IPC2-Proyecto 1

201901758 – Erick Ivan Mayorga Rodríguez

Resumen

El problema de diseño de distribución consiste en determinar el alojamiento de datos de una forma en la cual los costos de acceso y comunicación sean mínimos. Una solución propuesta en este caso consiste en aplicar una metodología de agrupamiento.

En el siguiente proyecto se aplicó este método, utilizando una matriz de frecuencia de accesos, la cual se transforma en una matriz de patrones de acceso y por último se utiliza para para comparar características similares dentro de la misma matriz y así poder dar como resultado una secuencia de números simplificada.

Aquellos datos de matriz de frecuencia de accesos reducida obtenidos a partir de las tuplas fueron guardados en un documento de tipo XML para la representación de los datos obtenidos, además para poder visualizar de manera gráfica el conjunto de datos de la matriz procesada se tiene un apartado el cuál crea un gráfico con la matriz reducida, a partir del uso de Graphviz, el cual proporciona un conjunto de herramientas para tener como resultado un archivo definido en lenguaje descriptivo dot, el cual se transformará a PNG

Palabras clave

- TDA: Tipo de dato abstracto, modelo que define valores que se pueden establecer en este mismo, para la realización de operaciones.
- Matriz: Arreglo bidimensional de números.
- Graphviz: Conjunto de herramientas de software para el diseño de diagramas en lenguaje descriptivo (DOT).
- DOT: lenguaje descriptivo en texto plano.
- Diseño: Actividad creativa con el fin de implementar objetos útiles y estetivos.

Abstract

A problem in layout design consists of how-to determinate the order or accommodation of the data in a way that reduces access and communication cost.

A solution for this case is to apply a grouping methodology.

In the presented project, is applied this methodology, Using an access frequency matrix, which change to an access pattern matrix, and finally is used to

compare similar characteristics in the data of the matrix.

Those data from the reduced access frequency matrix obtained from the analysis of the tuples, were saved in a XML document for the representation of the obtained data. Also a DOT type document was created with graphviz for the graphic representation.

Keywords

- *TDA: Abstract data type. Used to define values for itself.*
- *Matrix:two-dimensional arrangement.*
- *Graphviz: software used to desing graphs in Dot language*
- *DOT: graph description language.*
- *Desing: to make or draw plans for something, for example clothes or buildings*

Introducción

La solución consiste en una abstracción de la manera de alojar bases de datos en sitios distribuidos, de manera en la cual es costo total de la transmisión de datos para el procesamiento de todas las aplicaciones sea minimizado.

El problema de diseño de distribución consiste en determinar el alojamiento de datos de forma que el costo sea minimizado, para esta solución en especifico se utilizó la metodología de agrupamiento.

Para n tuplas y ns sitios, este método consiste en tener un matriz de frecuencia de acceso de los sitios de la instancia objetivo, además se transforma una matriz de patrones de acceso y se agrupan las tuplas con el mismo patrón.

Desarrollo del tema

A lo que se refiere la descripción anterior es que se tomó a consideración un patrón, en este caso ceros y unos, simulando valores booleanos para los datos que contiene la matriz, a su vez gracias a esto se realizó una comparación de dichos unos y ceros como referencia a sus valores verdaderos, teniendo la referencia de que valores establecidos en ceros y unos para cada fila son iguales, se sumaron los valores reales de las filas en la matriz sin mayor problema.

El resultado de estos valores de filas iguales sumadas, se volverá a colocar en una matriz, y dicha matriz será

traducida a lenguaje XML para ser almacenada como un archivo de matriz reducida.

Tomando como base la información de la matriz reducida, además del documento XML, se realizó un documento tipo dot con el software graphviz para tener una representación del resultado de la matriz de frecuencia reducida.

Cabe resaltar que el manejo de memoria de estos datos se realizó mediante objetos de tipo matriz, utilizando TDA, para abstraer que el tipo de dato matriz y el tipo de dato elemento que lleva la matriz con sus coordenadas “x” y “y” respectivamente.

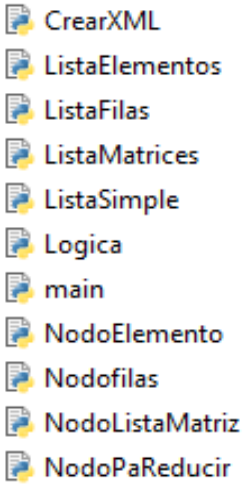
En la siguiente explicación de la metodología traducida al código de tomará a consideración segmentos de código que muestren el funcionamiento de la aplicación en relación a:

a. Clases
b. Funciones
c. Almacenamiento

El resultado final de la solución consiste en 12 clases con distintos elementos, en su mayoría clases nodo y clases lista.

