AG2 - Actividad Guiada 2

Nombre: Carlos Javier Bravo Intriago

https://colab.research.google.com/drive/1Nuj5Gf1vmRnA6Lg4XHXRs-Fx62q9rHlq?usp=sharing

https://github.com/carlosbravo1408/03MIAR-Algoritmos-de-Optimizacion-2025/tree/main/AG2

In [1]:
```
# paquete personalizado, creado exclusivamente para una mejor representación visual a los resultados de este Notebo
!pip install git+https://github.com/carlosbravo1408/03MIAR-Algoritmos-de-Optimizacion-2025.git
```

```
Collecting git+https://github.com/carlosbravo1408/03MIAR-Algoritmos-de-Optimizacion-2025.git
  Cloning https://github.com/carlosbravo1408/03MIAR-Algoritmos-de-Optimizacion-2025.git to /tmp/pip-req-build-bfvowk
54
  Running command git clone --filter=blob:none --quiet https://github.com/carlosbravo1408/03MIAR-Algoritmos-de-Optim
izacion-2025.git /tmp/pip-req-build-bfvowk54
  Resolved https://github.com/carlosbravo1408/03MIAR-Algoritmos-de-Optimizacion-2025.git to commit 8864ef251e1620866
dd6a7441bd8bb1b10cc1dc2
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
[notice] A new release of pip is available: 24.3.1 -> 26.0
[notice] To update, run: pip install --upgrade pip
```

In [2]:
```
import math
import time
import copy
import random
import decimal
import tracemalloc
from functools import wraps
from collections import defaultdict
from typing import List, Union, Tuple, Dict, Optional, Set, Callable

from MIAR03.beauty_printer import print_matrix, ConsoleColors

inf = float('inf')
NumericType = Union[float, int, decimal.Decimal]
FaresType = List[List[NumericType]]
PricesType = List[List[NumericType]]
PathsType = List[List[NumericType]]
TaskSolutionType = Tuple[NumericType, ...]
TaskCostMatrixType = List[List[NumericType]]


# Decorador medidor de tiempo
def measure_time(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.monotonic()
        result = func(*args, **kwargs)
        end_time = time.monotonic()
        elapsed_time = end_time - start_time
        if isinstance(result, tuple) \
                and len(result) >= 2  \
                and isinstance(result[-1], dict)  \
                and ("time" in result[-1] or "memory" in result[-1]):
            result[-1]["time"] = elapsed_time
            return result
        else:
            return result, {"time": elapsed_time}
    return wrapper


# Decorador medidor de Memoria
def measure_memory(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        tracemalloc.start()
        result = func(*args, **kwargs)
        _, peak = tracemalloc.get_traced_memory()
        tracemalloc.stop()
        # 1 << 10 = 1024, 1 << (10*2) = 1024**2  Conversion a MB
        peak = peak / (1 << (10 * 2))
        if isinstance(result, tuple) \
                and len(result) >= 2  \
                and isinstance(result[-1], dict)  \
                and ("time" in result[-1] or "memory" in result[-1]):
            result[-1]["memory"] = peak
            return result
        else:
            return result, {"memory": peak}
```

```
    return wrapper

# método para generar las matrices de tarea/agente aleatorias
def generate_random_task_list(
        n: int,
        start: int = 1,
        stop: int = 100,
        seed: Optional[int] = None
) -> List[List[int]]:
    if seed is not None:
        random.seed(seed)
    return [[random.randrange(start, stop) for _ in range(n)] for _ in range(n)]
```

1. Programación Dinámica. Viaje por el río

- **Definición**: Es posible dividir el problema en subproblemas más pequeños, guardando las soluciones para ser utilizadas más adelante.
- **Características** que permiten identificar problemas aplicables:
  -Es posible almacenar soluciones de los subproblemas para ser utilizados más adelante
  -Debe verificar el principio de optimalidad de Bellman: “en una secuencia óptima de decisiones, toda sub-secuencia también es óptima” (*)
  -La necesidad de guardar la información acerca de las soluciones parciales unido a la recursividad provoca la necesidad de preocuparnos por la complejidad espacial (cuantos recursos de espacio usaremos)

A. Problema

En un río hay `n` embarcaderos y debemos desplazarnos río abajo desde un embarcadero a otro. Cada embarcadero tiene precios diferentes para ir de un embarcadero a otro situado más abajo. Para ir del embarcadero i al j, puede ocurrir que sea más barato hacer un trasbordo por un embarcadero intermedio k. El problema consiste en determinar la combinación más barata.

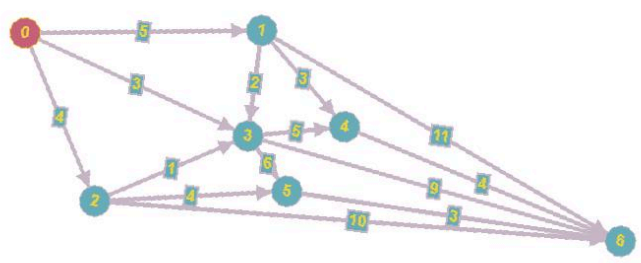


Fig 1. Problema del viaje por el río.

- Consideramos una tabla TARIFAS(i,j) para almacenar todos los precios que nos ofrecen los embarcaderos.
- Si no es posible ir desde i a j daremos un valor alto para garantizar que ese trayecto no se va a elegir en la ruta óptima(modelado habitual para restricciones)

In [3]:
```
#Viaje por el río - Programación dinámica

TARIFAS = [
    #   0    1    2    3    4    5    6
    [   0,   5,   4,   3, inf, inf, inf],  # 0
    [ inf,   0, inf,   2,   3, inf,  11],  # 1
    [ inf, inf,   0,   1, inf,   4,  10],  # 2
    [ inf, inf, inf,   0,   5,   6,   9],  # 3
    [ inf, inf, inf, inf,   0, inf,   4],  # 4
    [ inf, inf, inf, inf, inf,   0,   3],  # 5
    [ inf, inf, inf, inf, inf, inf,   0]   # 6
]

print_matrix(TARIFAS, title="Tarifas")
```

```
                  Tarifas
-----------------------------------------------
         0     1     2     3     4     5     6
   ----------------------------------------------
 0 |     0     5     4     3     ∞     ∞     ∞
 1 |     ∞     0     ∞     2     3     ∞    11
 2 |     ∞     ∞     0     1     ∞     4    10
 3 |     ∞     ∞     ∞     0     5     6     9
 4 |     ∞     ∞     ∞     ∞     0     ∞     4
 5 |     ∞     ∞     ∞     ∞     ∞     0     3
 6 |     ∞     ∞     ∞     ∞     ∞     ∞     0
```

In [4]:
```
@measure_time
@measure_memory
def precios(tarifas: FaresType) -> Tuple[PricesType, PathsType]:
    """
    - Autor: Mónica Sánchez Cuberes (Con ayuda de Claude de Anthropic). Corrección sugerida en el Foro
    - Modificado por: Carlos Javier Bravo Intriago
    - Fecha: Enero 2026
```

```
    ----------------------

    Cálculo de la matriz de PRECIOS y RUTAS\n
    Precios - contiene la matriz del mejor precio para ir de un nodo a otro\n
    Rutas - contiene los nodos intermedios para ir de un nodo a otro

    Este código se basa en el algoritmo de Floyd-Warshall, el cual fue
    sugerencia propuesta por Mónica Sánchez Cuberes como parte de una
    corrección sugerida en el Foro. Las salidas de las tuplas obedecen a la
    forma canónica de D (Matriz Distancias o pesos), y Π (Matriz de
    predecesores)

    Complejidad temporal O(N³)

    Cambios:
    ----------------------
    * [1] Se agregaron validaciones para evitar recorrer el bucle si no existe
      conexión
    * [2] Se omite que se rellene con valores de `i` por debajo de la diagonal
      principal si no existe conexión alguna con otro nodo


    ----------------
    :param tarifas: Matriz de tarifas (costos) entre nodos
    :return: Tupla con los arreglos de Precios y Rutas de forma canónica
    """
    n = len(tarifas)

    _precios = [tarifa[:] for tarifa in tarifas]
    rutas = [[i if i != j and tarifas[i][j] != inf else -1 for j in range(n)]
             for i in range(n)]  # [2]
    for k in range(n):
        for i in range(n):
            if _precios[i][k] == inf:  # [1]
                continue
            for j in range(n):
                if _precios[k][j] == inf:  # [1]
                    continue
                if _precios[i][k] + _precios[k][j] < _precios[i][j]:
                    _precios[i][j] = _precios[i][k] + _precios[k][j]
                    rutas[i][j] = rutas[k][j]  # Resultado en forma Canónica
    return _precios, rutas
```

In [5]:
```
(PRECIOS, RUTA), estadisticas = precios(TARIFAS)
print(estadisticas)
```

```
{'memory': 0.00140380859375, 'time': 0.00018922800018117414}
```

In [6]:
```
print_matrix(PRECIOS, title="Precios")
```

```
                  Precios
-----------------------------------------------
         0     1     2     3     4     5     6
    -----------------------------------------
 0 |     0     5     4     3     8     8    11
 1 |     ∞     0     ∞     2     3     8     7
 2 |     ∞     ∞     0     1     6     4     7
 3 |     ∞     ∞     ∞     0     5     6     9
 4 |     ∞     ∞     ∞     ∞     0     ∞     4
 5 |     ∞     ∞     ∞     ∞     ∞     0     3
 6 |     ∞     ∞     ∞     ∞     ∞     ∞     0
```

In [7]:
```
print_matrix(RUTA, title="Ruta")
```

```
                   Ruta
-----------------------------------------------
         0     1     2     3     4     5     6
    -----------------------------------------
 0 |    -1     0     0     0     1     2     5
 1 |    -1    -1    -1     1     1     3     4
 2 |    -1    -1    -1     2     3     2     5
 3 |    -1    -1    -1    -1     3     3     3
 4 |    -1    -1    -1    -1    -1    -1     4
 5 |    -1    -1    -1    -1    -1    -1     5
 6 |    -1    -1    -1    -1    -1    -1    -1
```

In [8]:
```
def calcular_ruta(ruta: PathsType, desde: int, hasta: int) -> List[int]:
    """
    Algoritmo recursivo que retorna la ruta de menor coste a partir del
    resultado de rutas bajo la forma canónica de Π (Matriz de predecesores)

    :param ruta: Matriz con las rutas de la forma canónica de Π (Matriz de
           predecesores)
    :param desde: Nodo origen
    :param hasta: Nodo destino
    :return: Ruta más corta desde el origen hasta el destino.
```

```
    """
    if desde == hasta:
        return [desde]
    predecesor = ruta[desde][hasta]
    if predecesor == -1:
        return []
    camino_previo = calcular_ruta(ruta, desde, predecesor)
    if not camino_previo:
        return []
    return camino_previo + [hasta]
```

In [9]:
```
print("La ruta es:")
print(calcular_ruta(RUTA, 0, 6))
```

```
La ruta es:
[0, 2, 5, 6]
```

In [10]:
```
TARIFAS_MONICA = [
    #   0    1    2    3    4
    [   0, inf,   2, inf, inf],  # 0
    [   1,   0, inf, inf, inf],  # 1
    [ inf, inf,   0, inf,  10],  # 2
    [ inf,   1, inf,   0, inf],  # 3
    [ inf, inf, inf,   5,   0]   # 4
]
print_matrix(TARIFAS_MONICA, title="Tarifas, Ejemplo de Monica Sánchez")

(PRECIOS_MONICA, RUTA_MONICA), statics = precios(TARIFAS_MONICA)
print(statics)
print_matrix(PRECIOS_MONICA, title="Precios, Ejemplo de Monica Sánchez")
print_matrix(RUTA_MONICA, title="Ruta, Ejemplo de Monica Sánchez")

print(f"La ruta es:{calcular_ruta(RUTA_MONICA, 1, 4)}")
```

```
Tarifas, Ejemplo de Monica Sánchez
-----------------------------------
          0     1     2     3     4
    --------------------------------
  0 |     0     ∞     2     ∞     ∞
  1 |     1     0     ∞     ∞     ∞
  2 |     ∞     ∞     0     ∞    10
  3 |     ∞     1     ∞     0     ∞
  4 |     ∞     ∞     ∞     5     0

{'memory': 0.001068115234375, 'time': 8.883600003173342e-05}
Precios, Ejemplo de Monica Sánchez
-----------------------------------
          0     1     2     3     4
    --------------------------------
  0 |     0    18     2    17    12
  1 |     1     0     3    18    13
  2 |    17    16     0    15    10
  3 |     2     1     4     0    14
  4 |     7     6     9     5     0

  Ruta, Ejemplo de Monica Sánchez
-----------------------------------
          0     1     2     3     4
    --------------------------------
  0 |    -1     3     0     4     2
  1 |     1    -1     0     4     2
  2 |     1     3    -1     4     2
  3 |     1     3     0    -1     2
  4 |     1     3     0     4    -1

La ruta es:[1, 0, 2, 4]
```

B. Resolviendo el problema del río con el algoritmo de Dijkstra

In [11]:
```python
Node = int
Weight = NumericType
Graph = Dict[Node, List[Tuple[Node, Weight]]]


def create_graph(fares: FaresType) -> Graph:
    """
    Método que convierte una matriz de Adyacencia en una estructura tipo grafo.
    :param fares: Matriz de Adyacencia
    :return: Grafo
    """
    graph = defaultdict(list)
    for i in range(len(fares)):
        for j in range(len(fares[i])):
            if math.isinf(fares[i][j]):
                continue
            graph[i].append((j, fares[i][j]))
    return graph
```

