Sistema Experto para Diagnóstico de Otitis en Prolog

📋 Índice

🎯 Objetivo del Sistema

Este sistema experto implementa un **motor de inferencia basado en reglas** para diagnosticar **otitis media aguda** a partir de síntomas iniciales. Utiliza:

- **Búsqueda en grafos** (BFS y DFS) para explorar posibles evoluciones de síntomas
- **Modo interactivo** que pregunta al usuario sobre la presencia de síntomas
- **Evaluación probabilística** basada en pesos asignados a cada síntoma
- **Visualización del proceso** mostrando cada paso de la búsqueda

📊 Resumen Ejecutivo

¿Qué hace el sistema?

El sistema modela la **evolución de síntomas** desde manifestaciones iniciales (dolor de oído, zumbido) hasta el diagnóstico final de **otitis**, pasando por síntomas intermedios y graves.

Características principales

- ✅ Búsqueda automática (BFS/DFS) desde un síntoma inicial
- ✅ Modo interactivo que guía al usuario pregunta por pregunta
- ✅ Cálculo de probabilidad del diagnóstico basado en pesos
- ✅ Visualización paso a paso del proceso de inferencia
- ✅ Casos de prueba predefinidos para demostración

Archivos del proyecto

```
L-gica-de-predicados---PROLOG/
├── agente_otitis.pl       # Sistema experto en Prolog (motor de inferencia)
├── agente_otitis.py       # Implementación en Python (algoritmos BFS/DFS)
├── app.py                 # Interfaz gráfica Tkinter
├── representar.html       # Visualización del grafo de síntomas
├── requirements.txt       # Dependencias Python
└── DOCUMENTACION_COMPLETA.md  # Este archivo
```

🔧 Requisitos y Configuración

Software necesario

1. **SWI-Prolog** (versión 8.0 o superior)
   - Descargar desde: https://www.swi-prolog.org/Download.html
   - Instalación en Windows: ejecutar el instalador `.exe`
   - Verificar instalación: abrir terminal y ejecutar `swipl --version`
2. **Python 3.8+** (opcional, solo para interfaz gráfica)
   - Tkinter, matplotlib, networkx (ver `requirements.txt`)

Instalación y ejecución

Opción 1: Solo Prolog (línea de comandos)

```
# 1. Navegar al directorio del proyecto
cd "D:\L-gica-de-predicados---PROLOG"

# 2. Iniciar SWI-Prolog cargando el archivo
swipl -s agente_otitis.pl
```

Al cargar, verás el menú de ayuda automáticamente.

Opción 2: Interfaz gráfica Python

```
# 1. Instalar dependencias
pip install -r requirements.txt

# 2. Ejecutar la interfaz gráfica
python app.py
```

📚 Introducción a Prolog

¿Qué es Prolog?

Prolog (Programming in Logic) es un lenguaje de programación **declarativo** basado en:

- **Lógica de predicados de primer orden**
- **Resolución por unificación y backtracking**
- **Paradigma declarativo**: describes QUÉ quieres lograr, no CÓMO hacerlo

Diferencias con lenguajes imperativos

Imperativo (Python, Java)	Declarativo (Prolog)
Instrucciones secuenciales	Hechos y reglas
Variables modificables	Variables inmutables (unificación)
Control explícito (if, for)	Backtracking automático

Imperativo (Python, Java)	Declarativo (Prolog)
Funciones con return	Predicados con éxito/fallo

Filosofía de Prolog

En lugar de escribir:

```
# Python
if dolor_oido and presion_oido:
    return "posible otitis"
```

En Prolog escribes:

```
% Prolog
diagnostico_posible(otitis) :-
    sintoma_presente(dolor_oido),
    sintoma_presente(presion_oido).
```

Prolog **busca automáticamente** todas las formas de hacer verdadero el predicado.

🔤 Sintaxis Básica de Prolog

1. Átomos (constantes)

```
dolor_oido      % átomo (empieza con minúscula)
otitis          % átomo
'Juan Pérez'    % átomo con espacios (requiere comillas)
```

2. Variables

```
X               % Variable (empieza con MAYÚSCULA)
Sintoma         % Variable
_               % Variable anónima (no importa su valor)
```

3. Hechos (declaraciones verdaderas)

```
sintoma_inicial(dolor_oido).
sintoma_inicial(zumbido).
% Formato: predicado(argumentos).
% Termina siempre con punto (.)
```

4. Reglas (implicaciones lógicas)

```
% Formato: cabeza :- cuerpo.
% Se lee: "cabeza es verdadero SI cuerpo es verdadero"

puede_diagnosticar(otitis) :-
    sintoma_presente(secrecion),
    sintoma_presente(dolor_oido).

% Si hay secreción Y dolor de oído, se puede diagnosticar otitis
```

5. Consultas (preguntas al sistema)

```
?- sintoma_inicial(dolor_oido).
% Respuesta: true.

?- sintoma_inicial(fiebre).
% Respuesta: false.

?- sintoma_inicial(X).
% Respuesta: X = dolor_oido ;
%            X = zumbido.
```

6. Operador `:-` (dos usos importantes)

Uso 1: Definir reglas

```
es_grave(X) :- sintoma_grave(X).
% "X es grave SI X es un síntoma grave"
```

Uso 2: Directivas de ejecución

```
:- ayuda.
% Ejecuta el predicado ayuda al cargar el archivo
```

