



lf Lauren / 03MAIR-Algoritmos-de-Optimizacion-2019

Watch

0

★ 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 ▾

Find file

Copy path

03MAIR-Algoritmos-de-Optimizacion-2019 / SEMINARIO / Luis Fauré Navarro.ipynb - SEMINARIO1

lf Lauren Add files via upload

405b6a3 a minute ago

1 contributor

561 lines (560 sloc) | 21.2 KB

Raw

Blame

History



```
1 {
2   "cells": [
3     {
```

```

4  "cell_type": "markdown",
5  "metadata": {
6    "colab_type": "text",
7    "id": "view-in-github"
8  },
9  "source": [
10   "<a href=\"https://colab.research.google.com/github/raul27868/03MAIR---Algoritmos-de-Optimizacion--2019/blob/master/SEMINAR
11  ]
12  },
13  {
14    "cell_type": "markdown",
15    "metadata": {
16      "colab_type": "text",
17      "id": "hVbXYX-RfPWh"
18    },
19    "source": [
20      "# Algoritmos de optimización - Seminario<br>\n",
21      "Nombre y Apellidos: Luis Fauré Navarro<br>\n",
22      "Url: https://github.com/lfauren/03MAIR-Algoritmos-de-Optimizacion-2019/tree/master/SEMINARIO<br>\n",
23      "Problema:\n",
24      ">1. Elección de grupos de población homogéneos <br>\n",
25      ">2. Organizar los horarios de partidos de La Liga<br>\n",
26      ">3. Combinar cifras y operaciones\n",
27      "\n",
28      "Descripción del problema:(copiar enunciado)\n",
29      "\n",
30      "3. Combinar cifras y operaciones. El problema consiste en analizar el siguiente problema y diseñar un algoritmo que lo resu
31      "Debe analizarse el problema para encontrar todos los valores enteros posibles planteando las siguientes cuestiones:\n",
32      "#### ¿Qué valor máximo y mínimo a priori se pueden obtener según las condiciones?\n",
33      "Aplicando lógica el Mínimo =  $4/2+1-9*8 = -69$  y el Máximo =  $9*8/1+7-2 = 77$ <br>\n",
34      "#### ¿Es posible encontrar todos los valores enteros posibles entre dicho mínimo y máximo?\n",
35      "A priori no se puede saber, porque uno podría hacer la resta del máximo con el mínimo, pero puede que se salten, además hay
36      "Usando el programa sale el Mínimo = -69 y el Máximo = 77, efectivamente. La cantidad de valores entre el intervalo [-69, 77

```

```

37     "\n",
38     "(*) La respuesta es obligatoria\n",
39     "\n",
40     "\n",
41     "\n",
42     "\n",
43     "\n",
44     "
45 ]
46 },
47 {
48     "cell_type": "markdown",
49     "metadata": {
50         "colab_type": "text",
51         "id": "3_-exlrTgLD-"
52     },
53     "source": [
54         "(*)¿Cuántas posibilidades hay sin tener en cuenta las restricciones?<br>\n",
55         "\n",
56         "(n!/(n-r)!)*m! Siendo n=dígitos, r=cuántos dígitos se van a considerar, m=operaciones<br>\n",
57         "(9!/(9-5)!)*4! = 362880<br>\n",
58         "\n",
59         "¿Cuántas posibilidades hay teniendo en cuenta todas las restricciones.<br>\n",
60         "\n",
61         "Hay dos casos los valores flotantes y los valores enteros. Lo que había que descartar eran los valores flotantes y cabe sef
62     ]
63 },
64 {
65     "cell_type": "code",
66     "execution_count": 1,
67     "metadata": {
68         "colab": {},
69         "colab_type": "code",

