Branch: master ▾ 03MAIR-Algoritmos-de-optimizacion / SEMINARIO / Seminario_Isabel_Vazquez.ipynb

Find file Copy path

404isabel Creado con Colaboratory c782667 2 minutes ago

1 contributor

1733 lines (1733 sloc) | 121 KB

Raw Blame History

Open in Colab

Algoritmos de optimización - Seminario

Nombre y Apellidos: Isabel Vázquez Trigás
Url: https://github.com/404isabel/03MAIR-Algoritmos-de-optimizacion/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. Un ejemplo sería para obtener el 2:

4+2-6/3*2 = 2

• Debe analizarse el problema para encontrar soluciones a todos los valores enteros posibles planteando las siguientes cuestiones:

- ¿Qué valor máximo y mínimo a priori se pueden obtener según las condiciones?
- ¿Es posible encontrar todos los valores posibles entre dicho mínimo y máximo ?

(*) La respuesta es obligatoria

Valor máximo:77 Valor mínimo:1

Explicación

Siempre voy a poder obtener un resultado para +*-/

a+b*c-d/e

Se traduce en x1+x2-x3

x1 (a) entre [1,9]

x2 (b x c) entre [2,72] (1x2) , (9 x 8)

x3 (d/e) entre [2,9] (2/1), (9/1)

Para obtener el máximo valor posible a obtener:

Si x2 es 72 (9*8),

entonces máximo valor posible de x1=7 y valor mínimo de x3=1 --> 7+72-2=77

En cuanto al mínimo:

Si maximizamos x3 a 9, entonces x3=9/1, por lo que el menor valor para x2, es 6 (2*3), lo que nos deja un valor de 4 para x1

Por ello, observamos que el menor valor posible es 1

4+2*3-9/1

Se puede obtener un resultado para todo el rango de números enteros: [1,77]

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

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones?

Respuesta

Teniendo en cuenta las restricciones:

Vmn(9,5) * 4!

Posibilidades para los números: 9!/(9-5)! = 15120

Posibilidades para los signos: 4! = 24

Posibilidades totales, teniendo en cuenta las restricciones: 15120*24 = 362880

Por otro lado, si tenemos en cuenta que siempre vamos a obtener una solución para el órden de operadores (+,*,-,/), sólo hay que tener en cuenta para el cálculo las posibilidades para los números:

Posibilidades totales: 15120

Si tener en cuenta las restricciones:

Suponiendo que podemos repetir tanto números como operadores:

Posiblidades para los operadores:4^4 = 246

Posibilidades para los números: 9^5 = 59049

Posibilidades totales: 246*59049 = 14526054

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

La mejor estructura de datos que se adapta al problema es:

- Un array de todos los números posibles, ordenados de menor a mayor NUMEROS=[1,2,3,4,5,6,7,8,9]
- Un array con todos los posibles operadores. OPERADORES=["+","*","-","/"]

Para el primer algoritmo de fuerza bruta que implemento, el órden de los operadores es irrelevante, sin embargo, para el siguiente algoritmo, necesito que vengan en el órden: +,*,-,/ (ya que éste es el órden que siempre me va a proporcionar un resultado para cualquier caso, y en la definición del problema se quiere encontrar una solución, no todas). Este orden me permite reducir el número de operaciones.

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

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

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

Respuesta

- Función objetivo: La función objetivo evalua cómo de alejados estamos del valor objetivo. El valor objetivo es aquella función formada por 5 números y 4 operadores, para la cual se cumple que al evaluarla se obtiene un número dado. Es decir, la función objetivo será: a+b*c-d/e. Por cada conjunto obtenido (a,b,c,d,e) se evaluará si se tiene como resultado el número pasado como parámetro.
- El problema no es de maximización ni de minimización, se trata de obtener una expresión en concreto que obtenga un número dado.

In [26]:
```
def obtenerExpresion(expresion,num):
  #Se comprueba que la longitud sea 9 y que la función eval de python nos retorne el número
  if len(expresion)==9 and eval(expresion)==num:
    print("Solución "+ expresion)
  else:
    print("No es solución")

#Ejemplos
obtenerExpresion("1+2*5-9/3",8)
obtenerExpresion("1+2*4-8/5",8)
```

```
Solución 1+2*5-9/3
No es solución
```

Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

In [27]:
```
from time import time
GUARDAR_RESULTADO=False
TIEMPOS=[]
```

```
def calcular_tiempo(f):

    def wrapper(*args, **kwargs):
        inicio = time()
        resultado = f(*args, **kwargs)
        tiempo = float(time() - inicio)
        print("\r\n Tiempo de ejecución para algoritmo: "+"{0:.25f}".format(tiempo)+"\r\n")
        if GUARDAR_RESULTADO==True:
          TIEMPOS.append(tiempo)
        return resultado

    return wrapper

### Precondiciones:

#Valor mínimo a obtener: 1
expresionMinima="9/3*2-6+1"
print("Valor mínimo posible a obtener "+str(int(eval(expresionMinima))))
#Valor máximo a obtener: 77
expresionMaxima="7+8*9/1-2"
#expresionMaxima="8*9+7/1-2"
print("Valor máximo posible a obtener "+str(int(eval(expresionMaxima))))
#Se le da prioridad al * y / (izquierda a derecha) frente a + y -

MINIMO=1
MAXIMO=77
NUMEROS=[1,2,3,4,5,6,7,8,9]
OPERADORES=["+","*","-","/"]
#Con cualquier órden de operaciones, este algoritmo funciona
#OPERACIONES=["+","-","*","/"]



def obtenerEnteroFB(num,ENTEROS,OPERACIONES):
    a=0
    b=0
    c=0
    d=0
    e=0
    o1=""
```

```
    o2=""
    o3=""
    o4=""

    for e1 in ENTEROS:
      a=e1
      for op1 in OPERACIONES:
        o1=op1
        for e2 in ENTEROS:
          if(e2!=e1):
            b=e2
            for op2 in OPERACIONES:
              if(op2!=op1):
                o2=op2
                for e3 in ENTEROS:
                    if(e3!=e1 and e3!=e2):
                      c=e3
                      for op3 in OPERACIONES:
                        if(op3!=op2 and op3!=op1):
                          o3=op3
                          for e4 in ENTEROS:
                            if(e4!=e3 and e4!=e2 and e4!=e1):
                              d=e4
                              for op4 in OPERACIONES:
                                if(op4!=op3 and op4!=op2 and op4!=op1):
                                  o4=op4
                                  for e5 in ENTEROS:
                                    if(e5!=e4 and e5!=e3 and e5!=e2 and e5!=e1):
                                      e=e5
                                      expresion=str(a)+str(o1)+str(b)+str(o2)+str(c)+str(o3)+st
r(d)+str(o4)+str(e)

                                      if(eval(expresion)==num):
                                        #print(str(expresion))
                                        return expresion



@calcular_tiempo
def obtenerEnteroFBPintar(num,ENTEROS,OPERACIONES):
```

```
  resultado=obtenerEnteroFB(num,ENTEROS,OPERACIONES)
  if resultado == None:
    print("No se puede obtener un valor para "+str(num)+" debido a que no está etre el máximo y
 el mínimo de la función para los datos de entrada")
  else:
    print("Resultado para lista de "+str(len(ENTEROS))+ " elementos, para el número: "+str(num)
+" :"+str(resultado))

print("\r\n")
obtenerEnteroFBPintar(78,NUMEROS,OPERADORES)
obtenerEnteroFBPintar(1,NUMEROS,OPERADORES)
obtenerEnteroFBPintar(8,NUMEROS,OPERADORES)
obtenerEnteroFBPintar(40,NUMEROS,OPERADORES)
obtenerEnteroFBPintar(77,NUMEROS,OPERADORES)
```

```
Valor mínimo posible a obtener 1
Valor máximo posible a obtener 77


No se puede obtener un valor para 78 debido a que no está etre el máximo y el mínimo de la funci
ón para los datos de entrada

 Tiempo de ejecución para algoritmo: 3.4646852016448974609375000

Resultado para lista de 9 elementos, para el número: 1 :1+2*6/3-4

 Tiempo de ejecución para algoritmo: 0.0021235942840576171875000

Resultado para lista de 9 elementos, para el número: 8 :1+2*5-9/3

 Tiempo de ejecución para algoritmo: 0.0012917518615722656250000

Resultado para lista de 9 elementos, para el número: 40 :1+6*7-9/3

 Tiempo de ejecución para algoritmo: 0.0494103431701660156250000

Resultado para lista de 9 elementos, para el número: 77 :7+8*9-2/1

 Tiempo de ejecución para algoritmo: 2.3665907382965087890625000
```

Calcula la complejidad del algoritmo por fuerza bruta

Respuesta

El algoritmo de fuerza bruta tiene la siguiente complejidad: O(n^9) (orden polinómico)

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

Respuesta

Se implementa algoritmo basado en el concepto de ramificación y poda (en profundidad).

Este algoritmo mejora con respecto al algoritmo de fuerza bruta debido a los siguientes motivos:

Mejora en complejidad, debido a que este algoritmo tiene en cuenta que siempre existe una solución para el órden de operadores (+,*,-,/). Por ese motivo, no se van a comprobar otros posibles órdenes, esto hace que pasemos de 9 bucles a 5.

El algoritmo implementado, se trata de un algoritmo basado en el concepto de ramificación y poda. Es un algoritmo iterativo (no recursivo) que va iterando a través de todos los nodos, y evaluando condiciones en todos los niveles para poder podar aquellos nodos que no nos van a conducir a una solución (y así no evaluarlos). De este modo evitamos tener que recorrer todos y cada uno de los nodos, resultando en una reducción significativa del total de las operaciones.

Para los datos del problema (números del 1 al 9), se aprecian reducciones de tiempos, pero, además, si ampliamos el tamaño de la lista de números se puede observar con más claridad que las mejoras son muy importantes. Para más información, ir al último apartado de este documento, donde se pueden ver comparativas.

Las explicaciones de cómo funciona el algoritmo se pueden seguir en los comentarios del código.

