AG2 - Actividad Guiada 2

Nombre: Alberto Rodriguez Arizaga

Actividad guiada de Algoritmos Optimización (AG2)

https://colab.research.google.com/drive/1rKq9jmnDJruGOsueXwQPu1dglB2s1uNg?usp=sharing

https://github.com/brtln05/03MIAR---Algoritmos-de-Optimizacion

Programación Dinámica. Viaje por el rio

Enunciado del problema

En un río hay **n** embarcaderos y debemos desplazarnos río abajo desde un embarcadero a otro. Cada embarcadero tiene precios diferentes para ir de un embarcadero a otro situado más abajo. Para ir del embarcadero i al j, puede ocurrir que sea más barato hacer un trasbordo por un embarcadero intermedio k. El problema consiste en determinar la combinación más barata.

```
#Viaje por el rio - Programación dinámica

# Declaramos el coste de cada una de las posibles rutas en nuestro problema

coste_indiv_ruta = [
[0,5,4,3,float("inf"),999,999],   # desde nodo 0
[999,0,999,2,3,999,11],  # desde nodo 1
[999,999, 0,1,999,4,10], # desde nodo 2
[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]
]

coste_indiv_ruta
```

```
[[0, 5, 4, 3, inf, 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]]
```

```
#Calculo de la matriz de precio total de la ruta y la propia ruta
```

```
#  ruta  - contiene los nodos intermedios para ir de un nodo a otro

def calcula_precios(coste_indiv_ruta):

  #Total de Nodos
  n = len(coste_indiv_ruta[0])

  #Inicialización de la tabla de precios
  coste_tot_ruta = [ [9999]*n for i in [9999]*n]  #n x n
  ruta = [ [""]*n for i in [""]*n]

  # Se recorren todos los nodos con dos bucles (desde - hasta)
  # para ir construyendo la matriz de coste total de ruta
  for i in range(n-1):
    for j in range(i+1, n):
      coste_min_ruta = coste_indiv_ruta[i][j]
      ruta[i][j] = i

      # Evaluamos el coste de ir desde el nodo i hasta el nodo j pasando por el k
      # Esto es, sumar el precio de ir hasta el k más del k al j
      # Si el precio es más barato el nodo se añade como ruta óptima
      for k in range(i, j):
        if coste_tot_ruta[i][k] + coste_indiv_ruta[k][j] < coste_min_ruta:
            coste_min_ruta = min(coste_min_ruta, coste_tot_ruta[i][k] + coste_indiv_ruta[k
            ruta[i][j] = k
        coste_tot_ruta[i][j] = coste_min_ruta

  return coste_tot_ruta,ruta

coste_tot_ruta,ruta = calcula_precios(coste_indiv_ruta)


#Calculo de la ruta usando la matriz ruta
def calcular_ruta(ruta, desde, hasta):
  if desde == ruta[desde][hasta]:

    return desde
  else:
    return str(calcular_ruta(ruta, desde, ruta[desde][hasta])) +  ' --> ' + str(ruta[desde

print("\n La ruta óptima es:")
calcular_ruta(ruta, 0,6)
```

```
 La ruta óptima es:
'0 --> 2 --> 5'
```

Problema de Asignacion de tarea

```
#Asignacion de tareas - Ramificación y Poda

#    T A R E A
#   A
#   G
#   E
#   N
#   T
#   E
          #T0 T1 T2 T3
costes = [[11,12,18,40], # Agente 0
          [14,15,13,22], # Agente 1
          [11,17,19,23], # Agente 2
          [17,14,20,28]] # Agente 3


     73


# Calculo del valor de una solucion parcial

def calc_valor(sol,costes):
  valor = 0
  for i in range(len(sol)):
    valor += costes[i][sol[i]]
  return valor


calc_valor((1,2,0,3),costes)

     64


def CI(sol,costes):
  VALOR = 0
  #Valores establecidos
  for i in range(len(sol)):
    VALOR += costes[i][sol[i]]

  #Estimacion
  for i in range( len(sol), len(costes)   ):
    VALOR += min( [ costes[j][i] for j in range(len(sol), len(costes))  ])
  return VALOR

def CS(sol,costes):
  VALOR = 0
  #Valores establecidos
  for i in range(len(sol)):
    VALOR += costes[i][sol[i]]
```

```
  #Estimacion
  for i in range( len(sol), len(costes)   ):
    VALOR += max( [ costes[j][i] for j in range(len(sol), len(costes))  ])
  return VALOR

CI((1,0,3),costes)
```

```
77
```

```
#Genera los hijos en base a kla situación actual de la rama seleccionada

def crear_hijos(nodo, n):
  hijos = []
  for i in range(n ):
    if i not in nodo:
      hijos.append({'s':nodo +(i,)    })
  return hijos

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

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

```
# Construccion iterativa de soluciones(arbol). En cada etapa asignamos un agente(ramas).
# Nodos del grafo construidos como { s:(1,2),CI:3,CS:5  }

def ramificacion_y_poda(costes):

  dimension = len(costes)
  mejor_solucion=tuple( i for i in range(len(costes)) )
  CotaSup = calc_valor(mejor_solucion,costes)
  #print("Cota Superior:", CotaSup)

  nodos=[]
  nodos.append({'s':(), 'ci':CI((),costes)    } )

  iteracion = 0

  while( len(nodos) > 0):
    iteracion +=1

    nodo_prometedor = [ min(nodos, key=lambda x:x['ci']) ][0]['s']
    # print("Nodo prometedor:", nodo_prometedor)

    # Ramificamos los hijos del nodo prometedor
    hijos =[ {'s':x['s'], 'ci':CI(x['s'], costes)   } for x in crear_hijos(nodo_prometedor

    # Revisamos la cota superior y nos quedamos con la mejor solucion si llegamos a una so
    nodo_final = [x for x in hijos if len(x['s']) == dimension  ]
    if len(nodo_final ) >0:
```

```
      #print("\n********Soluciones:",  [x for x in hijos if len(x['s']) == dimension  ], "
      if nodo_final[0]['ci'] < CotaSup:
        CotaSup = nodo_final[0]['ci']
        mejor_solucion = nodo_final

    # Descartamos los hijos no prometedores
    hijos = [x for x in hijos if x['ci'] < CotaSup   ]

    # Añadimos los hijos
    nodos.extend(hijos)

    # Eliminamos el nodo ramificado
    nodos =  [  x for x in nodos if x['s'] != nodo_prometedor    ]

  print("La solucion final es:" ,mejor_solucion , " en " , iteracion , " iteraciones" , "



ramificacion_y_poda(costes)
```

```
 La solucion final es: [{'s': (1, 2, 0, 3), 'ci': 64}]  en  10  iteraciones  para dim
```