



Instituto Tecnológico de Costa Rica

Proyecto 3

Programación Lógica

Jesus Araya Chaves 2021106062

Fabrizio Rios Montero 2021473111

Curso: Lenguajes de Programación

Número de grupo: 20

Profesor: Maria Auxiliadora Mora

Primer semestre del 2023

<b>Descripción del Problema</b>	<b>3</b>
<b>Diseño del Sistema</b>	<b>4</b>
Ejercicio 1.	4
Ejercicio 2.	5
<b>Análisis de Resultado</b>	<b>7</b>
<b>Ejecución del programa</b>	<b>8</b>
Problema 1	8
Problema 2	8
<b>Conclusiones Personales</b>	<b>10</b>

# Descripción del Problema

Para este trabajo, se plantean dos problemas a resolver utilizando principalmente el lenguaje de programación prolog, mientras que, uno de ellos pide utilizar la interfaz JPL para la conexión a la base de conocimientos.

El primer problema planteado consiste en modelar una red de urbanizaciones conectadas por caminos (en este caso, los caminos serían las aristas del grafo) en el lenguaje de programación Prolog. Cada urbanización (nodo) está conectada a otra por un camino que tiene una distancia en kilómetros (valor de la arista) y una comisión que la empresa cobra por recorrer ese camino.

El objetivo es encontrar el camino más corto entre dos nodos, así mismo, poder calcular el costo total de la comisión basado en el camino seleccionado. Para este problema hay que tener en cuenta que debemos crear el grafo de manera bidireccional, así los caminos pueden ser recorridos en cualquier dirección.

Para el segundo problema, se nos presenta un problema el cual tiene como objetivo diseñar y desarrollar un sistema de diagnóstico basado en una base de conocimientos en Prolog. El sistema tiene como objetivo asistir a los técnicos de soporte en la resolución de problema técnicos y responder a las preguntas frecuentes que puedan surgir en su trabajo diario. Se pide que la interfaz de consulta del sistema se implemente en java, utilizando la interfaz JPL para interactuar con la base de conocimientos de Prolog.

Este problema tiene algunos requisitos como:

1. Consulta recursiva.
2. Uso al menos una lista.
3. El sistema debe contemplar al menos 7 atributos diferentes.
4. El sistema debe ser capaz de producir al menos 10 resultados de diagnóstico diferentes.

# Diseño del Sistema

## Ejercicio 1.

El sistema implementa el algoritmo de Dijkstra para buscar el camino con la menor comisión entre dos puntos en un grafo. El algoritmo inicia con el punto inicial y explora sucesivamente los puntos adyacentes que aún no ha visitado, acumulando la comisión total. Una vez que se encuentra el punto final o se han visitado todos los puntos, el algoritmo se detiene y devuelve el camino encontrado y la comisión total.

Grafo del sistema (Base de conocimientos):

El grafo del sistema es un grafo dirigido bidireccional. Los puntos (o nodos) se representan por letras (a, b, c, d, e, f, g, h, i, j) y las rutas (o aristas) se definen con el predicado ruta(Origen, Destino, Distancia, Comision).

A continuación, se muestra una representación textual del grafo según las reglas dadas en la base de conocimientos:

```
a -10-> b -3-> c -2-> f -3-> i
|      |      |      |
|5     |1     |2     |3
V      V      V      V
b -5-> e -2-> h -3-> f -10-> j
|      |      |      |
|2     |1     |1     |5
V      V      V      V
b -5-> d -1-> h -6-> i
|      |      |
|2     |0.5   |3
V      V      V
g -1-> j
```

Donde las flechas representan rutas entre puntos, los números sobre las líneas representan distancias y los números debajo de las líneas representan comisiones.

Mecanismo de navegación:

El mecanismo de navegación se basa en el algoritmo de Dijkstra. Este algoritmo es un algoritmo para encontrar el camino más corto entre puntos en un grafo. En tu implementación, se utiliza una versión simplificada y modificada de Dijkstra que funciona de la siguiente manera:

Inicia con un punto inicial (I), un punto final (F), una lista de puntos visitados (S), una lista de puntos no visitados (Unvisited), una comisión parcial (ComisionParcial), una comisión acumulada (ComisionAcumulada) y una lista que representa el camino (L).

Busca rutas desde el punto actual a otros puntos que no han sido visitados aún.

Si el punto adyacente no es el punto final y no ha sido visitado, lo agrega a la lista de visitados, lo elimina de la lista de puntos no visitados y suma la comisión de la ruta a la comisión total acumulada. Luego, llama recursivamente al mismo predicado con el nuevo punto, las listas actualizadas y la comisión acumulada.

Si el punto adyacente es el punto final, lo agrega a la lista de visitados, suma la comisión de la ruta a la comisión acumulada y termina devolviendo la lista de visitados (que representa el camino) y la comisión total.

Este proceso continúa hasta que se ha visitado cada punto en el grafo, y se ha encontrado la ruta con el menor costo entre el punto inicial y el punto final.

