



2.AGENTES INTELIGENTES Y BÚSQUEDAS

M.SC. LUIS FERNANDO ESPINO BARRIOS
2020

I.AGENTES INTELIGENTES

AGENTES INTELIGENTES

- Son programas de inteligencia artificial basados en la lógica, se componen de sensores y actuadores para interactuar con su ambiente.

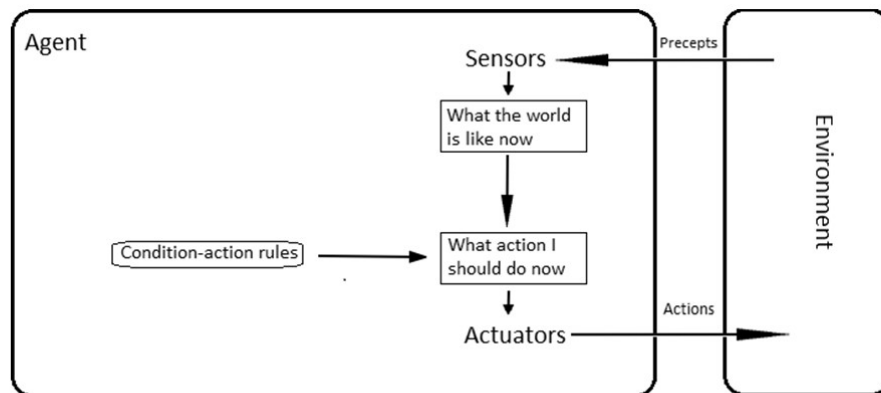
- Factores del ambiente
 - Completa o parcialmente observable
 - Determinísticos o estocástico
 - Discreto o continuo
 - Benigno o adversario

INCERTIDUMBRE

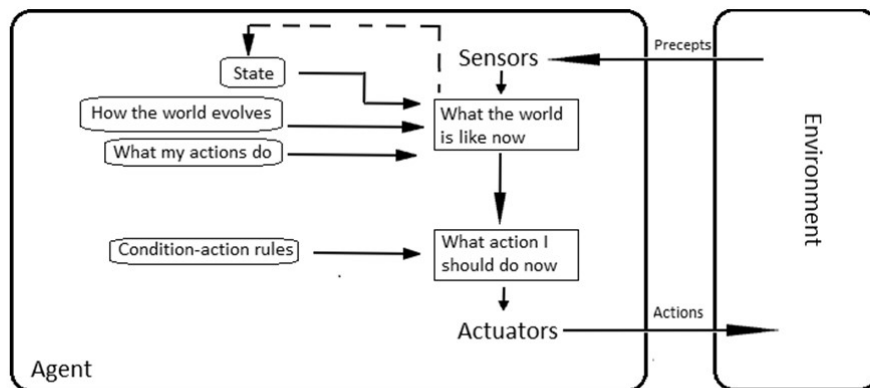
Uno de los problema de la inteligencia artificial es la falta de certeza para la toma directa de decisiones, esto debido a las siguientes razones

- Limitación de los sensores
- Presencia de adversarios
- Ambientes estocásticos
- Pereza
- Ignorancia