7. Operador `->` (condicional if-then-else)

```
% Formato: (Condición -> Entonces ; SiNo)

evaluar(X) :-
    (X > 0.7 ->
        write('Riesgo ALTO')
    ; X > 0.4 ->
        write('Riesgo MEDIO')
    ;
        write('Riesgo BAJO')
    ).
```

8. Comentarios

```
% Comentario de una línea

/* Comentario
   de múltiples
   líneas */
```

9. Listas

```
[a, b, c]           % Lista de 3 elementos
[]                  % Lista vacía
[H|T]               % H = cabeza, T = cola (resto)
[dolor_oido, zumbido, presion_oido]
```

🧠 Conceptos Fundamentales

1. Unificación

La **unificación** es el mecanismo para hacer coincidir términos:

```
?- X = dolor_oido.
% X se unifica con dolor_oido
% X = dolor_oido

?- sintoma_inicial(X).
% X se unifica sucesivamente con cada hecho
% X = dolor_oido ;
% X = zumbido
```

2. Backtracking (retroceso)

Prolog **busca todas las soluciones posibles** retrocediendo cuando encuentra un fallo:

```
puede_evolucionar(dolor_oido, presion_oido).
puede_evolucionar(dolor_oido, dolor_punzante).

?- puede_evolucionar(dolor_oido, X).
% X = presion_oido ;    <- primera solución
% X = dolor_punzante.   <- al presionar ';', backtracking encuentra segunda
```

3. Recursión

Prolog usa **recursión** en lugar de bucles:

```
% Caso base
suma_lista([], 0).

% Caso recursivo
suma_lista([H|T], Suma) :-
    suma_lista(T, SumaT),
    Suma is H + SumaT.

?- suma_lista([1,2,3], S).
% S = 6
```

4. Predicados dinámicos

Permiten **modificar la base de conocimiento** durante la ejecución:

```
:- dynamic(sintoma_presente/1).

% Agregar un hecho
?- assertz(sintoma_presente(dolor_oido)).

% Consultar
?- sintoma_presente(dolor_oido).
% true.

% Eliminar todos los hechos de un predicado
?- retractall(sintoma_presente(_)).
```

5. Predicados útiles

findall/3 - Encuentra todas las soluciones

```
?- findall(X, sintoma_inicial(X), Lista).
% Lista = [dolor_oido, zumbido]
```

member/2 - Verifica pertenencia a lista

```
?- member(2, [1,2,3]).
% true.
```

append/3 - Concatena listas

```
?- append([1,2], [3,4], L).
% L = [1,2,3,4]
```

length/2 - Longitud de lista

```
?- length([a,b,c], N).
% N = 3
```

🗂️ Modelo de Conocimiento

Estructura del grafo de síntomas

El sistema modela los síntomas en **niveles de gravedad**:

```
NIVEL 1 (iniciales)     NIVEL 2 (intermedios)     NIVEL 3 (graves)        NIVEL 4 (crítico)
-----------------       --------------------      ----------------        -----------------
dolor_oido         →    presion_oido         →    oido_tapado        →    secrecion    →   OTITI
                   →    dolor_punzante       →                       →
zumbido            →    perdida_audicion     →
                   →    presion_oido         →

                        resfriado            →    secrecion_nasal    →
```

Definición de nodos por categoría

```
% NIVEL 1: Síntomas iniciales (leves)
sintoma_inicial(dolor_oido).
sintoma_inicial(zumbido).

% NIVEL 2: Síntomas intermedios
sintoma_intermedio(presion_oido).
sintoma_intermedio(perdida_audicion).

% NIVEL 2.5: Resfriado (categoría especial)
sintoma_intermedio_avanzado(resfriado).

% NIVEL 3: Síntomas graves
sintoma_grave(oido_tapado).
sintoma_grave(dolor_punzante).
sintoma_grave(secrecion_nasal).

% NIVEL 4: Síntoma crítico
sintoma_critico(secrecion).

% Diagnóstico final
diagnostico_final(otitis).
```

Relaciones entre síntomas (aristas del grafo)

```
% NIVEL 1 → NIVEL 2
puede_evolucionar(dolor_oido, presion_oido).
puede_evolucionar(dolor_oido, dolor_punzante).
puede_evolucionar(zumbido, presion_oido).
puede_evolucionar(zumbido, perdida_audicion).

% NIVEL 2 → NIVEL 3
puede_evolucionar(presion_oido, oido_tapado).
puede_evolucionar(presion_oido, dolor_punzante).
puede_evolucionar(perdida_audicion, oido_tapado).

% NIVEL 2.5 → NIVEL 3
puede_evolucionar(resfriado, secrecion_nasal).

% NIVEL 3 → NIVEL 4
puede_evolucionar(oido_tapado, secrecion).
puede_evolucionar(dolor_punzante, secrecion).
puede_evolucionar(secrecion_nasal, secrecion).

% NIVEL 4 → DIAGNÓSTICO
puede_evolucionar(secrecion, otitis).
```

Pesos de síntomas (para evaluación probabilística)

```
% Formato: peso_sintoma(Sintoma, Peso)
% Peso entre 0.0 y 1.0

peso_sintoma(dolor_oido, 0.3).
peso_sintoma(zumbido, 0.2).
peso_sintoma(presion_oido, 0.4).
peso_sintoma(perdida_audicion, 0.5).
peso_sintoma(resfriado, 0.3).
peso_sintoma(oido_tapado, 0.6).
peso_sintoma(dolor_punzante, 0.7).
peso_sintoma(secrecion_nasal, 0.5).
peso_sintoma(secrecion, 0.9).
peso_sintoma(otitis, 1.0).
```

