PROYECTO 1 – IPC2

202307705 – Josué David Velásquez Ixchop

Resumen

El proyecto descrito consiste en el desarrollo de una solución de software en Python, enfocada en la implementación de tipos de datos abstractos (TDA) y la visualización de datos mediante Graphviz, bajo el paradigma de programación orientada a objetos (POO). El objetivo principal es procesar matrices de frecuencia de accesos en bases de datos distribuidas, generando matrices reducidas para optimizar los costos de transmisión de datos.

Este trabajo es relevante en el contexto técnico actual, ya que aborda un problema NP-Hard común en la optimización de bases de datos distribuidas. La técnica de reducción de matrices mediante patrones de acceso es innovadora y esencial para minimizar tiempos de procesamiento y el uso de memoria, especialmente en la gestión de grandes volúmenes de datos.

Las conclusiones destacan la importancia de una correcta implementación de TDA para manejar la complejidad del problema, así como la utilidad de herramientas como Graphviz para la visualización. Además, se subraya la eficiencia en el manejo de recursos al resolver problemas NP-Hard mediante la agrupación de patrones.

Palabras clave

1. Optimización
2. Visualización
3. Programación Orientada a Objetos
4. Agrupamiento
5. Bases de datos

Abstract

The described project involves the development of a software solution in Python, focused on implementing abstract data types (ADT) and data visualization using Graphviz, under the object-oriented programming (OOP) paradigm. The main objective is to process access frequency matrices in distributed databases, generating reduced matrices to optimize data transmission costs. This work is relevant in the current technical context as it addresses a common NP-Hard problem in the optimization of distributed databases. The matrix reduction technique using access patterns is innovative and essential for minimizing processing time and memory usage, especially when managing large data volumes. The conclusions emphasize the importance of correctly implementing ADTs to handle the problem's complexity and the usefulness of tools like Graphviz for visualization. Additionally, the efficiency in resource management when solving NP-Hard problems through pattern grouping is highlighted.

Keywords

1. *Optimization*
2. *Visualization*
3. *Object Oriented Programming*
4. *Clustering*
5. *Databases*

Introducción

El proyecto se enfoca en desarrollar una solución integral que implemente tipos de datos abstractos (TDA) y visualización de datos utilizando Graphviz, aplicando los principios de programación orientada a objetos (POO) en Python. La problemática principal es el alojamiento de objetos de bases de datos en sitios distribuidos, minimizando el costo total de transmisión de datos, un problema NP-Hard. Para abordar este desafío, se implementa una metodología de agrupamiento basada en patrones de acceso, reduciendo las matrices de frecuencia de acceso. El propósito de este ensayo es detallar cómo se aborda el problema, desde la implementación en POO hasta la visualización gráfica de los resultados, y cómo estas soluciones contribuyen a la eficiencia en la transmisión de datos en redes distribuidas. ¿Cómo podemos optimizar aún más la distribución de datos en bases de datos distribuidas? Este ensayo explorará las posibles respuestas.

Desarrollo del tema

Este proyecto se orienta hacia la optimización de matrices de frecuencia de acceso en bases de datos distribuidas, un problema caracterizado por su alta complejidad computacional y su clasificación como NP-Hard. La solución propuesta en Python utiliza tipos de datos abstractos (TDA), listas circulares y herramientas de visualización como Graphviz. A continuación, se detallan las diferentes etapas y componentes que conforman esta solución, desde la estructura de datos hasta el procesamiento de archivos y la generación de gráficos.

Subtema 1: Clases Fundamentales del Proyecto

El proyecto se basa en una serie de clases cuidadosamente diseñadas para manejar las estructuras de datos y las operaciones necesarias para procesar y optimizar matrices de frecuencia de acceso.

- **Clase Par:** Esta clase encapsula un par de coordenadas (x, y) que se utilizan como clave para acceder a los valores de las matrices. La clase incluye métodos para realizar operaciones matemáticas básicas, como suma, resta y escalado, lo que facilita la manipulación de las coordenadas dentro de la matriz. Además, implementa métodos de comparación y hashing, permitiendo que los objetos Par se utilicen como claves en estructuras de datos que requieren unicidad, como los diccionarios. Este diseño permite que las matrices de frecuencia de acceso sean manipuladas de manera eficiente, al mismo tiempo que mantiene la integridad de las coordenadas utilizadas para identificar cada valor.
- **Clase Mapa:** El Mapa es una estructura de datos que actúa como un diccionario personalizado, optimizado para almacenar y recuperar valores asociados con instancias de la clase Par. A diferencia de los diccionarios estándar de Python, el Mapa se construye específicamente para trabajar con la lógica del proyecto, proporcionando métodos especializados para agregar, obtener y verificar la existencia de elementos.

Internamente, el Mapa se implementa como una lista enlazada, donde cada nodo contiene una clave (un objeto Par) y su valor asociado. Esta estructura permite la gestión eficiente de los datos en las matrices, ya que facilita la manipulación y búsqueda de elementos en función de sus coordenadas.

- **Clase Lista:** La clase Lista implementa una lista enlazada simple, diseñada para almacenar secuencias de elementos de manera eficiente. Esta estructura es crucial para gestionar las filas de las matrices y los patrones de acceso dentro de las mismas. La clase incluye métodos para agregar elementos, obtener elementos por índice y calcular el tamaño de la lista. Además, se implementa un iterador, lo que permite recorrer la lista de manera sencilla, facilitando operaciones como la agrupación de filas con patrones de acceso similares. Esta capacidad de iterar sobre la lista es particularmente útil en el proceso de agrupación, donde es necesario comparar múltiples filas y sus patrones de acceso.
- **Clase Nodo:** El Nodo es la unidad básica de la lista circular y representa una matriz de frecuencia de acceso. Cada nodo almacena el nombre de la matriz, sus dimensiones (n y m), y un objeto Mapa que contiene los datos de la matriz. Los métodos dentro de la clase Nodo permiten obtener y establecer valores en la matriz, así como mostrarla en formato tabular. Este diseño modular facilita el manejo de múltiples matrices dentro del sistema, permitiendo que cada matriz se almacene como un nodo independiente dentro de la lista circular. Además, la clase Nodo es fundamental para implementar operaciones como la visualización de las matrices y la generación de gráficos con Graphviz.
- **Clase ListaCircular:** La ListaCircular es una lista circular simplemente enlazada, diseñada para almacenar múltiples matrices de frecuencia de acceso. Esta estructura es clave para manejar el ciclo continuo de matrices dentro del sistema, permitiendo recorrer todas las matrices sin necesidad de reiniciar el recorrido. La clase incluye métodos para agregar matrices (nodos), buscar matrices por nombre, eliminar matrices y mostrar todas las matrices almacenadas. Este diseño permite una gestión eficiente de múltiples matrices, lo que es esencial para el procesamiento de grandes volúmenes de datos. Además, la lista circular facilita la implementación de funciones que requieren un acceso cíclico a las matrices, como la generación de gráficos y la comparación de patrones de acceso.

Subtema 2: Procesamiento de Archivos XML y Carga de Matrices

Una de las funcionalidades clave del proyecto es la capacidad de cargar matrices de frecuencia de acceso desde archivos XML. Esta funcionalidad se implementa a través de la función procesar_archivo, que utiliza la biblioteca xml.etree.ElementTree para parsear el archivo XML y extraer las matrices y sus datos asociados. El proceso de carga comienza con la lectura del archivo XML, seguido de la validación de su estructura para asegurar que cumple con el formato esperado.

- **Validación de Estructura y Datos:** El archivo XML debe comenzar con la etiqueta

<matrices>, que contiene múltiples matrices representadas por la etiqueta <matriz>. Cada matriz tiene atributos nombre, n (número de filas) y m (número de columnas). Dentro de cada matriz, los datos individuales se representan con la etiqueta <dato>, que incluye atributos x (fila) y y (columna), así como el valor correspondiente. La función procesar_archivo verifica que estos atributos existan y que sus valores sean válidos, asegurando que los datos de la matriz sean consistentes antes de almacenarlos en la lista circular.

- **Almacenamiento en la ListaCircular:** Una vez validada, cada matriz se almacena como un nodo en la ListaCircular. Esta estructura permite manejar múltiples matrices simultáneamente, facilitando su acceso y manipulación en otras partes del sistema. La lista circular es ideal para este tipo de procesamiento, ya que permite un acceso cíclico a las matrices, lo que es útil para operaciones como la comparación de patrones de acceso y la generación de gráficos.

Subtema 3: Generación de Matrices Binarias y Reducción de Datos

Una de las tareas más importantes en el proyecto es la transformación de las matrices de frecuencia de acceso en matrices binarias. Este proceso simplifica la identificación de patrones de acceso, lo que a su vez facilita la agrupación de filas con patrones similares.

- **Conversión a Matrices Binarias:** La función convertir_a_binario recorre cada celda de la matriz y convierte su valor en binario. Si el valor es mayor que 0, se convierte en 1; de lo contrario, se convierte en 0. Esta conversión reduce la complejidad de la matriz, permitiendo un análisis más rápido y eficiente de los patrones de acceso.
- **Agrupación de Patrones:** Una vez convertidas en binario, las matrices se analizan para identificar filas con patrones de acceso similares. Este proceso de agrupación se realiza comparando las filas de la matriz binaria y agrupando aquellas que tienen patrones idénticos. La agrupación se implementa mediante la creación de una nueva matriz reducida, que consolida las filas agrupadas en una única fila representativa.
- **Matriz Reducida:** La matriz reducida resultante de la agrupación es mucho más compacta que la matriz original, lo que reduce significativamente el uso de memoria y mejora el rendimiento en términos de transmisión de datos. Esta optimización es crucial en el contexto de bases de datos distribuidas, donde la minimización de los costos de transmisión es uno de los principales objetivos del proyecto.

Subtema 4: Visualización de Resultados con Graphviz

Para validar y visualizar los resultados del procesamiento de las matrices, el proyecto incluye la generación de gráficos mediante la herramienta Graphviz. Las funciones crear_grafo y generar_grafica se encargan de representar tanto las matrices originales como las matrices reducidas en formato gráfico.

