TP1 - Exercício 1

Grupo 1

- Diogo Coelho da Silva A100092
- Pedro Miguel Ramôa Oliveira A97686

Problema proposto:

1. Pretende-se construir um horário semanal para o plano de reuniões de projeto de uma "StartUp" de acordo com as seguintes condições:
   - **a)** Cada reunião ocupa uma sala (enumeradas 1...S) durante um "slot" (tempo,dia). Assume-se os dias enumerados 1..D e, em cada dia, os tempos enumerados 1..T.
   - **b)** Cada reunião tem associado um projeto (enumerados 1..P) e um conjunto de participantes. Os diferentes colaboradores são enumerados 1..C.
   - **c)** Cada projeto tem associado um conjunto de colaboradores, dos quais um é o líder. Cada projeto realiza um dado número de reuniões semanais. São "inputs" do problema o conjunto de colaboradores de cada projeto, o seu líder e o número de reuniões semanais.
   - **d)** O líder do projeto participa em todas as reuniões do seu projeto; os restantes colaboradores podem ou não participar consoante a sua disponibilidade, num mínimo ("quorum") de 50% do total de colaboradores do projeto. A disponibilidade de cada participante, incluindo o lider, é um conjunto de "slots" ("inputs" do problema).

Proposta de resolução:

O problema que foi apresentado tem como objetivo a criação de um horário semanal otimizado para a marcação de reuniões para uma *startup*. O objetivo é alocar eficientemente recursos como salas e tempo, considerando a disponibilidade dos colaboradores e as restrições dadas no enunciado.

Na solução proposta iremos modelar o problema de acordo com as nossas necessidades e utilizaremos um solver SCIP para encontrar uma solução otimizada.

Foram consideradas restrições dadas pelo enunciado, como por exemplo, a obrigatoriedade da presença do lider em todas as reuniões do seu projeto, a disponibilidade dos colaboradores, a partição exclusiva dos membros do projeto, a alocação de apenas um projeto por sala em cada slot e a garantia de um quorum minimo em cada reunião.

1. Importar as bibliotecas importantes

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

```
import pandas as pd
```

- `pywraplp` : Importa o solver de programação linear da biblioteca OR-Tools para resolver o problema de otimização.
- `random` : Gera aleatoriedade para simular a disponibilidade de colaboradores e equipas dos projetos.
- `pandas` : Biblioteca usada para organizar os dados em formato de tabela (DataFrame), facilitando a manipulação e visualização.

2. Pedir variáveis de entrada ao utilizador

In [2]:
```
#Função que recolhe as variáveis do utilizador... tem valores padrão pré-definid
def get_input(prompt, default):
    user_input = input(f"{prompt} [{default}]: ")  # Mostra o valor padrão
    return int(user_input) if user_input else default  # Usa o valor padrão se o

S = get_input("Digite o número de salas", 5)
D = get_input("Digite o número de dias", 5)
H = get_input("Digite o número de slots por dia", 8)
P = get_input("Digite o número de projetos", 5)
C = get_input("Digite o número de colaboradores", 30)

#Pretty print das variáveis introduzidas pelo utilizador
print(f"Parâmetros recebidos: {S} salas, {D} dias, {H} slots por dia, {P} projet
```

```
Parâmetros recebidos: 5 salas, 5 dias, 8 slots por dia, 5 projetos, 30 colaborado
res.
```

Nesta parte do código são definidas as variáveis do problema. A função definida "get_input" pergunta ao utilizador os valores que pretende introduzir para o problema. Se o utilizador não colocar valores, existem valores pre-definidos pelo programa. Existe também controlo de erros, como por exemplo, a eventualidade de o introduzir valores não validos, como por exemplo uma string.

Variáveis recolhidas

- `S` -> número de salas (valor default = 5)
- `D` -> número de dias (valor default = 5)
- `H` -> número de horas (valor default = 8)
- `P` -> número de projetos (valor default = 5)
- `C` -> número de colaboradores (valor default = 30)

3. Definir de forma aleatória a alocação das slots de horário e dos colaboradores

In [3]:
```
#Cria uma lista de tuplos para definir as slots disponiveis de horário com base
Slots = [(d, h) for d in range(D) for h in range(H)]
#Define os colaboradores em set para nao existirem repetidos
Colabs = set(range(1, C + 1))

random.seed(42)
Colaboradores = [random.sample(Slots, 20) for _ in range(1, C + 1)]
```

```
'''
Primeiro é inicializado um array vazio para guardar os projetos. De seguida, enq
atribuindo colaboradores de forma random aos projetos
'''
Projectos = []
for idx in range(P):
    available_team_size = min(3, len(Colabs))  #Não pode ter menos do que 3 cola
    team = random.sample(list(Colabs), available_team_size)
    Projectos.append((random.randint(1, 5), team))
    Colabs = Colabs - set(team)
```

Nesta parte do código é criada uma estrutura para alocar slots de horários, colaboradores e os respetivos projetos de forma aleatória. Para isso usamos a biblioteca do python "random" e são definidas seeds para ser possível gerar números aleatórios.

