Algoritmos - Actividad Guiada 1

Nombre: Raul Reyero Diez
URL: https://colab.research.google.com/drive/1lJqfBFQFbfb39vkg9KlMqWj527tXl9uY?usp=sharing

http:github.com/rauxxxx/Algoritmos..

Torres de Hanoi con Divide y vencerás

In [5]:
```
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
```

Torres de Hanoi con Divide y vencerás MODIFICADO

Se han realizado las siguientes modificaciones:

- Agregada una variable movimientos para mantener en una lista los mismos
- Separación de la parte de implementación y ejemplificación de uso

In [5]:
```
def Torres_Hanoi(N, origen, destino, movimientos):
    if N == 1:
        movimientos.append((origen, destino))
    else:
        auxiliar = 6 - origen - destino
        Torres_Hanoi(N-1, origen, auxiliar, movimientos)
        movimientos.append((origen, destino))
        Torres_Hanoi(N-1, auxiliar, destino, movimientos)

# Ejemplo de uso
movs = []
```

```
Torres_Hanoi(3, 1, 3, movs)
for m in movs:
    print(f"Lleva la ficha {m[0]} hasta {m[1]}")
```

```
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
```

Torres de Hanoi con pila (Mejora en escalabilidad)

Se ha realizado esta versión para casos grandes que la recursión pudiera presentar fallas y para mejorar la escalabilidad de la solución. No se puede considerar mejor versión al 100% pues es menos intuitiva y dificil de manejar.

In [4]:
```
def Torres_Hanoi_iterativa(N, origen, destino):
    stack = []
    movimientos = []

    auxiliar = 6 - origen - destino
    stack.append((N, origen, destino, auxiliar, False))

    while stack:
        n, desde, hasta, aux, procesado = stack.pop()

        if n == 1:
            movimientos.append((desde, hasta))
        else:
            if not procesado:
                # Simula el recorrido recursivo en orden: izquierda, raíz, derec
                stack.append((n, desde, hasta, aux, True))  # Marcar como proces
                stack.append((n-1, desde, aux, hasta, False))  # Parte izquierda
            else:
                movimientos.append((desde, hasta))  # Raíz
                stack.append((n-1, aux, hasta, desde, False))  # Parte derecha

    return movimientos

# Ejemplo de uso
movs = Torres_Hanoi_iterativa(3, 1, 3)
for m in movs:
    print(f"Lleva la ficha {m[0]} hasta {m[1]}")
```

```
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

In [4]:
```
#Sucesión_de_Fibonacci
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
#Calculo del termino n-simo de la suscesión de Fibonacci
def Fibonacci(N:int):
  if N < 2:
    return 1
  else:
    return Fibonacci(N-1)+Fibonacci(N-2)

Fibonacci(5)
```

Out[4]: 8

Sucesión de Fibonacci MODIFICADO

Se han realizado las siguientes modificaciones:

- Técnica de memoización, la cual almacena resultados ya calculados, permitiendo que la complejidad pase de ser O(2**N) a O(N)

In [ ]:
```
#Sucesión_de_Fibonacci
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
#Calculo del termino n-simo de la suscesión de Fibonacci

#importación de la libreria que nos permite aplicar la técnica de memoización
from functools import lru_cache

@lru_cache(maxsize=None)
def Fibonacci_memo(N: int) -> int:
    if N < 2:
        return 1
    return Fibonacci_memo(N - 1) + Fibonacci_memo(N - 2)

#Ejemplo de uso
print(Fibonacci_memo(100))  # Mucho más rápido que la versión original
```

```
573147844013817084101
```

Sucesión de Fibonacci versión Iterativa

Esta versión se ha añadido pues evita el uso de la pila y acelera el cálculo (más eficiente en espacio)

In [10]:
```
#Sucesión_de_Fibonacci
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
#Calculo del termino n-simo de la suscesión de Fibonacci


def Fibonacci_iter(N: int) -> int:
    if N < 2:
        return 1
    a, b = 1, 1
    for _ in range(2, N + 1):
        a, b = b, a + b
    return b
```