- **Representación Gráfica de las Matrices:** Los gráficos generados muestran la estructura de las matrices, incluyendo sus dimensiones

(n y m) y los valores contenidos en cada celda. Esta representación gráfica es útil para visualizar patrones de acceso y validar la agrupación de filas en la matriz reducida.

- **Gráficos Comparativos:** Además de representar las matrices individuales, el proyecto genera gráficos comparativos que muestran la relación entre la matriz original y la matriz reducida. Estos gráficos permiten visualizar de manera clara el impacto de la agrupación de patrones y la reducción de datos, lo que facilita la comprensión de los resultados y su validación.

Conclusiones

Este proyecto demuestra que la optimización del alojamiento de datos en bases de datos distribuidas es posible mediante el uso de tipos de datos abstractos (TDA), listas circulares y visualización gráfica. La implementación en Python no solo permitió una solución modular y eficiente, sino que también facilitó el manejo de grandes volúmenes de datos, reduciendo significativamente los costos de transmisión en redes distribuidas.

Uno de los principales aportes de esta solución es la transformación de matrices de frecuencia de acceso en matrices binarias, seguida por la agrupación de patrones similares. Este enfoque no solo optimiza el almacenamiento de datos, sino que también mejora la eficiencia del sistema en términos de procesamiento y uso de recursos. Además, la generación de gráficos con Graphviz añade una capa de validación visual, asegurando que las optimizaciones realizadas se puedan observar y verificar de manera efectiva.

Es importante destacar que la correcta implementación de clases como Par, Mapa y ListaCircular fue clave para manejar la complejidad del problema y facilitar la manipulación de datos. Esta estructura modular puede ser aplicada en otros contextos donde se requiera el manejo eficiente de matrices u otros tipos de datos.

Una pregunta abierta que surge de este trabajo es: ¿Cómo podrían adaptarse o mejorarse estas técnicas para manejar matrices aún más grandes o con patrones de acceso más complejos? Además, sería interesante explorar cómo estas optimizaciones podrían aplicarse a otros problemas NP-Hard en el ámbito de las bases de datos y la transmisión de datos.

Finalmente, se recomienda a quienes deseen profundizar en esta temática, explorar técnicas avanzadas de optimización y algoritmos que puedan complementar o mejorar las estrategias aquí presentadas, así como analizar cómo la inteligencia artificial podría integrarse para prever patrones de acceso y optimizar aún más la distribución de datos.

Referencias bibliográficas

1. Díaz Daniel. (21 de julio de 2021). Guía para principiantes de la programación orientada a objetos (Poo) en Python. Guía para Principiantes de la Programación Orientada a Objetos (POO) en Python (kinsta.com)
2. EUROINNOVA. (6 de febrero del 2022). ¿Qué son y cuáles son los tipos de algoritmos? Conoce los tipos de algoritmos | Euroinnova
3. Microsoft. (9 de mayo de 2023). Procesar el archivo XML (visual basic). Procesamiento del archivo XML - Visual Basic | Microsoft Learn
4. UMA. (5 de agosto del 2021). Tipos abstractos de datos. tema 5.PDF (uma.es)

