PEAS para simulación de detección de una enfermedad (Anemia)

Nombre: Moisés Benalcázar

Materia: Aplicaciones basadas en el conocimiento

Fecha: 28/10/2025

Desarrollar un sistema experto, para simular un diagnóstico médico hasta encontrar la enfermedad.

El sistema experto que se va a desarrollar se enfoca en detectar si el paciente que realiza la consulta resulta positivo para presentar la enfermedad de anemia de manera general, mediante la información que le proporcionara el paciente al agente como: posibles auto diagnósticos he información sobre su nutrición.

Performance measure (**Medida de desempeño**)

- Tiempo de respuesta
- Exactitud del diagnóstico.
- Nivel de satisfacción por parte del cliente

Enviroment (**Entorno**)

- Grafo con síntomas de anemia.
- Anemia (enfermedad)
- Palidez, fatiga (síntomas)
- Labios o lengua pálidos, uñas quebradizas, caída del cabello (conficiones físicas)
- Poca (carne roja, pescado o legumbres), alto consumo de café o té, dieta normal (alimentación)

Atuators (**Actuadores**)

- Pantalla, por la que se mostrara los mensajes: Paciente con anemia / Paciente sin anemia.

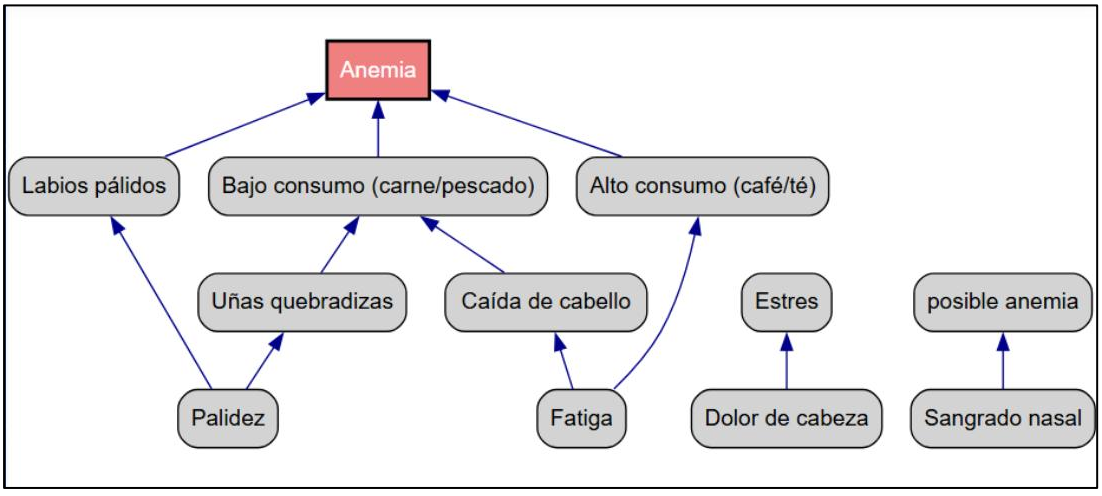
Sensors (**Sensores**)

- Entrada de los síntomas mediante el teclado.

Propiedades del agente

Parcialmente observable: El sistema no puede observar al paciente, así como ciertos síntomas que ayudarían a que el resultado sea más acertado y rápido de realizar.
Agente único: El sistema únicamente interactúa con el paciente, con ningún agente adicional dentro de su entorno.
Determinista: El sistema determina el estado siguiente dependiendo de las preguntas respondidas por el paciente.
Secuencial: El sistema se basa en una serie de respuestas para acercarse al mejor diagnóstico.
Estático: Si el sistema cambia su entorno, cambia el enfoque para el que se diseño (un sistema para simular un diagnóstico médico).
Discreto: El sistema tiene un número limitado de preguntas para realizar el diagnostico.
Conocido: El sistema conoce, los síntomas y posibles acontecimientos para que el paciente haya adquirido la enfermedad.

