MAPA DE HORÁRIOS

```
import pandas as pd
from collections import defaultdict


# ============================================================================
# 1) LEITURA DOS ARQUIVOS
#    Ajuste se seus arquivos têm nomes/pastas diferentes.
# ============================================================================

df_prof = pd.read_excel("/content/professores.xlsx")
df_demanda_raw = pd.read_excel("/content/demanda.xlsx")


# ============================================================================
# 2) MONTAR DICIONÁRIO DE DISPONIBILIDADE
#    disponibilidade[professor][turno][dia]["1".."5"] = True/False
#    Onde "dia" será "SEG", "TER", "QUA", "QUI", "SEX"
#    e cada tempo = 1..5  (ajuste se houver 6 tempos)
# ============================================================================

# PARAMETROS
dias_semana = ["SEG", "TER", "QUA", "QUI", "SEX"]
tempos = [1,2,3,4,5]  # 5 tempos/dia
max_tentativas_realloc = 3  # quantas vezes tentaremos rearranjar para alocar tudo

disponib = {}
for i, row in df_prof.iterrows():
    servidor = row["SERVIDOR"]
    turno    = row["TURNO"]
    if servidor not in disponib:
        disponib[servidor] = {}
    if turno not in disponib[servidor]:
        disponib[servidor][turno] = {}

    for d in dias_semana:
        if d not in disponib[servidor][turno]:
            disponib[servidor][turno][d] = {}
        for t in tempos:
            col = f"{d}{t}"  # Ex.: SEG1, SEG2...
            if col in row:
                disponib[servidor][turno][d][t] = (row[col] == 1)
            else:
                disponib[servidor][turno][d][t] = False


# ============================================================================
# 3) FILTRAR "APOIO" E MONTAR LISTA DE DEMANDAS
# ============================================================================
# Ignorar "APOIO" (não será alocado)
df_demand = df_demanda_raw[df_demanda_raw["DISC"] != "APOIO"].copy()

# Converter tudo em uma lista de dicts
demanda = []
for i, row in df_demand.iterrows():
    demanda.append({
        "prof":   row["SERVIDOR"],
        "turno":  row["TURNO"],
        "nivel":  row["NIVEL"],
        "serie":  str(row["SERIE/ANO"]),
        "turma":  str(row["TURMA"]),
        "disc":   row["DISC"],
        "ch":     int(row["CH.TURMA"])
    })


# ============================================================================
# 4) ESTRUTURA DE ARMAZENAMENTO
#    horario[turno][nivel][dia][(serie,turma)] = {tempo: (prof,disc)}
#    Assim conseguimos controlar a alocação de cada turma, dia e tempo.
# ============================================================================
horario = {}

def init_estrutura(turno, nivel, dia, serie, turma):
    if turno not in horario:
        horario[turno] = {}
    if nivel not in horario[turno]:
        horario[turno][nivel] = {}
    if dia not in horario[turno][nivel]:
        horario[turno][nivel][dia] = {}
```

```
        if (serie, turma) not in horario[turno][nivel][dia]:
            horario[turno][nivel][dia][(serie, turma)] = {}

    # Inicializa as chaves do dicionário para todos combos que aparecem
    for d in demanda:
        for dia in dias_semana:
            init_estrutura(d["turno"], d["nivel"], dia, d["serie"], d["turma"])


    # ============================================================================
    # 5) FUNÇÕES DE VERIFICAÇÃO DAS REGRAS
    # ============================================================================

    def professor_disponivel(prof, turno, dia, tempo):
        """Professor está disponível nesse dia/tempo?"""
        if prof not in disponib:
            return False
        if turno not in disponib[prof]:
            return False
        return disponib[prof][turno][dia][tempo]

    def professor_ocupado_esse_tempo(turno, nivel, dia, tempo, professor):
        """Professor já alocado em alguma turma nesse dia/tempo?"""
        aloc_turmas = horario[turno][nivel][dia]
        for (serie_t, dicTempos) in aloc_turmas.items():
            if tempo in dicTempos:
                (p, di) = dicTempos[tempo]
                if p == professor:
                    return True
        return False

    def turma_ja_ocupada(turno, nivel, dia, tempo, serie, turma):
        """Essa turma já tem aula nesse dia/tempo?"""
        aloc = horario[turno][nivel][dia][(serie, turma)]
        return (tempo in aloc)

    def count_prof_disc_same_day(turno, nivel, dia, serie, turma, professor, disc):
        """Quantas vezes (prof+disc) já aparece nesse dia para essa turma?"""
        aloc = horario[turno][nivel][dia][(serie, turma)]
        cnt = 0
        for (t, (p, d)) in aloc.items():
            if p == professor and d == disc:
                cnt += 1
        return cnt

    def check_buraco(turno, nivel, dia, serie, turma):
        """
        Verifica se há "buraco": REGRAS:
          - Se a turma tem X tempos ocupados, eles devem ser 1..X, sem pular
            e obrigatoriamente começar no tempo 1.
