

```
1 import numpy as np
```

```
1 !pip install ortools
```

```
Requirement already satisfied: ortools in /usr/local/lib/python3.6/dist-packages (7.5.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (1.12.0)
Requirement already satisfied: protobuf>=3.11.2 in /usr/local/lib/python3.6/dist-packages (3.12.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (45.2.0)
```

▼ Lendo arquivo

```
1 path = 'Problema.txt'
2 arquivo = open(path, 'r')
3 numeros = []
4
5 for linha in arquivo:
6     linha = linha.strip()
7     numeros.append(linha)
8
9 arquivo.close()
10 numeros
```

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

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

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

```
1 a = [0]*rest #inicializando array com o numero de restricoes
2 b = [0]*rest
3
4 for i in range(2, len(numeros)): #começando em dois pois oq vem depois da linha 2 sao a
5     aa = numeros[i].split(' ')
6     b[i-2] = aa[len(aa)-1]
7     del(aa[len(aa)-1])
8     a[i-2] = aa
9 print(a, b, c)
```

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

```
1 a = np.double( a )
2 b = np.double( b )
3 c = np.double( c )
4
5 print(a, b, c)
```

```
[[3. 5. 2.]
```

▼ Adicionando no modelo

```
1 def create_data_model(A, B, C, num_vars, num_rest):
2     data = {}
3     data['constraint_coeffs'] = A
4     data['bounds'] = B
5     data['obj_coeffs'] = C
6     data['num_vars'] = num_vars
7     data['num_constraints'] = num_rest
8     return data

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

1 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 restricoes =', 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     objective.SetMinimization() #Problema de minimização
41     status = solver.Solve()
42
43     solution_value = []
44     if status == pywraplp.Solver.OPTIMAL:
45         print('\nValor ótimo = ', solver.Objective().Value())
46         for j in range(Data['num_vars']):
47             print(x[j].name(), ' = ', x[j].solution_value())
48             solution_value.append(x[j].solution_value())
49         print()
50         print('Problema resolvido em %f milliseconds' % solver.wall_time())
51         print('Problema resolvido em %d nós' % solver.nodes())
52     else:
53         print('Nao tem solucao otima.')
54
55     return (solution_value, solver.Objective().Value())

```

```

1 print("=== Solução inicial ===")
2
3 solution = main(data, "MoreOrEqual")

```

```

↳ === Solução inicial ===

```

```

==== Solução ====

```

```

Numero de variaveis = 3
Numero de restrições = 2

```

```

Valor ótimo = 12.0
x[0] = 1.0
x[1] = 0.5000000000000001
x[2] = 0.24999999999999999

```

```

Problema resolvido em 2.000000 milliseconds
Problema resolvido em 0 nós

```

▼ Adicionar uma restrição no modelo

```

1 def append_data_model (Data, x, b): #Adiciona ao modelo uma restrição
2     if x == -1:
3         return Data
4
5     A = Data['constraint_coeffs']
6     B = Data['bounds']
7     C = Data['obj_coeffs']
8     num_vars = Data['num_vars']
9     num_rest = Data['num_constraints'] + 1
10
11     data = {} #Criando outro data para nao alterar oq vir do parametro
12     data['constraint_coeffs'] = A

```

```

13 data['bounds'] = B
14 data['obj_coeffs'] = C
15 data['num_vars'] = num_vars
16 data['num_constraints'] = num_rest
17
18 #Dar um append na restrição indicando qual coeficiente do x é, se for x1, o array é
19 x0 = np.array([[1, 0, 0]])
20 x1 = np.array([[0, 1, 0]])
21 x2 = np.array([[0, 0, 1]])
22
23 if x == 0: #x indica qual x's é na restrição
24     data['constraint_coeffs'] = np.append(A, x0, axis=0)
25 if x == 1:
26     data['constraint_coeffs'] = np.append(A, x1, axis=0)
27 if x == 2:
28     data['constraint_coeffs'] = np.append(A, x2, axis=0)
29
30 #Dar append no lado direito da igualdade
31 data['bounds'] = np.append(B, b)
32
33 return data

```

▼ Exemplo de quemo fica o modelo adicionando as restrições

```

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

```

```

1 print(data1)
2 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.,

```

```

1 main(data1, "LessOrEqual")
2 main(data2, "MoreOrEqual")

```

```

{

```

```
==== Solução ====
```

```
Numero de variaveis = 3
Numero de restricoes = 3
Nao tem solucao otima.
```

```
==== Solução ====
```

```
Numero de variaveis = 3
Numero de restricoes = 3
```

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

```
Problema resolvido em 2.000000 milliseconds
Problema resolvido em 0 nós
([0.75, 1.0, 0.0], 13.75)
```

▼ Árvore

```
1 class Tree(object):
2     def __init__(self, solution, modelo, left=None, right=None):
3         self.solution = solution
4         self.modelo = modelo
5         self.left = left
6         self.right = right
```

▼ Verificar se a variável é um número inteiro

```
1 def isInt (string):
2     t = True
3     for i in range(len(string)):
4         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 índice da variável (para poder adicionar depois na restrição)

```

1 def Verifica_integralidade(Solution):
2     epsilon = 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 < epsilon: #Se esse valor for menor do que oq já tem no epsilon,
12             epsilon = 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])
```

```
1 pilha.push([[1.0, 0.5000000000000001, 0.2499999999999999], 12.0]) #, caso a solucao nao
2 print(pilha.see())
```

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

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

```
↳
```

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

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

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

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

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

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

```
1 pilha = Pilha()
2 print(pilha.see())
```

```
↳ ([[], -9999], [[], 9999])
```

▼ Adicionando lado direito e esquerdo da árvore

```
1 def AddArv(arv, data_left, data_right):
2     solution_right = main(data_right, 'MoreOrEqual' )
3     solution_left  = main(data_left, 'LessOrEqual' )
4
5     arv.left  = Tree( solution_left, data_left  )
6     arv.right = Tree( solution_right, data_right )
7
8     pilha.push(solution_right)
9     pilha.push(solution_left )
10
11     print("\nNó pai\n ", arv.solution)
12     print("\nLADO ESQUERDO:\n", arv.left.solution)
13     print("LADO DIREITO: \n", arv.right.solution)
14
15     print("Dual: %f Primal: %f\n" % (pilha.see()[0][1], pilha.see()[1][1]))
```

▼ Percorrendo a árvore

```
1 def Profun (tree):
2     primal = pilha.see()[1][1]
3     dual = pilha.see()[0][1]
4
5     index = Verifica_integralidade(tree.solution[0])
6
7     if ( index != - 1 and tree.solution[1] != 0 and tree.solution[1] <= primal and tree.s
8
```



```
9     b_limite_para_cima = ("%0.f") % tree.solution[0][index] #lado arredondado. se for
10     b_limite_para_baixo = ("%d" ) % tree.solution[0][index] #lado inteiro. se for 0.5
11
12     data_left = append_data_model(tree.modelo, int(index), int(b_limite_para_baixo ))
13     data_right = append_data_model(tree.modelo, int(index), int(b_limite_para_cima ))
14
15     AddArv(tree, data_left, data_right) #Adiciona o lado esquerdo e direito na arvore
16
17     index_left = Verifica_integralidade(tree.left.solution[0])
18     index_right = Verifica_integralidade(tree.right.solution[0])
19
20     if index_left != -1 and tree.left.solution[1] <= primal and tree.left.solution[1] <
21         Profun (tree.left)
22
23     if index_right != -1 and tree.right.solution[1] <= primal and tree.right.solution[1] <
24         Profun (tree.right)
25
26     else :
27         return 0
```

```
1 tree = Tree(solution, data)
2 pilha.push(tree.solution)
3 print(pilha.see())
4
5 print("=====  
Iniciando a árvore =====\n")
6 Profun(tree)
7 Profun(tree)
```



```
(([1.0, 0.5000000000000001, 0.2499999999999999], 12.0), ([], 9999))
```

```
===== Iniciando a árvore =====
```

```
==== Solução ====
```

```
Numero de variaveis = 3  
Numero de restrições = 3
```

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

```
Problema resolvido em 6.000000 milliseconds  
Problema resolvido em 0 nós
```

```
==== Solução ====
```

```
Numero de variaveis = 3  
Numero de restrições = 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: 9999.000000
```

```
==== Solução ====
```

```
Numero de variaveis = 3  
Numero de restrições = 3
```

```
Valor ótimo = 13.75  
x[0] = 0.75  
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 restrições = 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: 9999.000000
```

```
==== Solução ====
```

```
Numero de variaveis = 3
```

```
Numero de restrições = 4
```

```
Valor ótimo = 15.0
```

```
x[0] = 1.0
```

```
x[1] = 1.0
```

```
x[2] = 0.0
```

```
Problema resolvido em 2.000000 milliseconds
```

```
Problema resolvido em 0 nós
```

```
==== Solução ====
```

```
Numero de variaveis = 3
```

```
Numero de restrições = 4
```

```
Valor ótimo = 16.0
```

```
x[0] = 0.0
```

```
x[1] = 1.0
```

```
x[2] = 0.75
```

```
Problema resolvido em 2.000000 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
```

```
1 solucao = pilha.see()
2 print("Solução que está na pilha = ")
3 print(solucao)
4 print()
```

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

```
1 print("Solução ótima: ")
```

```
2 x = solucao[1][0]
```