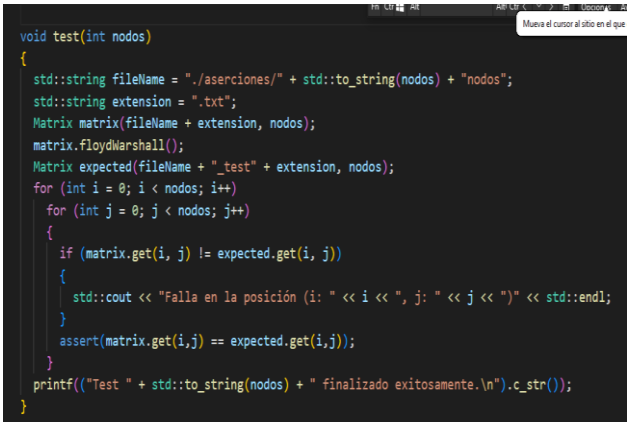

Fig. 5. Función test. Dado un valor del atributo de entrada, por ejemplo 3, se leerán los archivos “./aserciones/3nodos.txt” y “./aserciones/3nodos_test.txt”. El segundo de ellos tiene los valores esperados a la salida, mientras que el primero los valores de entrada del algoritmo.

IV. Diseño eh implementación de la segunda parte

Teniendo en cuenta el alcance del proyecto, se consideró que una implementación bajo el paradigma imperativo era suficiente. Además, siguiendo el código de ejemplo, y por el mismo motivo, se dispusieron todas las líneas dentro del mismo archivo base brindado por la cátedra.

En consecuencia, el código se encuentra en el archivo “nodo-lora-sensores.ino”, mientras que, por otro lado, el circuito está en el archivo “simulación kit arduino_2023.pdsprj”.

A. *Autómata finito*

En primer lugar, definimos una serie de números enteros que representan los distintos estados del autómata, de modo tal de que cuanto mayor es el número, más crítico es el estado.

```
#define HIGH_PH                 5
#define MEDIUM_TEMPERATURE      4
#define HIGH_TEMPERATURA        3
#define HIGH_TURBIDITY          2
#define HIGH_CONDUCTIVITY       1
#define NORMAL_VALUES           0
#define INITIAL_STATE          -1
```

Fig. 6 Estados del sistema

Cada pasaje de estado implica dos posibles acciones, una, para notificar el abandono de un estado (de ser necesario), y, la otra, para informar que el sistema se encuentra en un nuevo estado particular. Esto se da, ya sea porque se reestablecieron los valores normales, o bien, porque una variable de mayor criticidad entró en un rango de valores superior al umbral.

Para modelar este comportamiento se utilizaron dos máquinas de estado. La primera de ellas siempre se ejecuta, recibiendo el nuevo estado, al ser llamada por un método que detecta una medición por fuera del umbral. La segunda, recibe el viejo estado y solo se ejecuta si el nuevo estado es de menor criticidad.

Emplear estados y separar la máquina de estados que recibe el nuevo estado del grupo de condicionales que analizan los valores de las variables permite unificar el llamado al método “changeState()”. La ventaja es que el código es más limpio a causa de las responsabilidades mejor distribuidas.

```
void automata()
{
  int newState = NORMAL_VALUES;
  if(PH < 6 || PH > 9)
    newState = HIGH_PH;
  else if(temperatura > 25 && temperatura < 35)
    newState = MEDIUM_TEMPERATURE;
  else if(temperatura > 35)
    newState = HIGH_TEMPERATURA;
  else if(turbidez > 2000)
    newState = HIGH_TURBIDITY;
  else if(conductividad > 1500)
    newState = HIGH_CONDUCTIVITY;
  else
    newState = NORMAL_VALUES;
  if(newState != state)
    changeState(newState);
}
```

