Análisis y Diseño de Algoritmos

Sistema de Asignación de Mecánicos y Averías

Francisco Javier Mercader Martínez Pedro Alarcón Fuentes

En esta memoria se explicará el código utilizado para crear una sistema para la asignación de mecánicos a diferentes averías, evaluando las capacidades de los mecánicos para reparar averías específicas y optimizando el proceso de asignación. La estructura permite procesar varios casos de prueba y validar los resultados de cada asignación.

Índice de Funciones

---

1. Función `encontrar_archivos_in`

```python
def encontrar_archivos_in(directorio='.'):
    """
    Busca todos los archivos .in en el directorio especificado.

    :param directorio: Ruta del directorio donde buscar archivos `.in`.
    :return: Lista de rutas de archivos `.in` encontradas.
    """
    archivos_in = []
    for root, _, files in os.walk(directorio):
        for file in files:
            if file.endswith('.in'):
                archivos_in.append(os.path.join(root, file))
    return archivos_in
```

Descripción: Busca y devuelve una lista de archivos con extensión **`.in`** en el directorio especificado, que se utilizarán como entradas de datos para el programa.

---

2. Función `leer_entrada`

```python
def leer_entrada(file_path):
    """
    Lee el archivo de entrada y convierte la información en una estructura de
    datos adecuada.

    :param file_path: Ruta del archivo de entrada.
    :return: Número de casos de prueba (P) y lista de casos de prueba.
    """
    with open(file_path, 'r') as file:
        lineas = file.readlines()

    P = int(lineas[0].strip())
    casos = []
    indice = 1

    for _ in range(P):
        M, A = map(int, lineas[indice].strip().split())
        indice += 1

        capacidades = []
        for _ in range(M):
            capacidades.append(list(map(int, lineas[indice].strip().split())))
            indice += 1

        casos.append({'M': M, 'A': A, 'capacidades': capacidades})

    return P, casos
```

Descripción y objetivo de la función: La función **`leer_entrada`** se encarga de leer un archivo de entrada **`.in`** y extraer los datos necesarios para cada caso de prueba. Almacena el número de mecánicos y averías, así como las capacidades de cada mecánico para reparar averías específicas en una estructura de datos organizada.

Parámetros:

- **`file_path`** : La ruta al archivo de entrada

Proceso:

1. Lee todas las líneas del archivo de entrada y extrae el número de casos de prueba.
2. Para cada caso, lee el número de mecánicos (**`M`**) y averías (**`A`**).
3. Crea una matriz **`capacidades`** que indica las habilidades de cada mecánico para reparar las averías.
4. Almacena cada caso en una lista de diccionarios.

---

3. Función `factible`

```python
def factible(mecanico, averia, capacidades, asignaciones):
    """
    Verifica si un mecánico puede ser asignado a una avería.

    :param mecanico: Índice del mecánico.
    :param averia: Índice de la avería.
    :param capacidades: Matriz de capacidades de los mecánicos.
    :param asignaciones: Lista de asignaciones de averías.
    :return: True si el mecánico puede reparar la avería y aún no ha sido asignada.
    """
    return capacidades[mecanico][averia] == 1 and asignaciones[averia] == 0
```

Descripción: Verifica si un mecánico específico puede ser asignado a una avería dada. Esto depende de la capacidad del mecánico para repararla y de si la avería ha sido asignada.

Parámetros:

- **`mecanico`**: Índice del mecánico.
- **`averia`**: Índice de la avería.
- **`capacidades`**: Matriz que representa las capacidades de los mecánicos.
- **`asignaciones`**: Lista de asignaciones para controlar qué averías han sido ya asignadas.

---

4. Función `select`

```python
def select(mecanico, capacidades, asignaciones, A):
    """
    Selecciona la mejor avería que un mecánico puede reparar, si es posible.

    :param mecanico: Índice del mecánico.
    :param capacidades: Matriz de capacidades de los mecánicos.
    :param asignaciones: Lista de asignaciones de averías.
    :param A: Número de averías.
    :return: Índice de la avería seleccionada o -1 si no hay ninguna disponible.
    """
    for averia in range(A):
        if factible(mecanico, averia, capacidades, asignaciones):
            return averia
    return -1
```

Descripción: Esta función busca una avería que el mecánico puede reparar. Si encuentra una, devuelve el índice de la avería; si no, devuelve **-1**.

Proceso:

1. Recorre cada avería posible.
2. Usa la función **factible** para verificar si el mecánico puede asignarse a ella.
3. Devuelve el índice de la primera avería factible.