Os slots de horario vão ser criados em tuplos para respeitar as normas do enunciado, que diz que uma slot de horario é definida por uma hora e um dia.

Os colaboradores são definidos como um set para garantir que não há repetidos. A lista dos colaboradores vai conter uma lista de 20 slots de horarios escolhidos aleatóriamente da lista de horários disponiveis.

No fim é definida uma lista de projetos vazia. Para cada projeto são selecionados aleatoriamente um conjunto de colaboradores do set anteriormente criado. São definados aleatoriamente o número de reuniões semanais de cada projeto. No final do código, são removidos os colaboradores alocados para o projeto do set de colaboradores para evitar repetidos.

4. Inicializar o solver

In [4]:
```
# Inicializando o solver SCIP
horario = pywraplp.Solver.CreateSolver('SCIP')
# Variáveis de decisão: x[s, d, h, p, c] = 1 se o colaborador c participa na reu
x = {}
for s in range(S):
    for d in range(D):
        for h in range(H):
            for p in range(P):
                for c in range(1, C + 1): #C+1 Para nao contar com o colaborador
                    x[s, d, h, p, c] = horario.BoolVar(f'x[{s},{d},{h},{p},{c}]'
```

Nesta parte do código, utilizamos a biblioteca "ortools" para criar um solver do tipo SCIP, "Solving Constraint Integer Programs", geralmente usado para resolver problemas de otimização e de programação com restrições que vão ser apresentadas posteriormente.

Variáveis de decisão

`x[s, d, h, p, c]` é uma variável de decisão binária (0 ou 1), onde:

- `s` : sala
- `d` : dia
- `h` : hora
- `p` : projeto

- `c` : colaborador

A variável é igual a 1 se o colaborador `c` participa na reunião do projeto `p` no slot de horário representado pela combinação da sala `s` , dia `d` e hora `h` . Caso contrário, a variável será 0.

Vai ser utilizado um dicionário `x` para armazenar todas as variáveis de decisão resultantes do loop utilizado para gerar todas as combinações possíveis das variáveis apresentadas.

5. Restrições definidas para o solver

Restrição 1: O líder de cada projeto deve participar em todas as reuniões do seu projeto.

Nesta parte código, para cada projeto `p` , o líder é definido através da manipulação de listas por `Projectos[p][1][0]` , que vais buscar o primeiro colaborador da lista de colaboradores desse projeto. A restrição adicionada ao solver garante que o somatório das participações do líder da equipa em todas as combinações de sala (`s`), dia (`d`) e hora (`h`) para o projeto `p` é exatamente igual ao número de reuniões (R) que o projeto possui, representado por `Projectos[p][0]` .

A expressão no código representa a seguinte fórmula matemática:

$$\sum_{s=1}^{S}\sum_{d=1}^{D}\sum_{h=1}^{H} x[s,d,h,p,\text{líder}] = \text{ReuniõesProjeto}$$

Onde:

- `S` é o número de salas,
- `D` é o número de dias,
- `H` é o número de horas,
- `x[s, d, h, p, líder]` é uma variável binária que indica se o líder do projeto `p` está presente na sala `s` , no dia `d` e na hora `h` ,
- `ReuniõesProjeto` é o número total de reuniões que o projeto `p` deve ter.

In [5]:
```
for p in range(P):
    lider = Projectos[p][1][0]
    horario.Add(
        sum(x[s, d, h, p, lider] for s in range(S) for d in range(D) for h in ra
    )
```

Restrição 2: Colaboradores fora da disponibilidade não podem participar.

Este código garante que colaboradores que não estão disponíveis em um determinado dia (`d`) e hora (`h`) não possam participar das reuniões de qualquer projeto `p` em qualquer sala `s` . Para cada combinação de sala, dia, hora e colaborador, verifica-se se o par `(d, h)` está presente na lista de disponibilidade do colaborador `c` . Se não estiver, a restrição `x[s, d, h, p, c]` é forçada a ser igual a 0, ou seja, o colaborador `c` não pode participar.

