Branch: master ▾ 03MAIR-Algoritmos-de-Optimizacion-2019 / AG2 / Luis Fauré Navarro - AG2.ipynb

Find file Copy path

lfauren Creado mediante Colaboratory 83d6086 2 minutes ago

1 contributor

380 lines (380 sloc) | 15.1 KB

Raw Blame History

Luis Fauré Navarro - AG2

Actividad Guiada 2

Url: https://github.com/lfauren/03MAIR-Algoritmos-de-Optimizacion-2019/tree/master/AG2

In [0]:
```
from time import time
#Función para calcular el tiempo de ejecución
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))
        return resultado
    return wrapper
```

In [1]:
```
import math
import random

N=1000
LISTA_2D = [(random.randrange(1,N*10),random.randrange(1,N*10)) for _ in range(N)]
print(LISTA_2D[:5])
```

```
[(3906, 6809), (6130, 7665), (7189, 238), (2947, 1963), (1349, 3549)]
```

In [2]:
```
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[2]:
```
2.23606797749979
```

In [4]:
```
# Fuerza bruta
```

```
@calcular_tiempo
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]
  return A,B

distancia_fuerza_bruta(LISTA_2D)
```

```
Tiempo de ejecución para algoritmo: 1.134321928024292
```

Out[4]:
```
((5327, 3724), (7125, 1077))
```

In [5]:
```
def distancia_divide_y_venceras(L):
  if len(L) < 10:
    return distancia_fuerza_bruta(L)
  #pivote = 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[:15])
print(SOL)
```

```
Tiempo de ejecución para algoritmo: 5.793571472167969e-05
Tiempo de ejecución para algoritmo: 7.2479248046875e-05
Tiempo de ejecución para algoritmo: 1.8358230590820312e-05
Tiempo de ejecución para algoritmo: 0.00159454345703125
((8153, 6500), (9566, 1376))
```

In [6]:
```
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[0])
  PRECIOS = [[9999]*N for i in [9999]*N]
  RUTAS = [[""]*N for i in [""]*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(PRECIOS)
print()
print(RUTAS)

def calcular_ruta(RUTAS, desde, hasta):
  if desde == hasta:
    return desde
  else:
    return str(calcular_ruta(RUTAS,desde,RUTAS[desde][hasta])) + ',' + str(RUTAS[desde][hasta])

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

```
[[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[6]: '0,0,2,5'

In [0]:
```
# Estime conveniente agregar El problema de la asignación de tareas en Ramificación y poda expl
icado al final de la clase.
# cma: el coste mínimo absoluto alcanzable.
# cms: el coste de la mejor solución obtenida hasta el momento.
# Aunque me sale un error extraño de una función no definida se puede entender mejor el problem
a

def init_ms(c):
  n = len(c)
  cdiag1 = 0
  cdiag2 = 0
  for i in  range(n):
    cdiag1  += c[i][i]
    cdiag2  += c[i][n-i-1]
  if cdiag1 <= cdiag2:
    return [i for i in  range(n)], cdiag1
  else:
    return [n-i-1 for i in  range(n)], cdiag2

  def compute_cma(x,c):
    n = len(c)
    k = len(x)
    coste_x = 0
    for i in range(k):
      coste_x  += c[i][x[i]]
    sum_minf = sum(min(c[i][j] for j in range(n) if j not in x) for i in range(k,n))
    sum_minc = sum(min(c[i][j] for i in range(k,n)) for j in  range(n) if j not in x)
    return  coste_x + max(sum_minf ,sum_minc)

def asignar(c):
  n = len(c)
  ms, cms = init_ms(c)
  cma = compute_cma([],c)
  if cma <cms:
    nodo_raiz = [[], cma] # nodo_raiz: [tupla_vacia, cma_global]
```

```
    nodo_raiz = [[],cma] # nodo  raiz: [tupla  vacia , cma  global]
    lista = [nodo_raiz]   # lista  de  nodos  vivos
  while  len(lista) >0:       # mientras  queden  nodos  vivos
    x, cma = lista.pop() # extraemos  el nodo  más  prometedor
    tareas = [t for t in range(n) if t not in x]
    for t in  tareas:      # ramificacion
      x_new = x[:]+[t]
      cma_new = compute_cma(x_new ,c)
      if cma_new <cms:   # nodo  podado  si  cma_new  >= cms
        if len(x_new)==n: # x_new  es  solucion
          ms, cms = x_new , cma_new
          # cms  actualizado  --> poda de  nodos  con cma >=cms
          while  len(lista)>0 and  lista [0][1] >= cms:
            lista.pop(0)
        else: # insercion  en  orden (de  mayor a menor  cma)
          i=len(lista)-1
          while i>=0 and lista[i][1]<cma_new:
            i=i-1
          lista.insert(i+1,[x_new ,cma_new ])
  return ms, cms

A=[(9,2,7,8),(6,4,3,7),(5,8,1,8),(7,6,9,4)]
print(asignar(A))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-a2eba303e1ba> in <module>()
     49
     50 A=[(9,2,7,8),(6,4,3,7),(5,8,1,8),(7,6,9,4)]
---> 51 print(asignar(A))

<ipython-input-16-a2eba303e1ba> in asignar(c)
     25   n = len(c)
     26   ms, cms = init_ms(c)
---> 27   cma = compute_cma([],c)
     28   if cma <cms:
     29     nodo_raiz = [[],cma] # nodo  raiz: [tupla  vacia , cma  global]

NameError: name 'compute_cma' is not defined
```