```
import numpy as np
```

```
!pip install ortools
```

```
Requirement already satisfied: ortools in /usr/local/lib/python3.6/dist-packages (7.5
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: protobuf>=3.11.2 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
```

Lendo arquivo

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

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

arquivo.close()
numeros
```

```
['3 2', '5 10 8', '3 5 2 6', '4 4 4 7']
```

```
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)
```

```
[['3', '5', '2'], ['4', '4', '4']] ['6', '7'] ['5', '10', '8']
```

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

print(a, b, c)
```

```
[[3. 5. 2.]
 [4. 4. 4.]] [6. 7.] [ 5. 10.  8.]
```

```
Igualdade = input("Digite 1 se as restrições forem menores ou iguais ou 2 se forem maic
Obj = input("Digite 1 se a funcao objetivo é de maximização ou 2 se for de minimização"

if Obj == 1:
  obj = 'Max'
else:
  obj= 'Min'

if Igualdade == 1:
  igualdade = 'LessOrEqual'
else:
  igualdade = 'MoreOrEqual'
```

```
Digite 1 se as restrições forem menores ou iguais ou 2 se forem maiores ou iguais2
Digite 1 se a funcao objetivo é de maximização ou 2 se for de minimização2
```

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)
```

```
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'] ): # -1 para nao add a ultima rest
    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])

```

```
if igualdade == 'LessOrEqual':
  for i in range(data['num_constraints'] ):
    constraint = solver.RowConstraint(0, data['bounds'][i], '')
    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 = 3
Numero de restriçoes = 2

Valor ótimo =  12.0
x[0]  =  1.0
x[1]  =  0.5000000000000001
x[2]  =  0.2499999999999999

Problema resolvido em 4.000000 milliseconds
Problema resolvido em 0 nós
```

Função para resolver cada restrição

```
from ortools.linear_solver import pywraplp
```

```
 2
 3 def main (Data, igualdade): #Main para fazer o branch
 4
 5   solver = pywraplp.Solver('simple_mip_program', pywraplp.Solver.CBC_MIXED_INTEGER_PROG
 6
 7   infinity = solver.infinity()
 8   x = {}
 9
10   for j in range(Data['num_vars']):
11       x[j] = solver.NumVar(0, 1, 'x[%i]' % j) #Variáveis positivas entre 0 e 1
12
13   print("\n==== Solução ====\n ")
14   print('\n\nNumero de variaveis =', solver.NumVariables())
15
16   for i in range(Data['num_constraints'] - 1 ): # -1 para nao add a ultima rest
17     constraint = solver.RowConstraint(Data['bounds'][i], infinity, '')#limite inferior,
18     for j in range(Data['num_vars']):
19       constraint.SetCoefficient(x[j], Data['constraint_coeffs'][i][j])
20
21   n = Data['num_constraints'] - 1 # para adicionar a ultima restrição
22
23   if igualdade == 'MoreOrEqual':
24     constraint = solver.RowConstraint(Data['bounds'][n], infinity, '')
25     for j in range(Data['num_vars']):
26       constraint.SetCoefficient(x[j], Data['constraint_coeffs'][n][j])
27
28   if igualdade == 'LessOrEqual':
29     constraint = solver.RowConstraint(0, Data['bounds'][n], '')
30     for j in range(Data['num_vars']):
31       constraint.SetCoefficient(x[j], Data['constraint_coeffs'][n][j])
32
33   print('Numero de restriçoes =', solver.NumConstraints())
34
35   objective = solver.Objective()
36
37   for j in range(Data['num_vars']):
38       objective.SetCoefficient(x[j], Data['obj_coeffs'][j])
39
40   if obj == 'Max':
41     objective.SetMaximization() #Problema de maximização
42     status = solver.Solve()
43
44   if obj == 'Min':
45     objective.SetMinimization() #Problema de minimização
46     status = solver.Solve()
47
48   solution_value = []
49   if status == pywraplp.Solver.OPTIMAL:
50       print('\nValor ótimo = ', solver.Objective().Value())
51       for j in range(Data['num_vars']):
52           print(x[j].name(), ' = ', x[j].solution_value())
53           solution_value.append(x[j].solution_value())
54       print()
55       print('Problema resolvido em %f milliseconds' % solver.wall_time())
56       print('Problema resolvido em %d nós' % solver.nodes())
```

```
  else:
      print('Nao tem solucao otima.')

  return (solution_value, solver.Objective().Value())
```

Adicionar uma restrição no modelo

```
def append_data_model (Data, x, b): #Adiciona ao modelo uma restrição
    if x == -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
    x0 = np.array([[1, 0, 0]])
    x1 = np.array([[0, 1, 0]])
    x2 = np.array([[0, 0, 1]])

    if x == 0: #x indica qual x's é na restric]ção
      data['constraint_coeffs'] = np.append(A, x0, axis=0)
    if x == 1:
      data['constraint_coeffs'] = np.append(A, x1, axis=0)
    if x == 2:
      data['constraint_coeffs'] = np.append(A, x2, 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

```
data1 = append_data_model(data, 1, 0)
data2 = append_data_model(data, 1, 1)
```