```

```
70     "id": "iq6Fe32MgpDX"
71 },
72 "outputs": [
73   {
74     "name": "stdout",
75     "output_type": "stream",
76     "text": [
77       "362880.0\n"
78     ]
79   }
80 ],
81 "source": [
82   "def factorial(n):\n",
83   "    if n <1:\n",
84   "        return 1\n",
85   "    else:\n",
86   "        return n*factorial(n-1)\n",
87   "\n",
88   "num_posibilidades = (factorial(9)/factorial(9-5))*factorial(4)\n",
89   "print (num_posibilidades)"
90 ]
91 },
92 {
93   "cell_type": "markdown",
94   "metadata": {
95     "colab_type": "text",
96     "id": "dzynfVJahHoY"
97   },
98   "source": [
99     "Modelo para el espacio de soluciones<br>\n",
100     "(*) ¿Cual es la estructura de datos que mejor se adapta al problema? Argumentalo.(Es posible que hayas elegido una al princ
101   ]
102 },
```

```

103 {
104   "cell_type": "markdown",
105   "metadata": {
106     "colab_type": "text",
107     "id": "xFIJrOwKi2Ti"
108   },
109   "source": [
110     "Las mejores estructuras de datos en este caso fueron las listas de string para luego convertir con la función eval a una ex
111   ]
112 },
113 {
114   "cell_type": "markdown",
115   "metadata": {
116     "colab_type": "text",
117     "id": "7Z9U2W7bi-XS"
118   },
119   "source": [
120     "Según el modelo para el espacio de soluciones<br>\n",
121     "(*)¿Cual es la función objetivo?\n",
122     "\n",
123     "expresion = eval(i[0]+j[0]+i[1]+j[1]+i[2]+j[2]+i[3]+j[3]+i[4])\n",
124     "Donde los \"i\" son números y los \"j\" son operaciones aritméticas, i[xyznm] donde xyznm toma 5 dígitos del 1 al 9 y j[abc
125     "\n",
126     "(*)¿Es un problema de maximización o minimización?\n",
127     "\n",
128     "Es un problema de minimización, porque se buscan los valores enteros desde el menor -69 al mayor 77, osea, 147 valores de u
129   ]
130 },
131 {
132   "cell_type": "markdown",
133   "metadata": {
134     "colab_type": "text",
135     "id": "rlnTYgY1j6I2"

```

```
136     },
137     "source": [
138         "Diseña un algoritmo para resolver el problema por fuerza bruta"
139     ]
140 },
141 {
142     "cell_type": "markdown",
143     "metadata": {
144         "colab_type": "text",
145         "id": "70rDgxYXkC8r"
146     },
147     "source": [
148         "Este algoritmo funciona desde la función principal valores, la función val entrega los resultados. En la función valores to
149     ]
150 },
151 {
152     "cell_type": "code",
153     "execution_count": 13,
154     "metadata": {
155         "colab": {},
156         "colab_type": "code",
157         "id": "CJb5sQ0qkENy"
158     },
159     "outputs": [
160         {
161             "name": "stdout",
162             "output_type": "stream",
163             "text": [
164                 "Valores no repetidos [-69, -68, -67, -66, -65, -64, -63, -62, -61, -60, -59, -58, -57, -56, -55, -54, -53, -52, -51, -50,
165                 "máximo: 77 mínimo: -69 distancia(inclusive): 147\n",
166                 "Tiempo de ejecución para algoritmo: 3.2348897457122803\n"
167             ]
168         }
169     ]
170 }
```

```

169 ],
170 "source": [
171     "from itertools import permutations\n",
172     "from time import time\n",
173     "\n",
174     "#Función para calcular el tiempo de ejecución\n",
175     "def calcular_tiempo(f): \n",
176     "    def wrapper(*args, **kwargs): \n",
177     "        inicio = time() \n",
178     "        resultado = f(*args, **kwargs) \n",
179     "        tiempo = time() - inicio\n",
180     "        print(\"Tiempo de ejecución para algoritmo: \" + str(tiempo))\n",
181     "        return resultado \n",
182     "    return wrapper\n",
183     "\n",
184     "def valores():\n",
185     "    lista1, lista2=[], []\n",
186     "    for i in permutations('123456789', 5):\n",
187     "        for j in permutations('+-*/', 4):\n",
188     "            expresion = eval(i[0]+j[0]+i[1]+j[1]+i[2]+j[2]+i[3]+j[3]+i[4])\n",
189     "            if abs(expresion) - abs(int(expresion)) == 0:\n",
190     "                lista1.append(int(expresion))\n",
191     "    lista1 = sorted(lista1) # todos los valores enteros repetidos\n",
192     "    for k in lista1: # filtra los valores repetidos\n",
193     "        if k not in lista2:\n",
194     "            lista2.append(k)\n",
195     "    return lista2\n",
196     "\n",
197     "@calcular_tiempo\n",
198     "def val(a):\n",
199     "    lista=valores()\n",
200     "    print('Valores no repetidos', lista)\n",
201     "    minimo=lista[0]\n",