## Ejercicio 2.

Base de conocimiento

La base de conocimiento está representada por los hechos en Prolog, que están definidos por los predicados atributo y diagnóstico.

El atributo representa los distintos componentes del sistema y sus posibles estados problemáticos. Cada hecho de atributo se compone de un componente (como computadora o impresora) y una lista de posibles estados problemáticos (como lenta o sin\_encender).

El diagnóstico representa los problemas y soluciones posibles en el sistema. Cada hecho de diagnóstico consta de un nombre de problema, una lista de estados problemáticos y una solución sugerida.

Mecanismo de consulta

El mecanismo de consulta está representado por las reglas en Prolog y la interfaz de usuario en Java.

Las reglas en Prolog, consulta y diagnosticar, se utilizan para buscar en la base de conocimientos y devolver diagnósticos o atributos en base a la consulta del usuario. consulta verifica si una lista es sublista de otra, y diagnosticar busca un diagnóstico que coincida con los atributos proporcionados.

La interfaz de usuario en Java permite al usuario interactuar con el sistema. El usuario puede seleccionar si desea diagnosticar un problema o ejecutar una consulta personalizada. La interfaz luego formatea la entrada del usuario en una consulta Prolog, la ejecuta y muestra los resultados.

La entrada del usuario se obtiene a través de una línea de comandos donde el usuario puede especificar los atributos problemáticos del componente que está diagnosticando. Luego, estos atributos se pasan a la consulta Prolog para obtener el diagnóstico correspondiente.

La consulta se forma como una cadena de texto que contiene el predicado diagnosticar y la lista de atributos proporcionados por el usuario. Esta consulta se pasa a la interfaz de Java-to-Prolog para ejecutarla y recoger los resultados.

En conclusión, el diseño de la base de conocimientos y del mecanismo de consulta permite al usuario diagnosticar problemas en su sistema proporcionando los atributos problemáticos de los componentes y obtener una solución sugerida.

# Análisis de Resultado

La conclusión de estos ejercicios fue sumamente positiva, con todos los objetivos planteados alcanzados de manera exitosa.

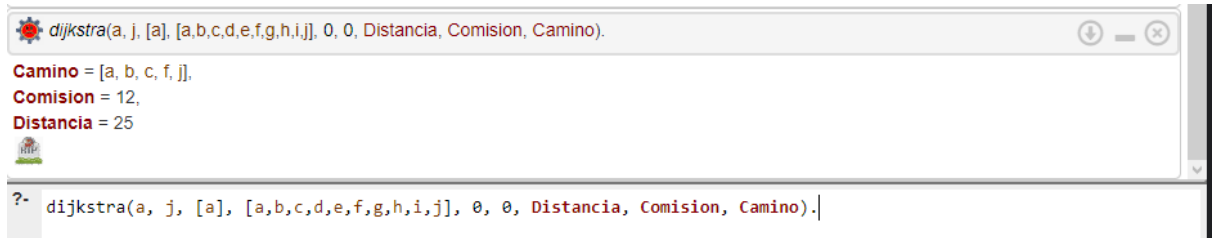
En el primer ejercicio, se creó un sistema efectivo en Prolog que determina la ruta más corta en un mapa que conecta varias urbanizaciones y calcula la comisión correspondiente a la ruta seleccionada. El sistema es capaz de recibir un punto de origen y un punto de entrega, y devuelve la ruta más corta y la comisión correspondiente por la entrega. Al considerar que las rutas son bidireccionales y al calcular las comisiones para nodos adyacentes y no adyacentes, el sistema se mostró robusto y versátil para manejar diferentes escenarios de planificación de rutas.

En el segundo ejercicio, se logró desarrollar un sistema de diagnóstico utilizando Prolog. Este sistema, alimentado por una base de conocimientos diseñada para asistir a los técnicos de soporte a resolver problemas técnicos, demostró ser capaz de responder a diversas consultas de diagnóstico. El sistema implementa todos los requisitos solicitados, incluyendo la implementación de al menos una consulta recursiva, la utilización de listas en al menos uno de los predicados, la contemplación de al menos 7 atributos y al menos 10 resultados de diagnóstico posibles. Adicionalmente, se logró la integración exitosa de este sistema de diagnóstico con Java a través de la interfaz JPL, lo que permitió una consulta interactiva y amigable con el usuario.

