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

PROYECTO 1

202010406 – Dennis Mauricio Corado Muñóz

Resumen

El proyecto desarrollado e implementado tenía como fin desarrollar el conocimiento respecto al uso de la memoria dinámica por medio de estructuras de datos, planteando un problema cuyo desarrollo fuese utilizando estructuras de datos, el problema consistía en una lectura de un archivo de extensión XML que contenía información de terrenos de exploración con posiciones y su respectivo gasto de combustible, derivado de este archivo se debía obtener la información a procesar, posteriormente encontrar el camino más óptimo para recorrer el terreno y haciendo uso de la herramienta "GraphViz" se debía generar una imagen que mostrara las posiciones del archivo en una matriz.

Palabras clave

Estructura de datos, TDA, Nodo, grafo, listas y matrices.

Abstract

The purpose of the developed and implemented project was to develop knowledge regarding the use of dynamic memory through data structures, posing a problem whose development was using data structures, the problem consisted of reading an XML extension file that contained information on exploration lands with positions and their respective fuel costs, derived from this file the information to be processed should be obtained, subsequently finding the most optimal way to travel the terrain and using the "GraphViz" tool an image should be generated which will show the positions of the file in an array.

Keywords

Data Structures, ADT, Node, graph, lists and matrix, arrays.

Introducción

La implementación de TDA's para presentar soluciones mejor estructuradas haciendo uso de la memoria dinámica representa una buena alternativa en el manejo de la información. A lo largo del proyecto planteado se hicieron usos de estructuras de datos como lo son, las matrices ortogonales, listas enlazadas simples y dobles, también se hizo uso de la teoría de grafos para poder brindar una solución al problema planteado del camino más optimo a utilizar esto para poder recorrer los distintos nodos contenidos dentro de la matriz. También se hizo uso de la herramienta para gráficos "GraphViz" esto para brindar una representación gráfica de los distintos nodos que poseía el terreno cargado al programa y así poder entender de mejor forma cómo es que este se estructura.

Desarrollo del tema

Para la resolución del problema planteado se hizo uso de estructuras de datos más específicamente de, matriz ortogonal, listas enlazadas dobles y simples, cola de prioridad. Para que estas almacenaran la información deseada se hicieron uso de diferentes nodos, cada uno con características diferentes.

Se usaron alrededor de 14 clases divididas en 10 archivos, esto con el objetivo de seccionar el código y que la persona que vea el código tenga un mejor entendimiento de lo que se hizo en cada uno, también se agregaron comentarios en este.

La clase "main" contiene una serie métodos que se usaran para darle inicio a la aplicación entre estos tenemos el menú que dentro de este se solicitara al usuario la opción a realizar y se hará uso de los demás métodos cada uno con su respectiva función, los métodos secundarios hacen uso de OOP para poder crear objetos de diferentes categorías o bien acceder a un método del objeto en particular.

En la carpeta "Modelas" se encuentran los archivos que contienen las clases que serán útiles para el desarrollo del problema, específicamente la matriz ortogonal, las listas enlazadas cada una con diferentes propósitos pero todas comparten la característica de un solo puntero y las clases de los distintos nodos.

En el archivo "CargarArchivo" se crea la clase "Cargar" que hace uso de la herramienta "ElementTree" para poder leer el contenido del archivo y crea objetos de tipo "ListaTerrenos" y "Matriz" para almacenar esta información.

En el archivo "ProcesarTerreno" se creo la clase "Procesar" que dentro contiene diferentes métodos que hacen uso de la OOP para poder crear objetos almacenados en la carpeta "Models" en este caso objetos de tipo "LinkedList()" y "PriorityQueue".

Luego se encuentra el archivo "EscribirArhivo" que se encarga como su nombre lo dice de escribir el archivo con la salido de las posiciones finales y las posiciones del camino optimo asi como el gasto de combustible haciendo uso de la herramienta "ElementTree".

Y finalmente el archivo "Graficar" que hace uso de concatenación de cadenas para generar una cadena final que será escribita en un archivo ".dot" y por medio de un comando que se ejecuta en el sistema se obtiene una imagen que corresponde a la gráfica de la matriz o en este caso del terreno.

Para darle un vistazo mas de cerca a la carpeta "Models" que como se menciona arriba contiene modelos que se usaran en las distintas fases de la resolucion de problema la clase matriz hace uso de los nodosMatriz que llevan toda la información que necesita la matriz asi como 4 apuntadores que les indicaran donde a la par de que otros nodos se encuentra, el archivo de "LinkedLists" contiene diferentes listas cada una con su información y métodos necesarios.

Conclusiones

El uso de TDA's hace que la programación sea mas efectiva evitando el uso de memoria estática brindando soluciones mas abstractas pero mejor estructuradas, las estructuras de datos pueden adaptarse según sea la necesidad y se pueden utilizar métodos según necesidad por lo tanto se concluyo que el uso de este tipo de elementos en la programación brinda una mejor solución y un mejor entendimiento del problema.

Referencias bibliográficas

Charles Severance (2015) *Python Para Informaticos.*
Kent D. Lee (2017) *Fundations of Programming Languages.*
Christian Huemer, Gerti Kappel, Martina Seildl, Marion Scholz (2015) *UML @ Classroom An Introduction to Object-Oriented Modeling.*

Anexos

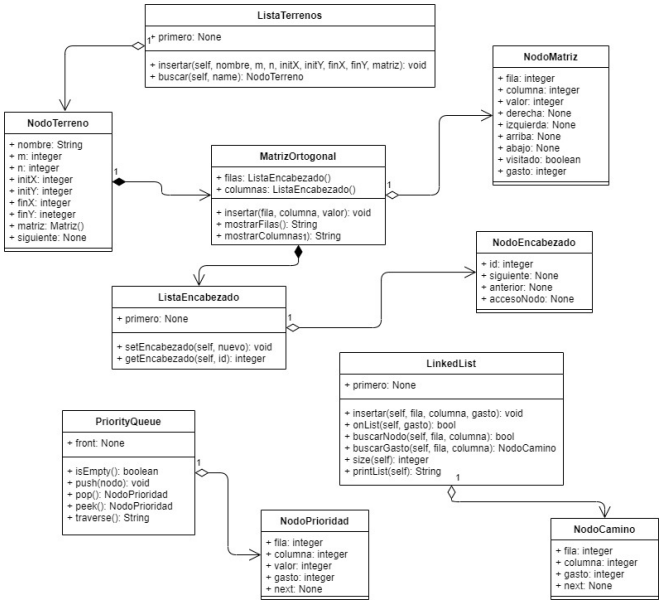

Figura 1. Diagrama de clases.
Fuente: Elaboración Propia (2021)

```python
class NodoMatriz:
    def __init__(self, fila, columna, valor):
        self.valor = valor
        self.fila = fila
        self.columna = columna
        self.derecha = None
        self.izquierda = None
        self.arriba = None
        self.abajo = None
        self.visitado = False
        self.gasto = 0
    You, 2 weeks ago • Matriz ortogonal creada
You, 18 minutes ago | 1 author (You)
class NodoEncabezado:
    def __init__(self, id):
        self.id = id
        self.siguiente = None
        self.anterior = None
        self.accesoNodo = None

You, 18 minutes ago | 1 author (You)
class NodoTerreno:
    def __init__(self, nombre, m, n, initX, initY, finX, finY, matriz):
        self.nombre = nombre
        self.m = m
        self.n = n
        self.initX = initX
        self.initY = initY
        self.finX = finX
        self.finY = finY
        self.matriz = matriz
        self.siguiente = None

You, 2 days ago | 1 author (You)
class NodoPrioridad:
    def __init__(self, fila, columna, valor, gasto):
        self.fila = fila
        self.columna = columna
        self.valor = valor
        self.gasto = gasto
        self.next = None
You, 18 minutes ago | 1 author (You)
```

Figura 2. Código para cada Nodo.
Fuente: Elaboración Propia (2021)

```python
class MatrizOrtogonal:      You, 18 minutes ago • Ultimos cambio

    def __init__(self):
        self.filas = ListaEncabezado()
        self.columnas = ListaEncabezado()

    def insertar(self, fila, columna, valor):
        nuevo = NodoMatriz(fila, columna, valor)

        # Insercion de encabezado por filas
        eFila = self.filas.getEncabezado(fila)
        if eFila == None:
            eFila = NodoEncabezado(fila)
            eFila.accesoNodo = nuevo
            self.filas.setEncabezado(eFila)
        else:
            if nuevo.columna < eFila.accesoNodo.columna:
                nuevo.derecha = eFila.accesoNodo
                eFila.accesoNodo.izquierda = nuevo
                eFila.accesoNodo = nuevo
            else:
                actual = eFila.accesoNodo
                while actual.derecha is not None:
                    if nuevo.columna < actual.derecha.columna:
                        nuevo.derecha = actual.derecha
                        actual.derecha.izquierda = nuevo
                        nuevo.izquierda = actual
                        actual.derecha = nuevo
                        break
                    actual = actual.derecha
                if actual.derecha is None:
                    actual.derecha = nuevo
                    nuevo.izquierda = actual
```

Figura 3. Código de Matriz Ortogonal
Fuente: Elaboración Propia (2021)

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

    def insertar(self, nombre, m, n, initX, initY, finX, finY, matriz):
        nuevo = NodoTerreno(nombre, m, n, initX, initY, finX, finY, matriz)
        if self.primero is None:
            self.primero = nuevo
        else:
            actual = self.primero
            while actual.siguiente is not None:
                actual = actual.siguiente
            actual.siguiente = nuevo

    def buscar(self, name):
        if self.primero is None:
            print("LISTA VACIA")
        else:
            actual = self.primero
            encontrado = False
            while actual and encontrado is False:
                if actual.nombre == name:
                    encontrado = True
                else:
                    actual = actual.siguiente
            if actual is None:
                print('ERROR: El terreno no esta en la lista.')
            return actual
                You, 2 days ago • Cambios minimos y nombres en archivos
```

Figura 4. Código Lista Simple Terrenos.
Fuente: Elaboración Propia (2021)