Fig. 7. La función autómata() es el método encargado de determinar el estado del sistema dados los valores medidos para el pH, la temperatura, la turbidez y la conductividad.

```
void changeState(int newState)
{
  if(newState < state)
    leaveCurrentState();
  state = newState;
  switch(state)
  {
    case HIGH_PH: onHighPh(); break;
    case MEDIUM_TEMPERATURE: onMediumTemperature(); break;
    case HIGH_TEMPERATURA: onHighTemperature(); break;
    case HIGH_TURBIDITY: onHighTurbidity(); break;
    case HIGH_CONDUCTIVITY: onHighConductivity(); break;
    default: break;
  }
}
```

Fig 8. La función changeState(newState) se encarga de actualizar el estado y ejecuta los métodos que corresponden al nuevo estado. Además, en caso de estarse pasando a un estado de menor criticidad, llama a la función encargada de informar el abandono del estado antiguo.

```
void leaveCurrentState()
{
  switch(state)
  {
    case HIGH_PH: onLowPh(); break;
    case MEDIUM_TEMPERATURE: onLowTemperature(); break;
    case HIGH_TEMPERATURA: onFromHighToMediumTemperature(); break;
    case HIGH_TURBIDITY: onLowTurbidity(); break;
    case HIGH_CONDUCTIVITY: onLowConductivity(); break;
    default: break;
  }
}
```

Fig. 9 La función leaveCurrentState() delega a quien corresponde la responsabilidad de informar el abandono de un estado particular.

B. *Limitaciones de un Autómata único*

Originalmente el problema parecía requerir un autómata por cada variable. Después de consultar con el docente se comprendió que el requerimiento de implementar un único autómata era necesario para que puedan evaluarse nuestros conocimientos.

Cuando los medios para informar al usuario son un recurso compartido, para nuestro caso un único led y un único parlante, este sistema es suficiente. Por el contrario, cuando el medio es dedicado, como es el caso de los mensajes MQTT, tenemos el inconveniente de que siempre se debe informar cuando se reestablecieron los valores normales para cierto tópico.

Imagínese que se informa por un tópico X que los valores superaran el umbral, luego si es necesario informar que una variable de mayor criticidad superó sus umbrales, esto se realiza en el tópico Y. Ahora, supongamos que se vuelve a repetir el mismo proceso y se publica en un tópico Z.

Planteado el escenario, el problema viene cuando se reestablecen los valores de la variable Y antes que los de la variable Z. Para el caso del recurso compartido no se deben generar cambios en la salida ya que estos deben seguir mostrando los que señalan a la variable de mayor criticidad. Sin embargo, sí que es necesario publicar un mensaje en el tópico Y, pero esto no es posible con el sistema planteado.

Más aún, continuando con el ejemplo, luego, se reestablecen los valores de la variable Z, por lo cual se avisa por el tópico Z y se informa el inconveniente para el nuevo estado por el tópico X (que aún tiene valores fuera del rango). Cuando esto se informa mediante el recurso compartido se obtiene el comportamiento esperado, ya que ahora los indicadores señalan a la variable X. Pero, cuando se publica por el medio dedicado, se obtiene una repetición incensaría de la información, ya que en dicho canal originalmente ya se había informado el problema para la variable X.

En conclusión, para lograr el comportamiento esperado, a la hora de solucionar este problema con medios dedicados, es necesario un seguimiento de cada variable por separado.

C. *Salidas*

Se implementó una serie de métodos para accionar cada periférico de salida, en nuestro caso prender o apagar el led y el sonido, así como publicar un tópico MQTT.

Respecto a uso de MQTT, se simuló el envío de mensajes imprimiéndolos por la consola Serie. Sin embargo, al quedar dicha responsabilidad en un método particular, implementar

la funcionalidad en un futuro no implicará modificar el código preexistente.

En el circuito se reemplazó el parlante por un led que se prende y apaga siguiendo la frecuencia que originalmente alternaba los tonos del sonido. Esta decisión se tomó debido a que el sonido resultaba muy invasivo en el entorno de desarrollo.

```
void ledOn()
{
  digitalWrite(LED_PIN, 1);
}

void ledOff()
{
  digitalWrite(LED_PIN, 0);
}
```

```
void speakerOn()
{
  tone(SPEAKER_PIN,SPEAKER_FRECUENCY);
}

void speakerOff()
{
  noTone(SPEAKER_PIN);
}
```

```
void sendMessage(String topic, String message)
{
  String request = "/" + topic + "/" + message;
  Serial.println(request);
}
```

Fig. 10 Funciones encargadas de accionar los periféricos de salida.

Cada salida implica varias acciones, con lo cual, por cada una de ellas se dispusieron métodos cuya responsabilidad es orquestar este comportamiento. Dichos métodos llaman a los mencionados anteriormente con cierta información y en el momento correspondiente.

Para cada estado hay dos posibles salidas (a excepción del estado Normal), una cuando se sale del estado debido a que se reestablecieron los valores normales, y, otra, cuando se entra en el estado debido a que las mediciones superaron el umbral respectivo y no hay una variable de mayor criticidad con valores anormales. Las funciones asociadas a dichas salidas se nombraron anteponiendo el prefijo “on”, comúnmente empleado para indicar que es un manejador de eventos. El nombre continúa con una palabra que hace alusión al valor, por ejemplo “High” cuando es un valor elevado. Y, finaliza, con el nombre de la variable, obteniendo resultados intuitivos tales como “onHighPh”, “onMediumTemperatura”, “onLowTurbidity”.

```
void onHighPh()
{
  speakerOn();
  ledOn();
  sendMessage(
    "alarmaPH",
    "Verter solucion acida/alcalina para regular el PH."
  );
}

void onLowPh()
{
  speakerOff();
  ledOff();
  sendMessage(
    "alarmaPH",
    "PH reestablecido a valores aceptables."
  );
}
```

Fig. 11 onHighPh() y onLowPh() son las funciones que se invocan cuando el pH cruza el umbra, ya sea en un sentido o en el otro. Ambos métodos interactúan con el led, el parlante y emiten un mensaje.

V. Resultados obtenidos

A. *Primera parte. Distancias mínimas*

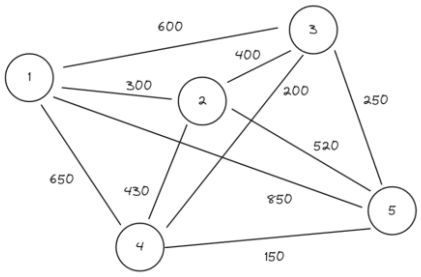

Fig. 12 Grafo formado por la ubicación de las sondas y sus distancias

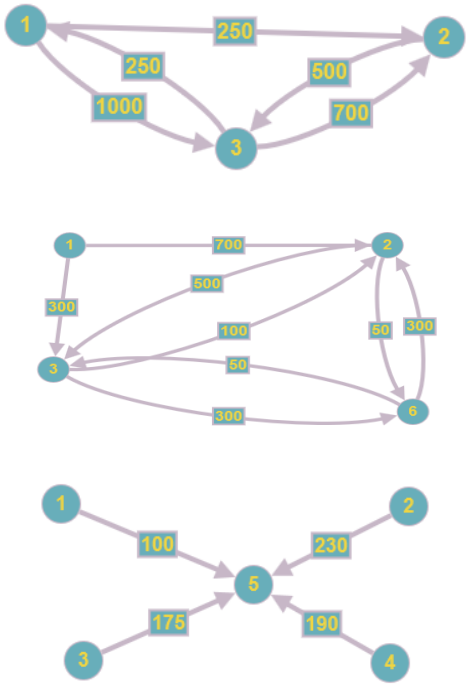

Fig 14 Grafos para las matrices de adyacencia de los test. Se utilizaron grafos dirigidos, ya que el algoritmo de Floyd-Wharshall que se estaba testeando debería funcionar correctamente con ellos.

```
Parte 1:

1) Realice la matriz de adyacencia y calcule la matriz de distancias minimas implementando el algoritmo de FLoyd en Lengu
aje C. La implementacion del algoritmo debe ser escalable a k nodos.

Respuesta:

Matriz adyacencia:
0 300 600 650 850
-2 0 400 430 530
-2 -2 0 200 250
-2 -2 -2 0 150
-2 -2 -2 -2 0

Matriz de distancias minimas:
0 300 600 650 800
-2 0 400 430 530
-2 -2 0 200 250
-2 -2 -2 0 150
-2 -2 -2 -2 0
```

Fig. 15 Salida por consola con las respuestas del primer ítem de la primera parte de la guía del trabajo final disponible en la sección “Referencias”.