```

```

202     "    maximo=lista[-1]\n",
203     "    distancia=len(lista)\n",
204     "    print('máximo:',maximo,'mínimo:',minimo,'distancia(inclusive):',distancia)\n",
205     "val(2)"
206 ]
207 },
208 {
209     "cell_type": "markdown",
210     "metadata": {
211         "colab_type": "text",
212         "id": "tgrnsC2nkFa8"
213     },
214     "source": [
215         "Calcula la complejidad del algoritmo por fuerza bruta"
216     ]
217 },
218 {
219     "cell_type": "markdown",
220     "metadata": {
221         "colab_type": "text",
222         "id": "9eUd3xwckP68"
223     },
224     "source": [
225         "Este cálculo está hecho sobre la función principal valores. Para calcular la complejidad del algoritmo hay que usar lo que
226         \"$\\frac{n!}{(n-5)!}\\cdot 4! = \\frac{9!}{(9-5)!}\\cdot 4! = \\frac{9!}{4!}\\cdot 4! = 9! = n!, (9+4+3)\\cdot n! + 2n + 2$
227     ]
228 },
229 {
230     "cell_type": "markdown",
231     "metadata": {
232         "colab_type": "text",
233         "id": "txbrmLmskReM"
234     },

```



```
235     "source": [  
236         "(*)Diseña un algoritmo que mejore la complejidad del algortimo por fuerza bruta. Argumenta porque crees que mejora el algor  
237     ]  
238 },  
239 {  
240     "cell_type": "markdown",  
241     "metadata": {  
242         "colab_type": "text",  
243         "id": "hLrVwt5QkZPD"  
244     },  
245     "source": [  
246         "El algoritmo mejora porque tiene menos código, trabaja con una sola lista, el anterior con dos. Quite un for, es decir, una  
247         "En este programa la función valores trabaja con una sola lista donde guarda los elementos [-69, 77], la lista contiene 147  
248     ]  
249 },  
250 {  
251     "cell_type": "code",  
252     "execution_count": 12,  
253     "metadata": {  
254         "colab": {},  
255         "colab_type": "code",  
256         "id": "m1eyM21Vkabg"  
257     },  
258     "outputs": [  
259     {  
260         "name": "stdout",  
261         "output_type": "stream",  
262         "text": [  
263             "Valores no repetidos [-69, -68, -67, -66, -65, -64, -63, -62, -61, -60, -59, -58, -57, -56, -55, -54, -53, -52, -51, -50,  
264             "máximo: 77 mínimo: -69 distancia(inclusive): 147\n",  
265             "Tiempo de ejecución para algoritmo: 3.3127963542938232\n"  
266         ]  
267     }  
}
```

```

268 ],
269 "source": [
270     "from itertools import permutations\n",
271     "from time import time\n",
272     "\n",
273     "#Función para calcular el tiempo de ejecución\n",
274     "def calcular_tiempo(f): \n",
275     "    def wrapper(*args, **kwargs): \n",
276     "        inicio = time() \n",
277     "        resultado = f(*args, **kwargs) \n",
278     "        tiempo = time() - inicio\n",
279     "        print(\"Tiempo de ejecución para algoritmo: \" + str(tiempo))\n",
280     "        return resultado \n",
281     "    return wrapper\n",
282     "\n",
283     "def valores():\n",
284     "    lista1=[]\n",
285     "    for i in permutations('123456789', 5):\n",
286     "        for j in permutations('+-*/', 4):\n",
287     "            expresion = eval(i[0]+j[0]+i[1]+j[1]+i[2]+j[2]+i[3]+j[3]+i[4])\n",
288     "            exp=int(expresion)\n",
289     "            if abs(expresion) - abs(exp) == 0 and exp not in lista1:\n",
290     "                lista1.append(exp)\n",
291     "    lista1 = sorted(lista1)\n",
292     "    return lista1\n",
293     "\n",
294     "@calcular_tiempo\n",
295     "def val(a):\n",
296     "    lista=valores()\n",
297     "    print('Valores no repetidos',lista)\n",
298     "    minimo=lista[0]\n",
299     "    maximo=lista[-1]\n",
300     "    distancia=len(lista)\n",

