Grafo de Pensum: Sistema de Gestión Curricular Basado en Estructuras de Datos

David Andrés Flores Valle (FV241458), Néstor Alejandro González García (GG241648), Tariq Alberto Ventura Arteaga (VA240315)

Resumen—Se presenta un sistema de gestión curricular para diversas carreras, implementado mediante estructuras de datos eficientes. El núcleo del modelo es un grafo dirigido acíclico (DAG) donde cada materia es representada como un nodo conectado a sus cursos prerrequisito y a los que desbloquea. Cada nodo almacena dos listas: requisitos (cursos previos) y desbloquea (cursos que dependen de él). Los datos académicos se guardan en MongoDB (una base de datos NoSQL) usando documentos JSON para representar planes de estudio de manera estructurada. Se justifican las decisiones técnicas con referencias a enfoques basados en grafos, bases de datos NoSQL y algoritmos de búsqueda. Además, se incluyen diagramas UML ilustrativos y se discuten las ventajas del enfoque propuesto.

Palabras clave — grafo dirigido acíclico, pensum, MongoDB, NoSQL, DFS, búsquedas en grafos, ordenamiento topológico.

# Introducción

La planificación del pensum de una carrera universitaria implica organizar las materias con sus dependencias de prerrequisitos. Una forma natural de representar los cursos y sus relaciones es mediante un grafo dirigido acíclico (DAG): cada nodo representa una materia y una arista dirigida indica que una materia es prerrequisito de otra. Este enfoque, también conocido como red de prerrequisitos, facilita visualizar la estructura curricular y calcular rutas de estudio. Por ejemplo, al aprobar un curso, todas las materias apuntadas desde él quedan habilitadas para cursar.

El presente trabajo describe un sistema de gestión curricular basado en DAG desarrollado en la Universidad Don Bosco. El modelo almacena cada materia como nodo con listas de *requisitos* y *desbloquea.* Los datos se guardan en MongoDB aprovechando su flexibilidad para datos académicos semiestructurados. Además, se emplea un algoritmo DFS para obtener un orden topológico de materias que garantiza cumplir prerrequisitos. A continuación, se explican los detalles de este diseño.

# Modelado Curricular con Grafos Dirigidos

class Materia:

codigo: string

nombre: string

requisitos: List // códigos de materias previas

desbloquea: List // códigos de materias dependientes

El modelo central es un grafo , donde cada vértice corresponde a una materia del pensum, y cada arista dirigida indica que la materia es prerrequisito de *v*. Al ser acíclico, se garantiza que no existen ciclos de prerrequisitos. Por ejemplo, si la materia A es prerrequisito de B y B de C, nunca habrá un camino inverso que cree un ciclo. Matemáticamente, este grafo permite obtener fácilmente un ordenamiento topológico para determinar secuencias de cursado viables.

Cada nodo-materia contiene dos listas principales:

**requisitos:** lista de materias previas que se requieren cursar antes de esta materia.

**desbloquea:** lista de materias que quedan habilitadas al aprobar esta materia.

Estas listas forman una lista de adyacencia típica de grafos. En efecto, cada materia tiene un arreglo con los identificadores de los cursos relacionados. Esta representación basada en lista de adyacencia permite recorrer eficientemente el grafo: por ejemplo, para listar las materias disponibles tras aprobar X basta con ver la lista desbloquea de X. La Figura 1 ilustra un ejemplo de grafo curricular sencillo con 6 materias: A, B, C son prerrequisitos para X e Y, mientras Z está aislado.

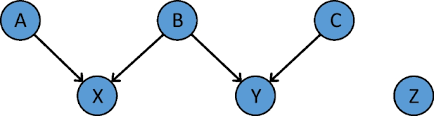


Figura 1. Grafo de pensum ejemplo. Cada nodo es una materia. Las flechas indican relaciones de prerrequisito: por ejemplo, A y B son requisitos para X. Z no tiene dependencias (nodo aislado).

En términos de implementación, cada materia puede representarse con una estructura o clase Materia que incluye atributos como código, nombre y dos listas (requisitos, desbloquea). Al registrar una relación de prerrequisito (por ej. “A es requisito de B”), se inserta B en la lista desbloquea de A y se inserta A en la lista requisitos de B. De esta forma, todos los enlaces del grafo quedan reflejados en las estructuras internas de los nodos

# Estructuras de Datos: Arreglos y Listas

Para implementar las listas *requisitos* y *desbloquea* se emplean arreglos dinámicos (por ejemplo, ArrayList en Java o list en Python). Estos arreglos permiten un acceso rápido y pueden crecer según se añaden cursos al pensum. Un pseudocódigo simplificado de la clase *Materia* sería:

Cada materia inicializa sus listas como vacías; luego, al procesar el pensum, se agregan los códigos correspondientes a cada lista. Este esquema corresponde exactamente a una representación por listas de adyacencia, en la que cada nodo conoce sus vecinos adyacentes en cada dirección. Las operaciones comunes sobre estas listas incluyen la búsqueda de materias sin prerrequisitos (*requisitos* vacíos) y la obtención de la lista de cursos disponibles tras aprobar cierta materia (*desbloquea*). Gracias a esta estructura, podemos iterar directamente sobre las dependencias de un nodo sin escanear toda la colección de materias.

# Gestión de Datos con MongoDB

La persistencia de la información se realiza con MongoDB, una base de datos NoSQL orientada a documentos. Cada entidad del sistema se guarda como un documento JSON flexible, lo que simplifica acomodar esquemas cambiantes (por ejemplo, al agregarse nuevas materias o campos). MongoDB fue elegido por su **esquema flexible** y fácil escalabilidad horizontal, características útiles cuando los planes de estudio pueden variar año con año.

Ejemplos de documentos JSON para este sistema:

*{*

*"\_id": Objec t Id("..."),*

*"anio": 2025,*

*"carrera": "Ingeniería en Ciencias de la Computación",*

*"materias": ["MATH101", "FIS102", "CS103", "..."]*

*}*

A **Pensum** (contiene año, carrera y lista de materias):

B **Materia** (código, nombre y dependencias):

*{*

*"\_id": "CS103",*

*"nombre": "Programación I",*

*"requisitos": ["MATH101", "CS102"],*

*"desbloquea": ["CS104", "CS105"]*

*}*

C **Carrera** (identificador de carrera y referencia al pensum):

*{*

*"\_id": "ICC",*

*"nombre": "Ingeniería en Ciencias de la Computación",*

*"pensum\_id": ObjectId("...")*

*}*

D **Usuario** (estudiante o docente):

*{*

*"\_id": ObjectId("..."),*

*"nombre": "Ana Romero",*

*"carrera": "ICC",*

*"materias\_cursadas": ["MATH101", "CS102"]*

*}*

En MongoDB se pueden definir índices en campos clave (por ejemplo, en *Materia.\_id* o en los campos de búsquedas frecuentes) para mejorar el rendimiento de consulta. Sin índices, una consulta debe escanear todos los documentos; con índices, MongoDB localiza rápidamente los documentos coincidentes. Esto permite responder eficientemente preguntas como “¿qué materias quedan habilitadas si el estudiante aprobó X?” o “¿quiénes cursan la carrera Y?”.

# Ordenamiento Topológico mediante DFS

Para generar una secuencia de estudio válida se aplica un ordenamiento topológico sobre el DAG de materias. Este ordenamiento produce una lista donde cada materia precede a las que dependen de ella. Una implementación típica usa Depth-First Search (DFS): al recorrer el grafo, cuando se termina de visitar los vecinos de un nodo, se inserta el nodo en una pila. Finalmente, vaciar la pila arroja el orden topológico requerido.

Los pasos son: 1. Marcar todas las materias como no visitadas. 2. Para cada materia no visitada, ejecutar *DFS(materia).*

3. En *DFS(v)*: marcar *v* como visitado; para cada curso *w* en *v.desbloquea* no visitado, llamar recursivamente a *DFS(w)*.

