Proyecto 1 IPC2

201801155 – Bryan Eduardo Caal Racanac

Resumen

Utilizando el lenguaje de programación python como base para el desarrollo de este proyecto, implementado estructuras básicas de la programación como programación secuencial, cíclicas y condicionales, y la técnica de programación denominada como programación orientada a objetos (POO), se desarrolló una solución integral que implementa tipos de datos abstractos y visualización de datos, bajo el concepto de programación orientada a objetos, por lo que se procedió a fusionar estos conocimientos en un solo proyecto, se realizó un programa en consola capaz de utilizar archivos con extensión XML como insumos para el desarrollo del proyecto efectuando este la lectura de estos archivos con una estructura diferente a la de los archivos comunes y realizando distintas acciones con estos datos como ordenar y comparar datos, generar archivos de salida con diferentes extensiones así como también poder visualizar TDA´s por medio de la herramienta Graphviz

Palabras clave
POO, XML, Python, Graphviz, TDA

Abstract

Using the python programming language as the basis for the development of this project, implemented basic programming structures such as sequential, cyclic and conditional programming, and the programming technique called object-oriented programming (OOP), a comprehensive solution was developed that implements abstract data types and data visualization, under the concept of object-oriented programming , so this knowledge was merged into a single project, a console program was made capable of using files with XML extension as inputs for the development of the project by reading these files with a different structure than the common files and performing different actions with this data such as sorting and comparing data , generate output files with different extensions as well as be able to view ADT's using the Graphviz tool.

Keywords
POO, XML, Python, Graphviz, TDA

Introducción

Actualmente la demanda de información es mucha y con el pasar de los años los formatos con los cuales la información puede ser proporcionada es igual de variada, debido a que no todos los formatos sirven para los mismo, como puede ser un documento informativo puede ser solo texto pero si necesitamos información dividida y agrupada por separado, el formato de texto común y corriente se vuelve muy complicado y tedioso de leer, con esta información tomada en cuenta se creó un programa el cual puede leer archivos de texto con la extensión .XML la cual consta de etiquetas padre que contienen etiquetas hijo que a su vez contienen información, con esta información podemos hacer diferentes procedimientos como agrupar la información parecida, o en este caso mostrar la información de manera gráfica para que el usuario tenga una mayor facilidad para poder interpretar la información que se le está presentando que si solo lee el archivo con extensión XML.

Desarrollo del tema

Este proyecto consiste en alojar objetos de bases de datos en sitios distribuidos, de manera que el costo total de la transmisión de datos para el procesamiento de todas las aplicaciones sea minimizado. Un objeto de base de datos es una entidad de una base de datos, esta entidad puede ser un atributo, un set de tuplas, una relación o un archivo. Los objetos de base de datos son unidades independientes que deben ser alojadas en los sitios de una red. Una definición formal del problema se presenta en la figura No. 1.

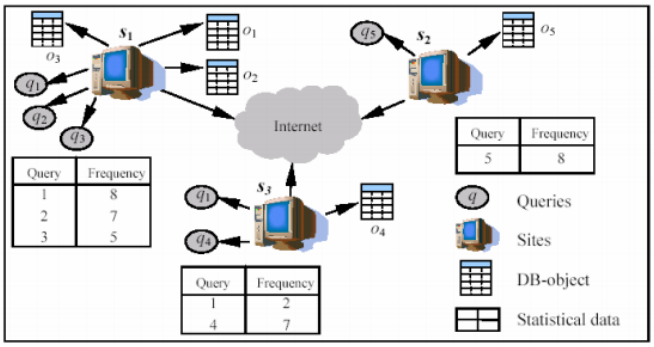

Figura No. 1: Problema de diseño de distribución en una base de datos

Fuente: Elaboración Propia 2021

La figura No. 1 muestra un set de objetos de bases de datos O = {o1, o2, …, o_no}, una red de computadoras que consiste en un set de sitios S = {s1, s2, …, s_ns}, donde un set de consultas Q = {q1, q2, … q_nq} son ejecutadas, los objetos de base de datos requeridos por cada consulta, un esquema inicial de alojamiento de objetos de bases de datos, y las frecuencias de acceso de cada consulta desde cada sitio en un período de tiempo. El problema consiste en obtener un nuevo esquema replicado de alojamiento que se adapte a un nuevo patrón de uso de la base de datos y minimice los costos de transmisión.

El problema de diseño de distribución consiste en determinar el alojamiento de datos de forma que los costos de acceso y comunicación son minimizados. Como muchos otros problemas reales, es un problema combinatorio NP-Hard. Algunas de las situaciones comunes que hemos observado cuando se resuelven instancias muy grandes de un problema NP-Hard son: Fuerte requerimiento de tiempo y fuerte demanda de recursos de memoria. Un método propuesto para resolver este tipo de problemas consiste en aplicar una metodología de agrupamiento.

Para “nt” tuplas y “ns” sitios, el método consiste en tener la matriz de frecuencia de acceso en los sitios F[nt][ns] de la instancia objetivo, transformarla en una matriz de patrones de acceso y agrupar las tuplas con el mismo patrón.

El patrón de acceso para una tupla es el vector binario indicando desde cuál sitio la tupla es accedida.
Por ejemplo:
Para la siguiente matriz de frecuencia de acceso:

2	3	0	4
0	0	6	3
3	4	0	2
1	0	1	5
0	0	3	1

Figura No. 2: Matriz de entrada

Fuente: Elaboración Propia 2021

Entonces, su correspondiente matriz de patrones de acceso es:

1	1	0	1
0	0	1	1
1	1	0	1
1	0	1	1
0	0	1	1

Figura No. 3: Matriz Binaria

Fuente: Elaboración Propia 2021

Se puede observar que las filas 1 y 3 tienen los mismos patrones de acceso, así como también las filas 2 y 5. Entonces, habrá tres grupos considerando el tercer grupo formado simplemente por la fila 4. La cardinalidad de un grupo es definida por el número de tuplas incluidas en él.
La matriz reducida de frecuencia de accesos obtenida de la suma de las tuplas en los grupos será:

5	7	0	6
0	0	9	4
1	0	1	5

Figura No. 4: Matriz de salida

Fuente: Elaboración Propia 2021

Se diseñó un programa que acepte “n” matrices de frecuencia de accesos y por cada una obtener los grupos formados por las tuplas con el mismo patrón de acceso. Finalmente, se debe obtener la matriz de frecuencia de acceso reducida.

Para realizar este programa el algoritmo que se utilizo fue siguiendo la siguiente lógica. Al ingresar el archivo las matrices numéricas se guardaban en listas las cuales a su vez se guardaban en nodos de una lista circular simple enlazada, la cual contenía toda la información para que esta fuera después utilizada, después de la lectura de todas las matrices se recorren todos los nodos y se obtienen las matrices, y se convierten a binarias, eso quiere decir que cualquier número que no sea 0 se convertirá en un 1 para todas

las filas en la matriz, luego se toma la primera fila como pivote y se compara con las siguientes filas y si encuentra una fila idéntica las suma y se crea una nueva matriz, luego se realiza el mismo procedimiento hasta que ya no queden filas iguales a la de la fila pivote, luego el pivote cambia a la fila que el sigue y el proceso se reanuda con un nuevo pivote, esto se repite hasta que el pivote sea un número menor a la longitud de la matriz, para tener una demostración más visual vea la siguiente figura.

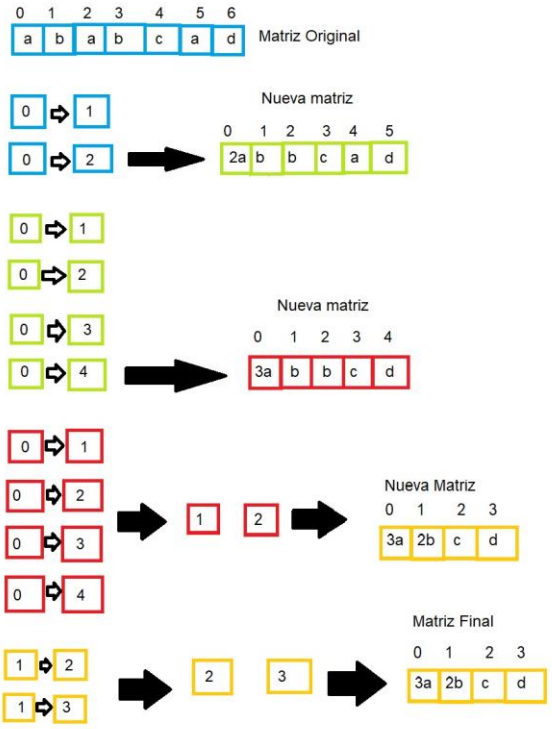

Figura No. 4: algoritmo de comparación y reducción de matrices

Fuente: Elaboración Propia 2021

Se utilizó la herramienta Graphviz para crear un grafo que muestra de manera gráfica la estructura del archivo de entrada procesado. Se mostró de una manera similar a la siguiente. En este ejemplo, "Ejemplo" será el nombre de la matriz, "n" y "m" serán las dimensiones y en la parte derecha se mostrarán los valores que contiene la matriz.

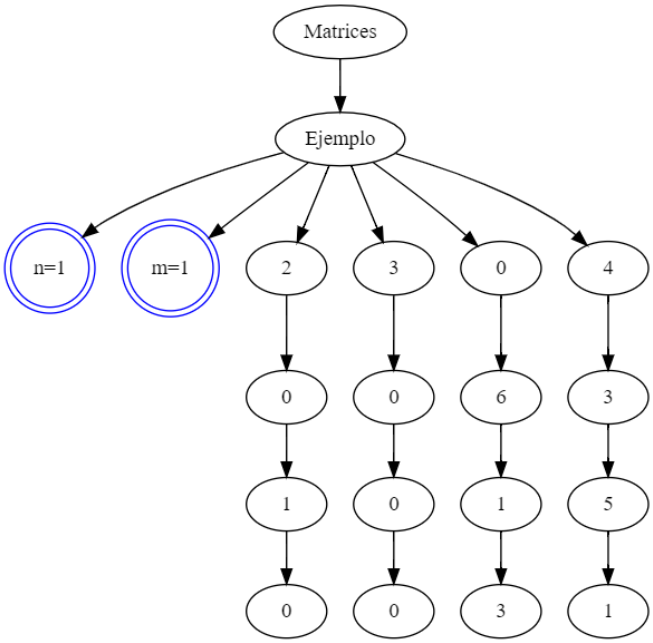

Figura No. 5: Grafica de matriz de entrada

Fuente: Elaboración Propia 2021

Los archivos de entrada y salida consistirán en archivos con extensión y estructura xml en el cual se limitará a utilizar únicamente las etiquetas:

Matrices: este será necesario para la lectura inicial del archivo, ya que será la etiqueta padre de todo.

matriz: esta etiqueta será la que indica que una nueva matriz será creada para su respectivo análisis y únicamente puede estar dentro de la etiqueta **matrices** y puede tener los atributos:

➢ **nombre**: este contendrá el identificador de la matriz leída (se deberá validar la existencia de matrices con el mismo nombre, para mantener la consistencia de los datos).

➢ **n**: será la fila que tendrá la matriz, si los datos dentro de ella son mayor o menor a este atributo será error.

➢ **m**: será la columna que tendrá la matriz, si los datos dentro de ella son mayor o menor a este atributo será error.

Dato: esta etiqueta únicamente podrá estar dentro de la etiqueta **matriz** y contendrán los valores respectivos a cada celda de la matriz, esta etiqueta puede tener los siguientes atributos.

➢ **x**: será la fila de la matriz y no puede ser mayor al atributo **n** de la matriz.

➢ **y**: será la columna de la matriz y no puede ser mayor al atributo **m** de la matriz

Ejemplo Matriz de entrada:

```
<matrices>
   <matriz nombre="Ejemplo" n=4 m=4>
         <dato x=1 y=1>2</dato>
         <dato x=1 y=2>3</dato>
         <dato x=1 y=3>0</dato>
         <dato x=1 y=4>4</dato>
         <dato x=2 y=1>0</dato>
         <dato x=2 y=2>0</dato>
         <dato x=2 y=3>6</dato>
         <dato x=2 y=4>3</dato>
         <dato x=3 y=1>3</dato>
         <dato x=3 y=2>4</dato>
         <dato x=3 y=3>0</dato>
         <dato x=3 y=4>2</dato>
         <dato x=4 y=1>1</dato>
         <dato x=4 y=2>0</dato>
         <dato x=4 y=3>1</dato>
         <dato x=4 y=4>5</dato>
         <dato x=5 y=1>0</dato>
         <dato x=5 y=2>0</dato>
         <dato x=5 y=3>3</dato>
         <dato x=5 y=4>1</dato>
   </matriz>
</matrices>
```

Ejemplo matriz de salida:

```
<matriz nombre="Ejemplo_Salida" n=3 m=4 g=3>
    <dato x=1 y=1>5</dato>
    <dato x=1 y=2>7</dato>
    <dato x=1 y=3>0</dato>
    <dato x=1 y=4>6</dato>
    <dato x=2 y=1>0</dato>
    <dato x=2 y=2>0</dato>
    <dato x=2 y=3>9</dato>
    <dato x=2 y=4>4</dato>
    <dato x=3 y=1>1</dato>
    <dato x=3 y=2>0</dato>
    <dato x=3 y=3>1</dato>
    <dato x=3 y=4>5</dato>
    <frecuencia g=1>2</frecuencia>
    <frecuencia g=2>2</frecuencia>
    <frecuencia g=4>1</frecuencia>
</matriz>
```

Como se mencionó anteriormente los datos fueron Guardados en una lista circular.

En ciencias de la computación, una lista enlazada es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

Una lista enlazada es un tipo de dato autorreferenciado porque contienen un puntero o enlace (en inglés link, del mismo significado) a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de

nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

Para que la aplicación se mas intuitiva con el usuario debido a que no todos están familiarizados con la programación se cuenta con un menú interactivo que se maneja por medio del teclado de la computadora, el cual tiene las opciones que se describieron anteriormente en el documento y con otras opciones como lo son visualizar los datos del estudiante como se puede ver en la siguiente imagen.

Conclusiones

Para poder ordenar grandes cantidades de datos en diferentes formatos se aconseja manejar lo que son las estructuras de datos mucho más avanzadas, implementación de TDA´s y tener en consideración todas las excepciones que se puedan ejecutar al momento de leer archivos con grandes cantidades de datos, como podemos ver la forma tradicional de guardar los datos en listas requiere de mucha memoria y no siempre es lo más óptimo debido a que cuando se tienen demasiados datos el recorrer toda la lista de principio a fin podría tardar mucho, el uso de listas enlazadas facilita un poco esta lectura de archivos y un almacenamiento mucho más ordenado, bajando así el rendimiento y los requerimiento que se le exigen a la maquina al momento de correr un programa.

La programación orientada a objetos es otra gran implementación en los programas de gran tamaño debido a que todo el código está dividido por tareas pequeñas e implementado en módulos, si todo el código se realizara en un solo script este podría tener muchos problemas al momento de este tener un error, por las grandes líneas de código se haría un tarea imposible el realizar un debug del mismo o localizar un error, por lo que si dividimos el código en pequeños pedazos se podrá observar con mayor facilidad donde está el error en el momento de la ejecución, y se podrá remplazar por otro modulo que realice la tarea de una forma más óptima.

Referencias bibliográficas

Antonakos, James L. and Mansfield, Kenneth C., Jr. *Practical Data Structures Using C/C++* (1999). Prentice-Hall. ISBN 0-13-280843-9, pp. 165–190

Collins, William J. *Data Structures and the Java Collections Framework* (2002,2005) New York, NY: McGraw Hill. ISBN 0-07-282379-8, pp. 239–303

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford *Introductions to Algorithms* (2003). MIT Press. ISBN 0-262-03293-7, pp. 205–213, 501–505