Documentación BackEnd **Christian Francisco Rojas Espinoza**

El código utiliza el algoritmo de back tracking para asignar horarios a cursos, asegurándose de que los profesores y aulas estén disponibles en los mismos horarios. Si se encuentra una solución completa, la función devuelve True, de lo contrario, devuelve False.

Profesor: Representa a un profesor. Cada profesor tiene un identificador, un nombre, una lista de cursos que imparte y una lista de horarios disponibles.

Aula: Representa un aula. Cada aula tiene un identificador, una capacidad y una lista de horarios disponibles.

Curso: Representa un curso. Cada curso tiene un identificador, un nombre, un número de horas por semana y una lista de restricciones.

Planificador Horarios: Representa un planificador de horarios. Cada planificador tiene una lista de profesores, una lista de aulas y una lista de cursos.

Funciones de Planificador Horarios:

- **asignar_horarios():** Ejecuta el algoritmo de asignación de horarios.
- **_backtracking():** Implementa el algoritmo de backtracking.
- **_es_disponible():** Determina si un profesor y un aula son compatibles.
- **cualhorariro():** Obtiene el horario disponible de un profesor y un aula.

Como funciona el algoritmo backtraking

Funcion def _backtracking (self, asignaciones=[]) :

- esta función implementa el algoritmo de backtracking para asignar horarios a los cursos.
- Toma una lista asignaciones como argumento, que representa las asignaciones parciales hechas hasta el momento.
- La función devuelve True si se encuentra una solución completa y False de lo contrario.

Condición de terminación:

```
if len(asignaciones) == len(self.cursos):
    print("¡Solución encontrada!")
    self.horarios_asignados = asignaciones.copy()
    return True
```

- Si la longitud de asignaciones es igual a la longitud total de cursos (len(self.cursos)), significa que se han asignado horarios a todos los cursos y se ha encontrado una solución completa.

Iteración sobre cursos:

Se itera sobre los cursos para asignar horarios.

- self.cursos: Es una lista que parece contener todos los cursos que deben ser programados.
- asignaciones: Es la lista de asignaciones parciales, que contiene información sobre qué cursos ya han sido asignados y en qué horarios.

La longitud de asignaciones representa cuántos cursos ya han sido asignados. Al hacer self.cursos[len(asignaciones)], se accede al próximo curso que aún no ha sido asignado.

```
curso_actual = self.cursos[len(asignaciones)]
```

Iteración sobre profesores:

Se itera sobre los profesores para encontrar un profesor que pueda enseñar el curso actual**.**

Primero verifica si el profesor tiene materia que dar (porque profesor.cursos es igual a una lista de las materias que da el profesor, y mientras se le esta asignando una se le elimina de la lista y si este profesor la lista de profesor.cursos es igual a 0 pues continua al siguiente profesor). Luego ve si el curso actual se esta iterando lo da ese profesor

```
for profesor in self.profesores:
    if len(profesor.cursos) == 0:
        continue
    if curso_actual.nombre in profesor.cursos:
```

Iteración sobre aulas:

Se itera sobre las aulas para asignar un aula al curso actual.

```
for aula in self.aulas:
```

Verificación de disponibilidad:

Se verifica si el profesor y el aula están disponibles en el mismo horario usando la función _es_disponible.

```
if self._es_disponible(profesor, aula):
```

Lo que hace _es_disponible es básicamente validar si es posible la asignación de ese profesor con su disponibilidad y la disponibilidad del aula , es decir compara hasta encontrar si una de las disponibilidades del profesor esta en la disponibilidad del aula

```
def _es_disponible(self, profesor, aula):
 for horario_profesor in profesor.disponibilidad:
         if horario_profesor in aula.disponibilidad:
             return True
 return False
```

