

Se trata de crear un programa (agente) que encuentre la secuencia de acciones correctas para solucionar el problema de las Torres de Hanoi para el caso de 3 aros.

$$Q'(s_t, a_t) = (1 - \nu)Q(s_t, a_t) + \nu[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]$$

1 Crear la tabla Q de [27,6]

1.1 Inicializar la tabla Q a cero,

1.2 Crear una tabla R de recompensas con todo a cero menos la entrada (estado,transición) que lleve a la solución, que tendrá una recompensa de 100.

```
R=np.zeros((27,6),dtype=int)
```

```
R[17,1]=100
```

```
R[23,3]=100
```

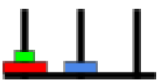
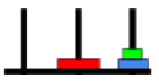
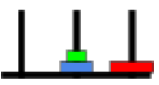
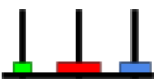
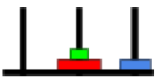




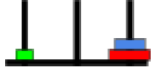
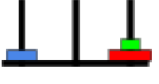




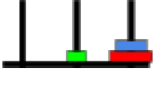

2 Crear la tabla T de [27,6] de transiciones (las imposibles las marcamos con -1).

3 Fase de entrenamiento, seleccionamos un estado y una acción (posible) de forma aleatoria y actualizamos su valor en la tabla Q (repetir muchas veces).

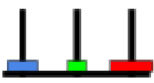

El factor de aprendizaje V puede inicializarse a 0.8 y el factor de descuento Y 0.95

ESTADOS ACCIONES	Tabla de transiciones	A → B a ₀	A → C a ₁	B → A a ₂	B → C a ₃	C → A a ₄	C → B a ₅
S0		S1	S2	-1	-1	-1	-1
S1		-1	S3	S0	S2	-1	-1
S2		S4	-1	-1	-1	S0	S1
S3		-1	-1	S5	S6	S1	-1
S4		-1	-1	S2	-1	S8	S7
S5		S3	S6	-1	-1	-1	S8
S6		S9	-1	-1	-1	S5	S3
S7		-1	S10	S8	S4	-1	-1

TAREA TORRES DE HANOI USANDO ALGORITMO Q-LEARNING PROGRAMACIÓN IA

S8		S7	S4	-1	S5	-1	-1
S9		-1	-1	S6	-1	S11	S12
S10		-1	-1	S13	S14	S7	-1
S11		S12	S9	-1	-1	-1	S15
S12		-1	-1	S11	S9	S16	-1
S13		S10	S14	-1	S17	-1	-1
S14		-1	-1	S18	-1	S13	S10
S15		S19	S20	-1	S11	-1	-1
S16		-1	S12	S21	S22	-1	-1
S17		S23	100 S26	-1	-1	-1	S13
S18		S14	-1	-1	-1	S24	S25
S19		-1	-1	S15	S20	-1	-1
S20		-1	-1	S22	-1	S15	S19
S21		S16	S22	-1	S24	-1	-1
S22		S20	-1	-1	-1	S21	S16
S23		-1	-1	S17	100 S26	S25	-1
S24		S25	S18	-1	-1	-1	S21

TAREA TORRES DE HANOI USANDO ALGORITMO Q-LEARNING PROGRAMACIÓN IA

S25		-1	S23	S24	S18	-1	-1
S26							

```
import numpy as np
Q=np.zeros((27,6),dtype=float)
```

```
R=np.zeros((27,6),dtype=int)
R[17,1]=100
R[23,3]=100
```

```
v=0.9 # Factor de aprendizaje learning rate
y=0.8 # Factor de descuento discount factor
```

Tabla de transiciones

```
import pandas as pd
df= pd.read_csv("T.csv",header=None)
T=df.to_numpy()
T
```

```
array([[ 1,  2, -1, -1, -1, -1],
       [-1,  3,  0,  2, -1, -1],
       [ 4, -1, -1, -1,  0,  1],
       [-1, -1,  5,  6,  1, -1],
       [-1, -1,  2, -1,  8,  7],
       [ 3,  6, -1, -1, -1,  8],
       [ 9, -1, -1, -1,  5,  3],
       [-1, 10,  8,  4, -1, -1],
       [ 7,  4, -1,  5, -1, -1],
       [-1, -1,  6, -1, 11, 12],
       [-1, -1, 13, 14,  7, -1],
       [12,  9, -1, -1, -1, 15],
       [-1, -1, 11,  9, 16, -1],
       [10, 14, -1, -1, -1, 15],
       [-1, -1, 18, -1, 13, 10],
       [19, 20, -1, 11, -1, -1],
       [-1, 12, 21, 22, -1, -1],
       [23, 26, -1, -1, -1, 13],
       [14, -1, -1, -1, 24, 25],
       [-1, -1, 15, 20, -1, -1],
       [-1, -1, 22, -1, 15, 19],
       [16, 22, -1, 24, -1, -1],
       [20, -1, -1, -1, 21, 16],
       [-1, -1, 17, 26, 25, -1],
       [25, 18, -1, -1, -1, 21],
       [-1, 23, 24, 18, -1, -1],
       [-1, -1, -1, -1, 17, 23]], dtype=int64)
```

```
1,2,-1,-1,-1,-1
-1,3,0,2,-1,-1
4,-1,-1,-1,0,1
-1,-1,5,6,1,-1
-1,-1,2,-1,8,7
3,6,-1,-1,-1,8
9,-1,-1,-1,5,3
-1,10,8,4,-1,-1
7,4,-1,5,-1,-1
-1,-1,6,-1,11,12
-1,-1,13,14,7,-1
12,9,-1,-1,-1,15
-1,-1,11,9,16,-1
10,14,-1,-1,-1,15
-1,-1,18,-1,13,10
19,20,-1,11,-1,-1
-1,12,21,22,-1,-1
23,26,-1,-1,-1,13
14,-1,-1,-1,24,25
-1,-1,15,20,-1,-1
-1,-1,22,-1,15,19
16,22,-1,24,-1,-1
20,-1,-1,-1,21,16
-1,-1,17,26,25,-1
25,18,-1,-1,-1,21
-1,23,24,18,-1,-1
-1,-1,-1,-1,17,23
```

