```
import numpy as np
```

```
!pip install ortools
```

```
Collecting ortools
  Downloading https://files.pythonhosted.org/packages/db/8f/7c099bcedd55df8f215ba42b5
     |████████████████████████████████| 27.9MB 148kB/s
Collecting protobuf>=3.11.2
  Downloading https://files.pythonhosted.org/packages/57/02/5432412c162989260fab61fa6
     |████████████████████████████████| 1.3MB 39.8MB/s
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
Installing collected packages: protobuf, ortools
  Found existing installation: protobuf 3.10.0
    Uninstalling protobuf-3.10.0:
      Successfully uninstalled protobuf-3.10.0
Successfully installed ortools-7.5.7466 protobuf-3.11.3
WARNING: The following packages were previously imported in this runtime:
  [google]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

Lendo arquivo

```
path = 'Problema.txt'
arquivo = open(path,'r')
numeros = []

for linha  in arquivo:
  linha = linha.strip()
  numeros.append(linha)

arquivo.close()
numeros
```

```
['4 3', '50 60 130 30', '5 5 1 4 5', '9 5 17 20 9', '10 10 12 18 1']
```

```
x1 = numeros[0].split(' ') #Variaveis e restricoes
c = numeros[1].split(' ') #Coecicientes das variaveis na funcao objetivo
```

```
rest = len(numeros)-2 #Numeros de restrições
var = int(x1[0]) #Números de variáveis
```

```
a = [0]*rest #inicializando array com o numero de restricoes
b = [0]*rest

for i in range(2, len(numeros)): #começando em dois pois oq vem depois da linha 2 sao a
    aa = numeros[i].split(' ')
    b[i-2] = aa[len(aa)-1]
```

```
    del(aa[len(aa)-1])
    a[i-2] = aa
print(a, b, c)
```

```
[['5', '5', '1', '4'], ['9', '5', '17', '20'], ['10', '10', '12', '18']] ['5', '9', '
```

```
a = np.double( a )
b = np.double( b )
c = np.double( c )

print(a, b, c)
```

```
[[ 5.  5.  1.  4.]
 [ 9.  5. 17. 20.]
 [10. 10. 12. 18.]] [5. 9. 1.] [ 50.  60. 130.  30.]
```

```

obj= 'Min'

igualdade = 'MoreOrEqual'
```

Adicionando no modelo

```
def create_data_model(A, B, C, num_vars, num_rest):
  data = {}
  data['constraint_coeffs'] = A
  data['bounds'] = B
  data['obj_coeffs'] = C
  data['num_vars'] = num_vars
  data['num_constraints'] = num_rest
  return data
```

```
data = create_data_model(a, b, c, var, rest)
```

```
from ortools.linear_solver import pywraplp
solver = pywraplp.Solver('simple_mip_program', pywraplp.Solver.CBC_MIXED_INTEGER_PROGRA

infinity = solver.infinity()
x = {}

for j in range(data['num_vars']):
    x[j] = solver.NumVar(0, 1, 'x[%i]' % j) #Variáveis que pertencem ao conjunto entre

print("\n==== Solução inicial ====\n ")
print('\n\nNumero de variaveis =', solver.NumVariables())

if igualdade == 'MoreOrEqual':
  for i in range(data['num_constraints'] ):
    constraint = solver.RowConstraint(data['bounds'][i], infinity, '')#limite inferior,
    for j in range(data['num_vars']):
```

```
      constraint.SetCoefficient(x[j], data['constraint_coeffs'][i][j])

print('Numero de restriçoes =', solver.NumConstraints())

objective = solver.Objective()

for j in range(data['num_vars']):
    objective.SetCoefficient(x[j], data['obj_coeffs'][j])

if obj == 'Max':
  objective.SetMaximization() #Problema de maximização
  status = solver.Solve()

if obj == 'Min':
  objective.SetMinimization() #Problema de minimização
  status = solver.Solve()

solution_value = []
if status == pywraplp.Solver.OPTIMAL:
    print('\nValor ótimo = ', solver.Objective().Value())
    for j in range(data['num_vars']):
        print(x[j].name(), ' = ', x[j].solution_value())
        solution_value.append(x[j].solution_value())
    print()
    print('Problema resolvido em %f milliseconds' % solver.wall_time())
    print('Problema resolvido em %d nós' % solver.nodes())
else:
    print('Nao tem solucao otima.')

solution = (solution_value, solver.Objective().Value())
```

```
==== Solução inicial ====



Numero de variaveis = 4
Numero de restriçoes = 3

Valor ótimo =  40.0
x[0]  =  0.19999999999999996
x[1]  =  0.0
x[2]  =  0.0
x[3]  =  1.0

Problema resolvido em 3.000000 milliseconds
Problema resolvido em 0 nós
```

Função para resolver cada restrição

```
from __future__ import print_function
from ortools.linear_solver import pywraplp

def main (Data, index, b): #Main para fazer o branch
```

```
 5
 6  solver = pywraplp.Solver('simple_mip_program',
 7                           pywraplp.Solver.CBC_MIXED_INTEGER_PROGRAMMING)
 8
 9  infinity = solver.infinity()
10  x = {}
11
12  for j in range(Data['num_vars']):
13      x[j] = solver.NumVar(0, 1, 'x[%i]' % j) #Variáveis positivas entre 0 e 1
14
15  print("\n==== Solução ====\n ")
16  print('\n\nNumero de variaveis =', solver.NumVariables())
17
18  for i in range(Data['num_constraints'] - 1 ): # -1 para nao add a ultima rest
19    constraint = solver.RowConstraint(Data['bounds'][i], infinity, '')#limite inferior,
20    for j in range(Data['num_vars']):
21      constraint.SetCoefficient(x[j], Data['constraint_coeffs'][i][j])
22
23  y = solver.NumVar(0, 1, 'y')
24
25  y = x[index]
26
27  solver.Add(y == b)
28
29  print('Numero de restriçoes =', solver.NumConstraints())
30
31  objective = solver.Objective()
32
33  for j in range(Data['num_vars']):
34      objective.SetCoefficient(x[j], Data['obj_coeffs'][j])
35
36  if obj == 'Max':
37    objective.SetMaximization() #Problema de maximização
38    status = solver.Solve()
39
40  if obj == 'Min':
41    objective.SetMinimization() #Problema de minimização
42    status = solver.Solve()
43
44  solution_value = []
45  if status == pywraplp.Solver.OPTIMAL:
46      print('\nValor ótimo = ', solver.Objective().Value())
47      for j in range(Data['num_vars']):
48          print(x[j].name(), ' = ', x[j].solution_value())
49          solution_value.append(x[j].solution_value())
50      print()
51      print('Problema resolvido em %f milliseconds' % solver.wall_time())
52      print('Problema resolvido em %d nós' % solver.nodes())
53  else:
54      print('Nao tem solucao otima.')
55
56
57  return (solution_value, solver.Objective().Value())
```

Adicionar uma restrição no modelo

```
def append_data_model (Data, index, b): #Adiciona ao modelo uma restrição
    if index == -1:
      return Data

    A = Data['constraint_coeffs']
    B = Data['bounds']
    C = Data['obj_coeffs']
    num_vars = Data['num_vars']
    num_rest = Data['num_constraints'] + 1

    data = {} #Criando outro data para nao alterar oq vir do parametro
    data['constraint_coeffs'] = A
    data['bounds'] = B
    data['obj_coeffs'] = C
    data['num_vars'] = num_vars
    data['num_constraints'] = num_rest

    #Dar um append na restrição indicando qual coeficiente do x é, se for x1, o array é

    var_x = [0]*data['num_vars']
    x=[]

    for i in range(0, data['num_vars']):
      if i == index:
        var_x[index] = 1
        x.append(var_x)
        data['constraint_coeffs'] = np.append(A, x, axis=0)

    #Dar append no lado direito da igualdade
    data['bounds'] = np.append(B, b)

    return data
```

Exemplo de quemo fica o modelo adicionando as restrições

```
= append_data_model(data, 2, 0) #modelo, index da variavel que sera relaxada e o lado d
= append_data_model(data, 2, 1)
```

```
print(data1)
print(data2)
```

```
{'constraint_coeffs': array([[ 5.,  5.,  1.,  4.],
       [ 9.,  5., 17., 20.],
       [10., 10., 12., 18.],
       [ 0.,  0.,  1.,  0.]]), 'bounds': array([5., 9., 1., 0.]), 'obj_coeffs': array
{'constraint_coeffs': array([[ 5.,  5.,  1.,  4.],
       [ 9.,  5., 17., 20.],
       [10., 10., 12., 18.],
       [ 0.,  0.,  1.,  0.]]), 'bounds': array([5., 9., 1., 1.]), 'obj_coeffs': array
```

