Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2024

Proyecto 1: implementación de POO y TDAs con XML

201800708 – Marvin Enrique Ajquejay Matías

Resumen

Se propone una metodología de agrupamiento para abordar este desafío.
Para un conjunto de tuplas y sitios, se utiliza una matriz de frecuencia de acceso (F[nt][ns]). Esta matriz se transforma en una matriz de patrones de acceso, donde cada patrón es un vector binario que indica desde qué sitio se accede a cada tupla. Las tuplas con patrones de acceso idénticos se agrupan, y se calcula una matriz reducida de frecuencia de acceso sumando las tuplas dentro de cada grupo.
El objetivo es diseñar un programa que acepte múltiples matrices de frecuencia de acceso, identifique los grupos de tuplas con patrones de acceso similares y genere la matriz reducida de frecuencia de acceso para cada caso. Este enfoque busca adaptar el esquema de alojamiento a nuevos patrones de uso, optimizando así los costos de transmisión.

Palabras clave

1. **Distribución**
2. **Agrupamiento**
3. **Patrones de acceso**
4. **Optimización**
5. **Matriz reducida**

Abstract

A clustering methodology is proposed to address this challenge. For a set of tuples and sites, an access frequency matrix (F[nt][ns]) is used. This matrix is transformed into an access pattern matrix, where each pattern is a binary vector indicating from which site each tuple is accessed. Tuples with identical access patterns are grouped, and a reduced access frequency matrix is calculated by summing the tuples within each group.
The objective is to design a program that accepts multiple access frequency matrices, identifies groups of tuples with similar access patterns, and generates the reduced access frequency matrix for each case. This approach aims to adapt the data allocation scheme to new usage patterns, thereby optimizing transmission costs.

Keywords

1. ***Clustering***
2. ***Access patterns***
3. ***Optimization***
4. ***Reduced matrix***
5. ***Data allocation***.

Introducción

El diseño de distribución de bases de datos es un problema crítico en la gestión de sistemas distribuidos, donde el objetivo es minimizar los costos de acceso y comunicación. Este problema es conocido por ser NP-Hard, lo que implica que resolver instancias grandes requiere un alto consumo de tiempo y recursos de memoria. Para abordar este desafío, se propone una metodología de agrupamiento.

Esta metodología utiliza una matriz de frecuencia de acceso (F[nt][ns]) para un conjunto de tuplas y sitios. La matriz se transforma en una matriz de patrones de acceso, donde cada patrón es un vector binario que indica desde qué sitio se accede a cada tupla. Las tuplas con patrones de acceso idénticos se agrupan, y se calcula una matriz reducida de frecuencia de acceso sumando las tuplas dentro de cada grupo.

El objetivo de este enfoque es diseñar un programa que acepte múltiples matrices de frecuencia de acceso, identifique los grupos de tuplas con patrones de acceso similares y genere la matriz reducida de frecuencia de acceso para cada caso. Esto permitirá adaptar el esquema de alojamiento a nuevos patrones de uso, optimizando así los costos de transmisión.

Desarrollo del tema

Programación Orientada a Objetos (POO)
La Programación Orientada a Objetos (POO) es un paradigma de programación que organiza el código en unidades denominadas clases, de las cuales se crean objetos. Cada objeto es una instancia de una clase y contiene datos (atributos) y funciones (métodos) que operan sobre esos datos1. Los principios fundamentales de la POO son la encapsulación, la herencia y el polimorfismo. La encapsulación permite ocultar los detalles internos de un objeto y exponer solo lo necesario. La herencia facilita la creación de nuevas clases basadas en clases existentes, promoviendo la reutilización del código. El polimorfismo permite que diferentes objetos respondan a la misma operación de distintas maneras2.

En el contexto del problema de diseño de distribución de bases de datos, la POO facilita la creación de modelos de datos complejos y la implementación de algoritmos de agrupamiento y optimización. Por ejemplo, se pueden definir clases para representar tuplas, sitios y patrones de acceso, y utilizar métodos para agrupar tuplas con patrones de acceso similares y calcular la matriz reducida de frecuencia de acceso.

XML (Extensible Markup Language)
XML es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos en un formato legible tanto por humanos como por máquinas3. A diferencia de HTML, XML no tiene etiquetas predefinidas, lo que permite a los usuarios definir sus propias etiquetas según sus necesidades. Esto lo convierte en una herramienta poderosa para almacenar y compartir datos estructurados.

En el procesamiento de bases de datos distribuidas, XML se utiliza para estructurar y transportar datos entre diferentes sistemas. Por ejemplo, las matrices de frecuencia de acceso y los patrones de acceso pueden

ser representados en XML, facilitando su intercambio y procesamiento en diferentes etapas del algoritmo de agrupamiento. Python ofrece varias bibliotecas para trabajar con XML, como xml.etree.ElementTree, xml.dom, y xml.sax, que permiten la manipulación y análisis de documentos XML4.

