Algoritmos II -

Proyecto Semestral

Uber

Integrantes:

Franco Yudica
Martín Farrés

Repositorio: [MartinFarres/Uber-Project (github.com)]

Estructuras de datos

Mapa

- Se encuentra estructurado como una clase, brindando información útil del mismo y fácil acceso a funciones y operaciones relacionadas con el mismo.

- Los vértices y sus adyacentes están representados como una *Hash Table* de *Hash Table*, es una implementación similar a la conocida lista de adyacencia, pero con Hash Table.

- Por cada par de vértices adyacentes almacenamos la distancia entre ambos.

Direcciones

- Simple estructura que almacena las esquinas $< e_1, e_2 >$ y sus distancias

```python
class Direction:
    def __init__(self, edge1, d1, edge2, d2):
        self.edge1 = edge1
        self.edge2 = edge2
        self.d1 = d1
        self.d2 = d2
```

Autos

- Se encuentra estructurado como una tabla hash, donde se guarda el objeto dirección dinámica(nombreAuto, dirección, tarifa).

```
cars_HT[name] = DynamicLoc(name, direction, price)
```

Personas

- Se encuentra estructurado como una tabla hash, donde se guarda el objeto dirección dinámica(nombrePersona, dirección, monto).

```
people_HT[name] = DynamicLoc(name, direction, price)
```

Direcciones Autos

- Hash Table ($< e_1, e_2 >$, $[auto_1, auto_2, \dots]$) utilizada para poder obtener rápidamente los autos que se encuentran en una calle.

  Supongamos que tenemos la calle $< e_1, e_2 >$, y queremos saber cuales son los autos que se encuentran en la misma, simplemente podríamos acceder de la siguiente manera:

```
e1 = 1
e2 = 2
cars_in_street = cars_dir_HT[(e1, e2)]
```

  (Nótese que solamente retorna los autos que se encuentran el la calle que va desde e_1 a e_2, pues es un grafo dirigido)

Data

- Es una clase que posee a todos las estructuras anteriores dentro. Se utiliza como un objeto de transición entre los datos serializados y los datos a usar dentro del programa, y viceversa.

Estado Actual

- Se encuentra implementadas todas las funciones de carga de datos;

```python
def load_fix_element(name, direction) -> bool:
    if name in static_loc_HT:
        return False

    static_loc_HT[name] = StaticLoc(name, direction)
    return True
```

```python
def load_movil_element(name, direction, price) -> bool:
    """
    """
    if name[0].upper() == "C":
        if name in cars_HT:
            return False
        cars_HT[name] = DynamicLoc(name, direction, price)

    if name[0].upper() == "P":
        if name in people_HT:
            return False
        people_HT[name] = DynamicLoc(name, direction, price)
    return True

```

- Además, ya están creadas las funciones de serialización y deserialización con pickle, al igual que, la funciones para extraer las variables necesarias para la creación del mapa.

```python
def initialization_data():
    data = Data()
    dir = os.path.abspath(os.getcwd()) + "\saveData.txt"
    if os.path.exists(dir):
        with open(dir, "br") as f:
            data = pickle.load(f)
    return data

```

```python
def read_map_var(path: str):
    mapV = []
    if os.path.exists(path):
        with open(path, "r") as f:
            lines = f.readlines()
            for line in lines:
                line = line.replace(" ", "")
                if mapV == []:
                    mapV.append(line[3:-2].split(","))
                else:
                    mapV.append(line[4:-2].split(">,<"))
    for i in range(0, len(mapV[1])):
        mapV[1][i] = mapV[1][i].split(",")
    return mapV
```

- Además, implementamos la estructura de datos *MinHeap* la cuál cumple un rol muy importante en los algoritmos de búsqueda, especialmente en la función que busca el camino más corto usando el algoritmo de Dijkstra.

- Contamos con una *primera iteración del algoritmo de Dijkstra* aplicado específicamente a este proyecto. Sin embargo hay que hacer algunos ajustes para casos especiales.

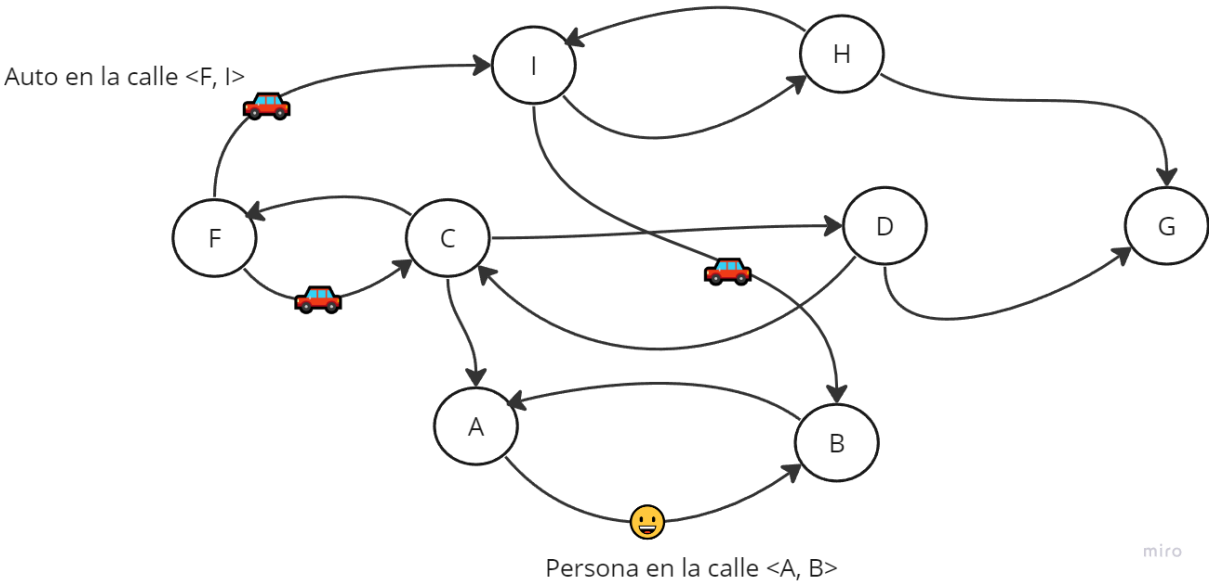
Plan

Tareas a realizar;

- Interfaz de consola a través de la cuál el usuario ingresa los datos (UI),
- Proceso de traslación de ubicaciones dinámicas / móviles
- Camino más corto entre dos direcciones (casi finalizado)
- Encontrar los autos más cercanos.

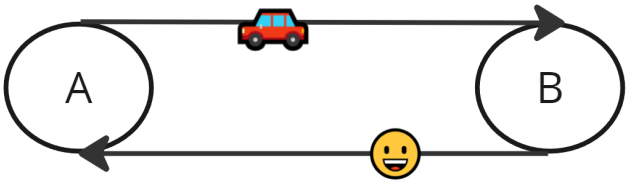
Plan para realizar la funciones de búsqueda de los autos más cercanos:
[UberProject, Online Whiteboard for Visual Collaboration (miro.com)]

Caso Recursivo (Dijkstra)



Caso Base

Auto en la calle <A, B>



Persona en la calle <A, B>

Visualizacion en arbol (Teorico)

Construido cuando se agregan las calles