Algoritmos de optimización - Seminario

Nombre y Apellidos: Javier Canales Navarrete
Github: https://github.com/Javicana/03MAIR-Algoritmos-de-Optimizacon-2021/tree/main/Trabajo_Final
Colab: https://colab.research.google.com/drive/1GtooUJcekEHF-ayqfKRijBzDX7Zl2ImU?usp=sharing

Problema 3: Combinar cifras y operaciones

Descripción:

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

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

Analizando el rango de valores que se puede optener en el problema, la combinación más grande que se puede conseguir es 9x8+7-1/6, que da 78.833 y la mínima sería -70.143 como combinación de 1+6/7-9x8.

Como solo se considerán las soluciones que son numeros enteros, es decir las que se encuentran comprendidas entre [-69,77], ya que por lo que podemos obtener 147 numeros distintos.

En primer lugar, si analizamos las posibles combinaciones de las cifras sin ningún tipo de restricción serían:

1. Para las cifras, hay 9 numeros [0-9] tomados de 5 en 5 --> Variación sin repetición = $VR_9^5 = 9^5 = 59049$ posibilidades
2. Las posibles combinaciones de los signos serían $4^4 = 256$ posibilidades

   Total = 15116544 posibilidades

Acontinuación se muestra un codigo para dichas posibilidades:

```
import itertools

#Combinaciones de cifras
cifras = list(range(9,0,-1))
combinaciones_cifras = [i for i in itertools.product(cifras,repeat=5)]
print('cifras=', cifras, 'cantidad de combinaciones =', (len(combinaciones_cifras)))

#Combinaciones de signos
signos =  ['*','/','+','-']
combinaciones_signos =  [i for i in itertools.product(signos, repeat=4)]
print('signos=',signos, 'cantidad de combinaciones =', (len(combinaciones_signos)))

#Calculod de posibilidades
count = 0
for i in combinaciones_cifras:
  for j in combinaciones_signos:
    count += 1
print('Total = ',count,'posibilidades')
```

```
cifras= [9, 8, 7, 6, 5, 4, 3, 2, 1] cantidad de combinaciones = 59049
signos= ['*', '/', '+', '-'] cantidad de combinaciones = 256
Total =  15116544 posibilidades
```

2. ¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Las posibles combinaciones de las cifras sin ningún tipo de restricción serían:

1. Tenemos 9 numeros [0-9] tomados de 5 en 5 --> Variación sin repetición = $V_9^5 = 9!/(9-5)! = 15120$ posibilidades
2. Las posibles combinaciones de los signos serían $4! = 24$ posibilidades

   Total = 362880 posibilidades

Acontinuación se muestra un codigo para dichas posibilidades:

```
import itertools

#Combinaciones de cifras
cifras = list(range(9,0,-1))
combinaciones_cifras = [i for i in itertools.permutations(cifras,5)]
print('cifras=', cifras, 'cantidad de combinaciones =', (len(combinaciones_cifras)))

#Combinaciones de signos
signos =  ['*','/','+','-']
combinaciones_signos =  [i for i in itertools.permutations(signos,4)]
print('signos=',signos, 'cantidad de combinaciones =', (len(combinaciones_signos)))

#Calculod de posibilidades
count = 0
for i in combinaciones_cifras:
  for j in combinaciones_signos:
    count += 1
print('Total = ',count,'posibilidades')
```

```
cifras= [9, 8, 7, 6, 5, 4, 3, 2, 1] cantidad de combinaciones = 15120
signos= ['*', '/', '+', '-'] cantidad de combinaciones = 24
Total =  362880 posibilidades
```

A estas restricciones de los signos tambien habría que añadir que solo se puede considerar combinación valida aquellas que dan un número entero

3. (*) ¿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)

Para la resolución de este problema, se ha aplicado un algoritmo de busqueda en profundidad, explorando cada nodo hasta el final y descartando la solución hasta optener una primera solución valida momento en el que se detiene la busqueda, para mejorar el rendimiento, se ha dividido en 2 la busqueda, una primera para números iguales o mayores que 0 y otra para numeros menores que 0.

4. (*)¿Cual es la función objetivo?

Se trata de un problema de busqueda de encontrar una solución, por lo el objetivo es encontrar la solución en el menor número de iteraciones posible.

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

Por tanto, se trata de un problema de busqueda con el objetivo de mínimizar el tiempo empleado para encontrarlo, por lo que para ello se ha de mínimizar el número de iteraciones empleadas.

6. Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

```
def find_result(num_objetivo):
  import itertools

  #Combinaciones de cifras

  cifras = list(range(9,0,-1))
  combinaciones_cifras = [i for i in itertools.permutations(cifras,5)]

  #Combinaciones de signos
```

```
  #Combinaciones de signos
  signos =  ['*','/','+','-']
  combinaciones_signos =  [i for i in itertools.permutations(signos)]

  #Algoritmo de busqueda de solución
  iteraciones = 0
  for i in combinaciones_cifras:
    for j in combinaciones_signos:
      iteraciones += 1
      if (eval(f'{i[0]}{j[0]}{i[1]}{j[1]}{i[2]}{j[2]}{i[3]}{j[3]}{i[4]}') == num_objetivo):
        result = f'{i[0]}{j[0]}{i[1]}{j[1]}{i[2]}{j[2]}{i[3]}{j[3]}{i[4]}'
        return iteraciones,result
  return

import random

#Verificación de el algoritmo
num_objetivo = random.randrange(-69,77)
iteraciones, resultado = find_result(num_objetivo)
print(f'Numero buscado = {num_objetivo}')
print(f'Comboración resultado --> {resultado} = {eval(resultado)}')
print(f'Numero de iteraciones empleadas = --> {iteraciones}')
```

```
Numero buscado = -37
Comboración resultado --> 9-8*6+4/2 = -37.0
Numero de iteraciones empleadas = --> 1052
```

```
#Evaluar tiempo promedio de resultados evaluando todos los numeros de rangos disponible [-69,77]
import time

#Rango de todos los valores que se pueden tomar
range_disp = range(-68,78)

#Inicio de temporizador y calculo de todas las soluciones
iteraciones_mean = 0
start = time.time()
for i in range_disp:
  iteraciones,_ = find_result(i)
  iteraciones_mean += iteraciones
end = time.time()
tiempo = end - start

print(f'Tiempo empleado = {tiempo}')
print(f'Tiempo promedio para calcular una solución {tiempo/len(range_disp)} ms')
print(f'Promedio iteraciones empleadas = {iteraciones_mean/len(range_disp)}')
```

```
Tiempo empleado = 17.41455912590027
Tiempo promedio para calcular una solución 0.11927780223219361 ms
Promedio iteraciones empleadas = 14803.191780821919
```