In [12]:
```python
graph = create_graph(fares=TARIFAS)
graph
```

Out[12]:
```
defaultdict(list,
            {0: [(0, 0), (1, 5), (2, 4), (3, 3)],
             1: [(1, 0), (3, 2), (4, 3), (6, 11)],
             2: [(2, 0), (3, 1), (5, 4), (6, 10)],
             3: [(3, 0), (4, 5), (5, 6), (6, 9)],
             4: [(4, 0), (6, 4)],
             5: [(5, 0), (6, 3)],
             6: [(6, 0)]})
```

In [13]:
```python
from heapq import heappush, heappop  # Cola de prioridad

# Observación: Este algoritmo se puede optimizar un poco más evitando crear
# la lista de rutas, lo ideal sería hacer referencia qué nodo es predecesor
# de otro, y de manera recursiva encontrar la ruta mínima.
@measure_time
@measure_memory
def dijkstra(
        graph: Graph, source: Node
) -> Tuple[Dict[Node, List[Node]], Dict[Node, Weight]]:
    """
    Algoritmo de Dijkstra modificado para resolver el problema del río, es
    una propuesta basada en SSSP (Single-Source Shortest Path) en el supuesto
    de que siempre se parta del mismo origen (en este caso el nodo 0), este
    Algoritmo retornará todas las rutas más cortas desde el origen antes definido.

    Complejidad temporal: O(|E| + |V|log|V|)

    :param graph: Grafo
    :param source: Nodo Origen
    :return: Colección de rutas y de distancias mínimas.
    """
    dist: Dict[Node, Weight] = {v: inf for v in graph}
    dist[source] = 0.0
    paths: Dict[Node, List[Node]] = {source: [source]}
    heap: List[Tuple[Weight, Node]] = [(0.0, source)]
    while heap:
        current_weight, current_node = heappop(heap)
        if current_weight > dist[current_node]:
            continue
        if current_node not in paths:
            paths[current_node] = []
        for node, weight in graph[current_node]:  # Visitando vecinos / hijos
            sum_weights = current_weight + weight
            if sum_weights < dist.get(node, inf):
                paths[node] = paths[current_node] + [node]
                dist[node] = sum_weights
                heappush(heap, (sum_weights, node))

    return paths, dist
```

In [14]:
```python
(paths, dist), stats = dijkstra(graph, 0)
print(stats)
print("\nRutas mas cortas desde el Nodo 0")
print(paths)
print("\nCostos minimizados desde el Nodo 0")
print(dist)
print("\nLa ruta desde 0 hasta 6 es:")
print(paths[6])
```

```
{'memory': 0.0014495849609375, 'time': 7.181899991337559e-05}

Rutas mas cortas desde el Nodo 0
{0: [0], 1: [0, 1], 2: [0, 2], 3: [0, 3], 4: [0, 3, 4], 5: [0, 2, 5], 6: [0, 2, 5, 6]}

Costos minimizados desde el Nodo 0
{0: 0.0, 1: 5.0, 2: 4.0, 3: 3.0, 4: 8.0, 5: 8.0, 6: 11.0}

La ruta desde 0 hasta 6 es:
[0, 2, 5, 6]
```

In [15]:
```python
graph_monica = create_graph(fares=TARIFAS_MONICA)
(paths_monica, dist_monica), stats = dijkstra(graph_monica, 1)
print(stats)
print("\nRutas mas cortas desde el Nodo 0")
print(paths_monica)
print("\nCostos minimizados desde el Nodo 0")
print(dist_monica)
print("\nLa ruta desde 1 hasta 4 es:")
print(paths_monica[4])
```

```
{'memory': 0.00023651123046875, 'time': 3.71399996765831e-05}

Rutas mas cortas desde el Nodo 0
{1: [1], 0: [1, 0], 2: [1, 0, 2], 4: [1, 0, 2, 4], 3: [1, 0, 2, 4, 3]}

Costos minimizados desde el Nodo 0
{0: 1.0, 1: 0.0, 2: 3.0, 3: 18.0, 4: 13.0}

La ruta desde 1 hasta 4 es:
[1, 0, 2, 4]
```

2. Problema de Asignación de tarea - Ramificación y Poda

El objetivo es asignar `n` tareas a `n` agentes de forma que el costo total sea el mínimo posible, bajo la restricción de que cada agente realiza exactamente una tarea y cada tarea es realizada por un solo agente.

	T	A	R	E	A
A					
G					
E					
N					
T					
E					

In [16]:
```python
COSTES = [
    #T0, T1, T2, T3
    [11, 12, 18, 40],  # A0
    [14, 15, 13, 22],  # A1
    [11, 17, 19, 23],  # A2
    [17, 14, 20, 28]   # A3
]

print_matrix(COSTES, header_col_char="T", header_row_char="A")
```

In [17]:
```python
def valor(
        solucion_parcial: TaskSolutionType,
        matriz_de_costes: TaskCostMatrixType
) -> NumericType:
    """
    Función que calcula el valor de una unica `solucion_parcial`

    :param solucion_parcial:
    :param matriz_de_costes:
    :return: Suma de costes de esa solución parcial.
    """
    acumulador = 0
    for i in range(len(solucion_parcial)):
        acumulador += matriz_de_costes[solucion_parcial[i]][i]
    return acumulador


valor((3, 2,), COSTES)
```

Out[17]: 34

In [18]:
```python
def coste_inferior(
        solucion_parcial:TaskSolutionType,
        matriz_de_costes:TaskCostMatrixType
) -> NumericType:
    """
    Coste inferior para soluciones parciales\n
```

```
    (1,3,) Se asigna la tarea 1 al agente 0 y la tarea 3 al agente 1
    :param solucion_parcial:
    :param matriz_de_costes:
    :return: coste inferior
    """
    acumulador = 0
    #Valores establecidos
    for i in range(len(solucion_parcial)):
        acumulador += matriz_de_costes[i][solucion_parcial[i]]

    #Estimacion
    for i in range(len(solucion_parcial), len(matriz_de_costes)):
        acumulador += min([matriz_de_costes[j][i] for j in range(len(solucion_parcial), len(matriz_de_costes))])
    return acumulador


def coste_superior(
        solucion_parcial: TaskSolutionType,
        matriz_de_costes: TaskCostMatrixType
) -> NumericType:
    """
    Coste superior para soluciones parciales\n
    (1,3,) Se asigna la tarea 1 al agente 0 y la tarea 3 al agente 1
    :param solucion_parcial:
    :param matriz_de_costes:
    :return: coste superior
    """
    acumulador = 0
    #Valores establecidos
    for i in range(len(solucion_parcial)):
        acumulador += matriz_de_costes[i][solucion_parcial[i]]

    #Estimacion
    for i in range(len(solucion_parcial), len(matriz_de_costes)):
        acumulador += max([matriz_de_costes[j][i] for j in range(len(solucion_parcial), len(matriz_de_costes))])
    return acumulador


print(coste_inferior((0, 1), COSTES))
print(coste_superior((0, 1), COSTES))
```

```
68
74
```

In [19]:

```
def crear_hijos(
        nodo: TaskSolutionType,
        dimension: int
) -> List[Dict[str, TaskSolutionType]]:
    """
    Método que genera nuevas ramificaciones, nuevos nodos hijos que sean
    posibles como siguiente elemento de la tupla de solución parcial\n
    >>> crear_hijos(nodo=(0,), dimension=4)
    [{'s': (0, 1)}, {'s': (0, 2)}, {'s': (0, 3)}]
    :param nodo:
    :param dimension:
    :return: Lista de nodos hijos posibles.
    """
    hijos = []
    for i in range(dimension):
        if i not in nodo:
            hijos.append({'s': nodo + (i,)})
    return hijos

crear_hijos((0,), 4)
```

Out[19]:

```
[{'s': (0, 1)}, {'s': (0, 2)}, {'s': (0, 3)}]
```

In [20]:

```
@measure_time
@measure_memory
def ramificacion_y_poda(
        matriz_de_costes: TaskCostMatrixType
) -> Tuple[TaskSolutionType, int, int]:
    """
    - Autor: Raúl Reyero
    ----------------------

    Algoritmo que busca asignar tareas a los respectivos agentes de tal forma
    que el costo entre estas sea el minimo.\n
    Construcción iterativa de soluciones(arbol). En cada etapa asignamos un
    agente(ramas).\n
    Nodos del grafo { s: (1, 2), CI: 3, CS: 5 }

    :param matriz_de_costes:
    :return: La lista de Tareas por agente.
    """

    dimension = len(matriz_de_costes)
```

```
    mejor_solucion = tuple(i for i in range(len(matriz_de_costes)))
    cota_sup = valor(mejor_solucion, matriz_de_costes)

    nodos = [{'s': (), 'ci': coste_inferior((), matriz_de_costes)}]

    iteracion = 0

    while len(nodos) > 0:
        iteracion += 1

        nodo_prometedor = [min(nodos, key=lambda x: x['ci'])][0]['s']

        # Ramificación: Se generan los hijos
        hijos = [
            {'s': x['s'], 'ci': coste_inferior(x['s'], matriz_de_costes)}
            for x in crear_hijos(nodo_prometedor, dimension)
        ]

        # Revisamos la cota superior y nos quedamos con la mejor solucion si
        # llegamos a una solucion final
        nodo_final = [x for x in hijos if len(x['s']) == dimension]
        if len(nodo_final) > 0:
            if nodo_final[0]['ci'] < cota_sup:
                cota_sup = nodo_final[0]['ci']
                mejor_solucion = nodo_final

        # Poda
        hijos = [x for x in hijos if x['ci'] < cota_sup]

        # Añadimos los hijos
        nodos.extend(hijos)

        # Eliminamos el nodo ramificado
        nodos = [x for x in nodos if x['s'] != nodo_prometedor]

    return mejor_solucion[0]['s'], mejor_solucion[0]['ci'], iteracion
```

In [21]:

```
solution, estadisticas = ramificacion_y_poda(COSTES)
print_matrix(
    COSTES,
    solution[0],
    title="Matriz de Costes",
    header_col_char="T",
    header_row_char="A"
)

print(f"Tiempo de ejecución: {estadisticas['time']}, Memoria usada: {estadisticas['memory']}")
print(f"Solución1: {solution[0]}, con un coste de {solution[1]} en {solution[2]} iteraciones")
```

```
     Matriz de Costes
Leyenda:  Sol 1
----------------------------
        T0    T1    T2    T3
    ------------------------
 A0 |    11    12    18    40
 A1 |    14    15    13    22
 A2 |    11    17    19    23
 A3 |    17    14    20    28
Tiempo de ejecución: 0.0006506509998871479, Memoria usada: 0.0021514892578125
Solución1: (1, 2, 0, 3), con un coste de 64 en 10 iteraciones
```

A. Implementación de estrategia de ramificación y poda más cola de prioridad:

En esta version, se ha decidido experimentar con un híbrido entre las estrategias Ramificación y Poda (Branch and Bound) y Búsqueda Primero el Mejor (Best-First Search) implementando una cola de prioridad (similar a Dijkstra Modificado). El objetivo es seleccionar en cada iteración el nodo prometedor con la menor Cota Inferior (`lower_bound`) estimada, guiando la exploración hacia la solución óptima de manera que el algoritmo explora primero las ramas que teóricamente ofrecen el menor costo total.

Para facilitar la ramificación, se ha creado una estructura llamada `Node` , la cual tiene por atributos:

- `agent` : Agente al que pertenece
- `task` : Tarea asignada
- `cost` : costo de esa tarea para el agente asignado
- `parent` : nodo predecesor
- `lower_cost` : cota inferior asociada a esa tarea y agente
- `satisfied_task` : tareas satisfechas hasta el agente perteneciente a ese Nodo

En las pruebas realizadas en el notebook AsignacionDeTareas se explora este algoritmo tanto si se obtiene el primer resultado que satisface al problema (como una parada temprana gracias a la cola de prioridad), o si se explora las soluciones posibles antes de retornar la solución con menor coste con respecto a una variante de *Fuerza bruta* para verificar si los resultados son deterministas y son iguales.

Dichas pruebas sugieren que el resultado entre elegir la primera solución que satisface al problema (parada temprana) es la misma que si se deja procesar todas las soluciones posibles restantes, lo que indica que el método que calcula la cota inferior `calculate_lower_cost` es admisible, y esto se debe a que suma los mínimos absolutos de las filas de los agentes restantes, filtrando las tareas ya tomadas por los agentes anteriores.

In [22]:
```python
# Clase auxiliar Nodo
class Node:
    def __init__(
            self,
            agent: int,
            task: Optional[int],
            cost: NumericType,
            parent: Optional['Node'] = None,
            lower_cost: int = 0
    ):
        self.agent: int = agent
        self.task: Optional[int] = task
        self.parent: 'Node' = parent
        self.cost: NumericType = cost
        self.accumulative_cost: NumericType = \
            cost + (0 if self.parent is None else self.parent.accumulative_cost)
        self.lower_bound: NumericType = self.accumulative_cost + lower_cost

    @property
    def satisfied_tasks(self) -> List[int]:
        tasks = []
        current = self
        while current:
            if current.task is not None:
                tasks.append(current.task)
            current = current.parent
        tasks.reverse()
        return tasks

    # sobrecarga de metodos para usarse en la cola de prioridad
    def __lt__(self, other: 'Node'):
        return self.lower_bound < other.lower_bound

    def __eq__(self, other: 'Node'):
        return self.lower_bound == other.lower_bound

    def __repr__(self):  # Para propósitos de depuración
        return f"satisfied:{self.satisfied_tasks}, \
        single_cost: {self.cost}, cost:{self.lower_bound}"
```

In [23]:
```python
def calculate_lower_cost(
        cost_matrix: TaskCostMatrixType,
        next_agent: int,
        taken_tasks: Set[int]
) -> int:
    """
    Calcula la suma de los mínimos de las filas restantes, ignorando las
    tareas ya tomadas.

    :param cost_matrix: Matriz de costes
    :param next_agent: Agente desde el que se desea verificar los mínimos
    :param taken_tasks: Tareas ya asignadas previamente
    :return:
    """
    min_future_cost = 0
    num_agents = len(cost_matrix)

    for agent_idx in range(next_agent, num_agents):
        row_min = float('inf')
        for task_idx, cost in enumerate(cost_matrix[agent_idx]):
            if task_idx not in taken_tasks and cost < row_min:
                row_min = cost
        if row_min != float('inf'):
            min_future_cost += row_min

    return min_future_cost
```

In [24]:
```python
@measure_time
@measure_memory
def task_assignment(
        cost_matrix: TaskCostMatrixType,
        get_first_searched: bool = True
) -> Tuple[List[NumericType], NumericType]:
    nodes = []
    num_agents = len(cost_matrix)
    tasks = {i for i in range(len(cost_matrix))}
    best_cost = float('inf')

    root = Node(
        agent=-1, task=None, cost=0,
```

```
        lower_cost=calculate_lower_cost(cost_matrix, 0, set())
    )
    solution = ([],)
    heappush(nodes, root)

    while nodes:
        # inspirado al algoritmo de Dijkstra, decola el de menor costo
        current_node = heappop(nodes)

        if current_node.lower_bound >= best_cost:
            continue
        satisfied_tasks = current_node.satisfied_tasks

        if current_node.agent == num_agents - 1:  # if len(current_satisfied_tasks) == num_agents:
            # si ya encuentra una primera solucion, puede retornarla,
            # aparentemente esta solucion es la de más bajo coste según las
            # pruebas realizadas en el Notebook AsignacionDeTareas.ipynb
            if get_first_searched:
                return satisfied_tasks, current_node.accumulative_cost
            # actualización de la cota superior
            if current_node.accumulative_cost < best_cost:
                best_cost = current_node.accumulative_cost
                solution = satisfied_tasks, current_node.accumulative_cost
            continue
        current_tasks = set(satisfied_tasks)
        next_agent = current_node.agent + 1
        available_tasks = tasks - current_tasks

        for task in available_tasks:
            current_cost = cost_matrix[next_agent][task]
            # asignando cota inferior
            current_accumulative_cost = current_node.accumulative_cost + current_cost
            # poda
            if current_accumulative_cost >= best_cost:
                continue
            remain_tasks = current_tasks.union({task})
            lower_cost = calculate_lower_cost(cost_matrix, next_agent + 1, remain_tasks)
            child_node = Node(
                agent=next_agent,
                task=task,
                cost=current_cost,
                parent=current_node,
                lower_cost=lower_cost
            )
            # ramificacion y poda
            if child_node.lower_bound < best_cost:
                heappush(nodes, child_node)

    return solution
```

In [25]:

```
prev_solution = copy.deepcopy(solution)
solution, estadisticas = task_assignment(COSTES)
print_matrix(
    COSTES,
    prev_solution[0],
    solution[0],
    title="Matriz de Costes",
    header_col_char="T",
    header_row_char="A"
)

print(f"Tiempo de ejecución: {estadisticas['time']}, Memoria usada: {estadisticas['memory']}")
print(f"Solución2: {solution[0]}, con un coste de {solution[1]}")
```

```
     Matriz de Costes
Leyenda: Sol 1  Sol 2  Ambos
-----------------------------
        T0    T1    T2    T3
    -------------------------
 A0 |    11    12    18    40
 A1 |    14    15    13    22
 A2 |    11    17    19    23
 A3 |    17    14    20    28

Tiempo de ejecución: 0.00028970299990760395, Memoria usada: 0.00626373291015625
Solución2: [0, 2, 3, 1], con un coste de 61
```

B. Pruebas de rendimiento con respecto al tamaño de la matriz

Se propone realizar una prueba con matrices aleatorias controladas por una semilla asignada (`seed`), que van desde dimensiones de 4x4 hasta 20x20. La finalidad es evaluar desde qué tamaño de la matriz ya resulta inviable usar alguna variante u otra.

El límite será si supera 120 segundos, una vez que se supere este tiempo, ya no se considerará ese algoritmo para la siguiente iteración.

Eventualmente, se puede concluir que con matrices de 12×12 en adelante se torna inviable ejecutar el algoritmo original. De 13×13 no llega a resolverse por más de 45 minutos.

Con la propuesta híbrida, se logra ejecutar matrices de hasta magnitud de 30×30 en tiempos mucho más cortos que el algoritmo original.

In [26]:
```
max_time = 120  # segundos
has_exceeded_limit_time1, has_exceeded_limit_time2 = False, False
seed = 47

for n in range(4, 30):
    if has_exceeded_limit_time1 and has_exceeded_limit_time2:
        break
    costs = generate_random_task_list(n, seed=seed)
    solution1, solution2 = [[], ], [[], ]
    stats1, stats2 = {}, {}
    elapsed_time1, elapsed_time2 = inf, inf
    memory_used1, memory_used2 = inf, inf

    if not has_exceeded_limit_time1:
        solution1, stats1 = task_assignment(costs)
        elapsed_time1, memory_used1 = stats1['time'], stats1['memory']
        if elapsed_time1 >= max_time or math.isclose(elapsed_time1, max_time):
            has_exceeded_limit_time1 = True
        print(f"{ConsoleColors.Cyan}Algoritmo Propuesto:\n"
              f"Solucion: {solution1[0]}\n"
              f" Costo: {solution1[1]}\n"
              f"Tiempo de ejecución: {elapsed_time1}s\n"
              f"Memoria usada: {memory_used1}Mb{ConsoleColors.Color_Off}")
        if has_exceeded_limit_time1:
            print(f"{ConsoleColors.Red}Correr matrices de {n}x{n} "
                  f"dimensiones es inviable con el algoritmo "
                  f"original. Tiempo transcurrido: {elapsed_time1}s{ConsoleColors.Color_Off}")

    if not has_exceeded_limit_time2:
        solution2, stats2 = ramificacion_y_poda(costs)
        elapsed_time2, memory_used2 = stats2['time'], stats2['memory']
        if elapsed_time2  >=  max_time or math.isclose(elapsed_time2, max_time):
            has_exceeded_limit_time2 = True
        print(f"{ConsoleColors.Magenta}Algoritmo Original:\n"
              f"Solución: {solution2[0]}\n"
              f"Costo: {solution2[1]}\n"
              f"Tiempo de ejecución: {elapsed_time2}s\n"
              f"Memoria usada: {memory_used2}Mb{ConsoleColors.Color_Off}")
        if has_exceeded_limit_time2:
            print(f"{ConsoleColors.Red}Correr matrices de {n}x{n} "
                  f"dimensiones es inviable con el algoritmo "
                  f"original. Tiempo transcurrido: {elapsed_time2}s{ConsoleColors.Color_Off}")
    print_matrix(
        costs,
        list(solution1[0]),
        list(solution2[0]),
        header_col_char="T",
        header_row_char="A",
        title=f"Matriz de {n}x{n}"
    )
    print("\n")
```

```
Algoritmo Propuesto:
Solucion: [1, 3, 2, 0]
 Costo: 99
Tiempo de ejecución: 0.00017252299994652276s
Memoria usada: 0.0033111572265625Mb
Algoritmo Original:
Solución: (1, 3, 2, 0)
Costo: 99
Tiempo de ejecución: 0.0003799520000029588s
Memoria usada: 0.001251220703125Mb
```

Matriz de 4x4

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3
A0	46	9	56	71
A1	59	74	44	33
A2	66	50	51	79
A3	6	54	85	4

```
Algoritmo Propuesto:
Solucion: [1, 4, 2, 0, 3]
 Costo: 110
Tiempo de ejecución: 0.00044877199979964644s
Memoria usada: 0.0080413818359375Mb
Algoritmo Original:
Solución: (1, 4, 2, 0, 3)
Costo: 110
Tiempo de ejecución: 0.0008507259999532835s
Memoria usada: 0.0010528564453125Mb
```

Matriz de 5x5

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4
A0	46	9	56	71	59
A1	74	44	33	66	50
A2	51	79	6	54	85
A3	4	14	1	34	62
A4	93	70	2	41	73

```
Algoritmo Propuesto:
Solucion: [1, 0, 5, 4, 2, 3]
 Costo: 91
Tiempo de ejecución: 0.00028097199992771493s
Memoria usada: 0.005508422851562 5Mb
Algoritmo Original:
Solución: (1, 0, 5, 4, 2, 3)
Costo: 91
Tiempo de ejecución: 0.019073792999733996s
Memoria usada: 0.061417579650878906Mb
```

Matriz de 6x6

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5
A0	46	9	56	71	59	74
A1	44	33	66	50	51	79
A2	6	54	85	4	14	1
A3	34	62	93	70	2	41
A4	73	55	30	31	67	66
A5	46	57	31	5	72	48

```
Algoritmo Propuesto:
Solucion: [1, 5, 3, 6, 4, 0, 2]
 Costo: 152
Tiempo de ejecución: 0.002571179999904416s
Memoria usada: 0.029834747314453125Mb
Algoritmo Original:
Solución: (1, 5, 3, 6, 4, 0, 2)
Costo: 152
Tiempo de ejecución: 0.015276400999937323s
Memoria usada: 0.03678607940673828Mb
```

Matriz de 7x7

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6
A0	46	9	56	71	59	74	44
A1	33	66	50	51	79	6	54
A2	85	4	14	1	34	62	93

A3	70	2	41	73	55	30	31
A4	67	66	46	57	31	5	72
A5	48	92	89	46	77	44	72
A6	74	98	26	31	94	5	17

Algoritmo Propuesto:
Solucion: [1, 4, 6, 3, 0, 7, 2, 5]
 Costo: 140
Tiempo de ejecución: 0.002986430999953882s
Memoria usada: 0.03571033477783203Mb
Algoritmo Original:
Solución: (1, 4, 6, 3, 0, 7, 2, 5)
Costo: 140
Tiempo de ejecución: 0.035262349999811704s
Memoria usada: 0.06287384033203125Mb

Matriz de 8x8

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7
A0	46	9	56	71	59	74	44	33
A1	66	50	51	79	6	54	85	4
A2	14	1	34	62	93	70	2	41
A3	73	55	30	31	67	66	46	57
A4	31	5	72	48	92	89	46	77
A5	44	72	74	98	26	31	94	5
A6	17	43	13	47	75	50	66	70
A7	33	13	84	30	90	43	28	61

Algoritmo Propuesto:
Solucion: [1, 6, 4, 0, 8, 5, 3, 2, 7]
 Costo: 119
Tiempo de ejecución: 0.0025676039999780187s
Memoria usada: 0.018810272216796875Mb
Algoritmo Original:
Solución: (1, 6, 4, 0, 8, 5, 3, 2, 7)
Costo: 119
Tiempo de ejecución: 0.08907242599980236s
Memoria usada: 0.12396049499511719Mb

Matriz de 9x9

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8
A0	46	9	56	71	59	74	44	33	66
A1	50	51	79	6	54	85	4	14	1
A2	34	62	93	70	2	41	73	55	30
A3	31	67	66	46	57	31	5	72	48
A4	92	89	46	77	44	72	74	98	26
A5	31	94	5	17	43	13	47	75	50
A6	66	70	33	13	84	30	90	43	28
A7	61	96	15	99	54	78	46	87	37
A8	99	31	72	34	46	69	73	6	62

Algoritmo Propuesto:
Solucion: [1, 6, 2, 3, 8, 0, 5, 9, 4, 7]
 Costo: 135
Tiempo de ejecución: 0.013857872000244242s
Memoria usada: 0.16722774505615234Mb
Algoritmo Original:
Solución: (1, 7, 2, 3, 8, 0, 5, 6, 4, 9)
Costo: 144
Tiempo de ejecución: 0.14731431000018347s
Memoria usada: 0.16855525970458984Mb

Matriz de 10x10

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
A0	46	9	56	71	59	74	44	33	66	50
A1	51	79	6	54	85	4	14	1	34	62
A2	93	70	2	41	73	55	30	31	67	66
A3	46	57	31	5	72	48	92	89	46	77
A4	44	72	74	98	26	31	94	5	17	43
A5	13	47	75	50	66	70	33	13	84	30
A6	90	43	28	61	96	15	99	54	78	46
A7	87	37	99	31	72	34	46	69	73	6
A8	62	40	73	51	34	94	67	29	78	15
A9	81	27	49	63	33	78	72	20	57	2

Algoritmo Propuesto:
Solucion: [1, 6, 8, 0, 3, 10, 7, 2, 9, 5, 4]
 Costo: 154
Tiempo de ejecución: 0.006539341000006971s
Memoria usada: 0.06669807434082031Mb
Algoritmo Original:
Solución: (1, 6, 4, 0, 3, 10, 7, 2, 9, 5, 8)
Costo: 211
Tiempo de ejecución: 4.099014699000236s
Memoria usada: 2.017566680908203Mb