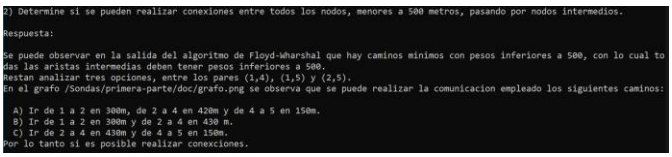

Fig. 16 Salida por consola con las respuestas del segundo ítem de la primera parte de la guía del trabajo final disponible en la sección “Referencias”.

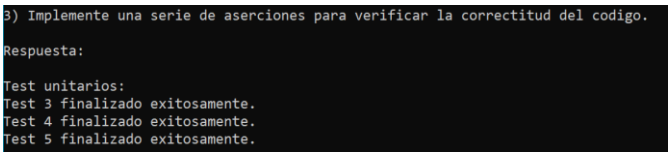

Fig. 17 Salida por consola con las respuestas del tercer ítem de la primera parte de la guía del trabajo final disponible en la sección “Referencias”.

B. Segunda parte, Red de Sondas.

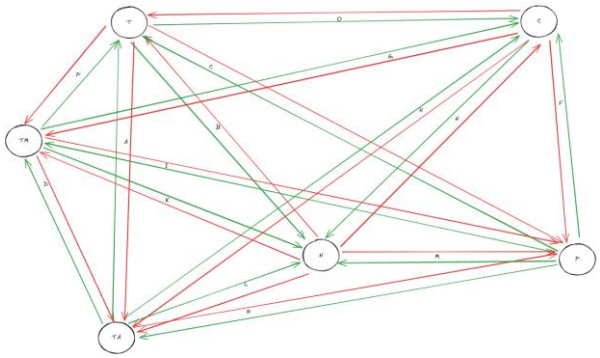

Fig. 16 grafo del autómata de Mealy. Las siglas en los nodos corresponden a la nomenclatura de la Tabla I.

Tabla I
ESTADOS

Abreviatura	Nombre	Descripción
P	PH alto	PH < 6 || PH > 9
TM	Temperatura media	TEMP > 25 && TEMP < 35
TA	Temperatura alta	TEMP > 35
T	Turbidez alta	TURB > 2000
C	Conductividad alta	COND > 1500
N	Normal	Ninguna de las anteriores

Tabla II
CONDICIONES

Nombre	Formula
C1	P & (TM | | !TM) & (TA | | !TA) & (T | | !T) & (C | | !C)
C2	!P & TM & (TA | | !TA) & (T | | !T) & (C | | !C)
C3	!P & !TM & TA & (T | | !T) & (C | !C)
C4	!P & !TM & !TA & T & (C | | !C)
C5	!P & !TM & !TA & !T & C
C6	!P & !TM & !TA & !T & !C

Tabla III
SALIDAS

Nombre	Descripción
SP	Sonido + Luz + Mensaje “Verter solución para regular el pH.”
STM	Mensaje “Reproducción de bacterias nocivas para la salud.”
STA	Luz + Mensaje “Prohibir descarga de agua”
ST	Mensaje “Extraer residuos”
SC	Mensaje “No consumir el agua”
SN	Mensaje “Todos los parámetros son correctos”

Tabla IV
AUTÓMATA DE MEALY

Estado	C1	C2	C3	C4	C5	C6
P	P/SP	TM/STM	TA/STA	T/ST	C/SC	N/SN
TM	P/SP	TM/STM	TA/STA	T/ST	C/SC	N/SN
TA	P/SP	TM/STM	TA/STA	T/ST	C/SC	N/SN
T	P/SP	-	TA/STA	T/ST	C/SC	N/SN
C	P/SP	-	TA/STA	T/ST	C/SC	N/SN
N	P/SP	-	TA/STA	T/ST	C/SC	N/SN

