



Why GitHub? ▾ Enterprise Explore ▾ Marketplace Pricing ▾

Search



Sign in

Sign up

404isabel / 03MAIR-Algoritmos-de-optimizacion

Watch

1

★ Star

0

Fork

0

<> Code

Issues 0

Pull requests 0

Projects 0

Insights

Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

Branch: master ▾

03MAIR-Algoritmos-de-optimizacion / AG1 / AG1\_Isabel\_Vazquez.ipynb

Find file

Copy path

404isabel Creado con Colaboratory

44840a4 a minute ago

1 contributor

431 lines (431 sloc) | 12.3 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("\r\n Tiempo de ejecución para algoritmo: "+"{0:.25f}".format(tiempo))
        return resultado

    return wrapper
```

### **Algoritmo quick\_sort**

```
In [35]: #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=[]

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

    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: 0.0000283718109130859375000  
 [244, 2035, 2554, 2824, 3506, 4054, 4337, 4447, 4519, 4993, 5265, 5470, 7182, 7379, 7580, 8357, 8373, 9187, 9222]

### Algoritmo quick\_sort, mediante técnica de reposicionamiento del pivote

En este caso se va recorriendo la lista de izquierda a derecha, y de derecha a izquierda El pivote se pone donde sabemos que todos los elementos de la izquierda son menores, y los de la derecha mayores

```

In [38]: def quicksortIndices(A, inicio, fin):
        # definimos los índices y calculamos el pivote
        i = inicio
        j = fin
        pivote = (A[i] + A[j]) / 2

        while i < j:
            while A[i] < pivote:
                i+=1
            while A[j] > pivote:
                j-=1

```

```

    # i y j están al lado o son el mismo
    if i <= j:
        x = A[j]
        A[j] = A[i]
        A[i] = x
        i += 1
        j -= 1

    # si inicio es menor que j o fin es mayor que i, seguimos iterando sobre la lista
    if inicio < j :
        A = quicksortIndices(A, inicio, j)
    elif fin > i:
        A = quicksortIndices(A, i, fin)

    return A

@calcular_tiempo
def QST(A, inicio, fin):
    return quicksortIndices(A, inicio, fin)

print(QST(A, 0, len(A) - 1))

```

Tiempo de ejecución para algoritmo: 0.0000104904174804687500000  
 [244, 2035, 2554, 3506, 2824, 4337, 4447, 4054, 4519, 7182, 5470, 5265, 4993, 7580, 7379, 8373, 8357, 9222, 9187]

## Cálculo de monedas

```

In [3]: @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

```

```

    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: 0.0036876201629638671875000  
[3, 0, 0, 2]

#### **Algoritmo modificado en el cual se devuelven las monedas necesarias**

In [5]: *#Mi algoritmo, devolviendo en un array las monedas necesarias, en vez de las monedas por posición*

```

@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: 0.0000090599060058593750000  
[25, 25, 25, 1, 1]

## Algoritmo de las 4 reinas

```
In [31]: N=2

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

etapa=0

@calcular_tiempo
def tiempoReinas(N,solucion,etapa):
    #Modificación: Añado comprobación para problema con 2 reinas
    if(N==2):
        print("No se puede resolver para 2 reinas")
        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):
```

```
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)

    #Modificación: quito este else
    #else:
    # None
    solucion[etapa] = 0

#reinas(N,solucion,etapa)

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

No se puede resolver para 2 reinas

Tiempo de ejecución para algoritmo: 0.0005507469177246093750000  
None

In [0]:

