Branch: master ▾ 03MAIR---Algoritmos-de-Optimizacion---2019 / AG1 / WilliamSteveRodriguezVillamizar_AG1.ipynb

Find file Copy path

wisrovi Creado con Colaboratory b79acc5 10 seconds ago

1 contributor

844 lines (844 sloc) | 28.8 KB

Raw Blame History

Open in Colab

AG1- Actividad Guiada 1

Nombre y Apellidos: William Steve Rodriguez Villamizar

Colab: https://colab.research.google.com/github/wisrovi/03MAIR---Algoritmos-de-Optimizacion---2019/blob/master/AG1/WilliamSteveRodriguezVillamizar_AG1.ipynb#scrollTo=cmrw2Nx-wnNu

Url: https://github.com/wisrovi/03MAIR---Algoritmos-de-Optimizacion---2019/blob/master/AG1/WilliamSteveRodriguezVillamizar_AG1.ipynb

In [0]:
```python
#Decorador de Ricardo Lebron
from functools import wraps
from time import time

def calcular_tiempo(f):
    @wraps(f)
    def cronometro(*args, **kwargs):
        t_inicial = time() #tomo la hora antes de ejecutar la funcion
        salida = f(*args, **kwargs)
        t_final = time() #tomo la hora despues de ejecutar la funcion
        print('Tiempo transcurrido (en segundos): {}'.format(t_final - t_inicial))
        return salida
    return cronometro
```

In [0]:
```python
import random
```

Torres de Hanoi (por recursividad):

In [3]:
```python
def torres_hanoi(N, desde=1, hasta=3):
  if N==1:
    print("Llevar desde " + str(desde) + " hasta " + str(hasta))
  else:
    torres_hanoi(N-1, desde, 6 - desde - hasta)
```

```
    print("Llevar desde " + str(desde) + " hasta " + str(hasta))
    torres_hanoi(N-1, 6 - desde - hasta, hasta)

@calcular_tiempo
def verMovimientos_JugarTorresHanoi(numero_discos):
  torres_hanoi(4)
verMovimientos_JugarTorresHanoi(4)
```

```
Llevar desde 1 hasta 2
Llevar desde 1 hasta 3
Llevar desde 2 hasta 3
Llevar desde 1 hasta 2
Llevar desde 3 hasta 1
Llevar desde 3 hasta 2
Llevar desde 1 hasta 2
Llevar desde 1 hasta 3
Llevar desde 2 hasta 3
Llevar desde 2 hasta 1
Llevar desde 3 hasta 1
Llevar desde 2 hasta 3
Llevar desde 1 hasta 2
Llevar desde 1 hasta 3
Llevar desde 2 hasta 3
Tiempo transcurrido (en segundos): 0.0017497539520263672
```

Fibonacci (por recursividad):

In [4]:
```
def Fibonacci(final, valor = 0, anterior = 0, listado = []):
  if valor ==0:
    listado.append(0)
    listado.append(1)
    Fibonacci(final, 1, 0, listado)
  else:
    if (valor + anterior) <= final:
      listado.append(valor + anterior)
      Fibonacci(final, valor + anterior, valor, listado)
    else:
      print(listado)

@calcular_tiempo
def VerNumerosFibonacci(numeroFinalFibonacci):
```

```
def VerNumerosFibonacci(numeroFinalFibonacci)    :
  Fibonacci(1000)

VerNumerosFibonacci(1000)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
Tiempo transcurrido (en segundos): 0.0016307830810546875
```

Inicio de la actividad guiada:

In [0]:
```
def quick_sort(A):
    if len(A)<=1:
        return A

    if len(A)==2:
        return [min(A), max (A)]

    pivote = (A[0] + A[1] + A[2])/3
    IZQ = []
    DER = []

    for i in A:
        if i <= pivote:
            IZQ.append(i)
        else:
            DER.append(i)
    return quick_sort(IZQ) + quick_sort(DER)

@calcular_tiempo
def QS(A):
    return quick_sort(A)
```

In [6]:
```
A = [9187, 244, 4054, 9222, 8373, 4993, 5265, 5470, 4519, 7182, 2035, 3506, 4337, 7580, 25
54, 2824, 8357, 4447, 7379]
A = list(map(lambda x: random.randrange(1,10000), range(1,600)))

print(QS(A))
```

```
Tiempo transcurrido (en segundos): 0.0016117095947265625
[15, 19, 33, 49, 50, 63, 72, 76, 89, 98, 101, 116, 131, 159, 161, 176, 187, 199, 237, 275,
```

```
277, 279, 294, 295, 302, 330, 334, 341, 369, 414, 428, 430, 449, 456, 476, 490, 526, 614,
620, 642, 658, 673, 717, 722, 739, 741, 771, 805, 813, 819, 848, 901, 928, 932, 932, 940,
960, 970, 1000, 1033, 1060, 1080, 1083, 1089, 1109, 1119, 1163, 1170, 1217, 1235, 1238, 12
77, 1278, 1333, 1348, 1359, 1361, 1374, 1409, 1427, 1434, 1446, 1462, 1480, 1487, 1498, 15
12, 1516, 1519, 1531, 1547, 1552, 1563, 1567, 1607, 1639, 1647, 1672, 1708, 1748, 1754, 17
57, 1768, 1769, 1773, 1781, 1823, 1831, 1837, 1849, 1912, 1939, 1981, 2002, 2008, 2009, 20
30, 2128, 2147, 2159, 2200, 2202, 2203, 2246, 2277, 2322, 2328, 2336, 2367, 2386, 2386, 24
01, 2409, 2429, 2445, 2452, 2457, 2470, 2473, 2497, 2502, 2510, 2530, 2552, 2560, 2570, 25
87, 2600, 2601, 2614, 2633, 2638, 2639, 2663, 2678, 2682, 2698, 2701, 2704, 2730, 2732, 27
36, 2743, 2793, 2820, 2824, 2827, 2840, 2851, 2854, 2870, 2882, 2893, 2894, 2942, 2984, 29
88, 3005, 3010, 3032, 3051, 3063, 3063, 3066, 3094, 3099, 3101, 3122, 3138, 3149, 3176, 31
77, 3188, 3209, 3225, 3259, 3277, 3281, 3288, 3293, 3339, 3380, 3427, 3434, 3441, 3461, 34
69, 3470, 3496, 3516, 3555, 3577, 3666, 3673, 3698, 3705, 3743, 3761, 3761, 3808, 3811, 38
30, 3881, 3899, 3910, 3930, 3944, 3945, 3951, 3998, 4004, 4034, 4039, 4092, 4107, 4115, 41
18, 4164, 4171, 4186, 4188, 4216, 4245, 4271, 4301, 4301, 4321, 4328, 4332, 4333, 4338, 44
16, 4426, 4430, 4463, 4487, 4492, 4532, 4563, 4571, 4605, 4610, 4629, 4651, 4652, 4657, 47
10, 4712, 4726, 4754, 4801, 4806, 4823, 4825, 4828, 4832, 4898, 4913, 4919, 4923, 4933, 49
69, 4970, 4993, 5008, 5058, 5074, 5095, 5099, 5104, 5128, 5135, 5139, 5163, 5164, 5172, 52
26, 5226, 5255, 5295, 5316, 5347, 5372, 5374, 5381, 5385, 5390, 5430, 5430, 5432, 5475, 54
85, 5493, 5508, 5520, 5524, 5534, 5543, 5573, 5585, 5594, 5600, 5620, 5644, 5646, 5721, 57
29, 5738, 5746, 5750, 5762, 5775, 5776, 5784, 5785, 5793, 5797, 5808, 5814, 5824, 5827, 58
33, 5859, 5870, 5883, 5890, 5916, 5941, 5991, 6001, 6032, 6050, 6106, 6130, 6152, 6177, 61
79, 6196, 6205, 6214, 6239, 6261, 6267, 6270, 6302, 6307, 6333, 6344, 6354, 6360, 6369, 63
97, 6417, 6422, 6426, 6434, 6436, 6442, 6474, 6489, 6494, 6530, 6539, 6540, 6549, 6575, 66
01, 6627, 6641, 6648, 6651, 6676, 6680, 6690, 6723, 6728, 6732, 6736, 6759, 6824, 6832, 68
35, 6862, 6877, 6899, 6912, 6932, 7001, 7004, 7058, 7084, 7092, 7119, 7139, 7154, 7157, 71
67, 7180, 7200, 7216, 7218, 7228, 7246, 7253, 7283, 7288, 7288, 7302, 7303, 7329, 7334, 73
40, 7353, 7382, 7391, 7402, 7466, 7481, 7494, 7495, 7499, 7510, 7531, 7549, 7550, 7586, 76
05, 7614, 7625, 7628, 7658, 7664, 7665, 7669, 7676, 7678, 7733, 7734, 7772, 7782, 7795, 78
11, 7815, 7818, 7822, 7825, 7829, 7832, 7845, 7850, 7852, 7869, 7890, 7902, 7906, 7916, 79
49, 7981, 8019, 8028, 8059, 8064, 8092, 8096, 8117, 8121, 8139, 8160, 8181, 8189, 8242, 82
54, 8254, 8274, 8277, 8278, 8308, 8342, 8364, 8380, 8404, 8429, 8434, 8438, 8457, 8460, 84
93, 8499, 8499, 8517, 8575, 8583, 8593, 8615, 8638, 8652, 8659, 8686, 8704, 8705, 8740, 87
41, 8743, 8745, 8780, 8799, 8802, 8804, 8809, 8827, 8840, 8864, 8888, 8898, 8903, 8919, 89
31, 8935, 8939, 8948, 8960, 8975, 8986, 9008, 9027, 9080, 9084, 9089, 9160, 9168, 9169, 92
11, 9244, 9249, 9276, 9294, 9300, 9321, 9324, 9361, 9372, 9381, 9393, 9414, 9418, 9422, 94
99, 9501, 9510, 9518, 9530, 9534, 9570, 9570, 9575, 9585, 9587, 9588, 9602, 9610, 9611, 96
29, 9671, 9706, 9754, 9778, 9786, 9796, 9828, 9850, 9854, 9875, 9886, 9916, 9928, 9928, 99
29, 9942, 9976]
```

Cambio Monedas:

In [0]:
```
@calcular_tiempo
def cambio_monedas(CANTIDAD, SISTEMA):
  SOLUCION = [0 for i in range(len(SISTEMA))]
  VALOR_ACUMULADO = 0

  for i in range(len(SISTEMA)):
    monedas = int((CANTIDAD - VALOR_ACUMULADO)/ SISTEMA[i])
    SOLUCION[i] = monedas
    VALOR_ACUMULADO += monedas * SISTEMA[i]
    if VALOR_ACUMULADO == CANTIDAD:
      return SOLUCION

  return SOLUCION
```

In [8]:
```
SISTEMA = [25, 10, 5, 1]
CANTIDAD = 234
solucion = cambio_monedas(CANTIDAD, SISTEMA)

print("Sistema: ", end="")
print(SISTEMA)

print("Solucion: ", end="")
print(solucion, end="")
print(" para una cantidad de: ", CANTIDAD)
```

```
Tiempo transcurrido (en segundos): 8.106231689453125e-06
Sistema: [25, 10, 5, 1]
Solucion: [9, 0, 1, 4] para una cantidad de:  234
```

Reinas:

In [0]:
```
#Algoritmo de vuelta atras para el problema de N reinas
def es_prometedora(solucion, etapa):
  for i in range(etapa + 1):
    if solucion.count(solucion[i]) > 1:
      return False

    #verificar si son individuales
```

```
    #verificar si son individuales
    for j in range(i+1, etapa + 1):
      if abs(i-j) == abs(solucion[i]-solucion[j]):
        return False
  return True


def reinas(N, solucion = [], etapa=0):
  if len(solucion)==0:
    solucion = [0 for i in range(N)]

  for i in range(1, N+1):
    solucion[etapa] = i

    if es_prometedora(solucion, etapa):
      if etapa == N-1:
        print("\n\nLa solucion es: ", end="")
        print(solucion)

      else:
        reinas(N, solucion, etapa+1)

    else:
      None

    solucion[etapa] = 0

@calcular_tiempo
def BuscarPosicionReinasEnTablero(NumeroReinas):
  reinas(NumeroReinas)
```

In [10]:
```
BuscarPosicionReinasEnTablero(8)
```

```
La solucion es: [1, 5, 8, 6, 3, 7, 2, 4]


La solucion es: [1, 6, 8, 3, 7, 4, 2, 5]
```

```
La solucion es: [1, 7, 4, 6, 8, 2, 5, 3]

La solucion es: [1, 7, 5, 8, 2, 4, 6, 3]

La solucion es: [2, 4, 6, 8, 3, 1, 7, 5]

La solucion es: [2, 5, 7, 1, 3, 8, 6, 4]

La solucion es: [2, 5, 7, 4, 1, 8, 6, 3]

La solucion es: [2, 6, 1, 7, 4, 8, 3, 5]

La solucion es: [2, 6, 8, 3, 1, 4, 7, 5]

La solucion es: [2, 7, 3, 6, 8, 5, 1, 4]

La solucion es: [2, 7, 5, 8, 1, 4, 6, 3]

La solucion es: [2, 8, 6, 1, 3, 5, 7, 4]

La solucion es: [3, 1, 7, 5, 8, 2, 4, 6]

La solucion es: [3, 5, 2, 8, 1, 7, 4, 6]

La solucion es: [3, 5, 2, 8, 6, 4, 7, 1]

La solucion es: [3, 5, 7, 1, 4, 2, 8, 6]
```

```
La solucion es: [3, 5, 8, 4, 1, 7, 2, 6]

La solucion es: [3, 6, 2, 5, 8, 1, 7, 4]

La solucion es: [3, 6, 2, 7, 1, 4, 8, 5]

La solucion es: [3, 6, 2, 7, 5, 1, 8, 4]

La solucion es: [3, 6, 4, 1, 8, 5, 7, 2]

La solucion es: [3, 6, 4, 2, 8, 5, 7, 1]

La solucion es: [3, 6, 8, 1, 4, 7, 5, 2]

La solucion es: [3, 6, 8, 1, 5, 7, 2, 4]

La solucion es: [3, 6, 8, 2, 4, 1, 7, 5]

La solucion es: [3, 7, 2, 8, 5, 1, 4, 6]

La solucion es: [3, 7, 2, 8, 6, 4, 1, 5]

La solucion es: [3, 8, 4, 7, 1, 6, 2, 5]

La solucion es: [4, 1, 5, 8, 2, 7, 3, 6]
```

```
La solucion es: [4, 1, 5, 8, 6, 3, 7, 2]

La solucion es: [4, 2, 5, 8, 6, 1, 3, 7]

La solucion es: [4, 2, 7, 3, 6, 8, 1, 5]

La solucion es: [4, 2, 7, 3, 6, 8, 5, 1]

La solucion es: [4, 2, 7, 5, 1, 8, 6, 3]

La solucion es: [4, 2, 8, 5, 7, 1, 3, 6]

La solucion es: [4, 2, 8, 6, 1, 3, 5, 7]

La solucion es: [4, 6, 1, 5, 2, 8, 3, 7]

La solucion es: [4, 6, 8, 2, 7, 1, 3, 5]

La solucion es: [4, 6, 8, 3, 1, 7, 5, 2]

La solucion es: [4, 7, 1, 8, 5, 2, 6, 3]

La solucion es: [4, 7, 3, 8, 2, 5, 1, 6]

La solucion es: [4, 7, 5, 2, 6, 1, 3, 8]
```

```
La solucion es: [4, 7, 5, 3, 1, 6, 8, 2]

La solucion es: [4, 8, 1, 3, 6, 2, 7, 5]

La solucion es: [4, 8, 1, 5, 7, 2, 6, 3]

La solucion es: [4, 8, 5, 3, 1, 7, 2, 6]

La solucion es: [5, 1, 4, 6, 8, 2, 7, 3]

La solucion es: [5, 1, 8, 4, 2, 7, 3, 6]

La solucion es: [5, 1, 8, 6, 3, 7, 2, 4]

La solucion es: [5, 2, 4, 6, 8, 3, 1, 7]

La solucion es: [5, 2, 4, 7, 3, 8, 6, 1]

La solucion es: [5, 2, 6, 1, 7, 4, 8, 3]

La solucion es: [5, 2, 8, 1, 4, 7, 3, 6]

La solucion es: [5, 3, 1, 6, 8, 2, 4, 7]

La solucion es: [5, 3, 1, 7, 2, 8, 6, 4]

La solucion es: [5, 3, 8, 4, 7, 1, 6, 2]
```

```
La solucion es: [5, 7, 1, 3, 8, 6, 4, 2]

La solucion es: [5, 7, 1, 4, 2, 8, 6, 3]

La solucion es: [5, 7, 2, 4, 8, 1, 3, 6]

La solucion es: [5, 7, 2, 6, 3, 1, 4, 8]

La solucion es: [5, 7, 2, 6, 3, 1, 8, 4]

La solucion es: [5, 7, 4, 1, 3, 8, 6, 2]

La solucion es: [5, 8, 4, 1, 3, 6, 2, 7]

La solucion es: [5, 8, 4, 1, 7, 2, 6, 3]

La solucion es: [6, 1, 5, 2, 8, 3, 7, 4]

La solucion es: [6, 2, 7, 1, 3, 5, 8, 4]

La solucion es: [6, 2, 7, 1, 4, 8, 5, 3]

La solucion es: [6, 3, 1, 7, 5, 8, 2, 4]

La solucion es: [6, 3, 1, 8, 4, 2, 7, 5]
```

```
La solucion es: [6, 3, 1, 8, 5, 2, 4, 7]

La solucion es: [6, 3, 5, 7, 1, 4, 2, 8]

La solucion es: [6, 3, 5, 8, 1, 4, 2, 7]

La solucion es: [6, 3, 7, 2, 4, 8, 1, 5]

La solucion es: [6, 3, 7, 2, 8, 5, 1, 4]

La solucion es: [6, 3, 7, 4, 1, 8, 2, 5]

La solucion es: [6, 4, 1, 5, 8, 2, 7, 3]

La solucion es: [6, 4, 2, 8, 5, 7, 1, 3]

La solucion es: [6, 4, 7, 1, 3, 5, 2, 8]

La solucion es: [6, 4, 7, 1, 8, 2, 5, 3]

La solucion es: [6, 8, 2, 4, 1, 7, 5, 3]

La solucion es: [7, 1, 3, 8, 6, 4, 2, 5]

La solucion es: [7, 2, 4, 1, 8, 5, 3, 6]
```

```
La solucion es: [7, 2, 6, 3, 1, 4, 8, 5]


La solucion es: [7, 3, 1, 6, 8, 5, 2, 4]


La solucion es: [7, 3, 8, 2, 5, 1, 6, 4]


La solucion es: [7, 4, 2, 5, 8, 1, 3, 6]


La solucion es: [7, 4, 2, 8, 6, 1, 3, 5]


La solucion es: [7, 5, 3, 1, 6, 8, 2, 4]


La solucion es: [8, 2, 4, 1, 7, 5, 3, 6]


La solucion es: [8, 2, 5, 3, 1, 7, 4, 6]


La solucion es: [8, 3, 1, 6, 2, 5, 7, 4]


La solucion es: [8, 4, 1, 3, 6, 2, 7, 5]
Tiempo transcurrido (en segundos): 0.0673530101776123
```