Apéndices

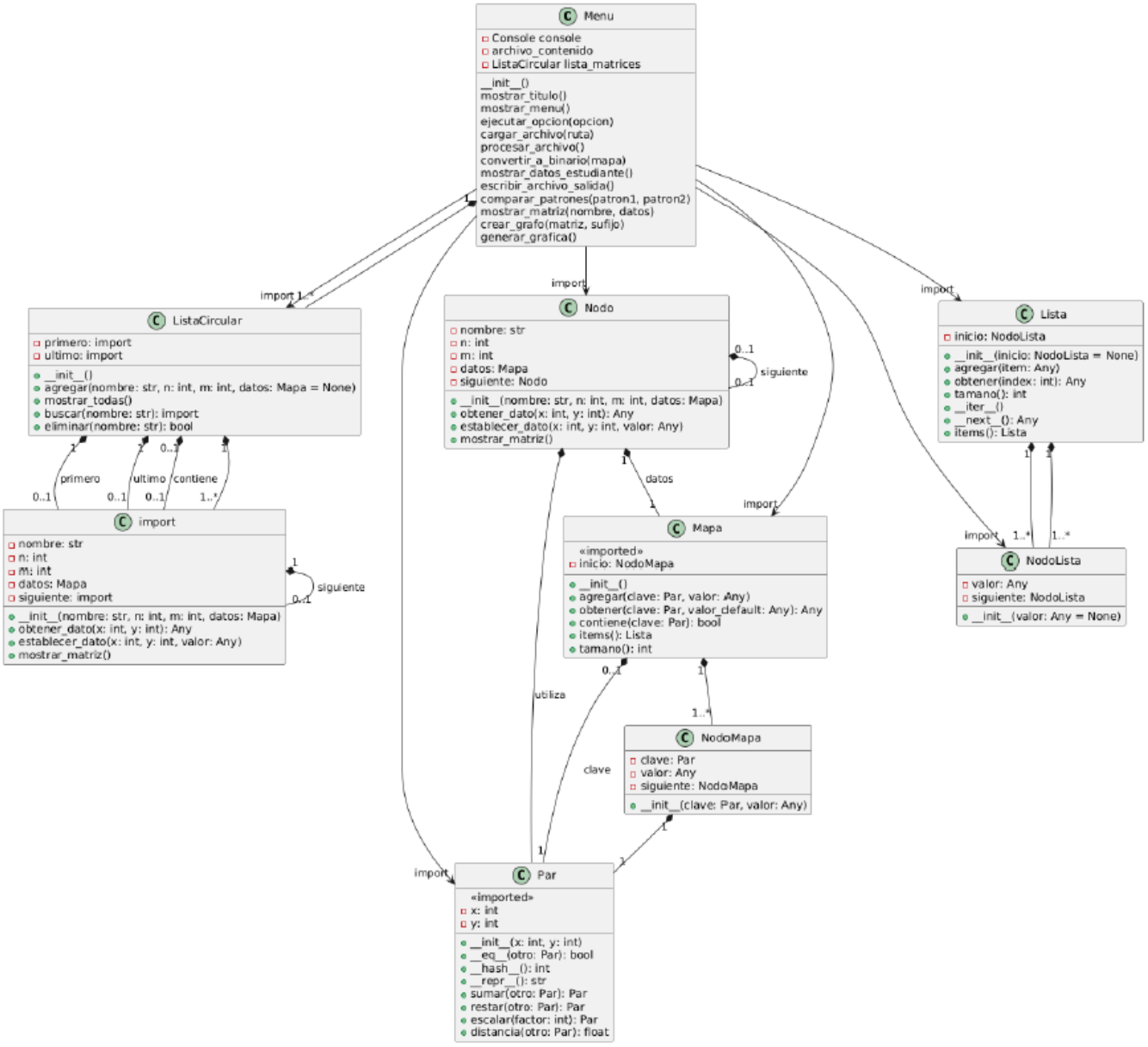

Figura 1. Diagrama de Clase
Fuente: elaboración propia

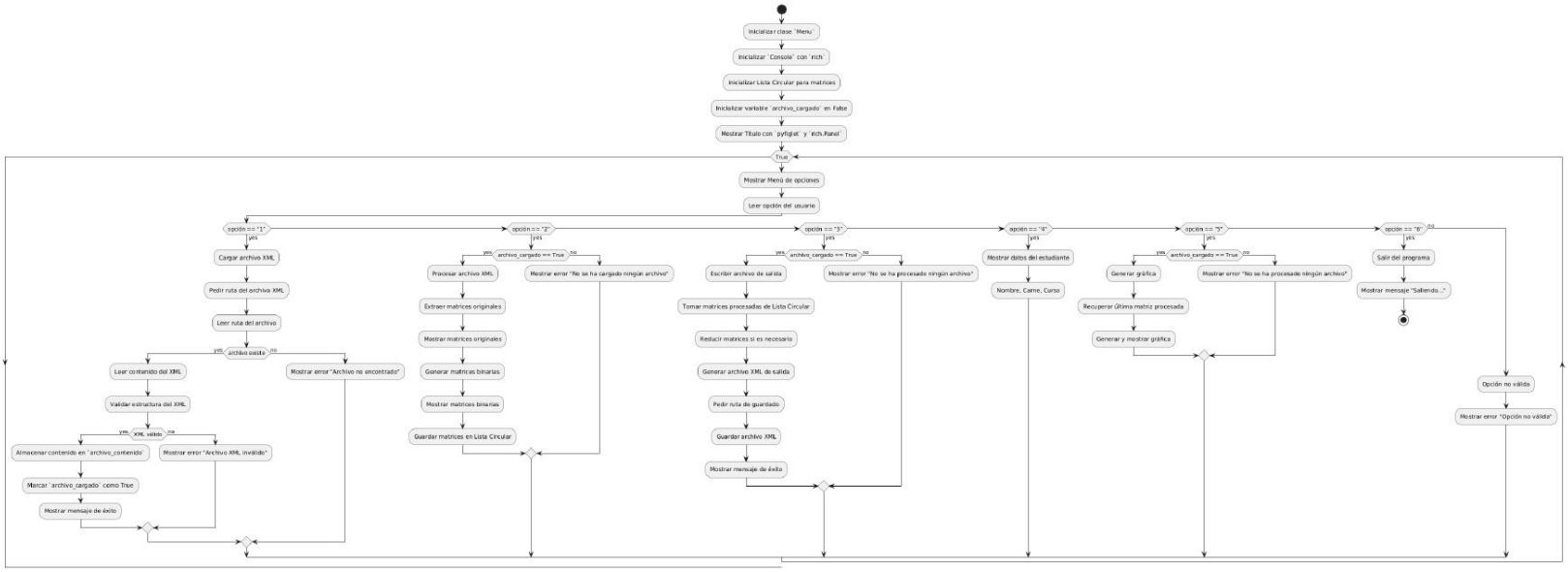

Figura 2. Diagrama de Actividades
Fuente: elaboración propia

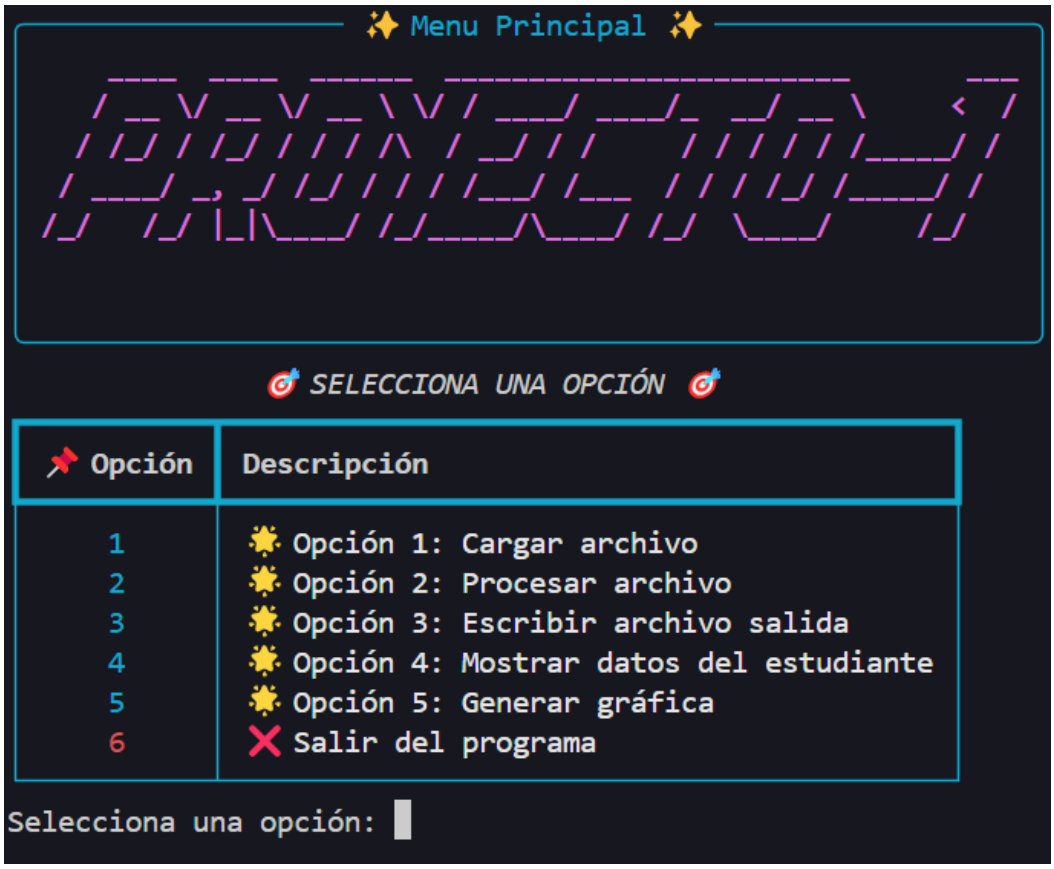