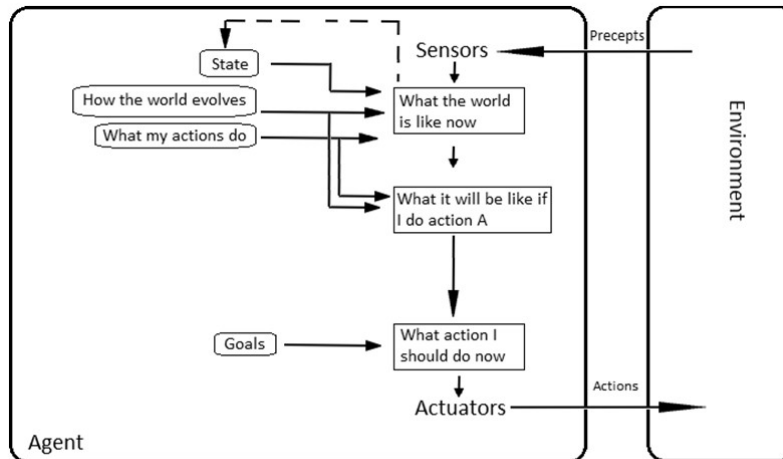
AGENTE REFLEJO SIMPLE



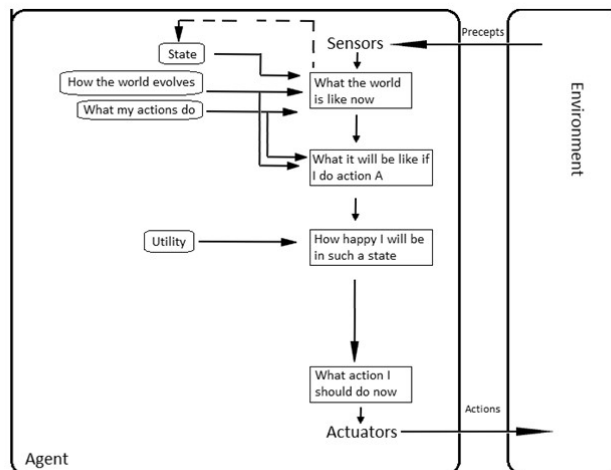
AGENTE BASADO EN MODELOS



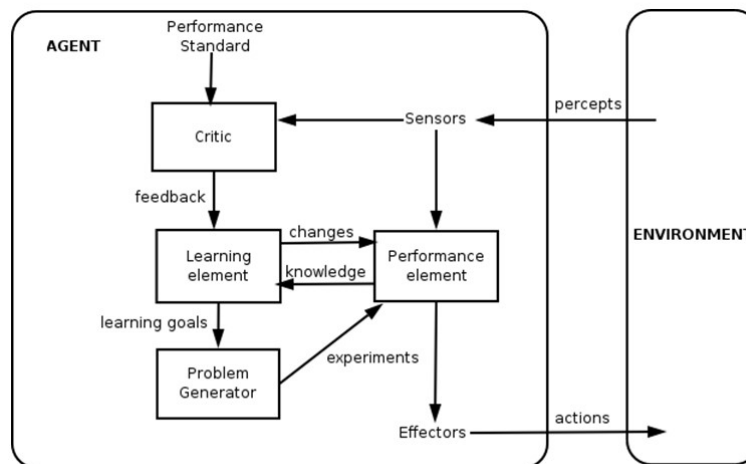
AGENTE BASADO EN OBJETIVOS



AGENTE BASADO EN UTILIDAD

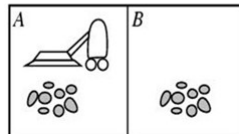


AGENTE DE APRENDIZAJE



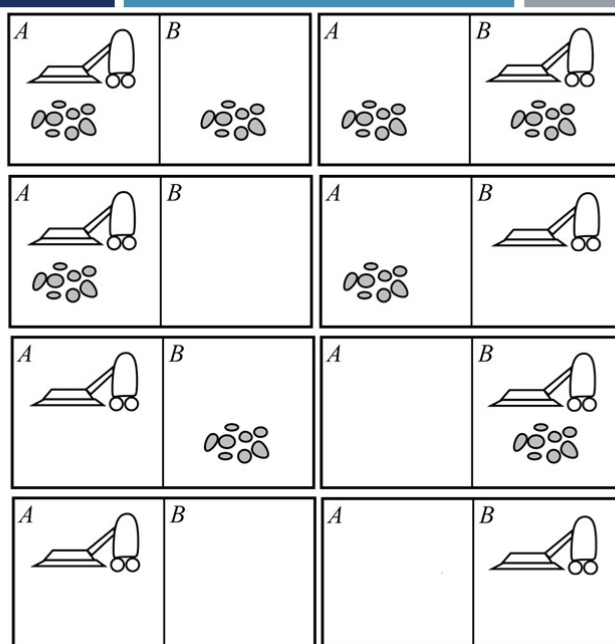
EJEMPLO DE AGENTE REFLEJO SIMPLE

The vacuum-cleaner world



```
• function REFLEX-VACUUM-AGENT  
  ([location,status]) returns an action  
  if status = Dirty then return Suck  
  else if location = A then return Right  
  else if location = B then return Left
```

