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.

I. 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 como parte de un proyecto académico en la Universidad Don Bosco. El modelo representa cada materia como un nodo con listas de materias prerrequisito y materias que se desbloquean tras aprobarla. Los datos académicos son persistidos utilizando MongoDB, una base de datos NoSQL orientada a documentos que permite representar de forma flexible los elementos del pensum en formato JSON.

Además de detallar el modelado del grafo y la estructura de datos utilizada, se implementa un algoritmo de búsqueda en profundidad (DFS) para obtener un orden topológico de las materias, garantizando así una secuencia válida de cursado que respeta los prerrequisitos establecidos. El sistema cuenta también con una interfaz gráfica interactiva que permite al usuario explorar visualmente el pensum, consultar materias habilitadas y navegar las relaciones entre cursos. Este trabajo pretende sentar las bases para herramientas más avanzadas que automaticen y optimicen la planificación académica en instituciones de educación superior.

II. Modelado Curricular con Grafos Dirigidos

El modelo central es un grafo $G = (V, E)$, donde cada vértice $v \in V$ corresponde a una materia del pensum, y cada arista dirigida $(u \rightarrow v) \in E$ indica que la materia u 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.

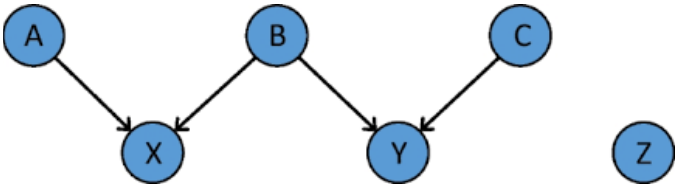

Figura 1. Grafo de pensum ejemplo. Cada nodo es una materia. Las flechas indican relaciones de prerrequisito: por ejemplo, A y B son requisitos para X. Z no tiene dependencias (nodo aislado).

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

III. 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:

```
class Materia:
    codigo: string
    nombre: string
    requisitos: List // códigos de materias previas
    desbloquea: List // códigos de materias dependientes
```

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.

IV. 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:

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

```
{
  "_id": Objec t Id("..."),
  "anio": 2025,
  "carrera": "Ingeniería en Ciencias de la Computación",
  "materias": ["MATH101", "FIS102", "CS103", "..."]
}
```

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?".

V. 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.

VI. Ordenamiento por Niveles con BFS

Además del ordenamiento topológico con DFS, se implementó también un enfoque de ordenamiento basado en BFS (Breadth-First Search), útil para representar la secuencia de materias por niveles o semestres. Este algoritmo se basa en calcular los niveles de cada nodo en función de la distancia desde los cursos sin prerrequisitos.

Pasos principales:
1. Inicializar una cola con todos los nodos sin prerrequisitos.
2. Asignar nivel 0 a estos nodos.
3. Mientras la cola no esté vacía:
- Sacar un nodo actual.
- Para cada curso que desbloquea:
- Reducir su contador de prerrequisitos pendientes.
- Si ya no tiene prerrequisitos por cursar, añadirlo a la cola y asignarle nivel actual + 1.

Este enfoque produce una clasificación de las materias por niveles crecientes, útil para proponer una estructura semestral aproximada del plan de estudios.

VII. Generalidades utilizadas en el proyecto

A. Arquitectura Hexagonal:

El diseño del sistema sigue principios de arquitectura hexagonal (o arquitectura de puertos y adaptadores), separando el núcleo de lógica de negocio del resto de componentes como bases de datos o interfaces gráficas.

Esta separación permite que el grafo de materias y los algoritmos de búsqueda operen independientemente del almacenamiento (MongoDB) o de la visualización (interfaz gráfica). Así, es posible sustituir estos componentes sin afectar el núcleo del sistema.

La lógica central (grafo, clases de materia, ordenamientos) reside en el dominio. MongoDB se conecta a través de un adaptador de persistencia, y la interfaz gráfica es otro adaptador de entrada/salida. Esta organización mejora la mantenibilidad, escalabilidad y testabilidad del sistema.