Interpretación:

- `secrecion` : 0.9 → evidencia muy fuerte de otitis
- `dolor_punzante` : 0.7 → evidencia fuerte
- `zumbido` : 0.2 → evidencia débil (puede tener otras causas)

📐 Estructura del Código

Secciones principales de `agente_otitis.pl`

```
% ============================================================
% 1. DECLARACIONES DINÁMICAS
% ============================================================
:- dynamic(sintoma_presente/1).
:- dynamic(visitado/1).

% ============================================================
% 2. DEFINICIÓN DEL GRAFO
% ============================================================
% Nodos por categorías
sintoma_inicial(...).
sintoma_intermedio(...).
% ... etc

% Aristas (relaciones)
puede_evolucionar(..., ...).

% Pesos
peso_sintoma(..., ...).

% ============================================================
% 3. PREDICADOS DE BÚSQUEDA
% ============================================================
% Vecinos de un nodo
vecinos(Nodo, Lista) :- findall(...).

% BFS (Breadth-First Search)
bfs_otitis(Inicio) :- ...
bfs_cola(Cola, Visitados, Camino) :- ...

% DFS (Depth-First Search)
dfs_otitis(Inicio) :- ...
dfs_pila(Pila, Visitados, Camino) :- ...

% ============================================================
% 4. DIAGNÓSTICO Y EVALUACIÓN
% ============================================================
diagnosticar_otitis(Sintoma, Metodo) :- ...
```

```
evaluar_probabilidad(Camino) :- ...
calcular_probabilidad_camino(Camino, Prob) :- ...

% ============================================================
% 5. MODO INTERACTIVO
% ============================================================
interactivo :- ...
explorar_interactivo(Actual) :- ...
preguntar_sintomas_interactivos(Actual, Vecinos) :- ...

% ============================================================
% 6. UTILIDADES
% ============================================================
formatear_sintoma(Sintoma) :- ...
mostrar_camino(Lista) :- ...
mostrar_cola(Cola) :- ...

% ============================================================
% 7. CASOS DE PRUEBA
% ============================================================
iniciar :- ...
casos :- ...
caso_dolor_bfs :- ...
caso_zumbido_dfs :- ...
% ... etc

% ============================================================
% 8. AYUDA Y AUTOEJECUTAR
% ============================================================
ayuda :- ...
:- ayuda.  % Se ejecuta al cargar el archivo
```

🔍 Algoritmos Implementados

1. BFS (Búsqueda en Amplitud)

Pseudocódigo

```
BFS(inicio):
    cola = [[inicio]]
    visitados = []

    mientras cola no esté vacía:
        camino = primer elemento de cola
        actual = último nodo de camino

        si actual es 'otitis':
            ÉXITO: retornar camino

        si actual no ha sido visitado:
            marcar actual como visitado
            obtener vecinos de actual

            para cada vecino:
                nuevo_camino = camino + [vecino]
                agregar nuevo_camino al final de cola

    FALLO: no se encontró otitis
```

Implementación en Prolog

```
bfs_otitis(Inicio) :-
    write('=== INICIANDO BFS ==='), nl,
    format('Sintoma inicial: ~w~n', [Inicio]),
    retractall(visitado(_)),
    bfs_cola([[Inicio]], [], Camino),
    write('=== DIAGNOSTICO COMPLETADO ==='), nl,
    mostrar_camino(Camino),
    evaluar_probabilidad(Camino).

bfs_cola([[Actual|Camino]|_], _, [Actual|Camino]) :-
    diagnostico_final(Actual),
    !,
    write('¡Diagnostico alcanzado!'), nl.

bfs_cola([[Actual|Camino]|RestoCola], Visitados, Resultado) :-
    \+ member(Actual, Visitados),
    !,
    format('Visitando: ~w (Camino: ', [Actual]),
    reverse([Actual|Camino], CaminoOrdenado),
    mostrar_camino(CaminoOrdenado),
    write(')'), nl,

    assertz(visitado(Actual)),
    findall(X, puede_evolucionar(Actual, X), Vecinos),
    findall([Vecino,Actual|Camino], member(Vecino, Vecinos), NuevosCaminos),
    append(RestoCola, NuevosCaminos, NuevaCola),

    format('Cola actual: '),
    mostrar_cola(NuevaCola), nl,

    bfs_cola(NuevaCola, [Actual|Visitados], Resultado).

bfs_cola([_|RestoCola], Visitados, Resultado) :-
    bfs_cola(RestoCola, Visitados, Resultado).
```

Características:

- Explora nivel por nivel
- Garantiza encontrar el camino **más corto**
- Usa cola FIFO (First In, First Out)

2. DFS (Búsqueda en Profundidad)

Pseudocódigo

```
DFS(inicio):
    pila = [[inicio]]
    visitados = []

    mientras pila no esté vacía:
        camino = tope de pila
        actual = último nodo de camino

        si actual es 'otitis':
            ÉXITO: retornar camino

        si actual no ha sido visitado:
            marcar actual como visitado
            obtener vecinos de actual (en orden inverso)

            para cada vecino:
                nuevo_camino = camino + [vecino]
                agregar nuevo_camino al tope de pila

    FALLO: no se encontró otitis
```