Figura 3. Menu del proyecto
Fuente: elaboración propia

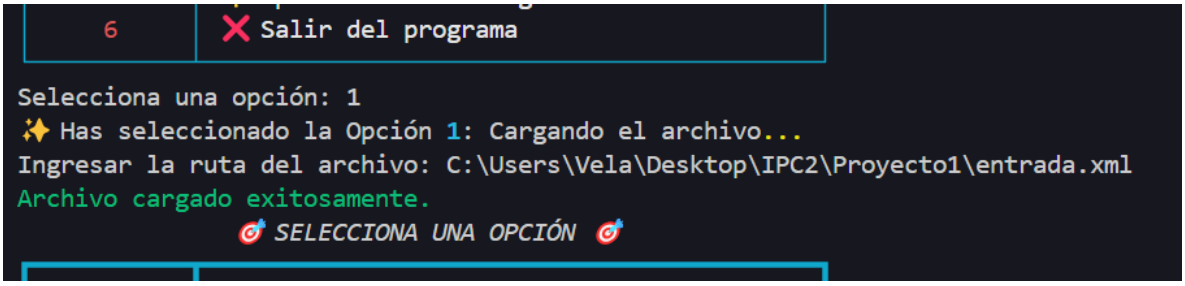

Figura 4. Funcionamiento opción 1
Fuente: elaboración propia

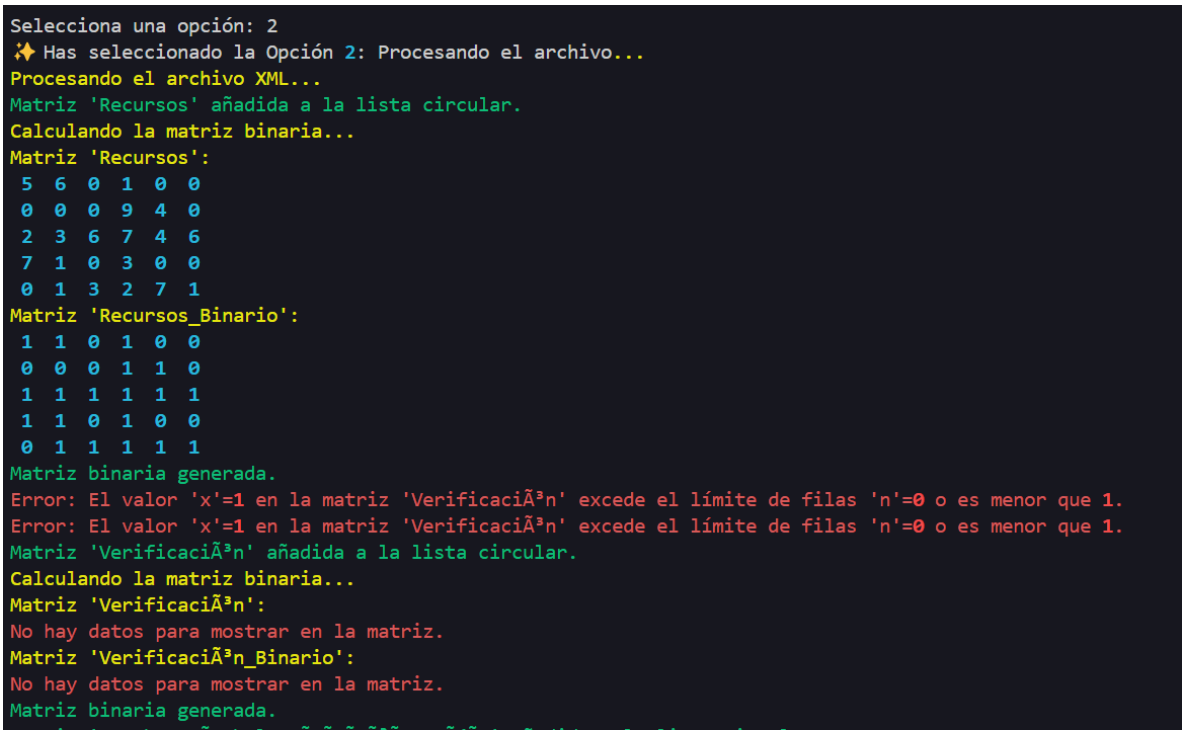

Figura 5. Funcionamiento opción 2
Fuente: elaboración propia

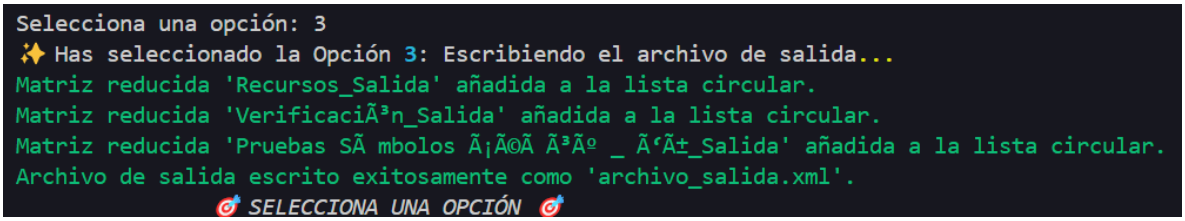

Figura 6. Funcionamiento opción 3
Fuente: elaboración propia

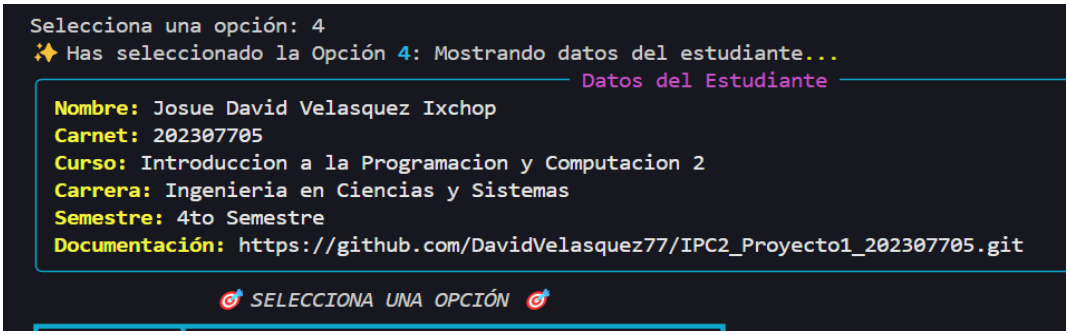

Figura 7. Funcionamiento opción 4
Fuente: elaboración propia

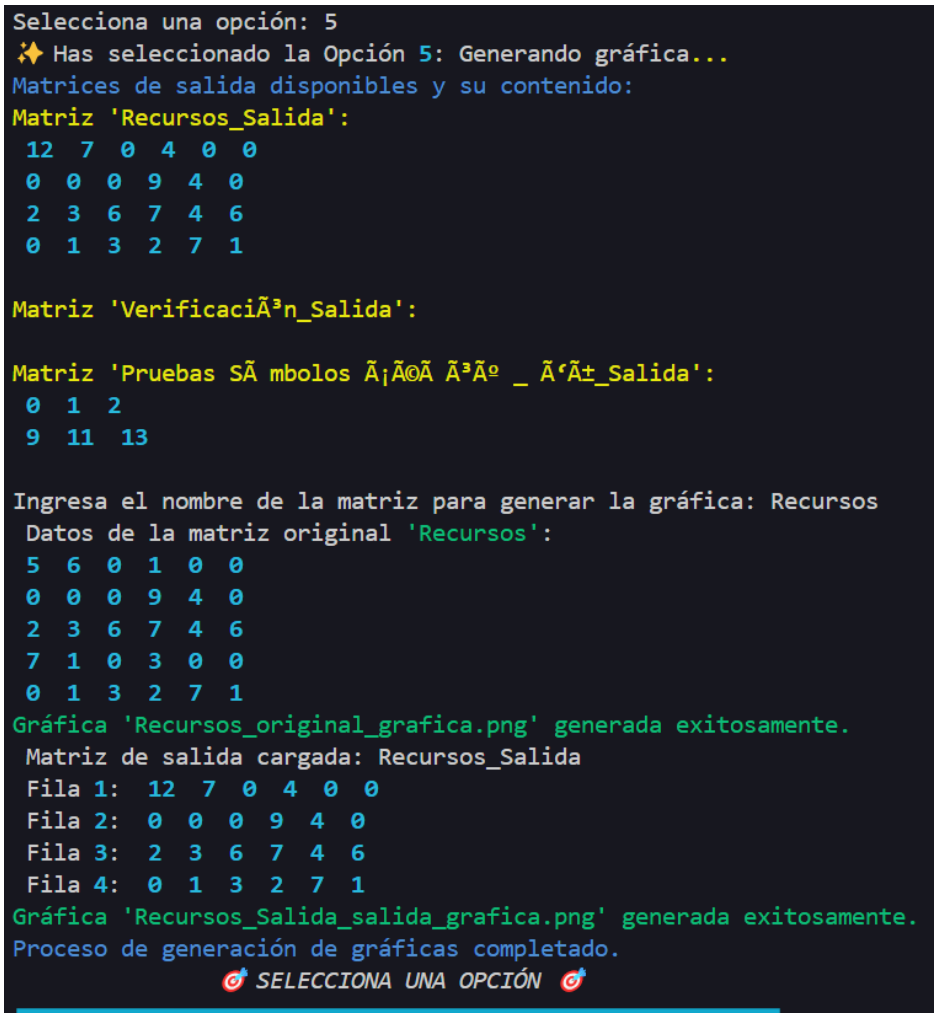

