



## **Instituto Tecnológico de Costa Rica**

Centro académico de Alajuela  
IC-4700. Lenguajes de programación  
Semestre I – 2023

### **Tarea Programada #3**

Prolog

Prof. María Mora Cross.

#### **Estudiantes:**

Brandon Retana Chacón.

Carné: 2021121141

Correo: [sthuar@estudiantec.cr](mailto:sthuar@estudiantec.cr)

Kevin Cubillo Chacón

Carné: 2021123138

Correo: [kevincubillo@estudiantec.cr](mailto:kevincubillo@estudiantec.cr)

Fecha de Entrega: 20/05/2023

<b>Descripción del problema:</b> .....	<b>3</b>
<b>Diseño del sistema:</b> .....	<b>4</b>
<b>Ejercicio 1:</b> .....	<b>4</b>
<b>Ejercicio 2:</b> .....	<b>7</b>
<b>Análisis de resultados:</b> .....	<b>8</b>
Objetivos alcanzados:.....	8
Objetivos no alcanzados:.....	10
<b>Instrucciones de uso de la aplicación:</b> .....	<b>10</b>
<b>Conclusión personal:</b> .....	<b>12</b>

## **Descripción del problema:**

### Ejercicio 1:

El problema planteado involucra a una empresa distribuidora de comida que utiliza un programa en Prolog para calcular la ruta más corta en un mapa que conecta 10 urbanizaciones. Esta empresa cobra una comisión por cada viaje, la cual se determina en función de los segmentos de ruta recorridos.

Para resolver este problema, se utiliza una tabla que muestra las distancias en kilómetros entre las diferentes urbanizaciones, así como las comisiones correspondientes a cada distancia recorrida. Esta información es crucial para calcular el costo total del viaje y determinar la ruta más corta que minimice tanto la distancia como la comisión.

### Ejercicio 2:

El problema abordado en este contexto está relacionado con los posibles fallos o problemas que pueden ocurrir en un sistema informático. El objetivo es diagnosticar y resolver estos problemas de manera efectiva y eficiente.

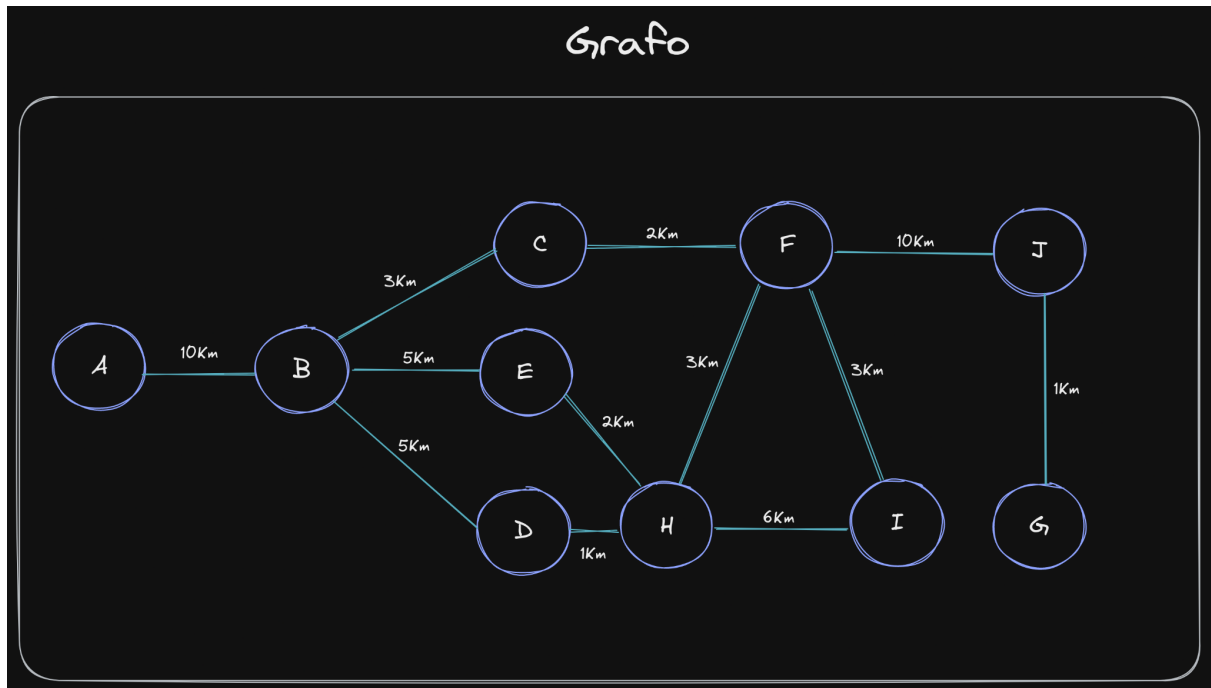
El código proporciona una serie de diagnósticos posibles y las acciones recomendadas asociadas a cada uno. Cada diagnóstico se identifica por un síntoma o problema específico, como demora del sistema, error de conexión, versión desactualizada, fallos en la instalación, memoria insuficiente, error en el registro, exceso de archivos temporales, problema de compatibilidad, errores en los logs y problema de red.