4. Una vez procesados todos los dependientes de *v*, apilar *v*.

5. Al terminar, vaciar la pila para obtener el orden de cursado.

Este algoritmo asegura que, en el orden final, cada curso aparece antes que los que dependen de él. Si existiera un ciclo de prerrequisitos, el DFS lo detectaría, indicando un error en el diseño curricular. En resumen, el ordenamiento topológico facilita proponer al estudiante un itinerario de materias que cumple todas las relaciones de prerrequisito.

# Conclusiones

Se ha propuesto un modelo integral de gestión curricular donde el pensum se representa como un grafo dirigido acíclico. Esta representación estructurada simplifica el análisis de prerrequisitos y la planificación de cursos.

El prototipo desarrollado en la Universidad Don Bosco (carrera de Ingeniería en Ciencias de la Computación) se encuentra en fase de validación con datos reales de pensums. Como trabajo futuro se planea incorporar interfaces gráficas interactivas y análisis estadísticos sobre el progreso de los estudiantes. En conjunto, el Grafo de Pensum demuestra cómo estructuras avanzadas de datos pueden apoyar la gestión curricular de forma efectiva.

# Referencias

[1] P. Stavrinides y K. M. Zuev, Course-prerequisite networks for analyzing and understanding academic curricula, Applied Network Science, vol. 8, art. 19, 2023 .

[2] GeeksforGeeks, Topological Sorting for Directed Acyclic Graph (DFS), 2024 (versión web) .

[3] MongoDB Inc., What is NoSQL?, Documentación MongoDB (en línea), 2023 .

[4] GeeksforGeeks, Performance Considerations in MongoDB Indexes, 2025 (versión web) .

[5] Youcademy, Directed Acyclic Graphs (DAGs), artículo en línea . E. H. Miller, "A note on reflector arrays," *IEEE Trans. Antennas Propagat.*, a ser publicado.

[6] Course-prerequisite networks for analyzing and understanding academic curricula | Applied Network Science | Full Text <https://appliednetsci.springeropen.com/articles/10.1007/s41109-023-00543-w>

[7] What Is NoSQL? NoSQL Databases Explained | MongoDB <https://www.mongodb.com/resources/basics/databases/nosql-explained>

[8] Directed Acyclic Graphs (DAGs)

<https://youcademy.org/directed-acyclic-graphs/>

Topological Sorting | GeeksforGeeks <https://www.geeksforgeeks.org/topological-sorting/>

[9] Performance Considerations in MongoDB Indexes | GeeksforGeeks <https://www.geeksforgeeks.org/performance-considerations-in-mongodb-indexes/>

[10] ChatGPT, OpenAI (2025). \*Asistente de redacción y revisión técnica\*.

# Anexos

[1] Codigo Fuente del proyecto:

<https://github.com/davidghjg2/Grafo-pensum>

[2] Mongo DB extensión para C#

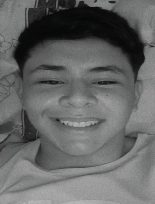
<https://www.mongodb.com/docs/languages/csharp/>

[3] Mongo DB Drivers para C#

<https://www.mongodb.com/docs/drivers/csharp/current/?msockid=1bf5b4923cfd6b7e335ba1f83d576a07>

# Biografías

 **[1] David Andrés Flores Valle** es un futuro ingeniero en ciencias de la computación, extrovertido y con algunas habilidades blandas bien desarrolladas y que le gusta todo lo relacionado con la lógica de sistemas y su parte estética, sabe tejer y armar diversos Cubos de Rubik.

 **[2] Néstor Alejandro Gonzales García** es estudiante de Ingeniería en Ciencias de la Computación, comprometido con su formación académica y profesional. Interesado en el desarrollo de software, especialmente en el área de aplicaciones móviles y web. Dispuesto a aprender nuevas tecnologías y adaptarse a distintos entornos de trabajo. Aficionado a los videojuegos y los deportes. Con buena disposición para el trabajo en equipo.

 **[2] Tariq Alberto Ventura Arteaga** nació el 15 de septiembre de 2004 en San Salvador,

El Salvador Curso su Bachillerato en el Instituto Técnico Ricaldone con un Bachillerato Técnico Vocacional en Desarrollo de Software y actualmente cursa la carrera de Ingeniería en Ciencias de la Computación en la Universidad Don Bosco.