```
#Ejemplo de uso
print(Fibonacci_iter(100))
```

```
573147844013817084101
```

Devolución de cambio por técnica voraz

In [3]:
```
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])
```

Out[3]:
```
[0, 1, 1, 0]
```

Devolución de cambio por técnica voraz MODIFICADO

Se ha implementado la siguiente mejora:

- Implementación explícita de devolución de 'None' cuando no existe combinación resultante de cambio

In [18]:
```
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


    # Si no se alcanza el valor exacto
      return None

print(cambio_monedas(15,[25,10,5,1]))
```

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

N-Reinas por técnica de vuelta atrás

In [7]:
```
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 mis
  for i in range(etapa+1):
    #print("El valor " + str(SOLUCION[i]) + " está " +  str(SOLUCION.count(SOLUC
    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:
        reinas(N, solucion, etapa+1)
    else:
      None

    solucion[etapa] = 0

reinas(8)
```

```
[1, 5, 8, 6, 3, 7, 2, 4]

[1, 6, 8, 3, 7, 4, 2, 5]

[1, 7, 4, 6, 8, 2, 5, 3]

[1, 7, 5, 8, 2, 4, 6, 3]

[2, 4, 6, 8, 3, 1, 7, 5]

[2, 5, 7, 1, 3, 8, 6, 4]

[2, 5, 7, 4, 1, 8, 6, 3]

[2, 6, 1, 7, 4, 8, 3, 5]

[2, 6, 8, 3, 1, 4, 7, 5]

[2, 7, 3, 6, 8, 5, 1, 4]

[2, 7, 5, 8, 1, 4, 6, 3]

[2, 8, 6, 1, 3, 5, 7, 4]

[3, 1, 7, 5, 8, 2, 4, 6]

[3, 5, 2, 8, 1, 7, 4, 6]

[3, 5, 2, 8, 6, 4, 7, 1]

[3, 5, 7, 1, 4, 2, 8, 6]

[3, 5, 8, 4, 1, 7, 2, 6]

[3, 6, 2, 5, 8, 1, 7, 4]

[3, 6, 2, 7, 1, 4, 8, 5]

[3, 6, 2, 7, 5, 1, 8, 4]

[3, 6, 4, 1, 8, 5, 7, 2]

[3, 6, 4, 2, 8, 5, 7, 1]

[3, 6, 8, 1, 4, 7, 5, 2]

[3, 6, 8, 1, 5, 7, 2, 4]

[3, 6, 8, 2, 4, 1, 7, 5]

[3, 7, 2, 8, 5, 1, 4, 6]

[3, 7, 2, 8, 6, 4, 1, 5]

[3, 8, 4, 7, 1, 6, 2, 5]

[4, 1, 5, 8, 2, 7, 3, 6]

[4, 1, 5, 8, 6, 3, 7, 2]
```

```
[4, 2, 5, 8, 6, 1, 3, 7]

[4, 2, 7, 3, 6, 8, 1, 5]

[4, 2, 7, 3, 6, 8, 5, 1]

[4, 2, 7, 5, 1, 8, 6, 3]

[4, 2, 8, 5, 7, 1, 3, 6]

[4, 2, 8, 6, 1, 3, 5, 7]

[4, 6, 1, 5, 2, 8, 3, 7]

[4, 6, 8, 2, 7, 1, 3, 5]

[4, 6, 8, 3, 1, 7, 5, 2]

[4, 7, 1, 8, 5, 2, 6, 3]

[4, 7, 3, 8, 2, 5, 1, 6]

[4, 7, 5, 2, 6, 1, 3, 8]

[4, 7, 5, 3, 1, 6, 8, 2]

[4, 8, 1, 3, 6, 2, 7, 5]

[4, 8, 1, 5, 7, 2, 6, 3]

[4, 8, 5, 3, 1, 7, 2, 6]

[5, 1, 4, 6, 8, 2, 7, 3]

[5, 1, 8, 4, 2, 7, 3, 6]

[5, 1, 8, 6, 3, 7, 2, 4]

[5, 2, 4, 6, 8, 3, 1, 7]

[5, 2, 4, 7, 3, 8, 6, 1]

[5, 2, 6, 1, 7, 4, 8, 3]

[5, 2, 8, 1, 4, 7, 3, 6]

[5, 3, 1, 6, 8, 2, 4, 7]

[5, 3, 1, 7, 2, 8, 6, 4]

[5, 3, 8, 4, 7, 1, 6, 2]

[5, 7, 1, 3, 8, 6, 4, 2]

[5, 7, 1, 4, 2, 8, 6, 3]

[5, 7, 2, 4, 8, 1, 3, 6]

[5, 7, 2, 6, 3, 1, 4, 8]
```

```
[5, 7, 2, 6, 3, 1, 8, 4]

[5, 7, 4, 1, 3, 8, 6, 2]

[5, 8, 4, 1, 3, 6, 2, 7]

[5, 8, 4, 1, 7, 2, 6, 3]

[6, 1, 5, 2, 8, 3, 7, 4]

[6, 2, 7, 1, 3, 5, 8, 4]

[6, 2, 7, 1, 4, 8, 5, 3]

[6, 3, 1, 7, 5, 8, 2, 4]

[6, 3, 1, 8, 4, 2, 7, 5]

[6, 3, 1, 8, 5, 2, 4, 7]

[6, 3, 5, 7, 1, 4, 2, 8]

[6, 3, 5, 8, 1, 4, 2, 7]

[6, 3, 7, 2, 4, 8, 1, 5]

[6, 3, 7, 2, 8, 5, 1, 4]

[6, 3, 7, 4, 1, 8, 2, 5]

[6, 4, 1, 5, 8, 2, 7, 3]

[6, 4, 2, 8, 5, 7, 1, 3]

[6, 4, 7, 1, 3, 5, 2, 8]

[6, 4, 7, 1, 8, 2, 5, 3]

[6, 8, 2, 4, 1, 7, 5, 3]

[7, 1, 3, 8, 6, 4, 2, 5]

[7, 2, 4, 1, 8, 5, 3, 6]

[7, 2, 6, 3, 1, 4, 8, 5]

[7, 3, 1, 6, 8, 5, 2, 4]

[7, 3, 8, 2, 5, 1, 6, 4]

[7, 4, 2, 5, 8, 1, 3, 6]

[7, 4, 2, 8, 6, 1, 3, 5]

[7, 5, 3, 1, 6, 8, 2, 4]

[8, 2, 4, 1, 7, 5, 3, 6]

[8, 2, 5, 3, 1, 7, 4, 6]
```

```
[8, 3, 1, 6, 2, 5, 7, 4]

[8, 4, 1, 3, 6, 2, 7, 5]
```

N-Reinas por técnica de vuelta atrás MODIFICADO

En este código es, de momento, en el que más modificaciones se han reaizado buscando darle un lavado de cara:

- Se ha evitado utilizar el .count() dentro de la funcion 'es_prometedora' pues se revisaba cada iteración del bucle
- Utilizacion de la indexacion 1-based en lugar de la 0-baed (no es la más óptima en Python)
- La función 'es_prometedora' se ha reescrito comparando solo la última reina, haciendo más limpio y eficiente
- No modificamos la lista original

In [8]:
```python
def es_prometedora(solucion, etapa):
    for i in range(etapa):
        if solucion[i] == solucion[etapa] or abs(etapa - i) == abs(solucion[etap
            return False
    return True



def reinas(N, solucion=None, etapa=0):
    if solucion is None:
        solucion = [0] * N

    for fila in range(N):
        solucion[etapa] = fila
        if es_prometedora(solucion, etapa):
            if etapa == N - 1:
                print(solucion)
            else:
                reinas(N, solucion, etapa + 1)

reinas(8)
```

```
[0, 4, 7, 5, 2, 6, 1, 3]
[0, 5, 7, 2, 6, 3, 1, 4]
[0, 6, 3, 5, 7, 1, 4, 2]
[0, 6, 4, 7, 1, 3, 5, 2]
[1, 3, 5, 7, 2, 0, 6, 4]
[1, 4, 6, 0, 2, 7, 5, 3]
[1, 4, 6, 3, 0, 7, 5, 2]
[1, 5, 0, 6, 3, 7, 2, 4]
[1, 5, 7, 2, 0, 3, 6, 4]
[1, 6, 2, 5, 7, 4, 0, 3]
[1, 6, 4, 7, 0, 3, 5, 2]
[1, 7, 5, 0, 2, 4, 6, 3]
[2, 0, 6, 4, 7, 1, 3, 5]
[2, 4, 1, 7, 0, 6, 3, 5]
[2, 4, 1, 7, 5, 3, 6, 0]
[2, 4, 6, 0, 3, 1, 7, 5]
[2, 4, 7, 3, 0, 6, 1, 5]
[2, 5, 1, 4, 7, 0, 6, 3]
[2, 5, 1, 6, 0, 3, 7, 4]
[2, 5, 1, 6, 4, 0, 7, 3]
[2, 5, 3, 0, 7, 4, 6, 1]
[2, 5, 3, 1, 7, 4, 6, 0]
[2, 5, 7, 0, 3, 6, 4, 1]
[2, 5, 7, 0, 4, 6, 1, 3]
[2, 5, 7, 1, 3, 0, 6, 4]
[2, 6, 1, 7, 4, 0, 3, 5]
[2, 6, 1, 7, 5, 3, 0, 4]
[2, 7, 3, 6, 0, 5, 1, 4]
[3, 0, 4, 7, 1, 6, 2, 5]
[3, 0, 4, 7, 5, 2, 6, 1]
[3, 1, 4, 7, 5, 0, 2, 6]
[3, 1, 6, 2, 5, 7, 0, 4]
[3, 1, 6, 2, 5, 7, 4, 0]
[3, 1, 6, 4, 0, 7, 5, 2]
[3, 1, 7, 4, 6, 0, 2, 5]
[3, 1, 7, 5, 0, 2, 4, 6]
[3, 5, 0, 4, 1, 7, 2, 6]
[3, 5, 7, 1, 6, 0, 2, 4]
[3, 5, 7, 2, 0, 6, 4, 1]
[3, 6, 0, 7, 4, 1, 5, 2]
[3, 6, 2, 7, 1, 4, 0, 5]
[3, 6, 4, 1, 5, 0, 2, 7]
[3, 6, 4, 2, 0, 5, 7, 1]
[3, 7, 0, 2, 5, 1, 6, 4]
[3, 7, 0, 4, 6, 1, 5, 2]
[3, 7, 4, 2, 0, 6, 1, 5]
[4, 0, 3, 5, 7, 1, 6, 2]
[4, 0, 7, 3, 1, 6, 2, 5]
[4, 0, 7, 5, 2, 6, 1, 3]
[4, 1, 3, 5, 7, 2, 0, 6]
[4, 1, 3, 6, 2, 7, 5, 0]
[4, 1, 5, 0, 6, 3, 7, 2]
[4, 1, 7, 0, 3, 6, 2, 5]
[4, 2, 0, 5, 7, 1, 3, 6]
[4, 2, 0, 6, 1, 7, 5, 3]
[4, 2, 7, 3, 6, 0, 5, 1]
[4, 6, 0, 2, 7, 5, 3, 1]
[4, 6, 0, 3, 1, 7, 5, 2]
[4, 6, 1, 3, 7, 0, 2, 5]
[4, 6, 1, 5, 2, 0, 3, 7]
```

