PROYECTO 1

201801627 – Eduardo René Agustin Mendoza

Resumen

Al tener guardados una cantidad enorme de datos en diferentes puntos puede ser un poco delicado ya que la información se tiene dispersa y esto hace que algunas aplicaciones tarden en verificar la veracidad de una persona, para esto se solicitó que al ingresar los datos se encuentre un patrón mucho más sencillo de poder almacenarlos esto ya que se van guardando en tuplas y estas tuplas tiene un código para poder ser insertadas, la solución dada fue una matriz binaria la cual se iba a reducir por medio de las filas que tuvieran el mismo código la cual genero una matriz reducida y mucho más simple de poder almacenar

Estos archivos se documentaron mediante el formato xml porque al tener una estructura en forma de árbol era mucho más sencillo mostrarlos y trabajar con ellos.

Palabras clave

Efectividad de almacenamiento, tuplas, Red de computadoras, minimizar costos, base de datos.

Abstract

Having stored a huge amount of data at different points can be a bit delicate since the information is scattered and this makes some applications take time to verify the veracity of a person, for this it was requested that when entering the data a Much simpler pattern to be able to store them this since they are stored in tuples and these tuples have a code to be inserted, the solution given was a binary matrix which was going to be reduced by means of the rows that had the same code. which I generate a reduced matrix and much simpler to be able to store

These files were documented using the xml format because having a tree-shaped structure made it much easier to display and work with them.

Keywords

Storage effectiveness, Tuples, Computer network, Minimize costs, Database.

Introducción

El desarrollo del programa está dentro del ámbito de programación de Python con la implementación de los TDA´s (Tipos de Datos Abstractos) esto para poder ir aprovechando el espacio en memoria que se va generando al ir guardando cada uno de los datos, no se utilizaron las listas predeterminadas de Python ya que su almacenamiento seria limitado y habría un desperdicio de memoria, a lo largo de esto se da una mejora al utilizar dichos TDA´s ya que se puede ingresar una infinidad de datos sin desperdiciar algún tipo de espacio ya que esto es vital en el proceso de los datos para no atrasarlo.

A lo largo del desarrollo de la aplicación se le añadió un menú el cual tiene las opciones para poder redirigirnos a las demás funciones de este, se comienza analizando el archivo xml ya que dicho documento trae una estructura de árbol la cual la hizo más factible para poder trabajarla, dentro de cada documento podía venir ya sea una matriz o n matrices esto para poder analizarlas esto hacía referencia de que podían venir más de una tupla para poder almacenarla con diferentes datos la cual se debe de comparar con una tupla idéntica solo que en binario esto se hizo para que la entrada se facilitara ya que al repetirse las filas en las tuplas se tardaba en analizar dichas matrices con esto se simplifica un poco más para que las filas repetidas al convertirlas en binaras se sumaran en su formato original así solo se ingresa una matriz reducida y facilita el trabajo y aumenta el procesamiento de cada una de las tuplas.

Desarrollo del tema

Este problema 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.

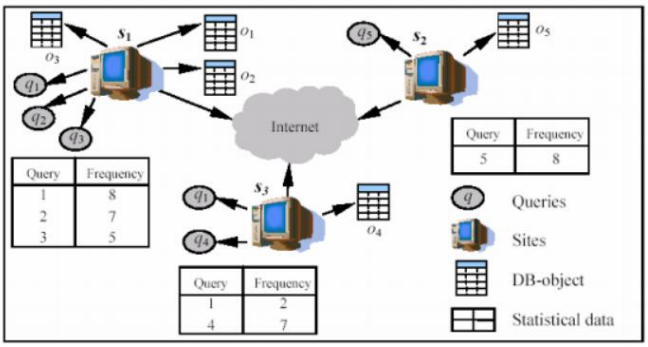

Figura 1. Problema de diseño de distribución en una base de datos.

La figura No. 1 muestra un set de objetos de bases de datos O = {o1, o2, …, ono}, una red de computadoras que consiste en un set de sitios S = {s1, s2, …, sns}, donde un set de consultas Q = {q1, q2, … qnq} 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.

En la figura Nº2 podemos ver como se encuentra nuestra matriz para poder tener acceso y poder guardar los datos

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

Figura 2. Matriz de frecuencia de acceso

Posterior a esta en la figura Nº3 se muestra su matriz correspondiente de patrones de acceso.

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

Figura 3. Matriz de patrones de acceso

Entonces se puede observar que tanto la fila 1 y 3 así como la fila 2 y 5 son iguales en su código de acceso por lo tanto tendremos 3 grupos considerados el primero serán las filas 1 y 3, el segundo la gila 2 y 5 y por último solamente la fila 4, esta se reducirá en forma en que estas filas en la matriz de frecuencia de accesos se sumaran dichas filas repetida y se dará como resultado una nueva y esta será la figura Nº4

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

Figura 4. Matriz Reducida.

Por lo tanto esta es nuestra nueva tupla que queda para ser ingresada en las demás bases de datos. Así fue como se ideo el programa para que pudiera aceptar n cantidades de matrices.

Las tuplas son una secuencia de valores en las cuales se pueden agrupar como si fuesen un único valor, varios valores que por su naturaleza deben de estar juntos, el tipo de dato que representa las tuplas son tuples los cuales son inmutables estas no pueden ser modificadas al ser ya creadas.

En este caso al programa se le asigna una matriz de entrada la cual se puede modificar y al ser está tratada se convierten en una tupla la cual sus valores ya no pueden ser modificados nuevamente y esto facilita la inserción de los datos.

Un TDA se define como una serie de n elementos en la cual se puede hacer cierta cantidad de operaciones, así como insertar, ver si esta vacía o no, eliminar

nodos, vaciar lista, mostrarla entre otras cada una de estas listas lleva consigo un nodo este contiene toda la información de lo que se ira guardando con un apuntador, dicho apuntador será único ya que dentro de la memoria solo existirá esa posición. Dentro de estos existen una cantidad especifica de listas en la cual entran los TDA cola (queue) y TDA pila (Stack) cada uno de estas estructuras tiene una simulación diferente por ejemplo una pila utiliza L.I.F.O = Last in, First out que sería último en entrar primero en salir esto lo podemos observar cuando tenemos apilados un par de libros que empezamos a extraer los libros de arriba hacía abajo y la cola F.I.F.O First in, First Out que sería primero en entrar y primero en salir esto podemos observarlo en una cola convencional de un banco que la primera persona que entra es la primera persona que sale, estos últimos tipos de datos son los que se utilizan más para almacenar un tipo de dato diferente ya que estos son para una larga duración entre otros.

La aplicación fue construida en el entorno de desarrollo de Python utilizando el IDE de Pycharm, la propuesta dada fue mejor el rendimiento del almacenamiento y minimizar los costos de trasmisión esto se dio para poder tener un esquema mejorado para que se adaptara al nuevo patrón que se almacenara.

La estructura de los datos se guardaran en archivos xml esto para poder tener un mejor despliegue de los datos ya que estos vendrán en una matriz los cuales se deben de trasformar en un código de entrada para poder ingresar a los servidores, el problema fue de que muchas de estos códigos se repetían por lo tanto la solución planteada fue que los códigos en este caso que eran binarios, si alguna fila de dicho documento se repetiría en su estructura, tengan el 100% de igualdad en el código binario, se irían sumando pero no el código si no al tener la matriz original comparar ambas líneas y verificar que sean las que se repiten entonces la suma de esa fila creara otro patrón de código el cual será una matriz reducida y este será mucho más fácil de poder ingresar a los servidores y ahorrara la lectura de filas repetidas, al momento de mostrar un dato de salida cada uno de los documentos que se dará tendrá la matriz reducida con la misma estructura xml.

Conclusiones

Sabemos que al trabajar con una extensa cantidad de datos es mucho mejor utilizar formatos de archivos que nos faciliten el trabajo y ahorren tiempo ya que el tiempo es muy valioso a la hora de verificar una gran cantidad de datos, con los códigos reducidos se agiliza este proceso.
A dicha solución del problema se implementaron las listas circulares y simples ya que en una circular iban a estar enlazadas todas las matrices que tienen los documentos y la simple fue nada más para almacenar los datos que traía cada una de las matrices para posteriormente analizarlas y comparar las filas si se repetían o no

Al trabajar con los TDA´s podemos ahorrarnos una infinidad de espacio ya que se utiliza lo necesario dentro del programa.
Las matrices que se plantean al final dan una mejor inserción a los servidores ya que se simplifica su código de acceso.

Referencias bibliográficas

C. J. Date, (1991). *An introduction to Database Systems.* Addison-Wesley Publishing Company, Inc.