RUTAS DE SENDERISMO

DESARROLLO DE PROGRAMA PARA RUTAS DE SENDERIMO,
BASADO EN EL MODELO DE CASCADA.

1-ANÁLISIS Y DEFINICIÓN DE REQUERIMIENTOS

IDENTIFICACIÓN DEL PROBLEMA

DIFICULTAD EN ELEGIR RUTAS SEGURAS Y OPTIMAS PARA SENDERISTAS.

RESTRICCIONES:

-SE NECESITAN DATOS PREVIOS DE TODAS LAS RUTAS.

-SOLO SE PUEDE APLICAR A UNA RUTA A LA VEZ.

SERVICIOS Y METAS:

-PROPORCIONAR LAS MEJORES RUTAS DE SENDERISMO.

-PROGRAMACIÓN SENCILLA
-PROGRAMA FACIL DE UTILIZAR
-ADAPTABILIDAD

2-DISEÑO DEL SISTEMA Y DEL SOFTWARE

DIAGRAMAS

antes de ponernos a programar, es necesario hacer algunos diagramas previos, el mas util de estos, es el uml

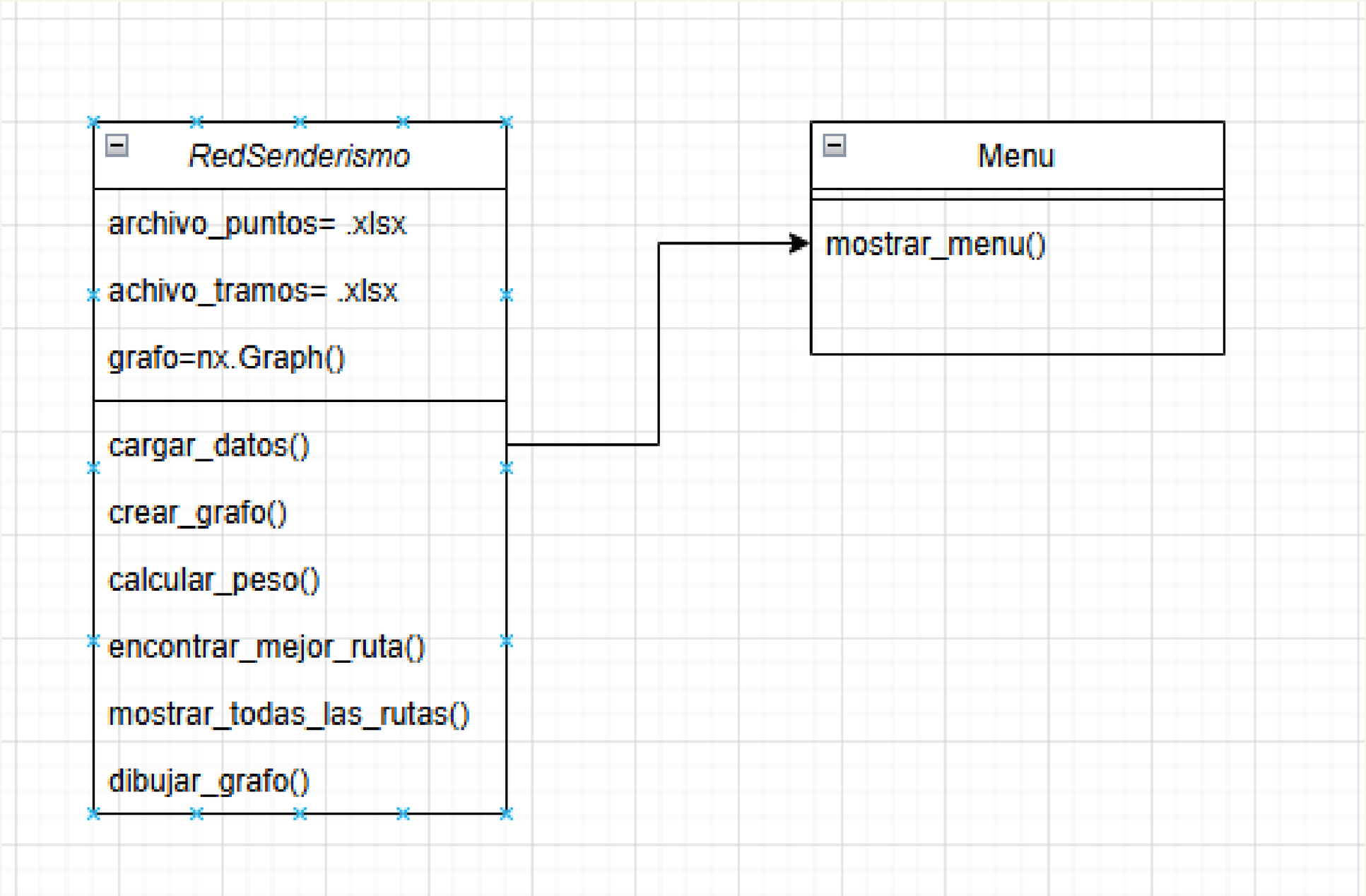
TIPO DE PROGRAMACIÓN

el programa estará hecho en python con progrmaamcion orientada a objetos ya que es la forma mas optima de modelar el tipo de programa que se requiere.

