Algoritmos de optimización - Seminario

Nombre y Apellidos:
Url: https://github.com/.../03MAIR---Algoritmos-de-Optimizacion---2019/tree/master/SEMINARIO
Problema:

> 1. Sesiones de doblaje
> 2. Organizar los horarios de partidos de La Liga
> 3. Combinar cifras y operaciones

Descripción del problema:(copiar enunciado)

> 3.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 ?

(*) La respuesta es obligatoria

In [1]:

```python
import itertools

def evaluate_expression(expression):

    try:
        return eval(expression)
    except ZeroDivisionError:
        return None

def find_combinations(target):
    digits = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    operators = ['+', '-', '*', '/']
    results = []
    result=''

    for digits_perm in itertools.permutations(digits, 4):
        for ops_perm in itertools.permutations(operators, 3):
            expression = f"{digits_perm[0]} {ops_perm[0]} {digits_perm[1]} {ops_perm[1]} {digits_perm[2]} {ops_perm[2]} {digits_perm[3]}"
            result = evaluate_expression(expression)
            if result is not None and result == target:
                #results.append(expression)
                result=expression
                return result

    #return results


target_number = 5
combinations = find_combinations(target_number)
print(f'{combinations} = {target_number}')
```

```
1 + 2 / 3 * 6 = 5
```

A las dos preguntas del enunciado:

> 1. El rango de valores posibles de este problema es el $[-70, 71]$ siendo sus extremos el valor máximo y mínimo.
>
> 1. Si, es posible encontrar todos los numeros enteros en este intervalo, adjunto codigo:

In [2]:

```python
c=0
result=1
while result!= None:
    result = find_combinations(c)
    print(f'{result} = {c}')
    c+=1


c=0
result=1
while result!= None:
    result = find_combinations(c)
    print(f'{result} = {c}')
    c-=1
```

```
1 - 2 * 3 + 5 = 0
1 * 2 + 3 - 4 = 1
1 + 2 * 3 - 5 = 2
1 + 2 * 3 - 4 = 3
1 * 2 - 3 + 5 = 4
1 + 2 / 3 * 6 = 5
1 * 2 - 3 + 7 = 6
1 * 2 - 3 + 8 = 7
1 * 2 - 3 + 9 = 8
1 / 2 * 4 + 7 = 9
1 / 2 * 4 + 8 = 10
1 - 2 + 3 * 4 = 11
1 / 2 * 6 + 9 = 12
1 + 2 * 8 - 4 = 13
1 - 2 + 3 * 5 = 14
1 + 2 * 9 - 4 = 15
1 + 2 * 9 - 3 = 16
1 - 2 + 3 * 6 = 17
1 - 3 + 4 * 5 = 18
1 - 2 + 4 * 5 = 19
1 - 2 + 3 * 7 = 20
1 + 3 * 8 - 4 = 21
1 - 3 + 4 * 6 = 22
1 - 2 + 3 * 8 = 23
1 + 3 * 9 - 4 = 24
1 + 6 / 2 * 8 = 25
1 - 2 + 3 * 9 = 26
1 - 2 + 4 * 7 = 27
1 - 3 + 5 * 6 = 28
1 - 2 + 5 * 6 = 29
1 - 3 + 4 * 8 = 30
1 - 2 + 4 * 8 = 31
1 - 4 + 5 * 7 = 32
1 - 3 + 5 * 7 = 33
1 - 2 + 5 * 7 = 34
1 - 2 + 4 * 9 = 35
2 - 1 + 5 * 7 = 36
1 - 4 + 5 * 8 = 37
1 - 3 + 5 * 8 = 38
1 - 2 + 5 * 8 = 39
1 - 3 + 6 * 7 = 40
1 - 2 + 6 * 7 = 41
1 - 4 + 5 * 9 = 42
1 - 3 + 5 * 9 = 43
1 - 2 + 5 * 9 = 44
1 - 4 + 6 * 8 = 45
1 - 3 + 6 * 8 = 46
1 - 2 + 6 * 8 = 47
1 + 6 * 9 - 7 = 48
2 - 1 + 6 * 8 = 49
1 - 5 + 6 * 9 = 50
1 - 4 + 6 * 9 = 51
1 - 3 + 6 * 9 = 52
1 - 2 + 6 * 9 = 53
1 - 3 + 7 * 8 = 54
1 - 2 + 7 * 8 = 55
1 + 7 * 9 - 8 = 56
2 - 1 + 7 * 8 = 57
1 - 6 + 7 * 9 = 58
1 - 5 + 7 * 9 = 59
1 - 4 + 7 * 9 = 60
1 - 3 + 7 * 9 = 61
1 - 2 + 7 * 9 = 62
7 * 8 - 2 + 9 = 63
2 - 1 + 7 * 9 = 64
2 / 1 + 7 * 9 = 65
1 - 7 + 8 * 9 = 66
1 - 6 + 8 * 9 = 67
1 - 5 + 8 * 9 = 68
1 - 4 + 8 * 9 = 69
1 - 3 + 8 * 9 = 70
1 - 2 + 8 * 9 = 71
None = 72
1 - 2 * 3 + 5 = 0
1 - 2 * 3 + 4 = -1
1 * 2 + 3 - 7 = -2
1 - 2 / 3 * 6 = -3
1 * 2 + 3 - 9 = -4
1 - 2 / 3 * 9 = -5
1 / 2 * 4 - 8 = -6
1 / 2 * 4 - 9 = -7
1 - 2 * 6 + 3 = -8
1 + 2 - 3 * 4 = -9
1 - 2 * 7 + 3 = -10
1 - 2 * 8 + 4 = -11
1 + 2 - 3 * 5 = -12
1 - 2 * 9 + 4 = -13
1 - 2 * 9 + 3 = -14
1 + 2 - 3 * 6 = -15
1 + 3 - 4 * 5 = -16
1 + 2 - 4 * 5 = -17
1 + 2 - 3 * 7 = -18
1 - 3 * 8 + 4 = -19
1 + 3 - 4 * 6 = -20
1 + 2 - 3 * 8 = -21
1 - 3 * 9 + 4 = -22
1 - 6 / 2 * 8 = -23
1 + 2 - 3 * 9 = -24
1 + 2 - 4 * 7 = -25
1 + 3 - 5 * 6 = -26
1 + 2 - 5 * 6 = -27
1 + 3 - 4 * 8 = -28
1 + 2 - 4 * 8 = -29
1 + 4 - 5 * 7 = -30
1 + 3 - 5 * 7 = -31
1 + 2 - 5 * 7 = -32
1 + 2 - 4 * 9 = -33
2 / 1 - 4 * 9 = -34
1 + 4 - 5 * 8 = -35
1 + 3 - 5 * 8 = -36
1 + 2 - 5 * 8 = -37
1 + 3 - 6 * 7 = -38
1 + 2 - 6 * 7 = -39
1 + 4 - 5 * 9 = -40
1 + 3 - 5 * 9 = -41
1 + 2 - 5 * 9 = -42
1 + 4 - 6 * 8 = -43
1 + 3 - 6 * 8 = -44
1 + 2 - 6 * 8 = -45
1 - 6 * 9 + 7 = -46
2 + 5 - 6 * 9 = -47
1 + 5 - 6 * 9 = -48
1 + 4 - 6 * 9 = -49
1 + 3 - 6 * 9 = -50
1 + 2 - 6 * 9 = -51
1 + 3 - 7 * 8 = -52
1 + 2 - 7 * 8 = -53
1 - 7 * 9 + 8 = -54
2 + 6 - 7 * 9 = -55
1 + 6 - 7 * 9 = -56
1 + 5 - 7 * 9 = -57
1 + 4 - 7 * 9 = -58
1 + 3 - 7 * 9 = -59
1 + 2 - 7 * 9 = -60
2 / 1 - 7 * 9 = -61
3 + 7 - 8 * 9 = -62
2 + 7 - 8 * 9 = -63
1 + 7 - 8 * 9 = -64
1 + 6 - 8 * 9 = -65
1 + 5 - 8 * 9 = -66
1 + 4 - 8 * 9 = -67
1 + 3 - 8 * 9 = -68
1 + 2 - 8 * 9 = -69
2 / 1 - 8 * 9 = -70
None = -71
```

(*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta

Para el caso en el que no hay ninguna restriccion yo entiendo que se podrian combinar absolutamente todos los elementos de cada lista con todos los elementos de la otra sin que no se puedan repetir, sin que tenga que ser de forma alternativa (numero, operador, numero ...) y pudiendo llegar a un string de 9 + 4 caracteres que no tiene porque tener sentido.

Ejemplo:

$/3 + 456 + 6 - +133$

Por lo que estariamos en un caso de combinacion con repeticion. La fórmula de combinación con repetición se utiliza para calcular el número de formas posibles de combinar r elementos tomados de un conjunto de n elementos con repetición. La fórmula se representa como:

$C(n + r - 1, r)$

Siendo en este caso 13 elementos (9 digitos + 4 operadores) y suponiendo que se hacen cadenas de 13 elementos la cantidad de posibilidades es :

$C(13 + 13 - 1, 13) = C(25, 13) = 3, 268, 760$

Caso Permutacion sin repeticion , conseguir las posibilidades de seleccionar 4 elementos de una lista de 9 sin repeticion con la seleccion de 3 operadores de una lista de 4 sin repeticion.

$P(n, r) = (n - r)!/n!$

Codigo abajo:

In [3]:

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

num_digits = 9
num_operators = 4


num_digits_selected = 4

num_operators_selected = 3


possibilities_digits = factorial(num_digits) // factorial(num_digits - num_digits_selected)
possibilities_operators = factorial(num_operators) // factorial(num_operators - num_operators_selected)

total_possibilities = possibilities_digits * possibilities_operators
print(f"El número total de posibilidades de permutación es: {total_possibilities}")
```

```
El número total de posibilidades de permutación es: 72576
```

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)

Respuesta

La estructura de datos que mejor se adapta al problema son las Python Lists ya que nos permiten trabajar con los elementos que estan dentro de este tipo de dato directamente y funciona muy bien con la libreria itertools Desde el primer momento se eligio la lista como tipo de dato.

Según el modelo para el espacio de soluciones
(*)¿Cual es la función objetivo?

(*)¿Es un problema de maximización o minimización?

Respuesta

No estamos buscando optimizar o maximizar algún valor específico, simplemente estamos explorando todas las combinaciones posibles para encontrar aquellas que cumplan la condición de igualar el valor objetivo.

Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

In [4]:

```python
import itertools

def evaluate_expression(expression):

    try:
        return eval(expression)
    except ZeroDivisionError:
        return None

def find_combinations(target):
    digits = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    operators = ['+', '-', '*', '/']
    results = []
    result=''

    for digits_perm in itertools.permutations(digits, 4):
        for ops_perm in itertools.permutations(operators, 3):
            expression = f"{digits_perm[0]} {ops_perm[0]} {digits_perm[1]} {ops_perm[1]} {digits_perm[2]} {ops_perm[2]} {digits_perm[3]}"
            result = evaluate_expression(expression)
            if result is not None and result == target:
                #results.append(expression)
                result=expression
                return result
```

Calcula la complejidad del algoritmo por fuerza bruta

Respuesta

La complejidad de este algoritmo por fuerza bruta es del orden de $O(n!)$ ya que se exploran todas las permutaciones hasta que se encuantra un valor que cumple la condicion.

Este algoritmo es óptimo para valores de entrada pequeños, pero a medida que entra un nuevo elemento u operado la complejidad crece considerablemente.

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

Respuesta

In [5]:

```python
  def evaluate_expression(expression):
    try:
        return eval(expression)
    except ZeroDivisionError:
        return None

def find_combinations(target, digits, operators):
    def backtrack(expression, used_digits, used_operators):
        if len(expression) == 7:
            result = evaluate_expression(expression)
            if result is not None and result == target:
                return expression
            return None

        results = []
        for digit in digits:
            if not used_digits[digit]:
                for op in operators:
                    if not used_operators[op]:
                        if len(expression) == 0 or expression[-1].isdigit() != str(digit).isdigit():
                            new_expression = expression + str(digit) + " " + op + " "
                            used_digits[digit] = True
                            used_operators[op] = True
                            result = backtrack(new_expression, used_digits, used_operators)
                            if result:
                                results.append(result)
                            used_digits[digit] = False
                            used_operators[op] = False

        return results

    used_digits = {digit: False for digit in digits}
    used_operators = {op: False for op in operators}
    return backtrack("", used_digits, used_operators)

target = 42
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9]
operators = ['+', '-', '*', '/']

results = find_combinations(target, digits, operators)
print("Expresiones que dan como resultado el valor deseado:")
for result in results:
    print(result)
```

```
Expresiones que dan como resultado el valor deseado:
```

(*)Calcula la complejidad del algoritmo

Respuesta

Si añadiesemos recursividad a este algoritmo por fuerza bruta la nueva complejidad seria exponencial y no factorial

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

Respuesta

In [6]:

```python
import random

random_digits = [random.randint(1, 99) for _ in range(9)]

random_digits.sort()
print(random_digits)
```

```
[1, 10, 12, 22, 22, 60, 64, 75, 81]
```

Aplica el algoritmo al juego de datos generado

Respuesta

In [7]:

```python
import itertools

def evaluate_expression(expression):

    try:
        return eval(expression)
    except ZeroDivisionError:
        return None

def find_combinations(target, digits):

    operators = ['+', '-', '*', '/']
    results = []
    result=''

    for digits_perm in itertools.permutations(digits, 4):
        for ops_perm in itertools.permutations(operators, 3):
            expression = f"{digits_perm[0]} {ops_perm[0]} {digits_perm[1]} {ops_perm[1]} {digits_perm[2]} {ops_perm[2]} {digits_perm[3]}"
            result = evaluate_expression(expression)
            if result is not None and result == target:
                #results.append(expression)
                result=expression
                return result

    #return results


target_number = 5
combinations = find_combinations(target_number, random_digits)
print(f'{combinations} = {target_number}')
```

```
1 * 10 - 60 / 12 = 5
```

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

ChatGPT uso de la libreria itertools , no sabia utilizarla y es una buena libreria para hacer combinaciones, permutaciones , etc...

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