



Buscador de rutas para el metro de Kiev mediante algoritmo A*

Grupo 18

Alumnos: Pedro Henrique Nobre Oliveira

UPM, Escuela Superior Técnica de Ingenieros Informáticos



Problema

Diseñar y desarrollar una aplicación para hallar el trayecto óptimo entre dos estaciones del metro de Kiev usando el algoritmo A^* y siguiendo una heurística determinada.

Fases del trabajo

Recolección datos

Obtener datos

¿Obtención gratuita y fiel de datos?

¿Formato?

Determinar heurística

Poca complejidad y alto rendimiento

¿Qué se suele hacer?

¿Cómo podemos mejorarlo?

Implementación

Implementación de la práctica

- ¿Lenguaje?
- Datos a objeto
- Algoritmo
- Interfaz gráfica
- Ejecutable



Recolección datos

Fuente de datos: Google Maps

Formato: json

Forma de obtención: manual

```
{
  "subway_stations": {
    "Akademmistechko": {
      "coordinates": [50.4666694, 30.3785173],
      "adjacent_stations": ["Zhytomyrska"]
    },
    "Zhytomyrska": {
      "coordinates": [50.4557332, 30.3628221],
      "adjacent_stations": ["Akademmistechko", "Sviatoshyn"]
    }
  }
}
```

(...)

```
},
"Chervoniy khutir": {
  "coordinates": [50.4088889, 30.694444444444446],
  "adjacent_stations": ["Boryspilska"]
},
"lines": {
  "green": ["Syrets", "Dorohozhychi", "Lukianivska", "Zoloti vorota", "Palats sportu", "Kontrakt"],
  "blue": ["Heroiv Dnipra", "Minska", "Obolon", "Petrivka", "Tarasa Shevchenka", "Kontrakt"],
  "red": ["Akademmistechko", "Zhytomyrska", "Sviatoshyn", "Nyvky", "Beresteiska", "Kontrakt"]
}
}
```

A*

- ¿Cómo mejorar la *performance* de Dijkstra?



Usar heurística

$$f(n) = g(n) + h(n)$$



Heurística

- Distancia euclidiana
- Coste de transbordo



UNIVERSIDAD
POLITÉCNICA
DE MADRID

Implementación

Python 3.8.0

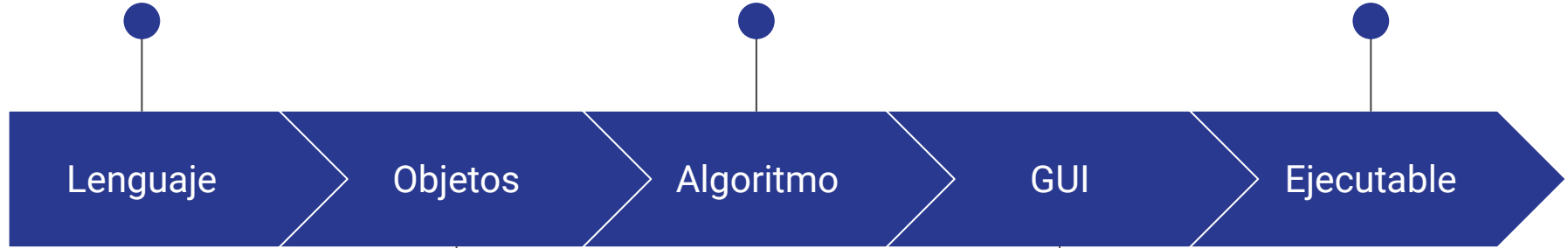
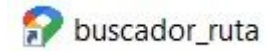
```
class Graph:
    def __init__(self, adjacent_list: dict, stations_dict: dict, lines: dict):
        self.adjacent_list = adjacent_list # Diccionario con distancia entre nodos adyacentes
        self.stations_dict = stations_dict # Diccionario con estaciones y sus coordenadas
        self.lines = lines

    def a_star_search(self, origin, destination) → list:
        ...

    def h(self, node: str, destination: str) → float:
        """
        Estimar distancia según una heurística consistente en calcular la distancia entre
        el nodo actual y el nodo destino y sumar un valor de transbordo si los nodos
        son de líneas distintas
        """
```



UNIVERSIDAD
POLITÉCNICA
DE MADRID



GeoPy

```
def get_adjacent_list() → (dict, dict, dict):
    """
    Poner datos en el formato adecuado
    Devuelve:
    La lista de adyacencias con las distancias entre nodos: dict
    Las estaciones con sus coordenadas: dict
    Las líneas y sus estaciones: dict
    """

    return adj_list, stations, lines
```



Ejecución



Route Finder

Origen

Detino

Calcular ruta

Resultado:

Es necesario seleccionar una estación

Salir

Figura 1: Caso error.

Route Finder

Origen

Akademmistechko

Detino

Osokorky

Calcular ruta

Resultado:

Akademmistechko - Zhytomyrska - Sviatoshyn - Nyvky - Beresteiska - Shuliavska - Politekhnichnyi instytut - Vokzalna - Universytet - Teatralna - Zoloti vorota - Palats sportu - Klovsk - Pecherska - Druzhby narovib - Vyduychi - Slavutych - Osokorky -

Salir

Figura 1: Caso éxito..

Conclusión



UNIVERSIDAD
POLITÉCNICA
DE MADRID