Para cada diagnóstico, se proporciona una lista de acciones que se recomienda realizar para solucionar el problema. Estas acciones pueden incluir reinicio del sistema, verificación de conexiones, limpieza de archivos temporales, actualización del software, reinstalación del software, verificación de recursos del sistema, análisis de logs, entre otras.

## Diseño del sistema:

### Ejercicio 1:

Para este ejercicio primero dibujamos el grafo para obtener una mejor comprensión del problema a resolver, el grafo contiene el nombre de los nodos y todas sus conexiones con la distancia en sus aristas:

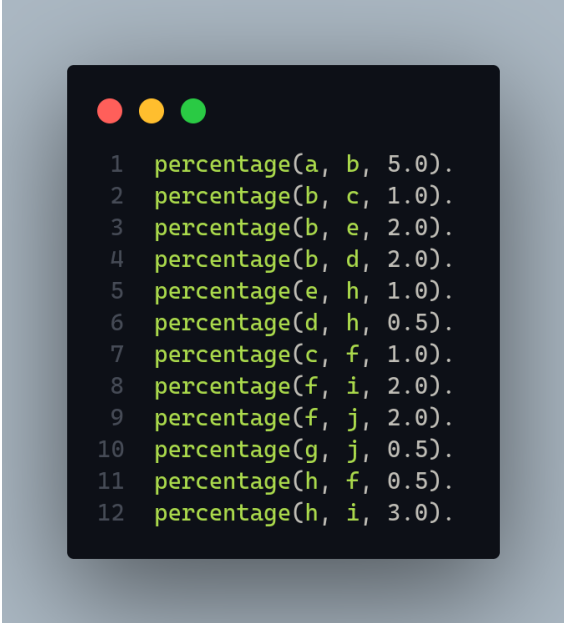


Para definir este grafo en el programa creamos los hechos en prolog, con las letras representativas de la ciudad, además del peso de su arista, esto lo hicimos de la siguiente manera:



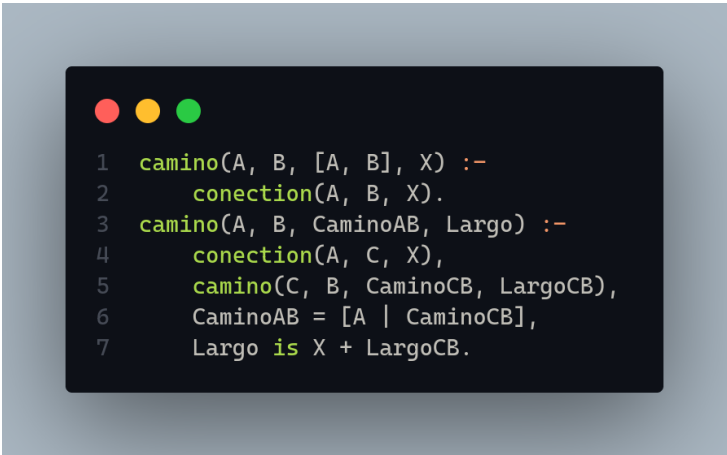
```
1  connection(a,b,10).
2  connection(b,c,3).
3  connection(b,e,5).
4  connection(b,d,5).
5  connection(e,h,2).
6  connection(d,h,1).
7  connection(c,f,2).
8  connection(f,i,3).
9  connection(f,j,10).
10 connection(g,j,1).
11 connection(h,f,3).
12 connection(h,i,6).
```

Para definir los porcentajes creamos otros hechos, los cuales usará el programa solo para recorrer los porcentajes y realizar la suma de los mismos, esta suma se usará más adelante para obtener el porcentaje de comisión que se obtuvo al hacer esta ruta.



```
1 percentage(a, b, 5.0).
2 percentage(b, c, 1.0).
3 percentage(b, e, 2.0).
4 percentage(b, d, 2.0).
5 percentage(e, h, 1.0).
6 percentage(d, h, 0.5).
7 percentage(c, f, 1.0).
8 percentage(f, i, 2.0).
9 percentage(f, j, 2.0).
10 percentage(g, j, 0.5).
11 percentage(h, f, 0.5).
12 percentage(h, i, 3.0).
```

Para encontrar los caminos entre dos nodos, se definió una función “camino” la cual recorre todos los nodos y encuentra en los hechos los caminos que hacen match:



```
1 camino(A, B, [A, B], X) :-
2     conection(A, B, X).
3 camino(A, B, CaminoAB, Largo) :-
4     conection(A, C, X),
5     camino(C, B, CaminoCB, LargoCB),
6     CaminoAB = [A | CaminoCB],
7     Largo is X + LargoCB.
```

Para encontrar la ruta más corta entre todos los caminos que se encontraron anteriormente, se utiliza la siguiente función, esta función utilizando algunas funciones predefinidas de prolog como el findall o sort, encuentra todos los caminos posibles llamando a camino, después de esto las ordena y obtiene el mejor resultado posible, además llama a la función printPath y printpercentage para imprimir el camino y el porcentaje de comisión.

```

1  shortestPath(A, B) :-
2      findall(p(Costo, Camino),
3              camino(A, B, Camino, Costo),
4              Caminos),
5      sort(Caminos, Lista),
6      Lista = [p(Costo1, Camino1) | _],
7      printPath(Camino1),
8      printpercentage(Camino1, Comision1),
9      writef(' (con distancia de %d km y una comision de %d%)\\n', [Costo1, Comision1]).

```

printPath:

```

1  printPath([]).
2  printPath([X]) :-
3      !, write(X).
4  printPath([X|T]) :-
5      write(X),
6      write(', '),
7      printPath(T).

```

printpercentage:

```

1  printpercentage([], 0).
2  printpercentage([X|T], Y) :-
3      car(T, Cabeza),
4      percentage(X, Cabeza, Comision),
5      printpercentage(T, C),
6      Y is Comision + C.

```

## Ejercicio 2:

La base de conocimiento desarrollada es un ejemplo básico para el soporte técnico de software. Contiene hechos y reglas que definen diferentes problemas de diagnóstico y las posibles soluciones asociadas a cada problema.

En la base de conocimiento, los problemas de diagnóstico se definen mediante el predicado “*diagnostico*”, donde el primer argumento es el nombre del problema y el segundo argumento es una lista de acciones recomendadas para resolver ese problema.

```
% Resultados de diagnostico
diagnostico(demora_del_sistema, [reinicio_del_sistema, verificacion_de_conexiones, limpieza_de_archivos_temporales]).
diagnostico(error_de_conexion, [verificacion_de_conexiones, reinicio_del_sistema]).
diagnostico(version_desactualizada, [actualizacion_del_software, reinicio_del_sistema]).
diagnostico(fallos_en_la_instalacion, [reinstalacion_del_software, verificacion_de_recursos_del_sistema]).
diagnostico(memoria_insuficiente, [verificacion_de_recursos_del_sistema]).
diagnostico(error_en_el_registro, [reinstalacion_del_software]).
diagnostico(exceso_de_archivos_temporales, [limpieza_de_archivos_temporales]).
diagnostico(problema_de_compatibilidad, [actualizacion_del_software, reinicio_del_sistema]).
diagnostico(errores_en_logs, [analisis_de_logs, verificacion_de_conexiones]).
diagnostico(problema_de_red, [verificacion_de_conexiones, reinicio_del_sistema]).
```

La regla “*soluciones\_para\_problemas*” toma una lista de problemas y devuelve una lista de soluciones asociadas a esos problemas. Esta regla se implementa de manera recursiva, tomando el primer problema de la lista y buscando sus acciones recomendadas en la base de conocimiento. Luego, genera una cadena de texto formateada con el diagnóstico y las acciones recomendadas. La recursión continúa con el resto de los problemas hasta que la lista esté vacía.

```
% Reglas
soluciones_para_problemas([], []).
soluciones_para_problemas([Problema | Resto], [Solucion | SolucionesResto]) :-
    diagnostico(Problema, Acciones),
    Diagnostico = Problema, % En este caso, asumimos que el diagnostico es el mismo que el problema
    format(atom(Solucion), "\nPosible diagnostico: ~w\nAcciones recomendadas:\n~w\n", [Diagnostico, Acciones]),
    soluciones_para_problemas(Resto, SolucionesResto).
```

