Manual Técnico

Proyecto 2

Llega Rapidito

Curso: Estructura de Datos
Estudiante: Herberth Julian Reyes Pacajoj
Registro académico: 202230236
Carnet: 3352309380901

Generalidades

Herramientas utilizadas

Sistema operativo: Ubuntu - versión 24.04.1 LTS

Ubuntu es un sistema operativo de código abierto basado en Linux, diseñado para ser accesible, seguro y fácil de usar. Es una de las distribuciones de Linux más populares en el mundo.

Lenguaje de programación: Python - versión 3.12.3

Python es un lenguaje de programación de alto nivel, interpretado y multiparadigma, diseñado para ser fácil de leer y escribir.

Herramienta de GUI: PyQt5 - versión 5.15.11

PyQt5 es un conjunto de enlaces de Python para la biblioteca Qt, que permite crear interfaces gráficas de usuario (GUIs) modernas y potentes.

Gráficos: GRAPHVIZ - versión 2.43

Graphviz es un programa de visualización gráfica open source. Esta herramienta permite la generación de gráficos para representar información en forma de diagramas de redes y gráficos abstractos.

IDE: Visual Studio Code - versión 1.96.2

VSCode es un editor de código fuente ligero, extensible y gratuito desarrollado por Microsoft.

Controlador del proyecto: GIT & GITHUB

Git es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo tipo de proyectos, con rapidez y eficiencia. GitHub es una plataforma de hospedaje de código para el control de versiones y la colaboración. Este permite que varias personas trabajen juntas en proyectos desde donde sea.

Clases y métodos

Clase: main

Método: __init__(self)

El método init de la clase main inicializa todas las estructuras de datos utilizadas en el programa, así mismo muestra la ventana inicial

Clase: Cliente

Método: __init__(self, dpi:int, nombres:str, apellidos:str, genero:chr, telefono:int, direccion:str)

El método init de la clase Cliente inicializa un nuevo Cliente con los parámetros requeridos (dpi, nombres, apellidos, genero, telefono y direccion). Los atributos inicializados están encapsulados

Método: Getters & Setters

Al ser privados los atributos de la clase Cliente, se usan getters y setters para manejar la información de un Cliente

Clase: Ruta

Método: __init__(self, origen: str, destino: str, tiempo: int)

El método init de la clase Ruta inicializa una nueva Ruta con los parámetros requeridos (origen, destino y tiempo). Los atributos inicializados están encapsulados

Método: Getters & Setters

Al ser privados los atributos de la clase Ruta, se usan getters y setters para manejar la información de una Ruta

Clases y métodos

Clase: Vehiculo

Método: __init__(self, placa: str, marca: str, modelo: int, precio: float)

El método init de la clase Vehiculo inicializa un nuevo Vehiculo con los parámetros requeridos (placa, marca, modelo y precio). Los atributos inicializados están encapsulados

Método: Getters & Setters

Al ser privados los atributos de la clase Vehiculo, se usan getters y setters para manejar la información de un Vehiculo

Clase: Viaje

Método: __init__(self, origen: str, destino: str, fecha: str, hora: str, cliente: Cliente, vehiculo: Vehiculo)

El método init de la clase Viaje inicializa un nuevo Viaje con los parámetros requeridos (origen, destino, fecha, hora, cliente y vehiculo). Los atributos inicializados están encapsulados. Además inicializa una lista vacía que pertenece a la ruta tomada en un viaje

Método: Getters & Setters

Al ser privados los atributos de la clase Viaje, se usan getters y setters para manejar la información de un Viaje

Método: obtener_tam_lista(self)

Este método sirve para obtener el tamaño de la lista que contiene los vértices que pertenecen a la ruta tomada en el viaje

Clases y métodos

Método: obtener_tiempo_total(self)

Este método se utiliza para obtener el tiempo total que tarda el viaje

Método: graficar_ruta(self)

Este método sirve para hacer el código DOT que va a servir para generar el gráfico de la ruta tomada en el viaje

Clase: CargarDatos

Método: cargar_clientes(self, ruta: str, lista: ListaCircularDoble)

Este método abre y lee un archivo con el parámetro ruta e inserta a la lista circular todos los clientes que obtiene del archivo

Método: cargar_vehiculos(self, ruta: str, arbolB: ArbolB)

Este método abre y lee un archivo con el parámetro ruta e inserta al Arbol B todos los vehiculos que obtiene del archivo

Método: cargar_rutas(self, ruta_a: str, grafo: ListaAdyacencia)

Este método abre y lee un archivo con el parámetro ruta e inserta a la lista de adyacencia todas las rutas que obtiene del archivo

Clase: Graficar

Método: graficar(self, path: str, grafico: str)

Este es el método que sirve para graficar todas las estructuras de datos, en este se pide el path en el que se guardará la imagen SVG y el código DOT del gráfico

Clases y métodos

Clase: NodoAB

Método: __init__(self, hoja: bool = False)

El método init de la clase NodoAB inicializa un nuevo nodo del Arbol B con atributos como: es hoja, lista de claves y una lista de hijos. Si no se pasa un atributo booleano, el nodo es una hoja

Método: Getters & Setters

Al ser privados los atributos de la clase NodoAB, se usan getters y setters para manejar la información de un NodoAB

Clase: ArbolB

Método: __init__(self, orden: int)

El método init de la clase ArbolB inicializa un nuevo Arbol B con un orden que depende del parametro con el mismo nombre, además incializa un nuevo nodo que es la raíz del mismo

Método: insertar_vehiculo(self, vehiculo: Vehiculo)

Este es el método inicial que sirve para insertar vehiculos en el Arbol B

Método: insertar_vehiculo_no_completo(self, raiz: NodoAB, vehiculo: Vehiculo)

Este es el método recursivo que sirve para insertar vehículos en el Arbol B cumpliendo con las condiciones necesarias del Arbol B y dependiendo de la configuración de este, se hacen diferentes operaciones

Clases y métodos

Método: dividir_pagina(self, raiz: NodoAB, posicion: int)

En el momento en el que una página sobrepasa su límite de claves (orden - 1), es necesario dividir dicha página para que el Arbol B este balanceado, este método se encarga de dicha división

Método: buscar_vehiculo(self, placa: str)

Este es el método inicial que sirve para buscar algún vehiculo dentro del Arbol B

Método: buscar(self, placa: str, raiz: NodoAB)

Este es el método recursivo que sirve para buscar algún vehiculo dentro del Arbol B, este método va buscando entre páginas e hijos dependiendo de la Placa del vehiculo

Método: obtener_placas(self)

Este es el método inicial que sirve para obtener todas las placas de los vehiculos que estan dentro del Arbol B

Método: recorrer_arbol(self, raiz: NodoAB, placas: list[str])

Este es el método recursivo que sirve para buscar obtener todas las placas de los vehiculos que están en el Arbol B

Método: graficar_arbol(self)

Este es el método inicial que sirve para generar el código DOT que se va a ejecutar para hacer el gráfico del Arbol B

Método: graficar(self, nodo: NodoAB, id: list[int] = [0])

Este es el método recursivo que sirve para generar el código DOT para hacer el gráfico del Arbol B

Clases y métodos

Clase: NodoLCDE

Método: __init__(self, cliente: Cliente)

El método init de la clase NodoLCDE inicializa un nuevo nodo de la Lista Circular Doblemente Enlazada con el cliente que se pasa y también se inicializan los apuntadores siguiente y anterior como None

Método: Getters & Setters

Al ser privados los atributos de la clase NodoLCDE, se usan getters y setters para manejar la información de un NodoLCDE

Clase: ListaCircularDoble

Método: __init__(self)

El método init de la clase ListaCircularDoble inicializa un nueva lista circular doblemente enlazada con los nodos cabeza y cola como None

Método: insertar(self, cliente: Cliente)

Este método sirve para insertar un nuevo cliente a la lista circular doblemente enlazada, cabe resaltar que la lista se va ordenando, dependiendo del dpi del cliente a insertar, al momento de insertar un nuevo nodo

Método: buscar(self, dpi: int)

Este método sirve para buscar al cliente con el DPI que se pasa como parámetro dentro de la lista circular doblemente enlazada

Clases y métodos

Método: eliminar(self, dpi: int)

Este método sirve para eliminar el cliente con el DPI que se pasa como parámetro dentro de la lista circular doblemente enlazada

Método: graficar(self)

Este es el método sirve para generar el código DOT para hacer el gráfico de la lista circular doblemente enlazada