---

5. Función solution

```python
def solution(P, casos):
    """
    Resuelve el problema de asignación para cada caso de prueba.

    :param P: Número de casos de prueba.
    :param casos: Lista de casos de prueba.
    :return: Lista con las soluciones de cada caso.
    """
    resultados = []

    for caso in casos:
        M, A, capacidades = caso['M'], caso['A'], caso['capacidades']
        asignaciones = [0] * A  # Inicializa las asignaciones
        averias_reparadas = 0

        for mecanico in range(M):
            averia_seleccionada = select(mecanico, capacidades, asignaciones, A)
            if averia_seleccionada != -1:
                asignaciones[averia_seleccionada] = mecanico + 1  # Asigna el mecánico
                averias_reparadas += 1

        resultados.append((averias_reparadas, asignaciones))

    return resultados
```

Descripción y objetivo de la función: **solution** implementa el proceso de asignación para cada caso de prueba. Para cada mecánico, selecciona una avería que pueda reparar, y si la encuentra, se asigna y actualiza el número total de averías reparadas.

Proceso:

1. Para cada caso, inicializa **asignaciones** con **0** para indicar avería ha sido asignada.
2. Para cada mecánico, selecciona la avería adecuada mediante la función **select**.
3. Lleva un conteo de averías asignadas y almacena el resultado de cada caso en una lista.

Resultado

```python
if __name__ == '__main__':
    file_paths = encontrar_archivos_in('.')

    for file_path in file_paths:
        P, casos = leer_entrada(file_path)
        resultados = solution(P, casos)

        print(P)
        for resultado in resultados:
            print(resultado[0])
            print(textwrap.fill(' '.join(map(str, resultado[1])), width=86))
        print()
```

```
## 20
## 4
## 1 2 3 5
## 7
## 1 2 3 5 4 9 7 0
## 3
## 1 2 3 0 0
## 7
## 1 4 2 5 3 7 6 0 0 0 0
## 1
## 0 2
## 5
## 1 3 2 6 4 0
## 4
## 3 2 4 1 0 0 0 0 0 0 0 0 0 0 0
## 2
## 1 6
## 11
## 1 2 6 4 3 8 9 7 5 0 10 11 0 0 0 0
## 8
## 3 2 4 1 5 6 7 8
## 4
## 0 2 1 4 3 0 0 0 0 0 0 0 0 0 0
## 2
## 1 0 2
## 9
## 3 1 6 2 4 5 8 7 9
## 3
## 1 2 4
## 11
## 2 3 1 4 7 5 8 10 9 11 6 0
## 6
## 3 1 2 4 5 6
```

```
## 10
## 2 1 6 3 7 5 8 10 4 11
## 2
## 1 4 0
## 8
## 1 2 3 4 5 8 6 7 0 0 0 0 0 0 0 0
## 11
## 1 2 4 6 3 5 9 7 13 8 10
##
## 50
## 6
## 2 1 3 4 5 6 0 0 0 0 0 0
## 5
## 3 1 4 2 0 5 0 0 0 0 0 0 0 0 0 0 0
## 4
## 2 1 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 7
## 1 2 3 5 7 0 6 4 0
## 12
## 2 5 1 6 3 4 10 8 7 9 11 15
## 10
## 2 1 3 4 6 8 11 9 7 10
## 9
## 3 1 2 4 6 8 5 9 0 7 0
## 2
## 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0
## 17
## 4 3 2 1 5 8 6 11 7 10 9 12 17 0 13 0 15 14 16 0 0 0 0
## 19
## 2 1 3 4 5 6 7 10 8 9 12 13 11 15 0 16 18 14 19 17
## 11
## 1 4 2 3 5 6 7 8 10 0 11 0 9 0 0 0 0 0 0 0
## 6
## 1 2 7 3 4 9
## 2
## 0 2 0 0 1 0 0 0 0 0 0 0 0 0
## 4
## 1 3 2 7
## 21
## 2 1 3 4 5 6 9 10 12 7 11 8 16 13 18 15 14 22 17 25 19
## 14
## 2 1 3 4 6 5 7 8 9 10 16 11 17 12
## 3
## 1 3 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 4
## 1 2 3 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 4
## 1 3 4 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 15
```

```
## 1 3 4 2 5 8 6 12 9 7 13 10 11 14 20
## 3
## 1 2 3 0 0 0 0 0
## 14
## 1 5 2 3 6 4 7 11 12 8 13 9 10 14
## 4
## 2 8 1 3
## 4
## 2 4 3 0 1 0 0 0 0 0 0 0 0 0 0 0 0
## 4
## 3 1 2 6
## 5
## 3 4 1 2 5
## 3
## 1 3 2 0 0 0
## 19
## 2 1 8 4 5 3 6 13 7 9 10 12 11 14 16 15 18 17 19
## 9
## 2 3 4 1 5 6 9 8 0 7 0 0 0
## 8
## 1 2 3 5 4 6 8 0 0 0 7 0 0 0 0 0
## 10
## 1 4 2 8 9 3 6 5 10 7
## 3
## 3 1 2
## 6
## 1 2 5 3 8 4
## 8
## 1 8 3 2 4 7 5 10
## 14
## 1 2 6 4 3 8 7 9 13 10 12 5 14 0 11 0 0 0 0 0
## 13
## 1 2 7 3 5 8 9 4 6 11 0 12 10 13 0
## 13
## 1 2 6 3 4 5 8 9 11 12 13 7 10
## 3
## 1 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 9
## 1 2 8 3 5 4 10 6 7
## 9
## 3 1 2 4 5 9 6 7 0 8 0 0 0 0
## 17
## 4 1 2 3 6 5 8 10 7 11 12 13 9 14 15 0 16 17 0 0
## 14
## 3 2 5 1 4 7 10 6 8 9 12 0 13 11 14 0 0 0 0 0 0 0 0 0
## 19
## 1 4 3 2 6 8 5 9 12 7 10 15 11 16 14 17 19 18 13 0 0 0
## 4
## 1 3 4 2 0
```

```
## 4
## 6 1 2 3
## 8
## 2 1 5 6 3 0 0 8 7 0 4 0 0 0 0 0 0 0 0 0 0 0 0
## 21
## 2 3 1 5 4 6 10 11 9 8 7 15 13 12 14 21 18 0 19 16 20 17 0
## 18
## 1 2 3 5 7 4 9 10 12 8 6 13 17 11 20 16 15 14
## 2
## 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 16
## 1 3 2 6 4 7 11 14 10 8 5 0 13 15 9 12 0 16 0
##
## 150
## 2
## 1 2
## 44
## 2 1 5 6 4 9 3 13 12 7 8 10 11 14 23 15 19 16 17 20 25 21 18 22 26 27 28 24 29 30 32 31
## 33 35 36 34 39 38 37 43 40 42 41 44 0 0 0 0 0 0
## 25
## 1 2 7 3 4 8 12 5 6 13 11 14 9 16 15 18 10 19 17 21 20 22 0 24 23 0 0 25
## 26
## 3 1 5 4 6 2 8 9 7 12 11 10 14 13 18 15 16 17 19 24 20 26 22 21 25 0 23
## 10
## 3 2 1 4 7 6 5 9 8 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0
## 14
## 2 3 1 4 5 8 10 6 9 7 12 0 11 14 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 5
## 3 2 1 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0
## 3
## 3 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 29
## 1 3 2 4 7 6 9 5 11 14 8 17 10 12 18 13 21 15 20 22 19 16 24 0 26 23 28 25 27 29 0 0 0
## 0 0 0 0 0 0 0 0 0
## 7
## 4 3 2 1 6 0 5 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0
## 17
## 5 1 3 10 8 2 4 6 7 13 9 11 12 14 15 17 0 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0
## 13
## 2 1 3 6 4 8 13 5 14 9 7 17 10
## 10
## 1 2 4 6 5 3 7 9 8 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 14
## 2 3 1 4 7 5 8 6 9 13 10 11 12 14
## 24
## 1 2 5 3 6 8 7 10 4 9 11 13 15 12 16 18 14 19 17 21 23 20 22 24
```

```
## 18
## 4 1 2 6 5 3 10 7 8 9 19 12 11 13 15 16 17 14
## 21
## 1 2 4 6 3 5 7 9 14 8 12 10 11 16 13 21 15 17 18 22 19
## 20
## 6 4 1 2 8 3 5 7 10 9 12 11 13 14 18 15 16 17 19 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0 0 0
## 8
## 2 3 1 5 0 4 6 7 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0
## 4
## 3 2 9 4
## 30
## 1 2 7 6 4 5 8 3 9 10 11 12 14 13 15 17 16 18 20 19 22 24 21 28 25 23 26 29 27 30 0 0 0
## 0 0 0 0 0 0
## 23
## 2 1 4 7 6 3 9 5 8 13 11 10 12 15 14 16 17 18 19 25 22 20 21
## 26
## 2 1 3 6 8 4 10 11 5 7 15 9 12 16 13 17 14 18 19 22 24 23 20 26 21 29
## 14
## 1 2 4 3 5 8 6 11 7 10 13 9 12 14
## 12
## 1 2 5 3 7 8 11 4 6 14 10 9
## 11
## 1 2 4 3 5 7 6 8 9 10 0 11 0 0 0 0 0 0 0 0
## 44
## 1 2 3 5 4 7 6 8 9 10 11 13 16 12 15 14 17 20 18 25 21 22 23 19 26 24 27 30 28 32 31 35
## 29 34 37 39 41 0 33 43 36 42 38 44 0 40 0 0
## 20
## 3 1 5 2 7 4 9 11 8 6 13 12 14 10 15 16 18 17 0 0 19 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0 0 0
## 10
## 2 3 1 5 4 7 9 6 8 11
## 30
## 1 4 3 6 2 12 8 5 13 7 9 10 11 14 23 15 19 17 16 21 24 22 20 26 27 30 18 28 25 0 29 0
## 14
## 2 5 1 6 7 10 3 4 9 8 11 13 12 14
## 7
## 7 2 4 1 3 9 11
## 7
## 2 1 4 3 6 5 7
## 20
## 1 5 2 3 7 4 6 8 9 10 12 14 19 11 13 16 15 17 20 18
## 19
## 1 4 2 9 3 5 10 13 12 6 7 14 11 16 8 15 21 19 18
## 4
## 3 2 1 4
## 11
## 2 3 1 6 4 9 5 12 7 10 8 0
```

```
## 35
## 3 2 7 6 1 4 13 5 8 9 10 15 12 11 17 14 16 18 20 22 21 19 23 27 24 25 26 31 28 29 30 32
## 33 0 34 35
## 26
## 1 2 3 4 5 6 7 9 11 8 10 12 14 13 18 19 15 16 20 21 22 26 17 24 23 25
## 24
## 2 1 4 3 5 7 6 9 10 11 8 14 15 12 13 16 22 17 21 18 19 23 25 20
## 30
## 1 2 3 4 6 5 7 11 8 10 9 12 14 13 15 18 16 20 19 26 21 17 23 22 24 28 25 30 29 27
## 11
## 1 4 2 5 6 9 8 10 3 11 12
## 7
## 1 2 3 4 6 5 7
## 25
## 2 7 1 4 3 9 11 5 6 8 10 12 16 14 13 15 17 19 23 25 18 21 0 22 20 0 24 0 0 0 0 0 0 0 0
## 12
## 1 2 3 4 5 9 6 11 10 7 8 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 18
## 4 5 1 6 2 3 8 7 11 9 12 10 14 13 15 16 18 0 17
## 2
## 2 3
## 5
## 1 3 5 2 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 9
## 1 2 4 3 5 6 7 10 13
## 25
## 2 1 3 4 5 7 6 8 14 10 9 13 15 11 17 16 12 22 18 19 21 20 25 23 24 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0 0 0 0
## 14
## 3 1 5 6 2 4 7 9 10 11 13 12 0 14 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0
## 34
## 2 4 7 6 5 1 8 3 10 9 11 12 13 14 15 17 16 18 19 22 20 24 25 21 23 26 32 30 28 29 31 33
## 27 34 0 0 0 0
## 40
## 1 2 3 10 4 5 6 8 9 7 11 13 12 18 17 14 16 15 19 20 22 21 25 24 23 27 28 26 31 32 33 30
## 34 37 29 35 36 38 39 40 0 0 0
## 3
## 3 1 5
## 17
## 6 2 3 1 4 7 5 10 8 12 11 17 9 16 13 14 0 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 25
## 1 5 2 3 4 8 7 6 9 12 10 15 13 16 11 14 17 18 23 0 20 22 0 19 21 24 25 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0 0 0 0 0
## 10
## 4 5 1 2 8 3 6 10 9 7 0 0 0 0 0 0 0 0 0 0 0
## 37
## 9 1 2 3 7 5 4 11 6 8 10 12 13 17 14 15 16 18 20 19 21 22 23 24 30 25 26 28 29 32 27 31
## 33 35 34 36 37
```

```
## 10
## 4 2 1 5 6 3 8 7 0 9 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 6
## 3 4 5 1 2 8
## 3
## 2 1 4
## 19
## 1 3 5 2 6 4 8 7 9 12 13 10 14 17 16 11 19 0 15 18
## 32
## 1 3 2 6 4 5 7 16 9 10 12 8 11 18 17 15 14 13 21 19 20 23 25 24 26 27 22 31 29 28 30 36
## 4
## 1 2 7 3
## 26
## 6 4 5 7 1 11 14 2 3 8 10 9 13 12 15 21 19 16 20 22 17 23 18 25 24 26 0 0 0 0 0
## 8
## 1 2 3 6 4 8 7 11
## 43
## 2 3 1 4 6 5 8 7 9 10 11 12 13 15 14 16 21 17 18 19 20 22 23 24 33 25 26 29 28 27 31 30
## 38 36 39 32 40 35 37 43 41 34 42
## 17
## 2 4 5 9 3 1 8 13 10 6 11 14 12 7 15 16 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 12
## 1 4 2 5 3 7 8 6 12 9 13 16
## 38
## 1 2 3 5 6 7 4 8 9 13 10 11 15 12 14 17 19 16 18 20 21 25 22 26 24 28 23 27 32 31 29 30
## 34 36 33 37 38 35
## 18
## 1 3 2 5 4 7 8 6 12 9 10 13 11 14 15 16 17 18 0 0 0 0 0 0 0 0 0 0
## 8
## 4 1 5 2 3 6 0 0 7 0 8 0
## 49
## 3 1 4 6 2 8 11 5 7 10 12 13 9 14 17 16 18 15 19 23 20 24 21 27 32 22 29 28 25 31 37 26
## 30 39 35 33 34 38 41 36 40 42 43 45 44 48 50 46 49
## 30
## 2 1 3 4 6 5 7 8 9 10 13 11 12 14 15 17 16 19 18 21 23 22 20 24 25 30 27 26 28 29
## 4
## 4 1 0 2 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 14
## 2 1 3 4 5 7 6 8 9 11 10 12 14 16
## 5
## 1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 33
## 2 1 3 4 5 6 7 8 13 10 9 12 14 15 16 17 18 11 19 21 23 25 26 20 24 22 27 28 31 30 29 33
## 32 0 0 0 0 0 0 0
## 16
## 1 2 3 5 8 7 4 10 6 11 14 15 9 13 16 12
## 4
## 1 3 4 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 19
```

```
## 1 2 3 4 6 7 5 8 9 10 11 13 12 14 18 20 16 15 17
## 3
## 7 1 6
## 44
## 1 3 7 5 4 6 8 12 2 9 11 10 14 15 13 18 17 24 16 20 19 22 25 29 21 26 28 30 23 31 34 27
## 32 33 35 36 37 39 38 40 41 44 0 42 43 0
## 16
## 1 3 2 4 5 6 7 11 10 8 12 13 9 14 17 16 0
## 10
## 1 2 3 5 8 4 7 6 0 9 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 9
## 1 2 6 4 3 7 5 10 9 0
## 41
## 1 2 5 3 6 4 8 7 10 11 13 12 9 15 20 17 16 14 18 24 21 19 25 22 23 26 27 28 30 33 29 34
## 31 32 37 35 39 36 41 38 0 0 40 0 0 0 0 0 0
## 17
## 2 4 1 5 7 6 9 8 10 3 12 11 13 14 17 19 15
## 32
## 1 3 4 5 6 2 8 9 10 7 11 12 13 14 15 21 17 20 19 18 22 23 16 26 25 27 29 24 28 32 30 33
## 22
## 1 2 5 8 3 4 6 10 7 11 12 9 14 16 13 21 15 20 22 18 19 0 0 0 0 17 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0 0 0 0 0
## 11
## 1 4 3 2 5 6 7 8 14 9 10
## 20
## 2 3 1 5 7 4 6 8 9 10 12 11 14 13 15 17 16 18 20 0 19 0 0 0 0 0 0
## 19
## 2 1 4 3 5 8 7 9 10 12 13 11 6 14 15 16 20 18 17
## 26
## 1 3 4 2 5 6 7 8 9 11 12 14 13 15 10 16 19 18 20 17 22 23 24 25 21 28
## 28
## 4 2 1 3 5 7 6 11 8 13 9 10 12 15 16 14 18 17 19 20 22 21 23 25 24 26 0 0 28 27 0 0 0 0
## 0 0 0 0
## 34
## 1 2 4 3 7 6 5 8 11 9 10 14 13 16 12 17 15 19 18 20 22 25 26 21 24 23 28 29 32 27 33 34
## 30 31
## 23
## 3 1 2 4 6 8 10 9 11 12 5 7 14 13 21 16 15 18 19 17 20 23 24
## 3
## 1 2 0 3 0 0 0 0 0
## 3
## 1 3 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 21
## 1 3 2 4 6 5 9 7 8 10 11 15 17 12 16 13 14 18 19 0 20 21 0 0 0 0 0 0
## 18
## 3 4 2 1 7 9 10 6 5 8 11 13 12 14 16 18 0 17 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 24
## 1 2 4 5 6 7 3 10 15 8 11 14 13 12 16 9 18 17 21 20 25 19 22 24
## 10
```

```
## 1 3 4 2 5 8 6 7 9 10
## 3
## 0 1 2 3 0 0 0 0 0 0 0 0 0 0 0 0
## 27
## 9 1 3 2 4 7 5 6 11 8 10 17 13 12 14 21 15 16 18 19 24 20 0 22 23 25 0 27 26 0 0 0 0 0
## 0
## 11
## 1 2 3 4 5 6 8 10 9 7 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 26
## 4 1 2 7 5 3 6 8 9 10 11 13 12 14 15 18 16 20 21 17 23 19 26 27 24 22 0
## 20
## 1 3 2 10 6 5 4 8 7 11 9 12 17 13 14 16 15 19 18 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0 0 0 0 0
## 15
## 1 6 2 5 3 7 4 8 11 9 12 13 10 0 0 14 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0
## 14
## 1 2 4 6 3 10 5 9 11 8 12 14 7 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0
## 5
## 1 3 4 2 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 27
## 1 2 3 4 5 7 8 6 9 10 11 15 12 17 13 14 16 23 18 21 19 20 22 27 28 24 25
## 44
## 1 2 3 6 7 11 8 9 5 10 15 4 17 12 13 14 19 21 18 22 16 24 20 27 25 28 23 29 32 31 34 30
## 26 35 33 37 39 40 41 42 36 43 44 46
## 5
## 1 3 2 4 5
## 2
## 0 1 0 2
## 2
## 1 0 0 2 0 0 0 0
## 6
## 1 4 2 3 0 5 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 32
## 3 1 2 4 7 6 8 5 9 10 13 14 11 12 17 15 21 16 22 20 26 19 18 23 24 25 27 0 28 29 31 30
## 0 0 33 0 0
## 5
## 1 2 5 3 4
## 14
## 3 1 4 7 2 5 8 9 12 6 11 10 14 0 13 0 0 0 0 0 0 0 0 0 0
## 29
## 1 9 2 3 5 6 4 8 7 11 12 10 14 17 13 16 18 15 20 19 21 22 23 25 24 26 30 27 31
## 5
## 5 1 3 2 6
## 30
## 5 2 1 8 9 4 3 6 11 13 7 12 14 15 18 10 16 25 19 23 22 17 20 26 27 30 24 21 28 29 0 0 0
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 15
```

```
## 1 2 3 4 6 5 7 9 12 8 11 10 14 17 13
## 14
## 1 2 3 7 4 11 5 6 9 8 10 13 14 12
## 13
## 1 3 4 7 2 6 5 9 10 8 11 12 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 20
## 1 4 2 5 3 6 7 14 8 9 20 11 10 12 13 16 15 18 19 0 17 0 0
## 16
## 1 3 5 2 9 8 7 13 14 4 0 6 10 12 15 16 11 0 0 0 0 0 0 0 0 0 0 0
## 13
## 1 2 3 4 5 7 8 10 6 11 9 13 12
## 19
## 1 2 3 8 4 5 9 6 11 7 12 10 14 13 17 16 18 0 15 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0
## 38
## 1 6 2 3 4 5 7 8 13 9 11 12 10 15 14 16 17 18 19 20 21 27 29 22 23 25 24 28 30 26 33 31
## 32 34 0 35 36 39 0 38 0 0 0
## 19
## 2 1 3 4 5 7 6 9 8 10 12 11 13 18 14 15 16 23 17
## 33
## 1 2 5 4 6 8 9 7 3 11 10 17 13 16 12 18 14 20 21 15 19 22 24 26 25 29 23 28 27 30 38 31
## 34
## 36
## 1 2 3 4 7 6 5 8 9 10 11 17 12 13 15 22 14 16 20 18 27 23 21 19 24 25 28 26 29 32 30 31
## 37 33 39 35
## 2
## 1 3
## 26
## 1 2 4 5 3 8 6 9 7 10 11 12 13 14 16 17 15 18 19 20 22 24 21 23 26 25
## 33
## 1 2 3 6 4 5 7 9 12 17 10 11 8 15 14 19 13 21 16 20 22 18 24 23 25 29 30 26 27 32 28 0
## 33 31 0
## 18
## 1 2 4 6 3 9 10 5 8 13 11 12 7 0 14 15 18 0 0 16 0 0 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0
## 40
## 1 3 2 7 4 5 8 9 6 10 11 14 12 15 13 16 17 18 19 20 22 26 25 23 28 24 21 27 31 32 33 29
## 30 34 38 35 39 36 37 40 0 0 0 0 0 0 0 0 0 0 0 0
## 20
## 1 3 4 2 5 6 7 8 9 10 11 15 13 17 18 12 16 19 0 14 0 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 0 0 0 0 0 0 0 0 0 0 0
## 13
## 3 1 2 7 9 5 10 12 4 6 13 8 11 0 0 0 0 0 0 0 0
## 33
## 2 5 1 4 6 9 7 3 13 11 8 10 14 12 16 17 15 19 18 20 22 24 21 23 25 26 27 28 30 29 31 32
## 34
## 34
## 1 5 2 3 4 6 8 10 7 9 12 11 14 13 18 16 21 15 17 22 19 20 23 27 24 29 26 25 30 28 33 31
## 32 34
```

```
## 4
## 2 4 3 1
## 13
## 2 5 4 1 6 7 11 8 3 10 9 12 14
## 29
## 1 6 2 3 4 5 7 8 9 10 12 11 13 14 15 18 16 17 22 20 19 21 27 28 24 25 26 29 0 0 0 0 0 0
## 0 23 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 28
## 3 5 2 1 7 4 8 6 9 14 11 10 13 16 12 15 18 17 19 20 21 24 23 25 22 26 0 0 27 28 0
## 22
## 2 3 1 5 6 7 10 8 9 13 4 14 11 15 17 16 12 18 19 20 21 0 22 0 0 0 0 0 0 0 0 0 0 0
## 27
## 1 2 3 4 7 9 13 5 8 6 12 14 10 11 17 16 18 15 20 19 22 23 21 25 24 26 27 0 0 0 0 0 0 0
## 0 0
## 4
## 7 3 1 5
```

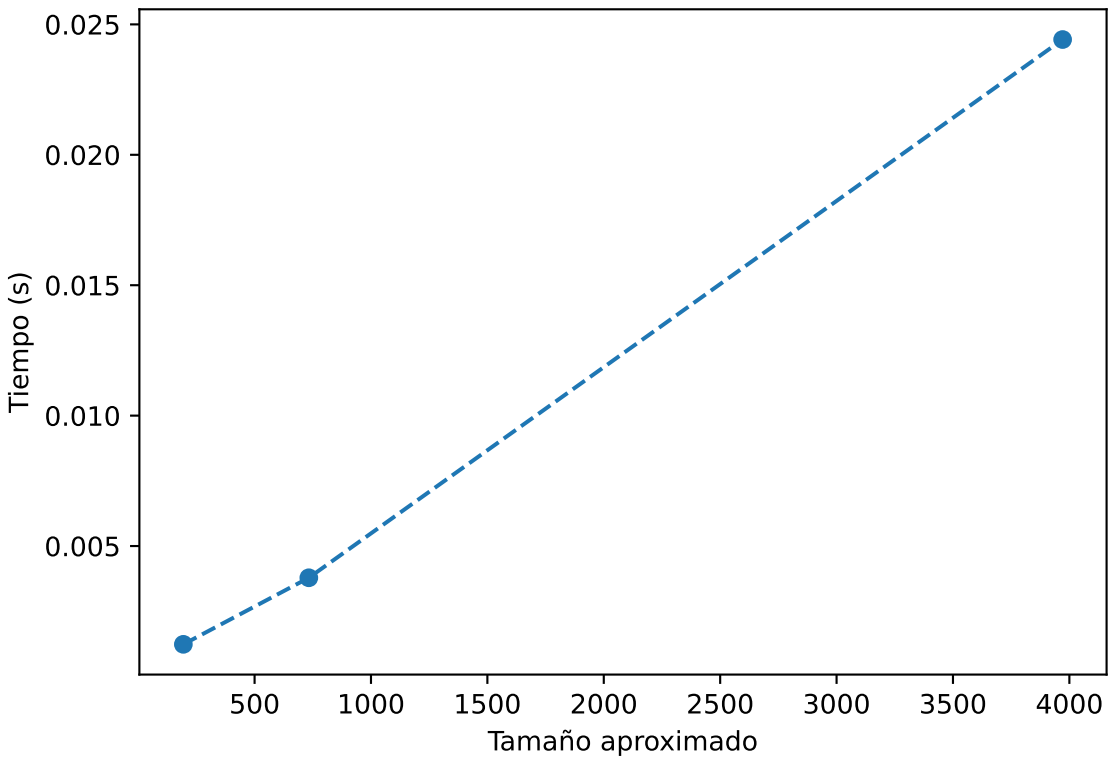
```python
def analyze_complexity(file_paths):
    """
    Mide el tiempo de ejecución para cada archivo y genera un gráfico comparativo.

    :param file_paths: Lista de rutas de archivos `.in`.
    """
    sizes = []
    times = []

    for file_path in file_paths:
        # Medir el tiempo de ejecución para leer y procesar cada archivo
        start_time = time.time()

        # Leer la entrada y procesar los casos de prueba
        P, casos = leer_entrada(file_path)
        resultados = solution(P, casos)

        end_time = time.time()
        elapsed_time = end_time - start_time

        # Calcular el tamaño aproximado de la entrada
        with open(file_path, 'r') as f:
            size = sum(1 for _ in f)

        # Agregar el tamaño y tiempo a las listas
        sizes.append(size)
        times.append(elapsed_time)

    # Graficar el resultado comparativo para todos los archivos
    plt.plot(sizes, times, marker='o', linestyle='--')
    plt.xlabel('Tamaño aproximado')
    plt.ylabel('Tiempo (s)')
```

```
    plt.show()
```

Esta función mide el tiempo de ejecución para cada archivo en **file_paths** y crea un gráfico comparativo de los tiempos frente a los tamaños de entrada.

- **Parámetros:**
  - **file_paths**: lista de rutas de archivos **.in** que se analizarán.
- **Proceso Interno:**
  1. Inicializa las listas **sizes** y **times** para almacenar los tamaños de entrada y los tiempos de ejecución.
  2. Para cada archivo en **file_paths**:
     - Mide el tiempo antes de la ejecución (**start_time**).
     - Ejecuta **leer_entrada** para leer el archivo y **solution** para resolver los casos de prueba.
     - Mide el tiempo después de la ejecución (**end_time**).
     - Calcula el tiempo transcurrido (**elapsed_time**).
     - Determina el tamaño del archivo en líneas (esto representa el tamaño de la entrada) y a los añade a **sizes**.
     - Añade el tiempo transcurrido a **times**.
  3. Genera una gráfica de líneas usando **matplotlib**, donde el eje x representa el tamaño de entrada y el eje y el tiempo de ejecución.

```
analyze_complexity(file_paths)
```



Observaciones del Gráfico

1. **Relación Lineal**:
    - La línea formada por los puntos muestra un crecimiento lineal en el tiempo de ejecución a medida que aumenta el tamaño del archivo de entrada. Esto indica que el tiempo de procesamiento aumenta de manera proporcional con el tamaño de los datos de entrada.
2. **Escala de Tiempo**:
    - El tiempo de ejecución es bajo (en el rango de milisegundos), lo cual es adecuado para los tamaños de entrada analizados. Sin embargo, esto puede cambiar si el tamaño de entrada crece significativamente, dado que la pendiente sugiere un incremento constante.
3. **Pendiente Constante**:
    - La pendiente de la línea es consistente, lo que sugiere que la complejidad del programa es lineal respecto al tamaño de los datos de entrada. En este contexto, el programa probablemente tiene una complejidad $O(N)$ para estos tamaños de entrada.

Conclusiones

1. **Eficiencia del Programa**:
    - Para los tamaños de entrada utilizados, el programa demuestra una eficiencia aceptable y un tiempo de respuesta rápido. Esto es positivo, especialmente si el objetivo es procesar archivos de entrada de tamaño similar.
2. **Escalabilidad**:
    - Aunque el tiempo de ejecución es bajo para los tamaños probados, si el tamaño de entrada continúa creciendo, el tiempo de ejecución también aumentará linealmente. Esto indica que el programa debería manejar bien entradas moderadamente grandes, pero puede volverse más lento en el caso de archivos extremadamente grandes.

En general, el gráfico sugiere que el programa tiene un comportamiento predecible y escalable en un contexto de tamaño de entrada moderado, y su eficiencia es adecuada para los tamaños de datos probados.