Figura 8. Funcionamiento opción 5
Fuente: elaboración propia

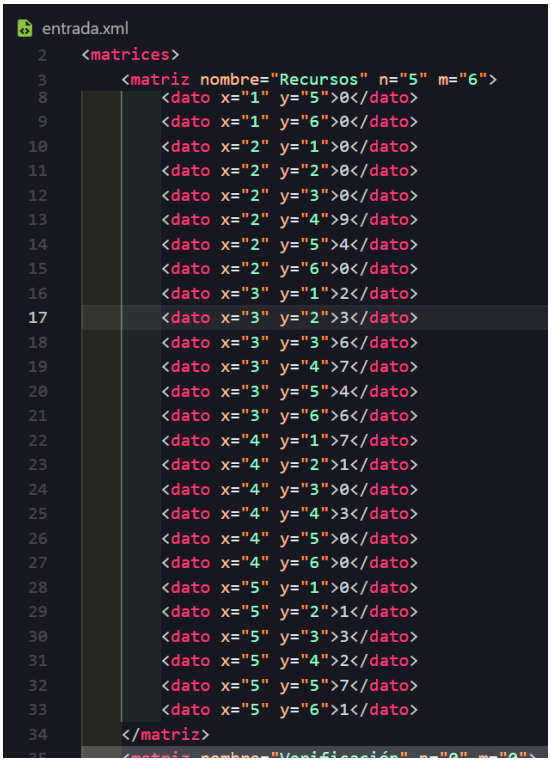

Figura 9. Archivo XML de entrada
Fuente: elaboración propia

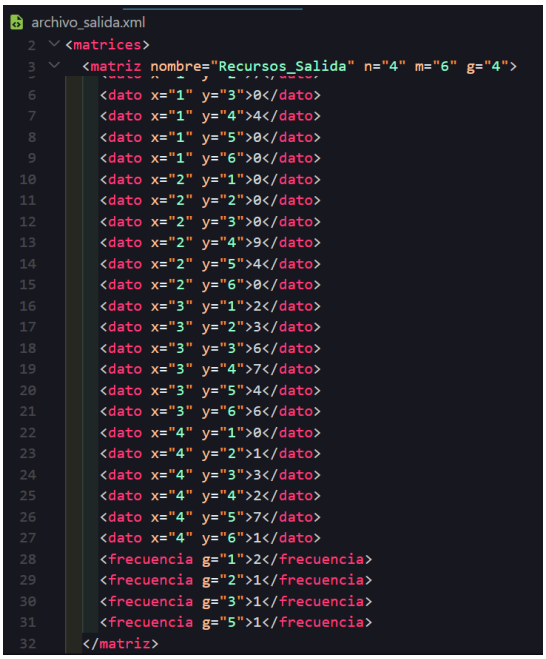

Figura 10. Archivo XML de salida
Fuente: elaboración propia