```

```

301     print('máximo:',maximo,'mínimo:',minimo,'distancia(inclusive):',distancia)\n",
302     "val(2)"
303 ]
304 },
305 {
306     "cell_type": "markdown",
307     "metadata": {
308         "colab_type": "text",
309         "id": "eGDB4K6uk1iN"
310     },
311     "source": [
312         "(*)Calcula la complejidad del algoritmo "
313     ]
314 },
315 {
316     "cell_type": "markdown",
317     "metadata": {
318         "colab": {},
319         "colab_type": "code",
320         "id": "nREzhkStlCq8"
321     },
322     "source": [
323         "Este cálculo está hecho sobre la función principal valores. Igual que el cálculo de complejidad del ejercicio anterior para
324         "$\\frac{n!}{(n-5)!} \\cdot 4! = \\frac{9!}{(9-5)!} \\cdot 4! = \\frac{9!}{4!} \\cdot 4! = 9! = n!$<br>\n",
325         "$ (9+2+2) \\cdot n! + 1 + 1 + 1 = 13n! + 3 = O(n!)$<br>"
326     ]
327 },
328 {
329     "cell_type": "markdown",
330     "metadata": {
331         "colab_type": "text",
332         "id": "3M6QjTsSkmYe"
333     },

```

```
334     "source": [  
335         "Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios"  
336     ]  
337 },  
338 {  
339     "cell_type": "markdown",  
340     "metadata": {  
341         "colab_type": "text",  
342         "id": "Jc3_OlyNkvjZ"  
343     },  
344     "source": [  
345         "La entrada de datos estaría dada por \"random.choice\" que admite strings. Lo otro interesante de esta función es el \"if r  
346     ]  
347 },  
348 {  
349     "cell_type": "code",  
350     "execution_count": 18,  
351     "metadata": {  
352         "colab": {},  
353         "colab_type": "code",  
354         "id": "sJ-N2etjkyWc"  
355     },  
356     "outputs": [  
357         {  
358             "name": "stdout",  
359             "output_type": "stream",  
360             "text": [  
361                 "5+3/1*7-9 = 17\n"  
362             ]  
363         }  
364     ],  
365     "source": [  
366         "import random\n",
```

```

367     "\n",
368     "def aleatorio():\n",
369     "     expresion = 2.3\n",
370     "     while (abs(expresion) - abs(int(expresion))) != 0:\n",
371     "         l1,l2=[],[]\n",
372     "         while len(l1) < 5:\n",
373     "             n = random.choice('123456789')\n",
374     "             if not n in l1:\n",
375     "                 l1.append(n)\n",
376     "                 while len(l2) < 4:\n",
377     "                     n = random.choice('+-*/')\n",
378     "                     if not n in l2:\n",
379     "                         l2.append(str(n))\n",
380     "                 cadena = l1[0]+l2[0]+l1[1]+l2[1]+l1[2]+l2[2]+l1[3]+l2[3]+l1[4]\n",
381     "                 expresion = eval(l1[0]+l2[0]+l1[1]+l2[1]+l1[2]+l2[2]+l1[3]+l2[3]+l1[4])\n",
382     "     return cadena,expresion\n",
383     "\n",
384     "cadena, expresion = aleatorio()\n",
385     "print(cadena, '=', int(expresion))"
386 ]
387 },
388 {
389     "cell_type": "markdown",
390     "metadata": {
391         "colab_type": "text",
392         "id": "zy5ZR0mjlGo1"
393     },
394     "source": [
395         "Aplica el algoritmo al juego de datos generado"
396     ]
397 },
398 {
399     "cell_type": "markdown",

```

```
400     "metadata": {
401         "colab_type": "text",
402         "id": "_Cmj-PVylMsa"
403     },
404     "source": [
405         "Este programa fusiona el programa anterior con el programa mejorado en su complejidad, aunque es un poco distinto, porque u
406     ]
407 },
408 {
409     "cell_type": "code",
410     "execution_count": 22,
411     "metadata": {
412         "colab": {},
413         "colab_type": "code",
414         "id": "Jkx8GeiYlUz1"
415     },
416     "outputs": [
417         {
418             "name": "stdout",
419             "output_type": "stream",
420             "text": [
421                 "Valores no repetidos [-63, -60, -49, -42, -36, -31, -29, -26, -25, -18, -17, -11, -10, -9, -5, -4, -3, -2, 0, 1, 3, 4, 5,
422                 "máximo: 75 mínimo: -63 distancia(inclusive): 59\n",
423                 "Tiempo de ejecución para algoritmo: 0.01564621925354004\n"
424             ]
425         }
426     ],
427     "source": [
428         "from itertools import permutations\n",
429         "from time import time\n",
430         "import random\n",
431         "\n",
432         "#Función para calcular el tiempo de ejecución\n",
```

```

433 "def calcular_tiempo(f): \n",
434 "    def wrapper(*args, **kwargs): \n",
435 "        inicio = time() \n",
436 "        resultado = f(*args, **kwargs) \n",
437 "        tiempo = time() - inicio\n",
438 "        print(\"Tiempo de ejecución para algoritmo: \" + str(tiempo))\n",
439 "        return resultado \n",
440 "    return wrapper\n",
441 "\n",
442 "def aleatorio():\n",
443 "    expresion = 2.3\n",
444 "    while (abs(expresion) - abs(int(expresion))) != 0:\n",
445 "        l1,l2=[],[]\n",
446 "        while len(l1) < 5:\n",
447 "            n = random.choice('123456789')\n",
448 "            if not n in l1:\n",
449 "                l1.append(n)\n",
450 "            while len(l2) < 4:\n",
451 "                n = random.choice('+-*/')\n",
452 "                if not n in l2:\n",
453 "                    l2.append(str(n))\n",
454 "            expresion = eval(l1[0]+l2[0]+l1[1]+l2[1]+l1[2]+l2[2]+l1[3]+l2[3]+l1[4])\n",
455 "    return expresion\n",
456 "\n",
457 "def valores():\n",
458 "    lista1=[]\n",
459 "    for i in range(100): \n",
460 "        expresion = int(aleatorio())\n",
461 "        if expresion not in lista1:\n",
462 "            lista1.append(expresion)\n",
463 "    lista1 = sorted(lista1)\n",
464 "    return lista1\n",
465 "\n",

```

```

466     "@calcular_tiempo\n",
467     "def val(a):\n",
468     "     lista=valores()\n",
469     "     print('Valores no repetidos',lista)\n",
470     "     minimo=lista[0]\n",
471     "     maximo=lista[-1]\n",
472     "     distancia=len(lista)\n",
473     "     print('máximo:',maximo,'mínimo:',minimo,'distancia(inclusive):',distancia)\n",
474     "val(2)"
475 ]
476 },
477 {
478     "cell_type": "markdown",
479     "metadata": {
480         "colab_type": "text",
481         "id": "eTFWUJQrtGcv"
482     },
483     "source": [
484         "Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo"
485     ]
486 },
487 {
488     "cell_type": "markdown",
489     "metadata": {
490         "colab_type": "text",
491         "id": "YKQ_mRBttWSP"
492     },
493     "source": [
494         "https://docs.python.org/3/library/itertools.html<br>\n",
495         "https://es.wikipedia.org/wiki/Variaci%C3%B3n_(combinatoria)<br>\n",
496         "https://python-para-impacientes.blogspot.com/2015/09/el-modulo-random.html<br>\n",
497         "https://blog.elcodiguero.com/python/eliminar-objetos-repetidos-de-una-lista.html<br>\n",
498         "Diapositivas de la asignatura: VIU-03MAIR-Sesion 05- VC3.pdf<br>\n",

```



```
499     "Diapositivas de la asignatura: VIU-03MAIR-Sesion 07- VC4.pdf"
500   ]
501 },
502 {
503   "cell_type": "markdown",
504   "metadata": {
505     "colab_type": "text",
506     "id": "kAkDPUyRtYyH"
507   },
508   "source": [
509     "Describe brevemente las lineas de como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posi
510   ]
511 },
512 {
513   "cell_type": "markdown",
514   "metadata": {
515     "colab_type": "text",
516     "id": "IE0uZmo-tZu-"
517   },
518   "source": [
519     "Sería interesante agregar la operación al cuadrado o al cubo. También que no sean dígitos los números, que sean números de
520     "Cabe mencionar que tuve problemas con la recursividad, implementé muchas funciones recursivas, pero el error típico que sal
521     "Por lo que mencionaba eran demasiadas permutations. Más arriba se menciona la cantidad de repeticiones o factoriales que se
522   ]
523 },
524 {
525   "cell_type": "code",
526   "execution_count": null,
527   "metadata": {},
528   "outputs": [],
529   "source": []
530 }
531 ],
```

```
532 "metadata": {
533   "colab": {
534     "collapsed_sections": [],
535     "include_colab_link": true,
536     "name": "Seminario(plantilla) - Algoritmos.ipynb",
537     "provenance": [],
538     "version": "0.3.2"
539   },
540   "kernel_spec": {
541     "display_name": "Python 3",
542     "language": "python",
543     "name": "python3"
544   },
545   "language_info": {
546     "codemirror_mode": {
547       "name": "ipython",
548       "version": 3
549     },
550     "file_extension": ".py",
551     "mimetype": "text/x-python",
552     "name": "python",
553     "nbconvert_exporter": "python",
554     "pygments_lexer": "ipython3",
555     "version": "3.7.0"
556   }
557 },
558 "nbformat": 4,
559 "nbformat_minor": 1
560 }
```