```
main(data1, 2, 0)
main(data2, 2, 1)
```

```
==== Solução ====



Numero de variaveis = 4
Numero de restriçoes = 4

Valor ótimo =  40.0
x[0]  =  0.19999999999999996
x[1]  =  0.0
x[2]  =  0.0
x[3]  =  1.0

Problema resolvido em 5.000000 milliseconds
Problema resolvido em 0 nós

==== Solução ====



Numero de variaveis = 4
Numero de restriçoes = 4

Valor ótimo =  160.0
x[0]  =  0.0
x[1]  =  0.0
x[2]  =  1.0
x[3]  =  1.0

Problema resolvido em 4.000000 milliseconds
Problema resolvido em 0 nós
([0.0, 0.0, 1.0, 1.0], 160.0)
```

Árvore

```
class Tree(object):
    def __init__(self, solution, modelo, left=None, right=None):
        self.solution = solution
        self.modelo = modelo
        self.left = left
        self.right = right
```

Verificar se a variável é um número inteiro

```
def isInt (string):
  t = True
  for i in range(len(string)):
    if string[i] == '.':

      j=i+1
      if j >=len(string):
        return t
      for j in range(j,len(string)):
        if int(string[j]) != 0:
          t = False
  return t
```

Verificar integralidade e a menor distância entre as variáveis

- O valor abs da diminuição com o valor da variável
- retorna o indice da variável (para poder adicionar depois na restrição)
- se retornar -1 quer dizer q todos os valores são inteiros

```
def Verifica_integralidade(Solution):
    episilon = 10e3; #colocando um valor grande para iniciar
    Index_Da_menor_dist = -1  # Se for -1 todos os x são inteiros

    # precisa verificar se os valores em x são inteiros e verificar se está próximo de

    for i in range(len(Solution)): #menor que 1 pois o ultimo é o valor de z
      if isInt(str(Solution[i])) == False: #Se for false, quer dizer que tem dígito
        distancia = abs(Solution[i] - 0.5)

        if distancia < episilon: #Se esse valor for menor do que oq já tem no episilon,
          episilon = distancia
          Index_Da_menor_dist = i

    return Index_Da_menor_dist
```

Pilha

- Na pilha é colocado a solução do dual e primal
- No momento da inserção da solução na pilha, precisa verificar:

```
- Se tiver vazia, só insere
- Se não tiver vazia, verifica se a solução é inteira (integralidade) e se a solu
```

```
!pip install pythonds
```

```
Collecting pythonds
  Downloading https://files.pythonhosted.org/packages/d5/23/3a6d8983605ba23ca972754a6
Installing collected packages: pythonds
Successfully installed pythonds-1.2.1
```

```
from pythonds.basic.stack import Stack
menosInfinity = -9999
maisInfinity = 9999

class Pilha:
  def __init__(self):
    self.items = [[[], menosInfinity], [[], maisInfinity]] #DUAL e PRIMAL minimização

  def isEmpty(self):
    return self.items == [[[], menosInfinity], [[], maisInfinity]]

  def push(self, item):

    vars = item[0]
    tamVars = len(vars)

    vAdd = item[1]
    dual = pilha.see()[0]
    primal = pilha.see()[1]

    if self.isEmpty() != True: #Se não tiver vazia
      t = -1
      for i in range(0, tamVars): #percorrer os x's
        if isInt(str(vars[i])) != True: #se tiver um NAO inteiro
          t = 0
          break
      if t == 0 and vAdd >= dual[1] and vAdd != 0 or dual[1] == maisInfinity: #se o val
        self.items.append(item)
        self.items.append(primal)

      if t == -1 and vAdd <= primal[1] and vAdd != 0 or primal[1] == menosInfinity: #se
        self.items.append(dual)
        self.items.append(item)

    elif self.isEmpty() == True: #Se tiver vazia, adiciona
      self.items.append(item)
      self.items.append(primal)

  def pop(self):
    return self.items.pop()

  def see(self):
    return (self.items[len(self.items)-2], self.items[len(self.items)-1])

pilha = Pilha()
```

Exemplo como fica a pilha

```
pilha.see()
```

```
([[], -9999], [[], 9999])
```

```
pilha.push([[0.75, 1.0, 0.0], 13.75]) #Adicioanndo um x's nao inteiros, adiciona no dua
print(pilha.see())
```

```
([[0.75, 1.0, 0.0], 13.75], [[], 9999])
```

```
pilha.push([[1.0, 0.5000000000000001, 0.2499999999999999], 12.0]) #, caso a solução nac
print(pilha.see())
```