MODELO A PROGRAMAR

para este proyecto, es necesario entender perfectamente lo que estamos modelando en el programa y pensar en las posibles situaciones a las cuales pueda estar sometido en un uso normal.

DIGRAMA UML


RedSenderismo
archivo_puntos= .xlsx
achivo_tramos= .xlsx
grafo=nx.Graph()
cargar_datos()
crear_grafo()
calcular_peso()
encontrar_mejor_ruta()
mostrar_todas_las_rutas()
dibujar_grafo()
Menu
mostrar_menu()

3.IMPLEMENTACION Y PRUEBA DE UNIDAD

01

PROGRAMACIÓN

Ya que tenemos el diagrama uml de la sección pasada, requerimos pasarlo a un lenguaje de programación.

03

COMPOBACIÓN

Verificar que cada unidad cumpla con su especificación.

4-INTEGRACIÓN Y PRUEBA DE SISTEMA

INTEGRACIÓN DE LOS PROGRAMAS INDIVIDUALES PARA SU PRUEBA TRABAJANDO COMO UN SISTEMA COMPLETO

ASEGURARSE QUE SE CUMPLAN LOS REQUERIMIENTOS DEL SOFTWARE.

FASE DE LIBERACIÓN

BIBLIOTECAS

```
# ##########################################################################
# **    Proyecto        : Ruta de senderismo
# **    Herramienta     : Visual Studio Code
# **    Fecha/Hora      : 24/04/2025 23:02 pm
# **    Descripción     :Este programa permite calcular la mejor ruta entre puntos de una ruta de senderismo,
# **  utilizando un grafo generado a partir de archivos Excel. Cada nodo representa un
# **  punto de interés, y cada arista representa un tramo entre dos puntos.
# **  La mejor ruta se determina en base al peso de cada tramo, considerando principalmente
# **  el desnivel, seguido de la distancia y el tiempo estimado de recorrido.
# **  También se muestran todas las rutas posibles
# **    By              : Hector Jimenez
# **    contact         : hjimenezm2101@alumno.ipn.mx
#  ############################################################################

# ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# :                        Librerias / Bibliotecas / Modulos                 |
# ::::::::::::::::::::::::::::::::::::::::::::::::::::2:::::::::::::::::::::
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```

```python
class RedSenderismo:
    def __init__(self, archivo_puntos, archivo_tramos):
        self.archivo_puntos = archivo_puntos
        self.archivo_tramos = archivo_tramos
        self.grafo = nx.Graph()
        self.cargar_datos()
        self.crear_grafo()

    def cargar_datos(self):
        self.df_puntos = pd.read_excel(self.archivo_puntos)
        self.df_tramos = pd.read_excel(self.archivo_tramos)
```

```python
def crear_grafo(self):
    for i, row in self.df_puntos.iterrows():
        nodo = row["Numero de Estacion"]
        self.grafo.add_node(nodo,
                            descripcion=row["Descripcion"],
                            tipo=row["Tipo"],
                            altura=row["Altura"])
    for i, row in self.df_tramos.iterrows():
        origen, destino = map(int, row["Tramo"].split(","))
        altura_origen = self.grafo.nodes[origen]["altura"]
        altura_destino = self.grafo.nodes[destino]["altura"]
        desnivel = abs(altura_origen - altura_destino)

        self.grafo.add_edge(origen, destino,
                            descripcion=row["Descripcion"],
                            distancia=row["Distancia"],
                            tiempo=row["Tiempo recorrido"],
                            desnivel=desnivel)
```

```python
def calcular_peso(self, u, v, datos_arista):
    a=0.6
    b=0.3
    c=0.1
    return a * abs(datos_arista["desnivel"]) + b * datos_arista["distancia"] + c * datos_arista["
```

ALGORITMO DE DIJKSTRA

```
def encontrar_mejor_ruta(self, inicio, fin):
    try:
        mejor_ruta = nx.shortest_path(
            self.grafo,
            source=inicio,
            target=fin,
            weight=self.calcular_peso
        )

        distancia_total = 0
        desnivel_total = 0
        tiempo_total = 0
        peso_total = 0
```

