```
<a href="https://drive.google.com/file/d/1QwhSuenBtsX-1S1DN-qaLRrpT3EIQBa/view?usp=sharing" target="_parent"><img src="https://colab.research.google.com/assets/colab-badge.svg" alt="Open In Colab"></a>
```

Open in Colab

Algoritmos - Actividad Guiada 3

Nombre: Diocles Germán Avendaño Ponce
URL:https://drive.google.com/file/d/1QwhSuenBtsX-1S1DN-qaLRrpT3EIQBa/view?usp=sharing
GIRA:https://github.com/gohanzz/algoritmos_optimizacion_mar2023/blob/main/src/AG3/Algoritmos_AG3_Diocles_German_Avendano_Ponce.ipynb

Carga de librerias

```
[ ] !pip install requests
    !pip install tsp1095

    Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.27.1)
    Requirement already satisfied: urllib3>1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (1.26.16)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2023.7.22)
    Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests) (2.0.12)
    Requirement already satisfied: idna>4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.4)
    Collecting tsp10095
      Downloading tsp10095-0.7.1-py2.py3-none-any.whl (25 kB)
    Requirement already satisfied: click>=6.0 in /usr/local/lib/python3.10/dist-packages (from tsp10095) (8.1.6)
    Collecting Deprecated>=1.2.9 (from tsp10095)
      Downloading Deprecated-1.2.14-py2.py3-none-any.whl (9.6 kB)
    Collecting networkx>=2.3 (from tsp10095)
      Downloading networkx-3.0-py3-none-any.whl (2.0 MB)
                                              2.0/2.0 MB 19.4 MB/s eta 0:00:00
    Collecting tabulate>=0.8.7 (from tsp10095)
      Downloading tabulate-0.8.10-py3-none-any.whl (29 kB)
    Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from Deprecated>=1.2.9->tsp10095) (1.14.1)
    Installing collected packages: tabulate, networkx, Deprecated, tsp10095
      Attempting uninstall: tabulate
        Found existing installation: tabulate 0.9.0
        Uninstalling tabulate-0.9.0:
          Successfully uninstalled tabulate-0.9.0
      Attempting uninstall: networkx
        Found existing installation: networkx 3.1
        Uninstalling networkx-3.1:
          Successfully uninstalled networkx-3.1
    Successfully installed Deprecated-1.2.14 networkx-2.8.8 tabulate-0.8.10 tsp10095-0.7.1
```

