PROYECTO 2

201900157 – Kevin Alexis López Contreras

Resumen

La solución del problema presentado se solucionó implementando tipos de datos abstractos y programación orientada a objetos (POO), ya que al utilizar un tipo de dato abstracto se recurrió a usar una lista ortogonal para tener los datos organizados de una forma adecuada y de forma concisa, esta lista ortogonal consiste de dos listas doblemente enlazada simple que contendrán los nodos de la matriz.

La implementación de Graphiz es utilizada para poder visualizar los datos abstractos integrados en la lista ortogonal realizada previamente, esta imagen generada se mostrara en una interface en la ventana principal del programa se crearon varias ventanas para que el programa sea más intuitivo y más fácil de usar. Al utilizar una lista ortogonal se obtiene un mejor manejo de la memoria dinámica y con esto rinde de una forma más rápida el programa, esta matriz ortogonal se ordena de forma que los nodos menores se ingresen primero al obtener esto se hace más fácil el manejo de dicha matriz ortogonal.

Palabras clave

Matriz ortogonal: Una matriz ortogonal es la especialización real de una matriz unitaria y, por tanto, siempre una matriz normal . Aunque aquí solo consideramos matrices reales, la definición se puede utilizar para matrices con entradas de cualquier campo.

TDA's: Un tipo de datos abstracto se define por su comportamiento (semántica) desde el punto de vista de un usuario , de los datos, específicamente en términos de posibles valores, posibles operaciones sobre datos de este tipo y el comportamiento de estas operaciones

Matrices: Una matriz es una matriz rectangular de números (u otros objetos matemáticos) para los

que se definen operaciones como la suma y la multiplicación .

Orientado a objetos: es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos se utilizan como metáfora para emular las entidades reales del negocio a modelar.

Muchos de los objetos prediseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.

Está basada en varias técnicas del sexenio: herencia, cohesión, abstracción, polimorfis mo, acoplamiento y encapsulamiento.

Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos.

Abstract

The solution of the presented problem was solved by implementing abstract data types and object-oriented programming (OOP), since when using an abstract data type, an orthogonal list was used to have the data organized in a suitable and concise way. , this orthogonal list consists of two simple doubly linked lists that will contain the nodes of the matrix.

The implementation of Graphiz is used to be able to visualize the abstract data integrated in the orthogonal list previously made, this generated image will be shown in an interface in the main window of the program, several windows were created to make the program more intuitive and easier to use. . When using an orthogonal list, a better handling of dynamic memory is obtained and with this the program renders faster, this orthogonal matrix is ordered so that the minor nodes are entered first. When obtaining this, the handling of said orthogonal matrix.

Keywords

Orthogonal matrix: An orthogonal matrix is the real specialization of a unitary matrix and, therefore, always a normal matrix. Although we only consider real arrays here, the definition can be used for arrays with inputs from any field.

ADTs: An abstract data type is defined by its behavior (semantics) from the point of view of a user, of the data, specifically in terms of possible values, possible operations on data of this type and the behavior of these operations

Matrices: A matrix is a rectangular matrix of numbers (or other mathematical objects) for which operations such as addition and multiplication are defined.

Object-oriented: it is a programming paradigm that innovates the way to obtain results. The objects are used as a metaphor to emulate the real entities of the business to be modeled.
Many of the predesigned objects in current programming languages allow grouping into libraries or libraries, however, many of these languages allow the user to create their own libraries.
It is based on various techniques of the six-year term: inheritance, cohesion, abstraction, polymorphism, coupling and encapsulation.
Its use became popular in the early 1990s. Today, there is a wide variety of programming languages that support object orientation.

Introducción

El objetivo o la utilidad de la idea de usar TDA's es la de conseguir una mayor flexibilidad. Y esto lo logramos mediante el concepto más general de abstracción. Como vimos en el ejemplo, el código que utiliza el TDA no conoce ni depende de la implementación de las operaciones.
La abstracción es probablemente el concepto más general y más importante de la programación (o cualquier otra resolución de problemas).

En parte tiene que ver con la famosa idea del divide y conquista. Porque *separamos* los problemas, hacemos que diferentes partes de la aplicación se

La idea de modularización se puede pensar como la de ocultamiento y encapsulamiento, pero a aplicadas a un nivel más amplio o general de la aplicación. No solo a una unidad, como vimos a un Tipo u objeto, si no a un conjunto de éstos.

Al agruparlos y dividirlos definimos módulos.

Desarrollo del tema

Las estructuras de datos se utilizaron en esta solución porque era la más factible de implementar ya que los datos que se utilizaran son adecuados para ingresarlos a una lista ortogonal ¿y por qué este tipo de estructura?

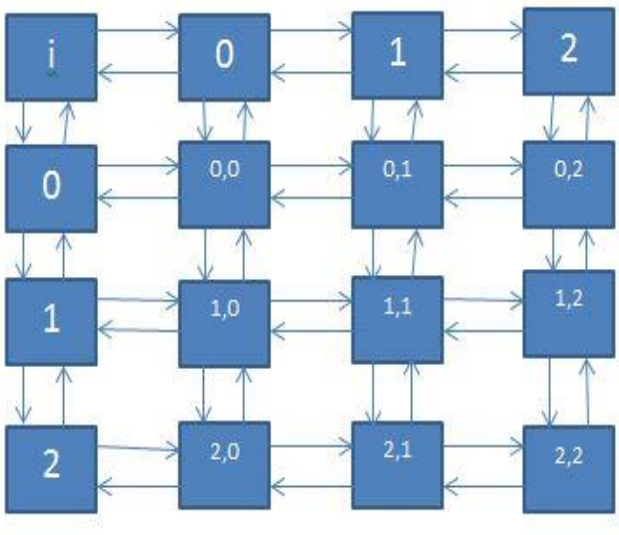
.

Pues bien la lista ortogonal se adecua a esta solución porque el programa maneja matrices ya entonces estos datos pueden retornar de una forma más sencilla y también se maneja de una forma mucho más correcta los espacios de memoria que no utiliza el programa, en algunos programas tienen ese peculiar problema que el programa es muy lento y esto puede ocurrir por muchos factores peo el principal factor de que ocurra esto es los espacios de

memoria que se quedan sin utilizar y esto el equipo de cómputo lo compila o lo interpreta y esto genera más demora para que se ejecute el programa por esta razón se utilizó una matriz ortogonal.

Para agregar cada nodo a una lista se requieren de funciones cíclicas esto ayuda a agregar los datos de una forma adecuada sin fugas de memoria dinámica.

*Figura 1.*Lista ortogonal.



Fuente :tutorialesprogramacionya.com

Conclusiones

1. La implementación del paradigma orientado a objetos fue factible ya que con programación orientado a objetos el programa se optimizo el tiempo de carga

2. La utilización de funciones cíclicas se implementó correctamente ya que no hubo ninguna fuga de memoria esto quiere decir que los nodos se insertaron de una forma adecuada.

3. Se utilizó listas ordenas y listas ortogonales de forma original reduciendo el tiempo el espacio de la memoria dinámica a utilizar

4. Se visualizaron los TDA´S con la herramienta Graphviz insertando cada nodo en la imagen.

5. Se Utilizaron archivos XML como insumos para la lógica y comportamiento de la solución.

6. Se utilizaron archivos html para mostrar los reportes de cada operación que se a realizado dándole estilos para una mejor presentación

Referencias bibliográficas

Hernández, R., Lázaro, J. C., Dormido, R., & Ros, S. (2001). Estructuras de datos y Algoritmos. Prentice Hall.

Joyanes Aguilar, L. (1998). Fundamentos de Programación, Algoritmos y Estructuras de Datos. McGraw-Hill.

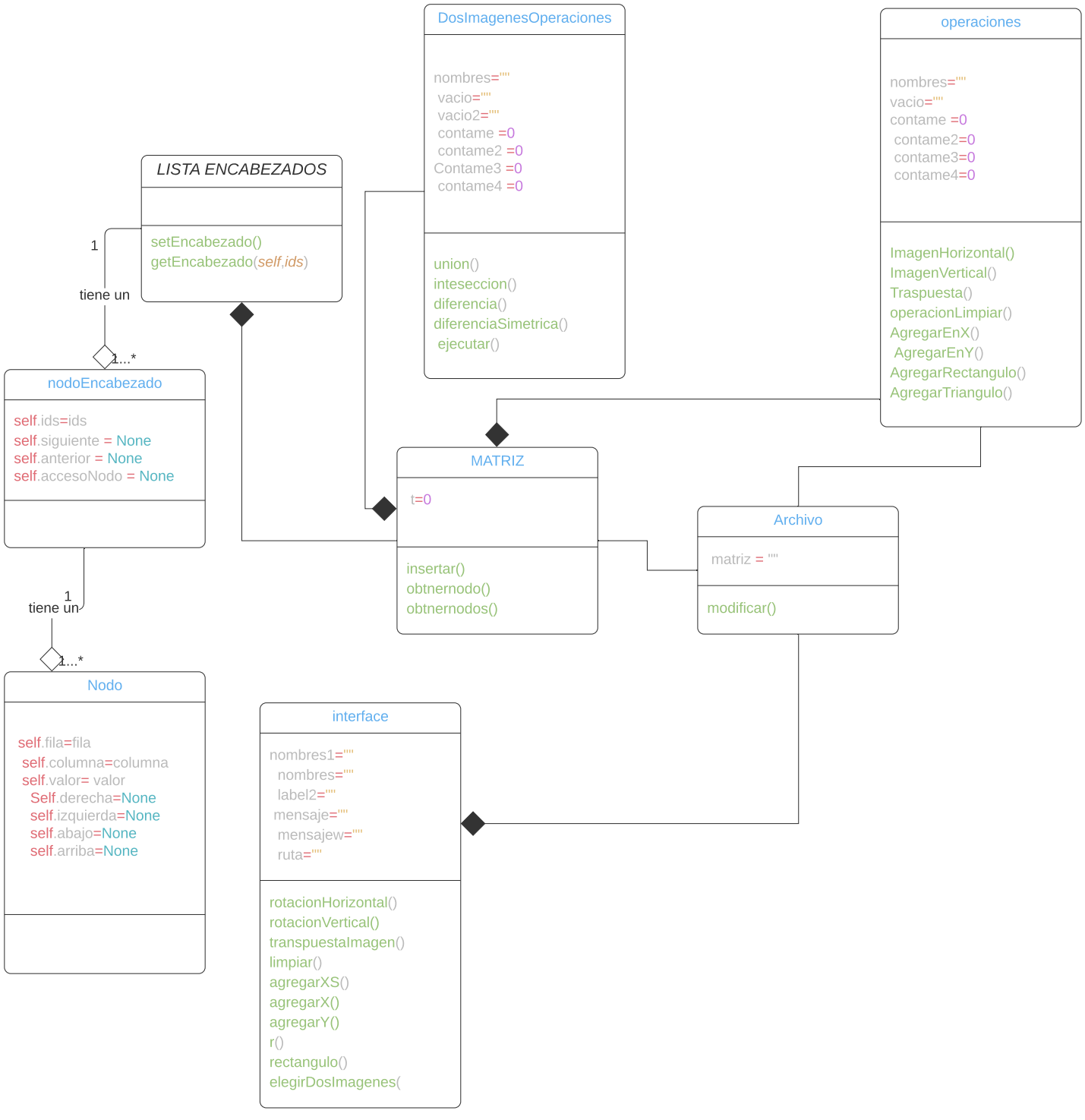

DosImagenesOperaciones
nombres=""
vacio=""
vacio2=""
contame =0
contame2 =0
Contame3 =0
contame4 =0
union()
inteseccion()
diferencia()
diferenciaSimetrica()
ejecutar()
operaciones
nombres=""
vacio=""
contame =0
contame2=0
contame3=0
contame4=0
ImagenHorizontal()
ImagenVertical()
Traspuesta()
operacionLimpiar()
AgregarEnX()
AgregarEnY()
AgregarRectangulo()
AgregarTriangulo()
LISTA ENCABEZADOS
setEncabezado()
getEncabezado(self,ids)
1
tiene un
1...*
nodoEncabezado
self.ids=ids
self.siguiente = None
self.anterior = None
self.accesoNodo = None
MATRIZ
t=0
insertar()
obtnernodo()
obtnernodos()
Archivo
matriz = ""
modificar()
1
tiene un
1...*
Nodo
self.fila=fila
self.columna=columna
self.valor= valor
Self.derecha=None
self.izquierda=None
self.abajo=None
self.arriba=None
interface
nombres1=""
nombres=""
label2=""
mensaje=""
mensajew=""
ruta=""
rotacionHorizontal()
rotacionVertical()
transpuestaImagen()
limpiar()
agregarXS()
agregarX()
agregarY()
r()
rectangulo()
elegirDosImagenes(

DIAGRAMA DE ACTIVIDADES DE OPERACIONES Y VALIDACIONES

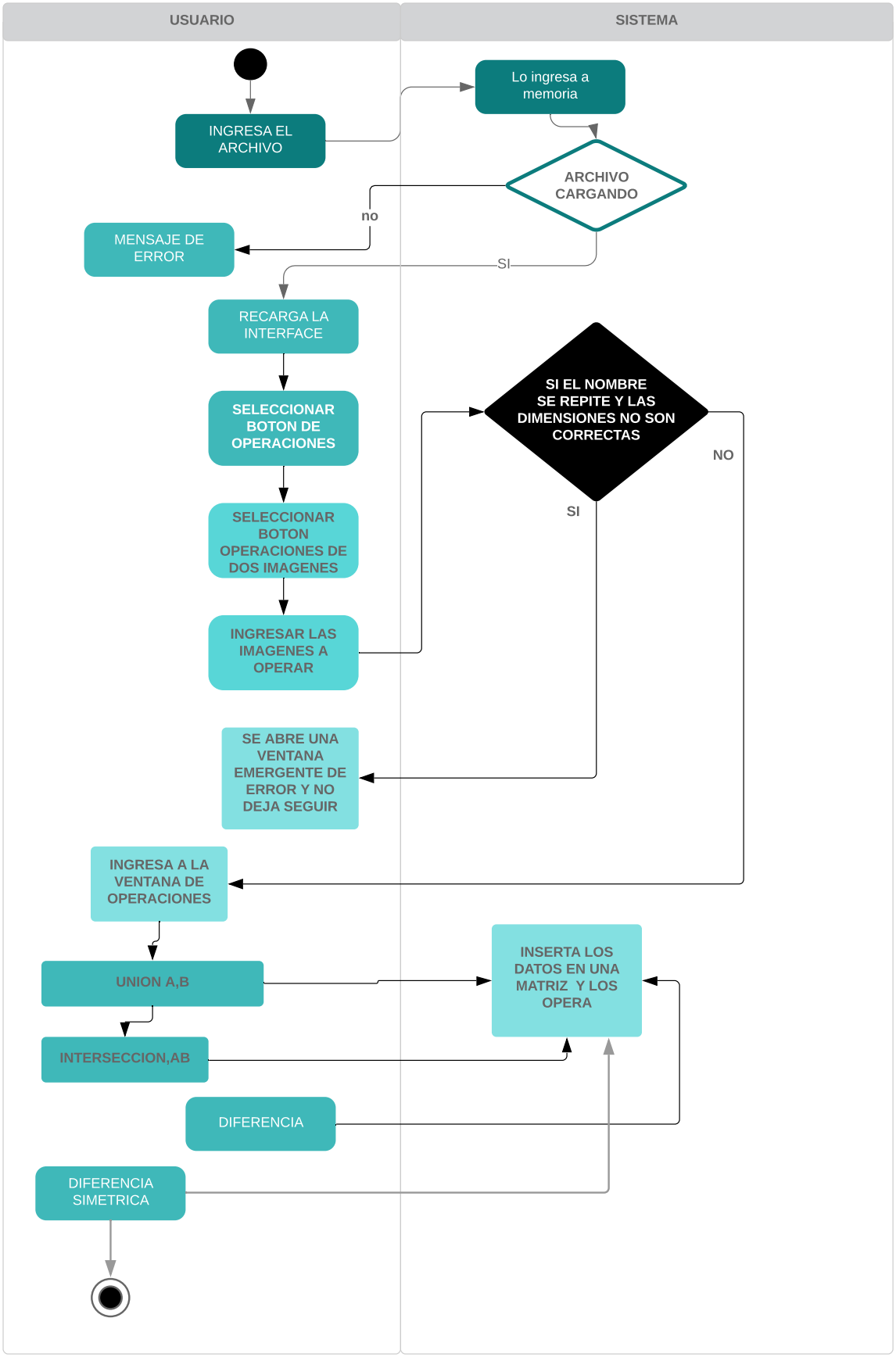


DIAGRAMA DE ACTIVIDADES DE OPERACIONES Y VALIDACIONES