



Why GitHub? ▾ Enterprise Explore ▾ Marketplace Pricing ▾

Search



Sign in

Sign up

vcebrian / 03MAIR---Algoritmos-de-optimizacion

Watch

1

★ Star

0

Fork

0

<> Code

Issues 0

Pull requests 0

Projects 0

Insights

Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

Branch: master ▾

03MAIR---Algoritmos-de-optimizacion / SEMINARIO / Seminario.ipynb

Find file

Copy path

vcebrian Creado con Colaboratory

62dd225 a minute ago

1 contributor

523 lines (523 sloc) | 18.4 KB



Raw

Blame

History



# Algoritmos de optimización - Seminario

Nombre y Apellidos: Víctor Cebrián Roselló Url: <https://github.com/vcebrian/03MAIR---Algoritmos-de-optimizacion/tree/master/SEMINARIO>

Problema:

## 3. Combinar cifras y operaciones

Descripción del problema: Disponemos de las 9 cifras del 1 al 9 (excluimos el cero) y de los cuatro signos básicos de las operaciones fundamentales: suma(+), resta(-), multiplicación(\*) y división(/). Debemos combinarlos alternativamente sin repetir ninguno de ellos para obtener una cantidad dada.

## Preguntas

### ¿Cuántas posibilidades hay sin tener en cuenta las restricciones?

Si no tuviéramos en cuenta las restricciones y pudiéramos repetir cualquier dígito y cualquier letra en cada combinación el número de combinaciones posibles sería:

$$9^5 * 4^4 = 15.116.544 \text{ combinaciones.}$$

### ¿Cuántas posibilidades hay teniendo en cuenta todas las restricciones?

En este caso no podemos repetir ni dígitos ni símbolos, por lo que para el primer dígito tendremos 9 posibilidades, para el segundo 8, para el tercero 7, y así sucesivamente. Para los símbolos tendremos 4 posibilidades, luego 3, luego 2 y finalmente solo una. De este modo, las posibles combinaciones son:

$$(9*8*7*6*5)*(4*3*2*1) = 362.880 \text{ combinaciones.}$$

### ¿Cual es la estructura de datos que mejor se adapta al problema?

Podemos añadir los dígitos a una lista y los símbolos a otra, de esta forma podemos ir recorriendo las listas y creando sublistas con los dígitos y símbolos ya utilizados para así ir evaluando todas las posibles combinaciones.

### ¿Cual es la función objetivo?

El objetivo es obtener el valor entero proporcionado a la función con una combinación de los dígitos y símbolos con las restricciones enunciadas.

### ¿Es un problema de maximización o minimización?

No se trata de un problema en el que haya que maximizar o minimizar ninguna función, sino de encontrar una solución válida (si existe) de entre las posibles soluciones.

### Diseña un algoritmo para resolver el problema por fuerza bruta

```
In [0]: #Decorador para calcular tiempos de ejecución
from time import time

#Función para calcular el tiempo de ejecución
def calcular_tiempo(f):
    def wrapper(*args, **kwargs):
        inicio = time()
        resultado = f(*args, **kwargs)
        tiempo = time() - inicio
        print("Tiempo de ejecución para algoritmo: "+str(tiempo))
        return resultado

    return wrapper
```

```
In [22]: #Algoritmo de fuerza bruta
@calcular_tiempo
def combina_cifras_y_operaciones(valor):
    #me creo listas con los símbolos y dígitos
    simbolos = ['*', '+', '-', '/']
    valores = [1,2,3,4,5,6,7,8,9]
```

```

#Una serie de bucles anidados van recorriendo las listas y sublistas con los elementos ya se
leccionados para evaluar la expresión
for d1 in range(1,10):
    for s1 in simbolos:
        for d2 in [x for x in valores if x != d1]:
            for s2 in [x for x in simbolos if x != s1]:
                for d3 in [x for x in valores if x != d1 and x != d2]:
                    for s3 in [x for x in simbolos if x != s1 and x != s2]:
                        for d4 in [x for x in valores if x != d1 and x != d2 and x != d3]:
                            for s4 in [x for x in simbolos if x != s1 and x != s2 and x != s3]:
                                for d5 in [x for x in valores if x != d1 and x != d2 and x != d3 and x != d
4]:
                                    #me creo cadena con la combinación de digitos y símbolos
                                    cadena = str(d1)+s1+str(d2)+s2+str(d3)+s3+str(d4)+s4+str(d5)
                                    #Evaluo la expresion y si es igual al valor de entreda se imprime y termi
namos la ejecución
                                    if (eval(cadena)==valor):
                                        print("----Fuenza bruta----")
                                        print(cadena,"=",str(valor))
                                        print("-----")
                                        return

#En caso de no haber encontrado ninguna solución se imprime que no hay solución
print("----Fuenza bruta----")
print('No hay solucion para',str(valor))
print("-----")

combina_cifras_y_operaciones(34)

```

----Fuenza bruta----

1+4\*9-6/2 = 34

-----

Tiempo de ejecución para algoritmo: 0.13257956504821777

### Calcula la complejidad del algoritmo por fuerza bruta

Si consideramos N el numero de digitos y consideramos los simbolos siempre constantes, es decir 4. El número de operaciones elementales que se realizan son:

elementales que se realizan son:

6 sumas + 1 asignación + 1 comparación.

Orden =  $7 \times N \times (N-1) \times (N-2) \times (N-3) \times (N-4) \times 4 \times 3 \times 2 \times 1$

Tendremos un orden polinomial.

