Branch: master ▾ 03MAIR---Algoritmos-de-Optimizacion---2019 / Seminario / WilliamSteveRodriguezVillamizar_Seminario-2.ipynb

Find file Copy path

wisrovi Creado con Colaboratory 597aa06 1 minute ago

1 contributor

887 lines (887 sloc) | 27.1 KB

Raw Blame History

Open in Colab

Algoritmos de optimización - Seminario

Nombre y Apellidos:
Github: https://github.com/wisrovi/03MAIR---Algoritmos-de-Optimizacion---2019/blob/master/Seminario/WilliamSteveRodriguezVillamizar_Seminario-2.ipynb
Problema:

Colab: https://colab.research.google.com/github/wisrovi/03MAIR---Algoritmos-de-Optimizacion---2019/blob/master/Seminario/WilliamSteveRodriguezVillamizar_Seminario-2.ipynb

> 1. Combinar cifras y operaciones

Descripción del problema:

Elección de grupos de población homogéneos

• 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.

....

(*) La respuesta es obligatoria

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

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta

Debido a que el algoritmo genera en diversas ocasiones soluciones repetidas y no todas las combinaciones de numeros y

signos son posibles, además de que cada valor ingresado tiene su própio número de soluciones único, siendo más facil generar soluciones para valores enteros que para valores flotantes.

No fue posible determinar una ecuación que entregara la cantidad de soluciones absoluta para todos los valores ingresados.

Por ello se ha elaborado un algoritmo para buscar toda la cantidad de respuestas posibles para un número dado, para determinar cuantas soluciones se pueden generar para la respuesta solicituda (se verá más adelante en el código).

En la evaluacion del algoritmo se ha logrado generar hasta 700 soluciones posibles para la respuesta de un entero de valor 10 (siendo este valor el que mayor soluciones se ha logrado encontrar.

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

Como este problema no tiene un dataseet inicial sino un dato de ingreso y con este generar una solución al problema, sólo se puede decir que hay mejores soluciones para datos de entrada de tipo int menores a 100 que para datos float de cualquier valor

Esto se evidencia debido a que tratando de evaluar el mayor numero de soluciones para un dato, se ha logrado minimo 250 soluciones diferentes para datos int mientras que para datos float sólo se han logrado 35 soluciones

Sin embargo si se pudiere usar una maquina con mayor GPU a la mia, la cantidad de soluciones debería aumentar de 35 a 100 y de 250 a 1000 para float e int respectivamente.

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

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

Respuesta: Minimizacion

Implementación en Pícipe de código para fuerza bruta

Implementaciones Basicas de codigo para fuerza bruta y algoritmo final

In [0]:
```
import random
from sympy.parsing.sympy_parser import parse_expr
import numpy as np
```

In [0]:
```
Diccionario_signos = {
    "suma" : "+",
    "resta" : "-",
    "multiplicacion" : "*",
    "division" : "/"
    }

Lista_Numeros_Posibles = (1,
           2,
           3,
           4,
           5,
           6,
           7,
           8,
           9
          )
```

In [0]:
```
def BuscarExiste(lista, item):
    buscandoEncontrando = False
    for i in lista:
        if i == item:
            buscandoEncontrando = True
            break
    return buscandoEncontrando

def getNumeroAleatorio(listaNumeros = []):
    """
    usamos recursividad para crear este set de datos
    """
    if len(listaNumeros) < 5:
```

```
        numer = random.choice(Lista_Numeros_Posibles)
        if not BuscarExiste(listaNumeros, numer):
            listaNumeros.append(numer)
        getNumeroAleatorio(listaNumeros)
    return tuple(listaNumeros)

def getSignosAleatorios(listaSignos = []):
    """
    usamos recursividad para crear este set de datos
    """
    if len(listaSignos) < 4:
        sign = random.choice( list(Diccionario_signos.keys()) )
        if not BuscarExiste(listaSignos, sign):
            listaSignos.append(sign)
        getSignosAleatorios(listaSignos)
    return tuple(listaSignos)

def HallarResultado(tuplaNumeros, tuplaSignos):
    expresion = """%s %s %s %s %s %s %s %s %s""" %(
        str(tuplaNumeros[0]),
        Diccionario_signos[tuplaSignos[0]],
        str(tuplaNumeros[1]),
        Diccionario_signos[tuplaSignos[1]],
        str(tuplaNumeros[2]),
        Diccionario_signos[tuplaSignos[2]],
        str(tuplaNumeros[3]),
        Diccionario_signos[tuplaSignos[3]],
        str(tuplaNumeros[4])
    )
    solucion = float(parse_expr(expresion))
    solucion = "{0:.2f}".format(solucion)
    solucion = float(solucion)
    return (expresion, solucion)

def HallarValorDeseado(valorDeseado, stepsMax=10000):
    for i in range(stepsMax):
        (expresion, solucion) = HallarResultado(getNumeroAleatorio([]), getSignosAleatorio
s([]))
        if valorDeseado == solucion:
            return (expresion, "=", valorDeseado)
```

```
def BuscarOperacionesPorListado(valoresDeseados):
    respuestas = []
    numerosNoEncontrados = []
    for valorDeseado in valoresDeseados:
        rta = HallarValorDeseado(valorDeseado)
        if rta is None:
            numerosNoEncontrados.append(valorDeseado)
            #print("None = ", valorDeseado )
        else:
            respuestas.append(rta)
            #print(rta)
    return respuestas, numerosNoEncontrados
```

Algoritmo Por fuerza bruta

Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

In [31]:
```
#Buscando la cantidad máxima de soluciones posibles
valor = 5.25
cantidadSolucionesBuscar = 500 #Número probado de soluciones diferentes encontradas 500 pa
ra int y 150 float


valoresDeseados = []
for i in range(cantidadSolucionesBuscar):
    valoresDeseados.append(valor)


print("Total numeros buscar respuesta: ", len(valoresDeseados))
respuestas, numerosNoEncontrados  =BuscarOperacionesPorListado(valoresDeseados)

print("Numeros no encontrados: ")
for rta in numerosNoEncontrados:
    print(rta)
    pass
```

```
respuestas = sorted(set(respuestas))
print("Numeros con respuesta encontrada: ", len(respuestas) )
for rta in respuestas:
    #print(rta)
    pass
```

```
Total numeros buscar respuesta:  500
Numeros no encontrados:
Numeros con respuesta encontrada:  337
```

In [38]:
```
print("solución a hallar: ", valor)
print()
print("solución mejor (usando los numero más pequeños posibles): " )
print(respuestas[0][0], respuestas[0][1], respuestas[0][2])
```

```
solución a hallar:  5.25

solución mejor (usando los numero más pequeños posibles):
1 * 2 + 4 - 6 / 8 = 5.25
```

Calcula la complejidad del algoritmo por fuerza bruta

Respuesta

O(n log n)

Algoritmo final (mejora al de fuerza bruta)

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

Respuesta

In [40]:
```
valoresDeseados = [-3.5]

print("Total numeros buscar respuesta: ", len(valoresDeseados))
respuestas, numerosNoEncontrados  =BuscarOperacionesPorListado(valoresDeseados)
```

```
print("solución a hallar: ", valoresDeseados[0])
print()
print("solución mejor (usando los numero más pequeños posibles): " )
print(respuestas[0][0], respuestas[0][1], respuestas[0][2])
```

```
Total numeros buscar respuesta:  1
solución a hallar:  -3.5

solución mejor (usando los numero más pequeños posibles):
2 - 7 + 4 / 8 * 3 = -3.5
```

In [41]:

```
valoresDeseados = [-3.5, 2, 9, -9.5, 4, 1.3, 1,2,3,4,5,6,7,8,9]

print("Total numeros buscar respuesta: ", len(valoresDeseados))
respuestas, numerosNoEncontrados  =BuscarOperacionesPorListado(valoresDeseados)

print("Numeros no encontrados: ")
for rta in numerosNoEncontrados:
    print(rta)
    pass

print("Numeros con respuesta encontrada: ")
for rta in respuestas:
    print(rta)
    pass
```

```
Total numeros buscar respuesta:  15
Numeros no encontrados:
1.3
Numeros con respuesta encontrada:
('3 * 5 / 6 - 8 + 2', '=', -3.5)
('4 / 8 * 2 + 7 - 6', '=', 2)
('9 + 6 - 4 / 2 * 3', '=', 9)
('2 / 4 + 8 - 6 * 3', '=', -9.5)
('7 - 3 * 4 + 9 / 1', '=', 4)
('6 - 8 + 1 / 3 * 9', '=', 1)
('5 + 2 * 3 / 6 - 4', '=', 2)
('1 - 4 * 6 / 8 + 5', '=', 3)
('4 / 2 - 7 + 1 * 9', '=', 4)
('2 + 5 / 4 * 8 - 7', '=', 5)
```

```
('8 - 4 + 3 / 9 * 6', '=', 6)
('6 / 2 - 8 + 3 * 4', '=', 7)
('8 * 3 / 6 - 1 + 5', '=', 8)
('9 + 6 * 4 / 8 - 3', '=', 9)
```