```
        print(f"\nMejor ruta de {inicio} a {fin}:")

        for i, nodo in enumerate(mejor_ruta):
            descripcion = self.grafo.nodes[nodo]["descripcion"]
            print(f"  Estación {nodo} - {descripcion}")
            if i < len(mejor_ruta) - 1:
                u, v = mejor_ruta[i], mejor_ruta[i + 1]
                datos = self.grafo[u][v]
                distancia_total += datos["distancia"]
                desnivel_total += datos["desnivel"]
                tiempo_total += datos["tiempo"]
                peso_total += self.calcular_peso(u, v, datos)
        print("\n")
        print(f" Resumen de ruta:")
        print(f"  Distancia total: {distancia_total:.2f} metros")
        print(f"  Subida total: {desnivel_total:.2f} metros")
        print(f"  Tiempo estimado caminando: {tiempo_total/60:.2f} minutos")

        self.dibujar_grafo(titulo="Mejor ruta resaltada", ruta_optima=mejor_ruta)

    except nx.NetworkXNoPath:
        print(f"No hay ruta entre {inicio} y {fin}.")
```

ALGORITMO DFS

```python
def mostrar_todas_las_rutas(self, inicio, fin):
    try:
        rutas = list(nx.all_simple_paths(self.grafo, source=inicio, target=fin))
        print(f"\nSe encontraron {len(rutas)} rutas posibles de {inicio} a {fin}:\n")

        for i, ruta in enumerate(rutas, start=1):
            distancia = 0
            desnivel = 0
            tiempo = 0

            print(f"Ruta {i}:")
            for i, nodo in enumerate(ruta):
                descripcion = self.grafo.nodes[nodo]["descripcion"]
                print(f"  Estación {nodo} - {descripcion}")
                if i < len(ruta) - 1:
                    u, v = ruta[i], ruta[i+1]
                    datos = self.grafo[u][v]
                    distancia += datos["distancia"]
                    desnivel += datos["desnivel"]
                    tiempo += datos["tiempo"]

            print(f"Distancia total: {distancia:.2f} m")
            print(f"subida total: {desnivel:.2f} m")
            print(f"Tiempo estimado: {tiempo/60:.2f} min\n")

            self.dibujar_grafo(titulo=f"Ruta {i} de {inicio} a {fin}", ruta_optima=ruta)

    except nx.NetworkXNoPath:
        print("No hay rutas posibles entre esos dos puntos.")
```

```python
def dibujar_grafo(self, titulo="Grafo completo", ruta_optima=None):
    pos = nx.spring_layout(self.grafo, seed=42)
    plt.figure(figsize=(8, 6))
    nx.draw(self.grafo, pos, with_labels=True, node_color='lightblue', node_size=800, edge_color='gray')
    nx.draw_networkx_labels(self.grafo, pos, labels={n: f"{n}" for n in self.grafo.nodes})
    if ruta_optima:
        path_edges = list(zip(ruta_optima, ruta_optima[1:]))
        nx.draw_networkx_edges(self.grafo, pos, edgelist=path_edges, edge_color='red', width=3)
        nx.draw_networkx_nodes(self.grafo, pos, nodelist=ruta_optima, node_color='red')
    plt.title(titulo)
    plt.axis('off')
    plt.show()
```

```python
class menu(RedSenderismo):
    def __init__(self, archivo_puntos, archivo_tramos):
        super().__init__(archivo_puntos,archivo_tramos)

    def mostrar_menu(self):
        print("Bienvenido al sistema de senderismo")
        print("1. Ver grafo completo")
        print("2. Calcular mejor ruta")
        print("3. Mostrar todas las rutas posibles")
        print("4. Salir")
        while True:
            opcion = input("Seleccione una opción: ")
            if opcion == "1":
                self.dibujar_grafo()
            elif opcion == "2":
                inicio = int(input("Ingrese estación de inicio: "))
                fin = int(input("Ingrese estación de destino: "))
                self.encontrar_mejor_ruta(inicio, fin)
            elif opcion == "3":
                inicio = int(input("Ingrese estación de inicio: "))
                fin = int(input("Ingrese estación de destino: "))
                self.mostrar_todas_las_rutas(inicio, fin)
            elif opcion == "4":
                print("Gracias por usar el programa")
                break
            else:
                print("Opción no válida")
```

```python
# +-------------------------------------------------------------------------+
# |                        V A R I A B L E S   G L O B A L E S               |
# +-------------------------------------------------------------------------+
archivo_puntos = "datos/puntos_interes.xlsx"
archivo_tramos = "datos/tramos.xlsx"

# ===========================================================================
# ||                                                                       ||
# ||       P R O G R A M A / F U N C I O N   P R I N C I P A L             ||
# ||                                                                       ||
# ===========================================================================
ruta= menu(archivo_puntos, archivo_tramos)
ruta.mostrar_menu()
```

5-OPERACIÓN Y MANTENIMIENTO

CORREGIR ERRORES
NO DETECTADOS
ANTERIORMENTE

MEJORAR LA IMPLEMENTACION
DE LAS UNIDADES DEL SISTEMA

¡MUCHAS GRACIAS!