Universidad San Carlos de Guatemala.
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería.
Introducción a la programación y computación 2. 1er. Semestre 2021.

DOCUMENTACIÓN

1. 201800937 – José Abraham Solórzano Herrera

Resumen

El programa consiste en analizar un matriz de frecuencia de acceso con su respectiva tupla y sitios de la instancia objeto, debe ser transformada en una matriz de patrones de acceso, se encarga de agregar las tuplas con el mismo patrón, la matriz de acceso para una tupla es el vector binario desde cuál sitio la tupla es accedida, por medio de la suma de las tuplas en los grupos pasa a ser una matriz reducida, el programa será capaz de aceptar "n" matrices de frecuencia de acceso y por cada una obtener los grupos formados por las tuplas con el mismo patrón de acceso; sin embargo, la solución al problema esta basada en tipos de datos abstractos (TDA) y por medio del software Graphviz se puede visualizar la matriz de frecuencia que es proporcionada a analizar. Se implementó una lista circular simplemente enlazada, con su respectiva clase Nodo y Lista Circular.

Palabras Claves

1. Matriz
2. TDA
3. Binario
4. Patrón

Abstract

The program consists of analyzing an access frequency matrix with its respective tuple and sites of the object instance, must be transformed into an array of access patterns, it is responsible for adding the tuples with the same pattern, the access matrix for a tuple is the binary vector from which place the tuple is accessed, through the sum of the tuples in the groups it becomes a reduced matrix, the program will be able to accept "n" access frequency matrices and for each one obtain the groups formed by the tuples with the same access pattern; However, the solution to the problem is based on abstract data types (ADT) and through the Graphviz software the frequency matrix that is provided to be analyzed can be visualized. A simply linked circular list was implemented, with its respective Node and Circular List class.

Keywords

1. Matrix
2. ADD
3. Binary
4. Pattern

Introducción

El 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. El diseño de distribución consiste en determinar el alojamiento de datos de forma que los costos de acceso y comunicación son minimizados, 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.

Desarrollo

Matriz

En matemática, una matriz es un arreglo bidimensional de números. Dado que puede definirse tanto la suma como el producto de matrices, en mayor generalidad se dice que son elementos de un anillo

Estructura de Datos

Las estructuras de datos nos permiten manipular cualquier tipo de dato, además de almacenar dinámicamente; se pueden representar de diversas formas, como enlazadas o contiguas.

Las estructuras contiguamente asignadas se componen por bloques de memoria única, incluye arrays, matrices, heaps, y hash tables; en cambio una estructura enlazada está compuesta por fragmentos de memoria unidos por punteros; Los contenedores son estructuras que permiten almacenar y recuperar datos en un orden determinado sin importar su contenido.

a. Estructura Enlazada

Las estructuras enlazadas son dadas por los punteros, son los encargados de apuntar a una dirección de memoria donde se encuentra ubicado un valor.

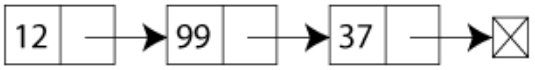


Figura I. Lista Enlazada
Fuente: Elaboración propia.

b. Lista Circular

Las lista circular es muy parecida a una estructura simple enlazada, la diferencia que tiene con la lista simple, es que no tiene fin; En las listas circulares, nunca se llega a una posición en la que ya no sea posible desplazarse. Cuando se llegue al último elemento, el desplazamiento volverá a comenzar desde el primer elemento.

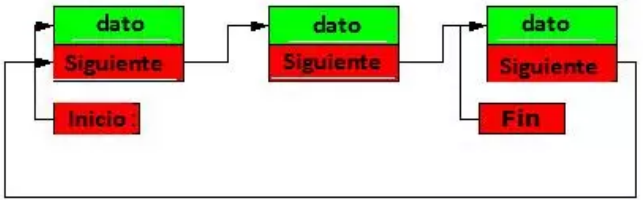


Figura II. Lista Circular
Fuente: Elaboración propia.

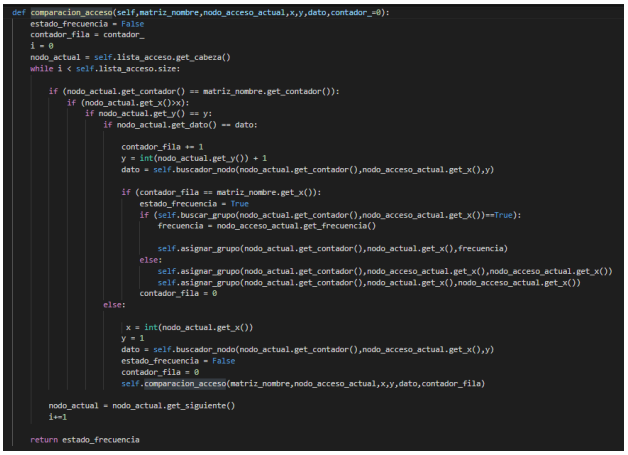
Recursividad

La recursividad es un concepto fundamental en matemáticas y en computación, es una alternativa diferente para implementar estructuras de repetición (ciclos), los módulos se hacen llamadas recursivas. Se puede usar en toda situación en la cual la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas. La recursividad consiste en que una función o un método sea llamado a sí mismo.

Caso recursivo: una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base.

Los pasos que sigue el caso recursivo son los siguientes:

1. El procedimiento se llama a sí mismo
2. El problema se resuelve, tratando el mismo problema pero de tamaño menor
3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.



```
def comparacion_acceso(self,matriz_nombre,nodo_acceso_actual,x,y,dato,contador_=0):
    estado_frecuencia = False
    contador_fila = contador_
    i = 0
    nodo_actual = self.lista_acceso.get_cabeza()
    while i < self.lista_acceso.size:

        if (nodo_actual.get_contador() == matriz_nombre.get_contador()):
            if (nodo_actual.get_x()>x):
                if nodo_actual.get_y() == y:
                    if nodo_actual.get_dato() == dato:

                        contador_fila += 1
                        y = int(nodo_actual.get_y()) + 1
                        dato = self.buscador_nodo(nodo_actual.get_contador(),nodo_acceso_actual.get_x(),y)

                        if (contador_fila == matriz_nombre.get_x()):
                            estado_frecuencia = True
                            if (self.buscar_grupo(nodo_actual.get_contador(),nodo_acceso_actual.get_x())==True):
                                frecuencia = nodo_acceso_actual.get_frecuencia()

                                self.asignar_grupo(nodo_actual.get_contador(),nodo_actual.get_x(),frecuencia)
                            else:
                                self.asignar_grupo(nodo_actual.get_contador(),nodo_acceso_actual.get_x(),nodo_acceso_actual.get_x())
                                self.asignar_grupo(nodo_actual.get_contador(),nodo_actual.get_x(),nodo_acceso_actual.get_x())
                            contador_fila = 0
                    else:

                        x = int(nodo_actual.get_x())
                        y = 1
                        dato = self.buscador_nodo(nodo_actual.get_contador(),nodo_acceso_actual.get_x(),y)
                        estado_frecuencia = False
                        contador_fila = 0
                        self.comparacion_acceso(matriz_nombre,nodo_acceso_actual,x,y,dato,contador_fila)

        nodo_actual = nodo_actual.get_siguiente()
        i+=1

    return estado_frecuencia
```

Figura III: Implementación de Recursividad.
Fuente: Elaboración propia.

En la Figura III se implementa recursividad, ayuda a que puede encontrar el patrón de frecuencia, empieza a validar fila por fila, si en dado caso toda la fila es verdadera se agrega la frecuencia, si en dado caso no llega a ser igual la fila se vuelve a recorrer el mismo método hasta encontrar el mismo patrón de la fila.

Lenguaje de Programación

En el programa se utilizó el lenguaje de programación python para poder desarrollar la solución al problema, Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad; además, Python es gratuito, incluso para propósitos empresariales, en los últimos años el lenguaje se ha hecho muy popular, gracias a varias razones como:

- La cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
- La sencillez y velocidad con la que se crean los programas. Un programa en Python puede tener de 3 a 5 líneas de código menos que su equivalente en Java o C.
- La cantidad de plataformas en las que podemos desarrollar, como Unix, Windows, OS/2, Mac, Amiga y otros.

Conclusiones

El lenguaje de programación python nos permite crear con sencillez y velocidad los programas, además de proporcionar un óptimo código a comparación de otros lenguajes como Java o C, entre otros.

La recursividad reside en el hecho de que se puede usar para resolver problemas con solución iterativa. aprovecha la ineficiencia de algunos algoritmos iterativos. La recursividad se debe usar cuando sea realmente necesaria, es decir, cuando no exista una solución iterativa simple.

Referencias bibliográficas:

https://github.com/bram814/-IPC2-Proyecto1_201800937.git

David Budgen(2015).Software Design..