Notação

Utilizaremos algumas letras minúsculas para facilitar a linguagem:

1. c $\in \{1, \ldots, C\}$
2. p $\in \{1, \ldots, P\}$
3. t $\in \{1, \ldots, T\}$
4. s $\in \{1, \ldots, S\}$

Implementação

Começamos por importar a biblioteca de programação linear do OR-Tools e criar uma instância do *solver*.

In [1]:
```python
from ortools.linear_solver import pywraplp

horario = pywraplp.Solver.CreateSolver('SCIP')
```

Inputs

1. Os parâmetros S, T, P, C
2. O conjunto de colaboradores de cada projeto, o seu líder e o número mínimo de reuniões semanais.
3. A disponibilidade de cada participante, incluindo o lider. Essa disponibilidade é um conjunto de “slots” representada numa matriz booleana de acessibilidade com uma linha por cada participante 1..C e uma coluna por “slot” 1..T

In [2]:
```python
# 1.

C, P, T, S = 8, 3, 6, 3

# 2.

# a chave do dicionário corresponde ao projeto p
# o triplo (x, y, z), sendo x um conjunto de colaboradores c do projeto p, y o colaborador c que é lider do projeto p e
# z, o número mínimo de horas semanais de do projeto p

projetos = {1: ({1, 2, 3}, 1, 3),
            2: ({4, 5, 6, 7, 3}, 4, 1),
            3: ({6, 7, 8}, 7, 1),
           }

# 3.

disponibilidade = {}

for c in range(1, C+1):
    disponibilidade[c] = {}
    for t in range(1, T+1):
        disponibilidade[c][t] = True


# alterar para true conforme necessário

# Colaborador 1
disponibilidade[1][1] = True
disponibilidade[1][2] = True
disponibilidade[1][3] = True
disponibilidade[1][5] = True
disponibilidade[1][6] = True

# Colaborador 2
disponibilidade[2][1] = True
disponibilidade[2][2] = True
disponibilidade[2][3] = True
disponibilidade[2][4] = True
disponibilidade[2][5] = True

# Colaborador 3
disponibilidade[3][2] = True
disponibilidade[3][3] = True
disponibilidade[3][5] = True
disponibilidade[3][6] = True

# Colaborador 4
disponibilidade[4][2] = True
disponibilidade[4][3] = True
disponibilidade[4][4] = True
disponibilidade[4][5] = True

# Colaborador 5
disponibilidade[5][1] = True
disponibilidade[5][2] = True
disponibilidade[5][3] = True
disponibilidade[5][5] = True

# Colaborador 6
disponibilidade[6][1] = True
disponibilidade[6][2] = True
disponibilidade[6][3] = True
disponibilidade[6][4] = True
disponibilidade[6][5] = True
disponibilidade[6][6] = True

# Colaborador 7
disponibilidade[7][1] = True
disponibilidade[7][2] = True
disponibilidade[7][3] = True
disponibilidade[7][5] = True
disponibilidade[7][6] = True

# Colaborador 8
disponibilidade[8][2] = True
disponibilidade[8][3] = True
disponibilidade[8][5] = True
disponibilidade[8][6] = True
```

Pretende-se definir o slot e sala em que cada colaborador participa da reunião do seu projeto, de modo a obedecer às restrições estabelecidas.Vamos usar uma família $X_{c,p,t,s}$ de variáveis binárias (i.e., que assumem valores inteiros $\{0,1\}$), com a seguinte semântica

$$X_{c,p,t,s} = 1 \textbf{ se e só se } \text{o colaborador c participa da reunião do projeto p, na sala s, no slot t}$$

In [3]:
```python
X = {}

for p in range(1, P+1):
    X[p] = {}
    for c in range(1, C+1):
        X[p,c] = {}
        for t in range(1, T+1):
            X[p,c,t] = {}
            for s in range(1, S+1):
                X[p,c,t,s] = horario.BoolVar(f'X[{p},{c},{t},{s}]')
```

Restrição 1

Os colaboradores não podem participar de uma reunião se não tiverem aquele slot disponível

$$\forall_{p<P}.\ \forall_{c<C}.\ \forall_{t<T}.\ \forall_{s<S}.\ ((\text{o colaborador c não tem o slot t disponível}) \implies X_{p,c,t,s} = 0)$$

In [4]:
```python
for c in range(1, C+1):
    for t in range(1, T+1):
        if not disponibilidade[c][t]:
            for s in range(1, S+1):
                for p in range(1, P+1):
                    horario.Add(X[p,c,t,s] == 0)
```

