

▼ Nombre: Salvador Gimeno

Actividad Guiada 2 - AG2

Github: <https://github.com/salvagimeno-ai/03MAIR-Algoritmos-de-optimizacion/tree/master/>

Google Colab: https://colab.research.google.com/drive/1cuQarEAKQruCZwSzKHpx8_fx18XP5i

```
# Decorador para calcular el tiempo de ejecucion
from time import time

def calcular_tiempo(f):

    def wrapper(*args, **kwargs):
        inicio = time()
        resultado = f(*args, **kwargs)
        tiempo = time() - inicio
        print("Tiempo de ejecución para algoritmo: "+str(tiempo))
```

Saved successfully!

```
from time import time
```

▼ Algoritmo con tecnica de programacion dinamica

```
# tala TARIFAS contiene el precio que cuesta ir desde cada nodo a todos los otros nodos de
# Para los casos donde no se puede ir de un nodo a otro, colocaremos un valor muy alto '99
# para esa combinacion, para que el algoritmo no tenga en cuenta esa posibilidad
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):
    N = len(TARIFAS)
    PRECIOS = [[999]*N for i in [9999]*N]
    RUTAS = [[""]*N for i in [9999]*N]

    for i in range(N-1):
        for j in range(i+1,N):
            MIN = TARIFAS[i][j]
            RUTAS[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] )
                RUTAS[i][j] = k
        PRECIOS[i][j] = MIN

    return PRECIOS, RUTAS

```

```

PRECIOS, RUTAS = precios(TARIFAS)

print('Matriz de precios: ' , PRECIOS)
#print(PRECIOS)
print()

print('Rutas: ' , RUTAS)

```

Matriz de precios: [[999, 5, 4, 3, 8, 8, 11], [999, 999, 999, 2, 3, 8, 7], [999, 999, 999, 999, 999, 999, 999], [999, 999, 999, 999, 999, 999, 999], [999, 999, 999, 999, 999, 999, 999], [999, 999, 999, 999, 999, 999, 999], [999, 999, 999, 999, 999, 999, 999]]

Rutas: [['', 0, 0, 0, 1, 2, 5], ['', '', 1, 1, 1, 3, 4], ['', '', '', 2, 3, 2, 5], ['', '', '', '', 3, 3, 3], ['', '', '', '', 4, 4], ['', '', '', '', 5], ['', '', '', '', '', ']]

Saved successfully!



a):

```

if desde == hasta:
    #print("Ir a :" + str(desde))
    return desde
else:
    return str(calcular_ruta(RUTAS, desde, RUTAS[desde][hasta])) + ',' + str(RUTAS[de

```

```

print('La ruta es: ', calcular_ruta(RUTAS, 0,6))

```

La ruta es: 0,0,2,5

```

precios(TARIFAS)

```

[[[999, 5, 4, 3, 8, 8, 11],
[999, 999, 999, 2, 3, 8, 7],
[999, 999, 999, 1, 6, 4, 7],
[999, 999, 999, 999, 5, 6, 9],
[999, 999, 999, 999, 999, 999, 4],
[999, 999, 999, 999, 999, 999, 3],
[999, 999, 999, 999, 999, 999, 999]],
[['', 0, 0, 0, 1, 2, 5],
['', '', 1, 1, 1, 3, 4],
['', '', '', 2, 3, 2, 5],
['', '', '', '', 3, 3, 3],
['', '', '', '', 4, 4],
['', '', '', '', 5],
['', '', '', '', '', ']]]

▼ Asignación de tareas - Ramificación y poda

```
import itertools
from functools import wraps

def calcular_tiempo(f):
    @wraps(f)
    def cronometro(*args, **kwargs):
        t_inicial = time()
        salida = f(*args, **kwargs)
        t_final = time()
        print('Tiempo transcurrido en segundos: {}'.format(t_final - t_inicial))
        return salida
    return cronometro

COSTES=[[11,12,18,40],    #cada fila = 1 agente
        [14,15,13,22],   #cada columna es el coste de cada tarea
        [11,17,19,23],
        [17,14,20,28]]
```

Saved successfully!



0 para cada tarea
rea 2),...

```
# definimos una funcion para calcular el coste de una asignacion de
# tareas dada
def valor(S, COSTES):
    VALOR =0
    for i in range(len(S)):
        VALOR += COSTES[S[i]][i]
    return VALOR

valor((0,1,2,3), COSTES)
```



73

```
# el metodo de fuerza bruta revisaria todas las combinaciones posibles
```


```
def fuerza_bruta(COSTES):
    mejor_valor=10e10    #inicializamos el mejor valor en un valor muy alto
    mejor_solucion = ''

    for S in list(itertools.permutations(range(len(COSTES)))):
        valor_tmp = valor(S, COSTES)
        if valor_tmp < mejor_valor:
            mejor_solucion = S
            mejor_valor = valor_tmp
    print("La mejor solución es: " , mejor_solucion)

fuerza_bruta(COSTES)
```

 La mejor solución es: (0, 3, 1, 2)

```
## TEST: Funcion para CALCULO PERMUTACIONES de una lista
#list(itertools.permutations([0,1,2,3]))
list(itertools.permutations(range(len(COSTES))))
```

 [(0, 1, 2, 3),
(0, 1, 3, 2),
(0, 2, 1, 3),
(0, 2, 3, 1),
(0, 3, 1, 2),
(0, 3, 2, 1),
(1, 0, 2, 3),
(1, 0, 3, 2),
(1, 2, 0, 3),
(1, 2, 3, 0),
(1, 3, 0, 2),
(1, 3, 2, 0),
(2, 0, 1, 3),
(2, 0, 3, 1),
(2, 1, 0, 3),
(2, 1, 3, 0),
(2, 2, 0, 1),
(2, 2, 1, 0)]

Saved successfully!