Cual horario se usará

Lo que se hace aquí es ver cuales de los horarios esta disponible para los 2 aula y profesor , y saber cual horario se asignara

```
horario = self.cualhorariro(profesor, aula)
```

cualHorario lo que hace es igual a lo de _es_disponible pero cualhorario lo que le hace return es a el horario en si por ejemplo “LUNESxMIERCOELES-9AM-A-10:55AM"”

```
 def cualhorariro(self, profesor, aula):
      for horario_profesor in profesor.disponibilidad:
              if horario_profesor in aula.disponibilidad:
                  return horario_profesor
```

Asignación de horario

Aquí ya con el curso_actual, profesor, aula y el horario que se acaba de elegir, ya se asigna

Esta asignación_actual se agrega a la lista de asignaciones

```
asignacion_actual = (curso_actual.nombre, profesor.nombre, aula.id, horario)

asignaciones.append(copy.deepcopy(asignacion_actual))
```

Eliminación de datos

Lo que hace esto es eliminar los datos, para que ya no se puedan utilizar, por ejemplo en un aula se elimina un disponibilidad de la lista de disponibilidad como podría ser “LUNESxMIERCOELES-9AM-A-10:55AM"” y esto para que no se le haga un asignación a otro profesor

```
profesor.eliminarDisponibilidad(horario)
aula.eliminarDisponibilidad(horario)
profesor.eliminarCursos(curso_actual.nombre)
```

Recursión y deshacer cambios:

- Se realiza una llamada recursiva para asignar el siguiente curso.
- Si la llamada recursiva devuelve True, significa que se encontró una solución completa.

Si no se encuentra una solución, se deshacen los cambios realizados en la asignación actual.

```
asignaciones.pop()
profesor.agregarCursos(curso_actual.nombre)
profesor.agregarDisponibilidad(horario)
aula.agregarDisponibilidad(horario)
```

Aquí se agrega lo que se elimino para que pueda ser asignado pero ahora el algoritmo intente de otra forma

Algoritmo backtraking

El algoritmo de asignación de horarios funciona de la siguiente manera

- Inicializa una lista vacía de asignaciones.
    - Para cada curso:
        - Para cada profesor:
            - Para cada aula:
                - Si el profesor y el aula son compatibles y el curso está disponible para ambos:
                - Asignar el curso al profesor y al aula en el horario disponible.
                - Eliminar el curso de la lista de cursos disponibles del profesor.
                - Eliminar la disponibilidad del horario del profesor y del aula.
                - Si esta asignación completa todos los cursos, entonces se ha encontrado una solución.
                - De lo contrario, retroceder y probar otra asignación

Aquí un ejemplo de cómo usarlo

1. **se crea las intencia de profesores,aulas y cursos**
2. **todos estos se ponen en una lista**
3. **Se crea una intencia de planificadorhorarios con las listas creadas de los profesores, aulas,cursos**
4. **Ejecutar el algoritmo para asignar los horarios**

Al final solo se muestra los horarios asignados a cada profesor con su aula y curso y hora

```
profesor1 = Profesor(1, "Profesor A", ["Matematicas","Algebra"], [LM9a11,LM11a1])
profesor2 = Profesor(2, "Profesor B", [ "Algebra","Matematicas"], [LM11a1,LM9a11])
profesor3 = Profesor(3, "Profesor C", [ "Matematicas","Algebra"], [MJ3a5,V1a5])

aula1 = Aula(101, 30, ["Proyector"])
aula2 = Aula(102, 30, ["Proyector"])
aula3 = Aula(103, 30, ["Proyector"])

matematicas = Curso(1, "Matematicas", 4, ["Proyector"])
algebra = Curso(2, "Algebra", 4, ["Proyector"])

profesores = [profesor1,profesor2,profesor3]
aulas = [aula1,aula2,aula3]
cursos = [matematicas, algebra,algebra,matematicas,matematicas,algebra]

#Crear una instancia de PlanificadorHorarios
planificador = PlanificadorHorarios(profesores, aulas, cursos)

# Ejecutar el algoritmo de asignación de horarios
planificador.asignar_horarios()

# Mostrar los horarios asignados
for asignacion in planificador.horarios_asignados:
    print(asignacion)
```