Restrição 2

Apenas 1 projeto pode ocupar uma sala s em um determinado slot t

Seja ld o líder do projeto p

$$\forall_{t<T}.\ \forall_{s<S} \sum_{p<P} X_{p,ld,t,s} \leq 1$$

In [5]:
```python
for t in range(1, T+1):
    for s in range(1, S+1):
        soma = 0
        for p in range(1, P+1):
            ld = projetos[p][1]
            soma += X[p,ld,t,s]
        horario.Add(soma <= 1)
```

Restrição 3

Seja ld o líder do projeto p e

$RS_p \equiv$ número mínimo de reuniões semanais do projeto p, temos que:

$$\forall_{p<P} \sum_{s<S,\, t<T} X_{p,ld,t,s} \geq RS_p$$

Isto é, o número de reuniões semanais do projeto p é maior ou igual a RS_p

In [6]:
```python
for p in range(1, P+1):
    RS = projetos[p][2]
    ld = projetos[p][1]
    soma = 0
    for t in range(1, T+1):
        for s in range(1, S+1):
            soma += X[p,ld,t,s]
    horario.Add(soma >= RS)
```

Restrição 4

Seja ld o colaborador c que é líder do projeto p, temos que:

$$\forall_{p<P}.\ \forall_{c<C}.\ \forall_{t<T}.\ \forall_{s<S}(X_{p,c,t,s}\ \leq X_{p,ld,t,s})$$

In [7]:
```python
for p in range(1, P+1):
    ld = projetos[p][1]
    for c in range(1, C+1):
        for t in range(1, T+1):
            for s in range(1, S+1):
                horario.Add(X[p,c,t,s] <= X[p,ld,t,s])
```

Restrição 5

Um colaborador c não pode estar em mais de uma reunião ao mesmo tempo, ou seja, ele não pode estar em mais de uma sala no mesmo slot t

$$\forall_{c<C}.\ \forall_{p<P}.\ \forall_{t<T}.\ (\sum_{s<S} X_{p,c,t,s} \leq 1)$$

In [8]:
```python
for p in range(1, P+1):
    for c in range(1, C+1):
        for t in range(1, T+1):
            soma = 0
            for s in range(1, S+1):
                soma += X[p,c,t,s]
            horario.Add(soma <= 1)
```

Restrição 5

As reuniões precisam de no mínimo uma participação de 50% do total de colaboradores de um projeto p para acontecer

Seja nc, o número total de colaboradores de um projeto p e, ld o líder do projeto p, temos que:

$$\forall_{p<P}.\ \forall_{t<T}.\ \forall_{s<S}.\ (\sum_{c<C \text{ e c pertence ao projeto p}} X_{p,c,t,s} \geq 0.5 \times nc \times X_{p,ld,t,s}))$$

In [9]:
```python
for p in range(1, P+1):
    nc = len(projetos[p][0])
    ld = projetos[p][1]
    mul = round(nc * 0.5)
    for t in range(1, T+1):
        for s in range(1, S+1):
            soma = 0
            for c in projetos[p][0]:
                soma += X[p,c,t,s]

            horario.Add(soma >= mul * X[p,ld,t,s])
```

Função objetivo 1

Maximizar o número de reuniões efetivamente realizadas

Seja ld o líder do projeto p

$$\sum_{p<P,\, s<S,\, t<T} X_{p,ld,t,s}$$

In [10]:
```python
f_max = 0

for p in range(1, P+1):
    ld = projetos[p][1]
    for t in range(1, T+1):
        for s in range(1, S+1):
            f_max += X[p,ld,t,s]

horario.Maximize(f_max)
```

Função objetivo 2

Minimizar o número médio de reuniões por participante

$$\sum_{p<P,\, c<C,\, s<S,\, t<T} X_{p,c,t,s}$$

In [11]:
```python
f_min = 0

for p in range(1, P+1):
    for c in range(1, C+1):
        for t in range(1, T+1):
            for s in range(1, S+1):
                f_min += X[p,c,t,s]

horario.Minimize(f_min)
```

Função objetivo 3

Maximizar uma "média" entre as duas funções objetivos anterior

In [12]:
```python
PESO = 0.05 # varia entre 0 e 1
f_media = (1-PESO)*f_max - PESO * f_min

horario.Maximize(f_media)
```

Testes

In [13]:
```python
status = horario.Solve()
if status == pywraplp.Solver.OPTIMAL:
    # print("yes")
    for p in range(1, P+1):
        for c in range(1, C+1):
            for t in range(1, T+1):
                for s in range(1, S+1):
                    x = X[p,c,t,s].solution_value()
                    # print(f'({p},{c},{t},{s}): {x}')

else:
    # print("no")
    pass
```

In [14]:
```python
# print(horario.Objective().Value())
```