In [28]:
```
#Algoritmo similar a ramificación y poda (5 nodos)

def esPrometedoraNodo2(x2,j,NUMEROS):
```

```
  return (x2%j==0 and x2/j!=j and x2/j<=len(NUMEROS) and x2/j>0)

#El nodo 4 no es prometedor si es imposible que para x1+x2-x3 pueda obtener un resultado <=num
def esPrometedoraNodo4(a,b,c,num,OPERADORES,NUMEROS):
  maximoX3=len(NUMEROS) #Lo máximo que voy a poder restar (d/e)
  resultado=eval(str(a)+OPERADORES[0]+str(b)+OPERADORES[1]+str(c)+OPERADORES[2]+str(maximoX3))
  return resultado<=num

def esPrometedoraNodo5(l,p):#El resultado de la división tiene que ser un entero
  return (l%p==0)


def calculoExpr(x2,num,NUMEROS):
  #x2: divido x2 entre 1,2,3,4.... siempre y cuando el resto sea 0, si fueran iguales, ej: 3*3,
 descarto esa posibilidad
  for j in NUMEROS:
      if esPrometedoraNodo2(x2,j,NUMEROS):#Nodo 2: Podo esta rama, si la rama no es prometedora
        #Se obtienen a la vez b y c (que son el resultado de una multiplicación)
        b=int(x2/j)
        c=j
        for k in NUMEROS:
          if(k!=b and k!=c):#Nodo 3: Podo esta rama, si no cumplo las condiciones (este nodo ha
rá referencia al valor de a)
            a=k
            for l in NUMEROS:
              if l!=a and l!=b and l!=c and esPrometedoraNodo4(a,b,c,num,OPERADORES,NUMEROS):#N
odo 4: Podo esta rama, si no es prometedora (hace referencia a d)
                hasta=int(len(NUMEROS)/2)+1
                for p in range(1,hasta): #No tiene sentido calcular todos los valores para e, s
ólo aquellos para los que pueda obtener de la división un valor entero
                  if (esPrometedoraNodo5(l,p) and p!=a and p!=b and p!=c and p!=l):#Nodo 5: Pod
o esta rama, si la rama no es prometedora (hace referencia a e)
                    d=l
                    e=p
                    expresion=str(a)+OPERADORES[0]+str(b)+OPERADORES[1]+str(c)+OPERADORES[2]+st
r(d)+OPERADORES[3]+str(e)
                    if(eval(expresion)==num):
                      return expresion
```

```
def calculoSimple(num,NUMEROS):
  #Tal y como hemos podido ver en el enunciado, siempre voy a poder obtener un resultado para +
*-/
  #a+b*c-d/e

  #Se traduce en x1+x2-x3
  #x1 entre [1,9]  a
  #x2 entre [2,72] (9*8) b*c
  #x3 entre [2,9] d/e

  #x2+(x1-x3)=num

  #Por ello el mayor valor posible es 77, ya que si x2 es 72,
  #entonces máximo valor posible de x1=7 y valor mínimo de x3=1 --> 7+72-2=77

  #Al final, todo se puede simplificar en :
  #x-8=num --> x=num+8
  #x+7=num --> x=num-7



  #Si queremos generalizar los cálculos obtenemos que:
  #x1 entre [min,max] a
  #x2 entre [min+1,max*(max-1)]
  #x3 entre [min+1,max]

  #x=num+(max-1)
  #x=num-(max+2)

  for i in range(0,len(NUMEROS)-1):#Nodo 1 (empiezo calculando x2 para b y c)
    x2=num+i
    x2prima=num-i
    #Buscar soluciones, partiendo de x2 (si no se obtienen soluciones, buscar a partir de x2pri
ma)
    exp=calculoExpr(x2,num,NUMEROS)
    if exp==None:
      exp2=calculoExpr(x2prima,num,NUMEROS)
      if(exp2!=None):
        return exp2
    else:
```

```
      return exp



@calcular_tiempo
def calculoSimplePintar(num,NUMEROS):

  resultado=calculoSimple(num,NUMEROS)
  if resultado == None:
    print("No se puede obtener un valor para "+str(num)+" debido a que no está etre el máximo y
 el mínimo de la función para los datos de entrada")
  else:
    print("Resultado para lista de "+str(len(NUMEROS))+ " elementos, para el número: "+str(num)
+" :"+str(resultado))

print("\r\n")

calculoSimplePintar(78,NUMEROS)
calculoSimplePintar(1,NUMEROS)
calculoSimplePintar(8,NUMEROS)
calculoSimplePintar(40,NUMEROS)
calculoSimplePintar(77,NUMEROS)
```

```
No se puede obtener un valor para 78 debido a que no está etre el máximo y el mínimo de la funci
ón para los datos de entrada

 Tiempo de ejecución para algoritmo: 0.00196814537048339843750000

Resultado para lista de 9 elementos, para el número: 1 :4+3*2-9/1

 Tiempo de ejecución para algoritmo: 0.00517892837524414062500000

Resultado para lista de 9 elementos, para el número: 8 :2+8*1-6/3

 Tiempo de ejecución para algoritmo: 0.00016617774963378906250000

Resultado para lista de 9 elementos, para el número: 40 :2+8*5-6/3

 Tiempo de ejecución para algoritmo: 0.00060391426086425781250000
```

```
Resultado para lista de 9 elementos, para el número: 77 :7+9*8-2/1

 Tiempo de ejecución para algoritmo: 0.000940322875976562500000000
```

(*)Calcula la complejidad del algoritmo

Respuesta

Complejidad del algoritmo: O(n^5) (orden polinómico)

Además del algoritmo de fuerza bruta y ramificación y poda, he diseñado otro algoritmo de fuerza bruta que tiene en cuenta el órden de los operadores (+,*,-,/)

Con este algoritmo, quería comprobar que las mejoras del algoritmo de ramificación y poda no sólo eran debidas a la reducción de la complejidad, sino a las comprobaciones efectuadas en cada nodo. A continuación, éste es el tercer algoritmo de prueba que he creado. Este algoritmo tiene también orden O(n^5). En el último apartado de este documento se pueden ver las pruebas efectuadas.

In [29]:
```
#Algoritmo de fuerza bruta, para el cual sólo tengo en cuenta el órden (+*-/), ya que este órde
n siempre nos va a devolver un resultado
#Con este algoritmo se observan mejoras, ya que se reduce la complejidad del mismo

def obtenerEnteroFBRest(num,ENTEROS,OPERACIONES):
    a=0
    b=0
    c=0
    d=0
    e=0
    o1=""
    o2=""
    o3=""
    o4=""

    for e1 in ENTEROS:
      a=e1
      o1="+"
      for e2 in ENTEROS:
```

```
        if(e2!=e1):
          b=e2
          o2="*"
          for e3 in ENTEROS:
            if(e3!=e1 and e3!=e2):
                c=e3
                o3="-"
                for e4 in ENTEROS:
                  if(e4!=e3 and e4!=e2 and e4!=e1):
                    d=e4
                    o4="/"
                    for e5 in ENTEROS:
                      if(e5!=e4 and e5!=e3 and e5!=e2 and e5!=e1):
                          e=e5
                          expresion=str(a)+str(o1)+str(b)+str(o2)+str(c)+str(o3)+str(d)+str(o
4)+str(e)
                          if(eval(expresion)==num):
                            return expresion


@calcular_tiempo
def obtenerEnteroFBRestPintar(num,ENTEROS,OPERACIONES):

  resultado=obtenerEnteroFBRest(num,ENTEROS,OPERACIONES)
  if resultado == None:
    print("No se puede obtener un valor para "+str(num)+" debido a que no está etre el máximo y
 el mínimo de la función para los datos de entrada")
  else:
    print("Resultado para lista de "+str(len(ENTEROS))+ " elementos, para el número: "+str(num)
+" :"+str(resultado))

print("\r\n")
obtenerEnteroFBRestPintar(78,NUMEROS,OPERADORES)
obtenerEnteroFBRestPintar(1,NUMEROS,OPERADORES)
obtenerEnteroFBRestPintar(8,NUMEROS,OPERADORES)
obtenerEnteroFBRestPintar(40,NUMEROS,OPERADORES)
obtenerEnteroFBRestPintar(77,NUMEROS,OPERADORES)
```

```
No se puede obtener un valor para 78 debido a que no está etre el máximo y el mínimo de la funci
```

```
ón para los datos de entrada

 Tiempo de ejecución para algoritmo: 0.14136648178100585937500000

Resultado para lista de 9 elementos, para el número: 1 :4+2*3-9/1

 Tiempo de ejecución para algoritmo: 0.04998087882995605468750000

Resultado para lista de 9 elementos, para el número: 8 :1+2*5-9/3

 Tiempo de ejecución para algoritmo: 0.00087833404541015625000000

Resultado para lista de 9 elementos, para el número: 40 :1+6*7-9/3

 Tiempo de ejecución para algoritmo: 0.00992751121520996093750000

Resultado para lista de 9 elementos, para el número: 77 :7+8*9-2/1

 Tiempo de ejecución para algoritmo: 0.11001515388488769531250000
```

