

# Algoritmos\_Elvis-Lopez\_AG1

July 6, 2025

## 1 Algoritmos - Actividad Guiada 1

Nombre: Elvis Xavier López Yajamin

<https://github.com/xzavierlopez/03MIAR-Algoritmos-de-Optimizacion/tree/master/AG1>

### 1.1 Torres de Hanoi con Divide y vencerás

```
[ ]: def Torres_Hanoi(N, desde, hasta):  
    if N ==1 :  
        print("Lleva la ficha " ,desde , " hasta " , hasta )  
  
    else:  
        #Torres_Hanoi(N-1, desde, 6-desde-hasta )  
        Torres_Hanoi(N-1, desde, 6-desde-hasta )  
        print("Lleva la ficha " ,desde , " hasta " , hasta )  
        #Torres_Hanoi(N-1,6-desde-hasta, hasta )  
        Torres_Hanoi(N-1, 6-desde-hasta , hasta )  
  
Torres_Hanoi(3, 1 , 3)
```

```
Lleva la ficha  1  hasta  3  
Lleva la ficha  1  hasta  2  
Lleva la ficha  3  hasta  2  
Lleva la ficha  1  hasta  3  
Lleva la ficha  2  hasta  1  
Lleva la ficha  2  hasta  3  
Lleva la ficha  1  hasta  3
```

```
[ ]: #Sucesión_de_Fibonacci  
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci  
#Calculo del termino n-simo de la sucesión de Fibonacci  
def Fibonacci(N:int):  
    if N < 2:  
        return 1  
    else:  
        return Fibonacci(N-1)+Fibonacci(N-2)
```

```
Fibonacci(5)
```

```
[ ]: 8
```

## 1.2 Devolución de cambio por técnica voraz

```
[ ]: def cambio_monedas(N, SM):  
    SOLUCION = [0]*len(SM)    #SOLUCION = [0,0,0,0,..]  
    ValorAcumulado = 0  
  
    for i,valor in enumerate(SM):  
        monedas = (N-ValorAcumulado)//valor  
        SOLUCION[i] = monedas  
        ValorAcumulado = ValorAcumulado + monedas*valor  
  
    if ValorAcumulado == N:  
        return SOLUCION  
  
cambio_monedas(15,[25,10,5,1])
```

```
[ ]: [0, 1, 1, 0]
```

## 1.3 N-Reinas por técnica de vuelta atrás

```
[ ]: def escribe(S):  
    n = len(S)  
    for x in range(n):  
        print("")  
        for i in range(n):  
            if S[i] == x+1:  
                print(" X ", end="")  
            else:  
                print(" - ", end="")  
  
def es_prometedora(SOLUCION,etapa):  
    #print(SOLUCION)  
    #Si la solución tiene dos valores iguales no es valida => Dos reinas en la_  
    ↪misma fila  
    for i in range(etapa+1):  
        #print("El valor " + str(SOLUCION[i]) + " está " + str(SOLUCION.  
        ↪count(SOLUCION[i])) + " veces")  
        if SOLUCION.count(SOLUCION[i]) > 1:  
            return False
```

```

    #Verifica las diagonales
    for j in range(i+1, etapa +1 ):
        #print("Comprobando diagonal de " + str(i) + " y " + str(j))
        if abs(i-j) == abs(SOLUCION[i]-SOLUCION[j]) : return False
    return True

def reinas(N, solucion=[], etapa=0):
    if len(solucion) == 0:
        solucion=[0 for i in range(N)]

    for i in range(1, N+1):
        solucion[etapa] = i

        if es_prometedora(solucion, etapa):
            if etapa == N-1:
                print(solucion)
                escribe(solucion)
                print()
            else:
                escribe(solucion)
                print()
                print("#####")
                print()
                reinas(N, solucion, etapa+1)
        else:
            escribe(solucion)
            print()
            print("*****")
            print()
            None

        solucion[etapa] = 0

reinas(4)