Python

Python es un lenguaje de programación de alto nivel conocido por su simplicidad y legibilidad5. Es ampliamente utilizado en el desarrollo de aplicaciones web, análisis de datos, inteligencia artificial y más. En el contexto de la distribución de bases de datos, Python proporciona herramientas y bibliotecas que facilitan la implementación de algoritmos de agrupamiento y optimización. Además, su compatibilidad con XML permite una fácil integración y manipulación de datos estructurados.

Por ejemplo, se puede utilizar Python para leer matrices de frecuencia de acceso desde archivos XML, transformar estas matrices en patrones de acceso, agrupar las tuplas con patrones similares y calcular la matriz reducida de frecuencia de acceso. Bibliotecas como NumPy y Pandas son útiles para manejar y procesar grandes conjuntos de datos de manera eficiente.

Tipos Abstractos de Datos (TDAs)

Los Tipos Abstractos de Datos (TDAs) son modelos matemáticos para tipos de datos definidos por su comportamiento desde el punto de vista del usuario6. Un TDA especifica las operaciones que se pueden realizar sobre un conjunto de datos y el comportamiento de estas operaciones, sin preocuparse por su implementación. Esto permite una mayor abstracción y modularidad en el diseño de algoritmos y estructuras de datos.

En el contexto del problema de diseño de distribución de bases de datos, los TDAs pueden ser utilizados para definir estructuras de datos que representan matrices de frecuencia de acceso y patrones de acceso. Por ejemplo, se puede definir un TDA para una matriz de frecuencia de acceso que incluya operaciones para transformar la matriz en patrones de acceso, agrupar tuplas y calcular la matriz reducida de frecuencia de acceso. Esto facilita la implementación y el mantenimiento del algoritmo de agrupamiento y optimización.

Relación con el Tema

La combinación de POO, XML, Python y TDAs proporciona una base sólida para abordar el problema de diseño de distribución de bases de datos descrito en la Figura No. 1. La POO permite una estructura modular y reutilizable, XML facilita la transferencia y estructuración de datos, Python ofrece las herramientas necesarias para implementar y ejecutar los algoritmos, y los TDAs proporcionan una forma de modelar y manipular los datos de manera eficiente. Este enfoque integral permite adaptar el esquema de alojamiento a nuevos patrones de uso, optimizando así los costos de transmisión y mejorando el rendimiento del sistema.

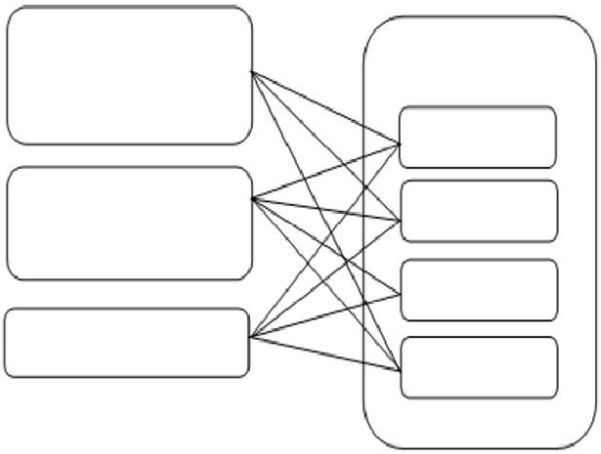

Figura 1. Listas enlazadas

Fuente: elaboración propia.

Documentación del Código

Descripción General

Este código tiene como objetivo manejar matrices cargadas desde un archivo XML y almacenarlas en una estructura de datos circular. Las matrices se pueden agregar, verificar su existencia, mostrar en

consola, y procesar posteriormente. La estructura principal utilizada para almacenar las matrices es una lista circular, y cada matriz es representada por un objeto de la clase MatrizAcceso.

Principales Clases y Funciones
MatrizAcceso
Esta clase representa una matriz y contiene la siguiente estructura:

Atributos:
nombre: Nombre de la matriz.
n: Número de filas.
m: Número de columnas.
Métodos:
agregar_dato(fila, columna, valor): Añade un valor específico a la posición indicada dentro de la matriz.
mostrar_matriz(): Muestra la matriz completa en consola. Si no se ha asignado ningún valor a una posición, muestra un 0.
ListaCircular
Esta clase maneja la estructura de lista circular que almacena los objetos MatrizAcceso.

Atributos:

puntero: Apunta al nodo actual en la lista circular.
Métodos:

agregar(matriz): Añade una nueva matriz a la lista circular.
verificar(nombre): Verifica si una matriz con un nombre específico ya existe en la lista.
mostrar_matrices(): Muestra en consola los nombres de todas las matrices almacenadas en la lista.
Nodo
Esta clase representa un nodo dentro de la lista circular.

