Actividad Guiada 2 de Algoritmos de Optimizacion

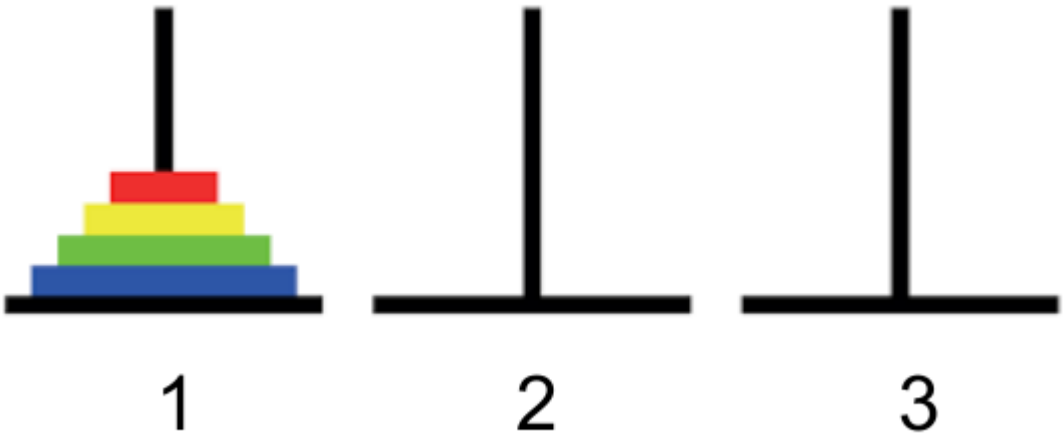
Nombre: Carlos Adrian Espinoza Alvarez

https://colab.research.google.com/drive/1HSEQEYyaBuXRs9GNzURs6MMGpUbcDJ8V?usp=sharing

https://github.com/AdrianEspinoza92/03MIAR---Algoritmos-de-Optimizacion.git

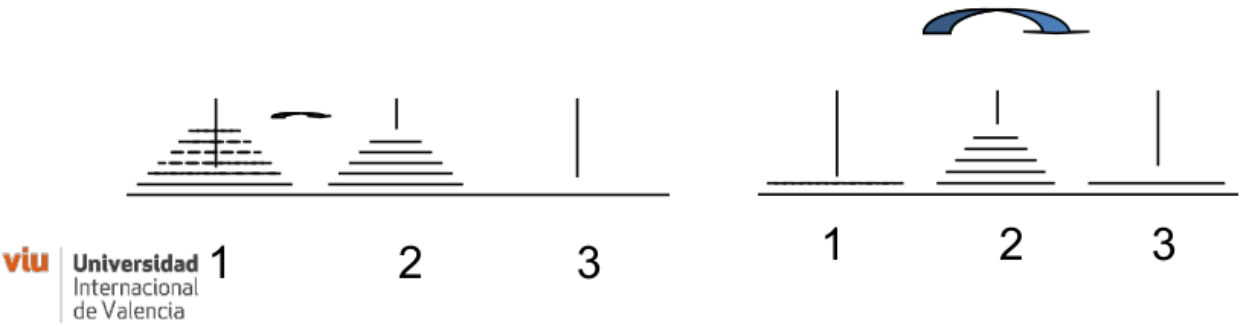
Haz doble clic (o pulsa Intro) para editar

Torres de Hanoi - Divide y venceras



Resolver(Total_fichas=4, Desde=1 , Hasta=3) es valido con:

- Resolver(Total_fichas=3, Desde=1, Hasta=2)
- Mover(Desde=1, Hasta=3)
- Resolver(Total_fichas=3, Desde=2, Hasta=3)



viu Universidad Internacional de Valencia

```
#Torres de Hanoi - Divide y venceras
##############################################################################


#################################################################
def Torres_Hanoi(N, desde, hasta):
  #N - Nº de fichas
  #desde - torre inicial
  #hasta - torre fina
  if N==1 :
    print("Lleva la ficha desde " + str(desde) + " hasta " + str(hasta))

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

Torres_Hanoi(10, 1, 3)
#################################################################
```

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

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

Cambio de monedas - Técnica voraz

Haz doble clic (o pulsa Intro) para editar

```
#Cambio de monedas – Técnica voraz
#############################################################################
SISTEMA = [12, 5 ,2, 1  ]
#############################################################
def cambio_monedas(CANTIDAD,SISTEMA):
#....
  SOLUCION = [0]*len(SISTEMA)
  ValorAcumulado = 0
```

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

    if CANTIDAD == ValorAcumulado:
      return SOLUCION

  print("No es posible encontrar solucion")
cambio_monedas(15,SISTEMA)


#################################################################
```

```
[1, 0, 1, 1]
```

N Reinas - Vuelta Atrás(Backtracking)

```
#N Reinas - Vuelta Atrás()
#############################################################################

#Verifica que en la solución parcial no hay amenzas entre reinas
#################################################################
def es_prometedora(SOLUCION,etapa):
#################################################################
  #print(SOLUCION)
  #Si la solución tiene dos valores iguales no es valida => Dos reinas en la mism
  for i in range(etapa+1):
    #print("El valor " + str(SOLUCION[i]) + " está " +  str(SOLUCION.count(SOLUCI
    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

#Traduce la solución al tablero
#################################################################
def escribe_solucion(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="")



#Proceso principal de N-Reinas
#################################################################
def reinas(N, solucion=[],etapa=0):
#################################################################
### ....
  if len(solucion) == 0:         # [0,0,0...]
    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)
      else:
        reinas(N, solucion, etapa+1)
    else:
      None

  solucion[etapa] = 0

reinas(8,solucion=[],etapa=0)
```

```
[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]
```

```
escribe_solucion([1, 5, 8, 6, 3, 7, 2, 4])
```

```
X - - - - - - -
- - - - - - X -
- - - - X - - -
- - - - - - - X
- X - - - - - -
- - - X - - - -
- - - - - X - -
- - X - - - - -
```

```
import random
```

```
# Generar listas de puntos aleatorios
LISTA_1D = [random.randrange(1, 10000) for _ in range(1000)]
LISTA_2D = [(random.randrange(1, 10000), random.randrange(1, 10000)) for _ in ran


# Función para encontrar los dos puntos más cercanos en 1D utilizando fuerza brut
def closest_points_1D_brute_force(points):
    min_distance = float('inf')
    closest_pair = None
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            distance = abs(points[i] - points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])
    return closest_pair, min_distance


# Función para encontrar los dos puntos más cercanos en 2D utilizando fuerza brut
def distance_2D(point1, point2):
    return ((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)**0.5

def closest_points_2D_brute_force(points):
    min_distance = float('inf')
    closest_pair = None
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            distance = distance_2D(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])
    return closest_pair, min_distance


# Ejecutar métodos en las listas generadas
result_1D = closest_points_1D_brute_force(LISTA_1D)
result_2D = closest_points_2D_brute_force(LISTA_2D)


# Mostrar resultados
print("Resultado en 1D (Fuerza Bruta):")
print(f"Puntos más cercanos: {result_1D[0]}, Distancia: {result_1D[1]}")

print("\nResultado en 2D (Fuerza Bruta):")
print(f"Puntos más cercanos: {result_2D[0]}, Distancia: {result_2D[1]}")
```

```
Resultado en 1D (Fuerza Bruta):
Puntos más cercanos: (5324, 5324), Distancia: 0

Resultado en 2D (Fuerza Bruta):
Puntos más cercanos: ((9932, 2658), (9934, 2664)), Distancia: 6.3245553203367
```

```
# Función Divide y Vencerás para 1D
def closest_points_1D_divide_and_conquer(points):
    if len(points) <= 1:
        return None, float('inf')
    if len(points) == 2:
        return (points[0], points[1]), abs(points[0] - points[1])

    points.sort()  # Ordenar los puntos
    mid = len(points) // 2
    left_points = points[:mid]
    right_points = points[mid:]

    left_closest, left_min_distance = closest_points_1D_divide_and_conquer(left_p
    right_closest, right_min_distance = closest_points_1D_divide_and_conquer(righ

    min_distance = min(left_min_distance, right_min_distance)
    closest_pair = left_closest if left_min_distance < right_min_distance else ri

    # Verificar puntos cercanos en los bordes
    border_points = [p for p in points if abs(p - points[mid]) < min_distance]
    for i in range(len(border_points)):
        for j in range(i + 1, len(border_points)):
            distance = abs(border_points[i] - border_points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (border_points[i], border_points[j])

    return closest_pair, min_distance


# Función Divide y Vencerás para 2D
def distance_2D(point1, point2):
    return ((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)**0.5

def closest_points_2D_divide_and_conquer(points):
    if len(points) <= 1:
        return None, float('inf')
    if len(points) == 2:
        return (points[0], points[1]), distance_2D(points[0], points[1])

    points.sort(key=lambda point: point[0])  # Ordenar los puntos por coordenada
    mid = len(points) // 2
    left_points = points[:mid]
    right_points = points[mid:]

    left_closest, left_min_distance = closest_points_2D_divide_and_conquer(left_p
    right_closest, right_min_distance = closest_points_2D_divide_and_conquer(righ

    min_distance = min(left_min_distance, right_min_distance)
    closest_pair = left_closest if left_min_distance < right_min_distance else ri

    # Buscar puntos cercanos al borde
    mid_x = points[mid][0]
    border_points = [point for point in points if abs(point[0] - mid_x) < min_dis
    border_points.sort(key=lambda point: point[1])  # Ordenar por coordenada y
```

```
    for i in range(len(border_points)):
        for j in range(i + 1, min(i + 7, len(border_points))):  # Comparar con lo
            distance = distance_2D(border_points[i], border_points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (border_points[i], border_points[j])

    return closest_pair, min_distance


# Ejecutar Divide y Vencerás en las listas generadas
result_1D_divide_and_conquer = closest_points_1D_divide_and_conquer(LISTA_1D)
result_2D_divide_and_conquer = closest_points_2D_divide_and_conquer(LISTA_2D)


# Mostrar resultados
print("Resultado en 1D (Divide y Vencerás):")
print(f"Puntos más cercanos: {result_1D_divide_and_conquer[0]}, Distancia: {result_

print("\nResultado en 2D (Divide y Vencerás):")
print(f"Puntos más cercanos: {result_2D_divide_and_conquer[0]}, Distancia: {result_
```

```
Resultado en 1D (Divide y Vencerás):
Puntos más cercanos: (9621, 9621), Distancia: 0

Resultado en 2D (Divide y Vencerás):
Puntos más cercanos: ((9932, 2658), (9934, 2664)), Distancia: 6.3245553203367
```