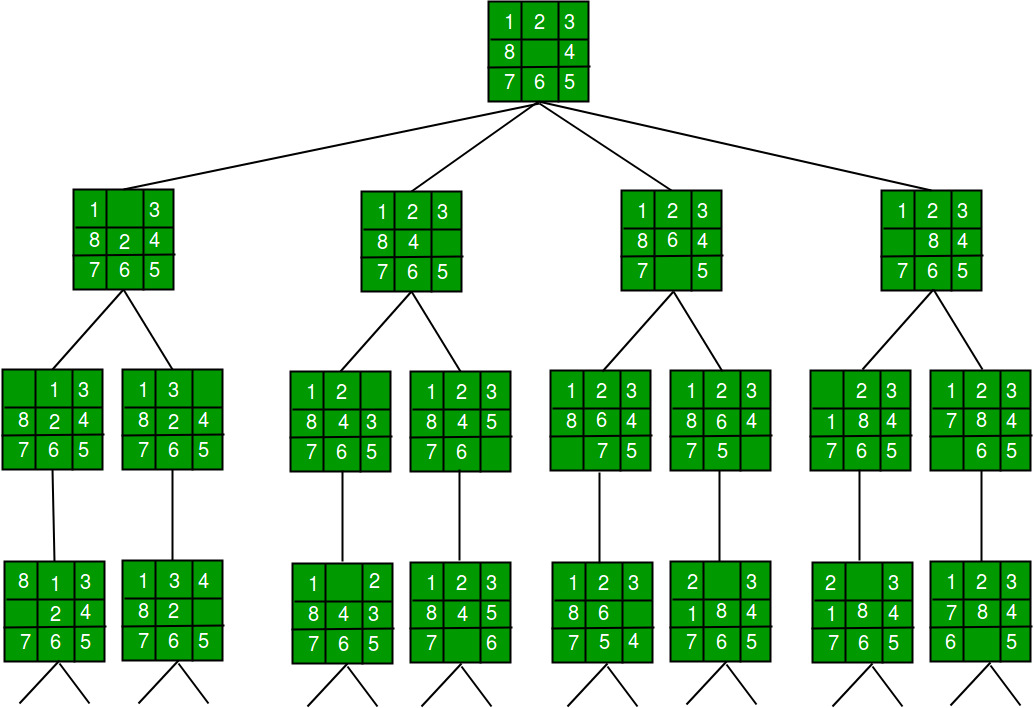
Q-LEARNING 8-PUZZLE



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Enunciado 2**](#_2c35568x6wg8)

[**2. Apartado 2**](#_mptzybqzhbp3)

[**2.1 Subapartado 1 2**](#_rjpj17ru25h3)

[**2.2 Código 2**](#_60o0rlbxshg)

[**2.3 Explicación 2**](#_3frf721zqlgo)

[**2.4 Subapartado 2 2**](#_cgrlc9c50fzf)

[**2.5 Código 2**](#_urko1tafyut3)

[**2.6 Explicación 3**](#_3heysd19e6zn)

[**2.7 Subapartado 3 3**](#_p4m843h40qnp)

[**2.8 Código 3**](#_f2119saf4207)

[**2.9 Explicación 4**](#_i2i4dzglkzia)

[**2.10 Subapartado 4 4**](#_xqwvw36lv451)

[**2.11 Código 4**](#_lh4thweyc9ho)

[**2.12 Explicación 4**](#_mwhk5ikyov4b)

[**3. Apartado 5**](#_p9ooysrjfclm)

[**3.1 Código 5**](#_cv3ac04oef1w)

[**3.2 Explicación 5**](#_ux9r5d9yj9rj)

[**3. Cálculo del aprendizaje 6**](#_uia2a1gi74vs)

[**3.1 Código 6**](#_gt9rcxhg8wdf)

[**3.2 Explicación 7**](#_nlo2pomcm5wa)

[**3.2 Resultado 8**](#_egeknlvnazcr)

## 

## **1. Objetivos de aprendizaje**

* Comprender y aplicar los conceptos de aprendizaje por refuerzo y el algoritmo Q-learning.
* Implementar un agente de aprendizaje por refuerzo que resuelva el problema del 8-puzzle
* Evaluar el rendimiento del agente y ajustar los hiperparámetros para mejorar la eficacia de aprendizaje

## **2. Apartado**

Se debe calcular la distancia a la solución desde cada estado examinando la tabla t.

### **2.1 Subapartado 1**

Se inicializan todos los estados con distancia -1.

### **2.2 Código**

def calculate\_steps\_to\_end(transitions, goal\_state):

steps\_to\_end = {}

queue = deque([(goal\_state, 0)]) # (state, distance)

visited = set()

while queue:

current\_state, distance = queue.popleft()

if current\_state in visited:

continue

visited.add(current\_state)

steps\_to\_end[current\_state] = distance - 1

for prev\_state in transitions.get(current\_state, []):

if prev\_state not in visited:

queue.append((prev\_state, distance + 1))

return steps\_to\_end

### **2.3 Explicación**

Como podemos ver, se marca steps\_to\_end[current\_state] = distance - 1 asigna a la clave correspondiente al estado actual (current\_state) en el diccionario steps\_to\_end un valor que es igual a la distancia calculada desde el estado objetivo (goal\_state) - 1.

### **2.4 Subapartado 2**

Se marca el estado solución con distancia 0

### **2.5 Código**

def calculate\_steps\_to\_end(transitions, goal\_state):

steps\_to\_end = {}

queue = deque([(goal\_state, 0)]) # (state, distance)

visited = set()

while queue:

current\_state, distance = queue.popleft()

if current\_state in visited:

continue

visited.add(current\_state)

steps\_to\_end[current\_state] = distance - 1

for prev\_state in transitions.get(current\_state, []):

if prev\_state not in visited:

queue.append((prev\_state, distance + 1))

return steps\_to\_end

### **2.6 Explicación**

En queue = deque([(goal\_state, 0)]) # (state, distance), se marca la distancia inicial del estado objetivo como 0. Este valor se propaga durante el proceso de cálculo de las distancias de los demás estados.

### **2.7 Subapartado 3**

Se marcan con distancia 1 los estados contiguos a la solución

### **2.8 Código**

def calculate\_steps\_to\_end(transitions, goal\_state):

steps\_to\_end = {}

queue = deque([(goal\_state, 0)]) # (state, distance)

visited = set()

while queue:

current\_state, distance = queue.popleft()

if current\_state in visited:

continue

visited.add(current\_state)

steps\_to\_end[current\_state] = distance - 1

for prev\_state in transitions.get(current\_state, []):

if prev\_state not in visited:

queue.append((prev\_state, distance + 1))

return steps\_to\_end

### **2.9 Explicación**

Los estados contiguos al current\_state se obtienen de la lista transitions.get(current\_state, []), que contiene los estados conectados al estado actual. Para cada estado contiguo prev\_state, se añade el valor incrementado +1, queue.append((prev\_state, distance + 1)).

### **2.10 Subapartado 4**

Se marcan con distancia k los estados (no marcados) contiguos a alguno de distancia k-1 se repite hasta que no se marque ninguno más.

### **2.11 Código**

def calculate\_steps\_to\_end(transitions, goal\_state):

steps\_to\_end = {}

queue = deque([(goal\_state, 0)]) # (state, distance)

visited = set()

while queue:

current\_state, distance = queue.popleft()

if current\_state in visited:

continue

visited.add(current\_state)

steps\_to\_end[current\_state] = distance - 1

for prev\_state in transitions.get(current\_state, []):

if prev\_state not in visited:

queue.append((prev\_state, distance + 1))

return steps\_to\_end

### **2.12 Explicación**

dentro del bucle while en la función calculate\_steps\_to\_end. Este bucle utiliza una cola (queue) para procesar estados en orden de distancia, marcando los estados contiguos no visitados con la distancia correspondiente.