Implementación en Prolog

```
dfs_otitis(Inicio) :-
    write('=== INICIANDO DFS ==='), nl,
    format('Sintoma inicial: ~w~n', [Inicio]),
    retractall(visitado(_)),
    dfs_pila([[Inicio]], [], Camino),
    write('=== DIAGNOSTICO COMPLETADO ==='), nl,
    mostrar_camino(Camino),
    evaluar_probabilidad(Camino).

dfs_pila([[Actual|Camino]|_], _, [Actual|Camino]) :-
    diagnostico_final(Actual),
    !,
    write('¡Diagnostico alcanzado!'), nl.

dfs_pila([[Actual|Camino]|RestoPila], Visitados, Resultado) :-
    \+ member(Actual, Visitados),
    !,
    format('Visitando: ~w (Camino: ', [Actual]),
    reverse([Actual|Camino], CaminoOrdenado),
    mostrar_camino(CaminoOrdenado),
    write(')'), nl,

    assertz(visitado(Actual)),
    findall(X, puede_evolucionar(Actual, X), Vecinos),
    reverse(Vecinos, VecinosInversos),
    findall([Vecino,Actual|Camino], member(Vecino, VecinosInversos), NuevosCaminos),
    append(NuevosCaminos, RestoPila, NuevaPila),

    format('Pila actual: '),
    mostrar_cola(NuevaPila), nl,

    dfs_pila(NuevaPila, [Actual|Visitados], Resultado).

dfs_pila([_|RestoPila], Visitados, Resultado) :-
    dfs_pila(RestoPila, Visitados, Resultado).
```

Características:

- Explora en profundidad primero
- **No** garantiza el camino más corto
- Usa pila LIFO (Last In, First Out)

Comparación BFS vs DFS

Aspecto	BFS	DFS
Estructura	Cola (FIFO)	Pila (LIFO)
Camino encontrado	**Más corto**	No necesariamente corto
Exploración	Por niveles	Por profundidad
Memoria	Mayor uso	Menor uso
Ejemplo salida	`dolor_oido → presion_oido → oido_tapado → secrecion → otitis`	Puede dar caminos más largos

🎮 Modo Interactivo

¿Cómo funciona?

El modo interactivo **guía al usuario** preguntando paso a paso sobre la presencia de síntomas:

```
interactivo :-
    write('=== MODO DIAGNOSTICO INTERACTIVO ==='), nl,
    write('Responda "si." o "no." a cada pregunta'), nl, nl,
    retractall(sintoma_presente(_)),
    retractall(visitado(_)),

    % Preguntar síntoma inicial
    write('Sintomas iniciales disponibles:'), nl,
    findall(S, sintoma_inicial(S), Iniciales),
    forall(member(S, Iniciales), (
        format('  - ~w~n', [S])
    )),
    nl,
    write('¿Cual es el sintoma inicial? '),
    read(Inicio),

    % Validar
    (sintoma_inicial(Inicio) ->
        assertz(sintoma_presente(Inicio)),
        explorar_interactivo(Inicio)
    ;
        write('Sintoma no valido.'), nl, fail
    ).
```

Flujo de ejecución

1. Usuario inicia: ?- interactivo.

2. Sistema muestra síntomas iniciales disponibles

3. Usuario elige uno (ej: dolor_oido)

4. Sistema pregunta por cada vecino:
   "¿Tiene presion_oido? (si/no): "

5. Si usuario responde "si":
   - Marca sintoma_presente(presion_oido)
   - Continúa explorando desde presion_oido

6. Si usuario responde "no":
   - Prueba con el siguiente vecino

7. Cuando llega a 'otitis':
   - Evalúa probabilidad basada en síntomas marcados
   - Muestra diagnóstico final

Implementación de la exploración

```
explorar_interactivo(Actual) :-
    diagnostico_final(Actual),
    !,
    write('=== DIAGNOSTICO ALCANZADO ==='), nl,
    findall(S, sintoma_presente(S), Sintomas),
    format('Sintomas presentes: ~w~n', [Sintomas]),
    evaluar_probabilidad(Sintomas).

explorar_interactivo(Actual) :-
    assertz(visitado(Actual)),
    findall(V, (puede_evolucionar(Actual, V), \+ visitado(V)), Vecinos),
    (Vecinos \= [] ->
        preguntar_sintomas_interactivos(Actual, Vecinos)
    ;
        write('No hay mas sintomas por explorar desde aqui.'), nl,
        fail
    ).

preguntar_sintomas_interactivos(_, []).

preguntar_sintomas_interactivos(Actual, [Vecino|Resto]) :-
    formatear_sintoma(Vecino),
    format('¿Tiene ~w? (si/no): ', [Vecino]),
    read(Respuesta),
    (Respuesta = si ->
        assertz(sintoma_presente(Vecino)),
        explorar_interactivo(Vecino)
    ; Respuesta = no ->
        preguntar_sintomas_interactivos(Actual, Resto)
    ;
        write('Respuesta invalida. Use "si." o "no."'), nl,
        preguntar_sintomas_interactivos(Actual, [Vecino|Resto])
    ).
```

Ejemplo de sesión interactiva

```
?- interactivo.
=== MODO DIAGNOSTICO INTERACTIVO ===
Responda "si." o "no." a cada pregunta

Sintomas iniciales disponibles:
  - dolor_oido
  - zumbido

¿Cual es el sintoma inicial? dolor_oido.

=== Explorando desde: dolor_oido ===

¿Tiene presion_oido? (si/no): si.

=== Explorando desde: presion_oido ===

¿Tiene oido_tapado? (si/no): si.

=== Explorando desde: oido_tapado ===

¿Tiene secrecion? (si/no): si.

=== Explorando desde: secrecion ===
=== DIAGNOSTICO ALCANZADO ===
Sintomas presentes: [dolor_oido, presion_oido, oido_tapado, secrecion]

Probabilidad calculada: 0.5500
Interpretación: Riesgo MEDIO-ALTO de otitis
```