# Ejecución del programa

## Problema 1

1. Para ejecutar este problema debemos ejecutar el código con la entrada que se especifica en el código.

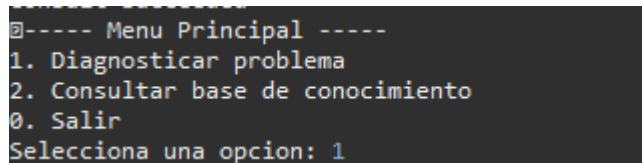


```
dijkstra(a, j, [a], [a,b,c,d,e,f,g,h,i,j], 0, 0, Distancia, Comision, Camino).  
Camino = [a, b, c, f, j].  
Comision = 12,  
Distancia = 25  
?- dijkstra(a, j, [a], [a,b,c,d,e,f,g,h,i,j], 0, 0, Distancia, Comision, Camino).|
```

2. Podemos verificar la ruta más corta, con la comisión y la distancia.

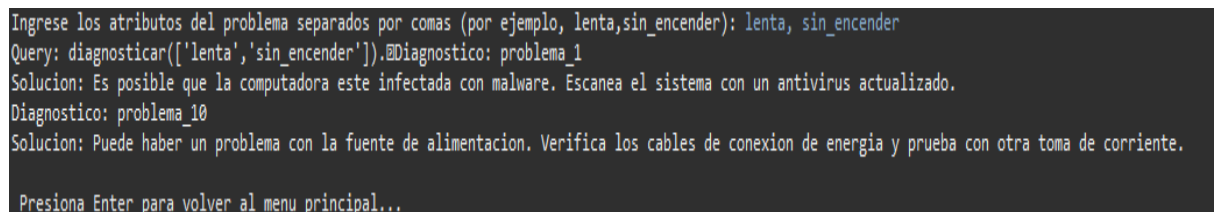
## Problema 2

1. Para este problema debemos ejecutar el código y veremos la interfaz.
2. Seleccionamos la primera opción digitando “1” en la consola.



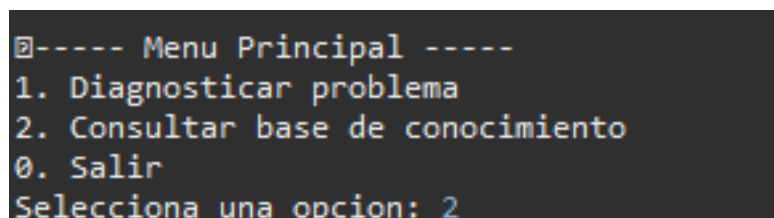
```
@----- Menu Principal -----  
1. Diagnosticar problema  
2. Consultar base de conocimiento  
0. Salir  
Selecciona una opcion: 1
```

3. Ahora podemos observar que la interfaz nos pide ingresar los atributos de nuestro problema separados por coma.
4. Ingresamos nuestros problemas, en este caso podemos ingresar “lenta, sin\_encender”.



```
Ingrese los atributos del problema separados por comas (por ejemplo, lenta,sin_encender): lenta, sin_encender  
Query: diagnosticar(['lenta','sin_encender']).@Diagnostico: problema_1  
Solucion: Es posible que la computadora este infectada con malware. Escanea el sistema con un antivirus actualizado.  
Diagnostico: problema_10  
Solucion: Puede haber un problema con la fuente de alimentacion. Verifica los cables de conexion de energia y prueba con otra toma de corriente.  
Presiona Enter para volver al menu principal...
```

5. Podemos ver que nos da las soluciones que están en la base de conocimiento.
6. Presionamos la tecla enter para volver al menú principal y digitar “2” para consultar la base de conocimientos.



```
@----- Menu Principal -----  
1. Diagnosticar problema  
2. Consultar base de conocimiento  
0. Salir  
Selecciona una opcion: 2
```

7. Copiamos uno de los ejemplos



8. Observamos los resultados de nuestra base de conocimientos.

```
Ingrese la consulta en Prolog: diagnostico(Problema, Atributos, Solucion), member(Atributo, Atributos).
@Solucion = 'Es posible que la computadora este infectada con malware. Escanea el sistema con un antivirus actualizado.'
Atributo = lenta
Problema = problema_1
Atributos = [lenta, sin_encender]
Solucion = 'Es posible que la computadora este infectada con malware. Escanea el sistema con un antivirus actualizado.'
Atributo = sin_encender
Problema = problema_1
Atributos = [lenta, sin_encender]
Solucion = 'Puede haber un problema con el adaptador de red. Verifica las conexiones y reinicia el enrutador si es necesario.'
Atributo = lenta
```

# Conclusiones Personales

## Conclusión Jesús:

Me gustó mucho el trabajar en este proyecto, especialmente en la parte de adaptar el algoritmo de Dijkstra al lenguaje de programación Prolog, ya que siento que fue un reto muy interesante y ayuda bastante a aprender acerca del lenguaje, también, aprendí mucho en lo que respecta a la utilización de la interfaz JPL para conectar prolog con java, y así poder observar como los dos lenguajes combinados pueden llegar a crear aplicaciones robustas y poderosas.

## Conclusión Fabricio:

La integración de Prolog con Java a través de la interfaz JPL fue la parte que más me gustó. Esta tarea me permitió experimentar cómo diferentes lenguajes de programación pueden interactuar y complementarse para generar soluciones más sólidas y completas. Trabajar con Prolog y Java me permitió comprender la importancia de la interacción entre diferentes paradigmas de programación en proyectos de desarrollo de software, y cómo estos pueden mejorar la eficiencia y la escalabilidad de las aplicaciones.