B. Estándares de Desarrollo:

Durante la implementación del sistema se aplicaron principios de Clean Code y los principios de diseño SOLID para asegurar un código mantenible, legible y escalable. Algunas prácticas destacadas incluyen:

- Single Responsibility: cada clase y función cumple una única responsabilidad bien definida.

- Open/Closed: el sistema permite extender funcionalidades (por ejemplo, nuevos algoritmos de ordenamiento) sin modificar el código base.

- Liskov Substitution: se respetan las jerarquías de clases sin romper funcionalidades.

- Interface Segregation y Dependency Inversion se aplican en el diseño modular del sistema y su separación entre lógica, datos y visualización..

VIII. Alcances y Limitaciones del proyecto

A. Alcances:

El sistema grafo de pensum permite una representación estructurada, visual e interactiva de los planes de estudio mediante un grafo dirigido acíclico (dag). Sus principales alcances incluyen:

- visualización gráfica del pensum: la interfaz gráfica desarrollada permite explorar visualmente las materias, sus relaciones de prerrequisito y las materias habilitadas según el avance del estudiante.

- análisis automático de prerrequisitos: se puede determinar de forma automática qué materias están habilitadas en función del historial académico del usuario.

- generación de rutas válidas de estudio: mediante ordenamiento topológico, el sistema sugiere itinerarios que respetan las dependencias curriculares.

- gestión flexible de datos académicos: gracias al uso de mongodb, se puede representar y almacenar planes de estudio de forma dinámica y escalable.

B. Limitaciones:

A pesar de sus capacidades, el sistema aún presenta algunas limitaciones que se deberán abordar en futuras versiones:

- soporte limitado para estructuras curriculares complejas: el modelo actual no contempla casos como materias optativas, equivalencias, prerrequisitos alternativos ("una de las siguientes") o convalidaciones externas.

- gestión de versiones del pensum: no se ha implementado un sistema para llevar control de versiones históricas de planes de estudio ni para comparar modificaciones entre años.

- falta de integración con plataformas institucionales: actualmente el sistema opera de forma aislada y no está vinculado con sistemas administrativos o plataformas educativas existentes.

- dependencia de conocimientos técnicos para mantenimiento: aunque la interfaz facilita el uso, la configuración y mantenimiento del sistema puede requerir habilidades en bases de datos nosql y manipulación de grafos.

IX. Conclusión

Se ha propuesto un modelo integral de gestión curricular en el cual el pensum se representa como un grafo dirigido acíclico (DAG), permitiendo una representación lógica y estructurada

de las relaciones entre materias. Este enfoque facilita el análisis automático de los prerrequisitos, el diseño de trayectorias de estudio válidas, y la gestión flexible de los contenidos curriculares mediante bases de datos NoSQL como MongoDB.

La implementación del sistema ha demostrado su viabilidad y funcionalidad en el contexto de la carrera de Ingeniería en Ciencias de la Computación de la Universidad Don Bosco, donde actualmente se encuentra en fase de validación con datos reales. Además, se ha desarrollado una interfaz gráfica interactiva que permite a los usuarios explorar visualmente el pensum y sus relaciones, mejorar la experiencia de navegación y apoyar la toma de decisiones académicas.

Entre los beneficios más destacados se encuentran la automatización del análisis de prerrequisitos, la posibilidad de generar ordenamientos topológicos personalizados para cada estudiante, y la adaptabilidad del sistema ante cambios curriculares. No obstante, también se han identificado limitaciones que deben ser abordadas en futuras iteraciones, como la falta de soporte para materias optativas y la gestión de versiones del pensum.

Como trabajo futuro se plantea la incorporación de análisis estadísticos sobre el progreso estudiantil, la integración con plataformas académicas existentes y la expansión del sistema para abarcar múltiples carreras e instituciones. En conjunto, el sistema "Grafo de Pensum" constituye una base sólida sobre la cual construir herramientas digitales modernas para la gestión curricular universitaria, apoyadas en fundamentos algorítmicos y estructuras de datos robustas.

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.

X. 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*.

XI. 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

XII. 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.