

Algoritmos - Actividad Guiada 1

Nombre: Johnny Andres Illescas Fernandez

URL: https://github.com/AndresGin/Algoritmos_de_Optimizacion.git

Torres de Hanoi con Divide y vencerás

```
In [1]: def Torres_Hanoi(N, desde, hasta):
    if N ==1 :
        print("Lleva la ficha " ,desde , " hasta " , hasta )

    else:
        #Torres_Hanoi(N-1, desde, 6-desde-hasta )
        Torres_Hanoi(N-1, desde, 6-desde-hasta )
        print("Lleva la ficha " ,desde , " hasta " , hasta )
        #Torres_Hanoi(N-1,6-desde-hasta, hasta )
        Torres_Hanoi(N-1, 6-desde-hasta , hasta )

    Torres_Hanoi(3, 1 , 3)
```

```
Lleva la ficha 1 hasta 3
Lleva la ficha 1 hasta 2
Lleva la ficha 3 hasta 2
Lleva la ficha 1 hasta 3
Lleva la ficha 2 hasta 1
Lleva la ficha 2 hasta 3
Lleva la ficha 1 hasta 3
```

```
In [2]: #Sucesión de Fibonacci
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
#Calculo del termino n-simo de La sucesión de Fibonacci
def Fibonacci(N:int):
    if N < 2:
        return 1
    else:
        return Fibonacci(N-1)+Fibonacci(N-2)

    Fibonacci(5)
```

Out[2]: 8

Devolución de cambio por técnica voraz

```
In [3]: def cambio_monedas(N, SM):
    SOLUCION = [0]*len(SM)    #SOLUCION = [0,0,0,0,..]
    ValorAcumulado = 0

    for i,valor in enumerate(SM):
        monedas = (N-ValorAcumulado)//valor
        SOLUCION[i] = monedas
        ValorAcumulado = ValorAcumulado + monedas*valor
```

```

    if ValorAcumulado == N:
        return SOLUCION

cambio_monedas(15,[25,10,5,1])

```

Out[3]: [0, 1, 1, 0]

N-Reinas por técnica de vueta atrás

```

In [4]: def describe(S):
    n = len(S)
    for x in range(n):
        print("")
        for i in range(n):
            if S[i] == x+1:
                print(" X ", end="")
            else:
                print(" - ", end="")

def es_prometedora(SOLUCION,etapa):
    #print(SOLUCION)
    #Si la solución tiene dos valores iguales no es valida => Dos reinas en la mis
    for i in range(etapa+1):
        #print("El valor " + str(SOLUCION[i]) + " está " + str(SOLUCION.count(SOLUCION[i]))
        if SOLUCION.count(SOLUCION[i]) > 1:
            return False

    #Verifica las diagonales
    for j in range(i+1, etapa +1 ):
        #print("Comprobando diagonal de " + str(i) + " y " + str(j))
        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(solucion)
                #describe(solucion)
                print()
            else:
                reinas(N, solucion, etapa+1)
        else:
            None

        solucion[etapa] = 0

reinas(8)

```

[1, 5, 8, 6, 3, 7, 2, 4]
[1, 6, 8, 3, 7, 4, 2, 5]
[1, 7, 4, 6, 8, 2, 5, 3]
[1, 7, 5, 8, 2, 4, 6, 3]
[2, 4, 6, 8, 3, 1, 7, 5]
[2, 5, 7, 1, 3, 8, 6, 4]
[2, 5, 7, 4, 1, 8, 6, 3]
[2, 6, 1, 7, 4, 8, 3, 5]
[2, 6, 8, 3, 1, 4, 7, 5]
[2, 7, 3, 6, 8, 5, 1, 4]
[2, 7, 5, 8, 1, 4, 6, 3]
[2, 8, 6, 1, 3, 5, 7, 4]
[3, 1, 7, 5, 8, 2, 4, 6]
[3, 5, 2, 8, 1, 7, 4, 6]
[3, 5, 2, 8, 6, 4, 7, 1]
[3, 5, 7, 1, 4, 2, 8, 6]
[3, 5, 8, 4, 1, 7, 2, 6]
[3, 6, 2, 5, 8, 1, 7, 4]
[3, 6, 2, 7, 1, 4, 8, 5]
[3, 6, 2, 7, 5, 1, 8, 4]
[3, 6, 4, 1, 8, 5, 7, 2]
[3, 6, 4, 2, 8, 5, 7, 1]
[3, 6, 8, 1, 4, 7, 5, 2]
[3, 6, 8, 1, 5, 7, 2, 4]
[3, 6, 8, 2, 4, 1, 7, 5]
[3, 7, 2, 8, 5, 1, 4, 6]
[3, 7, 2, 8, 6, 4, 1, 5]
[3, 8, 4, 7, 1, 6, 2, 5]
[4, 1, 5, 8, 2, 7, 3, 6]
[4, 1, 5, 8, 6, 3, 7, 2]

[4, 2, 5, 8, 6, 1, 3, 7]

[4, 2, 7, 3, 6, 8, 1, 5]

[4, 2, 7, 3, 6, 8, 5, 1]

[4, 2, 7, 5, 1, 8, 6, 3]

[4, 2, 8, 5, 7, 1, 3, 6]

[4, 2, 8, 6, 1, 3, 5, 7]

[4, 6, 1, 5, 2, 8, 3, 7]

[4, 6, 8, 2, 7, 1, 3, 5]

[4, 6, 8, 3, 1, 7, 5, 2]

[4, 7, 1, 8, 5, 2, 6, 3]

[4, 7, 3, 8, 2, 5, 1, 6]

[4, 7, 5, 2, 6, 1, 3, 8]

[4, 7, 5, 3, 1, 6, 8, 2]

[4, 8, 1, 3, 6, 2, 7, 5]

[4, 8, 1, 5, 7, 2, 6, 3]

[4, 8, 5, 3, 1, 7, 2, 6]

[5, 1, 4, 6, 8, 2, 7, 3]

[5, 1, 8, 4, 2, 7, 3, 6]

[5, 1, 8, 6, 3, 7, 2, 4]

[5, 2, 4, 6, 8, 3, 1, 7]

[5, 2, 4, 7, 3, 8, 6, 1]

[5, 2, 6, 1, 7, 4, 8, 3]

[5, 2, 8, 1, 4, 7, 3, 6]

[5, 3, 1, 6, 8, 2, 4, 7]

[5, 3, 1, 7, 2, 8, 6, 4]

[5, 3, 8, 4, 7, 1, 6, 2]

[5, 7, 1, 3, 8, 6, 4, 2]

[5, 7, 1, 4, 2, 8, 6, 3]

[5, 7, 2, 4, 8, 1, 3, 6]

[5, 7, 2, 6, 3, 1, 4, 8]

[5, 7, 2, 6, 3, 1, 8, 4]

[5, 7, 4, 1, 3, 8, 6, 2]

[5, 8, 4, 1, 3, 6, 2, 7]

[5, 8, 4, 1, 7, 2, 6, 3]

[6, 1, 5, 2, 8, 3, 7, 4]

[6, 2, 7, 1, 3, 5, 8, 4]

[6, 2, 7, 1, 4, 8, 5, 3]

[6, 3, 1, 7, 5, 8, 2, 4]

[6, 3, 1, 8, 4, 2, 7, 5]

[6, 3, 1, 8, 5, 2, 4, 7]

[6, 3, 5, 7, 1, 4, 2, 8]

[6, 3, 5, 8, 1, 4, 2, 7]

[6, 3, 7, 2, 4, 8, 1, 5]

[6, 3, 7, 2, 8, 5, 1, 4]

[6, 3, 7, 4, 1, 8, 2, 5]

[6, 4, 1, 5, 8, 2, 7, 3]

[6, 4, 2, 8, 5, 7, 1, 3]

[6, 4, 7, 1, 3, 5, 2, 8]

[6, 4, 7, 1, 8, 2, 5, 3]

[6, 8, 2, 4, 1, 7, 5, 3]

[7, 1, 3, 8, 6, 4, 2, 5]

[7, 2, 4, 1, 8, 5, 3, 6]

[7, 2, 6, 3, 1, 4, 8, 5]

[7, 3, 1, 6, 8, 5, 2, 4]

[7, 3, 8, 2, 5, 1, 6, 4]

[7, 4, 2, 5, 8, 1, 3, 6]

[7, 4, 2, 8, 6, 1, 3, 5]

[7, 5, 3, 1, 6, 8, 2, 4]

[8, 2, 4, 1, 7, 5, 3, 6]

[8, 2, 5, 3, 1, 7, 4, 6]

[8, 3, 1, 6, 2, 5, 7, 4]

[8, 4, 1, 3, 6, 2, 7, 5]

Viaje por el rio. Programación dinámica

```
In [5]: 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):
#####
    #Total de Nodos
    N = len(TARIFAS[0])

    #Inicialización de la tabla de precios
    PRECIOS = [ [9999]*N for i in range(N)]
    RUTA = [ [""]*N for i in range(N)]

    for i in range(0,N-1):
        RUTA[i][i] = i                #Para ir de i a i se "pasa por i"
        PRECIOS[i][i] = 0             #Para ir de i a i se se paga 0
        for j in range(i+1, N):
            MIN = TARIFAS[i][j]
            RUTA[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] )
                    RUTA[i][j] = k                #Anota que para ir de i a j hay que pasar po
            PRECIOS[i][j] = MIN

    return PRECIOS,RUTA
#####

PRECIOS,RUTA = Precios(TARIFAS)
#print(PRECIOS[0][6])

print("PRECIOS")
for i in range(len(TARIFAS)):
    print(PRECIOS[i])

print("\nRUTA")
for i in range(len(TARIFAS)):
    print(RUTA[i])

#Determinar la ruta con Recursividad
def calcular_ruta(RUTA, desde, hasta):
```

```

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

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

```

PRECIOS

```

[0, 5, 4, 3, 8, 8, 11]
[9999, 0, 999, 2, 3, 8, 7]
[9999, 9999, 0, 1, 6, 4, 7]
[9999, 9999, 9999, 0, 5, 6, 9]
[9999, 9999, 9999, 9999, 0, 999, 4]
[9999, 9999, 9999, 9999, 9999, 0, 3]
[9999, 9999, 9999, 9999, 9999, 9999, 9999]

```

RUTA

```

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

```

La ruta es:

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

Algoritmo de fuerza bruta o backtracking aplicado a resolver el problema de la permutación de un conjunto de números:

Problema Generar todas las permutaciones posibles de un conjunto de números.

Por ejemplo, el conjunto [1,2,3] debe generar todas las combinaciones posibles de estos números.

Explicación

El enfoque de fuerza bruta consiste en generar todas las posibles combinaciones de los números y verificar que se cumplan las condiciones. El algoritmo recorre todos los caminos posibles (es decir, todas las permutaciones) y explora todas las opciones sin eliminar ninguna.

```

In [6]: def permutaciones(arr, l, r):
        if l == r:
            print(arr)
        else:
            for i in range(l, r + 1):
                # Intercambiar los elementos arr[l] y arr[i]

```

```
arr[l], arr[i] = arr[i], arr[l]
# Llamada recursiva con el siguiente índice
permutaciones(arr, l + 1, r)
# Revertir el intercambio para volver al estado original
arr[l], arr[i] = arr[i], arr[l]

# Ejemplo de uso:
arr = [1, 2, 3]
n = len(arr)
permutaciones(arr, 0, n - 1)
```

```
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 2, 1]
[3, 1, 2]
```