

Algoritmos de optimización - Seminario

Nombre y Apellidos: 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?

Respuesta

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 [18]: # NLibrerías y colecciones
import itertools
import random
import math
from functools import lru_cache
```

```
In [2]: # 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 [3]: # 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, -37, -36, -35, -34, -33, -32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -21, -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, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77

Conteo total de valores únicos: 147

¿Cuántas 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 [4]: # Calcular el número total de combinaciones posibles
combinaciones_numeros = 9**9
combinaciones_operaciones = 4**8

total_combinaciones = combinaciones_numeros * combinaciones_operaciones
total_combinaciones
```

Out [4]: 25389989167104

¿Cuántas posibilidades hay teniendo en cuenta todas las restricciones?

```
In [4]: '''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

¿Cuál 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, argumentalo)

Respuesta

- Requisitos del Problema

No repetir números ni operaciones:

Necesitamos evaluar todas las permutaciones posibles de números y operaciones.

Almacenar resultados únicos: Debemos almacenar únicamente los resultados enteros únicos. Acceso eficiente: Necesitamos agregar elementos y verificar la existencia de elementos de manera eficiente.

- Justificación del Conjunto (set)

El conjunto garantiza que cada valor se almacene solo una vez, eliminando automáticamente los duplicados. Esto es crucial para asegurar que solo tengamos resultados enteros únicos.

Eficiencia: Las operaciones de agregar un elemento y verificar su existencia son $O(1)$ en promedio. Esto es mucho más eficiente que usar una lista, donde estas operaciones serían $O(n)$.

Simplicidad: El uso de un conjunto simplifica el código, ya que no necesitamos escribir lógica adicional para manejar duplicados.

```
In [12]: # 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))
```

Según el modelo para el espacio de soluciones

¿Cuál es la función objetivo?

¿Es un problema de maximización o minimización?

Respuesta

La función objetivo sería encontrar y enumerar todos los valores enteros posibles que se pueden obtener mediante las combinaciones de los números del 1 al 9 y las operaciones básicas (+, -, *, /) bajo las restricciones de no repetir números ni operaciones.

Definición de la Función Objetivo La función objetivo se puede formular como:

- Maximizar{resultados enteros unicos}

El objetivo específico es maximizar el número de resultados enteros únicos obtenidos, dado que:

Números: Se deben usar 5 números distintos seleccionados de los 9 disponibles (1 al 9).

Operaciones: Se deben usar 4 operaciones distintas seleccionadas de las 4 disponibles (+, -, *, /).

```
In [15]: # 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 [16]: # 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, -37, -36, -35, -34, -33, -32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -21, -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, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 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

Para calcular la complejidad del algoritmo por fuerza bruta, consideramos los siguientes factores:

1. Generación de Permutaciones de Números:

Estamos generando permutaciones de 5 números seleccionados de un conjunto de 9 números. Esto tiene una complejidad combinatoria representada por $P(9, 5) = \frac{9!}{(9-5)!}$.

2. Generación de Permutaciones de Operaciones:

Estamos generando permutaciones de 4 operaciones seleccionadas de un conjunto de 4 operaciones. Esto tiene una complejidad combinatoria representada por $(4!)$.

3. Evaluación de Expresiones:

Para cada combinación de números y operaciones, estamos construyendo y evaluando una expresión. La evaluación de una expresión aritmética es $O(1)$ en términos de tiempo de ejecución, ya que implica una cantidad fija de operaciones.

Cálculo de la Complejidad

1. Permutaciones de Números:

$$P(9, 5) = \frac{9!}{(9-5)!} = 9 \times 8 \times 7 \times 6 \times 5 = 15120$$

2. Permutaciones de Operaciones:

$$[4! = 24]$$

3. Número Total de Combinaciones:

El total de combinaciones de números y operaciones es el producto de las permutaciones de números y operaciones: $[P(9, 5) \times 4! = 15120 \times 24 = 362880]$

4. Evaluación de Cada Combinación:

Evaluamos cada una de las 362880 combinaciones de números y operaciones. La evaluación de cada expresión es $O(1)$, por lo que no afecta significativamente la complejidad total.

Complejidad Total

La complejidad total del algoritmo es:

$$[O(P(9, 5) \times 4!) = O(362880)]$$

Esto significa que el algoritmo tiene una complejidad factorial, que crece muy rápidamente con el aumento del tamaño del conjunto de números y operaciones.

- La complejidad de generar todas las permutaciones de 5 números seleccionados de 9 es $O(15120)$.
- La complejidad de generar todas las permutaciones de 4 operaciones es $O(24)$.
- El número total de combinaciones es $O(362880)$.
- La evaluación de cada combinación es $O(1)$.