```
[4, 6, 1, 5, 2, 0, 7, 3]
[4, 6, 3, 0, 2, 7, 5, 1]
[4, 7, 3, 0, 2, 5, 1, 6]
[4, 7, 3, 0, 6, 1, 5, 2]
[5, 0, 4, 1, 7, 2, 6, 3]
[5, 1, 6, 0, 2, 4, 7, 3]
[5, 1, 6, 0, 3, 7, 4, 2]
[5, 2, 0, 6, 4, 7, 1, 3]
[5, 2, 0, 7, 3, 1, 6, 4]
[5, 2, 0, 7, 4, 1, 3, 6]
[5, 2, 4, 6, 0, 3, 1, 7]
[5, 2, 4, 7, 0, 3, 1, 6]
[5, 2, 6, 1, 3, 7, 0, 4]
[5, 2, 6, 1, 7, 4, 0, 3]
[5, 2, 6, 3, 0, 7, 1, 4]
[5, 3, 0, 4, 7, 1, 6, 2]
[5, 3, 1, 7, 4, 6, 0, 2]
[5, 3, 6, 0, 2, 4, 1, 7]
[5, 3, 6, 0, 7, 1, 4, 2]
[5, 7, 1, 3, 0, 6, 4, 2]
[6, 0, 2, 7, 5, 3, 1, 4]
[6, 1, 3, 0, 7, 4, 2, 5]
[6, 1, 5, 2, 0, 3, 7, 4]
[6, 2, 0, 5, 7, 4, 1, 3]
[6, 2, 7, 1, 4, 0, 5, 3]
[6, 3, 1, 4, 7, 0, 2, 5]
[6, 3, 1, 7, 5, 0, 2, 4]
[6, 4, 2, 0, 5, 7, 1, 3]
[7, 1, 3, 0, 6, 4, 2, 5]
[7, 1, 4, 2, 0, 6, 3, 5]
[7, 2, 0, 5, 1, 4, 6, 3]
[7, 3, 0, 2, 5, 1, 6, 4]
```

Viaje por el rio. Programación dinámica

In [ ]:

```
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 [""]*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 po
        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:
```

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

Viaje por el rio. Programación dinámica MODIFICADO

Las modificadiones ingresadas en este código son las siguientes:

- Ajuste de la inicialización de la variable PRECIOS
- En la versión original se omite la última fila en los índices de i (for i in range(0, N-1))
- Reajuste en la selección del mínimo pues es reduntante volver a llamar a la función
- Mejoras en la función calcular_ruta además de devolver la ruta como lista en la versión modificada

In [17]:
```
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 _ in range(N)]
  RUTA = [ [""]*N for i in [""]*N]

  for i in range(0,N):
    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 = PRECIOS[i][k] + TARIFAS[k][j]
         RUTA[i][j] = 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_lista(RUTA, desde, hasta):
    if desde == hasta:
        return [desde]
    else:
        return calcular_ruta_lista(RUTA, desde, RUTA[desde][hasta]) + [hasta]

print("\nLa ruta es:")
print(calcular_ruta_lista(RUTA, 0, 6))  # Ejemplo: 0,3,5,6
```

```
PRECIOS
[0, 9999, 9999, 9999, 9999, 9999, 999]
[9999, 0, 9999, 9999, 9999, 9999, 11]
[9999, 9999, 0, 9999, 9999, 9999, 10]
[9999, 9999, 9999, 0, 9999, 9999, 9]
[9999, 9999, 9999, 9999, 0, 9999, 4]
[9999, 9999, 9999, 9999, 9999, 0, 3]
[9999, 9999, 9999, 9999, 9999, 9999, 3]

RUTA
[0, 0, 0, 0, 0, 0, 0]
['', 1, 1, 1, 1, 1, 1]
['', '', 2, 2, 2, 2, 2]
['', '', '', 3, 3, 3, 3]
['', '', '', '', 4, 4, 4]
['', '', '', '', '', 5, 5]
['', '', '', '', '', '', 6]

La ruta es:
[0, 6]
```