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

```
{'constraint_coeffs': array([[3., 5., 2.],
       [4., 4., 4.],
       [0., 1., 0.]]), 'bounds': array([6., 7., 0.]), 'obj_coeffs': array([ 5., 10.,
{'constraint_coeffs': array([[3., 5., 2.],
       [4., 4., 4.],
       [0., 1., 0.]]), 'bounds': array([6., 7., 1.]), 'obj_coeffs': array([ 5., 10.,
```

```
main(data1, "LessOrEqual")
main(data2, "MoreOrEqual")
```

```
==== Solução ====


Numero de variaveis = 3
Numero de restriçoes = 3
Nao tem solucao otima.

==== Solução ====


Numero de variaveis = 3
Numero de restriçoes = 3

Valor ótimo =  13.75
x[0]  =  0.75
x[1]  =  1.0
x[2]  =  0.0

Problema resolvido em 4.000000 milliseconds
Problema resolvido em 0 nós
([0.75, 1.0, 0.0], 13.75)
```

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] == '.':
```

```
5
6       j=i+1
7       if j >=len(string):
8         return t
9       for j in range(j,len(string)):
10        if int(string[j]) != 0:
11          t = False
12    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

```
1 def Verifica_integralidade(Solution):
2     episilon = 10e3; #colocando um valor grande para iniciar
3     Index_Da_menor_dist = -1  # Se for -1 todos os x são inteiros
4 
5     # precisa verificar se os valores em x são inteiros e verificar se está próximo de
6 
7     for i in range(len(Solution)): #menor que 1 pois o ultimo é o valor de z
8       if isInt(str(Solution[i])) == False: #Se for false, quer dizer que tem dígito
9         distancia = abs(Solution[i] - 0.5)
10
11        if distancia < episilon: #Se esse valor for menor do que oq já tem no episilon,
12          episilon = distancia
13          Index_Da_menor_dist = i
14
15    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
```

```
1 !pip install pythonds
```

```
Requirement already satisfied: pythonds in /usr/local/lib/python3.6/dist-packages (1.
```

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

```
4
5 class Pilha:
6   def __init__(self):
7     self.items = [[[], menosInfinity], [[], maisInfinity]] #DUAL e PRIMAL minimização
8
9   def isEmpty(self):
10    return self.items == [[[], menosInfinity], [[], maisInfinity]]
11
12  def push(self, item):
13
14    vars = item[0]
15    tamVars = len(vars)
16
17    vAdd = item[1]
18    dual = pilha.see()[0]
19    primal = pilha.see()[1]
20
21    if self.isEmpty() != True: #Se não tiver vazia
22      t = -1
23      for i in range(0, tamVars): #percorrer os x's
24        if isInt(str(vars[i])) != True: #se tiver um NAO inteiro
25          t = 0
26          break
27      if t == 0 and vAdd >= dual[1] and vAdd != 0 or dual[1] == maisInfinity: #se o val
28        self.items.append(item)
29        self.items.append(primal)
30
31      if t == -1 and vAdd <= primal[1] and vAdd != 0 or primal[1] == menosInfinity: #se
32        self.items.append(dual)
33        self.items.append(item)
34
35    elif self.isEmpty() == True: #Se tiver vazia, adiciona
36      self.items.append(item)
37      self.items.append(primal)
38
39  def pop(self):
40    return self.items.pop()
41
42  def see(self):
43    return (self.items[len(self.items)-2], self.items[len(self.items)-1])
44
45 pilha = Pilha()
```

Exemplo como fica a pilha

```
1 pilha.see()
```

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

```
1 pilha.push([[0.75, 1.0, 0.0], 13.75]) #Adicioanndo um x's nao inteiros, adiciona no dua
2 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):
  solution_right = main(data_right, 'MoreOrEqual' )
  solution_left  = main(data_left, 'LessOrEqual' )

  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

```
def Profun (tree):
  primal = pilha.see()[1][1]
  dual = pilha.see()[0][1]

  index = Verifica_integralidade(tree.solution[0])

  if ( index != - 1 and tree.solution[1] != 0 and tree.solution[1] <= primal and tree.s

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

    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) #Adiciona o lado esquerdo e direito na arvore

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

    if index_left != -1 and tree.left.solution[1] <= primal and tree.left.solution[1] <
      Profun (tree.left)

    if index_right != -1 and tree.right.solution[1] <= primal and tree.right.solution[1
      Profun (tree.right)

  else :
    return 0
```

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

print("======== Iniciando a árvore =========\n")
Profun(tree)
Profun(tree)
```

```
==== Solução ====



Numero de variaveis = 3
Numero de restriçoes = 3
Nao tem solucao otima.

Nó pai
  ([1.0, 0.5000000000000001, 0.2499999999999999], 12.0)

LADO ESQUERDO:
 ([], 0.0)
LADO DIREITO:
 ([0.75, 1.0, 0.0], 13.75)
Dual: 13.750000 Primal: 23.000000


==== Solução ====



Numero de variaveis = 3
Numero de restriçoes = 4

Valor ótimo =  15.0
x[0]  =  1.0
x[1]  =  1.0
x[2]  =  0.0

Problema resolvido em 5.000000 milliseconds
Problema resolvido em 0 nós

==== Solução ====



Numero de variaveis = 3
Numero de restriçoes = 4

Valor ótimo =  16.0
x[0]  =  0.0
x[1]  =  1.0
x[2]  =  0.75

Problema resolvido em 3.000000 milliseconds
Problema resolvido em 0 nós

Nó pai
  ([0.75, 1.0, 0.0], 13.75)

LADO ESQUERDO:
 ([0.0, 1.0, 0.75], 16.0)
LADO DIREITO:
 ([1.0, 1.0, 0.0], 15.0)
Dual: 16.000000 Primal: 15.000000
```

```
solucao = pilha.see()
print("Solução que está na pilha = ")
print(solucao)
print()
```

```
Solução que está na pilha =
(([0.0, 1.0, 0.75], 16.0), ([1.0, 1.0, 0.0], 15.0))
```

```
print("Solução ótima: ")
x = solucao[1][0]
```