        """
        aloc = horario[turno][nivel][dia][(serie, turma)]
        if not aloc:
            return False
        tempos_ocupados = sorted(aloc.keys())

        # Se o primeiro tempo ocupado não for 1, é buraco
        if tempos_ocupados[0] != 1:
            return True

        # Se há "gap" no meio
        ultimo = max(tempos_ocupados)
        for t in range(1, ultimo+1):
            if t not in aloc:
                return True
        return False

    def alocar_1tempo(item, dia, tempo):
        """Aloca 1 tempo no 'horario'."""
        turno = item["turno"]
        nivel = item["nivel"]
        serie = item["serie"]
        turma = item["turma"]
        prof  = item["prof"]
        disc  = item["disc"]
        horario[turno][nivel][dia][(serie, turma)][tempo] = (prof, disc)

    def desalocar_1tempo(item, dia, tempo):
        """Desaloca (se existir)."""
        turno = item["turno"]
        nivel = item["nivel"]
        serie = item["serie"]
        turma = item["turma"]
```

```
        aloc = horario[turno][nivel][dia][(serie, turma)]
        if tempo in aloc:
            del aloc[tempo]

    def pode_alocar(item, dia, tempo, forcar_2tempos=False):
        """
        Verifica se podemos alocar esse professor/disc no dia/tempo sem violar:
          1) professor disponível
          2) professor não em outra turma nesse tempo
          3) preferencialmente 1 tempo/dia/turma/disc, exceto se:
             - for outradisc do mesmo prof (regra 2)
             - ou 'forcar_2tempos=True' (tentativa de acomodar?).
          4) sem gerar buraco
        Retorna True/False.
        """
        prof = item["prof"]
        disc = item["disc"]
        turno= item["turno"]
        nivel= item["nivel"]
        serie= item["serie"]
        turma= item["turma"]

        # 1) professor disponível?
        if not professor_disponivel(prof, turno, dia, tempo):
            return False
        # 2) professor não em outra turma
        if professor_ocupado_esse_tempo(turno, nivel, dia, tempo, prof):
            return False
        # 3) quantos tempos (prof+disc) nesse dia?
        cpd = count_prof_disc_same_day(turno, nivel, dia, serie, turma, prof, disc)
        if cpd >= 1:
            # Se já tem 1 tempo do MESMO disc, e `forcar_2tempos=False`, retornamos False
            return forcar_2tempos

        # 4) se alocar, verifica buraco
        alocar_1tempo(item, dia, tempo)
        if check_buraco(turno, nivel, dia, serie, turma):
            # se gerou buraco, desfaz
            desalocar_1tempo(item, dia, tempo)
            return False

        return True


    # ============================================================================
    # 6) ALGORITMO PRINCIPAL DE ALOCAÇÃO
    #    - Agrupar as demandas por (turno,nivel,serie,turma)
    #    - Tentar distribuir ao longo da semana sem buracos, cada dia usando tempos contíguos do 1..X
    #    - Se faltar alocar algo, tentamos "forcar_2tempos=True" ou reembaralhar
    # ============================================================================

    # Agrupa as demandas em: dict[(turno,nivel,serie,turma)] = lista de (prof,disc,ch)
    demanda_por_turma = defaultdict(list)
    for d in demanda:
        key = (d["turno"], d["nivel"], d["serie"], d["turma"])
        demanda_por_turma[key].append((d["prof"], d["disc"], d["ch"]))

    def tenta_alocar_turma(turno, nivel, serie, turma, lista_demandas):
        """
        Aloca as aulas dessa TURMA ao longo da semana.
        lista_demandas = [(prof,disc,ch), ...]
        """
        nao_aloc = 0
        # Precisamos de "ch" total para cada (prof,disc)
        # Vamos varrer dia a dia, da SEG a SEX, e para cada dia
        # preenchemos do tempo 1..X.
        # A heurística: tentamos priorizar 1 tempo/dia. Se não couber,
        # permitimos 2 tempos no mesmo dia (forcar_2tempos=True).

        for (prof, disc, ch) in lista_demandas:
            ch_rest = ch
            # Tentar alocar ch_rest
            for d in dias_semana:
                for t in tempos:
                    if ch_rest <= 0:
                        break
                    item = {"prof":prof, "disc":disc,
                            "turno":turno, "nivel":nivel,
                            "serie":serie,"turma":turma}
                    # Tenta alocar preferencialmente 1 tempo/dia
                    if pode_alocar(item, d, t, forcar_2tempos=False):
                        ch_rest -= 1
                if ch_rest <= 0:
```

```
                    break

            # Se ainda sobrar, tentamos permitir "forcar_2tempos=True"
            if ch_rest > 0:
                for d in dias_semana:
                    for t in tempos:
                        if ch_rest <= 0:
                            break
                        item = {"prof":prof, "disc":disc,
                                "turno":turno, "nivel":nivel,
                                "serie":serie,"turma":turma}
                        if pode_alocar(item, d, t, forcar_2tempos=True):
                            ch_rest -= 1
                    if ch_rest <= 0:
                        break

            if ch_rest > 0:
                nao_aloc += ch_rest

        return nao_aloc

    def alocar_tudo():
        nao_aloc_total = 0
        for (turno, nivel, serie, turma), lista_dem in demanda_por_turma.items():
            nao_aloc_turma = tenta_alocar_turma(turno, nivel, serie, turma, lista_dem)
            nao_aloc_total += nao_aloc_turma
        return nao_aloc_total

    # Roda a alocação
    nao_alocados = alocar_tudo()
    if nao_alocados > 0:
        print(f"[Aviso] Não foi possível alocar {nao_alocados} tempos.")
    else:
        print("Todas as aulas foram alocadas segundo as regras (ou com pequenas exceções).")
```

```
Todas as aulas foram alocadas segundo as regras (ou com pequenas exceções).
```

```
    # ================================================================================
    # 7) GERA O DATAFRAME FINAL E DIVIDE EM ABAS POR (TURNO,NÍVEL)
    #    Formato: colunas = (SÉRIE-TURMA), linhas = (DIA, TEMPO),
    #    célula = "SERVIDOR+DISC"
    # ================================================================================

    # Precisamos descobrir as colunas para cada (tnr,niv)
    horario_final = horario  # renome simplificado
    colunas_por_tn = defaultdict(set)

    for tnr in horario_final:
        for niv in horario_final[tnr]:
            for d in dias_semana:
                turmas_dict = horario_final[tnr][niv][d]
                for (serie, tma), aloc in turmas_dict.items():
                    colunas_por_tn[(tnr,niv)].add(f"{serie}-{tma}")

    def gera_df_aba(tnr, niv):
        """Cria DataFrame com colunas = 'serie-turma', linhas = (DIA, TEMPO)."""
        turmas_col = sorted(list(colunas_por_tn[(tnr,niv)]))
        rows = []
        for d in dias_semana:
            for tm in tempos:
                row_dict = {
                    "TURNO": tnr,
                    "NIVEL": niv,
                    "DIA": d,
                    "TEMPO": tm
                }
                # inicializa
                for col_turma in turmas_col:
                    row_dict[col_turma] = ""

                # preenche
                turmas_dict = horario_final[tnr][niv][d]
                # turmas_dict => {(serie, turma): { tempo: (prof,disc) }}, mas nós guardamos invertido
                # No script construído, é 'turmas_dict[(serie, turma)] = { tempo: (prof,disc) }'.
                # Precisamos ver se no tempo `tm` existe algo
                for (serie, tma), dicTempos in turmas_dict.items():
                    if tm in dicTempos:
                        (p, dsc) = dicTempos[tm]
                        col_name = f"{serie}-{tma}"
                        row_dict[col_name] = f"{p}+{dsc}"
                rows.append(row_dict)
```

```
        df_ = pd.DataFrame(rows)
        return df_[ ["TURNO","NIVEL","DIA","TEMPO"] + turmas_col ]


    with pd.ExcelWriter("QUADRO_HORARIOS_FINAL.xlsx") as writer:
        for tnr in horario_final:
            for niv in horario_final[tnr]:
                df_aba = gera_df_aba(tnr, niv)
                nome_aba = f"{tnr}_{niv}"  # ex.: MAT_FUND
                df_aba.to_excel(writer, sheet_name=nome_aba, index=False)

    print("Geração final concluída: 'QUADRO_HORARIOS_FINAL.xlsx'. Cada aba corresponde a (TURNO, NÍVEL).")
```

```
Geração final concluída: 'QUADRO_HORARIOS_FINAL.xlsx'. Cada aba corresponde a (TURNO, NÍVEL).
```