VI. CONCLUSIONES

Visual Studio Code requirió de una configuración poco intuitiva, ya que la búsqueda en internet para lograr compilar C++ presentaba multiplicidad de opciones. Por dar un ejemplo, la necesidad de instalar MinGw, en un principio parecía ser un camino arduo y poco probable de ser el más simple. De hecho, durante el proceso probamos con la instalación de otras consolas y el empleo de “pacman” por línea de comandos.

Además, tuvimos inconvenientes con el sensor de temperatura. Incluso la solución que encontramos, el menos en el simulador, no lee bien la temperatura si seleccionamos valores de delay cercanos a un segundo.

Por otra parte, la implementación del algoritmo de Floyd Wharshall es bastante simple; se obtiene anidando bucles; aunque el resultado es muy impresionante. Dicho algoritmo no reemplaza el uso de grafos, si no que automatiza un modo de resolver un problema particular. Incluso, a la hora de interpretar el contenido de la matriz, emplear grafos, resulta muy práctico.

Por último, un único autómata para todas las variables es una buena opción a la hora de resolver el problema, si este implica un medio compartido para mostrar la información y las variables están ordenadas por prioridad. Sin embargo, si hay medios dedicados para cada variable, la solución planteada no es suficiente, en dicho caso es necesario un seguimiento individual de cada una.

VII. AGRADECIMIENTOS

Agradecemos al Ing. Jorge Osio por la confianza y tutoría durante el desarrollo del Trabajo Práctico final y consecuente confección del presente informé. Sus métodos didácticos y actitud positiva, meritorias en su calidad como ingeniero y docente, nos motivaron a completar cada desafío que durante el proceso fuimos encontrando.

VIII. REFERENCIAS

[1] Ing. Jorge Osio, " Lenguajes formales y autómatas, características del laboratorio ", En la materia Lenguajes Formales y Autómatas, de la carrera Ingeniería en Informática, de la Universidad Nacional Arturo Jauretche, Octubre 2023 [online] Disponible: https://drive.google.com/file/d/1UqJXq5n4_pvroEx4WFYUxrML_AuSUBZw/view?usp=sharing.

[2] “Compilado C/C++ desde Visual Studio Code | Windows 10”, En el portal Platzi.com, Fecha de acceso: Noviembre 2023 [online]. Disponible: https://platzi.com/tutoriales/1189-algoritmos-2017/2765-compilando-cc-desde-visual-studio-code-windows-10/.

IX. Anexo

En esta sección se deja a disposición el repositorio donde se encuentra alojado el proyecto.

Repositorio de Github:
https://github.com/LFAutomatas/Sondas

Glosario

-Grafo: es una estructura matemática que consiste en un conjunto de vértices) y un conjunto de aristas que conectan pares de nodos. En un *grafo dirigido* las aristas tienen dirección, y, en un *grafo ponderado* las aristas tienen un valor asociado, llamado peso.

-Algoritmo de Floyd-Wharshall: es un algoritmo para encontrar los caminos más cortos en un grafo ponderado.

-Matriz de adyacencia: es una forma de representar un grafo mediante una matriz bidimensional. En esta matriz, las filas y columnas representan los nodos del grafo, y los valores indican la distancia entre nodos.

-Aserción: es una herramienta que evalúa una expresión booleana en un punto específico de un programa. Las aserciones se utilizan para detectar errores.

-Autómata finito: es un modelo matemático y computacional que representa un sistema con un número finito de estados y transiciones entre ellos en respuesta a una secuencia de entrada.

-Arduino: es una plataforma de hardware y software de código abierto que se utiliza para el desarrollo de prototipos y la creación de proyectos electrónicos.

-Proteus: es un software de diseño electrónico utilizado para el diseño, simulación y verificación de circuitos electrónicos.