Construcción del grafo



Implementación de búsqueda

1. Búsqueda en Profundidad (DFS): El agente explora profundamente cada rama antes de retroceder.

```python
# 1. Definicion del Grafo (Entorno)
# Base de conocimiento para la anemia.
base_conocimiento_anemia = {
    'Fatiga': ['Caida de cabello', 'Alto consumo (cafe/te)'],
    'Palidez': ['Labios palidos', 'Unas quebradizas'],
    'Caida de cabello': ['Bajo consumo (carne/pescado)'],
    'Unas quebradizas': ['Bajo consumo (carne/pescado)'],
    'Labios palidos': ['Anemia'],
    'Alto consumo (cafe/te)': ['Anemia'],
    'Bajo consumo (carne/pescado)': ['Anemia'],
    'Anemia': [],
    'Dolor de cabeza': ['Estres'],
    'Sangrado nasal': ['posible anemia'],
}

# 2. Implementación de Búsqueda en Profundidad (DFS)
def busqueda_dfs(grafo, nodo_inicio, nodo_objetivo):

    visitados = set() #nodos visitados, inicialmente vacio
    pila = [nodo_inicio] # pila inicial para DFS
    padres = {nodo_inicio: None} # para reconstruir la ruta
    orden = [] #regresa el orden de recorrido

    while pila:
        nodo_actual = pila.pop() #LIFO primero en entrar, ultimo en salir

        if nodo_actual not in visitados: #si no ha sido visitado
            visitados.add(nodo_actual) #marcar como visitado
            orden.append(nodo_actual) #guardar el nodo actual en orden de recorrido

            if nodo_actual == nodo_objetivo: #si el nodo actual es el objetivo
                ruta = [] #ruta del recorrido de regreso
                n = nodo_actual #comenzar desde el nodo objetivo
                while n is not None: #mientras haya nodos en la ruta
                    ruta.append(n) #agregar el nodo a la ruta
                    n = padres.get(n) #mover al padre del nodo
                ruta.reverse() #invertir la ruta para obtener el orden correcto
                return True, orden, ruta #encontrado

            vecinos = grafo.get(nodo_actual, []) # obtener vecinos del nodo actual

            for vecino in reversed(vecinos):  #para cada vecino del nodo actual (reversed para mantener orden)
                if vecino not in visitados: #si el vecino no ha sido visitado
                    padres[vecino] = nodo_actual #establecer el padre
                    pila.append(vecino) #agregar a la pila para explorar

    return False, orden, []  # No encontrado

# 3. Agente de Diagnóstico con DFS
def agente_diagnostico_dfs(sintomas_paciente, base_conocimiento): #inicio de la funcion
    objetivo = "Anemia" #definir el objetivo

    for sintoma_inicial in sintomas_paciente: #para cada sintoma del paciente
        if sintoma_inicial not in base_conocimiento: #si el sintoma no esta en la base de conocimiento
            continue #saltar al siguiente sintoma
        print(f"Iniciando búsqueda DFS desde: '{sintoma_inicial}'...") #mensaje de inicio
        encontrado, orden_recorrido, ruta = busqueda_dfs(base_conocimiento, sintoma_inicial, objetivo)
        print("Orden de recorrido DFS:", " -> ".join(orden_recorrido)) #imprimir el orden de recorrido
        if encontrado: #si se encontro el objetivo
            print("--- Diagnóstico Final (DFS) ---")
            print("Resultado: Paciente con anemia")
            print("Ruta de diagnóstico:", " -> ".join(ruta)) #imprimir la ruta
            print("\n-------------------------------")
            return #finalizar la funcion

    print("--- Diagnóstico Final (DFS) ---")
    print("Resultado: Paciente sin anemia")
    print("\n-------------------------------")


# --- 4. Ejecución del Agente DFS ---
if __name__ == "__main__":

    # Caso 1: Paciente con palidez
    sintomas_1 = ['Palidez']
    agente_diagnostico_dfs(sintomas_1, base_conocimiento_anemia)

    # Caso 2: Paciente con Fatiga
    sintomas_2 = ['Fatiga']
    agente_diagnostico_dfs(sintomas_2, base_conocimiento_anemia)
```