Carga de los datos del problema

```
[ ] import urllib.request
    import tsp10095
    import math
    import random


    #http://atlas.i3b.de/pub/dmp-testdata/tsp/tsplib/
    #Wisconsin
      # http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf
      # https://tsp10095.readthedocs.io/en/stable/pages/usage.html
      # https://tsp10095.readthedocs.io/en/v0.6.1/modules.html
      # https://pypi.org/project/tsp10095/

    #Descargamos el fichero de datos(Matriz de distancias)
    file = "wi29.tsp"
    urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/wi29.tsp.gz", file + ".gz")
    !gzip -d wi29.tsp.gz
    #Coordenadas 15-city problem (Christofides/Eilon)
    #file = "eil51.tsp" urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/eil51.tsp.gz", file)

    #Coordenadas - 48 capitals of the US (Padberg/Rinaldi)
    #file = "att48.tsp" urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/att48.tsp.gz", file)
```

```
[ ] #Carga de datos y generación de objeto problem
    ##################################################################
    problem = tsp10095.load(file)

    #Nodos
    Nodos = list(problem.get_nodes())

    #Aristas
    Aristas = list(problem.get_edges())
    print(type(problem))
```

```
    <class 'tsp10095.models.StandardProblem'>
```

```
NAME0RE: wi29
TYPE: TSP
COMENTARIO: 42 Steeffe Schwali (Fricker)
DIMENSION: 29
EDGE_WEIGHT_TYPE: EUC_2D
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
0 27 43 16 30 26 18 12 46 29 37 31 28 24 10 15 40 54 70 62 57 78 62 16 132 131 131 106 111 57 28 43 78
27 0 34 43 27 40 13 32 61 37 49 41 38 34 17 12 40 54 70 62 43 64 48 16 106 105 105 80 85 31 13 16 51
43 34 0 26 12 16 19 16 40 42 44 40 40 40 30 35 40 20 36 28 40 30 14 30 100 101 101 76 81 5 18 18 20
16 43 26 0 14 20 34 30 35 35 18 15 42 44 40 40 35 40 56 48 35 46 40 40 116 115 115 90 95 21 30 30 35
30 27 12 14 0 14 20 16 40 30 32 28 28 28 18 23 40 54 70 62 45 66 42 28 102 101 101 76 81 7 16 16 23
26 40 16 20 14 0 34 30 20 16 30 34 30 30 30 30 40 54 70 62 30 66 42 30 102 101 101 76 81 7 16 16 23
18 13 19 34 20 34 0 4 46 29 37 31 28 24 10 15 40 54 70 62 43 64 48 16 106 105 105 80 85 31 13 16 51
12 32 16 30 16 30 4 0 42 25 33 27 24 20 6 11 36 50 66 58 39 60 44 12 102 101 101 76 81 27 9 12 47
46 61 40 35 35 20 46 42 0 17 14 18 18 22 36 31 6 20 36 28 18 36 22 36 116 115 115 90 95 35 49 44 19
29 37 42 35 30 16 29 25 17 0 14 18 18 22 19 14 11 6 20 12 18 20 6 29 99 98 98 73 78 30 36 31 18
37 49 44 40 32 30 31 27 14 14 0 4 32 36 33 28 25 14 6 12 18 6 8 37 107 106 106 81 86 38 44 39 26
31 41 40 40 28 34 28 24 18 18 4 0 28 32 29 24 21 10 2 8 14 2 4 33 103 102 102 77 82 34 40 35 22
28 38 40 40 28 30 28 20 18 22 32 28 0 4 18 15 21 18 30 22 10 32 24 18 75 74 74 49 54 40 34 29 18
24 24 30 30 18 30 24 16 22 22 36 32 4 0 14 11 25 22 34 26 14 36 28 22 79 78 78 53 58 36 30 25 14
10 16 36 34 23 30 10 6 36 19 33 29 18 14 0 5 36 50 66 58 39 60 44 12 102 101 101 76 81 29 19 14 45
15 12 40 40 28 40 15 11 31 14 28 24 15 11 5 0 31 45 61 53 34 55 39 7 97 96 96 71 76 24 14 9 40
40 40 40 35 40 40 28 36 6 11 14 14 21 25 36 31 0 14 30 22 12 30 16 36 116 115 115 90 95 35 45 40 19
54 54 20 40 54 54 40 50 20 6 6 2 10 14 50 45 14 0 16 8 4 16 8 54 90 89 89 64 69 20 30 25 8
70 70 36 56 70 70 56 66 36 20 12 8 30 34 66 61 30 16 0 8 20 8 20 70 106 105 105 80 85 36 46 41 24
62 62 28 48 62 62 48 58 28 12 18 14 22 26 58 53 22 8 8 0 12 8 12 62 98 97 97 72 77 28 38 33 16
57 43 40 35 40 30 43 39 18 18 28 24 10 14 39 34 18 4 20 12 0 20 4 43 103 102 102 77 82 35 45 40 19
78 64 30 46 66 66 36 46 36 20 32 28 32 36 60 55 12 16 8 8 20 0 20 64 100 99 99 74 79 30 40 35 16
62 48 14 40 42 42 48 40 22 6 8 4 18 22 44 39 6 8 12 12 4 20 0 29 99 98 98 73 78 14 24 19 14
16 16 30 40 28 30 16 12 36 29 37 33 28 24 12 7 36 54 70 62 43 64 48 0 106 105 105 80 85 31 13 16 51
132 106 100 116 102 102 106 102 116 99 107 103 75 79 97 97 116 90 106 98 103 98 99 106 0 1 1 26 21 101 93 90 65
131 105 101 115 101 101 105 101 115 98 106 102 74 78 96 96 115 89 105 97 102 97 98 105 1 0 0 25 20 100 92 89 64
131 105 101 115 101 101 105 101 115 98 106 102 74 78 96 96 115 89 105 97 102 97 98 105 1 0 0 25 20 100 92 89 64
106 80 76 90 76 76 80 76 90 73 81 77 49 53 71 71 90 64 80 72 77 72 73 80 26 25 25 0 5 76 66 63 39
111 85 81 95 81 81 85 81 95 78 86 82 54 58 76 76 95 69 85 77 82 77 78 85 21 20 20 5 0 81 71 68 44
57 31 5 21 7 7 31 27 35 30 38 34 30 25 19 14 35 20 36 28 35 28 29 31 101 100 100 76 81 0 10 5 20
28 13 18 30 16 16 13 9 49 36 44 40 34 29 14 9 45 30 46 38 45 38 39 13 93 92 92 66 71 10 0 5 30
43 16 18 30 16 16 16 12 44 31 39 35 29 25 16 11 40 25 41 33 40 33 34 16 90 89 89 64 69 5 5 0 25
78 51 20 35 23 23 51 47 19 18 26 22 18 14 45 40 19 8 20 16 19 16 17 51 65 64 64 39 44 20 30 25 0
```

Funcionas basicas

```
[ ]
    #Funciones basicas
    ##################################################################

    #Se genera una solucion aleatoria con cualquiera en en el nodo 0
    def crear_solucion(Nodos):
      solucion = [Nodos[0]]
      for a in Nodos[1:]:
        solucion = solucion + [random.choice(list(set(Nodos) - set([Nodos[0]]) - set(solucion)))]
      return solucion


    #Devuelve la distancia entre dos nodos
    def distancia(a,b, problem):
      return problem.get_weight(a,b)

    #Devuelve la distancia total de una trayectoria/solucion
    def distancia_total(solucion, problem):
      distancia_total = 0
      for i in range(len(solucion)-1):
        distancia_total = distancia(solucion[i] ,solucion[i+1] , problem)
      return distancia_total + distancia(solucion[len(solucion)-1] ,solucion[0], problem)
```

BUSQUEDA ALEATORIA

```
[ ] ##################################################################
    # BUSQUEDA ALEATORIA
    ##################################################################


    def busqueda_aleatoria(problem, N):
      #N es el numero de iteraciones
      Nodos = list(problem.get_nodes())

      mejor_solucion = []
      #mejor_distancia = 10e100                    #Inicializamos con un valor alto
      mejor_distancia = float('inf')               #Inicializamos con un valor alto

      for i in range(N):                           #Criterio de parada: repetir N veces para podernos incluir otros
        solucion = crear_solucion(Nodos)           #Genera una solucion aleatoria
        distancia = distancia_total(solucion, problem)  #Calcula el valor objetivo(distancia total)

        if distancia < mejor_distancia:            #Comparo con la mejor distancia hasta ahora
          mejor_solucion = solucion
          mejor_distancia = distancia


      print("Mejor solución:" , mejor_solucion)
      print("Distancia     :" , mejor_distancia)
      return mejor_solucion


    #Busqueda aleatoria con 5000 iteraciones
    solucion = busqueda_aleatoria(problem, 10000)
```

```
    Mejor solución: [0, 20, 8, 17, 27, 14, 18, 5, 10, 11, 12, 16, 16, 12, 9, 41, 6, 7, 1, 15, 35, 31, 19, 4, 3, 18, 24, 40, 21, 22, 2, 23, 28, 25, 37, 13, 36, 33, 26, 28, 33, 18]
    Distancia     : 3931
```

BUSQUEDA LOCAL

```
[ ] ##################################################################
    # BUSQUEDA LOCAL
    ##################################################################
    def genera_vecina(solucion):
    #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos se generan (N-1)x(N-2)/2 soluciones
    #Por ejemplo multiplicar para aplicar otros generadores distintos que 2-opt
    vecina = []
    mejor_solucion = []
    mejor_distancia = 10e100
    for i in range(1,len(solucion)-1):          #Recorremos todos los nodos en bucle doble para evaluar todos los intercambios 2-opt
      for j in range(i+1, len(solucion)):

        #Se genera una nueva solución intercambiando los dos nodos i,j:
        # (nuevamente el operador x que para listas en python les concatena) : e1: [1,2] + [3] = [1,2,3]
        vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:]

        #Se evalua la nueva solución
        distancia_vecina = distancia_total(vecina, problem)

        #... para guardarla si mejora las anteriores
        if distancia_vecina < mejor_distancia:
          mejor_distancia = distancia_vecina
          mejor_solucion = vecina
    return mejor_solucion


    #solucion = [1, 47, 13, 41, 48, 19, 43, 44, 17, 5, 22, 28, 3, 2, 29, 21, 50, 34, 38, 9, 10, 11, 30, 49, 18, 39, 15, 45, 15, 24, 43, 26, 31, 36, 35, 20, 8, 7, 23, 48, 27, 12, 37, 4, 18, 25, 14, 6, 51, 46, 32]
    print("Distancia Solucion Inicial:", distancia_total(solucion, problem))


    nueva_solucion = genera_vecina(solucion)
    print("Distancia Mejor Solucion Local:", distancia_total(nueva_solucion, problem))
```

```
    Distancia Solucion Inicial: 3781
    Distancia Mejor Solucion Local: 3371
```

```
[ ] #Busqueda Local:
    #   Sobre el operador de vecindad 2-opt(funcion genera_vecina)
    #   Los criterio de parada, se para cuando no es posible mejorar.
    def busqueda_local(problem):
      mejor_solucion = []

      #Genera una solución inicial de referencia(aleatoria)
      solucion_referencia = crear_solucion(Nodos)
      mejor_distancia = distancia_total(solucion_referencia, problem)

      iteracion=0              #Un contador para saber las iteraciones que hacemos
      while(1):
        iteracion +=1          #Incrementamos el contador
        #print("#",iteracion)

        #Obtenemos la mejor vecina ...
        vecina = genera_vecina(solucion_referencia)

        #... y la evaluamos para ver si mejoramos respecto a la encontrada hasta el momento
        distancia_vecina = distancia_total(vecina, problem)

        #Si no mejoramos hay que terminar. Hemos llegado a un mínimo local(según nuestro operador de vecindad 2-opt)
        if distancia_vecina < mejor_distancia:
          #mejor_solucion = copy.deepcopy(vecina)  #Con copia profunda. Las copias en python son por referencia
          mejor_solucion = vecina                 #Guarda la mejor solución encontrada
          mejor_distancia = distancia_vecina

        else:
          print("En la iteracion ", iteracion, ", la mejor solución encontrada es:" , mejor_solucion)
          print("Distancia     :" , mejor_distancia)
          return mejor_solucion

        solucion_referencia = vecina


    sol = busqueda_local(problem )
```

```
    En la iteracion  45 , la mejor solución encontrada es: [0, 14, 16, 15, 37, 7, 6, 4, 29, 21, 24, 40, 23, 41, 18, 26, 5, 19, 13, 8, 9, 39, 22, 38, 30, 28, 3, 1, 17, 31, 36, 35, 20, 33, 34, 32, 27, 2]
    Distancia     : 3535
```

SIMULTANEALING

```
[ ] ##################################################################
    # SIMULATED ANNEALING
    ##################################################################

    #Generador de 1 solución vecina 2-opt 100% aleatoria (intercambiar 2 nodos)
    #Opcionalmente alternando otra forma de elegir una vecina.
    def genera_vecina_aleatoria(solucion):

      #Se eligen dos nodos aleatoriamente
      i,j = sorted(random.sample( range(1,len(solucion)) , 2))

      #Devuelve una nueva solución para intercambiando los dos nodos elegidos al azar
      return solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:]


    #Función de probabilidad para aceptar peores soluciones
    def probabilidad(T,d):
      if random.random() <  math.exp( -1*d / T)  :
        return True
      else:
        return False

    #Funcion de decremento de temperatura
    def bajar_temperatura():
      return T*0.99


[ ] def recocido_simulado(problem, TEMPERATURA ):
      #problem - datos del problema
      #T - temperatura

      solucion_referencia = crear_solucion(Nodos)
      distancia_referencia = distancia_total(solucion_referencia, problem)

      mejor_solucion = []                  #* del seudocodigo
      mejor_distancia = 10e100             #* del seudocodigo


      N=0
      while TEMPERATURA > .0001:
        N+=1
        #Genera una solución vecina
        vecina =genera_vecina_aleatoria(solucion_referencia)

        #Calcula su valor(distancia)
        distancia_vecina = distancia_total(vecina, problem)

        #Si es la mejor solución de todas se guarda(siempre!!!)
        if distancia_vecina < mejor_distancia:
            mejor_solucion = vecina
            mejor_distancia = distancia_vecina

        #Si la nueva vecina es mejor se cambia
        #Si es peor se cambia según una probabilidad que depende de T y delta(distancia_referencia - distancia_vecina)
        if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distancia_referencia - distancia_vecina) ) :
          #solucion_referencia = copy.deepcopy(vecina)
          solucion_referencia = vecina
          distancia_referencia = distancia_vecina

        #Bajamos la temperatura
        TEMPERATURA = bajar_temperatura(TEMPERATURA)

      print("La mejor solución encontrada es: " , end="")
      print(mejor_solucion)
      print("con una distancia total de " , end="")
      print(mejor_distancia)
      return mejor_solucion


    sol = recocido_simulado(problem, 10000000)
```

```
    La mejor solución encontrada es: [0, 3, 32, 31, 17, 7, 1, 6, 5, 20, 10, 41, 23, 4, 27, 8, 21, 39, 22, 38, 30, 29, 9, 40, 24, 34, 33, 28, 36, 35, 37, 15, 14, 16, 19, 13, 18, 12, 11, 25, 26, 2]
    con una distancia total de 1834
```

Productos pagados de Colab - Cancela los contratos aquí