```

```

X - - -
- - - -
- - - -
- - - -
#####

```

```

X X - -
- - - -

```

```

- - - -
- - - -
*****

```

```

X  -  -  -
-  X  -  -
-  -  -  -
-  -  -  -
*****

```

```

X  -  -  -
-  -  -  -
-  X  -  -
-  -  -  -
#####

```

```

X  -  X  -
-  -  -  -
-  X  -  -
-  -  -  -
*****

```

```

X  -  -  -
-  -  X  -
-  X  -  -
-  -  -  -
*****

```

```

X  -  -  -
-  -  -  -
-  X  X  -
-  -  -  -
*****

```

```

X  -  -  -
-  -  -  -
-  X  -  -
-  -  X  -
*****

```

```

X  -  -  -

```

```

- - - -
- - - -
- X - -
#####

```

```

X - X -
- - - -
- - - -
- X - -
*****

```

```

X - - -
- - X -
- - - -
- X - -
#####

```

```

X - - X
- - X -
- - - -
- X - -
*****

```

```

X - - -
- - X X
- - - -
- X - -
*****

```

```

X - - -
- - X -
- - - X
- X - -
*****

```

```

X - - -
- - X -
- - - -
- X - X
*****

```

```

X  -  -  -
-  -  -  -
-  -  X  -
-  X  -  -
*****

```

```

X  -  -  -
-  -  -  -
-  -  -  -
-  X  X  -
*****

```

```

-  -  -  -
X  -  -  -
-  -  -  -
-  -  -  -
#####

```

```

-  X  -  -
X  -  -  -
-  -  -  -
-  -  -  -
*****

```

```

-  -  -  -
X  X  -  -
-  -  -  -
-  -  -  -
*****

```

```

-  -  -  -
X  -  -  -
-  X  -  -
-  -  -  -
*****

```

```

-  -  -  -
X  -  -  -
-  -  -  -
-  X  -  -
#####

```

```

- - X -
X - - -
- - - -
- X - -
#####

```

```

- - X X
X - - -
- - - -
- X - -
*****

```

```

- - X -
X - - X
- - - -
- X - -
*****

```

[2, 4, 1, 3]

```

- - X -
X - - -
- - - X
- X - -

```

```

- - X -
X - - -
- - - -
- X - X
*****

```

```

- - - -
X - X -
- - - -
- X - -
*****

```

```

- - - -
X - - -
- - X -
- X - -
*****

```

```

- - - -
X - - -
- - - -
- X X -
*****

```

```

- - - -
- - - -
X - - -
- - - -
#####

```

```

- X - -
- - - -
X - - -
- - - -
#####

```

```

- X X -
- - - -
X - - -
- - - -
*****

```

```

- X - -
- - X -
X - - -
- - - -
*****

```

```

- X - -
- - - -
X - X -
- - - -
*****

```

```

- X - -
- - - -
X - - -
- - X -
#####

```



```

-  X  -  X
-  -  -  -
X  -  -  -
-  -  X  -
*****

```

[3, 1, 4, 2]

```

-  X  -  -
-  -  -  X
X  -  -  -
-  -  X  -

-  X  -  -
-  -  -  -
X  -  -  X
-  -  X  -
*****

```

```

-  X  -  -
-  -  -  -
X  -  -  -
-  -  X  X
*****

```

```

-  -  -  -
-  X  -  -
X  -  -  -
-  -  -  -
*****

```

```

-  -  -  -
-  -  -  -
X  X  -  -
-  -  -  -
*****

```

```

-  -  -  -
-  -  -  -
X  -  -  -
-  X  -  -
*****

```

```

- - - -
- - - -
- - - -
X - - -
#####

```

```

- X - -
- - - -
- - - -
X - - -
#####

```

```

- X X -
- - - -
- - - -
X - - -
*****

```

```

- X - -
- - X -
- - - -
X - - -
*****

```

```

- X - -
- - - -
- - X -
X - - -
#####

```

```

- X - X
- - - -
- - X -
X - - -
*****

```

```

- X - -
- - - X
- - X -
X - - -

```

