ESTRUCTURA DE DATOS Y MANEJO DE MEMORIA

201905884 – Santiago Julián Barrera Reyes

Resumen

Las estructuras de datos son una forma de organizar datos de manera eficiente, podemos encontrar diferentes tipos de aplicaciones para estas, y algunas son altamente especializadas en tareas específicas.

Las empresas multinacionales consumen este recurso, ya que es eficiente, productivo y rápido. Hoy en día la demanda de estos algoritmos es altamente cotizada por lo que se inventaron herramientas de software para agilizar la creación de estructuras de datos, como Oracle Data Base, My SQL, PostgreSQL entre muchos otros.

En un contexto informático, las bases de datos nos brindan una ayuda para el manejo de datos de los usuarios empresas a la vanguardia de consumo de información como Google es indispensable el almacenar metadatos de cada usuario para lo cual consumen estructuras de datos o nuevos tipos de algoritmos de manejo de datos para esta tarea.

Siendo así las estructuras de datos un pilar en las ciencias de la computación, como en la programación.

Palabras Clave

Estructura, datos, algoritmos, lenguaje-modernos, bibliotecas, diseño, organización.

Abstract

Data structures are a way to organize data efficiently, we can find different types of applications for these, and some are highly specialized in specific tasks.

Multinational companies consume this resource, as it is efficient, productive, and fast. Today the demand for these algorithms is highly valued so software tools were invented to streamline the creation of data structures, such as Oracle Data Base, My SQL, PostgreSQL among many others.

In a computer context, databases provide us with an aid for the management of user data companies at the forefront of information consumption such as Google it is essential to store metadata of each user for which they consume data structures or new types of data management algorithms for this task.

Thus, data structures are a pillar in computer science, as in programming.

Keywords

Structure, data, algorithms, language-modern, libraries, design, organization.

Introducción

Las estructuras de datos son una forma de organizar datos de manera eficiente, podemos encontrar diferentes tipos de aplicaciones para estas, y algunas son altamente especializadas en tareas específicas.

Las estructuras de datos son usadas para grandes bases de datos y servicios de indización de internet. Por lo general, las estructuras de datos son clave en el manejo de algoritmos para bases de datos, también en algunos métodos formales de diseño y lenguajes de programación en los cuales destacan las estructuras de datos, en lugar de los algoritmos, como el factor clave de organización en el diseño de software.

Los lenguajes modernos por lo general vienen con bibliotecas estándar que implementan las estructuras de datos más comunes. Ejemplos de ello son la biblioteca stdlib de C, las colecciones de Java y las librerías .NET de Microsoft.

Desarrollo del tema

En el proyecto realizado se implementaron diferentes tipos de estructuras de datos las cuales cumplen con la función de almacenar y manejar de forma eficiente los datos proporcionados por el usuario.

La función del proyecto es generar un cambio de patrón vinculado a un costo de operación mínimo.

Para realizar de forma eficiente este proceso se debe almacenar la infamación, pero para poder lograr este propósito se debe de aplicar un paradigma de programación aplicado a la realidad.

Por ello se escogió el Paradigma de POO “Programación Orientada a Objetos”, el cual es necesario para generar clases las cuales contengan propiedades específicas y así heredar su funcionalidad a clases dependientes de esta.

Para generar una correcta organización de la información se implantan las estructuras de datos, las cuales están dirigidas a la eficiencia de almacenar información para esto se utilizaron dos tipos de listas:

A. Lista Enlazada Simple
B. Lista Enlazada Doble

Estas listas tienen la función de manejar el flujo de datos correspondiente a cada campo desarrollado.

Con el propósito de no generalizar el programa desarrollado se implementa una analogía la cual es representada en la siguiente tabla:

Clase	Contenido
Piso	Piso numero 1
Código	#52698
Patrón	WBBBW
Patrón	BWWWB

Fuente: elaboración propia, 2022 y 2 pág.

En la tabla se indica como clase contiene un parámetro asociado a esta, para su correcto almacenamiento y su futura utilización en el programa desarrollado.

De esta forma se puedo contener a una clase dentro aplicando la POO con los parámetros correctos para la centralización de información como se muestra en la siguiente tabla:

Clase	Clase	Clase
Piso	Piso 1	
	Código	
	#52698	Patrón
		WBBBW
		Patrón
		BWWWB

Fuente: elaboración propia, 2022 y 2 pág.

En la tabla se muestra como la clase Piso contiene a la clase Código que esta su vez contiene a la clase Patrón. Esto con el propósito de un manejo correcto de la información.

Para cada clase se implementaron listas las cuales se mencionaron con anterioridad. La lista simple enlazada se

representa con un solo apuntador a su siguiente esto quiere decir que no puede ver a su anterior un ejemplo de ello es:

Figura 1. Lista Enlazada Simple

Fuente: elaboración propia, 2022 y 3 pág.

La lista enlazada esta contiene los Pisos ingresados, pero se hace la salvedad que un piso como antes se mencionó contiene, el código y patrón ya que por medio del piso se pueden acceder a estos atributos.

Se explicará más adelante las funciones que posee una lista simple. Para poder contener los códigos, que estos a su vez contienen a los patrones se implementa lo que es una lista doblemente enlazada con el único fin de poder ver a su anterior.

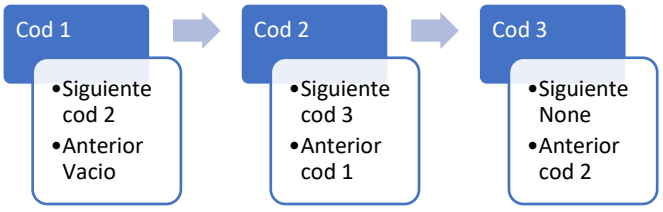

Figura 2. Lista Doblemente Enlazada

Fuente: elaboración propia, 2022 y 3 pág.

Se implemento la lista doblemente enlazada para los patrones los cuales contiene una secuencia para formar matrices, lo cual se explicará más adelante.

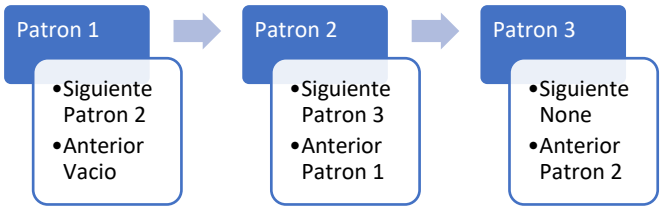

Figura 3. Lista Doblemente Enlazada

Fuente: elaboración propia, 2022 y 3 pág.

Para desarrollar el algoritmo de costo mínimo el cual es el requisito principal del programa desarrollado se debe de contemplar todo lo que contiene el piso.

Clase	Contenido
Piso	Nombre
	Filas
	Columnas
	Costo Volteo
	Costo Intercambio
	Patrones

Fuente: elaboración propia, 2022 y 3 pág.

Un piso se puede acceder a esta información con tenido en la memoria, y generar procesos uno de ellos es procesar el patrón de tal forma que genere una matriz para ello se hace una analogía y crea un algoritmo el cual contiene la información de este.

El primer paso para su elaboración es generar el constructo de la clase el cual contendrá los atributos del de cada letra en el patrón como se muestra:

```
class Codigo():

    def __init__(self, letra=None,fila=None,columna=None):
        self.letra = letra
        self.fila = fila
        self.columna = columna
```

Figura 4. Constructor del Patrón o código

Fuente: elaboración propia, 2022 y 3 pág.

En el constructor del patrón se nombra como la clase código ya que es un llamado a la secuencia que contiene el código por ello el nombre.

```
for f in range(1,int(fila)+1):
    for c in range(1,int(columna)+1):
        l = letra[cont]
        nuevoCodigo = Codigo(l, f, c)
        nuevoPatron.listaCodigos.append(nuevoCodigo)
        cont = cont + 1
```

Figura 5. Mandar atributos al constructor

Fuente: elaboración propia, 2022 y 4 pág.

En este sentido se sobre entiende que cada constructor contiene los atributos de cada acción asociada a si mismo.

Como se menciona una lista enlazada apunta a un solo nodo, pero que es un Nodo: no es mas que un punto de inserción al cual se vincula cierta información con esto se puede crear una lista enlazada simple.

```
class NodoSimple():

    def __init__(self,data=None,link=None):
        self.data = data
        self.link = link
```

Como se observa se tienen que instanciar un contendor de los atributos que contendrá nuestro constructor, en este caso es la clase data, a su vez se instancia un siguiente el cual apunta a la información siguiente de otro nodo como se muestra a continuación:

```
class PisoListaSimple():

    def __init__(self,nodecount=0):
        self.head = None
        self.nodecount = nodecount


    def append(self, nuevoPiso):
        if self.head is None:
            self.head = NodoSimple(data=nuevoPiso)
            self.nodecount = self.nodecount + 1
            return
        current = self.head
        while current.link:
            current = current.link
        current.link = NodoSimple(data=nuevoPiso)
        self.nodecount = self.nodecount + 1
```

Esto nos proporciona la creación de un Nodo al indicar que, en el NodoSimple, la data es igual al contenido de su constructor y esto facilita el manejo de la lista con la POO.

De forma similar se elabora el nodo de una lista doblemente enlazada:

```
class NodoDobleCod():

    def __init__(self, data=None,link=None, previous=None):
        self.data = data
        self.link = link
        self.previous = previous
```

Como se observa se tienen que instanciar un contendor de los atributos que contendrá nuestro constructor, en este caso es la clase data, a su vez se instancia un siguiente el cual apunta a la información siguiente de otro nodo, también cuenta con la instancia de un anterior el cual es previous, que tiene la función de apuntar al nodo que a sido ingresado con anterioridad.

Pero para vincular esta data el proceso no vario de ninguna forma:

```
class PatronListaDoble():

    def __init__(self):
        self.head = None


    def append(self, nuevoPatron):
        if self.head is None:
            self.head = NodoDoble(data=nuevoPatron)
        else:
            current =
NodoDoble(data=nuevoPatron,link=self.head)
            self.head.previous = current
            self.head = current
```

Con todo el material descrito se procede a crear el algoritmo de costo mínimo.

El cual se genera a través de ciclos if extrayendo la información y asociando a el cambio introducido en el programa.

```
for f in range(1, int(filas) + 1):
    for c in range(1, int(columnas) + 1):
        letra_c = lista_c.buscarLetra(f, c)
        letra_b = lista_b.buscarLetra(f, c)
        link_b = lista_b.buscarLetra(f, c + 1)
        down_b = lista_b.buscarLetra(f + 1, c)
        link_c = lista_c.buscarLetra(f, c + 1)
        down_c = lista_c.buscarLetra(f + 1, c)

        if letra_b == letra_c:
            nuevo = nuevoPiso.listaPatron.busquedaPatron(nombreCodigo).listaCodigos
            if letra_b == nuevo.buscarLetra(f, c):
                cadena = cadena + "Item no requiere acción.\n"
            elif nuevo.buscarLetra(f, c) == None:
                nuevoCodigo = Codigo(letra_c, f, c)
                nuevoPatron.listaCodigos.append(nuevoCodigo)
                cadena = cadena + "Item no requiere acción.\n"
        elif letra_b != letra_c:
            if (lista_b.buscarLetra(f, c + 1) and lista_b.buscarLetra(f + 1, c)) is not None:
                if (letra_b == link_c and link_b == letra_c):
                    nuevoCodigo = Codigo(link_b, f, c)
                    nuevoPatron.listaCodigos.append(nuevoCodigo)
                    nuevoCodigo = Codigo(letra_b, f, c + 1)
                    nuevoPatron.listaCodigos.append(nuevoCodigo)
                    cadena = cadena + f"Item requirio la accion Intercambio: {letra_b, f, c} switch {link_b, f, c + 1).\n"
                    inter = inter + intercambiar
                    lista_b.modificar(link_b, f, c)
```

Figura 6. Creación del algoritmo Costo Mínimo

Fuente: elaboración propia, 2022 y 4 pág.

El código nos brinda la opción de buscar letra, con la condición de que contemos con la fila y columna en la que se encuentra en ese momento.

El algoritmo utilizado es:

$$total_{Precio} = volteo + intercambio$$

$$total_{volteo} = cantidad_{pisos} * Precio_{volteo}$$

$$total_{intercambio} = cantidad_{pisos} * Precio_{intercambio}$$

Lo que entrega la cantidad y en base a ello podemos utilizar sus resultados para entablar el nuevo patrón como se muestra a continuación.

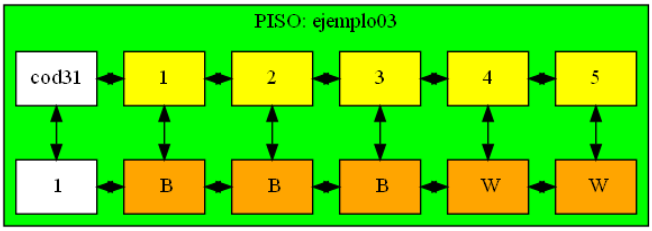

Figura 7. Creación del algoritmo Costo Mínimo

Fuente: elaboración propia, 2022 y 4 pág.

La imagen nos muestra la representación del patrón formado a partir del patrón base que se insertó con anterioridad.

Para generar de esta manera la grafica del patrón de forma matricial se necesita una librería capaz de realizar este cambio, por términos del proyecto nos obligan a trabajar con la librería graphviz de Python, el cual es el lenguajes que se utilizó para desarrollar este proyecto, refiriéndonos a la librería mencionada nos brinda la función de generar un grafica detallada en forma matricial junto con el lenguaje Python.

Una representación de esto es la que se muestra a continuación:

```
diagrama ='"
digraph S{
   node[shape=box fillcolor="white" style =filled]

   subgraph cluster_p{
   label = "PISO: "'+ nombrePiso +'""
   bgcolor = "green"
   raiz[label ='" + nombreCod + '"]
   edge[dir = "both"]
'"
```

De esta manera se ingresan los datos a esta herramienta para poder editar de forma automática la gráfica.

Tomando todo lo que se mencionó previamente, estas funciones necesitan de un menú para acceder a estas. Por lo que se implementa un menú el cual dará forma concreta a las funcionalidades de este.

```
   <--Menu Principal-->

Selecciona una opción:

        1) - Cargar Archivo
        2) - Graficar Patron
        3) - Cambiar Patron
        4) - Imprimir Acción
        0) - Salir

Inserta el numero de la opcion: >>
```

Figura 8. Menú de Ejecución

Fuente: elaboración propia, 2022 y 4 pág.

Como se puede observar el menú implementado permite que las opciones generen las acciones permitidas para los cambios

de patrones teniendo en cuenta que los cambios de patrones solo se pueden dar en el mismo piso.

Conclusiones:

En programación, una estructura de datos puede ser declarada inicialmente escribiendo una palabra reservada, luego un identificador para la estructura y un nombre para cada uno de sus miembros, sin olvidar los tipos de datos que estos representan. Generalmente, cada miembro se separa con algún tipo de operador, carácter o palabra reservada.

La mayoría de los lenguajes ensambladores y algunos lenguajes de bajo nivel, tales como BCPL, carecen de soporte de estructuras de datos. En cambio, muchos lenguajes de alto nivel y algunos lenguajes ensambladores de alto nivel, tales como MASM, tienen algún tipo de soporte incorporado para ciertas estructuras de datos, tales como los registros y arreglos.

Bibliografía:

Backman, k. (2012). Structured Programming with C++. Bookboom.com.

Cairó/Gardati. (2000). Estructura de Datos. México: Mc Graw hill.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (1991). Fundamentals of Software Engineering . New Jersey: Prentice Hall.

Horowits, E. (1978). Fundamentals of Computer Algorithms. New York: Computer Science Press.

Levin, G. (2004). Computaciónn y Programación Moderna, perspectiva integral de la Informática. México: Addison Wesley.

Loomis, M. E. (2013). Estructura de Datos y Organización de Archivos. México: Prentice Hall.

Extensión: de cuatro a siete páginas como máximo