Considerando el problema de la aspiradora, ¿cuántos estados diferentes se pueden generar?



```
def reflex_agent(location, state):
    if state=="DIRTY":
        return 'CLEAN'
    elif location=='A':
        return 'RIGHT'
    elif location=='B':
        return 'LEFT'
```

```
def test(states):
    while True:
        location = states[0]
        state = (states[2], states[1])[states[0]=='A']
        action = reflex_agent(location, state)
        print ("Location: "+location+" | Action: "+action)
        if action == "CLEAN":
            if location == 'A':
                states[1]="CLEAN"
            elif location == 'B':
                states[2]="CLEAN"
        elif action == "RIGHT":
            states[0]='B'
        elif action == "LEFT":
            states[0]='A'
        time.sleep(3)
```

```

import time

def reflex_agent(location, state):
    if state=="DIRTY":
        return 'CLEAN'
    elif location=='A':
        return 'RIGHT'
    elif location=='B':
        return 'LEFT'

def test(states):
    while True:
        location = states[0]
        state = (states[2], states[1])[states[0]=='A']
        action = reflex_agent(location, state)
        print ("Location: "+location+" | Action: "+action)
        if action == "CLEAN":
            if location == 'A':
                states[1]="CLEAN"
            elif location == 'B':
                states[2]="CLEAN"
        elif action == "RIGHT":
            states[0]='B'
        elif action == "LEFT":
            states[0]='A'
        time.sleep(3)

test(['A', 'DIRTY', 'DIRTY'])

```

```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 2
4) on win32
Type "copyright", "credits" or "license
>>>
===== RESTART: D:\pyth
Location: A | Action: CLEAN
Location: A | Action: RIGHT
Location: B | Action: CLEAN
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
|

```

CÓDIGO EN PYTHON PARA EL CURSO

<https://github.com/luisespino/ia>

EJERCICIO

- Considerar el problema de la aspiradora, modificar el código para que logre visitar los 8 estados.

2. BÚSQUEDAS

DEFINICIÓN DE BÚSQUEDA

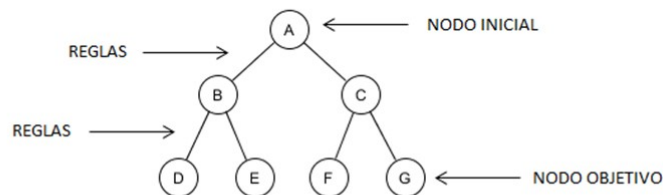
- Una búsqueda es un mecanismo para encontrar de manera automática una solución

ESPACIO DE ESTADOS

- Es el conjunto de información asociada a un problema específico
 - Definir posibles configuraciones
 - Establecer estado inicial
 - Especificar el estado de aceptación
 - Describir el conjunto de reglas

ÁRBOL DE BÚSQUEDA (CONCEPTUAL)

- Es una representación conceptual de las posibles soluciones que se forman mediante la aplicación de reglas.



TIPOS DE BÚSQUEDA

- Dependiendo si la búsqueda utiliza información generada durante el proceso, las búsquedas pueden ser:
 - Búsquedas no informadas
 - Búsquedas informadas

BÚSQUEDAS NO INFORMADAS

- Búsqueda por anchura – Breadth First Search
- Búsqueda por profundidad – Depth First Search

I. BÚSQUEDA POR ANCHURA

- Es una búsqueda que inicia por el nodo raíz y explora primero los nodos vecinos, luego expande el siguiente nivel de vecinos.
- Fue inventado por Edgarwd Moore en 1959.

- Conceptualmente es una cola FIFO (first-in first-out), en teoría de colas.
- En implementación de estructura de datos es una cola.

ALGORITMO

```
inicializar lista con nodo_inicial
mientras lista no esté vacía
    extraer primer_nodo de lista
    si primer_nodo es solución
        mostrar solución y salir
    generar sucesores
    agregar sucesores al final de la lista
mostrar no-solución y salir
```

EQUIVALENCIA ALGORITMO-CÓDIGO

```
inicializar lista con nodo_inicial
```

```
lista = [nodo_inicio]
```

```
mientras lista no esté vacía
```

```
while lista:
```

extraer primer_nodo de lista

nodo_actual = lista.pop(0)

si primer_nodo es solución mostrar solución y salir

if nodo_actual == nodo_fin:
 return print ("SOLUCION")

generar sucesores

temp = sucesores (nodo_actual)

agregar sucesores al final de la lista

if temp:
 lista.extend(temp)




mostrar no-solución y salir

`print ("NO-SOLUCION")`

```
def anchura(nodo_inicio, nodo_fin):
    lista = [nodo_inicio]
    while lista:
        nodo_actual = lista.pop(0)
        print (nodo_actual)
        if nodo_actual == nodo_fin:
            return print("SOLUCIÓN")
        temp = sucesores(nodo_actual)
        #temp.reverse()
        print (temp)
        if temp:
            lista.extend(temp)
            print(lista)
    print ("NO-SOLUCIÓN")
```

EJEMPLO I:

- Suponer una matriz de 3x3 que representa 9 posibles lugares que un ente puede desplazarse.
- El movimiento lo hace paso a paso alrededor de su actual posición.
- La finalidad es iniciar en una posición y finalizar en otra posición.

