



thynk unlimited

Agente inteligente
con bùsqueda de
enfermedades en

Grafos

By: Moises Benalcázar

next slide →

02



thynk unlimited

Introduction

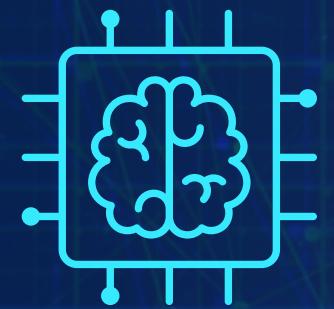
Se diseño y programó un agente inteligente de diagnóstico médico, capaz de recorrer un grafo mediante búsqueda a ciegas (DFS o BFS) para determinar si un paciente presenta o no Anemia.

El agente deberá formular su entorno con el modelo PEAS, aplicando el concepto de racionalidad y demostrando cómo toma decisiones basadas en búsqueda.

next slide →

Modelo PEAS

03



PERFORMANCE
MEASURE

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



ENVIRONMENT

- *Anemia .*
- *Palidez, fatiga .*
- *Labios o lengua pálidos, uñas quebradizas, caída del cabello.*
- *Poca (carne roja o pescado), alto consumo de café o té.*



ACTUATORS

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

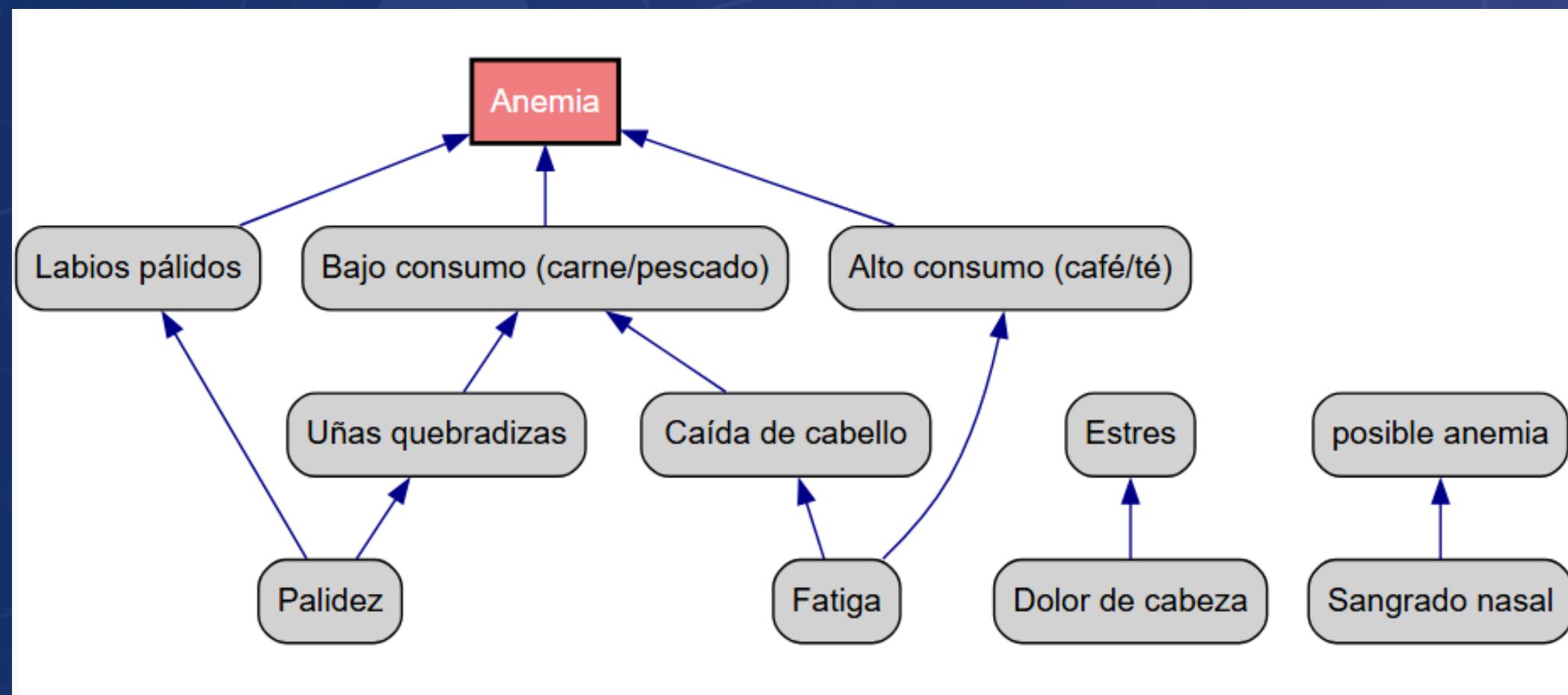


SENSORS

Entrada de los síntomas mediante el teclado.

next slide →

Grafo con información médica



next slide →

05

Código en python

Grafo con información médica

```
● ● ●  
1 # 1. Definicion del Grafo (Entorno)  
2 # Base de conocimiento para la anemia.  
3 base_conocimiento_anemia = {  
4     'Fatiga': ['Caida de cabello', 'Alto consumo (cafe/te)'],  
5     'Palidez': ['Labios palidos', 'Unas quebradizas'],  
6     'Caida de cabello': ['Bajo consumo (carne/pescado)'],  
7     'Unas quebradizas': ['Bajo consumo (carne/pescado)'],  
8     'Labios palidos': ['Anemia'],  
9     'Alto consumo (cafe/te)': ['Anemia'],  
10    'Bajo consumo (carne/pescado)': ['Anemia'],  
11    'Anemia': [],  
12    'Dolor de cabeza': ['Estres'],  
13    'Sangrado nasal': ['possible anemia'],  
14 }
```

06

Código en python

Búsqueda en amplitud BFS

```
● ● ●  
1 # 2. Implementacion de Busqueda en Amplitud (BFS)  
2 def busqueda_bfs(grafo, nodo_inicio, nodo_objetivo):  
3  
4     visitados = set([nodo_inicio]) #nodos visitados  
5     cola = deque([nodo_inicio]) #cola para BFS  
6     padres = {nodo_inicio: None} #para reconstruir la ruta  
7     orden = [] # Para registrar el orden de recorrido  
8  
9     while cola:    #mientras haya nodos en la cola hacer  
10        nodo_actual = cola.popleft() #sacar el primer nodo de la cola  
11        orden.append(nodo_actual) #guarda el nodo actual en orden de recorrido  
12  
13        if nodo_actual == nodo_objetivo: #si el nodo actual es el objetivo  
14            ruta = [] #ruta del recorrido de regreso  
15            n = nodo_actual #comenzar desde el nodo objetivo  
16            while n is not None: #mientras haya nodos en la ruta  
17                ruta.append(n) #agregar el nodo a la ruta  
18                n = padres.get(n) #mover al parent del nodo  
19            ruta.reverse() #invertir la ruta para obtener el orden correcto  
20            return True, orden, ruta #encontrado  
21  
22        for vecino in grafo.get(nodo_actual, []): # si no es el objetivo, explorar vecinos  
23            if vecino not in visitados: #si el vecino no ha sido visitado  
24                visitados.add(vecino) #marcar como visitado  
25                padres[vecino] = nodo_actual #establecer el parent  
26                cola.append(vecino) #agregar a la cola para explorar  
27  
28    return False, orden, [] # No encontrado
```

07

Código en python

Búsqueda en amplitud DFS

```
● ● ●  
1 # 2. Implementación de Búsqueda en Profundidad (DFS)  
2 def busqueda_dfs(grafo, nodo_inicio, nodo_objetivo):  
3  
4     visitados = set() #nodos visitados, inicialmente vacio  
5     pila = [nodo_inicio] # pila inicial para DFS  
6     padres = {nodo_inicio: None} # para reconstruir la ruta  
7     orden = [] #regresa el orden de recorrido  
8  
9     while pila:  
10        nodo_actual = pila.pop() #LIFO primero en entrar, ultimo en salir  
11  
12        if nodo_actual not in visitados: #si no ha sido visitado  
13            visitados.add(nodo_actual) #marcar como visitado  
14            orden.append(nodo_actual) #guardar el nodo actual en orden de recorrido  
15  
16        if nodo_actual == nodo_objetivo: #si el nodo actual es el objetivo  
17            ruta = [] #ruta del recorrido de regreso  
18            n = nodo_actual #comenzar desde el nodo objetivo  
19            while n is not None: #mientras haya nodos en la ruta  
20                ruta.append(n) #agregar el nodo a la ruta  
21                n = padres.get(n) #mover al parent del nodo  
22            ruta.reverse() #invertir la ruta para obtener el orden correcto  
23            return True, orden, ruta #encontrado  
24  
25        vecinos = grafo.get(nodo_actual, []) # obtener vecinos del nodo actual  
26  
27        for vecino in reversed(vecinos): #para cada vecino del nodo actual (reversed para mantener orden)  
28            if vecino not in visitados: #si el vecino no ha sido visitado  
29                padres[vecino] = nodo_actual #establecer el parent  
30                pila.append(vecino) #agregar a la pila para explorar  
31  
32    return False, orden, [] # No encontrado
```

08

Ejecución

Diagnóstico con amplitud BFS

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

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

Diagnóstico con amplitud DFS

```
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
```



Conclusiones

- *El sistema experto diseñado para la simulación de diagnóstico de la anemia se basa en un agente que interactua 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 mas corto pero puede omitir síntomas en su proceso.*

next slide



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*

next slide →





Gracias !

*Gracias por su atención durante la presentación
de agentes inteligentes.*