Soluciones a los problemas: Clase 8 - Algoritmos voraces (greedy) 2

Problemas: Clase 8 - Algoritmos voraces (greedy) 2

Máximo de Dos Enteros

```
# Leer dos numeros enteros separados por espacio
a, b = [int(x) for x in input().split()]

# Imprimir el mayor de los dos numeros
print(max(a, b))
```

Palíndromo 9

```
# Leer la cadena de caracteres
s = input()

# Crear la cadena invertida
s1 = s[::-1]

# Comparar la cadena original con la invertida
if s == s1:
    print("Si")   # Es palindromo
else:
    print("No")   # No es palindromo
```

Prefijo común

```
# Leer la primera palabra
a = input()
# Leer la segunda palabra
b = input()

sol = ""  # Variable para guardar el prefijo comun mas largo

# Recorrer hasta la longitud minima de las dos palabras
for i in range(min(len(a), len(b))):
    if a[i] == b[i]:
        sol += a[i]  # Agregar letra si es igual en ambas palabras
    else:
        break  # Parar si las letras no coinciden
```

```
# Imprimir el prefijo comun mas largo encontrado
print(sol)
```

Mínimo número de saltos para llegar al final

```
# Leer la lista de enteros que representan max pasos desde cada posicion
li = list(map(int, input().split()))

pos = 0        # posicion actual
sol = 1        # contador de saltos
n = len(li)    # longitud de la lista

# Si solo hay un elemento, no se necesita salto
if n == 1:
    print(0)
else:
    while pos < n:
        # Si en la posicion actual no se puede avanzar, es imposible llegar al
final
        if li[pos] == 0:
            print(-1)
            break

        # Si desde la posicion actual se puede llegar o pasar el ultimo indice,
imprimir saltos y terminar
        if pos + li[pos] >= n - 1:
            print(sol)
            break

        p = -1   # siguiente posicion para saltar
        may = -1 # maxima distancia alcanzable desde las posiciones posibles

        # Buscar el siguiente salto que permite avanzar mas lejos
        for i in range(pos + 1, min(n, pos + li[pos] + 1)):
            if may < i + li[i]:
                may = i + li[i]
                p = i

        pos = p    # actualizar posicion al siguiente salto elegido
        sol += 1   # incrementar contador de saltos
```

Conectar cuerdas con costo mínimo

```
import heapq

heap = []

n = int(input())                    # Leer numero de cuerdas
li = list(map(int, input().split()))  # Leer longitudes de cuerdas
```

```
# Insertar todas las cuerdas en un heap minimo
for e in li:
    heapq.heappush(heap, e)

sol = 0  # costo total acumulado

# Mientras haya mas de una cuerda en el heap
while len(heap) > 1:
    a = heapq.heappop(heap)  # extraer cuerda mas corta
    b = heapq.heappop(heap)  # extraer segunda cuerda mas corta
    sol += a + b             # sumar costo de unirlas
    heapq.heappush(heap, a + b)  # insertar la cuerda resultante

print(sol)  # imprimir costo minimo total
```

```
# Esta solucion es menos eficiente porque ordena la lista en cada iteracion,
# lo que aumenta el tiempo de ejecucion para listas grandes.

n = int(input())                    # Leer numero de cuerdas
li = list(map(int, input().split()))  # Leer longitudes de cuerdas

sol = 0  # costo total acumulado

while len(li) > 1:
    li.sort()                    # ordenar la lista en cada iteracion
    a = li[0]                   # cuerda mas corta
    b = li[1]                   # segunda cuerda mas corta
    sol += a + b                # sumar costo de unirlas
    del li[0]                   # eliminar primera cuerda
    del li[0]                   # eliminar segunda cuerda
    li.append(a + b)            # agregar cuerda resultante

print(sol)  # imprimir costo minimo total
```

Selección de Actividades

```
def seleccionActividades(inicios, finales):
    # Creo una lista de tuplas (inicio, fin) para manejar las actividades
    actividades = list(zip(inicios, finales))
    # Ordeno las actividades por su tiempo de finalizacion (de menor a mayor)
    actividades.sort(key=lambda x: x[1])

    # Siempre selecciono la primera actividad que termina
    cantidad = 1
    ultimo_fin = actividades[0][1]

    # Recorro las actividades restantes para seleccionar las que no se solapen
    for i in range(1, len(actividades)):
```

```
        # Si el inicio de la actividad actual es mayor o igual al fin de la ultima
seleccionada,
        # puedo tomar esta actividad
        if actividades[i][0] >= ultimo_fin:
            cantidad += 1
            ultimo_fin = actividades[i][1]  # Actualizo el fin de la ultima
actividad elegida

    return cantidad  # Devuelvo el maximo numero de actividades seleccionadas

# Leo la cantidad de actividades
n = int(input())
# Leo los tiempos de inicio
inicios = list(map(int, input().split()))
# Leo los tiempos de finalizacion
finales = list(map(int, input().split()))

# Imprimo el resultado de la funcion
print(seleccionActividades(inicios, finales))
```

Fusionar intervalos superpuestos

```
def solve(intervalos):
    if not intervalos:
        return 0

    # Ordeno los intervalos por el inicio
    intervalos.sort()

    resultado = [intervalos[0]]

    for inicio, fin in intervalos[1:]:
        ultimo_inicio, ultimo_fin = resultado[-1]
        # Si el inicio del intervalo actual es menor o igual al fin del ultimo
intervalo en resultado
        # significa que se solapan o tocan y debemos fusionarlos
        if inicio <= ultimo_fin:
            resultado[-1][1] = max(ultimo_fin, fin)  # Actualizo el fin del
intervalo fusionado
        else:
            resultado.append([inicio, fin])  # No se solapan, agrego el intervalo
nuevo

    return len(resultado)  # Devuelvo la cantidad de intervalos fusionados


n = int(input())
intervalos = []
for _ in range(n):
    inicio, fin = map(int, input().split())
    intervalos.append([inicio, fin])
```

```
print(solve(intervalos))
```