Branch: master ▾ 03MAIR-Algoritmos-de-Optimizacion-2019 / Seminario / Seminario(plantilla)_GMG.ipynb

Find file Copy path

GonzMG Terminado Seminario 338d966 2 minutes ago

1 contributor

509 lines (509 sloc) | 15.8 KB

Raw Blame History

Open in Colab

Algoritmos de optimización - Seminario

Nombre y Apellidos: Gonzalo Molina Gallego
Url: https://github.com/GonzMG/03MAIR-Algoritmos-de-Optimizacion-2019/tree/master/Seminario
Problema:

> 1. Combinar cifras y operaciones

Descripción del problema:
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.

Pregunta 1
a ¿Cuantas posibilidades hay sin tener en cuenta las restricciones?
b ¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta
1a: Teniendo en cuenta las retricciones (no se admiten repetidos de dígitos y operadores), esta pregunta puede ser contestada utilizando la definición de Muestreo Ordenado sin Remplazo: *nPk = (n!) / (n-k)!* siendo *n* la longitud de la lista (9 en el caso de los numeros) y *k* la cantidad de numeros que hay que escoger. Esta metrica se utiliza para calcular el número de formas, sin tener en cuenta el orden, en que los objetos k pueden ser elegidos entre n objetos.
Posibilidades de números:
9P5= 15120 -->Cantidad de posibilidades para los números
4! = 24 --> Cantidad de posibilidades para los signos
Combinando ambas obtenemos: *nPk · len(operadores)!* = **362880**.
1b: Al no tener restricciones, las posibilidades aumentan. Podemos repetir el operador que haga falta y el número que necesitemos.
9^5: 59049 --> Posibilidades para los números.
4^4: 256 --> Posibilidades para los operadores
Total de posibilidades = **15116544**.

In [28]:
```
import math

def restricciones(x,y):
  div1 = math.factorial(x)
  div2 = math.factorial(x-y)
  return div1/div2


print(restricciones(9,5))
```

```
15120.0
1287.0
```

Pregunta 2

Modelo para el espacio de soluciones

a ¿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)

In [0]:
```
N = ["1","2","3","4","5","6","7","8","9"]
O = ["+","-","*","/"]
```

2a: La mejor forma de estructurar los datos de este problema es mediante dos listas que contienen **carácteres**. Es muy importante que la combinación de ambas listas forme un string para su evaluación.

El orden de la lista de operadores siempre será ese, ya que no tiene importancia donde se coloquen, mientras los números estén bien colocados. En el algoritmo de fuerza bruta, la única restricción que se aplica es que el número no esté repetido en la lista (y que no sea una solución repetida), pero en futuras mejoras, se utilizarán diferentes listas pivote para mejorar este algoritmo.

Pregunta 3

Según el modelo para el espacio de soluciones

a ¿Cual es la función objetivo?

b ¿Es un problema de maximización o minimización?

Respuesta

In [0]:
```
def es_correcto(exp, value):
  if len(expr) == 9  and eval(expr) == value: return True
```

```
  else: return False
```

3a: La función objetivo en este problema es la que pasando como parámetro de entrada una cadena de texto (expresión de la operación) devuelve True si el resultado de esa operación es el número deseado y False si no lo es.
3b: Este problema no se trata de maximización o minimización, hay que realizar la búsqueda en un espacio de soluciones de una expresión que su resultado sea un valor deseado. Por lo que no se trata de un problema de optimización.

Pregunta 4
Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

In [0]:
```
from itertools import permutations

def calcular(l): #a partir de una lista de numeros, devuelve la expresion
    expr = ""
    for i in range(len(l)):
        expr += str(l[i])
        if i < len(l) - 1:
            expr += O[i]
    return expr

def space(n):
    return list(permutations(range(1,n+1),5))

def fuerza_bruta(val):
    pos = space(9) # posibles soluciones para 1..9 numeros
    for p in pos:
        expr = calcular(p)
        if eval(expr) == val:
            return expr
    return "Err"
```

In [38]:
```
fuerza_bruta(4)
```

Out[38]:
```
'1+4-2*3/6'
```

a Calcula la complejidad del algoritmo por fuerza bruta

4a: Mi solución para la fuerza bruta es generar todo el conjunto de soluciones posibles y luego iterar sobre este conjunto para encontrar la expresión adecuada. Este enfoque siempre devolverá la solución (si está dentro del rango de soluciones posibles). Al generar todas las cominaciones de números posibles, el peor caso de la complejidad de este algoritmo sería **O(nPk)**.

Pregunta 6
a: Diseña un algoritmo que mejore la complejidad del algortimo por fuerza bruta. Argumenta porque crees que mejora el algoritmo por fuerza bruta

Respuesta: He planteado resolver este algoritmo mediante la técnica del A*, ajustado a este problema. Utilizando como heurística la función de evaluar una expresión aritmética

In [0]:

b: Calcula la complejidad del algoritmo

Respuesta

In [0]:

Pregunta 7
a Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios
b Aplica el algoritmo al juego de datos generado

Respuesta
7ab: Para este problema no tiene sentido imaginar una conjunto de datos de entrada aleatorios, ya que el unico parámetro de entrada que introducimos a este algoritmo es el valor que queremos conseguir alcanzar. Una implementación posible a esto, es generar un número decimal para ver como se comporta el algoritmo:

In [48]:
```
import random

rnumber = random.uniform(0.0,77.0)
rnumber
```

Out[48]: 53.14881018711584

In [49]:
```
fuerza_bruta(rnumber)
```

Out[49]:
```
'Err'
```

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

- https://www.geeksforgeeks.org/program-calculate-value-ncr/
- https://www.probabilitycourse.com/chapter2/2_1_4_unordered_with_replacement.php
- https://en.wikipedia.org/wiki/Reverse_Polish_notation

Pregunta 8
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:

1. Como hemos podido observar, los numerós de tipo *float* son muy complicados que aparezcan en el espacio de soluciones actual. Una forma de avanzar en este problema, sería la de añadir más posibles soluciones al espacio. Esto se podría realizar añadiendo más tipos de operadores como la raíz cuadrada, logaritmos, etc.
2. Modificando la modelación del problema y convirtiendolo a un algoritmo de optimización, en que se necesite minimizar el error entre el valor deseado y el más cercano calculado.
3. Al hacer este cambio, se podría añadir un algoritmo genético de manera más eficiente, ya que por como está planteado el problema, es más abordable desde el punto de vista de ramificación y poda.