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

Respuesta

In [30]:
```
# En principio, según la descripción del problema, no tiene sentido generar datos aleatorios pa
ra los operadores
# En cuanto a los números, atendiendo a las restricciones del problema tampoco tiene cabida gen
erar números aleatorios, ya que sólo son 77, y podemos probarlos todos.
# Pero, tal y como he implementado el algoritmo sí que puedo generar listas variables en tamañ
o. (Para esa lista de gran tamaño, genero números aleatorios)

import random

#Lista de números a probar con todos los datos de la definición del problema [1,77]
lista1=[x for x in range(1,78)]
numerosDefinicion=[1,2,3,4,5,6,7,8,9]
#Para una lista de números de gran tamaño, creo un juego de datos aleatorios, comprendidos entr
e los valores 1,2000
numerosListaGrande=[x for x in range(1,2356)]
```

```
numerosListaGrande=[x for x in range(1,2550)]
lista2=[]
for i in range(10):
  lista2.append(random.randint(1,len(numerosListaGrande)))
print(lista2)
```

```
[1570, 1564, 1018, 661, 336, 1781, 651, 1788, 1632, 732]
```

Aplica el algoritmo al juego de datos generado

Respuesta

In [31]:
```
#Cálculo de todos los números posibles según la definición inicial del problema
for i in lista1:
  calculoSimplePintar(i,numerosDefinicion)
```

```
Resultado para lista de 9 elementos, para el número: 1 :4+3*2-9/1

 Tiempo de ejecución para algoritmo: 0.010751962661743164062500000

Resultado para lista de 9 elementos, para el número: 2 :4+3*2-8/1

 Tiempo de ejecución para algoritmo: 0.009334564208984375000000000

Resultado para lista de 9 elementos, para el número: 3 :2+3*1-8/4

 Tiempo de ejecución para algoritmo: 0.000432729721069335937500000

Resultado para lista de 9 elementos, para el número: 4 :2+4*1-6/3

 Tiempo de ejecución para algoritmo: 0.000297307968139648437500000

Resultado para lista de 9 elementos, para el número: 5 :2+5*1-6/3

 Tiempo de ejecución para algoritmo: 0.000278949737548828125000000

Resultado para lista de 9 elementos, para el número: 6 :2+6*1-8/4

 Tiempo de ejecución para algoritmo: 0.000237464904785156250000000

Resultado para lista de 9 elementos, para el número: 7 :2+7*1-6/3
```

```
Resultado para lista de 9 elementos, para el número: 7 :12+7*1-6/3
 Tiempo de ejecución para algoritmo: 0.00029897689819335937500000
Resultado para lista de 9 elementos, para el número: 8 :2+8*1-6/3
 Tiempo de ejecución para algoritmo: 0.00024533271789550781250000
Resultado para lista de 9 elementos, para el número: 9 :2+9*1-6/3
 Tiempo de ejecución para algoritmo: 0.00023603439331054687500000
Resultado para lista de 9 elementos, para el número: 10 :3+9*1-4/2
 Tiempo de ejecución para algoritmo: 0.00321745872497558593750000
Resultado para lista de 9 elementos, para el número: 11 :1+6*2-8/4
 Tiempo de ejecución para algoritmo: 0.00014638900756835937500000
Resultado para lista de 9 elementos, para el número: 12 :1+7*2-9/3
 Tiempo de ejecución para algoritmo: 0.00599670410156250000000000
Resultado para lista de 9 elementos, para el número: 13 :1+7*2-6/3
 Tiempo de ejecución para algoritmo: 0.00010371208190917968750000
Resultado para lista de 9 elementos, para el número: 14 :1+5*3-4/2
 Tiempo de ejecución para algoritmo: 0.00385880470275878906250000
Resultado para lista de 9 elementos, para el número: 15 :2+5*3-8/4
 Tiempo de ejecución para algoritmo: 0.00037765502929687500000000
Resultado para lista de 9 elementos, para el número: 16 :4+5*3-6/2
 Tiempo de ejecución para algoritmo: 0.00387382507324218750000000
Resultado para lista de 9 elementos, para el número: 17 :1+9*2-6/3
```

```
Resultado para lista de 9 elementos, para el número: 17 :1+9-2-6/3
 Tiempo de ejecución para algoritmo: 0.000082492828369140625000
Resultado para lista de 9 elementos, para el número: 18 :2+6*3-8/4
 Tiempo de ejecución para algoritmo: 0.00101613998413085937500000
Resultado para lista de 9 elementos, para el número: 19 :1+5*4-6/3
 Tiempo de ejecución para algoritmo: 0.00103378295898437500000000
Resultado para lista de 9 elementos, para el número: 20 :2+5*4-6/3
 Tiempo de ejecución para algoritmo: 0.00025248527526855468750000
Resultado para lista de 9 elementos, para el número: 21 :2+7*3-8/4
 Tiempo de ejecución para algoritmo: 0.00069952011108398437500000
Resultado para lista de 9 elementos, para el número: 22 :4+7*3-6/2
 Tiempo de ejecución para algoritmo: 0.00036621093750000000000000
Resultado para lista de 9 elementos, para el número: 23 :1+8*3-4/2
 Tiempo de ejecución para algoritmo: 0.00016784667968750000000000
Resultado para lista de 9 elementos, para el número: 24 :1+9*3-8/2
 Tiempo de ejecución para algoritmo: 0.00830721855163574218750000
Resultado para lista de 9 elementos, para el número: 25 :4+8*3-6/2
 Tiempo de ejecución para algoritmo: 0.00038838386535644531250000
Resultado para lista de 9 elementos, para el número: 26 :1+9*3-4/2
 Tiempo de ejecución para algoritmo: 0.00021910667419433593750000
Resultado para lista de 9 elementos, para el número: 27 :2+9*3-8/4
```

```
Resultado para lista de 9 elementos, para el número: 27 :2+9*3-6/4
 Tiempo de ejecución para algoritmo: 0.000483751296997070312500 0
Resultado para lista de 9 elementos, para el número: 28 :2+7*4-6/3
 Tiempo de ejecución para algoritmo: 0.000336408615112304687500 0
Resultado para lista de 9 elementos, para el número: 29 :1+6*5-4/2
 Tiempo de ejecución para algoritmo: 0.000315427780151367187500 0
Resultado para lista de 9 elementos, para el número: 30 :2+6*5-8/4
 Tiempo de ejecución para algoritmo: 0.000372886657714843750000 0
Resultado para lista de 9 elementos, para el número: 31 :1+8*4-6/3
 Tiempo de ejecución para algoritmo: 0.000231027603149414062500 0
Resultado para lista de 9 elementos, para el número: 32 :2+8*4-6/3
 Tiempo de ejecución para algoritmo: 0.000384569168090820312500 0
Resultado para lista de 9 elementos, para el número: 33 :3+8*4-2/1
 Tiempo de ejecución para algoritmo: 0.000700950622558593750000 0
Resultado para lista de 9 elementos, para el número: 34 :1+7*5-4/2
 Tiempo de ejecución para algoritmo: 0.000326156616210937500000 0
Resultado para lista de 9 elementos, para el número: 35 :2+7*5-6/3
 Tiempo de ejecución para algoritmo: 0.000466585159301757812500 0
Resultado para lista de 9 elementos, para el número: 36 :2+9*4-6/3
 Tiempo de ejecución para algoritmo: 0.000426292419433593750000 0
Resultado para lista de 9 elementos, para el número: 37 :3+9*4-2/1
```

Resultado para lista de 9 elementos, para el número: 37 :3+9*4-2/1

 Tiempo de ejecución para algoritmo: 0.000286579132080078125000

Resultado para lista de 9 elementos, para el número: 38 :1+8*5-6/2

 Tiempo de ejecución para algoritmo: 0.000571012496948242187500

Resultado para lista de 9 elementos, para el número: 39 :1+8*5-4/2

 Tiempo de ejecución para algoritmo: 0.000107526779174804687500

Resultado para lista de 9 elementos, para el número: 40 :2+8*5-6/3

 Tiempo de ejecución para algoritmo: 0.000245809555053710937500

Resultado para lista de 9 elementos, para el número: 41 :1+7*6-4/2

 Tiempo de ejecución para algoritmo: 0.000233888626098632812500

Resultado para lista de 9 elementos, para el número: 42 :2+7*6-8/4

 Tiempo de ejecución para algoritmo: 0.000441551208496093750000

Resultado para lista de 9 elementos, para el número: 43 :3+7*6-2/1

 Tiempo de ejecución para algoritmo: 0.000446557998657226562500

Resultado para lista de 9 elementos, para el número: 44 :1+9*5-4/2

 Tiempo de ejecución para algoritmo: 0.000121831893920898437500

Resultado para lista de 9 elementos, para el número: 45 :2+9*5-6/3

 Tiempo de ejecución para algoritmo: 0.000241994857788085937500

Resultado para lista de 9 elementos, para el número: 46 :3+9*5-2/1

 Tiempo de ejecución para algoritmo: 0.000657558441162109375000

Resultado para lista de 9 elementos, para el número: 47 :1+8*6-4/2

```
Resultado para lista de 9 elementos, para el número: 47 :1+8*6-4/2
 Tiempo de ejecución para algoritmo: 0.000276327133178710937500 0
Resultado para lista de 9 elementos, para el número: 48 :6+9*5-3/1
 Tiempo de ejecución para algoritmo: 0.003728151321411132812500 0
Resultado para lista de 9 elementos, para el número: 49 :3+8*6-2/1
 Tiempo de ejecución para algoritmo: 0.000407695770263671875000 0
Resultado para lista de 9 elementos, para el número: 50 :4+8*6-2/1
 Tiempo de ejecución para algoritmo: 0.000556468963623046875000 0
Resultado para lista de 9 elementos, para el número: 51 :1+9*6-8/2
 Tiempo de ejecución para algoritmo: 0.000322580337524414062500 0
Resultado para lista de 9 elementos, para el número: 52 :2+9*6-4/1
 Tiempo de ejecución para algoritmo: 0.000215768814086914062500 0
Resultado para lista de 9 elementos, para el número: 53 :1+9*6-4/2
 Tiempo de ejecución para algoritmo: 0.000105857849121093750000 0
Resultado para lista de 9 elementos, para el número: 54 :2+9*6-8/4
 Tiempo de ejecución para algoritmo: 0.000393390655517578125000 0
Resultado para lista de 9 elementos, para el número: 55 :1+8*7-4/2
 Tiempo de ejecución para algoritmo: 0.000118017196655273437500 0
Resultado para lista de 9 elementos, para el número: 56 :2+8*7-6/3
 Tiempo de ejecución para algoritmo: 0.000369548797607421875000 0
Resultado para lista de 9 elementos, para el número: 57 :3+8*7-2/1
```

```
Resultado para lista de 9 elementos, para el número: 57 :3+8*7-2/1
 Tiempo de ejecución para algoritmo: 0.000484466552734375000000
Resultado para lista de 9 elementos, para el número: 58 :4+8*7-2/1
 Tiempo de ejecución para algoritmo: 0.000425815582275390625000
Resultado para lista de 9 elementos, para el número: 59 :5+8*7-2/1
 Tiempo de ejecución para algoritmo: 0.000569581985473632812500
Resultado para lista de 9 elementos, para el número: 60 :1+9*7-8/2
 Tiempo de ejecución para algoritmo: 0.000166416168212890625000
Resultado para lista de 9 elementos, para el número: 61 :1+9*7-6/2
 Tiempo de ejecución para algoritmo: 0.000135183334350585937500
Resultado para lista de 9 elementos, para el número: 62 :1+9*7-4/2
 Tiempo de ejecución para algoritmo: 0.000117301940917968750000
Resultado para lista de 9 elementos, para el número: 63 :2+9*7-6/3
 Tiempo de ejecución para algoritmo: 0.000314474105834960937500
Resultado para lista de 9 elementos, para el número: 64 :3+9*7-2/1
 Tiempo de ejecución para algoritmo: 0.000429630279541015625000
Resultado para lista de 9 elementos, para el número: 65 :4+9*7-2/1
 Tiempo de ejecución para algoritmo: 0.000445365905761718750000
Resultado para lista de 9 elementos, para el número: 66 :5+9*7-2/1