Clase: NodoLS

Método: __init__(self, valor)

El método init de la clase NodoLS inicializa un nuevo nodo de una Lista Simple con el valor abstracto que se pasa y también se inicializa el apuntador al siguiente NodoLS como None

Método: Getters & Setters

Al ser privados los atributos de la clase NodoLS, se usan getters y setters para manejar la información de un NodoLS

Clase: ListaSimple

Método: __init__(self)

El método init de la clase ListaSimple inicializa un nueva lista simple con el nodo cabeza como None

Método: insertar(self, valor)

Este método sirve para insertar al final de la lista simple un nuevo nodo con el valor abstracto que se pasa como parámetro

Clases y métodos

Método: insertar_frente(self, valor)

Este método sirve para insertar al principio de la lista simple un nuevo nodo con el valor abstracto que se pasa como parámetro

Método: insertar_viaje(self, valor)

Este método sirve para insertar al final de la lista simple un nuevo nodo que tiene como valor un objeto Viaje que se pasa como parámetro

Método: buscar_viaje(self, id: int)

Este método sirve para buscar el viaje con el ID que se pasa como parámetro que este dentro de la Lista Simple

Método: buscar_ciudad(self, valor: str)

Este método sirve para buscar el vértice de la ciudad que se pasa como parámetro que este dentro de la Lista Simple de vecinos

Método: graficar_viajes(self)

Este es el método sirve para generar el código DOT para hacer el gráfico de la lista simple que contiene todos los viajes registrados

Clase: NodoC

Método: __init__(self, valor)

El método init de la clase NodoC inicializa un nuevo nodo de una Cola con el valor abstracto que se pasa y también se inicializa el apuntador al siguiente NodoC como None

Método: Getters & Setters

Al ser privados los atributos de la clase NodoC, se usan getters y setters para manejar la información de un NodoC

Clases y métodos

Clase: Cola

Método: __init__(self)

El método init de la clase Cola inicializa un nueva cola con el nodo cabeza como None

Método: encolar(self, valor)

Este método ingresa un nuevo nodo, con el valor que se pasa como parámetro, al final de la cola

Método: desencolar(self)

Este método saca al nodo cabeza de la cola y lo devuelve. Se usa la política FIFO

Método: ordenar(self)

Este método ordena la cola con el método burbuja para simular una cola de prioridad

Método: buscar(self, valor)

Este método buscar el valor pasado como parámetro en los nodos de la cola

Método: esta_vacia(self)

Este método sirve para saber si la cola está vacía

Clases y métodos

Clase: Vertice

Método: __init__(self, ciudad: str, peso: int = 0, padre = None)

El método init de la clase Vertice inicializa un nuevo Vertice con la ciudad, el peso y el padre que se pasa y también se inicializa una lista simple que va a contener a los vecinos del vertice

Método: Getters & Setters

Al ser privados los atributos de la clase Vertice, se usan getters y setters para manejar la información de un Vertice

Clase: ListaAdyacencia

Método: __init__(self)

El método init de la clase ListaAdyacencia inicializa un nueva lista de adyacencia con una lista simple que reprensenta a los vértices cabecera

Método: insertar_ruta(self, ruta: Ruta)

Este método sirve para insertar una nueva ruta a la lista de adyacencia, es decir, se inserta el origen a los vértices cabeceras (sino está) y se le agrega como vecino el destino

Método: obtener_ruta(self, origen: str, destino: str)

Este método sirve para obtener la ruta que un viaje va a tener dependiendo del origen y el destino que son pasados como parámetro

Clases y métodos

Método: obtener_ruta_corta(self, destino: str, nodo_visitados: Cola, nodos:Cola)

Este método sirve para obtener la ruta corta en el grafo entre un origen y un destino basándose en la búsqueda de costo uniforme utilizando una cola de prioridad

Método: esta_visitado(self, vertice: Vertice, nodos_visitados: Cola)

Este método sirve para verificar si un vértice ya ha sido visitado y es utilizado para hallar la ruta corta

Método: esta_vacia(self)

Este método sirve para verificar si la lista de adyacencia está vacía

Método: graficar(self)

Este es el método sirve para generar el código DOT para hacer el gráfico del grafo que representa la lista de adyacencia