```
([[0.75, 1.0, 0.0], 13.75], [[], 9999])
```

```
pilha.push([[0.5, 1, 18], 17]) #se a solucao do dual for melhor, adiciona
print(pilha.see())
```

```
([[0.5, 1, 18], 17], [[], 9999])
```

```
pilha.push([[1, 1, 1], 5]) #se for com criterio de integralidade, adiciona no primal
print(pilha.see())
```

```
([[0.5, 1, 18], 17], [[1, 1, 1], 5])
```

```
pilha.push([[1, 1, 1], 18]) #se for maior nao adiciona no primal
print(pilha.see())
```

```
([[0.5, 1, 18], 17], [[1, 1, 1], 5])
```

```
pilha.push([[1, 1, 1], 1]) #se for menor adiciona no primal
print(pilha.see())
```

```
([[0.5, 1, 18], 17], [[1, 1, 1], 1])
```

```
pilha = Pilha() #reinicializando a pilha
print(pilha.see())
```

```
([[], -9999], [[], 9999])
```

Adicionando lado direito e esquerdo da árvore

```
def AddArv(arv, data_left, data_right, index):
  solution_right = main(data_right, index, 1 )
  solution_left  = main(data_left, index, 0 )

```

```
  arv.left  = Tree( solution_left, data_left  )
  arv.right  = Tree( solution_right, data_right )

  pilha.push(solution_right)
  pilha.push(solution_left )

  print("\nNó pai\n ", arv.solution)
  print("\nLADO ESQUERDO:\n", arv.left.solution)
  print("LADO DIREITO: \n", arv.right.solution)

  print("Dual: %f Primal: %f\n" % (pilha.see()[0][1], pilha.see()[1][1]))
```

Percorrendo a arvore

```
no = 0
def Profun (tree, no):
  no = no +1
  print("Nó %d" % (no))

  primal = pilha.see()[1][1]
  dual = pilha.see()[0][1]

  index = Verifica_integralidade(tree.solution[0])
  print(tree.solution[1])
  print(primal)
  print(dual)


  b_limite_para_cima  = ("%0.f") % tree.solution[0][index] #lado arredondado. se for 0.
  b_limite_para_baixo = ("%d"  ) % tree.solution[0][index] #lado inteiro. se for 0.5 fi

  data_left  = append_data_model(tree.modelo, int(index), int(b_limite_para_baixo ))
  data_right = append_data_model(tree.modelo, int(index), int(b_limite_para_cima ))

  AddArv(tree, data_left, data_right, index) #Adiciona o lado esquerdo e direito na arv

  index_left = Verifica_integralidade(tree.left.solution[0])
  index_right = Verifica_integralidade(tree.right.solution[0])

  #Se o valor da solução for menor que o primal e maior que o dual, aprofunda
  if index_left != -1 and tree.left.solution[1] < primal and tree.left.solution[1]  >=d
    Profun (tree.left, no)

  if index_right != -1 and tree.right.solution[1] < primal and tree.right.solution[1] >
    Profun (tree.right, no)
  else :
    return 0
```

```
tree = Tree(solution, data)
pilha.push(tree.solution)
print(pilha.see())
```

```
print("======== Iniciando a árvore =========\n")
# Profun(tree)
Profun(tree, no)
```

```
(([0.19999999999999996, 0.0, 0.0, 1.0], 40.0), [[], 9999])
======== Iniciando a árvore =========

Nó 1
40.0
9999
40.0

==== Solução ====



Numero de variaveis = 4
Numero de restriçoes = 4

Valor ótimo =  50.0
x[0]  =  1.0
x[1]  =  0.0
x[2]  =  0.0
x[3]  =  0.0

Problema resolvido em 5.000000 milliseconds
Problema resolvido em 0 nós

==== Solução ====



Numero de variaveis = 4
Numero de restriçoes = 4

Valor ótimo =  42.0
x[0]  =  0.0
x[1]  =  0.19999999999999996
x[2]  =  0.0
x[3]  =  1.0

Problema resolvido em 4.000000 milliseconds
Problema resolvido em 0 nós

Nó pai
  ([0.19999999999999996, 0.0, 0.0, 1.0], 40.0)

LADO ESQUERDO:
 ([0.0, 0.19999999999999996, 0.0, 1.0], 42.0)
LADO DIREITO:
 ([1.0, 0.0, 0.0, 0.0], 50.0)
Dual: 42.000000 Primal: 50.000000

Nó 2
42.0
50.0
42.0

==== Solução ====



Numero de variaveis = 4
Numero de restriçoes = 5
```