```
In [19]: # 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 resulta

```

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

Diseña un algoritmo que mejore la complejidad del algoritmo 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 continuación 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 [20]: # 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, -37, -36, -35, -34, -33, -32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -21, -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, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 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 \times Permutaciones de Numeros \times Permutaciones de Operaciones)

$O = (126 \times 120 \times 24)$

$O = 362880$

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 [21]: # 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 [22]: # 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 = [7, 6, 5, 2, 4], Operaciones = ['*', '/', '+', '-']
 Juego de datos 2: Números = [5, 1, 7, 8, 3], Operaciones = ['*', '/', '+', '-']
 Juego de datos 3: Números = [9, 2, 7, 4, 8], Operaciones = ['+', '*', '-', '/']
 Juego de datos 4: Números = [6, 7, 8, 9, 2], Operaciones = ['*', '-', '+', '/']
 Juego de datos 5: Números = [6, 9, 4, 3, 7], Operaciones = ['*', '/', '-', '+']
 Juego de datos 6: Números = [5, 3, 9, 2, 1], Operaciones = ['+', '*', '-', '/']
 Juego de datos 7: Números = [1, 4, 3, 2, 7], Operaciones = ['/', '*', '-', '+']
 Juego de datos 8: Números = [8, 2, 5, 9, 3], Operaciones = ['/', '+', '-', '*']
 Juego de datos 9: Números = [3, 2, 4, 6, 7], Operaciones = ['/', '-', '*', '+']
 Juego de datos 10: Números = [4, 2, 3, 7, 1], Operaciones = ['-', '/', '*', '+']

Aplica el algoritmo al juego de datos generado

Respuesta

```
In [23]: 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 = [5, 6, 4, 3, 1], Operaciones = ['/', '+', '-', '*']
Juego de datos 2: Números = [9, 6, 7, 5, 2], Operaciones = ['/', '*', '+', '-']
Juego de datos 3: Números = [7, 5, 9, 8, 6], Operaciones = ['/', '*', '-', '+']
Juego de datos 4: Números = [7, 8, 5, 9, 2], Operaciones = ['/', '-', '+', '*']
Juego de datos 5: Números = [6, 9, 5, 1, 4], Operaciones = ['-', '+', '*', '/']
Juego de datos 6: Números = [3, 4, 6, 7, 1], Operaciones = ['/', '+', '-', '*']
Juego de datos 7: Números = [2, 8, 4, 5, 7], Operaciones = ['-', '/', '*', '+']
Juego de datos 8: Números = [7, 8, 6, 1, 2], Operaciones = ['+', '*', '/', '-']
Juego de datos 9: Números = [3, 8, 4, 9, 1], Operaciones = ['+', '/', '-', '*']
Juego de datos 10: Números = [1, 3, 2, 8, 6], Operaciones = ['+', '-', '*', '/']

Resultados para el juego de datos 1:

-23, -17, -16, -11, -9, -5, -2, -1, 1, 3, 4, 6, 7, 11, 12, 13, 17, 19, 21, 22, 23, 26, 29, 31

Conteo total de valores únicos: 24

Resultados para el juego de datos 2:

-55, -35, -23, -15, -7, 1, 5, 9, 13, 17, 25, 29, 41, 49, 61, 65

Conteo total de valores únicos: 16

Resultados para el juego de datos 3:

0, 10, 14

Conteo total de valores únicos: 3

Resultados para el juego de datos 4:

-54, -34, -24, -22, -14, -4, 18, 22, 24, 30, 32, 34, 38, 40, 42, 48, 62, 64

Conteo total de valores únicos: 18

Resultados para el juego de datos 5:

-45, -35, -25, -17, -10, -5, 0, 2, 10, 17, 20, 23, 25, 28, 35, 37, 43, 47, 53, 55

Conteo total de valores únicos: 20

Resultados para el juego de datos 6:

-35, -25, -19, -14, -11, -9, -7, -4, -1, 0, 1, 2, 5, 6, 8, 9, 11, 13, 14, 15, 17, 19, 20, 21, 23, 25, 27, 28, 29, 31, 41, 43

Conteo total de valores únicos: 32

Resultados para el juego de datos 7:

-49, -31, -27, -25, -19, -9, -7, -4, -1, 2, 3, 5, 6, 8, 9, 11, 14, 15, 17, 18, 23, 27, 29, 35, 41, 45, 53, 59

Conteo total de valores únicos: 28

Resultados para el juego de datos 8:

-52, -48, -39, -37, -32, -21, -16, -12, -3, 0, 2, 3, 4, 5, 9, 11, 12, 13, 14, 15, 16, 17, 18, 23, 28, 30, 33, 36, 39, 43, 45, 48, 52, 53, 54, 58, 60

Conteo total de valores únicos: 37

Resultados para el juego de datos 9:

-65, -28, -25, -24, -20, -19, -15, -14, -11, -4, -3, -2, -1, 4, 5, 7, 8, 9, 10, 11, 13, 14, 16, 19, 20, 21, 23, 26, 27, 28, 29, 30, 31, 34, 38, 41, 71, 73

Conteo total de valores únicos: 38

Resultados para el juego de datos 10:

-43, -20, -16, -13, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 7, 8, 11, 12, 13, 15, 16, 17, 19, 20, 21, 22, 24, 26, 28, 47, 49

Conteo total de valores únicos: 35

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 tener en cuenta los siguientes factores

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 []: