Branch: master ▾ 03MAIR-Algoritmos-de-optimizacion / AG1 / AG1_Isabel_Vazquez.ipynb

Find file Copy path

404isabel Creado con Colaboratory 9f5d16b a minute ago

1 contributor

389 lines (389 sloc) | 10.7 KB

Raw Blame History

AG1 - Actividad Guiada 1

Isabel Vázquez Trigás

https://github.com/404isabel/03MAIR-Algoritmos-de-optimizacion/tree/master/AG1

Función para calcular tiempo

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("Tiempo de ejecución para algoritmo: "+str(tiempo))
        print("\r\n Tiempo de ejecución para algoritmo: "+"{0:.25f}".format(tiempo))
        return resultado

    return wrapper
```

Algoritmo quick_sort

In [3]:
```
#quick_sort

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

A = [9187, 244, 4054, 9222, 8373, 4993, 5265, 5470, 4519, 7182, 2035, 3506, 4337, 7580, 2554, 2
824, 8357, 4447, 7379]


def quick_sort(A):
  if len(A)==1:
    return A
```

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

  else: #Modificación: se mete este código en un else
    izq=[]
    der=[]


    #mejorable la elección del pivote (máx-min)/2. Igual calculando de una manera eficiente la
 mediana. (La mediana recorre la lista completa, y eso incrementa el valor de la complejidad)

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

    #Mirar si puedo hacerlo con un while, para eliminar iteraciones
    for i in A:
      if i<pivote:
        izq.append(i)
      else:
        der.append(i)

  return quick_sort(izq)+quick_sort(der)

quick_sort(A)
print(QS(A))
```

```
 Tiempo de ejecución para algoritmo: 0.000025033950805664062500
[244, 2035, 2554, 2824, 3506, 4054, 4337, 4447, 4519, 4993, 5265, 5470, 7182, 7379, 7580, 8357,
8373, 9187, 9222]
```

Cálculo de monedas

In [94]:
```
@calcular_tiempo
def cambio_monedas(cantidad,sistema):

  print(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



sistema=[25,10,5,1]
#Devuelve el número de monedas de cada posición
print(cambio_monedas(77,sistema))
```

```
[25, 10, 5, 1]
Tiempo de ejecución para algoritmo: 0.000342845916748046875000
[3, 0, 0, 2]
```

Algoritmo modificado en el cual se devuelven las monedas necesarias

In [95]:
```
#Mi algoritmo, devolviendo en un array las monedas necesarias.
@calcular_tiempo
def cambioMonedas(cantidad, monedas):
  monedas.sort(reverse=True)
  resultado=[]
  i=0
  total=0
  while(i<len(monedas) and total<cantidad):
    if(cantidad >= monedas[i] and monedas[i]+total<=cantidad):
      resultado.append(monedas[i])
      total+=monedas[i]
    else:
      i+=1
  return resultado

r1=cambioMonedas(77,[25,10,5,1])
print(r1)
```

```
Tiempo de ejecución para algoritmo: 0.000009536743164062500000
[25, 25, 25, 1, 1]
```

Algoritmo de las 4 reinas

Algoritmo de las 4 reinas

In [36]:
```
N=4

solucion = [0 for i in range(N)]

etapa=0

@calcular_tiempo
def tiempoReinas(N,solucion,etapa):
  return reinas(N,solucion,etapa)

def es_prometedora(solucion,etapa):
  for i in range(etapa+1):
    if solucion.count(solucion[i])>1:
      return False

    #Verificar las diagonales
    for j in range(i+1,etapa+1):
      if abs(i-j) == abs(solucion[i]-solucion[j]):
        return False
  return True

def escribe(S):
  n = len(S)
  for x in range(n):
    print("")
    for i in range(n):
      if solucion[i] == x+1:
        print(" X " , end="")
      else:
        print(" - ", end="")


def reinas(N, solucion, etapa):

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

```
        print("\r\n La solución es \r\n")
        print(solucion)
        escribe(solucion)
      else:
        #es prometedora
        reinas(N,solucion, etapa+1)

    #Si no es prometedora, no hago nada, he quitado el else
    #else:
      #no es prometedora
     # None
    solucion[etapa] = 0

#reinas(N,solucion,etapa)

print(tiempoReinas(N,solucion,etapa))
```

```
 La solución es

[2, 4, 1, 3]

 -  -  X  -
 X  -  -  -
 -  -  -  X
 -  X  -  -
 La solución es

[3, 1, 4, 2]

 -  X  -  -
 -  -  -  X
 X  -  -  -
 -  -  X  -
 Tiempo de ejecución para algoritmo: 0.007866621017456054687500
None
```

In [0]:

In [0]: