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

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

```
      "\n",
      "(*) La respuesta es obligatoria\n",
      "\n",
      "\n",
      "\n",
      "\n",
      "\n",
      "                                          "
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "3_-exlrTgLd-"
     },
     "source": [
      "(*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?<br>\n",
      "\n",
      "(n!/(n-r)!)*m! Siendo n=digitos, r=cuantos dígitos se van a considerar, m=operaciones<br>\n",
      "(9!/(9-5)!)*4! = 362880<br>\n",
      "\n",
      "¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.<br>\n",
      "\n",
      "Hay dos casos los valores flotantes y los valores enteros. Lo que había que descartar eran los valores flotantes y cabe señ
     ]
    },
    {
     "cell_type": "code",
     "execution_count": 1,
     "metadata": {
      "colab": {},
      "colab_type": "code",
```

```
      "id": "iq6Fe32MgpDX"
     },
     "outputs": [
      {
       "name": "stdout",
       "output_type": "stream",
       "text": [
        "362880.0\n"
       ]
      }
     ],
     "source": [
      "def factorial(n):\n",
      "    if n <1:\n",
      "        return 1\n",
      "    else:\n",
      "        return n*factorial(n-1)\n",
      "    \n",
      "num_posibilidades = (factorial(9)/factorial(9-5))*factorial(4)\n",
      "print (num_posibilidades)"
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "dzynfVJahHoY"
     },
     "source": [
      "Modelo para el espacio de soluciones<br>\n",
      "(*) ¿Cual es la estructura de datos que mejor se adapta al problema? Argumentalo.(Es posible que hayas elegido una al princ
     ]
    },
```

```
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "xFIJrOwKi2Ti"
    },
    "source": [
     "Las mejores estructuras de datos en este caso fueron las listas de string para luego convertir con la función eval a una ex
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "7Z9U2W7bi-XS"
    },
    "source": [
     "Según el modelo para el espacio de soluciones<br>\n",
     "(*)¿Cual es la función objetivo?\n",
     "\n",
     "expresion = eval(i[0]+j[0]+i[1]+j[1]+i[2]+j[2]+i[3]+j[3]+i[4])\n",
     "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
     "\n",
     "(*)¿Es un problema de maximización o minimización?\n",
     "\n",
     "Es un problema de minimización, porque se buscan los valores enteros desde el menor -69 al mayor 77, osea, 147 valores de u
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "rlnTYgY1j6I2"
```

```
   },
   "source": [
    "Diseña un algoritmo para resolver el problema por fuerza bruta"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "70rDgxYXkC8r"
   },
   "source": [
    "Este algoritmo funciona desde la función principal valores, la función val entrega los resultados. En la función valores tc
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 13,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "id": "CJb5sQ0qkENy"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Valores no repetidos [-69, -68, -67, -66, -65, -64, -63, -62, -61, -60, -59, -58, -57, -56, -55, -54, -53, -52, -51, -50,
      "máximo: 77 mínimo: -69 distancia(inclusive): 147\n",
      "Tiempo de ejecución para algoritmo: 3.2348897457122803\n"
     ]
    }
```

```
   ],
   "source": [
    "from itertools import permutations\n",
    "from time import time\n",
    "\n",
    "#Función para calcular el tiempo de ejecución\n",
    "def calcular_tiempo(f): \n",
    "    def wrapper(*args, **kwargs):      \n",
    "        inicio = time()       \n",
    "        resultado = f(*args, **kwargs)       \n",
    "        tiempo = time() - inicio\n",
    "        print(\"Tiempo de ejecución para algoritmo: \"+str(tiempo))\n",
    "        return resultado \n",
    "    return wrapper\n",
    "\n",
    "def valores():\n",
    "    lista1,lista2=[],[]\n",
    "    for i in permutations('123456789', 5):\n",
    "        for j in permutations('+-*/', 4):\n",
    "            expresion = eval(i[0]+j[0]+i[1]+j[1]+i[2]+j[2]+i[3]+j[3]+i[4])\n",
    "            if abs(expresion) - abs(int(expresion)) == 0:\n",
    "                lista1.append(int(expresion))\n",
    "    lista1 = sorted(lista1)  # todos los valores enteros repetidos\n",
    "    for k in lista1:         # filtra los valores repetidos\n",
    "        if k not in lista2:\n",
    "            lista2.append(k)\n",
    "    return lista2\n",
    "\n",
    "@calcular_tiempo\n",
    "def val(a):\n",
    "    lista=valores()\n",
    "    print('Valores no repetidos',lista)\n",
    "    minimo=lista[0]\n",
```

```
    "    maximo=lista[-1]\n",
    "    distancia=len(lista)\n",
    "    print('máximo:',maximo,'mínimo:',minimo,'distancia(inclusive):',distancia)\n",
    "val(2)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "tgrnsC2nkFa8"
   },
   "source": [
    "Calcula la complejidad del algoritmo por fuerza bruta"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "9eUd3xwckP68"
   },
   "source": [
    "Este cálculo está hecho sobre la función principal valores. Para calcular la complejidad del algoritmo hay que usar lo que
    "$\\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
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "txbrmLmskReM"
   },
```

```
    "source": [
     "(*)Diseña un algoritmo que mejore la complejidad del algortimo por fuerza bruta. Argumenta porque crees que mejora el algor
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "hLrVwt5QkZPD"
    },
    "source": [
     "El algoritmo mejora porque tiene menos código, trabaja con una sola lista, el anterior con dos. Quite un for, es decir, una
     "En este programa la función valores trabaja con una sola lista donde guarda los elementos [-69, 77], la lista contiene 147
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 12,
    "metadata": {
     "colab": {},
     "colab_type": "code",
     "id": "m1eyM21Vkabg"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "Valores no repetidos [-69, -68, -67, -66, -65, -64, -63, -62, -61, -60, -59, -58, -57, -56, -55, -54, -53, -52, -51, -50,
       "máximo: 77 mínimo: -69 distancia(inclusive): 147\n",
       "Tiempo de ejecución para algoritmo: 3.3127963542938232\n"
      ]
     }
```

```
   ],
   "source": [
    "from itertools import permutations\n",
    "from time import time\n",
    "\n",
    "#Función para calcular el tiempo de ejecución\n",
    "def calcular_tiempo(f): \n",
    "    def wrapper(*args, **kwargs):      \n",
    "        inicio = time()       \n",
    "        resultado = f(*args, **kwargs)       \n",
    "        tiempo = time() - inicio\n",
    "        print(\"Tiempo de ejecución para algoritmo: \"+str(tiempo))\n",
    "        return resultado \n",
    "    return wrapper\n",
    "    \n",
    "def valores():\n",
    "    lista1=[]\n",
    "    for i in permutations('123456789', 5):\n",
    "        for j in permutations('+-*/', 4):\n",
    "            expresion = eval(i[0]+j[0]+i[1]+j[1]+i[2]+j[2]+i[3]+j[3]+i[4])\n",
    "            exp=int(expresion)\n",
    "            if abs(expresion) - abs(exp) == 0 and exp not in lista1:\n",
    "                lista1.append(exp)\n",
    "    lista1 = sorted(lista1)\n",
    "    return lista1\n",
    "\n",
    "@calcular_tiempo\n",
    "def val(a):\n",
    "    lista=valores()\n",
    "    print('Valores no repetidos',lista)\n",
    "    minimo=lista[0]\n",
    "    maximo=lista[-1]\n",
    "    distancia=len(lista)\n",
```

```
      "    print('máximo:',maximo,'mínimo:',minimo,'distancia(inclusive):',distancia)\n",
      "val(2)"
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "eGDB4K6uk1iN"
     },
     "source": [
      "(*)Calcula la complejidad del algoritmo "
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab": {},
      "colab_type": "code",
      "id": "nREzhkStlCq8"
     },
     "source": [
      "Este cálculo está hecho sobre la función principal valores. Igual que el cálculo de complejidad del ejercicio anterior para
      "$\\frac{n!}{(n-5)!}\\cdot 4! = \\frac{9!}{(9-5)!}\\cdot 4! = \\frac{9!}{4!}\\cdot 4! = 9! = n!$<br>\n",
      "$(9+2+2)\\cdot n! + 1 + 1 + 1 = 13n! + 3 = O(n!)$<br>"
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "3M6QjTsSkmYe"
     },
     "source": [
```

```
    "source": [
     "Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios"
    ]
   },
   {
    "cell_type": "markdown",
    "metadata": {
     "colab_type": "text",
     "id": "Jc3_OlyNkvjZ"
    },
    "source": [
     "La entrada de datos estaría dada por \"random.choice\" que admite strings. Lo otro interesante de esta función es el \"if r
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 18,
    "metadata": {
     "colab": {},
     "colab_type": "code",
     "id": "sJ-N2etjkyWc"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "5+3/1*7-9 = 17\n"
      ]
     }
    ],
    "source": [