2. Búsqueda en Amplitud (BFS): El agente explora nivel por nivel, asegurando encontrar el camino más corto.

```python
from collections import deque
# 1. Definicion del Grafo (Entorno)
# Base de conocimiento para la anemia.
base_conocimiento_anemia = {
    'Fatiga': ['Caida de cabello', 'Alto consumo (cafe/te)'],
    'Palidez': ['Labios palidos', 'Unas quebradizas'],
    'Caida de cabello': ['Bajo consumo (carne/pescado)'],
    'Unas quebradizas': ['Bajo consumo (carne/pescado)'],
    'Labios palidos': ['Anemia'],
    'Alto consumo (cafe/te)': ['Anemia'],
    'Bajo consumo (carne/pescado)': ['Anemia'],
    'Anemia': [],
    'Dolor de cabeza': ['Estres'],
    'Sangrado nasal': ['posible anemia'],
}

# 2. Implementacion de Busqueda en Amplitud (BFS)
def busqueda_bfs(grafo, nodo_inicio, nodo_objetivo):

    visitados = set([nodo_inicio]) #nodos visitados
    cola = deque([nodo_inicio]) #cola para BFS
    padres = {nodo_inicio: None} #para reconstruir la ruta
    orden = [] # Para registrar el orden de recorrido

    while cola:    #mientras haya nodos en la cola hacer
        nodo_actual = cola.popleft() #sacar el primer nodo de la cola
        orden.append(nodo_actual) #guarda el nodo actual en orden de recorrido

        if nodo_actual == nodo_objetivo: #si el nodo actual es el objetivo
            ruta = [] #ruta del recorrido de regreso
            n = nodo_actual #comenzar desde el nodo objetivo
            while n is not None: #mientras haya nodos en la ruta
                ruta.append(n) #agregar el nodo a la ruta
                n = padres.get(n) #mover al padre del nodo
            ruta.reverse() #invertir la ruta para obtener el orden correcto
            return True, orden, ruta #encontrado

        for vecino in grafo.get(nodo_actual, []): # si no es el objetivo, explorar vecinos
            if vecino not in visitados: #si el vecino no ha sido visitado
                visitados.add(vecino) #marcar como visitado
                padres[vecino] = nodo_actual #establecer el padre
                cola.append(vecino) #agregar a la cola para explorar

    return False, orden, []  # No encontrado

# 3. Agente de Diagnostico (BFS)
def agente_diagnostico_bfs(sintomas_paciente, base_conocimiento): #inicio de la funcion
    objetivo = "Anemia" #definir el objetivo

    for sintoma_inicial in sintomas_paciente: #para cada sintoma del paciente
        if sintoma_inicial not in base_conocimiento: #si el sintoma no esta en la base de conocimiento
            continue #saltar al siguiente sintoma
        print(f"Iniciando busqueda BFS desde: '{sintoma_inicial}'...")  #mensaje de inicio
        encontrado, orden_recorrido, ruta = busqueda_bfs(base_conocimiento, sintoma_inicial, objetivo) #llamar a la funcion de busqueda BFS
        print("Orden de recorrido BFS:", " -> ".join(orden_recorrido)) #imprimir el orden de recorrido
        if encontrado: #si se encontro el objetivo
            print("--- Diagnostico Final (BFS) ---")
            print("Resultado: Paciente con anemia")
            print("Ruta de diagnostico (la mas corta):", " -> ".join(ruta)) #imprimir la ruta
            print("\n-------------------------------")
            return #finalizar la funcion

    # Si ningun sintoma llevo al objetivo
    print("--- Diagnostico Final (BFS) ---")
    print("Resultado: Paciente sin anemia")
    print("\n-------------------------------")

# 4. Ejecucion del Agente BFS
if __name__ == "__main__":
    # Caso 1: Paciente con palidez
    sintomas_1 = ['Palidez']
    agente_diagnostico_bfs(sintomas_1, base_conocimiento_anemia)

    # Caso 2: Paciente con sangrado nasal
    sintomas_2 = ['Sangrado nasal']
    agente_diagnostico_bfs(sintomas_2, base_conocimiento_anemia)
```

Resultados:

Búsqueda en amplitud:

```
Iniciando busqueda BFS desde: 'Palidez'...
Orden de recorrido BFS: Palidez -> Labios palidos -> Unas quebradizas -> Anemia
--- Diagnostico Final (BFS) ---
Resultado: Paciente con anemia
Ruta de diagnostico (la mas corta): Palidez -> Labios palidos -> Anemia

------------------------------
Iniciando busqueda BFS desde: 'Sangrado nasal'...
Orden de recorrido BFS: Sangrado nasal -> posible anemia
--- Diagnostico Final (BFS) ---
Resultado: Paciente sin anemia
```

Búsqueda en profundidad:

```
Iniciando búsqueda DFS desde: 'Palidez'...
Orden de recorrido DFS: Palidez -> Labios palidos -> Anemia
--- Diagnóstico Final (DFS) ---
Resultado: Paciente con anemia
Ruta de diagnóstico: Palidez -> Labios palidos -> Anemia

------------------------------
Iniciando búsqueda DFS desde: 'Sangrado nasal'...
Orden de recorrido DFS: Sangrado nasal -> posible anemia
--- Diagnóstico Final (DFS) ---
Resultado: Paciente sin anemia
```

Evaluación del agente

1. **¿Qué diferencias observan entre DFS y BFS en su agente?**

   Las principales diferencias son en su estructura teniendo en cuenta que por el tipo de búsqueda cambia la forma de guardar la información.

   En BFS búsqueda en amplitud la información se guarda en una cola la cual se maneja en una búsqueda por nivel, analizando todo, buscando la ruta más corta para llegar a su objetivo.

   En la DFS búsqueda en profundidad guarda su información en una pila o stack, el cual realiza una búsqueda por rama hasta llegar a la hoja o nodo objetivo, garantizando un menor tiempo hasta su objetivo, pero no la mejor ruta hasta alcanzar su objetivo.

2. **¿Cuál método es más racional en su entorno (según PEAS)?**
   El de búsqueda por DFS es la mejor forma ya que nuestro agente se encuentra rodeado por nodos con síntomas nos informara un poco más hasta llegar a la enfermedad 'Anemia' en nuestro caso; por otro lado, el BFS nos daría la mejor ruta o más corta la cual no está mal, pero es un algoritmo que usa más memoria y tiempo.

3. **¿Cómo mide el agente su éxito o desempeño?**
   Para medir su éxito se verifica si encuentra la enfermedad con los síntomas de un paciente real, par medir su tiempo se lo hace mediante el tiempo de ejecución y para medir el nivel de satisfacción se consulta al usuario que tal le pareció.

4. **¿Qué riesgos existirían si el diagnóstico dependiera solo del algoritmo?**
   Que se puede omitir información importante ante un diagnóstico real, este agente será solo como un consultor en caso de duda, para que el paciente se acerque a realizarse una revisión.

5. **¿Cómo podría mejorarse el agente incorporando conocimiento probabilístico?**
   Se podría mejorar colocando las probabilidades como peso en los síntomas para que si se acerca o cruza por los niveles más altos significa que tiene más probabilidad certera de que tenga Anemia.

Conclusiones

- El sistema experto diseñado para la simulación de diagnóstico de la anemia se basa en un agente que interactúa con los pacientes para identificar síntomas comunes de la enfermedad.

- Se implementaron dos técnicas de búsqueda, en el cual yo pienso que el mejor para este caso debido a su naturaleza secuencial es DFS el cual cada síntoma es analizado progresivamente hasta alcanzar un diagnóstico; por otro lado, el BFS encuentra el camino más corto pero puede omitir síntomas en su proceso.

Recomendaciones

- Mejorar la experiencia del usuario diseñando una mejor interfaz que sea interactiva y amigable, permitiendo entradas dinámicas o selección de los síntomas mediante botones.

- Establecer un sistema de nivel de satisfacción de los usuarios para poder medir la experiencia y satisfacción del usuario, siendo punto importante para la medida de desempeño

Anexos

Enlace a GitHub para acceder al video, la presentación y los códigos que se realizó para el agente inteligente:
https://github.com/Sebas8173/UniversityRepository/tree/main/Apl%20Basadas%20en%20el%20Conocimiento/Agente%20inteligente