In [0]:
```
from time import time
def calcular_tiempo(f):

    def wrapper(*args, **kwargs):
      inicio = time()
      resultado = f(*args, **kwargs)
      tiempo =float(time() - inicio)
      print("\r\n Tiempo de ejecución para algortimo: "+"{0:.25f}".format(tiempo))
      return resultado

    return wrapper
```

Leonardo Pacheco -AG2

Actividad Guiada 2

Url:https://github.com/pachecoleonardo/03MAIR---Algoritmos-de-Optimizacion/tree/master/AG2

In [9]:
```
import random
import math

N=100
LISTA_2D = [ (random.randrange(1,N*10), random.randrange(1,N*10))for _ in range(N)]
print(LISTA_2D)
```

```
[(382, 215), (772, 934), (126, 724), (244, 125), (315, 574), (306, 148), (967, 800), (211, 347),
(413, 64), (698, 463), (166, 23), (723, 633), (656, 457), (316, 983), (120, 347), (307, 722), (2
23, 712), (948, 743), (376, 560), (497, 618), (479, 321), (416, 416), (305, 758), (135, 763), (2
80, 456), (649, 348), (947, 329), (870, 262), (269, 587), (509, 239), (699, 458), (101, 218), (9
8, 105), (395, 278), (248, 812), (611, 181), (792, 689), (144, 888), (416, 322), (460, 312), (50
3, 599), (43, 149), (694, 339), (569, 542), (853, 819), (714, 728), (51, 287), (508, 640), (928,
74), (823, 725), (24, 376), (383, 31), (804, 906), (304, 990), (211, 48), (830, 693), (73, 708),
(390, 197), (558, 877), (33, 194), (322, 178), (598, 315), (214, 715), (659, 255), (227, 133),
(470, 159), (917, 442), (458, 748), (961, 901), (945, 306), (110, 588), (925, 924), (584, 502),
(565, 111), (905, 479), (763, 325), (311, 178), (167, 480), (357, 639), (701, 924), (222, 103),
(213, 347), (361, 95), (574, 875), (131, 274), (521, 890), (838, 26), (987, 916), (142, 230), (7
06, 389), (452, 762), (739, 630), (418, 722), (965, 34), (677, 310), (621, 805), (939, 530), (75
6, 208), (434, 548), (743, 660)]
```

```
0, 208), (454, 548), (745, 660)]
```

In [10]:
```
def distancia(A,B):
  if type(A) is int or type(A) is float:
    return abs(B-A)
  else:
    return math.sqrt(sum([   (A[i]-B[i])**2 for i in range(len(A)) ]))

distancia((1,3), (2,5))
```

Out[10]:
```
2.23606797749979
```

In [14]:
```
#Fuerza Bruta
def distancia_fuerza_bruta(L):
  mejor_distancia = 100000e10

  A,B = (),()

  for i in range(len(L)):
    for j in range(i+1, len(L)):
      if distancia(L[i],L[j]) < mejor_distancia:
                A,B=L[i],L[j]
                mejor_distancia =distancia(L[i],L[j])
  return [A,B]

distancia_fuerza_bruta(LISTA_2D)
```

Out[14]:
```
[(211, 347), (213, 347)]
```

In [33]:
```
def distancia_divide_y_venceras(L):
  #Si hay pocos por Fuerza Bruta
  if len(L) < 10:
    return distancia_fuerza_bruta(L)

  #Dividir en listas grandes
  #pivite = sum([L[i][0]for i in range(len(L))]) / len(L)


  LISTA_IZQ = sorted(L, key=lambda x: x[0])[:len(L)//2]
  LISTA_DER = sorted(L, key=lambda x: x[0])[len(L)//2:]
```

```
  PUNTOS_LISTA_IZQ = distancia_divide_y_venceras(LISTA_IZQ)
  PUNTOS_LISTA_DER = distancia_divide_y_venceras(LISTA_DER)

  return distancia_fuerza_bruta(PUNTOS_LISTA_IZQ + PUNTOS_LISTA_DER)

@calcular_tiempo
def LANZA(L):
  return distancia_divide_y_venceras(L)

SOL = LANZA(LISTA_2D[:100])

print(SOL)
```

```
 Tiempo de ejecución para algortimo: 0.0017397403717041015625000
[(698, 463), (699, 458)]
```

In [39]:
```
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]
]

#Paseo por el rio
def Precios(TARIFAS):
  N =len(TARIFAS[0])

  PRECIOS = [[9999]*N for i in [9999]*N]
  RUTAS = [[""]*N for i in [9999]*N ]

  print(PRECIOS)
  print(RUTAS)

  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(PRECIOS)

print()

print (RUTAS)

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

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

```
[[9999, 9999, 9999, 9999, 9999, 9999, 9999], [9999, 9999, 9999, 9999, 9999, 9999, 9999], [9999,
9999, 9999, 9999, 9999, 9999, 9999], [9999, 9999, 9999, 9999, 9999, 9999, 9999], [9999, 9999, 99
99, 9999, 9999, 9999, 9999], [9999, 9999, 9999, 9999, 9999, 9999, 9999], [9999, 9999, 9999, 999
9, 9999, 9999, 9999]]
[['', '', '', '', '', '', ''], ['', '', '', '', '', '', ''], ['', '', '', '', '', '', ''], ['',
'', '', '', '', '', ''], ['', '', '', '', '', '', ''], ['', '', '', '', '', '', ''], ['', '',
'', '', '', '', '']]
[[9999, 5, 4, 3, 8, 8, 11], [9999, 9999, 999, 2, 3, 8, 7], [9999, 9999, 9999, 1, 6, 4, 7], [999
9, 9999, 9999, 9999, 5, 6, 9], [9999, 9999, 9999, 9999, 9999, 999, 4], [9999, 9999, 9999, 9999,
9999, 9999, 3], [9999, 9999, 9999, 9999, 9999, 9999, 9999]]
```

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

La ruta es:
```

Out[39]: '0,0,2,5'