\*\*\*\*\*

```
-  X  -  -  
-  -  -  -  
-  -  X  X  
X  -  -  -
```

\*\*\*\*\*

```
-  X  -  -  
-  -  -  -  
-  -  X  -  
X  -  -  X
```

\*\*\*\*\*

```
-  X  -  -  
-  -  -  -  
-  -  -  -  
X  -  X  -
```

\*\*\*\*\*

```
-  -  -  -  
-  X  -  -  
-  -  -  -  
X  -  -  -
```

#####

```
-  -  X  -  
-  X  -  -  
-  -  -  -  
X  -  -  -
```

\*\*\*\*\*

```
-  -  -  -  
-  X  X  -  
-  -  -  -  
X  -  -  -
```

\*\*\*\*\*

```
-  -  -  -  
-  X  -  -  
-  -  X  -
```

```

X  -  -  -
*****

```

```

-  -  -  -
-  X  -  -
-  -  -  -
X  -  X  -
*****

```

```

-  -  -  -
-  -  -  -
-  X  -  -
X  -  -  -
*****

```

```

-  -  -  -
-  -  -  -
-  -  -  -
X  X  -  -
*****

```

## 1.4 Viaje por el río. Programación dinámica

```

[ ]: TARIFAS = [
    [0,5,4,3,999,999,999],
    [999,0,999,2,3,999,11],
    [999,999, 0,1,999,4,10],
    [999,999,999, 0,5,6,9],
    [999,999, 999,999,0,999,4],
    [999,999, 999,999,999,0,3],
    [999,999,999,999,999,999,0]
]

#####
def Precios(TARIFAS):
    #####
    #Total de Nodos
    N = len(TARIFAS[0])

    #Inicialización de la tabla de precios
    PRECIOS = [ [9999]*N for i in [9999]*N]

```

```

RUTA = [ [""]*N for i in range(N)]

for i in range(0,N-1):
    RUTA[i][i] = i          #Para ir de i a i se "pasa por i"
    PRECIOS[i][i] = 0       #Para ir de i a i se se paga 0
    for j in range(i+1, N):
        MIN = TARIFAS[i][j]
        RUTA[i][j] = i

        for k in range(i, j):
            if PRECIOS[i][k] + TARIFAS[k][j] < MIN:
                MIN = min(MIN, PRECIOS[i][k] + TARIFAS[k][j] )
                RUTA[i][j] = k          #Anota que para ir de i a j hay que pasar
    por k
        PRECIOS[i][j] = MIN

    return PRECIOS,RUTA
#####

PRECIOS,RUTA = Precios(TARIFAS)
#print(PRECIOS[0][6])

print("PRECIOS")
for i in range(len(TARIFAS)):
    print(PRECIOS[i])

print("\nRUTA")
for i in range(len(TARIFAS)):
    print(RUTA[i])

#Determinar la ruta con Recursividad
def calcular_ruta(RUTA, desde, hasta):
    if desde == hasta:
        #print("Ir a :" + str(desde))
        return ""
    else:
        return str(calcular_ruta( RUTA, desde, RUTA[desde][hasta])) + \
            ',' + \
            str(RUTA[desde][hasta] \
            )

print("\nLa ruta es:")
calcular_ruta(RUTA, 0,6)

```

PRECIOS  
[0, 5, 4, 3, 8, 8, 11]

```
[9999, 0, 999, 2, 3, 8, 7]
[9999, 9999, 0, 1, 6, 4, 7]
[9999, 9999, 9999, 0, 5, 6, 9]
[9999, 9999, 9999, 9999, 0, 999, 4]
[9999, 9999, 9999, 9999, 9999, 0, 3]
[9999, 9999, 9999, 9999, 9999, 9999, 9999]
```

RUTA

```
[0, 0, 0, 0, 1, 2, 5]
['', 1, 1, 1, 1, 3, 4]
['', '', 2, 2, 3, 2, 5]
['', '', '', 3, 3, 3, 3]
['', '', '', '', 4, 4, 4]
['', '', '', '', '', 5, 5]
['', '', '', '', '', '', '']
```

La ruta es:

```
[ ]: ',0,2,5'
```