Distancia de dos puntos:

In [0]:
```
import math
```

In [12]:
```
numeroDimensiones = 3

def GenerarLista(dimension=1):
  if dimension == 1:
    return list(random.randrange(1,10000) for x in range(1,1000))
```

```
  if dimension == 2:
    return list((random.randrange(1,10000),random.randrange(1,10000)) for x in range(1,100
0))
  if dimension == 3:
    return list((random.randrange(1,10000),random.randrange(1,10000),random.randrange(1,10
000)) for x in range(1,1000))


A = GenerarLista(numeroDimensiones)

def DistanciaPuntos(listaPuntos, dimension = 1):
  distancia_menor = 10e100
  puntos_distancia_menor = [0,0]
  for indice_punto in range(len(listaPuntos)):
    if indice_punto>0:
      punto_actual = listaPuntos[indice_punto]
      punto_anterior = listaPuntos[indice_punto - 1]
      if dimension == 1:
        try:
          distancia = abs(punto_actual - punto_anterior)
        except:
          print("A ocurrido un error, la lista ingresada no tiene una dimension.")
          return ""
      if dimension == 2:
        try:
          x1,y1 = punto_actual
          x2,y2 = punto_anterior
          cateto1 = abs(x1 - x2)
          cateto2 = abs(y1 - y2)
          distancia = math.sqrt(cateto1**2+cateto2**2)
        except:
          print("A ocurrido un error, la lista ingresada no tiene dos dimensiones.")
          return ""
      if dimension == 3:
        try:
          x1,y1,z1 = punto_actual
          x2,y2,z2 = punto_anterior
          cateto1 = abs(x1 - x2)
          cateto2 = abs(y1 - y2)
          cateto3 = abs(z1 - z2)
```

```
            distancia = math.sqrt(cateto1+cateto2+cateto3)
          except:
            print("A ocurrido un error, la lista ingresada no tiene dos dimensiones.")
            return ""

        if distancia < distancia_menor:
            distancia_menor = distancia
            puntos_distancia_menor[0] = punto_actual
            puntos_distancia_menor[1] = punto_anterior
    print("Punto1 : ", end="")
    print(puntos_distancia_menor[0], end="")
    print(" - Punto2 : ", end="")
    print(puntos_distancia_menor[1], end="")
    print(" -> Distancia : ", end="")
    print(distancia_menor)




    return distancia_menor, (puntos_distancia_menor[0], puntos_distancia_menor[1])

print("\nUna dimensión:")
solucion = DistanciaPuntos(GenerarLista(1), 1)
print("\nDos dimensiones:")
solucion = DistanciaPuntos(GenerarLista(2), 2)
print("\nTres dimensiones:")
solucion = DistanciaPuntos(GenerarLista(3), 3)
```

```
Una dimensión:
Punto1 : 6084 - Punto2 : 6083 -> Distancia : 1

Dos dimensiones:
Punto1 : (9258, 3634) - Punto2 : (9312, 3569) -> Distancia : 84.50443775329198
```