Algoritmos de optimización - Seminario

JOHNNY ANDRES ILLESCAS FERNANDEZ
Url: https://github.com/AndresGin/Algoritmosdeoptimizacion.git
Problema:

Combinar cifras y operaciones

El problema consiste en analizar el siguiente problema y diseñar un algoritmo que lo resuelva.

- Disponemos de las 9 cifras del 1 al 9 (excluimos el cero) y de los 4 signos básicos de las operaciones fundamentales: suma(+), resta(-), multiplicación(*) y división(/)
- Debemos combinarlos alternativamente sin repetir ninguno de ellos para obtener una cantidad dada. Un ejemplo sería para obtener el 4: 4+2-6/3*1 = 4

Debe analizarse el problema para encontrar todos los valores enteros posibles planteando las siguientes cuestiones:

- ¿Qué valor máximo y mínimo se pueden obtener según las condiciones del problema?
- ¿Es posible encontrar todos los valores enteros posibles entre dicho mínimo y máximo ?
- Nota: Es posible usar la función de python “eval” para evaluar una expresión:

¿Qué valor máximo y mínimo se pueden obtener según las condiciones del problema?

Para abordar este problema, debemos considerar todas las posibles permutaciones de los números del 1 al 9 y combinarlas con los signos de operación alternando operaciones y números. Además, debemos tener en cuenta las reglas de prioridad de las operaciones matemáticas (multiplicación y división antes de suma y resta) y que no se pueden repetir números ni signos de operación.

- Permutaciones de Números:

Generar todas las permutaciones posibles de los números del 1 al 9.

- Permutaciones de Operaciones:

Generar todas las combinaciones posibles de los cuatro signos de operaciones (sin repetición).

- Construcción de Expresiones:

Combinar cada permutación de números con cada permutación de operaciones, respetando la alternancia (número-operación-número).

- Evaluación de Expresiones:

Evaluar cada expresión respetando la precedencia de las operaciones. Registrar los valores obtenidos para determinar el mínimo y el máximo.

Implementación

Debido a la gran cantidad de combinaciones posibles, es recomendable utilizar un enfoque programático para evaluar todas las posibilidades. A continuación, se presenta una implementación en Python para encontrar los valores mínimo y máximo:

In [3]:
```
# NLibrerias y colecciones
import itertools
import random
import math
from functools import lru_cache
```

In [4]:
```
# Números del 1 al 9
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Operaciones básicas
operaciones = ['+', '-', '*', '/']

# Generar todas las permutaciones de los números y operaciones
permutaciones_numeros = list(itertools.permutations(numeros))
permutaciones_operaciones = list(itertools.permutations(operaciones, 4))

# Función para evaluar una expresión dada
def evaluar_expresion(numeros, operaciones):
    expresion = ""
    for i in range(len(operaciones)):
        expresion += str(numeros[i]) + operaciones[i]
    expresion += str(numeros[-1])
    try:
        return eval(expresion)
    except ZeroDivisionError:
        return None

# Encontrar el valor mínimo y máximo
valores = []
for perm_numeros in permutaciones_numeros:
    for perm_operaciones in permutaciones_operaciones:
        valor = evaluar_expresion(perm_numeros, perm_operaciones)
        if valor is not None and valor == int(valor):
            valores.append(int(valor))

valor_minimo = min(valores)
valor_maximo = max(valores)

print("Valor Minimo: ", valor_minimo)
print("Valor Maximo: ",valor_maximo)
```

```
Valor Minimo:  -69
Valor Maximo:  77
```

¿Es posible encontrar todos los valores enteros posibles entre dicho mínimo y máximo ?

- Explicación del Código

Importar itertools: Se importa itertools para generar permutaciones.

Definir números y operaciones: Se definen los números del 1 al 9 y las operaciones básicas.

Generar permutaciones: Se generan todas las permutaciones posibles de números y operaciones.

Evaluar expresiones: Se construyen y evalúan expresiones para cada combinación de permutaciones de números y operaciones.

Filtrar y almacenar valores enteros: Se almacenan en un conjunto los valores que son enteros para evitar duplicados.

Ordenar y mostrar los valores enteros: Se ordenan los valores enteros obtenidos y se muestran en pantalla.

In [5]:
```python
# Números del 1 al 9
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Operaciones básicas
operaciones = ['+', '-', '*', '/']

# Generar todas las permutaciones de los números y operaciones
permutaciones_numeros = list(itertools.permutations(numeros))
permutaciones_operaciones = list(itertools.permutations(operaciones, 4))

# Función para evaluar una expresión dada
def evaluar_expresion(numeros, operaciones):
    expresion = ""
    for i in range(len(operaciones)):
        expresion += str(numeros[i]) + operaciones[i]
    expresion += str(numeros[-1])
    try:
        return eval(expresion)
    except ZeroDivisionError:
        return None

# Encontrar todos los valores enteros posibles
valores = set()
for perm_numeros in permutaciones_numeros:
    for perm_operaciones in permutaciones_operaciones:
        valor = evaluar_expresion(perm_numeros, perm_operaciones)
        if valor is not None and valor == int(valor):
            valores.add(int(valor))

# Mostrar todos los valores enteros obtenidos
valores_ordenados = sorted(valores)
print("Todos los valores enteros posibles obtenidos son:")
```

```
print(", ".join(map(str, valores_ordenados)))
# Mostrar el conteo total de valores únicos
print("Conteo total de valores únicos:", len(valores))
```

```
Todos los valores enteros posibles obtenidos son:
-69, -68, -67, -66, -65, -64, -63, -62, -61, -60, -59, -58, -57, -56, -55, -54, -
53, -52, -51, -50, -49, -48, -47, -46, -45, -44, -43, -42, -41, -40, -39, -38, -3
7, -36, -35, -34, -33, -32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -2
1, -20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4,
-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 4
0, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 6
0, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
Conteo total de valores únicos: 147
```

¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

Si consideramos que hay 5 posiciones para números y 4 posiciones para operaciones, sin la restricción de que los números y operaciones sean diferentes, el cálculo sería: Para calcular el número total de posibilidades sin tener en cuenta las restricciones (es decir, permitiendo la repetición de números y operaciones), se debe considerar lo siguiente:

Tenemos 9 posiciones para números.

Tenemos 8 posiciones para operaciones (entre los 9 números).

Dado que hay 4 posibles operaciones para cada una de las 8 posiciones y 9 posibles números para cada una de las 9 posiciones, el número total de combinaciones posibles es:

combinaciones_numeros = 9**9

combinaciones_operaciones = 4**8

In [6]:
```
# Calcular el número total de combinaciones posibles
combinaciones_numeros = 9**9
combinaciones_operaciones = 4**8

total_combinaciones = combinaciones_numeros * combinaciones_operaciones
total_combinaciones
```

Out[6]: 25389989167104

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones?

In [7]:
```
'''Mostrar el conteo total de valores únicos '''

print("Conteo total de valores únicos:", len(valores))
```

```
Conteo total de valores únicos: 147
```

Modelo para el espacio de soluciones

¿Cual es la estructura de datos que mejor se adapta al problema? Argumentalo.(Es posible que hayas elegido una al principio y veas la necesidad de cambiar, arguentalo)

- Análisis del espacio de soluciones:

Cifras:

Disponemos de 9 cifras: 1, 2, 3, 4, 5, 6, 7, 8, y 9. Operaciones: Disponemos de 4 operaciones básicas: suma (+), resta (-), multiplicación (*) y división (/), que deben usarse de manera alterna y sin repetir.

Condición de uso alternado:

Las cifras y las operaciones se deben alternar de tal forma que no se repitan las operaciones entre sí y se combinen con los números.

- Generación de combinaciones:

Podemos generar todas las permutaciones de los números y las operaciones disponibles, asegurándonos de cumplir con la restricción de alternancia. Una opción sería usar las funciones de permutación de Python, que generarían todas las combinaciones posibles de números y operaciones.

Una vez generadas las combinaciones, podemos evaluar las expresiones usando eval para calcular los resultados.

- Evaluación de las expresiones:

Usaremos la funciónes en Python para calcular el valor de cada combinación generada. Es necesario manejar posibles excepciones como divisiones por cero, que invalidarían algunas combinaciones.

- Valor máximo y mínimo:

Al evaluar todas las combinaciones, podemos guardar los valores obtenidos. El valor máximo será el mayor de estos resultados, mientras que el mínimo será el menor.

- Determinación de todos los enteros posibles:

Una vez encontrados los valores máximo y mínimo, debemos verificar si es posible obtener todos los enteros entre esos límites. Esto implica que debemos registrar todos los resultados y comprobar si todos los números enteros en ese intervalo pueden obtenerse con alguna combinación.

In [23]:
```python
# Función para evaluar las expresiones
def evaluar_expresiones(expresiones):
    resultados = set()  # Usamos un set para evitar duplicados
    for expresion in expresiones:
```

```
        try:
            resultado = eval(expresion)
            # Solo guardamos si es un número entero
            if isinstance(resultado, int):
                resultados.add(resultado)
        except ZeroDivisionError:
            # Ignorar las expresiones con división por cero
            continue
    return resultados
```

Según el modelo para el espacio de soluciones

¿Cual es la función objetivo?

Dado que el objetivo del problema es encontrar todos los valores enteros posibles, así como determinar el valor máximo y el valor mínimo obtenibles, podemos descomponer la función objetivo en dos enfoques:

- Función objetivo de evaluación de expresiones:

En cada combinación de números y operaciones, la función objetivo sería:

f(números, operadores) = eval(expresión)

Donde la expresión alterna números y operadores. El uso de la función eval() en Python actúa como la función objetivo que evalúa cada combinación de números y operaciones para obtener un valor final.

- Función objetivo global del problema:

El objetivo general es:

Encontrar el valor máximo y el valor mínimo dentro del conjunto de todos los resultados obtenidos al evaluar las combinaciones posibles. Determinar si se pueden generar todos los valores enteros entre el valor mínimo y el valor máximo. Entonces, la función objetivo se podría definir como maximizar y minimizar los resultados obtenidos mediante la evaluación de las combinaciones

¿Es un problema de maximización o minimización?

Este problema no se plantea exclusivamente como un problema de maximización o minimización en el sentido clásico. Más bien, el problema es exploratorio y tiene dos objetivos principales:

Encontrar el valor máximo y mínimo: Una parte del problema implica encontrar los valores máximo y mínimo posibles que se pueden obtener a partir de las combinaciones de los números del 1 al 9 y las operaciones básicas (+, -, *, /). Este componente tiene aspectos tanto de maximización como de minimización, pero no es un problema de optimización clásico, ya que no se busca optimizar una sola función objetivo, sino explorar todas las combinaciones posibles.

Verificar si todos los valores enteros entre el máximo y el mínimo son obtenibles: Esta parte del problema es más una cuestión de completitud o cobertura de valores enteros entre el mínimo y el máximo obtenidos.

Respuesta: Aunque la tarea implica encontrar un valor máximo y un valor mínimo, no es propiamente un problema de maximización en el sentido estricto de la optimización. Es más un problema de búsqueda exhaustiva de soluciones en un espacio finito, con el objetivo de identificar límites y cobertura de soluciones posibles.

In [11]:
```
# Encontrar el valor máximo
valores = []
for perm_numeros in permutaciones_numeros:
    for perm_operaciones in permutaciones_operaciones:
        valor = evaluar_expresion(perm_numeros, perm_operaciones)
        if valor is not None and valor == int(valor):
            valores.append(int(valor))

valor_maximo = max(valores)

print("Valor Maximo: ", valor_maximo)
```

```
Valor Maximo:  77
```

Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

- Combinaciones posibles de 5 números seleccionados del conjunto {1, 2, 3, 4, 5, 6, 7, 8, 9}
- Todas las permutaciones posibles de las 4 operaciones básicas {+, -, *, /}.
- Luego, evaluamos todas las expresiones posibles y almacenamos los resultados enteros únicos.

In [24]:
```
# Números del 1 al 9
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Operaciones básicas
operaciones = ['+', '-', '*', '/']

# Generar todas las permutaciones de 5 números de 9
permutaciones_numeros = list(itertools.permutations(numeros, 5))

# Generar todas las permutaciones de las 4 operaciones
permutaciones_operaciones = list(itertools.permutations(operaciones, 4))

# Función para evaluar una expresión dada
def evaluar_expresion(numeros, operaciones):
    expresion = ""
    for i in range(len(operaciones)):
        expresion += str(numeros[i]) + operaciones[i]
    expresion += str(numeros[-1])
    try:
```

```
        return eval(expresion)
    except ZeroDivisionError:
        return None

# Encontrar todos los valores enteros posibles
valores = set()  # Usamos un conjunto para almacenar resultados únicos
for perm_numeros in permutaciones_numeros:
    for perm_operaciones in permutaciones_operaciones:
        valor = evaluar_expresion(perm_numeros, perm_operaciones)
        if valor is not None and valor == int(valor):
            valores.add(int(valor))  # Agregar valor al conjunto

# Mostrar todos los valores enteros obtenidos
valores_ordenados = sorted(valores)
print("Todos los valores enteros posibles obtenidos son:")
print(", ".join(map(str, valores_ordenados)))

# Mostrar el conteo total de valores únicos
print("Conteo total de valores únicos:", len(valores))
```

```
Todos los valores enteros posibles obtenidos son:
-69, -68, -67, -66, -65, -64, -63, -62, -61, -60, -59, -58, -57, -56, -55, -54, -
53, -52, -51, -50, -49, -48, -47, -46, -45, -44, -43, -42, -41, -40, -39, -38, -3
7, -36, -35, -34, -33, -32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -2
1, -20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4,
-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 4
0, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 6
0, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
Conteo total de valores únicos: 147
```

Calcula la complejidad del algoritmo por fuerza bruta

Respuesta

Cálculo de la Complejidad del Algoritmo por Fuerza Bruta

El cálculo de la complejidad por fuerza bruta implica evaluar el número total de combinaciones posibles que deben explorarse y calcular el costo computacional de evaluar cada una de estas combinaciones. En este caso, estamos trabajando con combinaciones de números y operadores, por lo que el cálculo de la complejidad se basa en las siguientes variables:

- Cifras disponibles (números):

Tienes las cifras del 1 al 9. El número de permutaciones de estas cifras es:

Permutaciones = 9!

- Operadores disponibles: Tienes 4 operadores (+, -, *, /), y necesitas 8 operadores para rellenar los espacios entre los 9 números. El número de permutaciones de los operadores (teniendo en cuenta que puedes repetir operadores) es:

Permutaciones de los operadores = $P(4,8)=8!(8-4)!=8!=$Resultado

- Combinaciones totales: El número total de combinaciones posibles de números y operadores sería el producto del número de permutaciones de los números y de los operadores:

Combinaciones totales=9!×P(4,8)

- Complejidad temporal:

Si asumimos que evaluar cada combinación de cifras y operadores toma un tiempo constante, la complejidad temporal total del algoritmo de fuerza bruta es proporcional al número de combinaciones

- Complejidad espacial:

La complejidad espacial del algoritmo dependerá de la cantidad de combinaciones que se almacenen a lo largo del proceso. Si almacenamos todas las combinaciones generadas (en lugar de evaluarlas y descartarlas inmediatamente), la complejidad espacial sería similar a la complejidad temporal.

In [13]:
```python
# Cálculo de las permutaciones de 5 números seleccionados de 9
P_numeros = math.perm(9, 5)

# Cálculo de las permutaciones de 4 operaciones
P_operaciones = math.perm(4, 4)

# Número total de combinaciones
total_combinaciones = P_numeros * P_operaciones

# Mostrar resultados
print("Permutaciones de números (9P5):", P_numeros)
print("Permutaciones de operaciones (4!):", P_operaciones)
print("Número total de combinaciones:", total_combinaciones)

# Verificar si es el mismo resultado calculado anteriormente
assert total_combinaciones == 362880, "El cálculo no coincide con el resultado e
```

```
Permutaciones de números (9P5): 15120
Permutaciones de operaciones (4!): 24
Número total de combinaciones: 362880
```

Diseña un algoritmo que mejore la complejidad del algortimo por fuerza bruta. Argumenta porque crees que mejora el algoritmo por fuerza bruta

Respuesta

Para mejorar la complejidad del algoritmo por fuerza bruta, podemos utilizar técnicas de programación dinámica o poda (pruning) para reducir el espacio de búsqueda. Tenemos a continuacion un enfoque basado en la poda del espacio de búsqueda y el uso de un almacenamiento intermedio para resultados parciales, lo que puede mejorar la eficiencia.

La idea principal es reducir el número de combinaciones evaluadas mediante el uso de poda y almacenamiento intermedio de resultados parciales. Esto evita evaluaciones redundantes y reduce el espacio de búsqueda.

Almacenamiento Intermedio (Memoización): Almacenar resultados parciales de subexpresiones para evitar recalcular resultados que ya se han evaluado.

Poda (Pruning): Abandonar evaluaciones de subexpresiones que no pueden llevar a resultados válidos. Por ejemplo, si una subexpresión resulta en un número no entero o negativo (si solo buscamos positivos), se puede descartar.

Algoritmo Mejorado

Generar Combinaciones de Números y Operaciones: Usar una generación iterativa en lugar de permutaciones completas para poder aplicar poda.

Evaluar Expresiones con Memoización: Utilizar un diccionario para almacenar y reutilizar resultados de subexpresiones.

Poda del Espacio de Búsqueda: Abandonar ramas del espacio de búsqueda que no pueden llevar a soluciones válidas.

In [14]:
```python
# Números del 1 al 9
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Operaciones básicas
operaciones = ['+', '-', '*', '/']

# Generar todas las combinaciones de 5 números de 9
combinaciones_numeros = list(itertools.combinations(numeros, 5))

# Evaluar la expresión con memoización
@lru_cache(None)
def evaluar_expresion(numeros, operaciones):
    expresion = ""
    for i in range(len(operaciones)):
        expresion += str(numeros[i]) + operaciones[i]
    expresion += str(numeros[-1])
    try:
        resultado = eval(expresion)
        if resultado == int(resultado):  # Solo consideramos resultados enteros
            return int(resultado)
    except ZeroDivisionError:
        pass
    return None

# Encontrar todos los valores enteros posibles con poda
valores = set()
for comb_numeros in combinaciones_numeros:
    for perm_numeros in itertools.permutations(comb_numeros):
        for perm_operaciones in itertools.permutations(operaciones):
            valor = evaluar_expresion(perm_numeros, perm_operaciones)
            if valor is not None:
                valores.add(valor)

# Mostrar todos los valores enteros obtenidos
```

```
valores_ordenados = sorted(valores)
print("Todos los valores enteros posibles obtenidos son:")
print(", ".join(map(str, valores_ordenados)))

# Mostrar el conteo total de valores únicos
print("Conteo total de valores únicos:", len(valores))
```

```
Todos los valores enteros posibles obtenidos son:
-69, -68, -67, -66, -65, -64, -63, -62, -61, -60, -59, -58, -57, -56, -55, -54, -
53, -52, -51, -50, -49, -48, -47, -46, -45, -44, -43, -42, -41, -40, -39, -38, -3
7, -36, -35, -34, -33, -32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -2
1, -20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4,
-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 4
0, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 6
0, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
Conteo total de valores únicos: 147
```

Calcula la complejidad del algoritmo

Respuesta

- Complejidad Total

Para cada combinación de números y permutaciones de operaciones, la evaluación de la expresión se realiza una vez, gracias a la memoización. Así que la complejidad total es:

O=
(Combinaciones de Numeros×Permutaciones de Numeros×Permutaciones de Operaciones)

O=(126×120×24)

O = 362880

Sin embargo, debido a la poda, la cantidad real de evaluaciones puede ser menor. La poda evita algunas evaluaciones, pero no cambia el orden de la complejidad en el peor de los casos. Entonces, en el peor de los casos, la complejidad sigue siendo

O(362880)

- Comparación con la Fuerza Bruta

La complejidad del algoritmo de fuerza bruta también es

O(362880)

Pero el algoritmo mejorado es más eficiente en la práctica debido a la poda y memoización.

La poda reduce el espacio de búsqueda al eliminar combinaciones no válidas tempranamente.

La memoización evita recalcular resultados para subexpresiones, reduciendo el tiempo total de evaluación.

In [15]:
```python
# Cálculo de las permutaciones de 5 números seleccionados de 9
def permutaciones_numeros(n, k):
    return math.factorial(n) // math.factorial(n - k)

# Cálculo de las permutaciones de 4 operaciones
def permutaciones_operaciones(n):
    return math.factorial(n)

# Valores para el problema
n_numeros = 9
k_numeros = 5
n_operaciones = 4

# Cálculo de las permutaciones
P_numeros = permutaciones_numeros(n_numeros, k_numeros)
P_operaciones = permutaciones_operaciones(n_operaciones)

# Número total de combinaciones
total_combinaciones = P_numeros * P_operaciones

# Mostrar resultados
print(f"Permutaciones de números (9P5): {P_numeros}")
print(f"Permutaciones de operaciones (4!): {P_operaciones}")
print(f"Número total de combinaciones: {total_combinaciones}")
```

```
Permutaciones de números (9P5): 15120
Permutaciones de operaciones (4!): 24
Número total de combinaciones: 362880
```

Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios

Respuesta

In [16]:
```python
# Números del 1 al 9
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Operaciones básicas
operaciones = ['+', '-', '*', '/']

# Función para generar una combinación aleatoria de 5 números y 4 operaciones
def generar_datos_aleatorios():
    numeros_seleccionados = random.sample(numeros, 5)
    operaciones_seleccionadas = random.sample(operaciones, 4)
    return numeros_seleccionados, operaciones_seleccionadas

# Generar datos de entrada aleatorios
datos_aleatorios = [generar_datos_aleatorios() for _ in range(10)]

# Mostrar los datos generados
for i, (nums, ops) in enumerate(datos_aleatorios):
    print(f"Juego de datos {i+1}: Números = {nums}, Operaciones = {ops}")
```

```
Juego de datos 1: Números = [5, 6, 1, 4, 9], Operaciones = ['/', '*', '-', '+']
Juego de datos 2: Números = [9, 4, 6, 3, 5], Operaciones = ['+', '-', '*', '/']
Juego de datos 3: Números = [1, 2, 4, 6, 9], Operaciones = ['/', '-', '*', '+']
Juego de datos 4: Números = [2, 1, 3, 7, 6], Operaciones = ['+', '/', '-', '*']
Juego de datos 5: Números = [8, 7, 2, 6, 1], Operaciones = ['-', '*', '+', '/']
Juego de datos 6: Números = [5, 2, 1, 4, 7], Operaciones = ['/', '+', '-', '*']
Juego de datos 7: Números = [9, 1, 8, 7, 5], Operaciones = ['-', '*', '+', '/']
Juego de datos 8: Números = [2, 7, 4, 8, 3], Operaciones = ['-', '/', '*', '+']
Juego de datos 9: Números = [3, 4, 2, 9, 8], Operaciones = ['*', '+', '/', '-']
Juego de datos 10: Números = [6, 3, 9, 8, 5], Operaciones = ['+', '*', '-', '/']
```

Aplica el algoritmo al juego de datos generado

Respuesta

In [17]:

```python
import random
import itertools
from functools import lru_cache

# Números del 1 al 9
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Operaciones básicas
operaciones = ['+', '-', '*', '/']

# Función para generar una combinación aleatoria de 5 números y 4 operaciones
def generar_datos_aleatorios():
    numeros_seleccionados = random.sample(numeros, 5)
    operaciones_seleccionadas = random.sample(operaciones, 4)
    return numeros_seleccionados, operaciones_seleccionadas

# Generar datos de entrada aleatorios
datos_aleatorios = [generar_datos_aleatorios() for _ in range(10)]

# Mostrar los datos generados
for i, (nums, ops) in enumerate(datos_aleatorios):
    print(f"Juego de datos {i+1}: Números = {nums}, Operaciones = {ops}")

# Evaluar la expresión con memoización
@lru_cache(None)
def evaluar_expresion(numeros, operaciones):
    expresion = ""
    for i in range(len(operaciones)):
        expresion += str(numeros[i]) + operaciones[i]
    expresion += str(numeros[-1])
    try:
        resultado = eval(expresion)
        if resultado == int(resultado):  # Solo consideramos resultados enteros
            return int(resultado)
    except ZeroDivisionError:
        pass
    return None

# Aplicar el algoritmo mejorado a cada conjunto de datos generados
for i, (nums, ops) in enumerate(datos_aleatorios):
    valores = set()
    for perm_numeros in itertools.permutations(nums):
        for perm_operaciones in itertools.permutations(ops):
```

```
            valor = evaluar_expresion(perm_numeros, perm_operaciones)
            if valor is not None:
                valores.add(valor)

    # Mostrar todos los valores enteros obtenidos
    valores_ordenados = sorted(valores)
    print(f"\nResultados para el juego de datos {i+1}:")
    print(", ".join(map(str, valores_ordenados)))
    print("Conteo total de valores únicos:", len(valores))
```

```
Juego de datos 1: Números = [2, 5, 4, 9, 8], Operaciones = ['*', '+', '/', '-']
Juego de datos 2: Números = [1, 6, 5, 4, 2], Operaciones = ['*', '/', '-', '+']
Juego de datos 3: Números = [6, 2, 9, 3, 7], Operaciones = ['/', '*', '+', '-']
Juego de datos 4: Números = [7, 9, 6, 3, 1], Operaciones = ['+', '-', '/', '*']
Juego de datos 5: Números = [9, 8, 1, 2, 3], Operaciones = ['-', '*', '+', '/']
Juego de datos 6: Números = [6, 9, 3, 5, 8], Operaciones = ['/', '-', '+', '*']
Juego de datos 7: Números = [5, 9, 7, 4, 8], Operaciones = ['/', '+', '-', '*']
Juego de datos 8: Números = [5, 1, 4, 2, 9], Operaciones = ['*', '+', '-', '/']
Juego de datos 9: Números = [4, 7, 8, 9, 6], Operaciones = ['/', '+', '-', '*']
Juego de datos 10: Números = [1, 6, 7, 8, 3], Operaciones = ['+', '-', '/', '*']

Resultados para el juego de datos 1:
-65, -41, -37, -35, -29, -27, -11, -7, -5, -3, -2, 0, 1, 3, 5, 7, 8, 9, 10, 11, 1
2, 13, 15, 17, 20, 21, 25, 33, 35, 37, 39, 45, 47, 51, 69, 75
Conteo total de valores únicos: 36

Resultados para el juego de datos 2:
-27, -24, -17, -16, -12, -10, -6, -3, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 1
3, 15, 16, 18, 21, 22, 24, 27, 28, 29, 31, 32
Conteo total de valores únicos: 33

Resultados para el juego de datos 3:
-59, -57, -37, -17, -13, -9, -5, -3, -2, -1, 2, 3, 5, 6, 7, 8, 11, 12, 13, 15, 1
6, 17, 21, 23, 25, 27, 31, 41, 43, 63
Conteo total de valores únicos: 30

Resultados para el juego de datos 4:
-60, -54, -44, -38, -30, -14, -10, -6, -4, -2, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
20, 22, 24, 26, 28, 36, 40, 44, 48, 50, 58, 60, 62, 64, 66
Conteo total de valores únicos: 35

Resultados para el juego de datos 5:
-67, -32, -22, -21, -17, -13, -12, -7, -4, -3, -2, -1, 3, 4, 5, 7, 8, 9, 10, 11,
13, 14, 17, 18, 20, 21, 22, 23, 24, 25, 30, 31, 33, 34, 38, 71, 73
Conteo total de valores únicos: 37

Resultados para el juego de datos 6:
-65, -40, -35, -31, -29, -19, -13, -5, -4, -2, -1, 0, 5, 7, 8, 9, 10, 11, 12, 13,
14, 15, 17, 20, 21, 23, 25, 33, 35, 37, 39, 43, 46, 47, 50, 51, 69, 75
Conteo total de valores únicos: 38

Resultados para el juego de datos 7:
-56, -36, -24, -6, 0, 6, 8, 10, 12, 16, 18, 20, 28, 40, 42, 50, 60, 66
Conteo total de valores únicos: 18

Resultados para el juego de datos 8:
-42, -39, -29, -12, -9, -2, 0, 2, 3, 4, 5, 6, 12, 13, 14, 15, 17, 18, 19, 22, 27,
33, 39, 43, 44, 46, 47
Conteo total de valores únicos: 27

Resultados para el juego de datos 9:
-55, -45, -31, -5, -1, 1, 4, 5, 7, 9, 10, 11, 13, 14, 15, 17, 19, 35, 49, 59, 67
Conteo total de valores únicos: 21

Resultados para el juego de datos 10:
-53, -47, -38, -31, -11, -8, -7, -5, -3, -2, 1, 3, 4, 7, 10, 13, 17, 19, 21, 22,
23, 25, 37, 44, 47, 52, 53, 55, 57, 59
Conteo total de valores únicos: 30
```

Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo

Respuesta

https://docs.python.org/3/library/itertools.html

https://www.geeksforgeeks.org/python-functools-lru_cache/

Describe brevemente las lineas de como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posibles variaciones del problema y/o variaciones al alza del tamaño

Respuesta

Considero que se deben tene en cuenta losmsiguientes facores

1. Evaluación Exhaustiva: Utilizar el código proporcionado para realizar una evaluación exhaustiva y encontrar los valores máximos y mínimos posibles.
2. Optimización y Poda: Aplicar técnicas de optimización y poda para mejorar la eficiencia del algoritmo.
3. Programación Dinámica y Heurísticas: Explorar enfoques de programación dinámica y heurísticas para problemas de mayor tamaño.
4. Visualización de Resultados: Visualizar los resultados y analizar patrones para obtener información adicional sobre el problema.

In [ ]: