Equipo1_Backend

Algoritmo de Rutas - Análisis de Algoritmos - D01. Por **Mateo Gonzalez.**

El siguiente módulo presenta una solución del Problema de Enrutamiento de Vehículos (VRP) basado en el algoritmo de Clarke & Wright, también llamado de Ahorros, haciendo uso de Python para su codificación.

Ejemplo

El uso de este módulo está diseñado para ser utilizado mediante la exportación de la función «algoritmo» y de la clase «NodoUbicacion» para su correcto funcionamiento.

import NodoUbicacion import algoritmo

rutas = algoritmo()

Este módulo sigue las guías presentadas por Python Enhancement Proposals aceptadas en su documentación versión 3.12.

Docstrings basados en la Guía de Estilo de Google.

class D01_Equipo1_Backend.**NodoUbicacion**(*peso: int*, *ubicacion: Ubicacion*, *anterior: object*, *siguiente: object*)

Bases: **object**

Clase tipo NODO con atributos específicos de una ubicación con pesos.

La visualización de las ubicaciones como NODOS permite la implementación de las rutas en forma de una estructura de lista enlazada, permitiendo una mayor información entre ubicaciones y próximos destinos.

peso
Un entero indicando el costo del paquete.
ubicacion
Coordenadas X e Y de la ubicación. Dos flotantes.
anterior
Nodo Anterior de la lista enlazada.
siguiente
Nodo Siguiente de la lista enlazada.

property **anterior**: *object*
Soy la propiedad “anterior”. Tengo un setter.
property **siguiente**: *object*
Soy la propiedad “siguiente”. Tengo un setter.
property **ubicacion**: *Ubicacion*
Soy la propiedad “ubicacion”. Tengo un setter.

D01_Equipo1_Backend.**algoritmo**(*peso_max_camion_1: int, peso_max_camion_2: int, *args: Ubicacion*) → list[tuple[Ruta, int]]

Algoritmo principal del código. Función exportable.

Esta función desarrolla el algoritmo de Clarke & Wright de paso a paso, creando las variables necesarias para lograrlo y utilizando las demás funciones cuando se necesiten.

Parámetros:

- **peso_max_camion_1** – Peso máximo soportado por el camión de la ruta 1.
- **peso_max_camion_2** – Peso máximo soportado por el camión de la ruta 2.
- ***args** – Todas las coordenadas de las ubicaciones a conectar en el problema, representadas como tuplas de dos flotantes (X e Y).

Devuelve:
Lista de dos tuplas con dos tipos de datos, la ruta obtenida y su peso.

D01_Equipo1_Backend.**añadir_a_ruta**(*ruta: Ruta, max_peso: int, peso: int, ubicacion: NodoUbicacion, posicion: bool*) → int

Adición de una ubicación a una ruta elegida.

Permite añadir una nueva ubicación a una ruta. Elige como almacén de la ruta a la primera ubicación a la que se le haya asignado. Esta adición solo puede realizarse como el primer destino o el último.

Parámetros:

- **ruta** – Ruta a la que se le va a añadir una ubicación.
- **max_peso** – Peso máximo soportado por el camión de la ruta.
- **peso** – Peso actual del camión.
- **ubicacion** – Ubicación representada en Nodo. Obtiene los atributos de anterior y siguiente en esta función.
- **posicion** – En donde se agregará la ubicación (Primero o Último).

Devuelve:
Peso actual de la ruta después del nuevo paquete.

Error

IndexError: Sucede cuando se agrega la primera ubicación, puesto que se busca un nodo siguiente a pesar de no contarlo.

`D01_Equipo1_Backend.distancia_euclidiana(u1: Ubicacion, u2: Ubicacion) → float`

Función aritmética para obtener la distancia euclidiana.

Regresa la distancia obtenida de las dos ubicaciones.

Parámetros:

- **u1** – Ubicación 1. En forma de una tupla de dos flotantes.
- **u2** – Ubicación 2. En forma de una tupla de dos flotantes.

Devuelve:
Flotante con la distancia euclidiana obtenida.

`D01_Equipo1_Backend.matriz(ubicaciones: tuple) → list`

Creación de una matriz de adyacencia.

Crea una matriz de adyacencia automáticamente a partir de las coordenadas de cada ubicación y la función «distancia_euclidiana».

Parámetros:
ubicaciones – Tupla de ubicaciones (en forma de tupla de dos flotantes).

Devuelve:
Matriz de adyacencia de distancias en forma de una lista anidada.

```
D01_Equipo1_Backend.seleccion_ruta(ruta_1: Ruta, ruta_2: Ruta, ubicaciones: list[NodoUbicacion], posibles_rutas: tuple[int, int], peso_max_1: int, peso_max_2: int, peso_1: int, peso_2: int) → tuple[int, int] | None
```

Selección de la ruta a la que se le añade la nueva ubicación.

Permite la selección de la ruta (1 o 2) a la que se le añadirá una ubicación nueva. Se realiza por distintos parámetros como saber si ambas rutas pueden realizar la adición o si está vacía. Utiliza de la función «añadir_a_ruta» para añadir la ruta y retornar el resultado obtenido.

Parámetros:

- **ruta_1** – Ruta 1 a la que se le puede añadir una ubicación.
- **ruta_2** – Ruta 2 a la que se le puede añadir una ubicación.
- **ubicaciones** – Lista de todas las ubicaciones en forma de Nodos.
- **posibles_rutas** – Índices de las posibles nuevas ubicaciones. Conforme a la lista de ubicación (en Nodos) completa.
- **peso_max_1** – Peso máximo soportado por el camión de la ruta 1.
- **peso_max_2** – Peso máximo soportado por el camión de la ruta 2.
- **peso_1** – Peso actual del camión de la ruta 1.
- **peso_2** – Peso actual del camión de la ruta 2.

Devuelve:
Peso actual de ambas rutas, en enteros. Se regresa un valor nulo si la adición no es posible.

Tipo del valor devuelto:
tuple[int, int]

D01_Equipo1_Backend.**tabla_de_ahorros**(*distancias: list*) → dict[tuple[int, int], float]

Creación de una tabla de ahorros.

Crea una tabla de lo que se ahorra por ir de ubicación a ubicación comparado con la distancia almacén a ubicación.

Parámetros:
distancias – Matriz de adyacencia de distancias en forma de lista.

Devuelve:
Diccionario de los índices de las ubicaciones y el ahorro obtenido.

D01_Equipo1_Backend.**ubicaciones_a_nodos**(*ubicaciones: tuple*) → list[NodoUbicacion]

Cambio del tipo de dato de las ubicaciones.

Cambia las ubicaciones (representadas en tuplas de dos flotantes) a objetos de la clase NodoUbicacion.

Parámetros:
ubicaciones – Tupla de ubicaciones (en forma de tupla de dos flotantes).

Devuelve:
Lista de todas las ubicaciones en forma de Nodos.