7. Calcula la complejidad del algoritmo por fuerza bruta

Se trata de un algoritmo de complejidad $O(n^2)$, ya que esta compuesto por 2 bucles for.

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

```
def find_result2(num_objetivo):

  #Combinaciones de cifras
  cifras = list(range(9,0,-1))
  combinaciones_cifras = [i for i in itertools.permutations(cifras,5)]

  #Algoritmo de busqueda de solución
  iteraciones = 0
  for i in combinaciones_cifras:
    iteraciones += 1
    if num_objetivo >= 0:
      signos = ['*','-','+','/']

      if (eval(f'{i[0]}{signos[0]}{i[1]}{signos[1]}{i[2]}{signos[2]}{i[3]}{signos[3]}{i[4]}') == num_objetivo):
        result = f'{i[0]}{signos[0]}{i[1]}{signos[1]}{i[2]}{signos[2]}{i[3]}{signos[3]}{i[4]}'
        return iteraciones, result
    else:
```

```
    else:
      signos = ['/','+','-','*']
      if (eval(f'{i[0]}{signos[0]}{i[1]}{signos[1]}{i[2]}{signos[2]}{i[3]}{signos[3]}{i[4]}') == num_objetivo):
        result = f'{i[0]}{signos[0]}{i[1]}{signos[1]}{i[2]}{signos[2]}{i[3]}{signos[3]}{i[4]}'
        return iteraciones, result
  return

#Verificación de el algoritmo
num_objetivo = random.randrange(-69,78)
num_objetivo = -69
iteraciones, resultado = find_result2(num_objetivo)
print(f'Numero buscado = {num_objetivo}')
print(f'Comboración resultado --> {resultado} = {eval(resultado)}')
print(f'Numero de iteraciones empleadas = --> {iteraciones}')
```

```
Numero buscado = -69
Comboración resultado --> 6/3+1-9*8 = -69.0
Numero de iteraciones empleadas = --> 6271
```

```
#Evaluar tiempo promedio de resultados evaluando todos los numeros de rangos disponible [-69,77]
#Evaluar tiempo promedio de resultados evaluando todos los numeros de rangos disponible [-69,77]
import time

#Rango de todos los valores que se pueden tomar
range_disp = range(-68,78)

#Inicio de temporizador y calculo de todas las soluciones
iteraciones_mean = 0
start = time.time()
for i in range_disp:
  iteraciones, _ = find_result2(i)
  iteraciones_mean += iteraciones
end = time.time()
tiempo = end - start

print(f'Tiempo empleado = {tiempo}')
print(f'Tiempo promedio para calcular una solución {tiempo/len(range_disp)} ms')
print(f'Promedio iteraciones empleadas = {iteraciones_mean/len(range_disp)}')
```

```
Tiempo empleado = 1.9814841747283936
Tiempo promedio para calcular una solución 0.013571809415947902 ms
Promedio iteraciones empleadas = 1452.1575342465753
```

Este algoritmo ayuda a optimizar el tiempo de ejecución, esto es debido a que en lugar de explorar todos los nodos, realizamos una división por la mitad del espacio de soluciones de modo que si el número buscado es menor a 0, realizamos un blucle y si es mayor realizamos otro, reduciendo así la complejidad del algoritmo a O(n).

Para lograrlo se ha analizado los resultados optenidos en función de la posición de los signos de operación de modo que para signos = ['','-','+','/'], *optenemos numeros de un rango de [-5,77] y para signos = ['/','+','-',''*], optenemos valores de [-69,11].

9.(*)Calcula la complejidad del algoritmo

Se trata de un algoritmo de complejidad $O(n)$, ya que esta compuesto por 1 bucle for.

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

Para generar datos aleatorios, hemos de generar valores enteros entre [-69,77], de modo que quedaría así:

```
num_objetivo = random.randrange(-69,77)
```

Aplica el algoritmo al juego de datos generado

```
iteraciones, resultado = find_result2(num_objetivo)
print(f'Numero buscado = {num_objetivo}')
print(f'Comboración resultado --> {resultado} = {eval(resultado)}')
print(f'Numero de iteraciones empleadas = --> {iteraciones}')
```

```
Numero buscado = 13
Comboración resultado --> 9*2-8+3/1 = 13.0
Numero de iteraciones empleadas = --> 1285
```

11. 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

Este algoritmo tiene una complejidad lineal O(n), por lo que presenta un rendimiento razonable, no obstante sería interesante analizar otras alternativas que no diesen lugar a explorar los nodos completos para determinar si una solución es válida.

Por otro lado, si quisieramos que esta algoritmo funcionase para un mayor rango de números y no quedase limitado entre [-69,77], deberiamos contemplar la posibilidad de poder repetir números y signos para una mayor variedad de operaciones.