Matriz de 11x11

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
A0	46	9	56	71	59	74	44	33	66	50	51
A1	79	6	54	85	4	14	1	34	62	93	70
A2	2	41	73	55	30	31	67	66	46	57	31
A3	5	72	48	92	89	46	77	44	72	74	98
A4	26	31	94	5	17	43	13	47	75	50	66
A5	70	33	13	84	30	90	43	28	61	96	15
A6	99	54	78	46	87	37	99	31	72	34	46
A7	69	73	6	62	40	73	51	34	94	67	29
A8	78	15	81	27	49	63	33	78	72	20	57
A9	2	22	71	81	98	11	10	35	66	80	70
A10	61	56	47	98	5	78	34	83	78	67	87

Algoritmo Propuesto:
Solucion: [1, 10, 9, 11, 0, 2, 7, 3, 8, 6, 5, 4]
 Costo: 137
Tiempo de ejecución: 0.014600296000025992s
Memoria usada: 0.16137123107910156Mb
Algoritmo Original:
Solución: (1, 10, 9, 11, 0, 2, 7, 5, 3, 6, 8, 4)
Costo: 139
Tiempo de ejecución: 139.69301868900038s
Memoria usada: 11.803703308105469Mb
Correr matrices de 12x12 dimensiones es inviable con el algoritmo original. Tiempo transcurrido: 139.69301868900038s

Matriz de 12x12

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
A0	46	9	56	71	59	74	44	33	66	50	51	79
A1	6	54	85	4	14	1	34	62	93	70	2	41
A2	73	55	30	31	67	66	46	57	31	5	72	48
A3	92	89	46	77	44	72	74	98	26	31	94	5
A4	17	43	13	47	75	50	66	70	33	13	84	30
A5	90	43	28	61	96	15	99	54	78	46	87	37
A6	99	31	72	34	46	69	73	6	62	40	73	51
A7	34	94	67	29	78	15	81	27	49	63	33	78
A8	72	20	57	2	22	71	81	98	11	10	35	66
A9	80	70	61	56	47	98	5	78	34	83	78	67
A10	87	61	91	82	41	3	74	32	28	45	56	89
A11	24	23	61	26	17	13	18	94	73	45	68	51

Algoritmo Propuesto:
Solucion: [1, 2, 7, 9, 5, 0, 11, 6, 10, 8, 3, 4, 12]
 Costo: 132
Tiempo de ejecución: 0.030400838999867l2s
Memoria usada: 0.30182838439941406Mb

Matriz de 13x13

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
A0	46	9	56	71	59	74	44	33	66	50	51	79	6
A1	54	85	4	14	1	34	62	93	70	2	41	73	55
A2	30	31	67	66	46	57	31	5	72	48	92	89	46
A3	77	44	72	74	98	26	31	94	5	17	43	13	47
A4	75	50	66	70	33	13	84	30	90	43	28	61	96
A5	15	99	54	78	46	87	37	99	31	72	34	46	69
A6	73	6	62	40	73	51	34	94	67	29	78	15	81
A7	27	49	63	33	78	72	20	57	2	22	71	81	98
A8	11	10	35	66	80	70	61	56	47	98	5	78	34
A9	83	78	67	87	61	91	82	41	3	74	32	28	45
A10	56	89	24	23	61	26	17	13	18	94	73	45	68
A11	51	14	35	57	1	84	16	54	9	48	3	34	6
A12	18	24	95	66	63	26	99	64	68	59	73	73	2

Algoritmo Propuesto:
Solucion: [12, 3, 5, 8, 1, 9, 10, 6, 13, 11, 7, 2, 0, 4]
 Costo: 132
Tiempo de ejecución: 0.0342690570000741s
Memoria usada: 0.3170318603515625Mb

Matriz de 14x14

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54
A1	85	4	14	1	34	62	93	70	2	41	73	55	30	31
A2	67	66	46	57	31	5	72	48	92	89	46	77	44	72
A3	74	98	26	31	94	5	17	43	13	47	75	50	66	70
A4	33	13	84	30	90	43	28	61	96	15	99	54	78	46
A5	87	37	99	31	72	34	46	69	73	6	62	40	73	51
A6	34	94	67	29	78	15	81	27	49	63	33	78	72	20
A7	57	2	22	71	81	98	11	10	35	66	80	70	61	56
A8	47	98	5	78	34	83	78	67	87	61	91	82	41	3
A9	74	32	28	45	56	89	24	23	61	26	17	13	18	94
A10	73	45	68	51	14	35	57	1	84	16	54	9	48	3
A11	34	6	18	24	95	66	63	26	99	64	68	59	73	73
A12	2	89	10	23	20	70	44	82	6	80	31	92	8	49
A13	16	3	12	39	7	71	40	31	79	94	69	20	71	75

Algoritmo Propuesto:
Solucion: [12, 7, 3, 2, 13, 4, 9, 0, 5, 14, 1, 11, 6, 10, 8]
 Costo: 130
Tiempo de ejecución: 0.08866739899985987s
Memoria usada: 0.7785148620605469Mb

Matriz de 15x15

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85
A1	4	14	1	34	62	93	70	2	41	73	55	30	31	67	66
A2	46	57	31	5	72	48	92	89	46	77	44	72	74	98	26
A3	31	94	5	17	43	13	47	75	50	66	70	33	13	84	30
A4	90	43	28	61	96	15	99	54	78	46	87	37	99	31	72
A5	34	46	69	73	6	62	40	73	51	34	94	67	29	78	15
A6	81	27	49	63	33	78	72	20	57	2	22	71	81	98	11
A7	10	35	66	80	70	61	56	47	98	5	78	34	83	78	67
A8	87	61	91	82	41	3	74	32	28	45	56	89	24	23	61
A9	26	17	13	18	94	73	45	68	51	14	35	57	1	84	16
A10	54	9	48	3	34	6	18	24	95	66	63	26	99	64	68
A11	59	73	73	2	89	10	23	20	70	44	82	6	80	31	92
A12	8	49	16	3	12	39	7	71	40	31	79	94	69	20	71
A13	75	98	35	80	74	11	73	63	44	72	7	15	53	54	46
A14	39	73	98	63	77	18	9	66	15	60	84	59	3	56	73

Algoritmo Propuesto:
Solucion: [12, 6, 1, 0, 15, 9, 3, 2, 5, 7, 10, 4, 13, 14, 8, 11]
 Costo: 137
Tiempo de ejecución: 0.1468274530002418s
Memoria usada: 1.1847305297851562Mb

Matriz de 16x16

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4
A1	14	1	34	62	93	70	2	41	73	55	30	31	67	66	46	57
A2	31	5	72	48	92	89	46	77	44	72	74	98	26	31	94	5
A3	17	43	13	47	75	50	66	70	33	13	84	30	90	43	28	61
A4	96	15	99	54	78	46	87	37	99	31	72	34	46	69	73	6
A5	62	40	73	51	34	94	67	29	78	15	81	27	49	63	33	78
A6	72	20	57	2	22	71	81	98	11	10	35	66	80	70	61	56
A7	47	98	5	78	34	83	78	67	87	61	91	82	41	3	74	32
A8	28	45	56	89	24	23	61	26	17	13	18	94	73	45	68	51
A9	14	35	57	1	84	16	54	9	48	3	34	6	18	24	95	66
A10	63	26	99	64	68	59	73	73	2	89	10	23	20	70	44	82
A11	6	80	31	92	8	49	16	3	12	39	7	71	40	31	79	94
A12	69	20	71	75	98	35	80	74	11	73	63	44	72	7	15	53
A13	54	46	39	73	98	63	77	18	9	66	15	60	84	59	3	56
A14	73	67	98	9	37	91	63	65	15	97	32	46	43	22	60	3
A15	76	13	86	59	59	36	82	70	88	89	89	4	72	25	87	37

Algoritmo Propuesto:
Solucion: [15, 5, 16, 14, 11, 4, 12, 6, 8, 0, 10, 13, 2, 1, 3, 7, 9]
 Costo: 151

Tiempo de ejecución: 0.6654418889997942s
Memoria usada: 4.936328887939453Mb

Matriz de 17x17

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14
A1	1	34	62	93	70	2	41	73	55	30	31	67	66	46	57	31	5
A2	72	48	92	89	46	77	44	72	74	98	26	31	94	5	17	43	13
A3	47	75	50	66	70	33	13	84	30	90	43	28	61	96	15	99	54
A4	78	46	87	37	99	31	72	34	46	69	73	6	62	40	73	51	34
A5	94	67	29	78	15	81	27	49	63	33	78	72	20	57	2	22	71
A6	81	98	11	10	35	66	80	70	61	56	47	98	5	78	34	83	78
A7	67	87	61	91	82	41	3	74	32	28	45	56	89	24	23	61	26
A8	17	13	18	94	73	45	68	51	14	35	57	1	84	16	54	9	48
A9	3	34	6	18	24	95	66	63	26	99	64	68	59	73	73	2	89
A10	10	23	20	70	44	82	6	80	31	92	8	49	16	3	12	39	7
A11	71	40	31	79	94	69	20	71	75	98	35	80	74	11	73	63	44
A12	72	7	15	53	54	46	39	73	98	63	77	18	9	66	15	60	84
A13	59	3	56	73	67	98	9	37	91	63	65	15	97	32	46	43	22
A14	60	3	76	13	86	59	59	36	82	70	88	89	89	4	72	25	87
A15	37	6	99	34	70	62	50	9	93	14	67	91	76	77	62	69	96
A16	47	75	40	53	99	88	27	94	81	12	14	66	40	52	43	69	47

Algoritmo Propuesto:
Solucion: [16, 15, 14, 3, 7, 9, 17, 10, 13, 8, 0, 2, 6, 5, 4, 12, 1, 11]
Costo: 167
Tiempo de ejecución: 5.553894197999853s
Memoria usada: 38.14542102813721Mb

Matriz de 18x18

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14	1
A1	34	62	93	70	2	41	73	55	30	31	67	66	46	57	31	5	72	48
A2	92	89	46	77	44	72	74	98	26	31	94	5	17	43	13	47	75	50
A3	66	70	33	13	84	30	90	43	28	61	96	15	99	54	78	46	87	37
A4	99	31	72	34	46	69	73	6	62	40	73	51	34	94	67	29	78	15
A5	81	27	49	63	33	78	72	20	57	2	22	71	81	98	11	10	35	66
A6	80	70	61	56	47	98	5	78	34	83	78	67	87	61	91	82	41	3
A7	74	32	28	45	56	89	24	23	61	26	17	13	18	94	73	45	68	51
A8	14	35	57	1	84	16	54	9	48	3	34	6	18	24	95	66	63	26
A9	99	64	68	59	73	73	2	89	10	23	20	70	44	82	6	80	31	92
A10	8	49	16	3	12	39	7	71	40	31	79	94	69	20	71	75	98	35
A11	80	74	11	73	63	44	72	7	15	53	54	46	39	73	98	63	77	18
A12	9	66	15	60	84	59	3	56	73	67	98	9	37	91	63	65	15	97
A13	32	46	43	22	60	3	76	13	86	59	59	36	82	70	88	89	89	4
A14	72	25	87	37	6	99	34	70	62	50	9	93	14	67	91	76	77	62
A15	69	96	47	75	40	53	99	88	27	94	81	12	14	66	40	52	43	69
A16	47	14	12	80	96	55	47	7	45	75	8	51	82	35	33	27	38	27
A17	60	85	39	57	57	41	87	8	20	75	62	1	4	55	59	67	57	79

Algoritmo Propuesto:
Solucion: [17, 14, 9, 8, 3, 2, 0, 18, 1, 12, 10, 7, 11, 4, 15, 5, 13, 6, 16]
Costo: 152
Tiempo de ejecución: 4.378263648000029s
Memoria usada: 26.930593490600586Mb

Matriz de 19x19

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14	1	34
A1	62	93	70	2	41	73	55	30	31	67	66	46	57	31	5	72	48	92	89
A2	46	77	44	72	74	98	26	31	94	5	17	43	13	47	75	50	66	70	33
A3	13	84	30	90	43	28	61	96	15	99	54	78	46	87	37	99	31	72	34
A4	46	69	73	6	62	40	73	51	34	94	67	29	78	15	81	27	49	63	33
A5	78	72	20	57	2	22	71	81	98	11	10	35	66	80	70	61	56	47	98
A6	5	78	34	83	78	67	87	61	91	82	41	3	74	32	28	45	56	89	24
A7	23	61	26	17	13	18	94	73	45	68	51	14	35	57	1	84	16	54	9

A8	48	3	34	6	18	24	95	66	63	26	99	64	68	59	73	73	2	89
10																		
A9	23	20	70	44	82	6	80	31	92	8	49	16	3	12	39	7	71	40
31																		
A10	79	94	69	20	71	75	98	35	80	74	11	73	63	44	72	7	15	53
54																		
A11	46	39	73	98	63	77	18	9	66	15	60	84	59	3	56	73	67	98
9																		
A12	37	91	63	65	15	97	32	46	43	22	60	3	76	13	86	59	59	36
82																		
A13	70	88	89	89	4	72	25	87	37	6	99	34	70	62	50	9	93	14
67																		
A14	91	76	77	62	69	96	47	75	40	53	99	88	27	94	81	12	14	66
40																		
A15	52	43	69	47	14	12	80	96	55	47	7	45	75	8	51	82	35	33
27																		
A16	38	27	60	85	39	57	57	41	87	8	20	75	62	1	4	55	59	67
57																		
A17	79	33	38	14	39	16	22	55	44	64	55	31	65	73	66	34	41	72
3																		
A18	78	87	49	77	76	90	77	89	80	60	79	72	23	88	90	49	6	24
79																		

