Guia/ reporte de backend

Backend de Proyecto Final

Christian Francisco Rojas Espinoza - **Backend**

Introducción

El backend de este proyecto desempeña un papel fundamental al abordar la tarea de generar horarios académicos de manera eficiente y personalizada. Su propósito principal es optimizar la asignación de profesores, aulas y cursos, considerando factores como la disponibilidad de salones, la agenda de los profesores y los requisitos específicos de cursos y equipamiento de aulas. Para lograr esto, se utiliza un algoritmo de backtracking implementado en Python que todos los datos que organiza vienen de los Excel de cada profesor, cursos, aulas que estas se extraen con la biblioteca de pandas.

Proceso de Generación de Horarios:

Entrada de Datos:

Los datos esenciales, como la disponibilidad de aulas y profesores, los requisitos de cursos y el equipamiento de aulas, se extraen de un archivo Excel mediante la biblioteca Pandas.

Estructuras de Datos:

Se utilizan listas de objetos para representar entidades clave, como profesores, aulas y cursos. Estos objetos se construyen a partir de los datos proporcionados.

Algoritmo de Backtracking:

El algoritmo de backtracking se encuentra implementado en objetos dedicados, como profesores, aulas y cursos. Estos objetos contienen la lógica necesaria para asignar horarios de manera eficiente. El algoritmo backtraking se escogió para este programa para poder encontrar el mejor horario posible para los profesores y de manera eficiente, ya que

Registro de Horarios Asignados:

La información sobre los horarios asignados se registra en un objeto llamado " PlanificadorHorarios". Aquí se almacenan los horarios específicos en los que cada profesor imparte un curso en una determinada aula.

Arquitectura del Código

El programa inicia cargando los datos del Excel para con esos datos crear cada objeto, que los objetos que crea son Profesores, Aulas, Cursos, de ahí se listan cada uno en una lista, solo los cursos se listan diferentes, ya que puede haber varias materias repetidas por los profesores, así que se necesita que verifique todas las materias que dan los profes y poner esa cantidad de veces una materia, y esto de listar es para poder iterarlos mejor para que entren en el algoritmo de backtraking, que esta está dentro de un objeto llamado PlanificadorHorarios, que en este es donde se guardan todos los horarios asignados que devolvió el algoritmo.

Backend Código

Contiene 5 archivos Python y 3 archivos Excel

1. **aula.py**
2. **cruso.py**
3. **profesor.py**
4. **planificador_horarios.py**
5. **Main.py**

- **aulas.xlsx**
- **cursos.xlsx**
- **profesores.xlsx**

aula.py

```python
class Aula:
    LM9a11 = "LUNESxMIERCOLES-9AM-A-10:55AM"
    LM11a1 = "LUNESxMIERCOLES-11AM-A-12:55PM"
    LM1a3 = "LUNESxMIERCOLES-1PM-A-2:55PM"
    LM3a5 = "LUNESxMIERCOLES-3PM-A-4:55PM"
    MJ9a11 = "MARTESxJUEVES-9AM-A-10:55AM"
    MJ11a1 = "MARTESxJUEVES-11AM-A-12:55PM"
    MJ1a3 = "MARTESxJUEVES-1PM-A-2:55PM"
    MJ3a5 = "MARTESxJUEVES-3PM-A-4:55PM"
    V9a1 = "VIERNES-9AM-A-1PM"
    V1a5 = "VIERNES-1PM-A-5PM"

    disponibilidad_default = [LM9a11, LM11a1, LM1a3, LM3a5, MJ9a11, MJ11a1, MJ1a3, MJ3a5, V9a1, V1a5]

    def __init__(self, id, capacidad, equipamiento, disponibilidad=None):
        self.id = id
        self.capacidad = capacidad
        self.equipamiento = equipamiento
        self.disponibilidad = disponibilidad.copy() if disponibilidad else Aula.disponibilidad_default.copy()

    def eliminarDisponibilidad(self, horario):
        if horario in self.disponibilidad:
            self.disponibilidad.remove(horario)

    def agregarDisponibilidad(self, horario):
        self.disponibilidad.append(horario)
```

Aula.py contiene la clase Aula que este tiene un id, capacidad, equipamiento, y también disponibilidad que tiene una disponibilidad default que es básicamente significa que tiene todos los horarios disponibles, cada que se crea un aula, ya mediante se llana es cuando se elimina una disponibilidad con la función eliminarDisponibilidad, y como se usa el algoritmo backtraking el programa si no encuentra la solución en una parte se devuelve a su última acción para tomar eso así que lo que elimino necesita agregarlo de nuevo por eso la función de agregarDisponibilidad. También algo importante que mencionar las constantes que están dentro de aula también están globales solo que a la hora de que están globales y usarlos para el default había varios problemas, como cuando se eliminaba una disponibilidad eliminaba la disponibilidad de todas las aulas.

cruso.py

```python
class Curso:
    def __init__(self, id, nombre, horas_semana, restricciones):
        self.id = id
        self.nombre = nombre
        self.horas_semana = horas_semana
        self.restricciones = restricciones
```

Curso.py solo contiene la clase Curso que esta está compuesto de id, nombre, horas Semana y restricciones.

Profesor.py

```python
class Profesor:
    def __init__(self, id, nombre, cursos=None, disponibilidad=None):
        self.id = id
        self.nombre = nombre
        self.cursos = cursos if cursos is not None else []
        self.disponibilidad = disponibilidad if disponibilidad is not None else []

    def eliminarDisponibilidad(self, horario):
        if horario in self.disponibilidad:
            self.disponibilidad.remove(horario)

    def agregarDisponibilidad(self, horario):
        self.disponibilidad.append(horario)

    def eliminarCursos(self, materia):
        if materia in self.cursos:
            self.cursos.remove(materia)

    def agregarCursos(self, materia):
        self.cursos.append(materia)
```

Profesor.py solo contiene la Clase profesor que solo contiene id, nombre, cursos, disponibilidad y 4 funciones eliminarDisponiblidad que este elimina una disponibilidad del profesor para a la hora de asignarle un horario con esa disponibilidad, o agregarDisponibilidad por si el algoritmo no encontró la solución agregando ese horario a tal aula, también esta el e elminarCurso y agregarCurso, que básicamente se utiliza de la misma forma que disponibilidad pero ahora con cursos, para saber que curso le falta asignar a ese profesor, y si todos los profes no tienen cursos que dar es que ya se asignaron todos los horarios.

PanificadorHorarios.py

```python
import copy

class PlanificadorHorarios:
    def __init__(self, profesores, aulas, cursos):
        self.profesores = profesores
        self.aulas = aulas
        self.cursos = cursos
        self.horarios_asignados = []

    def asignar_horarios(self):
        #Asigna horarios a los cursos utilizando el algoritmo de backtracking.
        self.horarios_asignados = []  # Limpiar asignaciones anteriores
        self._backtracking()
```

Este contiene la clase planifiacionHorario que contiene profesores, aulas, cursos y horariosAsignados, cuando se crea un planificador de horarios se le pasa la lista de los profesores, aulas y cursos, para poder asignarle un horario que es donde entre asignarHorario que adentro de este esta el donde se ejecuta el algoritmo backtraking.

```python
def _backtracking(self, asignaciones=[]): # Algoritmo de backtracking para asignar horarios a los cursos.

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

    curso_actual = self.cursos[len(asignaciones)]

    for profesor in self.profesores:
        if len(profesor.cursos) == 0:
            continue
        if curso_actual.nombre not in profesor.cursos:
            continue
        if curso_actual.nombre in profesor.cursos:
            for aula in self.aulas:
                if self._es_disponible(profesor, aula) and self.es_disponibleAulaCurso(curso_actual,aula):
                    horario = self.cualhorariro(profesor, aula)
                    asignacion_actual = (curso_actual.nombre, profesor.nombre, aula.id, horario)
                    asignaciones.append(copy.deepcopy(asignacion_actual))
                    profesor.eliminarDisponibilidad(horario)
                    aula.eliminarDisponibilidad(horario)
                    profesor.eliminarCursos(curso_actual.nombre)

                    if self._backtracking(asignaciones):
                        print("Solución encontrada")
                        return True

                    asignaciones.pop()
                    print("RESET")
                    profesor.agregarCursos(curso_actual.nombre)
                    profesor.agregarDisponibilidad(horario)
                    aula.agregarDisponibilidad(horario)
                    print(f"Deshaciendo asignación de {curso_actual.nombre} en {aula.id}")
            print(f"No se encontró asignación para {curso_actual.nombre}")
        return False
```

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*

Explicación del código de backtraking.