📊 Evaluación de Probabilidad

Fórmula de cálculo

La probabilidad se calcula como el **promedio de los pesos** de los síntomas en el camino:

$$P = \frac{\sum_{i=1}^{n} peso(sintoma_i)}{n}$$

Implementación

```
evaluar_probabilidad(Camino) :-
    calcular_probabilidad_camino(Camino, Prob),
    format('~n=== EVALUACION DE PROBABILIDAD ===~n', []),
    format('Probabilidad calculada: ~4f~n', [Prob]),

    (Prob >= 0.7 ->
        write('Interpretacion: Riesgo ALTO de otitis'), nl,
        write('Recomendacion: Consultar medico URGENTE'), nl
    ; Prob >= 0.5 ->
        write('Interpretacion: Riesgo MEDIO-ALTO de otitis'), nl,
        write('Recomendacion: Consultar medico pronto'), nl
    ; Prob >= 0.3 ->
        write('Interpretacion: Riesgo MODERADO'), nl,
        write('Recomendacion: Monitorear sintomas'), nl
    ;
        write('Interpretacion: Riesgo BAJO'), nl,
        write('Recomendacion: Sintomas leves, observar evolucion'), nl
    ).

calcular_probabilidad_camino(Camino, Probabilidad) :-
    findall(Peso, (
        member(Sintoma, Camino),
        peso_sintoma(Sintoma, Peso)
    ), Pesos),

    (Pesos \= [] ->
        sum_list(Pesos, Suma),
        length(Pesos, N),
        Probabilidad is Suma / N
    ;
        Probabilidad is 0.0
    ).
```

Umbrales de interpretación

Probabilidad	Interpretación	Recomendación
≥ 0.7	**Riesgo ALTO**	Consultar médico URGENTE
0.5 - 0.69	**Riesgo MEDIO-ALTO**	Consultar médico pronto

Probabilidad	Interpretación	Recomendación
0.3 - 0.49	**Riesgo MODERADO**	Monitorear síntomas
< 0.3	**Riesgo BAJO**	Observar evolución

Ejemplo de cálculo

Camino: `[dolor_oido, presion_oido, oido_tapado, secrecion, otitis]`

Pesos:

- `dolor_oido` : 0.3
- `presion_oido` : 0.4
- `oido_tapado` : 0.6
- `secrecion` : 0.9
- `otitis` : 1.0

$$P = \frac{0.3 + 0.4 + 0.6 + 0.9 + 1.0}{5} = \frac{3.2}{5} = 0.64$$

Interpretación: Riesgo MEDIO-ALTO de otitis (0.64)

💡 Ejemplos de Uso

1. Ejecución básica

```
# Iniciar SWI-Prolog con el archivo
swipl -s agente_otitis.pl
```

2. Menú principal

```
?- iniciar.
=== SISTEMA EXPERTO PARA DIAGNOSTICO DE OTITIS ===

Comandos disponibles:
1. iniciar.                    - Mostrar este menu
2. casos.                      - Ver todos los casos de prueba
3. interactivo.                - Modo diagnostico interactivo
4. diagnosticar_otitis(S, M).  - Diagnosticar desde sintoma S con metodo M
   Metodos: bfs o dfs
5. ayuda.                      - Mostrar informacion de ayuda

Sintomas iniciales disponibles:
  - dolor_oido
  - zumbido
  - resfriado (categoria especial)
```

3. Casos de prueba predefinidos

```
?- casos.
=== CASOS DE PRUEBA DISPONIBLES ===

1. caso_dolor_bfs.          - Dolor de oido con BFS
2. caso_dolor_dfs.          - Dolor de oido con DFS
3. caso_zumbido_bfs.        - Zumbido con BFS
4. caso_zumbido_dfs.        - Zumbido con DFS
5. caso_resfriado_bfs.      - Resfriado con BFS
6. caso_resfriado_dfs.      - Resfriado con DFS

Ejemplo de uso:
?- caso_dolor_bfs.
```

4. Ejecutar caso específico

```
?- caso_dolor_bfs.
=== CASO: Dolor de oido con BFS ===

=== INICIANDO BFS ===
Sintoma inicial: dolor_oido
Paso 1: Visitando dolor_oido
  Vecinos encontrados: [presion_oido, dolor_punzante]
  Cola: [[presion_oido, dolor_oido], [dolor_punzante, dolor_oido]]

Paso 2: Visitando presion_oido
  Vecinos encontrados: [oido_tapado, dolor_punzante]
  Cola: [[dolor_punzante, dolor_oido], [oido_tapado, presion_oido, dolor_oido], ...]

Paso 3: Visitando dolor_punzante
  Vecinos encontrados: [secrecion]
  Cola: [...]

Paso 4: Visitando oido_tapado
  Vecinos encontrados: [secrecion]

Paso 5: Visitando secrecion
  Vecinos encontrados: [otitis]

Paso 6: Visitando otitis
¡Diagnostico alcanzado!

=== DIAGNOSTICO COMPLETADO ===
Camino encontrado: dolor_oido → presion_oido → oido_tapado → secrecion → otitis

=== EVALUACION DE PROBABILIDAD ===
Probabilidad calculada: 0.6400
Interpretacion: Riesgo MEDIO-ALTO de otitis
Recomendacion: Consultar medico pronto
```

5. Diagnosticar desde cualquier síntoma

```
?- diagnosticar_otitis(zumbido, dfs).
=== INICIANDO DFS ===
Sintoma inicial: zumbido
...
Camino encontrado: zumbido → perdida_audicion → oido_tapado → secrecion → otitis
Probabilidad: 0.6500
```

6. Modo interactivo completo

```
?- interactivo.
=== MODO DIAGNOSTICO INTERACTIVO ===
Responda "si." o "no." a cada pregunta

Sintomas iniciales disponibles:
  - dolor_oido
  - zumbido

¿Cual es el sintoma inicial? dolor_oido.

=== Explorando desde: dolor_oido ===

¿Tiene presion_oido? (si/no): si.
¿Tiene oido_tapado? (si/no): no.
¿Tiene dolor_punzante? (si/no): si.
¿Tiene secrecion? (si/no): si.

=== DIAGNOSTICO ALCANZADO ===
Sintomas presentes: [dolor_oido, presion_oido, dolor_punzante, secrecion]
Probabilidad: 0.5750
Interpretacion: Riesgo MEDIO-ALTO de otitis
```

7. Consultas personalizadas

```
% Listar todos los síntomas iniciales
?- findall(X, sintoma_inicial(X), L).
L = [dolor_oido, zumbido].

% Encontrar vecinos de un síntoma
?- findall(X, puede_evolucionar(dolor_oido, X), Vecinos).
Vecinos = [presion_oido, dolor_punzante].

% Verificar peso de un síntoma
?- peso_sintoma(secrecion, P).
P = 0.9.

% Listar todos los síntomas graves
?- findall(X, sintoma_grave(X), Graves).
Graves = [oido_tapado, dolor_punzante, secrecion_nasal].
```

🔧 Cómo Extender el Sistema

1. Añadir un nuevo síntoma

```
% Paso 1: Declarar el síntoma en la categoría apropiada
sintoma_grave(mareo).

% Paso 2: Definir su peso
peso_sintoma(mareo, 0.65).

% Paso 3: Conectarlo en el grafo
puede_evolucionar(presion_oido, mareo).
puede_evolucionar(mareo, otitis).
```

2. Añadir una nueva relación entre síntomas existentes

```
% Conectar zumbido directamente con oido_tapado
puede_evolucionar(zumbido, oido_tapado).
```

3. Modificar pesos para ajustar probabilidades

```
% Aumentar el peso de fiebre si se considera más indicativo
peso_sintoma(fiebre, 0.8).  % antes era 0.5
```

4. Añadir un nuevo nivel de síntomas

```
% Crear categoría de síntomas sistémicos
sintoma_sistemico(fiebre).
sintoma_sistemico(escalofrios).

% Conectarlos
puede_evolucionar(secrecion, fiebre).
puede_evolucionar(fiebre, otitis).
```

5. Crear un nuevo caso de prueba

```
caso_mareo_bfs :-
    write('=== CASO: Mareo con BFS ==='), nl, nl,
    diagnosticar_otitis(mareo, bfs).
```

6. Modificar umbrales de evaluación

```
evaluar_probabilidad(Camino) :-
    calcular_probabilidad_camino(Camino, Prob),

    % Umbrales personalizados
    (Prob >= 0.8 ->  % Cambio: antes era 0.7
        write('Riesgo CRITICO')
    ; Prob >= 0.6 ->  % Cambio: antes era 0.5
        write('Riesgo ALTO')
    ; ...
    ).
```

7. Añadir reglas de combinación de síntomas

```
% Diagnóstico directo si se cumplen múltiples síntomas
diagnostico_directo(otitis) :-
    sintoma_presente(secrecion),
    sintoma_presente(dolor_punzante),
    sintoma_presente(fiebre).

% Usar en modo interactivo
explorar_interactivo(Actual) :-
    diagnostico_directo(otitis),
    !,
    write('¡Diagnostico directo por combinacion de sintomas!'), nl.
```

🐛 Depuración y Pruebas

1. Activar modo trace (seguimiento paso a paso)

```
?- trace.
% Ahora cada paso de ejecución se mostrará

?- diagnosticar_otitis(dolor_oido, bfs).
% Verás cada unificación, llamada y retorno
```

2. Desactivar trace

```
?- notrace.
```

3. Usar spy para depurar predicados específicos

```
% Espiar solo las llamadas a bfs_cola
?- spy(bfs_cola).

% Ejecutar
?- diagnosticar_otitis(dolor_oido, bfs).

% Detener espionaje
?- nospy(bfs_cola).
```

4. Verificar estado de la base de datos

```
% Ver todos los síntomas presentes
?- sintoma_presente(X).

% Ver todos los nodos visitados
?- visitado(X).

% Limpiar manualmente
?- retractall(sintoma_presente(_)).
?- retractall(visitado(_)).
```

5. Probar predicados individuales

```
% Probar vecinos
?- vecinos(dolor_oido, V).
V = [presion_oido, dolor_punzante].

% Probar cálculo de probabilidad
?- calcular_probabilidad_camino([dolor_oido, presion_oido, secrecion], P).
P = 0.5333333333333333.

% Probar formateo
?- formatear_sintoma(dolor_oido).
Dolor de Oído
```