Algoritmo Propuesto:
Solucion: [17, 13, 10, 19, 9, 14, 5, 7, 8, 3, 0, 2, 16, 4, 15, 18, 6, 1, 12, 11]
Costo: 129
Tiempo de ejecución: 0.14437709599997106s
Memoria usada: 0.7774105072021484Mb

Matriz de 20x20

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14	1	34	62
A1	93	70	2	41	73	55	30	31	67	66	46	57	31	5	72	48	92	89	46	77
A2	44	72	74	98	26	31	94	5	17	43	13	47	75	50	66	70	33	13	84	30
A3	90	43	28	61	96	15	99	54	78	46	87	37	99	31	72	34	46	69	73	6
A4	62	40	73	51	34	94	67	29	78	15	81	27	49	63	33	78	72	20	57	2
A5	22	71	81	98	11	10	35	66	80	70	61	56	47	98	5	78	34	83	78	67
A6	87	61	91	82	41	3	74	32	28	45	56	89	24	23	61	26	17	13	18	94
A7	73	45	68	51	14	35	57	1	84	16	54	9	48	3	34	6	18	24	95	66
A8	63	26	99	64	68	59	73	73	2	89	10	23	20	70	44	82	6	80	31	92
A9	8	49	16	3	12	39	7	71	40	31	79	94	69	20	71	75	98	35	80	74
A10	11	73	63	44	72	7	15	53	54	46	39	73	98	63	77	18	9	66	15	60
A11	84	59	3	56	73	67	98	9	37	91	63	65	15	97	32	46	43	22	60	3
A12	76	13	86	59	59	36	82	70	88	89	89	4	72	25	87	37	6	99	34	70
A13	62	50	9	93	14	67	91	76	77	62	69	96	47	75	40	53	99	88	27	94
A14	81	12	14	66	40	52	43	69	47	14	12	80	96	55	47	7	45	75	8	51
A15	82	35	33	27	38	27	60	85	39	57	57	41	87	8	20	75	62	1	4	55
A16	59	67	57	79	33	38	14	39	16	22	55	44	64	55	31	65	73	66	34	41
A17	72	3	78	87	49	77	76	90	77	89	80	60	79	72	23	88	90	49	6	24
A18	79	24	80	10	63	53	8	29	60	23	77	51	1	80	18	96	10	72	50	1
A19	50	64	95	59	81	48	94	16	66	52	14	12	54	53	71	14	80	23	86	64

Algoritmo Propuesto:
Solucion: [17, 1, 6, 16, 15, 20, 7, 0, 18, 11, 12, 8, 10, 9, 4, 2, 5, 19, 13, 14, 3]
Costo: 158
Tiempo de ejecución: 3.1968483539999397s
Memoria usada: 17.628472328186035Mb

Matriz de 21x21

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14	1	34	62	93
A1	70	2	41	73	55	30	31	67	66	46	57	31	5	72	48	92	89	46	77	44	72
A2	74	98	26	31	94	5	17	43	13	47	75	50	66	70	33	13	84	30	90	43	28
A3	61	96	15	99	54	78	46	87	37	99	31	72	34	46	69	73	6	62	40	73	51
A4	34	94	67	29	78	15	81	27	49	63	33	78	72	20	57	2	22	71	81	98	11
A5	10	35	66	80	70	61	56	47	98	5	78	34	83	78	67	87	61	91	82	41	3
A6	74	32	28	45	56	89	24	23	61	26	17	13	18	94	73	45	68	51	14	35	57
A7	1	84	16	54	9	48	3	34	6	18	24	95	66	63	26	99	64	68	59	73	73
A8	2	89	10	23	20	70	44	82	6	80	31	92	8	49	16	3	12	39	7	71	40
A9	31	79	94	69	20	71	75	98	35	80	74	11	73	63	44	72	7	15	53	54	46
A10	39	73	98	63	77	18	9	66	15	60	84	59	3	56	73	67	98	9	37	91	63
A11	65	15	97	32	46	43	22	60	3	76	13	86	59	59	36	82	70	88	89	89	4
A12	72	25	87	37	6	99	34	70	62	50	9	93	14	67	91	76	77	62	69	96	47
A13	75	40	53	99	88	27	94	81	12	14	66	40	52	43	69	47	14	12	80	96	55
A14	47	7	45	75	8	51	82	35	33	27	38	27	60	85	39	57	57	41	87	8	20
A15	75	62	1	4	55	59	67	57	79	33	38	14	39	16	22	55	44	64	55	31	65
A16	73	66	34	41	72	3	78	87	49	77	76	90	77	89	80	60	79	72	23	88	90
A17	49	6	24	79	24	80	10	63	53	8	29	60	23	77	51	1	80	18	96	10	72
A18	50	1	50	64	95	59	81	48	94	16	66	52	14	12	54	53	71	14	80	23	86
A19	64	61	57	77	12	51	68	65	42	56	96	26	41	51	10	17	92	63	43	84	84
A20	14	33	63	12	46	4	51	80	11	77	29	67	83	27	65	33	92	98	40	65	78

Algoritmo Propuesto:
Solucion: [15, 0, 6, 13, 16, 4, 21, 14, 8, 2, 19, 9, 18, 12, 10, 11, 20, 5, 17, 7, 3, 1]
Costo: 138
Tiempo de ejecución: 1.7035663120000208s
Memoria usada: 8.646531105041504Mb

Matriz de 22x22

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14	1	34	62	93	70
A1	2	41	73	55	30	31	67	66	46	57	31	5	72	48	92	89	46	77	44	72	74	98
A2	26	31	94	5	17	43	13	47	75	50	66	70	33	13	84	30	90	43	28	61	96	15
A3	99	54	78	46	87	37	99	31	72	34	46	69	73	6	62	40	73	51	34	94	67	29
A4	78	15	81	27	49	63	33	78	72	20	57	2	22	71	81	98	11	10	35	66	80	70
A5	61	56	47	98	5	78	34	83	78	67	87	61	91	82	41	3	74	32	28	45	56	89
A6	24	23	61	26	17	13	18	94	73	45	68	51	14	35	57	1	84	16	54	9	48	3
A7	34	6	18	24	95	66	63	26	99	64	68	59	73	73	2	89	10	23	20	70	44	82
A8	6	80	31	92	8	49	16	3	12	39	7	71	40	31	79	94	69	20	71	75	98	35
A9	80	74	11	73	63	44	72	7	15	53	54	46	39	73	98	63	77	18	9	66	15	60
A10	84	59	3	56	73	67	98	9	37	91	63	65	15	97	32	46	43	22	60	3	76	13

```
A11 |    86    59    59    36    82    70    88    89    89     4    72    25    87    37     6    99    34    70
62    50     9    93
A12 |    14    67    91    76    77    62    69    96    47    75    40    53    99    88    27    94    81    12
14    66    40    52
A13 |    43    69    47    14    12    80    96    55    47     7    45    75     8    51    82    35    33    27
38    27    60    85
A14 |    39    57    57    41    87     8    20    75    62     1     4    55    59    67    57    79    33    38
14    39    16    22
A15 |    55    44    64    55    31    65    73    66    34    41    72     3    78    87    49    77    76    90
77    89    80    60
A16 |    79    72    23    88    90    49     6    24    79    24    80    10    63    53     8    29    60    23
77    51     1    80
A17 |    18    96    10    72    50     1    50    64    95    59    81    48    94    16    66    52    14    12
54    53    71    14
A18 |    80    23    86    64    61    57    77    12    51    68    65    42    56    96    26    41    51    10
17    92    63    43
A19 |    84    84    14    33    63    12    46     4    51    80    11    77    29    67    83    27    65    33
92    98    40    65
A20 |    78    15    72     8    37    93    23    63    90    31    52    79    61     1    76    52    78    31
81    71    48    14
A21 |    44     9    21    76    14    31    24    11    19    66    22    93    22    10    12    17    76    51
45    16    54    38
```

Algoritmo Propuesto:
Solucion: [17, 10, 1, 20, 12, 22, 9, 7, 2, 15, 21, 3, 5, 18, 19, 13, 4, 0, 14, 6, 8, 11, 16]
 Costo: 159
Tiempo de ejecución: 26.73870970200005s
Memoria usada: 131.0201177597046Mb

Matriz de 23x23

Leyenda: Sol 1 Sol 2 Ambos

```
          T0    T1    T2    T3    T4    T5    T6    T7    T8    T9   T10   T11   T12   T13   T14   T15   T16   T17
T18   T19   T20   T21   T22
    ------------------------------------------------------------------------------------------------------------
 A0 |    46     9    56    71    59    74    44    33    66    50    51    79     6    54    85     4    14     1
34    62    93    70     2
 A1 |    41    73    55    30    31    67    66    46    57    31     5    72    48    92    89    46    77    44
72    74    98    26    31
 A2 |    94     5    17    43    13    47    75    50    66    70    33    13    84    30    90    43    28    61
96    15    99    54    78
 A3 |    46    87    37    99    31    72    34    46    69    73     6    62    40    73    51    34    94    67
29    78    15    81    27
 A4 |    49    63    33    78    72    20    57     2    22    71    81    98    11    10    35    66    80    70
61    56    47    98     5
 A5 |    78    34    83    78    67    87    61    91    82    41     3    74    32    28    45    56    89    24
23    61    26    17    13
 A6 |    18    94    73    45    68    51    14    35    57     1    84    16    54     9    48     3    34     6
18    24    95    66    63
 A7 |    26    99    64    68    59    73    73     2    89    10    23    20    70    44    82     6    80    31
92     8    49    16     3
 A8 |    12    39     7    71    40    31    79    94    69    20    71    75    98    35    80    74    11    73
63    44    72     7    15
 A9 |    53    54    46    39    73    98    63    77    18     9    66    15    60    84    59     3    56    73
67    98     9    37    91
A10 |    63    65    15    97    32    46    43    22    60     3    76    13    86    59    59    36    82    70
88    89    89     4    72
A11 |    25    87    37     6    99    34    70    62    50     9    93    14    67    91    76    77    62    69
96    47    75    40    53
A12 |    99    88    27    94    81    12    14    66    40    52    43    69    47    14    12    80    96    55
47     7    45    75     8
A13 |    51    82    35    33    27    38    27    60    85    39    57    57    41    87     8    20    75    62
1     4    55    59    67
A14 |    57    79    33    38    14    39    16    22    55    44    64    55    31    65    73    66    34    41
72     3    78    87    49
A15 |    77    76    90    77    89    80    60    79    72    23    88    90    49     6    24    79    24    80
10    63    53     8    29
A16 |    60    23    77    51     1    80    18    96    10    72    50     1    50    64    95    59    81    48
94    16    66    52    14
A17 |    12    54    53    71    14    80    23    86    64    61    57    77    12    51    68    65    42    56
96    26    41    51    10
A18 |    17    92    63    43    84    84    14    33    63    12    46     4    51    80    11    77    29    67
83    27    65    33    92
A19 |    98    40    65    78    15    72     8    37    93    23    63    90    31    52    79    61     1    76
52    78    31    81    71
A20 |    48    14    44     9    21    76    14    31    24    11    19    66    22    93    22    10    12    17
76    51    45    16    54
A21 |    38    71    82    29    87    45    99    12    83    41    14     5    26    72    18    43    12    41
99    92    85    55    65
A22 |    70    74    97    99    88    46    67    67    55    82    57    56    87    53    94    63     3    25
42     7    77    55    13
```

Algoritmo Propuesto:
Solucion: [15, 9, 2, 7, 3, 5, 11, 8, 13, 23, 16, 0, 10, 6, 22, 12, 19, 17, 21, 20, 14, 18, 4, 1]
 Costo: 151
Tiempo de ejecución: 3.175706178999917s
Memoria usada: 15.47855281829834Mb

Matriz de 24x24

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14	1	34	62	93	70	2	41
A1	73	55	30	31	67	66	46	57	31	5	72	48	92	89	46	77	44	72	74	98	26	31	94	5
A2	17	43	13	47	75	50	66	70	33	13	84	30	90	43	28	61	96	15	99	54	78	46	87	37
A3	99	31	72	34	46	69	73	6	62	40	73	51	34	94	67	29	78	15	81	27	49	63	33	78
A4	72	20	57	2	22	71	81	98	11	10	35	66	80	70	61	56	47	98	5	78	34	83	78	67
A5	87	61	91	82	41	3	74	32	28	45	56	89	24	23	61	26	17	13	18	94	73	45	68	51
A6	14	35	57	1	84	16	54	9	48	3	34	6	18	24	95	66	63	26	99	64	68	59	73	73
A7	2	89	10	23	20	70	44	82	6	80	31	92	8	49	16	3	12	39	7	71	40	31	79	94
A8	69	20	71	75	98	35	80	74	11	73	63	44	72	7	15	53	54	46	39	73	98	63	77	18
A9	9	66	15	60	84	59	3	56	73	67	98	9	37	91	63	65	15	97	32	46	43	22	60	3
A10	76	13	86	59	59	36	82	70	88	89	89	4	72	25	87	37	6	99	34	70	62	50	9	93
A11	14	67	91	76	77	62	69	96	47	75	40	53	99	88	27	94	81	12	14	66	40	52	43	69
A12	47	14	12	80	96	55	47	7	45	75	8	51	82	35	33	27	38	27	60	85	39	57	57	41
A13	87	8	20	75	62	1	4	55	59	67	57	79	33	38	14	39	16	22	55	44	64	55	31	65
A14	73	66	34	41	72	3	78	87	49	77	76	90	77	89	80	60	79	72	23	88	90	49	6	24
A15	79	24	80	10	63	53	8	29	60	23	77	51	1	80	18	96	10	72	50	1	50	64	95	59
A16	81	48	94	16	66	52	14	12	54	53	71	14	80	23	86	64	61	57	77	12	51	68	65	42
A17	56	96	26	41	51	10	17	92	63	43	84	84	14	33	63	12	46	4	51	80	11	77	29	67
A18	83	27	65	33	92	98	40	65	78	15	72	8	37	93	23	63	90	31	52	79	61	1	76	52
A19	78	31	81	71	48	14	44	9	21	76	14	31	24	11	19	66	22	93	22	10	12	17	76	51
A20	45	16	54	38	71	82	29	87	45	99	12	83	41	14	5	26	72	18	43	12	41	99	92	85
A21	55	65	70	74	97	99	88	46	67	67	55	82	57	56	87	53	94	63	3	25	42	7	77	55
A22	13	57	99	75	16	13	43	56	95	8	43	97	24	86	53	61	95	67	61	89	67	16	30	26
A23	81	4	70	64	42	19	87	21	38	10	7	96	78	39	6	83	80	86	1	33	45	25	72	28

Algoritmo Propuesto:
Solucion: [15, 8, 7, 24, 14, 11, 18, 9, 5, 2, 1, 23, 17, 16, 13, 4, 10, 0, 3, 19, 22, 12, 20, 21, 6]
 Costo: 161
Tiempo de ejecución: 94.62980059700021s
Memoria usada: 401.1124029159546Mb

Matriz de 25x25

Leyenda: Sol 1 Sol 2 Ambos

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24
A0	46	9	56	71	59	74	44	33	66	50	51	79	6	54	85	4	14	1	34	62	93	70	2	41	73
A1	55	30	31	67	66	46	57	31	5	72	48	92	89	46	77	44	72	74	98	26	31	94	5	17	43
A2	13	47	75	50	66	70	33	13	84	30	90	43	28	61	96	15	99	54	78	46	87	37	99	31	72
A3	34	46	69	73	6	62	40	73	51	34	94	67	29	78	15	81	27	49	63	33	78	72	20	57	2
A4	22	71	81	98	11	10	35	66	80	70	61	56	47	98	5	78	34	83	78	67	87	61	91	82	41

```
 A5 |     3    74    32    28    45    56    89    24    23    61    26    17    13    18    94    73    45    68
51    14    35    57     1    84    16
 A6 |    54     9    48     3    34     6    18    24    95    66    63    26    99    64    68    59    73    73
2     89    10    23    20    70    44
 A7 |    82     6    80    31    92     8    49    16     3    12    39     7    71    40    31    79    94    69
20    71    75    98    35    80    74
 A8 |    11    73    63    44    72     7    15    53    54    46    39    73    98    63    77    18     9    66
15    60    84    59     3    56    73
 A9 |    67    98     9    37    91    63    65    15    97    32    46    43    22    60     3    76    13    86
59    59    36    82    70    88    89
A10 |    89     4    72    25    87    37     6    99    34    70    62    50     9    93    14    67    91    76
77    62    69    96    47    75    40
A11 |    53    99    88    27    94    81    12    14    66    40    52    43    69    47    14    12    80    96
55    47     7    45    75     8    51
A12 |    82    35    33    27    38    27    60    85    39    57    57    41    87     8    20    75    62     1
4     55    59    67    57    79    33
A13 |    38    14    39    16    22    55    44    64    55    31    65    73    66    34    41    72     3    78
87    49    77    76    90    77    89
A14 |    80    60    79    72    23    88    90    49     6    24    79    24    80    10    63    53     8    29
60    23    77    51     1    80    18
A15 |    96    10    72    50     1    50    64    95    59    81    48    94    16    66    52    14    12    54
53    71    14    80    23    86    64
A16 |    61    57    77    12    51    68    65    42    56    96    26    41    51    10    17    92    63    43
84    84    14    33    63    12    46
A17 |     4    51    80    11    77    29    67    83    27    65    33    92    98    40    65    78    15    72
8     37    93    23    63    90    31
A18 |    52    79    61     1    76    52    78    31    81    71    48    14    44     9    21    76    14    31
24    11    19    66    22    93    22
A19 |    10    12    17    76    51    45    16    54    38    71    82    29    87    45    99    12    83    41
14     5    26    72    18    43    12
A20 |    41    99    92    85    55    65    70    74    97    99    88    46    67    67    55    82    57    56
87    53    94    63     3    25    42
A21 |     7    77    55    13    57    99    75    16    13    43    56    95     8    43    97    24    86    53
61    95    67    61    89    67    16
A22 |    30    26    81     4    70    64    42    19    87    21    38    10     7    96    78    39     6    83
80    86     1    33    45    25    72
A23 |    28    48    48    18    26    58    79     5    95    39    78    86    39    54    15    55    54    45
97    64    33     6    93    73    93
A24 |    89    63    30    35    52     8     4    51    25    99    54    47    67    12    45    93    80    81
44    57     9     9    61    35     9
```

Algoritmo Propuesto:
Solucion: [17, 7, 13, 21, 10, 25, 20, 18, 19, 5, 4, 12, 6, 3, 15, 23, 9, 11, 1, 0, 2, 16, 24, 8, 22, 14]
 Costo: 154
Tiempo de ejecución: 235.53334240999993s
Memoria usada: 1020.1602363586426Mb
Correr matrices de 26x26 dimensiones es inviable con el algoritmo original. Tiempo transcurrido: 235.53334240999993s

Matriz de 26x26

Leyenda: Sol 1 Sol 2 Ambos

```
--------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
        T0    T1    T2    T3    T4    T5    T6    T7    T8    T9   T10   T11   T12   T13   T14   T15   T16   T17
T18   T19   T20   T21   T22   T23   T24   T25
    ----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
 A0 |    46     9    56    71    59    74    44    33    66    50    51    79     6    54    85     4    14     1
34    62    93    70     2    41    73    55
 A1 |    30    31    67    66    46    57    31     5    72    48    92    89    46    77    44    72    74    98
26    31    94     5    17    43    13    47
 A2 |    75    50    66    70    33    13    84    30    90    43    28    61    96    15    99    54    78    46
87    37    99    31    72    34    46    69
 A3 |    73     6    62    40    73    51    34    94    67    29    78    15    81    27    49    63    33    78
72    20    57     2    22    71    81    98
 A4 |    11    10    35    66    80    70    61    56    47    98     5    78    34    83    78    67    87    61
91    82    41     3    74    32    28    45
 A5 |    56    89    24    23    61    26    17    13    18    94    73    45    68    51    14    35    57     1
84    16    54     9    48     3    34     6
 A6 |    18    24    95    66    63    26    99    64    68    59    73    73     2    89    10    23    20    70
44    82     6    80    31    92     8    49
 A7 |    16     3    12    39     7    71    40    31    79    94    69    20    71    75    98    35    80    74
11    73    63    44    72     7    15    53
 A8 |    54    46    39    73    98    63    77    18     9    66    15    60    84    59     3    56    73    67
98     9    37    91    63    65    15    97
 A9 |    32    46    43    22    60     3    76    13    86    59    59    36    82    70    88    89    89     4
72    25    87    37     6    99    34    70
A10 |    62    50     9    93    14    67    91    76    77    62    69    96    47    75    40    53    99    88
27    94    81    12    14    66    40    52
A11 |    43    69    47    14    12    80    96    55    47     7    45    75     8    51    82    35    33    27
38    27    60    85    39    57    57    41
A12 |    87     8    20    75    62     1     4    55    59    67    57    79    33    38    14    39    16    22
55    44    64    55    31    65    73    66
A13 |    34    41    72     3    78    87    49    77    76    90    77    89    80    60    79    72    23    88
90    49     6    24    79    24    80    10
A14 |    63    53     8    29    60    23    77    51     1    80    18    96    10    72    50     1    50    64
95    59    81    48    94    16    66    52
```

```
A15 |    14    12    54    53    71    14    80    23    86    64    61    57    77    12    51    68    65    42
56    96    26    41    51    10    17    92
A16 |    63    43    84    84    14    33    63    12    46     4    51    80    11    77    29    67    83    27
65    33    92    98    40    65    78    15
A17 |    72     8    37    93    23    63    90    31    52    79    61     1    76    52    78    31    81    71
48    14    44     9    21    76    14    31
A18 |    24    11    19    66    22    93    22    10    12    17    76    51    45    16    54    38    71    82
29    87    45    99    12    83    41    14
A19 |     5    26    72    18    43    12    41    99    92    85    55    65    70    74    97    99    88    46
67    67    55    82    57    56    87    53
A20 |    94    63     3    25    42     7    77    55    13    57    99    75    16    13    43    56    95     8
43    97    24    86    53    61    95    67
A21 |    61    89    67    16    30    26    81     4    70    64    42    19    87    21    38    10     7    96
78    39     6    83    80    86     1    33
A22 |    45    25    72    28    48    48    18    26    58    79     5    95    39    78    86    39    54    15
55    54    45    97    64    33     6    93
A23 |    73    93    89    63    30    35    52     8     4    51    25    99    54    47    67    12    45    93
80    81    44    57     9     9    61    35
A24 |     9    21    27    30    55    61    26    38     1    15    78    73     2    33    69    28    69    48
61    28    43    52     5    87    83    76
A25 |    54    42    69    45     1    14     3    40     7    78    28    68    79     1     5    71    76     7
50    98    60    25    12    36    87    82
```

Descenso del gradiente

In [27]:
```
import matplotlib.pyplot as plt  # Generación de gráficos (otra opción Seaborn)
import numpy as np  # Tratamiento matriz N-dimensionales y otras (fundamental)
from sympy import symbols, sin, cos, exp
from sympy.plotting import plot3d
```

Vamos a buscar el mínimo de la función paraboloide:

$$f(x,y) = x^2 + y^2$$

Obviamente, se encuentra en $(x,y) = (0,0)$, pero probaremos cómo llegamos a él a través del descenso del gradiente.

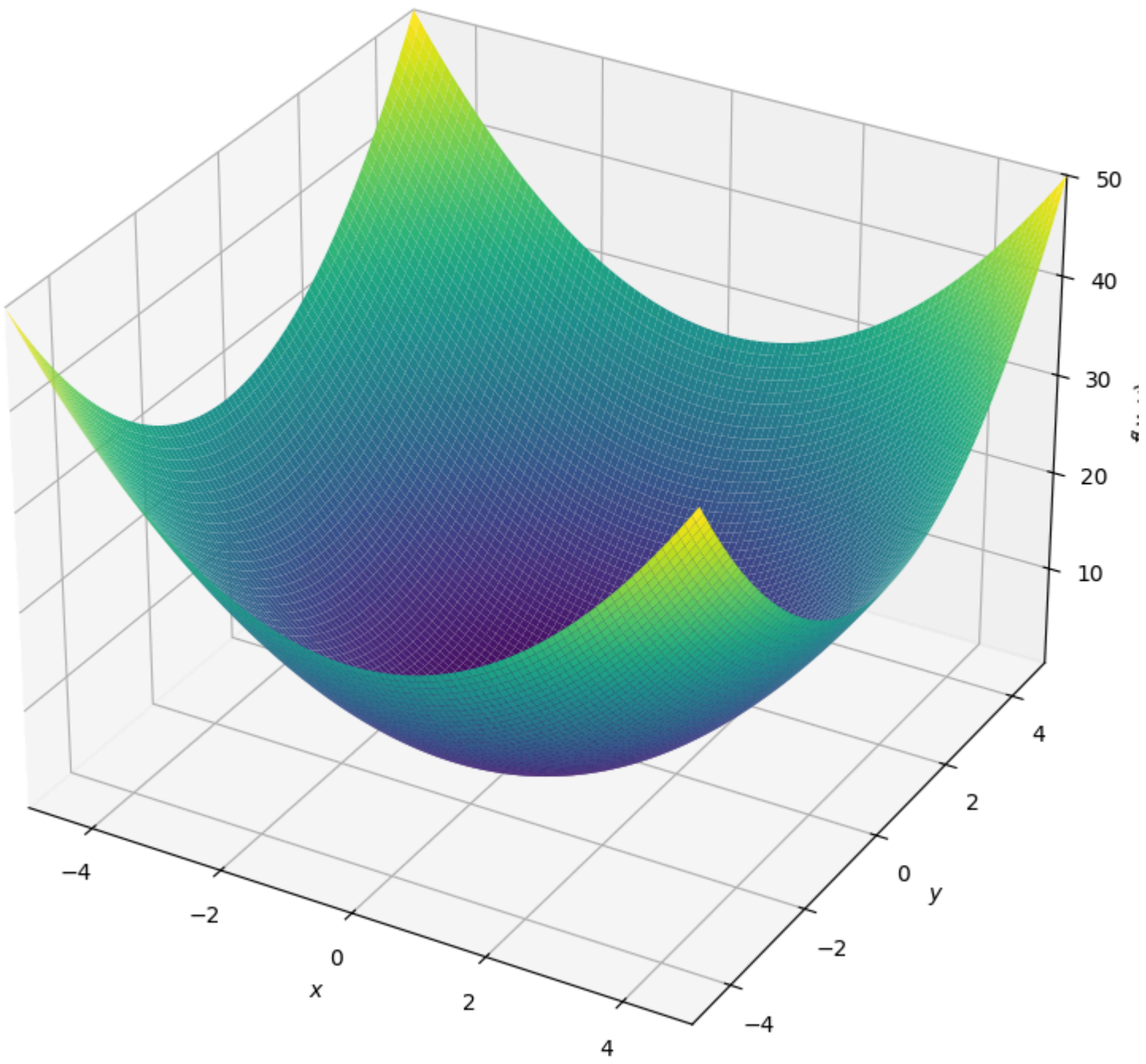
$$\nabla f = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

In [28]:
```
#Definimos la función paraboloide

def f(x: np.ndarray, y: np.ndarray) -> np.ndarray:
    return x ** 2 + y ** 2

def df(X: np.ndarray) -> np.ndarray:
    return np.array([2 * X[0], 2 * X[1]], dtype=float)

df(np.array([1, 2]))
```

Out[28]:
```
array([2., 4.])
```

In [29]:
```
x, y = symbols('x,y')
def plot_surface(expression, range_x, range_y, title) -> None:
    plot3d(expression, range_x, range_y, title=title, size=(8, 8))
    plt.show()
```

In [30]:
```
plot_surface(x**2+y**2, (x, -5, 5), (y, -5, 5),r"Superficie $f(x,y)=x^2 + y^2$")
```

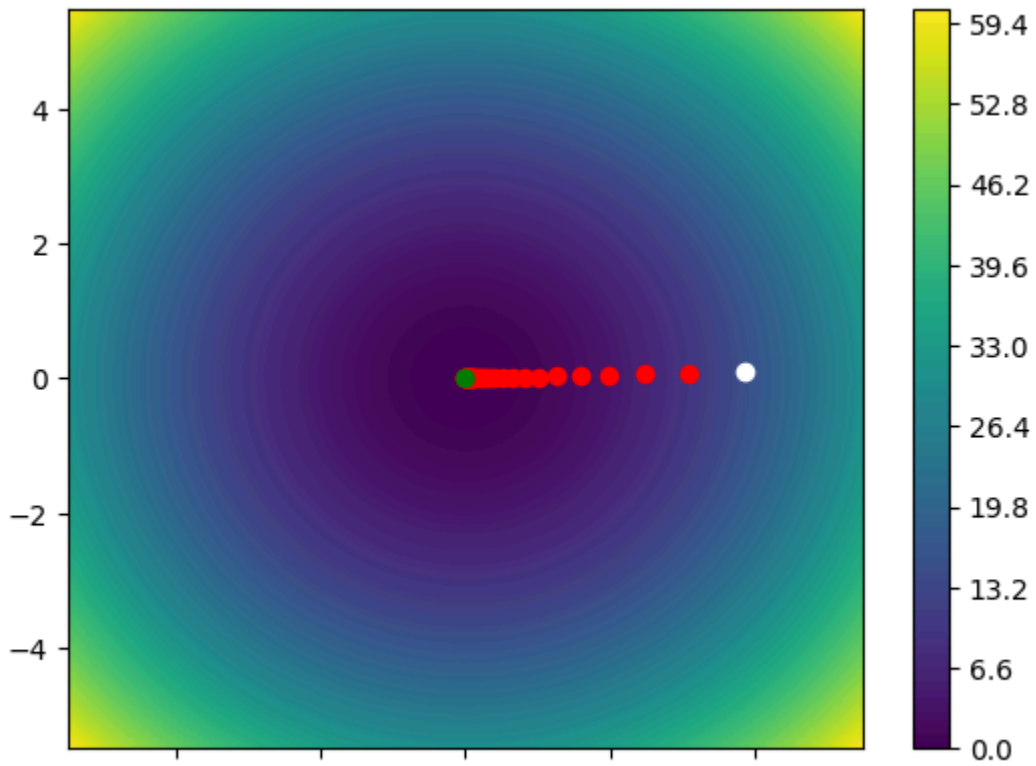


In [31]:
```python
def graphic_gradient_descent(
        f: Callable[[np.ndarray, np.ndarray], np.ndarray],
        df: Callable[[np.ndarray], np.ndarray],
        resolution: int = 100,
        min_range: float = -5.5,
        max_range: float = 5.5,
        learning_rate: float = .1,  #Tasa de aprendizaje. Fija. Sería más efectivo reducirlo a medida que nos acerc
        max_iterations: int = 50,
        tol: float = 1e-6,
):
    #Prepara los datos para dibujar mapa de niveles de Z
    x = np.linspace(min_range, max_range, resolution)
    y = np.linspace(min_range, max_range, resolution)
    X, Y = np.meshgrid(x, y)
    Z = f(X, Y)

    # Pinta el mapa de niveles de Z
    plt.contourf(X, Y, Z, resolution, cmap="viridis")
    plt.colorbar()

    # Generamos un punto aleatorio inicial y pintamos de blanco
    p = np.array([random.uniform(-5, 5), random.uniform(-5, 5)])
    plt.plot(p[0], p[1], "o", c="white")

    for _ in range(max_iterations):
        try:
            grad = df(p)
        except OverflowError:
            print("No hay convergido a alguna solución, se ha desbordado, "
                  "revise la tasa de aprendizaje, o con algún otro punto de "
                  "origen")
            return
        gnorm = np.linalg.norm(grad, ord=2)
        # verificamos con la norma2 si ya está cerca de la tolerancia, para
        # finalizar el bucle, evitando calculos extra
        if gnorm < tol:
            break
        p = p - learning_rate * grad
        plt.plot(p[0], p[1], "o", c="red")
```

```
    #Dibujamos el punto final y pintamos de verde
    plt.plot(p[0], p[1], "o", c="green")
    print("Solución:", p, f(p[0],p[1]))
    plt.show()
```

In [32]:
```
graphic_gradient_descent(f, df)
```

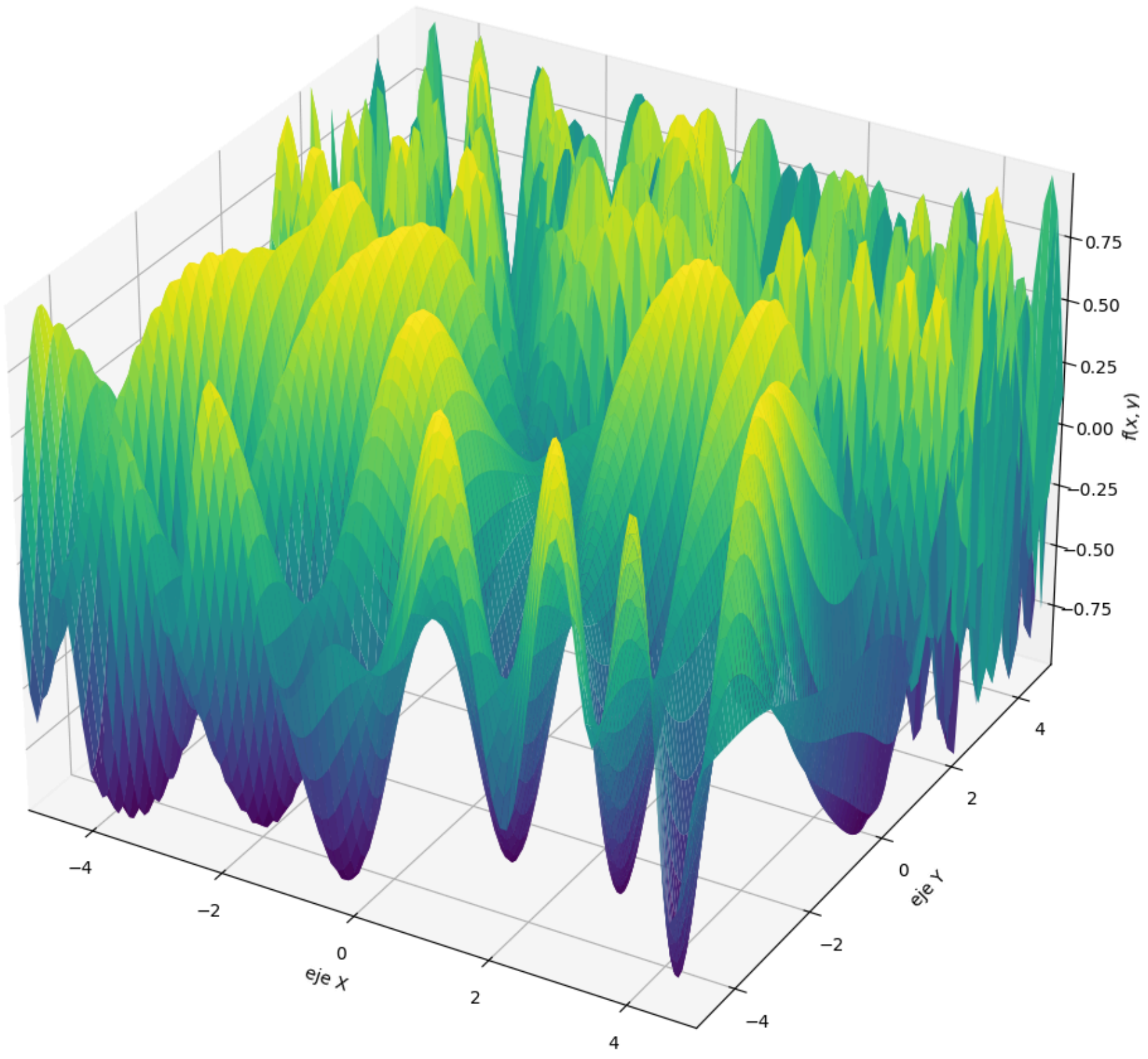
```
Solución: [5.52561443e-05 1.45789976e-06] 3.055366956633174e-09
```



¿Te atreves a optimizar la función?:

$$f(x,y) = sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right) \cdot cos\left(2x + 1 - e^y\right)$$

In [33]:
```
x, y = symbols('x y')
plot3d(
    sin(0.5 * x**2 - 0.25 * y**2 + 3) * cos(2*x + 1 - exp(y)),
    (x, -5, 5),
    (y, -5, 5),
    title=r"Superficie $f(x,y)=sin \left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3 \right) \cdot cos\left(2x + 1 - e^y \r
    xlabel='eje X',
    ylabel='eje Y',
    size=(10, 10)
)
plt.show()
```



Calculando el gradiente de la función.

Dada la función $f(x, y)$, para encontrar el gradiente se procede a derivar respectivamente con respecto a x e y.

$$\begin{aligned}
\frac{df}{dx} &= cos\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)(x)cos\left(2x + 1 - e^y\right) + sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)(-1)sin\left(2x + 1 - e^y\right)(2) \\
&= xcos\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)cos\left(2x + 1 - e^y\right) - 2sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)sin\left(2x + 1 - e^y\right) \\
\frac{df}{dy} &= cos\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)(-\frac{y}{2})cos\left(2x + 1 - e^y\right) + sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)(-1)sin\left(2x + 1 - e^y\right)(-e^y) \\
&= -\frac{y}{2}cos\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)cos\left(2x + 1 - e^y\right) + e^y sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)sin\left(2x + 1 - e^y\right)
\end{aligned}$$

Por ende el gradiente de la funcion $f(x, y)$ es:

$$\nabla f = \begin{bmatrix} xcos\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)cos\left(2x + 1 - e^y\right) - 2sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)sin\left(2x + 1 - e^y\right) \\ -\frac{1}{2}ycos\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)cos\left(2x + 1 - e^y\right) + e^y sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)sin\left(2x + 1 - e^y\right) \end{bmatrix}$$