La primera posibilidad es realizando caminos posibles
#Seleccionamos un estado al azar

```

s=0 #Partimos del estado inicial
entrenar=0
while(entrenar<100000):
    a=np.random.randint(6) # Acción aleatoria al azar número entero en [0,5]
    while T[s,a]==-1:
        a=np.random.randint(6)
    # T[s,a] es una transición posible
    siguiente=T[s,a] # Estado siguiente
    Q[s,a]=(1-v)*Q[s,a]+v*(R[s,a]+y*max(Q[siguiente,]))
    print (s,"-->",siguiente)
    if siguiente!=26:
        s=siguiente # Estado siguiente
    else:
        s=0
    entrenar+=1

```

La segunda posibilidad es probando transiciones posibles desde estados aleatorios.

```

#Seleccionamos un estado al azar
entrenar=0
while(entrenar<100000):
    s=np.random.randint(26) #estado aleatorio [0,25]
    a=np.random.randint(6) # Acción aleatoria al azar número entero en [0,5]
    while T[s,a]==-1:
        a=np.random.randint(6)
    # T[s,a] es una transición posible
    siguiente=T[s,a] # Estado siguiente
    Q[s,a]=(1-v)*Q[s,a]+v*(R[s,a]+y*max(Q[siguiente,]))
    entrenar+=1

```

TAREA TORRES DE HANOI USANDO ALGORITMO Q-LEARNING PROGRAMACIÓN IA

```
array([[ 13.4217728,  16.777216,  0.,  0.,  0.,  0. ],
       [ 0.,  13.4217728,  13.4217728,  16.777216,  0.,  0. ],
       [ 20.97152,  0.,  0.,  0.,  13.4217728,  13.4217728 ],
       [ 0.,  0.,  16.777216,  16.777216,  13.4217728,  0. ],
       [ 0.,  0.,  16.777216,  0.,  20.97152,  26.2144 ],
       [ 13.4217728,  16.777216,  0.,  0.,  0.,  20.97152 ],
       [ 20.97152,  0.,  0.,  0.,  16.777216,  13.4217728 ],
       [ 0.,  32.768,  20.97152,  20.97152,  0.,  0. ],
       [ 26.2144,  20.97152,  0.,  16.777216,  0.,  0. ],
       [ 0.,  0.,  16.777216,  0.,  20.97152,  26.2144 ],
       [ 0.,  0.,  32.768,  40.96,  26.2144,  0. ],
       [ 26.2144,  20.97152,  0.,  0.,  0.,  20.97152 ],
       [ 0.,  0.,  20.97152,  20.97152,  32.768,  0. ],
       [ 32.768,  40.96,  0.,  0.,  0.,  20.97152 ],
       [ 0.,  0.,  51.2,  0.,  32.768,  32.768 ],
       [ 20.97152,  26.2144,  0.,  20.97152,  0.,  0. ],
       [ 0.,  26.2144,  40.96,  32.768,  0.,  0. ],
       [ 80.,  100.,  0.,  0.,  0.,  32.768 ],
       [ 40.96,  0.,  0.,  0.,  51.2,  64. ],
       [ 0.,  0.,  20.97152,  26.2144,  0.,  0. ],
       [ 0.,  0.,  32.768,  0.,  20.97152,  20.97152 ],
       [ 32.768,  32.768,  0.,  51.2,  0.,  0. ],
       [ 26.2144,  0.,  0.,  0.,  40.96,  32.768 ],
       [ 0.,  0.,  80.,  100.,  64.,  0. ],
       [ 64.,  51.2,  0.,  0.,  0.,  40.96 ],
       [ 0.,  80.,  51.2,  51.2,  0.,  0. ],
       [ 0.,  0.,  0.,  0.,  0.,  0.]])
```

for t in range(0,26):

print ("s",t," accion:",np.argmax(Q[t,]))

```
s 0 accion: 1
s 1 accion: 3
s 2 accion: 0
s 3 accion: 2
s 4 accion: 5
s 5 accion: 5
s 6 accion: 0
s 7 accion: 1
s 8 accion: 0
s 9 accion: 5
s 10 accion: 3
s 11 accion: 0
s 12 accion: 4
s 13 accion: 1
s 14 accion: 2
s 15 accion: 1
s 16 accion: 2
s 17 accion: 1
s 18 accion: 5
s 19 accion: 3
s 20 accion: 2
s 21 accion: 3
s 22 accion: 4
s 23 accion: 3
s 24 accion: 0
s 25 accion: 1
```