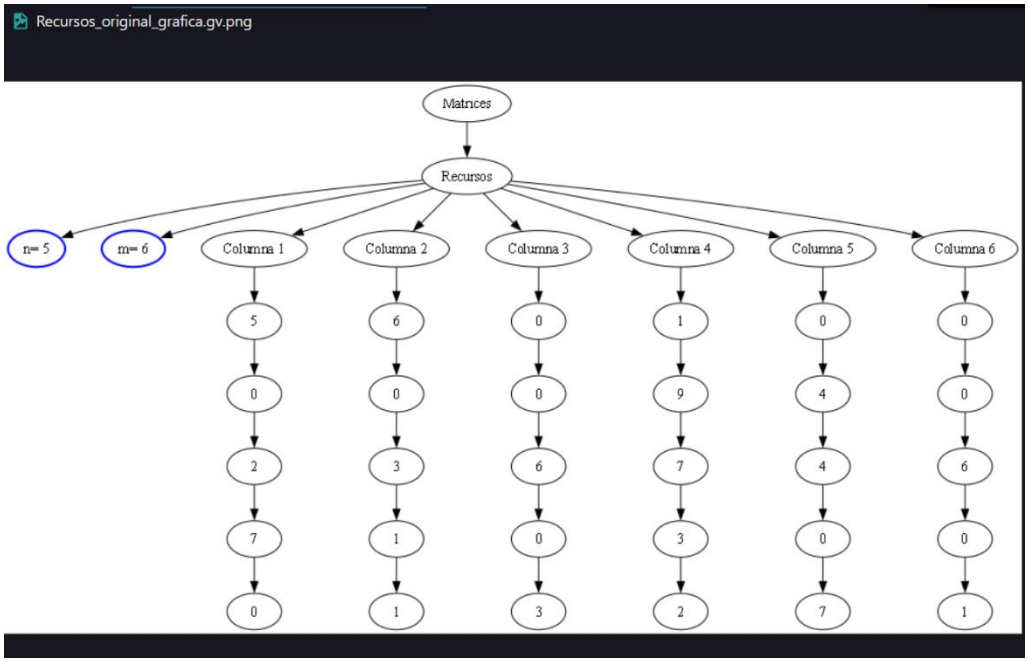

Figura 11. Grafica de matriz de entrada realiza en Graphviz
Fuente: elaboración propia

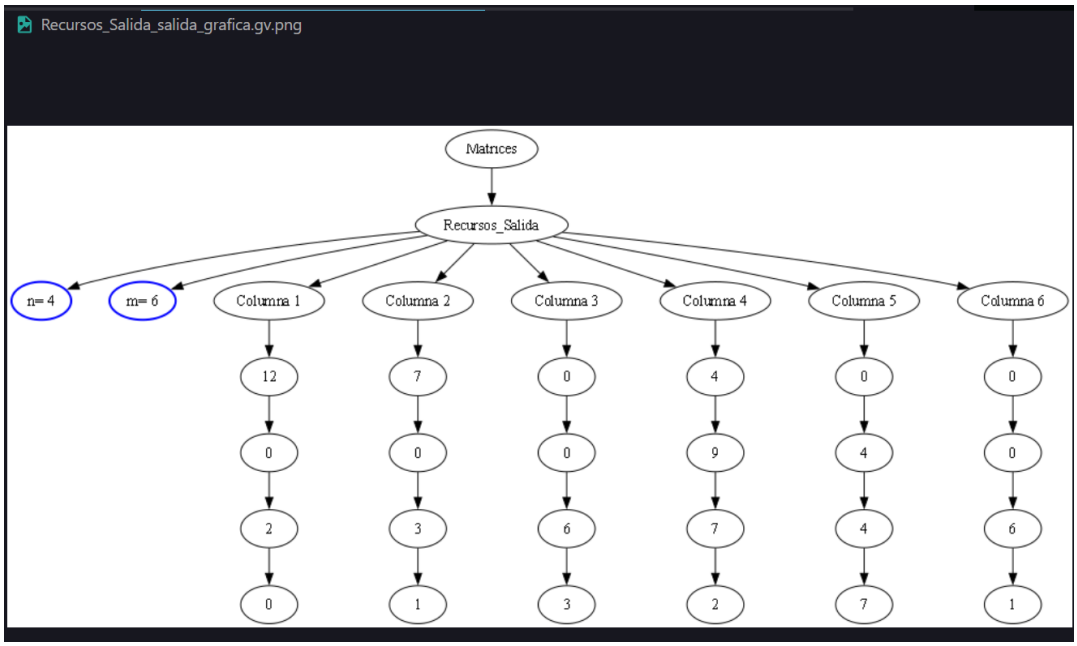

Figura 12. Grafica de matriz de salida realiza en Graphviz
Fuente: elaboración propia