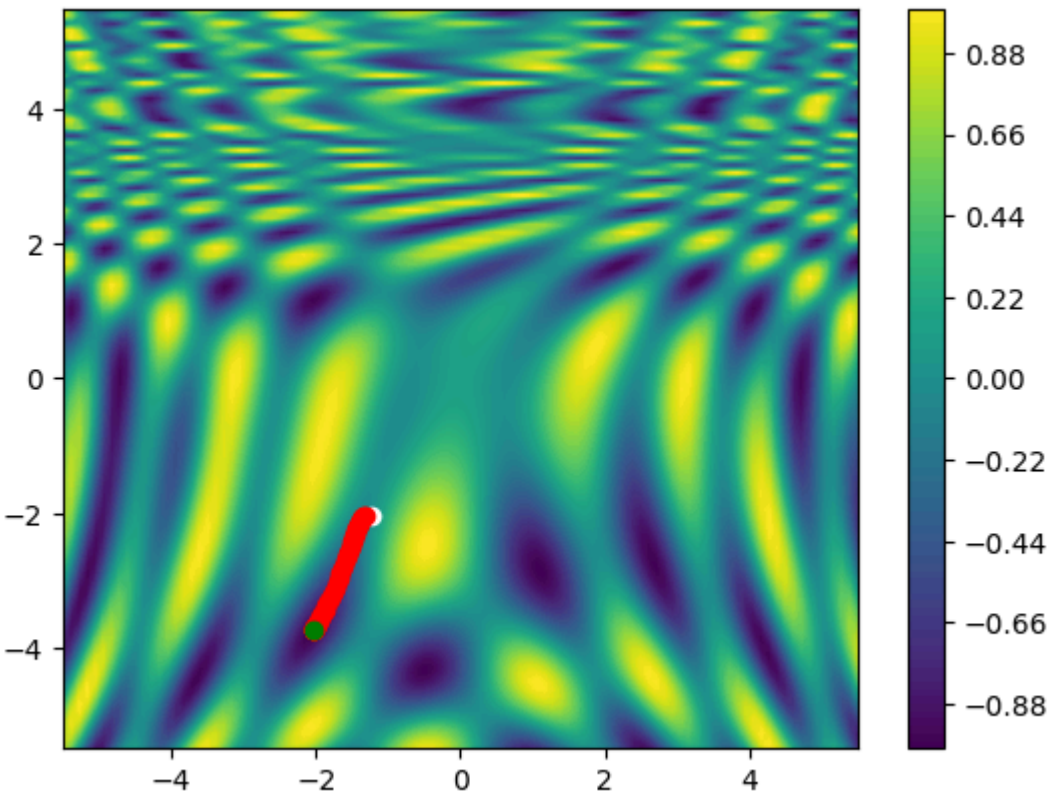
In [34]:
```python
#Definimos la función
def f(X: np.ndarray, Y: np.ndarray) -> np.ndarray:
    return np.sin(0.5 * X**2 - 0.25 * Y**2 + 3) * np.cos(2 * X + 1 - np.exp(Y))

# este gradiente reescrito es algo larga con respecto al gradiente anterior
def df(X: np.ndarray) -> np.ndarray:
    x, y = X[0], X[1]
    # A = x^2/2 - y^2/4 + 3
    arg_sin = 0.5 * x**2 - 0.25 * y**2 + 3
```

```
    # B = 2x + 1 - e^y
    exp_y = np.exp(y)
    arg_cos = 2 * x + 1 - exp_y
    sin_A = np.sin(arg_sin)
    cos_A = np.cos(arg_sin)
    sin_B = np.sin(arg_cos)
    cos_B = np.cos(arg_cos)
    df_dx = x * cos_A * cos_B - 2 * sin_A * sin_B
    df_dy = -0.5 * y * cos_A * cos_B + exp_y * sin_A * sin_B
    return np.array([df_dx, df_dy])

print(f(0, 0))
print(df(np.array([0, 0])))
```

```
0.1411200080598672
[-0.  0.]
```

In [44]:
```
graphic_gradient_descent(f, df)
```

```
Solución: [-2.04707006 -3.74644763] -0.9995957334286385
```



Aproximando el gradiente con la aproximación de la derivada

Para obtener el gradiente de la función de manera aproximada, se hace uso de la definición formal de la derivada:

$$f'(a) = \lim_{x \to 0} \frac{f(a+h) - f(a)}{h}$$

Esta forma de derivar también es conocida como derivada hacia adelante.

In [36]:
```
def df_approx_forward(X: np.ndarray, a:float = 0.01) -> np.ndarray:
    X = X.astype(float)
    grad = np.zeros_like(X)
    for i in range(len(X)):
        X_plus = X.copy()
        X_plus[i] += a
        grad[i] = (f(X_plus[0], X_plus[1]) - f(X[0], X[1])) / a
    return grad

print("a=0.01")
print(df_approx_forward(np.array([0, 0])))
print(f"Diferencia: {np.linalg.norm(df(np.array([0, 0])) - df_approx_forward(np.array([0, 0])))}")
print("a=0.0001")
print(df_approx_forward(np.array([0, 0]), 0.0001))
print(f"Diferencia: {np.linalg.norm(df(np.array([0, 0])) - df_approx_forward(np.array([0, 0]), 0.0001))}")
```

```
a=0.01
[-0.0077713   0.00176216]
Diferencia: 0.007968579217248499
a=0.0001
[-7.77236248e-05  1.76931084e-05]
Diferencia: 7.971203127061816e-05
```

El uso de la derivada hacia adelante según Wikipedia debería ser aceptable siempre y cuando el valor de `a` sea relativamente bajo. Su error es de orden $O(h)$ (lineal).

Sin embargo, existe otra aproximación de derivación numerica que ofrece un error más bajo con respecto a derivar hacia adelante. Dicho método se lo conoce como derivación por diferencias centradas, el cual tiene un error en el orden $O(h^2)$ (cuadrático).

En órdenes de Error, lo ideal es que tenga mayor magnitud, por lo que la aproximación bajo el uso de diferencias centradas ofrece un resultado mucho más cercano a obtener el gradiente por metodos matemáticos.

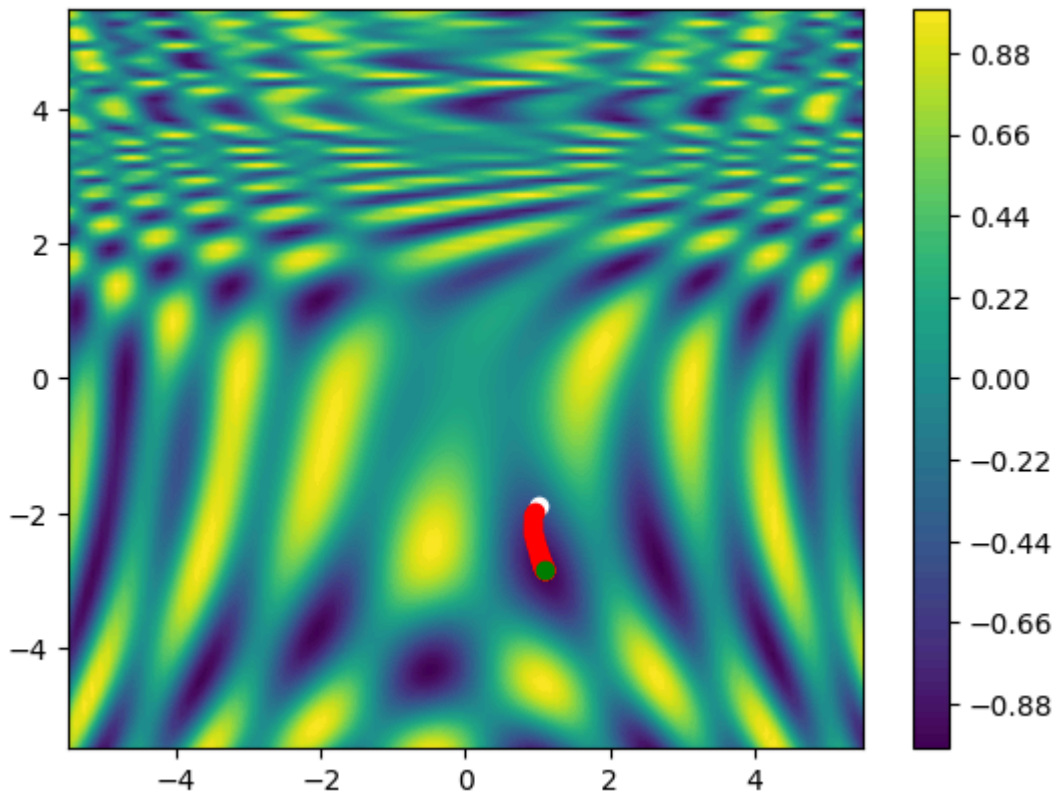
$$f'(a) = \lim_{x \to 0} \frac{f(a+h) - f(a-h)}{2h}$$

In [37]:
```python
def df_approx_central(X: np.ndarray, a:float = 0.01) -> np.ndarray:
    X = X.astype(float)
    grad = np.zeros_like(X)

    for i in range(len(X)):
        X_plus = X.copy()
        X_minus = X.copy()
        X_plus[i] += a
        X_minus[i] -= a
        f_plus = f(X_plus[0], X_plus[1])
        f_minus = f(X_minus[0], X_minus[1])
        grad[i] = (f_plus - f_minus) / (2*a)

    return grad

print("a=0.01")
print(df_approx_central(np.array([0, 0])))
print(f"Diferencia: {np.linalg.norm(df(np.array([0, 0])) - df_approx_central(np.array([0, 0])))}")
print("a=0.0001")
print(df_approx_central(np.array([0, 0]), 0.0001))
print(f"Diferencia: {np.linalg.norm(df(np.array([0, 0])) - df_approx_central(np.array([0, 0]), 0.0001))}")
```

```
a=0.01
[ 0.00000000e+00 -7.05729669e-06]
Diferencia: 7.0572966942572535e-06
a=0.0001
[ 0.00000000e+00 -7.05546732e-10]
Diferencia: 7.05546732149287e-10
```

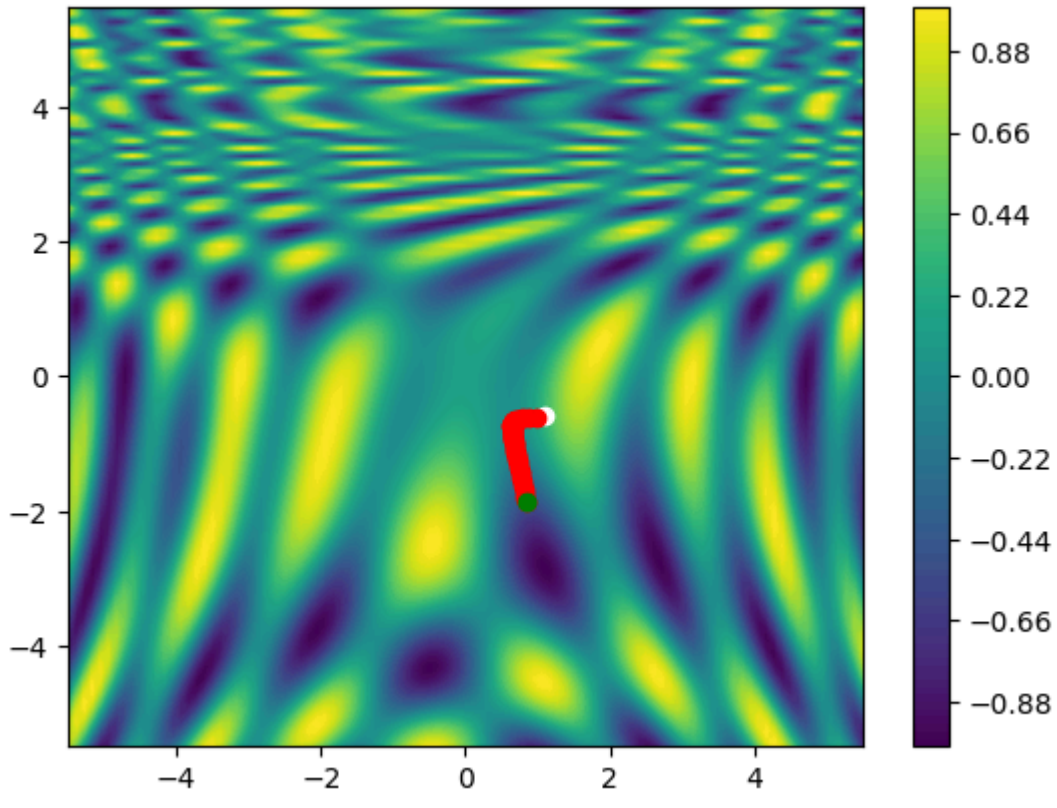
In [38]:
```python
graphic_gradient_descent(f, df_approx_forward)
```

```
Solución: [ 1.09489787 -2.85338377] -0.9999319599850555
```



In [39]:
```python
graphic_gradient_descent(f, df_approx_central)
```

```
Solución: [ 0.84433574 -1.86153924] -0.4975716074176599
```



Nota sobre el uso de herramientas de IA generativa

En la elaboración de este notebook se emplearon herramientas de inteligencia artificial generativa de forma puntual y como apoyo complementario, con los siguientes fines específicos:

1. Asistencia en la creación de métodos para la impresión de tablas de costes y la visualización de resultados, con el objetivo de presentar de forma más clara y amigable las salidas del problema de asignación de tareas.
2. Orientación para la creación de un paquete instalable relacionado con la mejora de la visualización de resultados, con el fin de desacoplar código auxiliar y evitar la saturación del notebook con implementaciones no centrales para el análisis.
3. Formateo de secciones del documento mediante HTML, con el propósito de mejorar la coherencia estructural y la presentación visual del contenido.
4. Síntesis y clarificación de conceptos complejos, incluyendo el análisis de complejidad computacional (Big O Notation) y la explicación de los algoritmos, utilizadas como apoyo conceptual previo para que el autor pudiera posteriormente replicar e implementar las soluciones por cuenta propia.
5. Apoyo en la interpretación de los resultados obtenidos al aplicar distintos algoritmos de asignación de tareas, con el objetivo de orientar optimizaciones posteriores del código, las cuales fueron finalmente diseñadas e implementadas por el autor.
6. Asistencia en el formateo de ecuaciones matemáticas mediante $LaTeX$, para una presentación clara y consistente de los desarrollos teóricos.
7. Corroboración del cálculo de gradientes, utilizando herramientas computacionales externas como Wolfram Alpha, empleadas exclusivamente con fines de verificación.

El resto del contenido del documento: la redacción, el desarrollo principal del código, el diseño metodológico y la interpretación de los resultados es de autoría propia o se basa en material proporcionado en clase y en recursos disponibles públicamente, los cuales han sido reinterpretados, adaptados y reelaborados para los fines de este trabajo.

Referencias Bibliográficas

- [1] "Numerical differentiation," Wikipedia, The Free Encyclopedia. [Online]. [Accedido: 01-feb-2026]