A expressão no código representa a seguinte fórmula matemática:

$$\forall c \in \{1, 2, \ldots, C\}, \forall s \in \{1, 2, \ldots, S\}, \forall d \in \{1, 2, \ldots, D\}, \forall h \in \{1, 2, \ldots, H\}, \quad \text{se} \quad (c$$

In [6]:
```
for s in range(S):
    for d in range(D):
        for h in range(H):
            for p in range(P):
                for c in range(1, C + 1):
                    if (d, h) not in Colaboradores[c-1]:
                        horario.Add(x[s, d, h, p, c] == 0)
```

Restrição 3: Apenas colaboradores do projeto podem participar.

Este código assegura que apenas os colaboradores designados para um projeto específico `p` podem participar das reuniões desse projeto. Para cada combinação de sala (`s`), dia (`d`), hora (`h`) e colaborador (`c`), verifica-se se o colaborador `c` está presente na lista de colaboradores do projeto `p` (`Projectos[p][1]`). Se o colaborador não estiver na lista, a restrição `x[s, d, h, p, c]` é forçada a ser igual a 0, significando que ele não pode participar da reunião.

A lógica da restrição pode ser expressa como:

$$\forall c \in \{1, 2, \ldots, C\}, \forall s \in \{1, 2, \ldots, S\}, \forall d \in \{1, 2, \ldots, D\}, \forall h \in \{1, 2, \ldots, H\}, \quad \text{se} \quad c$$

In [7]:
```
for s in range(S):
    for d in range(D):
        for h in range(H):
            for p in range(P):
                for c in range(1, C + 1):
                    if c not in Projectos[p][1]:
                        horario.Add(x[s, d, h, p, c] == 0)
```

Restrição 4: Uma sala pode ter apenas um projeto por slot de tempo.

Este código garante que, em um determinado slot de tempo (definido por uma combinação de sala `s` , dia `d` e hora `h`), apenas um projeto pode ser realizado. Para cada sala, dia e hora, a restrição é adicionada ao solver para que a soma das variáveis `x[s, d, h, p, Projectos[p][1][0]]` para todos os projetos `p` não exceda 1. Isso significa que não pode haver mais de um projeto ocorrendo na mesma sala no mesmo horário.

A lógica da restrição pode ser expressa como:

$$\forall s \in \{1, 2, \ldots, S\}, \forall d \in \{1, 2, \ldots, D\}, \forall h \in \{1, 2, \ldots, H\}, \quad \sum_{p=1}^{P} x[s, d, h, p, \text{líder}] \leq 1$$

Onde:

- (x[s, d, h, p, líder]) é a variável de decisão indicando a presença do líder do projeto (p) na sala (s) no dia (d) e na hora (h).

- A soma representa a soma das variáveis de todos os projetos no slot de tempo especificado.

In [8]:
```
for s in range(S):
    for d in range(D):
        for h in range(H):
            horario.Add(sum(x[s, d, h, p, Projectos[p][1][0]] for p in range(P))
```

Restrição 5: Um colaborador só pode estar em uma sala em um dado slot.

Este código assegura que um colaborador específico não pode participar de mais de uma reunião ao mesmo tempo em diferentes salas. Para cada combinação de dia `d`, hora `h` e projeto `p`, a restrição adicionada ao solver força a soma das variáveis `x[s, d, h, p, c]` para todas as salas `s` a ser no máximo 1. Isso significa que, para um determinado colaborador `c` em um dia e hora específicos, ele pode estar presente em apenas uma sala, garantindo que não haja sobreposição de reuniões.

A lógica da restrição pode ser expressa como:

$$\forall c \in \{1, 2, \ldots, C\}, \forall d \in \{1, 2, \ldots, D\}, \forall h \in \{1, 2, \ldots, H\}, \quad \sum_{s=1}^{S} x[s, d, h, p, c] \leq 1$$

Onde:

- (x[s, d, h, p, c]) é a variável de decisão que indica a presença do colaborador (c) na sala (s) no dia (d) e na hora (h).
- A soma representa a soma das variáveis para todas as salas em um dado dia e hora.

In [9]:
```
for d in range(D):
    for h in range(H):
        for p in range(P):
            for c in Projectos[p][1]:
                horario.Add(sum(x[s, d, h, p, c] for s in range(S)) <= 1)
```

Restrição 6: Quorum de 50% dos colaboradores, incluindo o líder.

Esta restrição garante que, para que uma reunião de um projeto `p` ocorra em um determinado slot de tempo (definido pela combinação de sala `s`, dia `d` e hora `h`), deve haver pelo menos 50% dos colaboradores do projeto presentes, incluindo o líder. O quorum é calculado como a metade do número total de colaboradores do projeto (`len(Projectos[p][1]) // 2 + 1`), arredondando para cima. A restrição imposta ao solver assegura que a soma das variáveis `x[s, d, h, p, c]` para todos os colaboradores `c` do projeto `p` seja maior ou igual ao quorum multiplicado pela presença do líder.