(3, 0, 2, 1),
(3, 1, 0, 2),
(3, 1, 2, 0),
(3, 2, 0, 1),
(3, 2, 1, 0)]

```
# ESTIMACIÓN DEL COSTE INFERIOR PARA UNA SOLUCION PARCIAL
def CI(S, COSTES):
    VALOR = 0

    for i in range(len(S)):
        VALOR += COSTES[i][S[i]]

    #Estimaciones
    for i in range(len(COSTES)):
        if i not in S:
            VALOR += min ([ COSTES[j][i] for j in range((len(S)), len(COSTES)) ])

    return VALOR


CI((1,2), COSTES)

#definimos una funcion para expandir (crear hijos)
def crear_hijos(NODO, N): # N seria la dimensionalidad del problema
    HIJOS = [] #aquí vamos a ir acumulando los hijos
    for i in range(N):
        if i not in NODO:
            HIJOS.append( {'s':NODO+(i,)} )

    return HIJOS
```

```
# hijos de...
# (0,2) --> (0,2,1), (0,2,3) - podriamos asignar la tarea 1 y la 3
```

```
crear_hijos((0,2),4)
```

```
 [{'s': (0, 2, 1)}, {'s': (0, 2, 3)}]
```

```
COSTES=[[11,12,18,40], #cada fila = 1 agente
         [14,15,13,22], #cada columna es el coste de cada tarea
         [11,17,19,23],
         [17,14,20,28]]
```

```
# [11,12,18,40] - costes del agente 0 para cada tarea
# 11 (tarea 0), 12 (tarea 1), 18 (tarea 2),...
```

```
def ramificacion_y_poda(COSTES):
    DIMENSION = len(COSTES)
    MEJOR_SOLUCION = tuple(i for i in range(DIMENSION))
```

```
COSTES)
```

Saved successfully!



```
NODOS.append({'s':(), 'ci': CI((), COSTES)})
```

```
#NODOS.append({'s':(1,), 'ci': 34 })
```

```
#print(NODOS)
```

```
iteracion=0
```

```
while (len(NODOS) > 0):
    iteracion +=1
    print('\n#', iteracion)
```

```
nodo_prometedor = min(NODOS, key=lambda x:x['ci'])
#return nodo_prometedor
```

```
#obtenemos los hijos
HIJOS = [ {'s':x['s'], 'ci':CI(x['s'], COSTES) } for x in crear_hijos(nodo_promete
```

```
#ver si alguno de los hijos son estados finales
#y los colocamos en la lista NODO_FINAL
NODO_FINAL = [ x for x in HIJOS if len(x['s']) == DIMENSION ]
if len(NODO_FINAL) > 0:
    #haremos una revision
    if NODO_FINAL[0]['ci'] < CotaSup:
        #establecemos una nueva CotaSup
        CotaSup = NODO_FINAL[0]['ci']
        #actualizamos como solucion el nodo final
        MEJOR_SOLUCION = NODO_FINAL[0]
```

```
#PODA: eliminamos los hijos q no son prometedores
```

```
HIJOS = [x for x in HIJOS if x['ci'] < CotaSup ]

print(nodo_prometedor)
#eliminamos el nodo expandido
NODOS = [x for x in NODOS if x['s'] != nodo_prometedor['s'] ]

#añadimos los nuevos hijos a la lista de nodos pendientes de analizar
NODOS.extend(HIJOS)

#print(NODOS)

#return HIJOS
print("La mejor solucion es: ", MEJOR_SOLUCION)

ramificacion_y_poda(COSTES)
```



Saved successfully!



```
# 1
{'s': (), 'ci': 58}

# 2
{'s': (1,), 'ci': 58}

# 3
{'s': (1, 2), 'ci': 59}

# 4
{'s': (0,), 'ci': 60}

# 5
{'s': (0, 2), 'ci': 61}

# 6
{'s': (0, 2, 3), 'ci': 61}

# 7
{'s': (1, 3), 'ci': 64}

# 8
{'s': (1, 2, 0), 'ci': 64}
```

Saved successfully!



```
# 10
{'s': (1, 2, 3), 'ci': 65}

# 11
{'s': (0, 3), 'ci': 66}

# 12
{'s': (1, 0), 'ci': 68}

# 13
{'s': (0, 1), 'ci': 68}

# 14
{'s': (0, 2, 1), 'ci': 69}
La mejor solucion es: {'s': (0, 2, 3, 1), 'ci': 61}
```

Saved successfully!