     "import random\n",
```

```
    "\n",
    "def aleatorio():\n",
    "    expresion = 2.3\n",
    "    while (abs(expresion) - abs(int(expresion))) != 0:\n",
    "        l1,l2=[],[]\n",
    "        while len(l1) < 5:\n",
    "            n = random.choice('123456789')\n",
    "            if not n in l1:\n",
    "                l1.append(n)\n",
    "            while len(l2) < 4:\n",
    "                n = random.choice('+-*/')\n",
    "                if not n in l2:\n",
    "                    l2.append(str(n))\n",
    "        cadena = l1[0]+l2[0]+l1[1]+l2[1]+l1[2]+l2[2]+l1[3]+l2[3]+l1[4]\n",
    "        expresion = eval(l1[0]+l2[0]+l1[1]+l2[1]+l1[2]+l2[2]+l1[3]+l2[3]+l1[4])\n",
    "    return cadena,expresion\n",
    "\n",
    "cadena, expresion = aleatorio()\n",
    "print(cadena,'=',int(expresion))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "colab_type": "text",
    "id": "zy5ZROmjlGo1"
   },
   "source": [
    "Aplica el algoritmo al juego de datos generado"
   ]
  },
  {
   "cell_type": "markdown",
```

```
    "metadata": {
     "colab_type": "text",
     "id": "_Cmj-PVylMsa"
    },
    "source": [
     "Este programa fusiona el programa anterior con el programa mejorado en su complejidad, aunque es un poco distinto, porque u
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 22,
    "metadata": {
     "colab": {},
     "colab_type": "code",
     "id": "Jkx8GeiYlUz1"
    },
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "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,
       "máximo: 75 mínimo: -63 distancia(inclusive): 59\n",
       "Tiempo de ejecución para algoritmo: 0.01564621925354004\n"
      ]
     }
    ],
    "source": [
     "from itertools import permutations\n",
     "from time import time\n",
     "import random\n",
     "\n",
     "#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",
```

```
      "@calcular_tiempo\n",
      "def val(a):\n",
      "    lista=valores()\n",
      "    print('Valores no repetidos',lista)\n",
      "    minimo=lista[0]\n",
      "    maximo=lista[-1]\n",
      "    distancia=len(lista)\n",
      "    print('máximo:',maximo,'mínimo:',minimo,'distancia(inclusive):',distancia)\n",
      "val(2)"
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "eTFWUJQrtGcv"
     },
     "source": [
      "Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo"
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "YKQ_mRBttWsP"
     },
     "source": [
      "https://docs.python.org/3/library/itertools.html<br>\n",
      "https://es.wikipedia.org/wiki/Variaci%C3%B3n_(combinatoria)<br>\n",
      "https://python-para-impacientes.blogspot.com/2015/09/el-modulo-random.html<br>\n",
      "https://blog.elcodiguero.com/python/eliminar-objetos-repetidos-de-una-lista.html<br>\n",
      "Diapositivas de la asignatura: VIU-03MAIR-Sesion 05- VC3.pdf<br>\n",
```

```
      "Diapositivas de la asignatura: VIU-03MAIR-Sesion 07- VC4.pdf"
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "kAkDPUyRtYyH"
     },
     "source": [
      "Describe brevemente las lineas de como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posib
     ]
    },
    {
     "cell_type": "markdown",
     "metadata": {
      "colab_type": "text",
      "id": "IE0uZmo-tZu-"
     },
     "source": [
      "Sería interesante agregar la operación al cuadrado o al cubo. También que no sean dígitos los números, que sean números de
      "Cabe mencionar que tuve problemas con la recursividad, implementé muchas funciones recursivas, pero el error típico que sal
      "Por lo que mencionaba eran demasiadas permutations. Más arriba se menciona la cantidad de repeticiones o factoriales que se
     ]
    },
    {
     "cell_type": "code",
     "execution_count": null,
     "metadata": {},
     "outputs": [],
     "source": []
    }
   ],
```

```
  "metadata": {
   "colab": {
    "collapsed_sections": [],
    "include_colab_link": true,
    "name": "Seminario(plantilla) - Algoritmos.ipynb",
    "provenance": [],
    "version": "0.3.2"
   },
   "kernelspec": {
    "display_name": "Python 3",
    "language": "python",
    "name": "python3"
   },
   "language_info": {
    "codemirror_mode": {
     "name": "ipython",
     "version": 3
    },
    "file_extension": ".py",
    "mimetype": "text/x-python",
    "name": "python",
    "nbconvert_exporter": "python",
    "pygments_lexer": "ipython3",
    "version": "3.7.0"
   }
  },
  "nbformat": 4,
  "nbformat_minor": 1
}
```