A lógica da restrição pode ser expressa como:

$$\forall s \in \{1, 2, \ldots, S\}, \forall d \in \{1, 2, \ldots, D\}, \forall h \in \{1, 2, \ldots, H\}, \forall p \in \{1, 2, \ldots, P\},$$
$$c \in \text{Colal}$$

Onde:

- (x[s, d, h, p, c]) é a variável de decisão que indica a presença do colaborador (c) na sala (s) no dia (d) e na hora (h).
- TotalColaboradores(p)é o número total de colaboradores do projeto (p).
- O lado direito da equação representa a necessidade de ter pelo menos 50% dos colaboradores, incluindo o líder, presentes.

In [10]:
```
for s in range(S):
    for d in range(D):
        for h in range(H):
            for p in range(P):
                quorum = len(Projectos[p][1]) // 2 + 1  # 50% ou mais com o líde
                horario.Add(sum(x[s, d, h, p, c] for c in Projectos[p][1]) >= qu
```

Restrição 7:Esta restrição força o uso de mais do que uma sala.

In [11]:
```
for s in range(S):
  horario.Add(sum(x[s, d, h, p, c] for d in range(D) for h in range(H) for p in
```

Função Objetivo

A função objetivo do modelo procura maximizar a participação total dos colaboradores em reuniões de projetos. Isso é feito somando todas as variáveis de decisão (x[s, d, h, p, c]), que indicam se um colaborador (c) está presente na sala (s) no dia (d) e na hora (h) para o projeto (p).

In [12]:
```
horario.Maximize(
    sum(x[s, d, h, p, c] for s in range(S) for d in range(D) for h in range(H) f
)
```

6. Resolver o problema e apresentar o resultado

O código abaixo apresentado é responsável por resolver o problema apresentado e gerar uma tabela com as reuniões alocadas se for possível gerar uma solução ótima.

1. O código chama o método `.Solve()` da biblioteca `ortools` para tentar encontrar uma solução ótima para o problema. Se o status desta operação for `OPTIMAL`, o código vai iterar sobre todas as variaveis e vai verificar quais foram as reuniões alocadas. Para cada reunião alocada recolhe os colaboradores.
2. Após isto os dados coletados são postos numa lista chamada `schedule_data`, onde cada entrada contém:
   - O número do projeto
   - O número da sala
   - O dia e a hora combinados num tuplo
   - A lista de participantes de cada projeto
3. Se não for encontrada uma solução ótima, a tabela é preenchida com uma mensagem indicando que nenhuma solução foi encontrada.
4. Um DataFrame do pandas é criado a partir da lista `schedule_data`, com colunas para projeto, sala, dia/hora e participantes.

In [13]:
```
# Resolver o problema
status = horario.Solve()

# Verificar a solução e preparar a tabela de resultados
tabelaHorario = []


if status == pywraplp.Solver.OPTIMAL:
    for p in range(P):
        for s in range(S):
            for d in range(D):
                for h in range(H):
                    if round(x[s, d, h, p, Projectos[p][1][0]].solution_value())
                        participants = [c for c in Projectos[p][1] if round(x[s,
                        tabelaHorario.append([p + 1, s + 1, f"{d + 1},{h + 1}",
else:
    tabelaHorario.append([None, None, None, "Nenhuma solução ótima encontrada."]

df_schedule = pd.DataFrame(tabelaHorario, columns=["Projeto", "Sala", "Dia_Hora"


def style_schedule(df):
    return df.style.set_table_styles(
        [{'selector': 'tr:nth-child(even)',
          'props': [('background-color', '#f2f2f2')]},   # Cor alternada para li
         {'selector': 'th, td',
          'props': [('border', '1px solid black')]}      # Bordas em volta das c
        ]
    ).set_properties(**{'text-align': 'center'})          # Alinhar o texto no c

# Aplicar estilo e exibir diretamente no notebook
styled_table = style_schedule(df_schedule)

# Exibir a tabela estilizada no Jupyter Notebook sem o índice
display(styled_table.hide(axis='index'))
```

Projeto	Sala	Dia_Hora	Participantes
1	1	1,2	[24, 16]
1	5	2,5	[24, 16]
1	5	3,1	[24, 16]
1	5	3,7	[24, 16]
2	4	1,4	[9, 8]
2	5	5,2	[9, 27]
2	5	5,5	[9, 27]
3	5	5,4	[20, 19, 25]
3	5	5,7	[20, 19, 25]
4	5	1,2	[11, 3]
4	5	4,6	[11, 18, 3]
4	5	5,6	[11, 18, 3]
5	5	4,7	[12, 15]
5	5	4,8	[12, 13]
5	5	5,3	[12, 15]