Función de backtraking :

- 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) de ahí entra a otro if verificando si en el curso actual que se está iterando el profesor da ese curso, y si no es asi, pues cambia el profesor. Luego ve si el curso actual se está iterando lo da ese profesor

```
    for profesor in self.profesores:
        if len(profesor.cursos) == 0:
            continue
        if curso_actual.nombre not in profesor.cursos:
            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 y esDisponbileAulaCurso.

```
if self._es_disponible(profesor, aula) and self.es_disponibleAulaCurso(curso_actual,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
```

Es disponibleAulaCurso, lo que hace es validar si se puede asignar esa aula con el curso que se están utilizando en esa iteración

```
def es_disponibleAulaCurso(self, curso, aula):
    for restriciones in curso.restricciones:
            if restriciones in aula.equipamiento:
                return True
    return False
```

Cual horario se usará

Lo que se hace aquí es ver cuáles de los horarios está disponible para los 2 aula y profesor , y saber cuál 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 retorna es a el horario que se usara. 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 eliminó para que pueda ser asignado pero ahora el algoritmo intente de otra forma

Main.py

Main.py lo primero que tiene es el import de pandas que es la biblioteca usada para poder leer los archivos Excel, de ahí se importan todos los otros códigos Python es decir se importan todas las otras clases

```python
import pandas as pd
from profesor import Profesor
from aula import Aula
from curso import Curso
from planificador_horarios import PlanificadorHorarios
```

Luego se definen unas constantes globales para los horarios de profesores y aulas, y se leen los archivos Excel.

```python
# Definición de horarios
LM9a11 = "LUNESxMIERCOLES-9AM-A-10:55AM"
LM11a1 = "LUNESxMIERCOLES-11AM-A-12:55PM"
LM1a3 = "LUNESxMIERCOLES-1PM-A-2:55PM"
LM3a5 = "LUNESxMIERCOLES-3PM-A-4:55PM"
MJ9a11 = "MARTESxJUEVES-9AM-A-10:55AM"
MJ11a1 = "MARTESxJUEVES-11AM-A-12:55PM"
MJ1a3 = "MARTESxJUEVES-1PM-A-2:55PM"
MJ3a5 = "MARTESxJUEVES-3PM-A-4:55PM"
V9a1 = "VIERNES-9AM-A-1PM"
V1a5 = "VIERNES-1PM-A-5PM"

#Lee los archivos exel
profesores_df = pd.read_excel('profesores.xlsx')
aulas_df = pd.read_excel('aulas.xlsx')
cursos_df = pd.read_excel('cursos.xlsx')
```

Lo que se hace a continuación con los archivos de Excel que se leyeron, ahora se creara los objetos con la información de cada uno, y de ahí se agregaran a una lista de profesores, aulas, y cursos para que estos sean utilizados en el algoritmo.

```
profesores = []
for index, row in profesores_df.iterrows():
    if 'cursos ' in profesores_df.columns:
        cursoss = row['cursos '].split(',') if pd.notna(row['cursos ']) else []
        disponibilidad = row['disponibilidad'].split(',') if pd.notna(row['disponibilidad']) else []
        profesor = Profesor(row['id '], row['nombre'], cursoss, disponibilidad)
        profesores.append(profesor)
    else:
        print("Column 'cursos ' no encontrada.")

aulas = []
for index, row in aulas_df.iterrows():

    if 'id ' in aulas_df.columns:
        equipamiento = row['equipamiento'].split(',') if pd.notna(row['equipamiento']) else []
        aula = Aula(row['id '], row['capacidad'], equipamiento)
        aulas.append(aula)
    else:
        print("Column 'id ' no encontrada.")

cursos = []
for index, row in cursos_df.iterrows():
    restricciones = row['restricciones'].split(',') if pd.notna(row['restricciones']) else []
    curso = Curso(row['id'], row['nombre'], row['horas_semana'], restricciones)
    cursos.append(curso)
```

Pero la lista de cursos, solo hace la lista de cursos disponibles para dar, así que se crea otra lista, esta se crea en base a los cursos que dan los profesores, esto se hace viendo que curso da un profesor y agregándolo a la lista cursosFinales, ya que hay cursos que se repiten en los profes y se necesita saber cuántos cursos se darán y cuáles y cursosFinales ayuda en eso.

```
cursosFinales=[]
for curso  in cursos:
    for profe in profesores:
        if curso.nombre in profe.cursos:
            cursosFinales.append(curso)
```

Finalmente se crea la instancia de planificador Horarios y de ahí se ejecuta la función de asignar horarios, que ahí adentro es donde está el algoritmo backtraking.

```
#Crear una instancia de PlanificadorHorarios
planificador = PlanificadorHorarios(profesores, aulas, cursosFinales)
# Ejecutar el algoritmo de asignación de horarios
planificador.asignar_horarios()
```

Y para ver cada asignación de horario, se guardan en una lista de Horario Asignado de la clase Planificador Horarios

Un ejemplo de como verlo en la terminal

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

Ejemplo:

Materia, Profesor, Aula, Horario

```
('MATEMATICA DISCRETA', 'Beatriz Hernández Pérez', 101, 'VIERNES-1PM-A-5PM')
('CALCULO DIFERENCIAL E INTEGRAL', 'Carlos Rodríguez García', 101, 'MARTESxJUEVES-9AM-A-10:55AM')
('MECANICA', 'Carolina García Sánchez', 102, 'LUNESxMIERCOLES-9AM-A-10:55AM')
('ADMINISTRACION DE PROYECTOS TECNOLOGICOS', 'Alberto Castro Martínez', 101, 'LUNESxMIERCOLES-1PM-A-2:55PM')
('EXPRESION ORAL Y ESCRITA', 'Alejandro García Fernández', 103, 'MARTESxJUEVES-9AM-A-10:55AM')
('PROGRAMACION ORIENTADA A OBJETOS', 'Claudia Sánchez Rodríguez', 102, 'VIERNES-1PM-A-5PM')
('ALGEBRA LINEAL', 'Carlos Rodríguez García', 101, 'MARTESxJUEVES-3PM-A-4:55PM')
('ECUACIONES DIFERENCIALES', 'Carlos Rodríguez García', 102, 'LUNESxMIERCOLES-11AM-A-12:55PM')
('CIRCUITOS ELECTRICOS Y ELECTROMAGNETISMO', 'Eduardo Pérez Sánchez', 102, 'LUNESxMIERCOLES-3PM-A-4:55PM')
('SISTEMAS DIGITALES', 'Eduardo Pérez Sánchez', 103, 'VIERNES-1PM-A-5PM')
```

Conclusión

Al concluir esta guía sobre el backend, he profundizado completamente en la comprensión detallada de cada aspecto del código. El núcleo esencial de este proceso ha sido el algoritmo que organiza todas las asignaciones de horarios. Descubrir cómo funciona este algoritmo de backtracking y cómo implementarlo para lograr asignaciones de horarios efectivas ha sido un enriquecedor.

Cada sección de esta Guia/Reporte se ha hecho para poder comprender cada parte de este código, desde la inicialización de listas con los objetos hasta como el algoritmo backtraking funciona paso a paso en el código.

Referencias

Datta, S., & Datta, S. (2022, 11 noviembre). *Backtracking algorithm |Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/backtracking-algorithms#:~:text=Backtracking%20is%20an%20algorithmic%20technique,satisfy%20them%20will%20be%20removed.

Chio Code. (2022, 13 octubre). Explicando backtracking | Fuerza bruta pero elegante [Vídeo]. YouTube. https://www.youtube.com/watch?v=ip2jC_kXGtg