1		2	3
4		5	6
7		8	9 

```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Ma
4)] on win32
Type "copyright", "credits" or "lic
>>>
===== RESTART: D:\python\
1
[2, 4, 5]
[2, 4, 5]
2
[1, 3, 4, 5, 6]
[4, 5, 1, 3, 4, 5, 6]
4
[1, 2, 5, 7, 8]
[5, 1, 3, 4, 5, 6, 1, 2, 5, 7, 8]
5
[1, 2, 3, 4, 6, 7, 8, 9]

```

```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5,
6, 7, 9]
1
[2, 4, 5]
[2, 3, 4, 6, 7, 8, 9, 2, 4, 5, 2, 5, 6, 1, 2, 5, 7, 8, 1, 2, 3, 4, 6, 7, 8, 9, 2
, 3, 5, 8, 9, 2, 4, 5, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5, 6,
7, 9, 2, 4, 5]
2
[1, 3, 4, 5, 6]
[3, 4, 6, 7, 8, 9, 2, 4, 5, 2, 5, 6, 1, 2, 5, 7, 8, 1, 2, 3, 4, 6, 7, 8, 9, 2, 3
, 5, 8, 9, 2, 4, 5, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5, 6, 7,
9, 2, 4, 5, 1, 3, 4, 5, 6]
3
[2, 5, 6]
[4, 6, 7, 8, 9, 2, 4, 5, 2, 5, 6, 1, 2, 5, 7, 8, 1, 2, 3, 4, 6, 7, 8, 9, 2, 3, 5
, 8, 9, 2, 4, 5, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5, 6, 7, 9,
2, 4, 5, 1, 3, 4, 5, 6, 2, 5, 6]
4
[1, 2, 5, 7, 8]
[6, 7, 8, 9, 2, 4, 5, 2, 5, 6, 1, 2, 5, 7, 8, 1, 2, 3, 4, 6, 7, 8, 9, 2, 3, 5, 8
, 9, 2, 4, 5, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5, 6, 7, 9, 2,
4, 5, 1, 3, 4, 5, 6, 2, 5, 6, 1, 2, 5, 7, 8]
6
[2, 3, 5, 8, 9]
[7, 8, 9, 2, 4, 5, 2, 5, 6, 1, 2, 5, 7, 8, 1, 2, 3, 4, 6, 7, 8, 9, 2, 3, 5, 8, 9
, 2, 4, 5, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5, 6, 7, 9, 2, 4,
5, 1, 3, 4, 5, 6, 2, 5, 6, 1, 2, 5, 7, 8, 2, 3, 5, 8, 9]
7
[4, 5, 8]
[8, 9, 2, 4, 5, 2, 5, 6, 1, 2, 5, 7, 8, 1, 2, 3, 4, 6, 7, 8, 9, 2, 3, 5, 8, 9, 2
, 4, 5, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5, 6, 7, 9, 2, 4, 5,
1, 3, 4, 5, 6, 2, 5, 6, 1, 2, 5, 7, 8, 2, 3, 5, 8, 9, 4, 5, 8]
8
[4, 5, 6, 7, 9]
[9, 2, 4, 5, 2, 5, 6, 1, 2, 5, 7, 8, 1, 2, 3, 4, 6, 7, 8, 9, 2, 3, 5, 8, 9, 2, 4
, 5, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 7, 8, 9, 4, 5, 8, 4, 5, 6, 7, 9, 2, 4, 5, 1,
3, 4, 5, 6, 2, 5, 6, 1, 2, 5, 7, 8, 2, 3, 5, 8, 9, 4, 5, 8, 4, 5, 6, 7, 9]
9
SOLUCIÓN
>>> |
Ln: 70 Col: 4

```

2. BÚSQUEDA POR PROFUNDIDAD

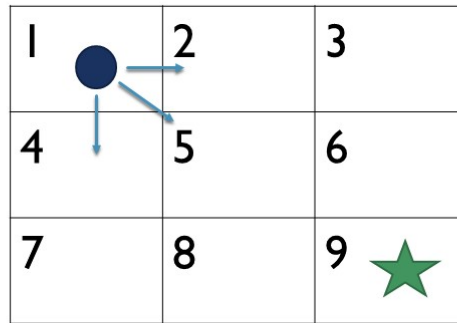
- Es un algoritmo que busca el nodo más profundo entre una de las ramas.
- Sus orígenes se remontan al siglo XIX por Charles Pierre Trémaux

- Conceptualmente es una cola LIFO (last-in first-out), en teoría de colas.
- En implementación de estructura de datos es una pila.