**Diseña un algoritmo que mejore la complejidad del algoritmo por fuerza bruta. Argumenta por que crees que es mejor que el algoritmo por fuerza bruta.**

Voy a diseñar un algoritmo heurístico que de forma aleatoria busca posibles soluciones

```
In [4]: import random

def solucion_aleatoria():

    simbolos = ['*', '+', '-', '/']
    valores = [1,2,3,4,5,6,7,8,9]
    solucion = ''

    for i in range(9):
        if i%2==0:
            valor = valores[random.randint(0,len(valores)-1)]
            solucion=solucion+str(valor)
            valores.remove(valor)
        else:
            simbol = simbolos[random.randint(0,len(simbolos)-1)]
            solucion=solucion+simbol
            simbolos.remove(simbol)

    return solucion

solucion_aleatoria()
```

```
Out[4]: '7*4+8/1-3'
```

```
In [13]: @calcular_tiempo
def busqueda_heuristica(N, valor):
    for i in range(N):
```

```

for i in range(10):
    sol = solucion_aleatoria()
    if eval(sol)==valor:
        print("----Heurístico----")
        print(sol,"=",str(valor))
        print("-----")
        return

print("----Heurístico----")
print('Solucion no encontrada para',str(valor))
print("-----")

import timeit

%timeit
busqueda_heuristica(1000,11)

```

```

----Heurístico----
7+6-8*1/4 = 11
-----
Tiempo de ejecución para algoritmo: 0.00720977783203125

```

### Calcula la complejidad del algoritmo

El algoritmo heurístico tiene una complejidad  $O(1)$ , es decir, es independiente de  $N$  ya que por mas dígitos que tengamos las operaciones para calcular la solución aleatoria son las mismas.

### Según el problema, diseña un juego de datos de entrada aleatorios

Generaré 10 valores aleatorios para aplicarlos a cada uno de los algoritmos (fuerza bruta y heurístico) para comprobar la mejora aportada por el heurístico.

```

In [0]: #vamos a generar un conjunto de datos de entrada y aplicar el algoritmo para comparar resultado
s
valores = [(random.randrange(-100,100)) for _ in range(10)]

```

### Aplicar algoritmo a los datos generados

```

In [161]: for i in valores:

```

```
11 [10]. for i in valores:
```

```
    combina_cifras_y_operaciones(i)
    busqueda_heuristica(5000,i)
```

```
----Fuenza bruta----
```

```
2+5-6*9/1 = -47
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.5064129829406738
```

```
----Heurístico----
```

```
3/1+4-6*9 = -47
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.011141538619995117
```

```
----Fuenza bruta----
```

```
No hay solucion para -100
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 3.2911157608032227
```

```
----Heurístico----
```

```
Solucion no encontrada para -100
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.14031434059143066
```

```
----Fuenza bruta----
```

```
1+7*8-6/2 = 54
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.15699243545532227
```

```
----Heurístico----
```

```
8*7/1+2-4 = 54
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.045009613037109375
```

```
----Fuenza bruta----
```

```
1*2+4-6/3 = 4
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.0008981227874755859
```

```
----Heurístico----
```

```
2-1*9/3+5 = 4
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.023300886154174805
```

```
----Fuenza bruta----
```

```
1+4*9-6/3 = 35
```

```
-----
```

```
Tiempo de ejecución para algoritmo: 0.11865520477294922
```

```
----Heurístico----
```

```

5*6-9/3+8 = 35
-----
Tiempo de ejecución para algoritmo: 0.022387981414794922
----Fuerza bruta----
1+9/3-6*7 = -38
-----
Tiempo de ejecución para algoritmo: 0.1832106113433838
----Heurístico----
Solucion no encontrada para -38
-----
Tiempo de ejecución para algoritmo: 0.14667010307312012
----Fuerza bruta----
No hay solucion para -79
-----
Tiempo de ejecución para algoritmo: 3.302333354949951
----Heurístico----
Solucion no encontrada para -79
-----
Tiempo de ejecución para algoritmo: 0.14523863792419434
----Fuerza bruta----
1+2*9-6/3 = 17
-----
Tiempo de ejecución para algoritmo: 0.10049891471862793
----Heurístico----
9/4*8-2+1 = 17
-----
Tiempo de ejecución para algoritmo: 0.0014672279357910156
----Fuerza bruta----
1*2+3-8/4 = 3
-----
Tiempo de ejecución para algoritmo: 0.0007131099700927734
----Heurístico----
8/4-1*6+7 = 3
-----
Tiempo de ejecución para algoritmo: 0.007016658782958984
----Fuerza bruta----
No hay solucion para -86
-----
Tiempo de ejecución para algoritmo: 3.2873077392578125
----Heurístico----

```



.....  
Solucion no encontrada para -86

-----  
Tiempo de ejecución para algoritmo: 0.1416318416595459

Podemos ver que el algoritmo Eurístico con valores de N apropiados encuentra una solución con tiempos considerablemente mejores que el algoritmo for fuerza bruta.

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

Es posible incrementar la complejidad del problema añadiendo mayor numero de digitos en el conjunto de entrada, en lugar de 1 a 9 podemos incrementar el problema por ejemplo con cifras de 1 a 19 o incluso mayores.

Además tambien es posible hacerlo modificando las restricciones en cuanto a repeticiones de caracteres y simbolos.

En ambos casos las posibles combinaciones se incrementan y el algoritmo por fuerza bruta puede ser inviable debido al gran número de combinaciones posibles.