El código en Java interactúa con la base de conocimiento de Prolog utilizando la biblioteca JPL (Java-Prolog Bridge). El programa Java permite al usuario seleccionar problemas de un menú interactivo. Luego, utiliza la consulta “*soluciones\_para\_problemas*” para obtener las soluciones recomendadas para los problemas seleccionados.

La función “*seleccionarProblemas()*” permite al usuario seleccionar los problemas ingresando los números correspondientes separados por comas. Después de seleccionar los problemas, se llama a la función “*obtenerSoluciones()*” para realizar la consulta a la base de conocimiento de Prolog y obtener las soluciones recomendadas. Finalmente, las soluciones se imprimen en la consola de Java.

El mecanismo de consulta en Java utiliza la clase Query de JPL para realizar la consulta en Prolog. La consulta se construye como una cadena de texto, donde se llama al predicado “soluciones\_para\_problemas” con la lista de problemas seleccionados como argumento. Se verifica si hay una solución para la consulta y, si es así, se obtienen todas las soluciones disponibles. Luego, se formatea la cadena de texto de las soluciones y se imprime en la consola de Java.

```
// Función para seleccionar los problemas del menú
private static List<String> seleccionarProblemas() {
    List<String> problemasSeleccionados = new ArrayList<>();
    Scanner scanner = new Scanner(System.in);

    System.out.println(x:"Seleccione los problemas que tiene (ingrese el número correspondiente):");
    System.out.println(x:"1. Demora del sistema");
    System.out.println(x:"2. Error de conexión");
    System.out.println(x:"3. Versión desactualizada");
    System.out.println(x:"4. Fallos en la instalación");
    System.out.println(x:"5. Memoria insuficiente");
    System.out.println(x:"6. Error en el registro");
    System.out.println(x:"7. Exceso de archivos temporales");
    System.out.println(x:"8. Problema de compatibilidad");
    System.out.println(x:"9. Errores en logs");
    System.out.println(x:"10. Problema de red");

    System.out.print(s:"\nDigite las opciones separadas por coma: ");
    String input = scanner.nextLine().trim();
}
```

```
// Función para obtener las soluciones recomendadas para los problemas seleccionados
private static void obtenerSoluciones(List<String> problemas) {
    Query query = new Query("soluciones_para_problemas(" + problemas.toString() + ", Soluciones)");
    if (query.hasSolution()) {
        java.util.Map<String, Term>[] solutions = query.allSolutions();

        String soluciones = solutions[0].get("Soluciones").toString().replace("[", "").
            replace("]", "").replace("'", "").replace(", ", "\n");

        System.out.println(soluciones);
    } else { System.out.println(x:"No se encontraron soluciones para los problemas seleccionados");}
}
```

## **Análisis de resultados:**

### Objetivos alcanzados:

#### Ejercicio 1:

- Se ha diseñado el grafo de conexión entre las urbanizaciones, representando el mapa de distribución de la empresa de comida.



- Se han establecido los hechos y reglas necesarios en Prolog para calcular la ruta más corta entre un punto de origen y un punto de entrega, teniendo en cuenta la distancia en kilómetros.
- Se han definido los hechos y reglas en Prolog para determinar la comisión correspondiente por la entrega, considerando los porcentajes de comisión de la tabla de distancias.
- Se ha implementado un sistema en Prolog capaz de responder consultas del tipo: "Dado un punto de origen y un punto de entrega, ¿cuál es la ruta más corta y la comisión correspondiente por la entrega?".
- El sistema de Prolog es capaz de manejar rutas de forma bidireccional, considerando tanto nodos adyacentes como no adyacentes para calcular la distancia y la comisión total.
- Se ha documentado adecuadamente el sistema de Prolog, incluyendo la descripción de los hechos, reglas y consultas implementadas, así como el proceso de cálculo de la ruta más corta y la comisión correspondiente.