Se marcan con distancia K los estados (no marcados) contiguos a alguno de distancia K−1:

Esto se realiza iterativamente a medida que los estados se agregan a la cola con una distancia acumulada. El algoritmo BFS garantiza que todos los estados de distancia K se procesen antes que los de K+1.

Se repite hasta que no se marque ninguno más El bucle continúa mientras haya estados en la cola, lo que asegura que todos los estados alcanzables se procesen:

## **3. Apartado**

Se usa esa información para compararla con el tamaño de path para cada estado de la tabla t.

### **3.1 Código**

def compare\_steps\_and\_paths(steps\_to\_end, path\_lengths):

discrepancies = []

for state, steps in steps\_to\_end.items():

path\_length = path\_lengths.get(state, None)

if path\_length is not None and steps != path\_length:

discrepancies.append((state, steps, path\_length))

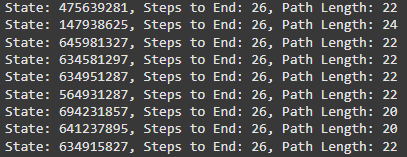
return discrepancies

### **3.2 Explicación**

Se usa la información de la distancia a la solución (steps\_to\_end) para compararla con el tamaño del camino (path\_lengths) desde el estado inicial hasta cada estado en la tabla T. La comparación ocurre en la función compare\_steps\_and\_paths.

Resultado si hay discrepancia:

State: , Steps to End: , Path Length:



Si no las hay:

No discrepancies found.

## **3. Cálculo del aprendizaje**

### **3.1 Código**

import numpy as np

path = []

# Mostrar los pasos para resolver el 8-puzzle desde un estado inicial aleatorio

def resolver\_8\_puzzle(T, Q, steps\_to\_end, estado\_inicial, estado\_objetivo="123456789"):

estado\_actual = estado\_inicial

pasos = 0

while estado\_actual != estado\_objetivo:

acciones = Q[estado\_actual]

# Filtrar acciones válidas

acciones\_validas = [a for a in Q[estado\_actual] if T[estado\_actual][a] is not None]

if not acciones\_validas:

break

accion\_mejor = max(acciones\_validas, key=acciones.get)

siguiente\_estado = T[estado\_actual][accion\_mejor]

if siguiente\_estado is None:

break

estado\_actual = siguiente\_estado

pasos += 1

if pasos > 100: # Limitar a 100 pasos para evitar bucles infinitos

break

# Calcular porcentaje de aprendizaje

if estado\_actual == estado\_objetivo:

distancia\_ideal = steps\_to\_end.get(estado\_inicial, None)

if distancia\_ideal is not None and pasos > 0:

return (distancia\_ideal / pasos) \* 100

return 0

# Simulación de episodios

def entrenar\_y\_evaluar(T, Q, steps\_to\_end, num\_episodios=1000000, evaluacion\_cada=100000):

aprendizaje\_porcentaje = []

for episodio in range(1, num\_episodios + 1):

# Simulación de un episodio (aquí puedes entrenar `Q` según tu lógica)

# Evaluar cada `evaluacion\_cada` episodios

if episodio % evaluacion\_cada == 0:

print(f"Evaluando en el episodio {episodio}...")

estados\_prueba = np.random.choice(list(T.keys()), size=10, replace=False)

porcentajes = [

resolver\_8\_puzzle(T, Q, steps\_to\_end, estado\_inicial=estado)

for estado in estados\_prueba

]

promedio\_aprendizaje = np.mean(porcentajes)

aprendizaje\_porcentaje.append((episodio, promedio\_aprendizaje))

print(f"Porcentaje de aprendizaje promedio: {promedio\_aprendizaje:.2f}%")

return aprendizaje\_porcentaje

# Ejemplo de uso

estado\_inicial = None # Deja como None para seleccionar un estado aleatorio

num\_episodios = 1000000 # Total de episodios

evaluacion\_cada = 100000 # Evaluar cada 100,000 episodios

# Asegúrate de tener `T`, `Q`, y `steps\_to\_end` definidos

resultados\_aprendizaje = entrenar\_y\_evaluar(T, Q, steps\_to\_end, num\_episodios, evaluacion\_cada)

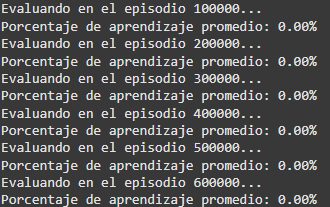
### **3.2 Explicación**

El código implementa un proceso para evaluar una solución al problema utilizando tablas de transición (T) y valores de recompensa/acción (Q). Comienza importando la librería numpy y definiendo variables iniciales como path, que probablemente se emplea para registrar los pasos del proceso. La función principal, resolver\_8\_puzzle, simula la resolución del rompecabezas desde un estado inicial hasta el objetivo (123456789). En cada iteración, filtra las acciones válidas basándose en las tablas Q y selecciona la mejor acción para avanzar al siguiente estado. Si no hay acciones válidas, si se alcanzan 100 pasos, o si se llega al estado objetivo, el bucle termina. Si se resuelve el puzzle, la función calcula un porcentaje de aprendizaje comparando los pasos ideales (steps\_to\_end) con los pasos realmente tomados.

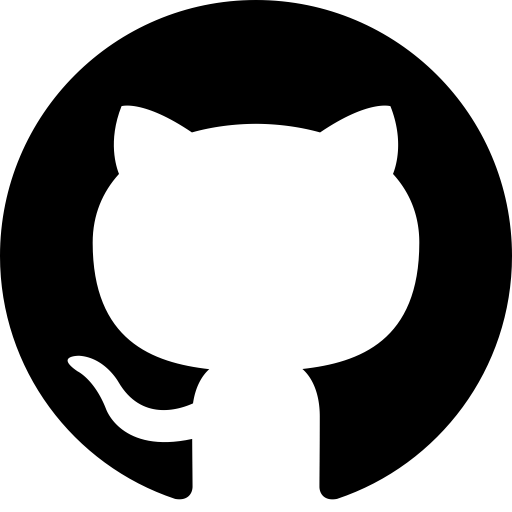
La función entrenar\_y\_evaluar ejecuta simulaciones en múltiples episodios para entrenar y evaluar el modelo. Cada cierto número de episodios (evaluacion\_cada), se evalúa el desempeño en estados iniciales aleatorios y se calcula el porcentaje promedio de aprendizaje, que se almacena e imprime para monitorear el progreso. Finalmente, un ejemplo de uso muestra cómo configurar parámetros como el número de episodios y realizar evaluaciones periódicas. Para ejecutarse correctamente, requiere que las tablas T, Q y steps\_to\_end estén definidas previamente. Este enfoque permite medir la eficiencia del aprendizaje y la solución al problema en un entorno controlado y reproducible.

### **3.2 Resultado**

Según el resultado mostrado creo que está entrenando pero el estado inicial no se encuentra en el campo de entrenamiento, ya que tarda alrededor de 5 minutos entrenando, sin embargo muestra un 0 de aprendizaje, ya que si tardase ese tiempo pese a no tener bien ajustadas las w (los pesos), habría un aprendizaje mínimo reflejado en el resultado que no podemos ver en este caso.



## **3. Github y Colab**

[](https://github.com/AdrianYArmas/IaBigData/tree/main/Pro/1%20Introducci%C3%B3n%20a%20la%20programaci%C3%B3n%20de%20Inteligencia%20Artificial/1.7%20%20Q-LEARNING%208-PUZZLE) [](https://colab.research.google.com/drive/1AkIpTZp1d4s79zeGVMZ_q3sy3d9NZOAb?usp=sharing)