In [42]:

```
valoresDeseados = range (-10, 11, 1)

print("Total numeros buscar respuesta: ", len(valoresDeseados))
respuestas, numerosNoEncontrados  =BuscarOperacionesPorListado(valoresDeseados)

print("Numeros no encontrados: ")
for rta in numerosNoEncontrados:
    print(rta)
    pass

respuestas = sorted(set(respuestas))
print("Numeros con respuesta encontrada: ")
for rta in respuestas:
    print(rta)
    pass
```

```
Total numeros buscar respuesta:  21
Numeros no encontrados:
Numeros con respuesta encontrada:
('1 - 5 + 6 / 4 * 8', '=', 8)
('1 - 8 / 2 * 4 + 6', '=', -9)
('2 - 7 / 1 + 4 * 3', '=', 7)
('2 - 8 / 4 + 5 * 1', '=', 5)
('3 + 5 - 6 / 4 * 8', '=', -4)
('3 - 5 / 2 * 4 + 6', '=', -1)
('3 - 9 + 2 / 4 * 6', '=', -3)
('4 + 8 / 1 - 6 * 2', '=', 0)
('4 - 9 + 3 / 2 * 6', '=', 4)
('5 + 1 - 4 * 9 / 3', '=', -6)
('5 + 4 * 9 / 6 - 2', '=', 9)
('5 - 4 / 2 * 7 + 1', '=', -8)
('5 - 6 + 8 * 1 / 4', '=', 1)
('7 + 1 / 2 * 6 - 4', '=', 6)
('7 + 2 / 4 * 6 - 8', '=', 2)
('7 + 4 - 9 * 6 / 3', '=', -7)
```

```
('8 - 7 / 1 * 2 + 4', '=', -2)
('8 / 4 + 5 * 3 - 7', '=', 10)
('9 + 2 / 1 - 3 * 7', '=', -10)
('9 - 3 * 6 + 8 / 2', '=', -5)
('9 - 4 / 2 * 7 + 8', '=', 3)
```

In [0]:

```
valoresDeseados = []
valores = np.arange(-10, 11, 0.05)
for valorDeseado in valores:
    valorDeseado = float("{0:.2f}".format(valorDeseado))
    valoresDeseados.append(valorDeseado)

print("Total numeros buscar respuesta: ", len(valoresDeseados))
respuestas, numerosNoEncontrados  =BuscarOperacionesPorListado(valoresDeseados)
print("Con rta: ",len(respuestas))
print("Sin rta: ",len(numerosNoEncontrados))



print("Numeros no encontrados: ")
for rta in numerosNoEncontrados:
    print(rta)
    pass

print("Numeros con respuesta encontrada: ")
for rta in respuestas:
    print(rta)
    pass
```

```
Total numeros buscar respuesta:  420
```

Este algoritmo mejora al algoritmo de fuerza bruta debido a que al no tener que generar muchas soluciones para la misma entrada para luego elegir una, sino que se elije la qprimera que cumpla con lo solicitado, logrando así respuestas más rápidas.

(*)Calcula la complejidad del algoritmo

Respuesta

O(n^2)

O(n^2)

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

In [61]:
```
sizeDataseet = 20


valoresDeseados = []
for valorDeseado in range(sizeDataseet):
    valorAzar = random.random()*10
    valorAzar = float("{0:.2f}".format(valorAzar))
    valoresDeseados.append(valorAzar)

print(valoresDeseados)
```

```
[2.3, 9.82, 2.15, 8.72, 3.18, 4.42, 4.31, 2.75, 4.76, 1.19, 0.51, 7.58, 9.02, 2.12, 0.22,
6.34, 1.52, 8.96, 8.13, 5.09]
```

Aplica el algoritmo al juego de datos generado

In [62]:
```
print("Total numeros buscar respuesta: ", len(valoresDeseados))
respuestas, numerosNoEncontrados  =BuscarOperacionesPorListado(valoresDeseados)
print("Con rta: ",len(respuestas))
print("Sin rta: ",len(numerosNoEncontrados))



print("Numeros no encontrados: ")
for rta in numerosNoEncontrados:
    print(rta)
    pass

print("Numeros con respuesta encontrada: ")
for rta in respuestas:
    print(rta)
    pass
```

```
Total numeros buscar respuesta:  20
Con rta:  3
Sin rta:  17
Numeros no encontrados:
```

```
Numeros no encontrados:
2.3
9.82
2.15
8.72
3.18
4.42
4.31
4.76
1.19
0.51
7.58
9.02
6.34
1.52
8.96
8.13
5.09
Numeros con respuesta encontrada:
('2 / 8 * 3 + 9 - 7', '=', 2.75)
('4 - 3 + 9 * 1 / 8', '=', 2.12)
('1 - 3 + 5 / 9 * 4', '=', 0.22)
```

Conclusiones

Para este problema se ha usado recursividad y el algoritmo de divide y venceras para la solución del problema, pero usando algoritmos heuristicos seguro se podrá lograr obtener los mismos resultados en un tiempo menor

Las restricciones del problema hacen que el algoritmo le resulte dificil generar soluciones para valores float no multiplos de 0.25....