## Ejercicio 2:

- Se ha diseñado una base de conocimiento en Prolog que contiene información relevante para el soporte técnico, incluyendo la resolución de problemas técnicos y respuestas a preguntas frecuentes.
- Se ha implementado la base de conocimiento en Prolog, asegurando su correcta funcionalidad y capacidad para responder consultas relacionadas con el diagnóstico.
- Se ha creado un modelo de base de conocimiento debidamente documentado, describiendo los predicados, hechos y reglas utilizados, así como su propósito y funcionamiento.
- Se ha incluido al menos una consulta recursiva en la base de conocimiento, permitiendo la exploración y búsqueda de información de manera eficiente.
- Se han utilizado listas en al menos uno de los predicados de la base de conocimiento, mejorando la capacidad de almacenamiento y manipulación de datos.
- Se han definido al menos 7 atributos relevantes para el diagnóstico y se han establecido al menos 10 posibles resultados de diagnóstico.

- Se ha desarrollado un mecanismo de consulta utilizando la interfaz JPL en Java, permitiendo la interacción del usuario con el sistema de diagnóstico.

#### Objetivos no alcanzados:

- Ninguno

#### **Instrucciones de uso de la aplicación:**

##### Ejercicio 1:

1. Ejecute el programa Prolog en su entorno de desarrollo o desde la línea de comandos.
2. Para encontrar la ruta más corta y la comisión correspondiente, llame a la función caminoMasCorto(A, B) pasando como argumentos los nombres de los puntos de origen (A) y de entrega (B).
3. El programa calculará la ruta más corta y la comisión correspondiente entre los dos puntos de entrega.
4. El resultado se mostrará en la consola de Prolog, indicando la ruta más corta (los nodos visitados en orden) y la comisión total.
5. Repita los pasos 2-4 para realizar consultas adicionales con diferentes puntos de origen y entrega.

```
?- caminoMasCorto(A, B).  
d, h (con distancia de 1 km y una comision de 0.5%)  
true .  
  
?- caminoMasCorto(C, F).  
d, h (con distancia de 1 km y una comision de 0.5%)  
true
```

## Ejercicio 2:

1. Compilar y ejecutar el programa Java en su entorno de desarrollo o desde la línea de comandos.
2. Se mostrará un menú interactivo con una lista de problemas disponibles. Cada problema está numerado.
3. Seleccione los problemas con los que necesita ayuda ingresando los números correspondientes separados por comas. Por ejemplo, si desea seleccionar los problemas 1, 3 y 5, debe ingresar "1,3,5". Presione Enter después de ingresar los números.
4. El programa procesará su selección y mostrará las soluciones recomendadas para los problemas seleccionados.
5. Las soluciones se mostrarán en la consola de Java, cada una con un diagnóstico y una lista de acciones recomendadas para resolver el problema.
6. Si desea realizar consultas adicionales, repita los pasos 3-5.

```
Seleccione los problemas que tiene (ingrese el número correspondiente):
```

```
1. Demora del sistema
2. Error de conexión
3. Versión desactualizada
4. Fallos en la instalación
5. Memoria insuficiente
6. Error en el registro
7. Exceso de archivos temporales
8. Problema de compatibilidad
9. Errores en logs
10. Problema de red
```

```
Digite las opciones separadas por coma: 1,2,3
```

```
Posible diagnostico: demora_del_sistema
```

```
Acciones recomendadas:
```

```
reinicio_del_sistema
verificacion_de_conexiones
limpieza_de_archivos_temporales
```

```
Posible diagnostico: error_de_conexion
```

```
Acciones recomendadas:
```

```
verificacion_de_conexiones
reinicio_del_sistema
```

```
Posible diagnostico: version_desactualizada
```

```
Acciones recomendadas:
```

```
actualizacion_del_software
reinicio_del_sistema
```

## **Conclusión personal:**

### Kevin:

Con el diseño y programación de este proyecto pude aprender la importancia de la programación lógica en la creación de bases de conocimiento, las cuales permiten realizar consultas de forma eficiente y brindar resultados precisos. Del mismo modo que aprendí usar este conocimiento desde un programa alterno hecho en Java a través de una interfaz específica.

### Brandon:

De este proyecto puedo concluir que Prolog puede ser muy útil para tener bases de conocimiento, lo que facilita la creación de reglas lógicas para hacer diagnósticos rápidos y no tener que crear una serie if anidados, como se haría en otros lenguajes como python, java o c++.