Atributos:
matriz: Contiene el objeto de tipo MatrizAcceso.
siguiente: Apunta al siguiente nodo en la lista circular.
opcion1(ruta, lista_matrices)

Esta función carga matrices desde un archivo XML ubicado en la ruta especificada, y las agrega a la lista circular si no existen previamente.

Parámetros:

ruta: Ruta del archivo XML.
lista_matrices: Objeto de tipo ListaCircular donde se almacenarán las matrices.
Proceso:

Carga el archivo XML.
Itera sobre los elementos matriz en el archivo.
Para cada matriz, verifica si ya existe en la lista.
Si no existe, crea una nueva matriz y la agrega a la lista circular.
ProcesarArchivo(lista_matrices)
Esta función procesa y muestra las matrices almacenadas en la lista circular.

Parámetros:

lista_matrices: Objeto de tipo ListaCircular que contiene las matrices a procesar.
Proceso:

Itera sobre todas las matrices en la lista circular y muestra su contenido en consola.
Uso del Código
Cargar Matrices desde un Archivo XML:

Se utiliza la función opcion1(ruta, lista_matrices) para cargar matrices desde un archivo XML. Si la matriz ya existe en la lista, se omite y se muestra un mensaje de advertencia.
Verificación de Existencia de Matrices:

La función verificar(nombre) en la clase ListaCircular permite verificar si una matriz con un nombre específico ya ha sido cargada.
Mostrar Matrices en Consola:

Se puede mostrar el contenido de todas las matrices almacenadas utilizando mostrar_matriz() de MatrizAcceso o mostrar_matrices() de ListaCircular.
Procesamiento de Matrices:

La función ProcesarArchivo(lista_matrices) permite procesar todas las matrices cargadas, mostrando sus valores en la consola.
Dependencias
El código utiliza las siguientes bibliotecas:

xml.etree.ElementTree: Para manipular y recorrer el archivo XML.

Documentación de Graphviz en Python
Graphviz es una herramienta de visualización de gráficos que permite crear y representar diagramas de nodos y aristas de forma sencilla. En Python, la biblioteca graphviz proporciona una interfaz para generar y renderizar gráficos mediante el motor de visualización de Graphviz.
Instalación
Para utilizar Graphviz en Python, es necesario instalar la biblioteca graphviz mediante el gestor de paquetes pip. Además, se debe tener instalado el software de Graphviz en el sistema operativo.
Uso Principal

- **Creación de Grafos**: La biblioteca permite crear grafos dirigidos (con flechas) o no dirigidos (sin flechas), agregando nodos y aristas que representan las relaciones entre ellos.
- **Personalización de Gráficos**: Es posible personalizar los gráficos mediante atributos como forma, color, y etiquetas para los nodos y aristas.
- **Exportación de Gráficos**: Los gráficos generados pueden exportarse a diferentes formatos, como PNG, PDF, y SVG, lo que facilita su inclusión en documentos y presentaciones.

Funcionalidades Destacadas

1. **Definición de Nodos y Aristas**: Permite definir nodos (puntos en el grafo) y aristas (conexiones entre los nodos), con opciones para añadir etiquetas descriptivas.
2. **Atributos de Estilo**: Soporta la personalización visual mediante la definición de estilos, colores, y formas de los nodos y aristas.
3. **Visualización y Renderizado**: Genera archivos visuales en varios formatos, integrándose fácilmente con documentos o sistemas web.

Beneficios

- **Fácil Integración**: La biblioteca se integra fácilmente con scripts y aplicaciones Python, permitiendo la generación automática de diagramas.
- **Alta Personalización**: Proporciona numerosas opciones de personalización para la visualización detallada de estructuras de datos, diagramas de flujo, redes, y más.
- **Eficiencia y Compatibilidad**: Optimizada para trabajar con grandes conjuntos de datos y compatible con múltiples formatos de salida.

Aplicaciones Comunes

- **Visualización de Redes y Grafos**: Utilizado en el análisis de redes sociales, redes de computadoras, y sistemas biológicos.
- **Documentación y Diagramas de Flujo**: Ideal para representar flujos de trabajo, diagramas de estado, y estructuras de software.
- **Educación y Presentación**: Útil para crear representaciones visuales en contextos educativos y presentaciones profesionales.

Conclusiones

El diseño de distribución de bases de datos es un desafío complejo y crítico en la gestión de sistemas distribuidos, especialmente debido a su naturaleza NP-Hard que implica altos requerimientos de tiempo y recursos de memoria. La metodología de agrupamiento propuesta ofrece una solución viable para optimizar el alojamiento de datos, adaptándose a nuevos patrones de uso y minimizando los costos de transmisión.

La combinación de Programación Orientada a Objetos (POO), XML, Python y Tipos Abstractos de Datos (TDAs) proporciona una base sólida para implementar esta metodología. La POO facilita la creación de modelos de datos complejos y la implementación de algoritmos eficientes. XML permite la estructuración y transferencia de datos de manera flexible y estandarizada. Python, con sus potentes bibliotecas y herramientas, simplifica el procesamiento y análisis de grandes conjuntos de datos. Los TDAs, por su parte, ofrecen una forma estructurada y eficiente de modelar y manipular los datos necesarios para el algoritmo de agrupamiento.

En conjunto, estos enfoques permiten desarrollar un programa que no solo optimiza el esquema de alojamiento de datos, sino que también mejora el rendimiento general del sistema, adaptándose dinámicamente a los cambios en los patrones de uso y reduciendo significativamente los costos de transmisión.

Referencias bibliográficas

1. Guttag, J. V. (2013). *Introduction to Computation and Programming Using Python: With Application to Understanding Data*. MIT Press.
2. Liskov, B., & Guttag, J. (2000). *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*. Addison-Wesley.
3. McLaughlin, B., Hunter, J., & McLaughlin, B. (2006). *Java and XML*. O'Reilly Media.
4. Sebesta, R. W. (2018). *Concepts of Programming Languages* (12th ed.). Pearson.

ANEXOS

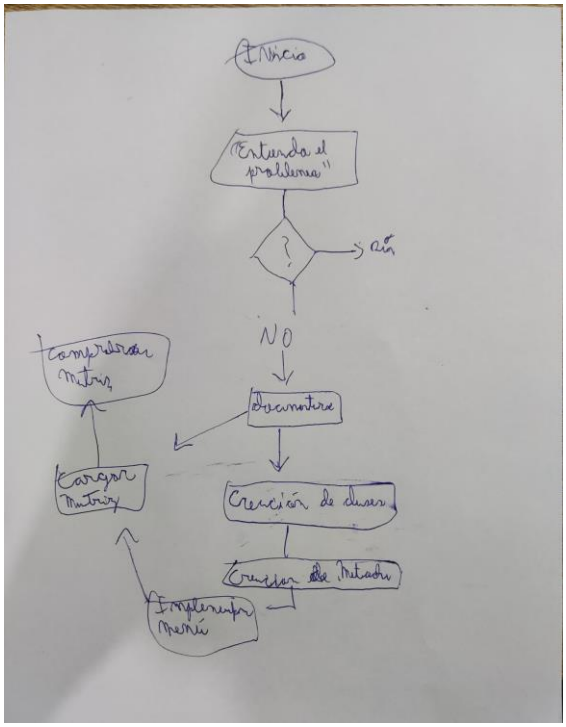

Figura 1- Diagrama de actividades

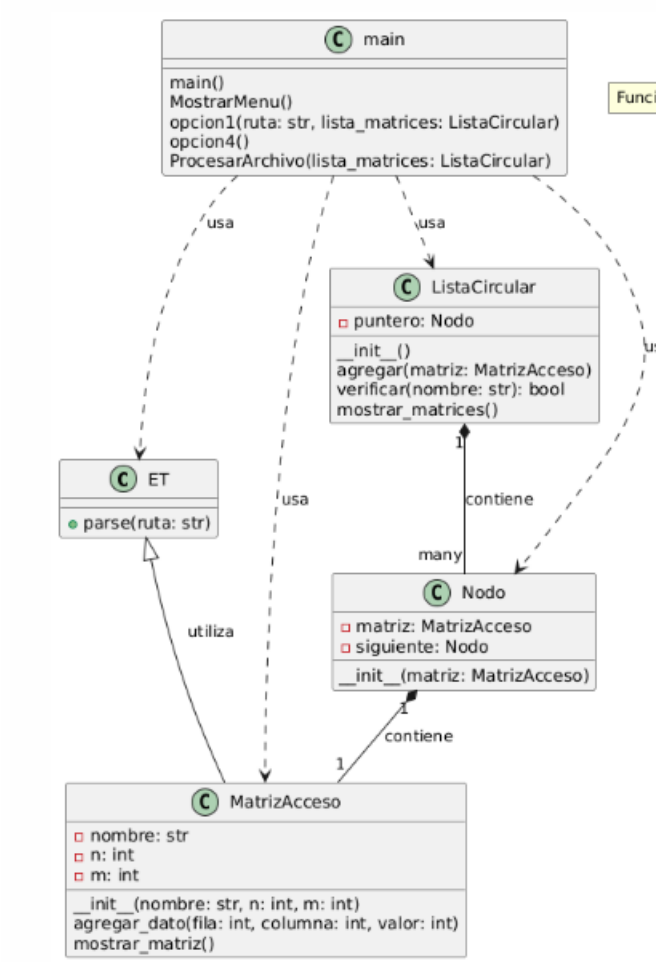

Figura 2 - Diagrama de clases