Dichas clases están conectadas entre si y se utilizan algunas de manera recursiva, en especial las tipo lista (llegando a tener listas de listas de un mismo tipo) para poder dar solución al problema relacionado con la manera de guardar los datos en objetos de manera dinámica guardando referencias a los atributos específicos de la matriz en cuestión.

En el siguiente gráfico se pueden observar las clases utilizadas.



A continuación, se explicarán los objetos considerados más importantes en la elaboración de la solución

Main: Este segmento del código plantea el lugar en el cual se va a interpretar la ruta y se realizará la apertura del archivo, además de una forma de abrir el mismo de manera simplificada. Agregando, contiene otras funciones básicas como los primeros mensajes mostrados en consola al momento de ejecutar, además de los métodos que llevan al acceso de las distintas funciones planteadas en el menú principal.

```
def main_menu():
    opcion = -1
    while opcion != 4:
        opciones()
        opcion = int(input())
        if opcion == 1:
            cargar_archivo()
        elif opcion == 2:
            procesar_archivo()
        elif opcion == 3:
            escribir_salida()
        elif opcion == 4:
            mostrar_datos_estudiantes()
        elif opcion == 5:
            graficar()
        elif opcion == 6:
            salir()
        else:
            print("Algo")
```

ListaMatrices: este objeto como tal usa referencias de tipo NodoListaMatriz para realizar las distintas operaciones, principalmente la operación insertar al final, realizada por el método del mismo nombre, además tiene instancias de los elementos que irán al inicio y el final de la lista, dando como resultado que sea un objeto con características de lista circular simple con recorrido en sentido de las manecillas del reloj.

```
class listaCircularM:
    def __init__(self):
        self.primero = None
        self.ultimo = None

    def insertaralfinal(self, Matriz):
        if self.primero == None:
            self.primero = self.ultimo = Matriz
            self.ultimo.siguiente = self.primero
        else:
            temp = self.ultimo
            self.ultimo = temp.siguiente = Matriz
            self.ultimo.siguiente = self.primero
```

NodoListaMatriz: las varivales que declara este nodo son el nombre de la matriz, la cantidad n de columnas, m de filas, además de un objeto de tipo lista que cumple con la función de almacenar todos los datos que pueda llegar a tener este nodo que trata de tener las características principales de los atributos de una matriz.

```
from ListaElementos import ListaElementos

class Matriz:
    ListaElementos = ListaElementos()
    def __init__(self, nombre, filas, columnas, ListaElementos):
        self.filas = filas
        self.columnas = columnas
        self.nombre = nombre
        self.ListaElementos = ListaElementos
        self.siguiente = None
```

ListaElementos: este objeto tiene como función representar una lista doblemente enlazada de los objetos tipo dato, los cuales tendrán como características Nodos de tipo elemento, los cuales con métodos como insertar, serán introducidos con sus debidas referencias para nodo anterior y nodo siguiente.

```
class ListaElementos:
    def __init__(self):
        self.primero = None


    def insertar(self, Elemento):

        unNodo = Elemento
        if self.primero == None:
            self.primero= unNodo
        else:
            temp = self.primero
            while temp.siguiente != None:
                temp = temp.siguiente
            temp.siguiente = unNodo
            unNodo.anterior = temp
```

NodoElemento: este objeto represeta el cada dato que puede existir en la matriz, cuenta con variables como dato, posición en x, posición en y, es un datos de frecuencia(Utilizado para reciclarse en caso de que sea ya una matriz reducida), además de la referencia a los respectivos nodos siguiente y anterior.

```
class Elemento:
    def __init__(self, dato, enx, eny, ):
        self.dato = dato
        self.enx = enx
        self.eny = eny
        self.frecuencia = None
        self.estado = None
        self.siguiente = None
        self.anterior = None
```

Conclusiones

- El uso de la metodología de agrupación es altamente recomendable a la hora de optimizar la manera en la cual datos con características similares se almacenarán en memoria.
- La librería Xml Tree utilizada en Python, es una alternativa bastante simple para poder interpretar de manera correcta archivos escritos en leguaje de etiquetas XML.
- El uso de un Tipo de Dato Abstracto permite establecer operaciones con objetos de características que pudieron ser agrupadas en la instancia del mismo, como recorridos, impresiones y muchas otras más.