6. Validar integridad del grafo

```
% Verificar que todos los nodos tienen vecinos (excepto otitis)
?- findall(X, (sintoma_inicial(X), \+ puede_evolucionar(X, _)), Huerfanos).

% Verificar que todos los síntomas tienen peso
?- findall(X, (sintoma_inicial(X), \+ peso_sintoma(X, _)), SinPeso).
```

7. Usar leash para controlar nivel de detalle en trace

```
% Mostrar solo llamadas y salidas
?- leash(+call +exit).

% Mostrar todo
?- leash(+all).
```

8. Debugging común

Problema: "No encuentra camino a otitis"

```
% Verificar conexión completa
?- diagnosticar_otitis(dolor_oido, bfs).

% Si falla, verificar:
?- puede_evolucionar(secrecion, otitis).
% Debe ser true
```

Problema: "Probabilidad siempre es 0"

```
% Verificar que los pesos están definidos
?- peso_sintoma(dolor_oido, P).

% Si falla, añadir:
peso_sintoma(dolor_oido, 0.3).
```

Problema: "Síntomas no se marcan en modo interactivo"

```
% Verificar dynamic
?- current_predicate(sintoma_presente/1).
% Debe ser true

% Verificar que assertz funciona
?- assertz(sintoma_presente(test)).
?- sintoma_presente(test).
% Debe ser true
```

📖 Glosario de Términos

Términos de Prolog

Término	Definición	Ejemplo
Átomo	Constante que empieza con minúscula	`dolor_oido` , `otitis`
Variable	Identificador que empieza con mayúscula	`X` , `Sintoma` , `_`
Hecho	Afirmación verdadera	`sintoma_inicial(dolor_oido).`
Regla	Implicación lógica	`es_grave(X) :- sintoma_grave(X).`
Predicado	Relación o propiedad	`puede_evolucionar/2` (aridad 2)
Unificación	Proceso de hacer coincidir términos	`X = dolor_oido`
Backtracking	Retroceso para buscar alternativas	Al usar `;` en consultas
Aridad	Número de argumentos de un predicado	`peso_sintoma/2` tiene aridad 2
Clausula	Hecho o regla	`sintoma_inicial(X).`
Meta	Objetivo a probar	`?- sintoma_inicial(X).`

Operadores

Operador	Significado	Uso
`:-`	"si" (regla)	`cabeza :- cuerpo.`
`:-`	directiva	`:- ayuda.`
`->`	if-then	`(Cond -> Then ; Else)`
`;`	OR lógico	`(A ; B)`
`,`	AND lógico	`(A, B)`
`\+`	negación por fallo	`\+ visitado(X)`
`=`	unificación	`X = valor`
`is`	evaluación aritmética	`X is 2 + 3`
`==`	igualdad estricta	`X == Y`
`\=`	no unificable	`X \= Y`

Predicados built-in útiles

Predicado	Descripción	Ejemplo
`write/1`	Imprimir en consola	`write('Hola')`
`nl/0`	Nueva línea	`nl`
`format/2`	Imprimir con formato	`format('Valor: ~w~n', [X])`
`read/1`	Leer de consola	`read(X)`
`findall/3`	Encuentra todas las soluciones	`findall(X, p(X), L)`
`member/2`	Pertenencia a lista	`member(a, [a,b,c])`
`append/3`	Concatenar listas	`append([1,2], [3], L)`
`length/2`	Longitud de lista	`length([a,b,c], N)`
`reverse/2`	Invertir lista	`reverse([1,2,3], R)`
`sum_list/2`	Suma de elementos	`sum_list([1,2,3], S)`

Predicado	Descripción	Ejemplo
`assertz/1`	Añadir hecho al final	`assertz(p(a))`
`retractall/1`	Eliminar todos los hechos	`retractall(p(_))`
`fail/0`	Falla siempre	`fail`
`!`	Corte (cut)	`p(X) :- q(X), !.`

Términos del dominio (Otitis)

Término	Definición
Otitis	Inflamación del oído (medio, externo o interno)
Otitis media aguda	Infección del oído medio con síntomas agudos
Otalgia	Dolor de oído
Otorrea / Secreción	Supuración del oído
Tinnitus / Zumbido	Percepción de sonido sin fuente externa
Hipoacusia / Pérdida audición	Disminución de la capacidad auditiva
Trompa de Eustaquio	Conducto que conecta oído medio con faringe

📚 Recursos Adicionales

Tutoriales de Prolog

1. **Learn Prolog Now!** (gratuito)
   - URL: http://www.learnprolognow.org/
   - Descripción: Tutorial interactivo paso a paso, ideal para principiantes
2. **SWI-Prolog Documentation**
   - URL: https://www.swi-prolog.org/pldoc/index.html
   - Descripción: Documentación oficial completa
3. **99 Prolog Problems**
   - URL: https://www.ic.unicamp.br/~meidanis/courses/mc336/2009s2/prolog/problemas/
   - Descripción: Ejercicios prácticos para mejorar habilidades

Libros recomendados

1. **"Programming in Prolog"** - Clocksin & Mellish
   - Nivel: Principiante a intermedio
   - Clásico de referencia
2. **"The Art of Prolog"** - Sterling & Shapiro
   - Nivel: Intermedio a avanzado
   - Conceptos formales y técnicas avanzadas
3. **"Prolog Programming for Artificial Intelligence"** - Ivan Bratko
   - Nivel: Intermedio
   - Enfocado en IA y algoritmos de búsqueda

Videos y cursos

1. **Derek Banas - Prolog Tutorial** (YouTube)
   - Duración: ~30 minutos
   - Introducción rápida y práctica
2. **Coursera - Introduction to Logic** (Universidad de Stanford)
   - Incluye sección de Prolog
   - Gratis para auditar