ALGORITMO

inicializar lista con nodo_inicial
mientras lista no esté vacía
 extraer primer_nodo de lista
 si primer_nodo es solución
 mostrar solución y salir
 generar sucesores
 agregar sucesores al inicio de la lista
mostrar no-solución y salir

```
def profundidad (nodo_inicio, nodo_fin):  
    lista = [nodo_inicio]  
    while lista:  
        nodo_actual = lista.pop(0)  
        print (nodo_actual)  
        if nodo_actual == nodo_fin:  
            #print(len(lista))  
            return print("SOLUCIÓN")  
        temp = sucesores(nodo_actual)  
        temp.reverse()  
        print (temp)  
        if temp:  
            temp.extend(lista)  
            lista = temp  
            print(lista)  
    print ("NO-SOLUCIÓN")
```



Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v. 4)] on win32

Type "copyright", "credits" or "license()" for more information

>>>

===== RESTART: D:\python\ia\2 anchura_profundidad.py

1

[5, 4, 2]

[5, 4, 2]

5

[9, 8, 7, 6, 4, 3, 2, 1]

[9, 8, 7, 6, 4, 3, 2, 1, 4, 2]

9

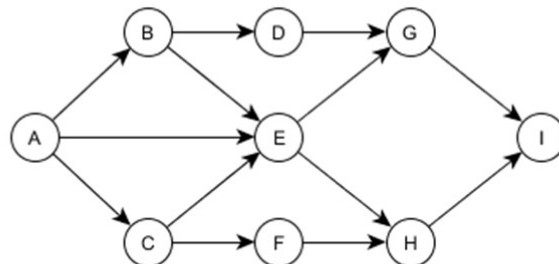
SOLUCIÓN

>>> |

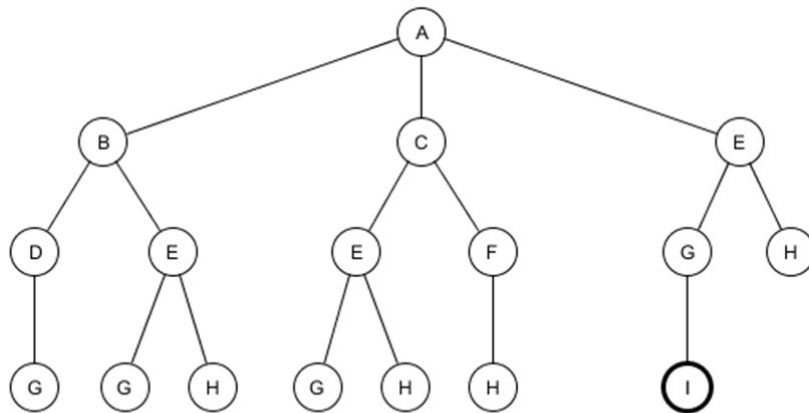
EJEMPLO 2:

- Suponer un grafo unidireccional de costo uniforme y considerar el nodo A como inicial y el nodo I como final.
- Elaborar el árbol de búsqueda por anchura mejorado con orden ascendente y el árbol de búsqueda por profundidad con orden ascendente.

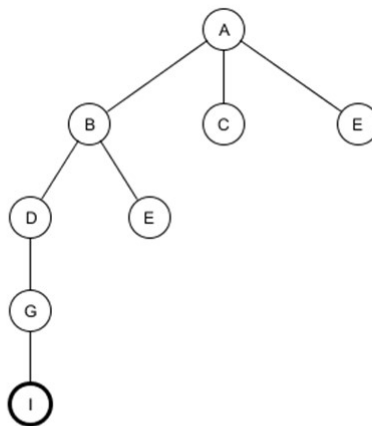
EJEMPLO 2:



ÁRBOL DE BÚSQUEDA POR ANCHURA MEJORADO



ÁRBOL DE BÚSQUEDA POR PROFUNDIDAD



FIN