 Tiempo de ejecución para algoritmo: 0.000581026077270507812500
Resultado para lista de 9 elementos, para el número: 67 :6+9*7-2/1
```

```
Resultado para lista de 9 elementos, para el numero: 67 :6+9*7-2/1
 Tiempo de ejecución para algoritmo: 0.000916481018066406250000
Resultado para lista de 9 elementos, para el número: 68 :2+9*8-6/1
 Tiempo de ejecución para algoritmo: 0.000289678573608398437500
Resultado para lista de 9 elementos, para el número: 69 :2+9*8-5/1
 Tiempo de ejecución para algoritmo: 0.000425338745117187500000
Resultado para lista de 9 elementos, para el número: 70 :1+9*8-6/2
 Tiempo de ejecución para algoritmo: 0.000134706497192382812500
Resultado para lista de 9 elementos, para el número: 71 :1+9*8-4/2
 Tiempo de ejecución para algoritmo: 0.000291109085083007812500
Resultado para lista de 9 elementos, para el número: 72 :2+9*8-6/3
 Tiempo de ejecución para algoritmo: 0.000410318374633789062500
Resultado para lista de 9 elementos, para el número: 73 :3+9*8-2/1
 Tiempo de ejecución para algoritmo: 0.000489234924316406250000
Resultado para lista de 9 elementos, para el número: 74 :4+9*8-2/1
 Tiempo de ejecución para algoritmo: 0.000550985336303710937500
Resultado para lista de 9 elementos, para el número: 75 :5+9*8-2/1
 Tiempo de ejecución para algoritmo: 0.000712394714355468750000
Resultado para lista de 9 elementos, para el número: 76 :6+9*8-2/1
 Tiempo de ejecución para algoritmo: 0.000791311264038085937500
```

```
Resultado para lista de 9 elementos, para el numero: // :7+9*8-2/1

 Tiempo de ejecución para algoritmo: 0.000950574874877929687500000
```

In [32]:
```
#Cálculo de todos los números posibles para la lista aleatoria de números generada
for i in lista2:
  calculoSimplePintar(i,numerosListaGrande)
```

```
Resultado para lista de 2355 elementos, para el número: 1570 :2+1570*1-6/3

 Tiempo de ejecución para algoritmo: 0.00186729431152343750000000

Resultado para lista de 2355 elementos, para el número: 1564 :2+1564*1-6/3

 Tiempo de ejecución para algoritmo: 0.00129556655883789062500000

Resultado para lista de 2355 elementos, para el número: 1018 :2+1018*1-6/3

 Tiempo de ejecución para algoritmo: 0.00167679786682128906250000

Resultado para lista de 2355 elementos, para el número: 661 :2+661*1-6/3

 Tiempo de ejecución para algoritmo: 0.00148415565490722656250000

Resultado para lista de 2355 elementos, para el número: 336 :2+336*1-6/3

 Tiempo de ejecución para algoritmo: 0.00166273117065429687500000

Resultado para lista de 2355 elementos, para el número: 1781 :2+1781*1-6/3

 Tiempo de ejecución para algoritmo: 0.00191831588745117187500000

Resultado para lista de 2355 elementos, para el número: 651 :2+651*1-6/3

 Tiempo de ejecución para algoritmo: 0.00181007385253906250000000

Resultado para lista de 2355 elementos, para el número: 1788 :2+1788*1-6/3

 Tiempo de ejecución para algoritmo: 0.00091433525085449218750000
```

```
Resultado para lista de 2355 elementos, para el número: 1632 :2+1632*1-6/3

 Tiempo de ejecución para algoritmo: 0.000816345214843750000000

Resultado para lista de 2355 elementos, para el número: 732 :2+732*1-6/3

 Tiempo de ejecución para algoritmo: 0.000828742980957031250000
```

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

Respuesta

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

Se podría avanzar en el estudio de este problema en los siguientes aspectos:

- Obteniendo no una solución, sino todas las posibles
- Agregando más operadores, ejemplo: raíces cuadradas y potencias
- Eliminando restricciones al problema:
  - Con números que se puedan repetir
  - Con operadores que se puedan repetir
  - Con un rango numérico más amplio (no sólo 1 a 9)

  Este último aspecto (rango numérico más amplio) ya se ha tenido en cuenta en todos los algoritmos que he implementado.

  A continuación, voy a mostrar los resultados obtenidos de utilizar dichos algoritmos para distintas entradas de datos. Se va a poder observar cómo el algoritmo implementado es muy superior a los dos algoritmos de fuerza bruta.

In [33]:
```
#En el algoritmo que he implementado, ya contemplo rangos numéricos más amplios, no necesariame
nte desde 1 a 9
#Se muestran a continuación los resultados obtenidos para distintos números y diferente rango n
umérico, no sólo el tamaño sino para diferentes rangos
print("""""""" Algoritmo de calculaSimple""")
```

```
print("###### Algoritmo de calculoSimple")
lista=[x for x in range(1,5000)]
calculoSimplePintar(232,lista)

lista=[x for x in range(200,5000)]
calculoSimplePintar(232,lista)

lista=[x for x in range(23,544)]
calculoSimplePintar(232,lista)

print("\r\n ###### Algoritmo de fuerza bruta")
#El primer algoritmo por fuerza bruta tarda muchísimo para listas grandes, se queda colgado. Pr
uebo con una lista de 20 elementos.:
#obtenerEnteroFBRestPintar(232,lista,OPERACIONES)
obtenerEnteroFBPintar(232,[x for x in range(1,20)],OPERADORES)
print("\r\n ###### Algoritmo de fuerza bruta con restricciones")
#El segundo también se queda colgado con listas tan grandes, aun así, responde  mejor para la l
ista de 20 elementos
#obtenerEnteroFBRestPintar(232,lista,["OPERACIONES"])
obtenerEnteroFBRestPintar(232,[x for x in range(1,20)],OPERADORES)
obtenerEnteroFBRestPintar(232,[x for x in range(1,100)],OPERADORES)
```

```
###### Algoritmo de calculoSimple
Resultado para lista de 4999 elementos, para el número: 232 :2+232*1-6/3

 Tiempo de ejecución para algoritmo: 0.0021996498107910156250000

Resultado para lista de 4800 elementos, para el número: 232 :200+1*232-400/2

 Tiempo de ejecución para algoritmo: 0.1092288494110107421875000

Resultado para lista de 521 elementos, para el número: 232 :23+8*29-46/2

 Tiempo de ejecución para algoritmo: 0.0025343894958496093750000


 ###### Algoritmo de fuerza bruta
Resultado para lista de 19 elementos, para el número: 232 :1+13*18-6/2

 Tiempo de ejecución para algoritmo: 2.5065202713012695312500000
```

```
 ###### Algoritmo de fuerza bruta con restricciones
Resultado para lista de 19 elementos, para el número: 232 :1+13*18-6/2

 Tiempo de ejecución para algoritmo: 0.435221910476684570312500

Resultado para lista de 99 elementos, para el número: 232 :1+3*78-6/2

 Tiempo de ejecución para algoritmo: 13.797085523605346679687500
```

GRÁFICOS DE RENDIMIENTO

In [34]:
```
#Tamaños diversos de listas, de 50 en 50 elementos
listaGrafico=[9,59,109,159,209]

GUARDAR_RESULTADO=True
TIEMPOS=[]

#Fuerza bruta
print("\r\n Fuerza bruta:")
for i in(listaGrafico):
  obtenerEnteroFBPintar(45,[x for x in range(1,i+1)],OPERADORES)
tiemposFuerzaBruta=[]
for j in TIEMPOS:
  tiemposFuerzaBruta.append(j)
TIEMPOS=[]

#Fuerza bruta restricciones
print("\r\n Fuerza bruta con restricciones:")
for i in(listaGrafico):
  obtenerEnteroFBRestPintar(45,[x for x in range(1,i+1)],OPERADORES)
tiemposFuerzaBrutaR=[]
for k in TIEMPOS:
  tiemposFuerzaBrutaR.append(k)
TIEMPOS=[]


#Cálculo simple
```

```
print("Cálculo simple:")
for i in(listaGrafico):
  calculoSimplePintar(45,[x for x in range(1,i+1)])
tiemposCalculoSimple=[]
for l in TIEMPOS:
  tiemposCalculoSimple.append(l)
TIEMPOS=[]

GUARDAR_RESULTADO=False
```

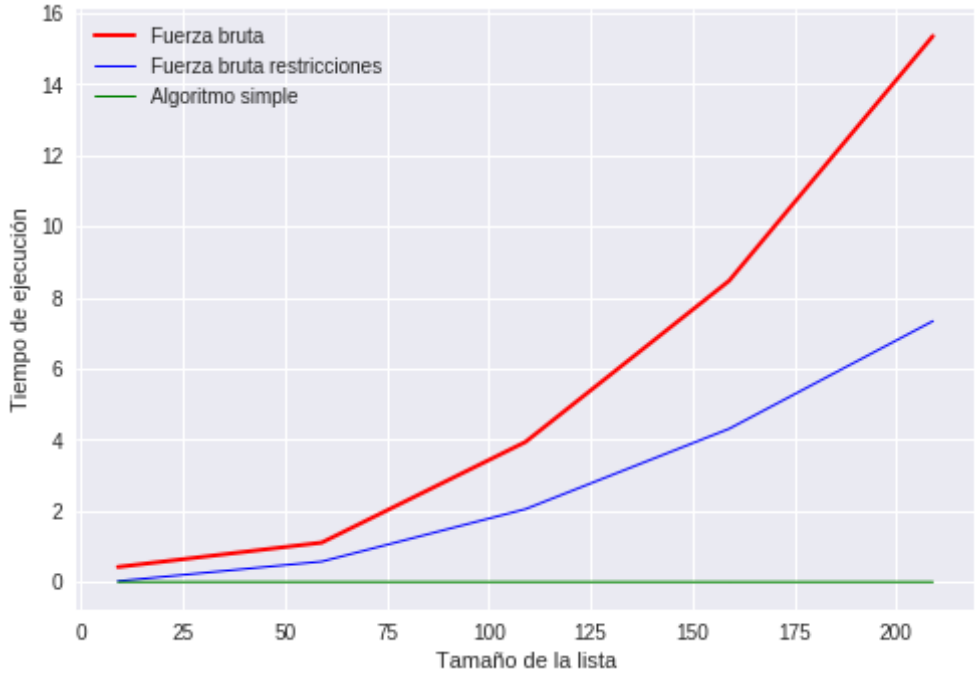
```
 Fuerza bruta:
Resultado para lista de 9 elementos, para el número: 45 :2+5*9-6/3

 Tiempo de ejecución para algoritmo: 0.422111034393310546875000

Resultado para lista de 59 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 1.107134819030761718750000

Resultado para lista de 109 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 3.937138557434082031250000

Resultado para lista de 159 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 8.472073554992675781250000

Resultado para lista de 209 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 15.355464220046997070312500


 Fuerza bruta con restricciones:
Resultado para lista de 9 elementos, para el número: 45 :2+5*9-6/3

 Tiempo de ejecución para algoritmo: 0.023424863815307617187500

Resultado para lista de 59 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 0.575208425521850585937500
```

```
Resultado para lista de 109 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 2.0495519638061523437500000

Resultado para lista de 159 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 4.3091304302215576171875000

Resultado para lista de 209 elementos, para el número: 45 :1+2*23-6/3

 Tiempo de ejecución para algoritmo: 7.3381392955780029296875000

Cálculo simple:
Resultado para lista de 9 elementos, para el número: 45 :2+9*5-6/3

 Tiempo de ejecución para algoritmo: 0.0004756450653076171875000

Resultado para lista de 59 elementos, para el número: 45 :2+45*1-6/3

 Tiempo de ejecución para algoritmo: 0.0003216266632080078125000

Resultado para lista de 109 elementos, para el número: 45 :2+45*1-6/3

 Tiempo de ejecución para algoritmo: 0.0002908706665039062500000

Resultado para lista de 159 elementos, para el número: 45 :2+45*1-6/3

 Tiempo de ejecución para algoritmo: 0.0001599788665771484375000

Resultado para lista de 209 elementos, para el número: 45 :2+45*1-6/3

 Tiempo de ejecución para algoritmo: 0.0004165172576904296875000
```

Gráficos de rendimiento para los tres algoritmos

In [35]:
```
import matplotlib.pyplot as plt
fig,axis = plt.subplots(1,1)
```

```
axis.plot(listaGrafico,tiemposFuerzaBruta,'r',label='Fuerza bruta',linewidth=2)
axis.plot(listaGrafico,tiemposFuerzaBrutaR,'b',label='Fuerza bruta restricciones',linewidth=1)
axis.plot(listaGrafico,tiemposCalculoSimple,'g',label='Algoritmo simple',linewidth=1)
axis.legend(loc='best')
axis.set_xlabel('Tamaño de la lista')
axis.set_ylabel('Tiempo de ejecución')
plt.show() # Se muestra la figura
```



Tal y como se aprecia, a medida que aumenta el tamaño de la lista, los tiempos para fuerza bruta y fuerza bruta con restricciones, crecen mucho más que para el algoritmo simple. Esto es debido, tal y como se ha explicado en apartaodos anteriores, a su menor complejidad y menor número de operaciones

Gráfico de rendimiento para algoritmo calculoSimple

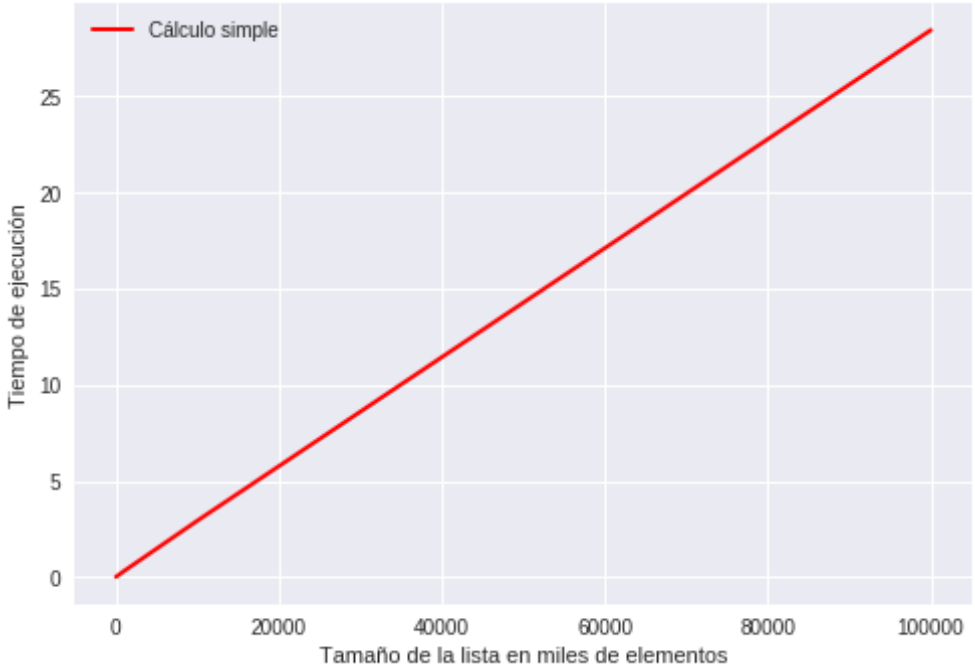
In [36]:
```
TIEMPOS=[]
GUARDAR_RESULTADO=True
tiemposCalculoSimpleT=[]
```

```
calculoSimplePintar(18,[x for x in range(1,10000)])
calculoSimplePintar(180,[x for x in range(1,100000)])
calculoSimplePintar(1809,[x for x in range(1,1000000)])
calculoSimplePintar(18097,[x for x in range(1,10000000)])
calculoSimplePintar(180978,[x for x in range(1,100000000)])

for t in TIEMPOS:
  tiemposCalculoSimpleT.append(t)

TIEMPOS=[]
GUARDAR_RESULTADO=False
```

```
Resultado para lista de 9999 elementos, para el número: 18 :2+18*1-6/3

 Tiempo de ejecución para algoritmo: 0.00545978546142578125000000

Resultado para lista de 99999 elementos, para el número: 180 :2+180*1-6/3

 Tiempo de ejecución para algoritmo: 0.02889657020568847656250000

Resultado para lista de 999999 elementos, para el número: 1809 :2+1809*1-6/3

 Tiempo de ejecución para algoritmo: 0.30420088768005371093750000

Resultado para lista de 9999999 elementos, para el número: 18097 :2+18097*1-6/3

 Tiempo de ejecución para algoritmo: 2.92697548866271972656250000

Resultado para lista de 99999999 elementos, para el número: 180978 :2+180978*1-6/3

 Tiempo de ejecución para algoritmo: 28.46446442604064941406250000
```

In [37]:
```
import matplotlib.pyplot as plt
fig,axis = plt.subplots(1,1)
axis.plot([10000/1000,100000/1000,1000000/1000,10000000/1000,100000000/1000],tiemposCalculoSimp
leT,'r',label='Cálculo simple',linewidth=2)
axis.legend(loc='best')
axis.set_xlabel('Tamaño de la lista en miles de elementos')
axis.set_ylabel('Tiempo de ejecución')
```

```
plt.show() # Se muestra la figura
```



Se ha hecho un gráfico aparte para calculo simple para tamaños más grandes de listas. El gráfico está puesto en miles de elementos. Se aprecia un crecimiento lineal, a medida que el número a calcular y el listado de números es mayor.