Comunidades y foros

1. **Stack Overflow** - Tag [prolog]
   - URL: https://stackoverflow.com/questions/tagged/prolog
   - Comunidad activa para resolver dudas
2. **SWI-Prolog Discourse**
   - URL: https://swi-prolog.discourse.group/
   - Foro oficial de la comunidad SWI-Prolog

Herramientas

1. **SWISH** (SWI-Prolog online)
   - URL: https://swish.swi-prolog.org/
   - Entorno Prolog en el navegador, sin instalación
2. **Visual Prolog** (IDE comercial)
   - Para desarrollo más complejo
3. **VSCode Extensions**
   - "VSC-Prolog" - Sintaxis highlighting y snippets

🎓 Ejercicios Prácticos

Nivel Principiante

1. **Consultas básicas**

```
% Ejecutar y entender:
?- sintoma_inicial(X).
?- peso_sintoma(secrecion, P).
?- puede_evolucionar(dolor_oido, X).
```

2. **Añadir un síntoma simple**
   - Añade `fiebre` como síntoma grave
   - Conéctalo: `secrecion` → `fiebre` → `otitis`
   - Asígnale peso 0.8
   - Prueba con BFS
3. **Modificar un peso**
   - Cambia el peso de `zumbido` de 0.2 a 0.5
   - Observa cómo cambia la probabilidad en los casos de prueba

Nivel Intermedio

4. **Crear un nuevo caso de prueba**

```
caso_personalizado :-
    write('Mi caso de prueba'), nl,
    diagnosticar_otitis(zumbido, bfs).
```

5. **Implementar contador de pasos**
   - Modifica `bfs_cola/3` para contar y mostrar el número total de pasos
6. **Añadir validación de caminos**
   - Crea un predicado `validar_camino(Camino)` que verifique que cada transición existe en `puede_evolucionar/2`

Nivel Avanzado

7. **Implementar A* (A-star)**
   - Usa los pesos como heurística
   - Implementa cola de prioridad

8. **Sistema de reglas compuestas**
    - Crea reglas tipo: "SI (dolor_oido Y secrecion) ENTONCES otitis con 95% confianza"
9. **Guardar historial de diagnósticos**

```
:- dynamic(historial_diagnostico/3).
% historial_diagnostico(Fecha, Sintomas, Probabilidad)
```

10. **Interfaz de explicación**
    - Implementa `explicar_diagnostico(Camino)` que genere explicación en lenguaje natural

⚠️ Limitaciones y Advertencias

Limitaciones técnicas

1. **Simplicidad del modelo**
    - El grafo es estático y lineal
    - No modela comorbilidades ni interacciones complejas
2. **Evaluación probabilística básica**
    - Usa promedio simple, no fusión bayesiana
    - No considera condicionalidades (ej: "dolor_oido Y fiebre juntos aumentan probabilidad más que la suma individual")
3. **No aprende**
    - No hay mecanismo de machine learning
    - Los pesos son fijos y manuales

Limitaciones clínicas

⚠️ ADVERTENCIA IMPORTANTE ⚠️

Este sistema es únicamente educativo y NO debe usarse para:

- Diagnóstico médico real
- Toma de decisiones clínicas
- Sustitución de consulta médica profesional

Siempre consulte a un profesional de la salud calificado para síntomas reales.

Posibles mejoras futuras

1. **Incorporar incertidumbre**
   - Usar factores de certeza (Certainty Factors)
   - Implementar redes bayesianas
2. **Ampliar el modelo**
   - Añadir más diagnósticos (otitis externa, mastoiditis, etc.)
   - Modelar complicaciones y evoluciones temporales
3. **Aprendizaje automático**
   - Ajustar pesos automáticamente desde casos históricos
   - Validación con datos clínicos reales
4. **Interfaz mejorada**
   - Integración con base de datos médica
   - Exportación de informes
   - Multiidioma

📝 Conclusión

Este sistema experto en Prolog es una **herramienta educativa** que demuestra:

✅ Implementación de **motores de inferencia** (forward/backward chaining simulados con BFS/DFS)
✅ Uso de **predicados dinámicos** para estado mutable
✅ **Backtracking** y búsqueda en grafos
✅ **Interacción con usuario** mediante lectura de entrada
✅ **Evaluación cuantitativa** con pesos y probabilidades

Próximos pasos recomendados

1. **Ejecutar todos los casos de prueba** para familiarizarte con el comportamiento
2. **Usar** `trace` para seguir la ejecución paso a paso
3. **Modificar el grafo** añadiendo nuevos síntomas y relaciones
4. **Experimentar con los pesos** para ver cómo afectan las evaluaciones
5. **Implementar extensiones** (ver sección "Ejercicios Prácticos")
6. **Estudiar los recursos adicionales** para profundizar en Prolog

📞 Soporte y Contribuciones

¿Encontraste un error?

- Verifica que estás usando **SWI-Prolog 8.0+**
- Consulta la sección de **Depuración y Pruebas**
- Revisa el **Glosario** si no entiendes un término

¿Quieres contribuir?

- Añade nuevos síntomas clínicamente relevantes
- Mejora los algoritmos de evaluación
- Traduce la documentación
- Crea tests unitarios

Última actualización: Noviembre 8, 2025
Versión: 1.0
Licencia: MIT (ver archivo LICENSE)

¡Buena suerte con tu exposición y aprendizaje de Prolog! 🚀