Uber.py

Enzo Palau y Paula Martinez

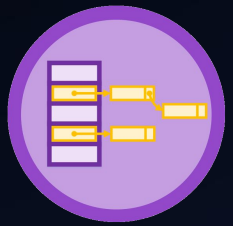
https://github.com/Paulonia14/proyectofinal

Problemas encontrados

- Cargar argumentos
- Carga de elementos y su guardado en memoria
- Cálculo del viaje
  - Distancias a los autos
  - Cálculo de las direcciones intermedias
- Y por último, algunos bugs…

Carga de elementos y su guardado en memoria

- Tanto para fix y movil elements…
  - Uso de tablas Hash!
  - Diccionarios de python {}



  - Ejemplo: Key:C1 ➡ Value: ([1,2,2,1] , 1500)

- Implementamos pickle!

Cálculo del viaje

- Carga de mapa —> Uso de matriz de adyacencia
  - Traducir Mapa
    - Uso de eval()
  - Cálculo de Distancia
    - ¡Junto a la carga del mapa!
      Pero, ¿Cómo?...

FLOYD-WARSHALL ALGORITHM

La ejecución del algoritmo encontrará las longitudes de los caminos más cortos entre todos los pares de vértices de un grafo ponderado.

Uso de programación dinámica con matrices

Mejora paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la distancia es la óptima.

Rendimiento en el peor de los casos	$\Theta(\|V\|^3)$
Rendimiento en el mejor de los casos	$\Theta(\|V\|^3)$
Rendimiento medio	$\Theta(\|V\|^3)$
Complejidad espacial en el peor de los casos	$\Theta(\|V\|^2)$

Funcionamiento:

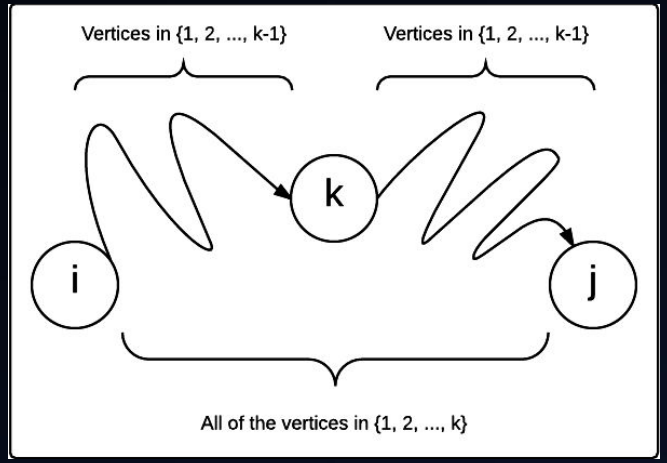
El primer paso es llenar una matriz de la siguiente manera:

$$W_{ij} = \begin{cases} 0 & if\ i = k \\ w_{ij} & if\ such\ edge\ exists. \\ \infty & otherwise \end{cases}$$

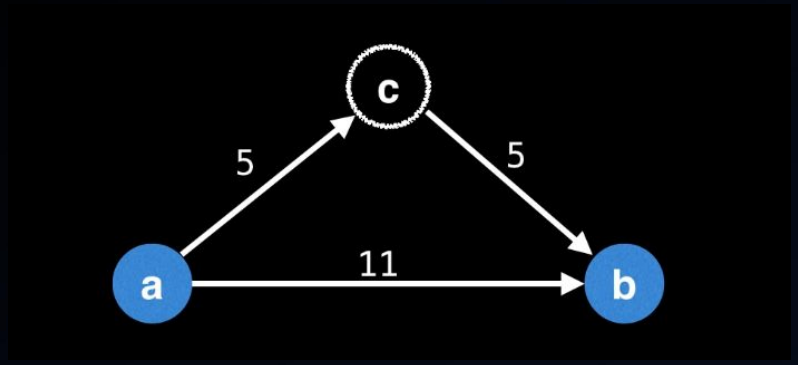
Luego pasaremos a la ejecución de el algoritmo completando de la siguiente manera:

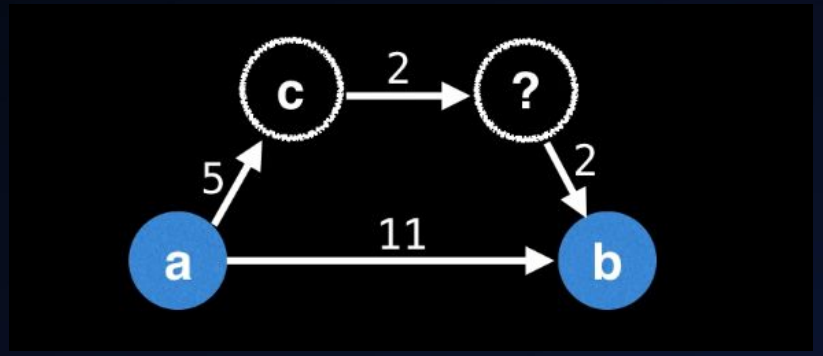
$$\text{Ruta más corta}(i, j, k) = \min(\text{Ruta más corta}(i, j, k-1), \text{Ruta más corta}(i, k, k-1) + \text{Ruta más corta}(k, j, k-1)).$$

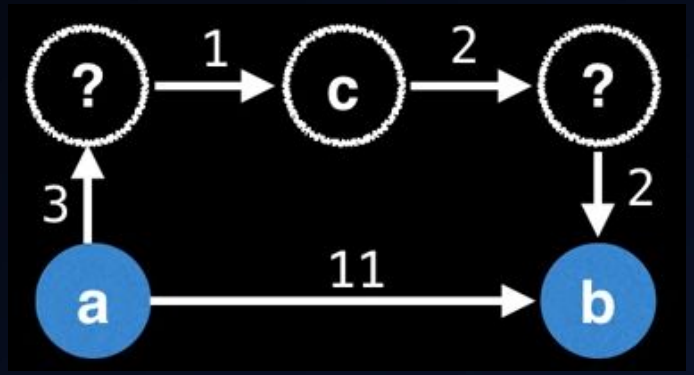
Básicamente, la configuración de esta función pregunta esto: ¿Es el vértice k un intermedio de nuestro camino más corto (cualquier vértice en el camino además del primero o el último)?



Por ejemplo……


c
5
5
a
11
b


c
2
?
5
2
a
11
b


?
1
c
2
?
3
2
a
11
b

Matriz de Recorridos

Es posible reconstruir las rutas con modificaciones simples al algoritmo

El primer paso es crear una matriz (de la misma dimensión que la anterior) en la cual si por ejemplo entre (i,j) hay incidencia entonces colocamos i

Luego cada vez que actualicemos la matriz de Floyd-Warshall, en esa misma ubicación colocaremos el vértice intermedio K

Por ejemplo:

	0	1	2	3
0	-	0	0	1
1	2	-	1	1
2	2	0	-	3
3	0	0	0	-

si queremos llegar de 2 a 1….

2 0 1

Direcciones Intermedias

(Un dolor de cabeza)

- Sacamos el sentido de las calles
- Calculamos los “pedacitos” de la arista inicial
- Primero verificamos si están en la misma calle y calculamos la distancia entre ellos (o si se tiene que ir por otro lado)
- Finalmente calculamos los que no están en la misma calle, dividiéndolos en **cuatro** casos

Posibles mejoras de la solución

- Exceso de condicionales (IF) en el cálculo de las direcciones intermedias….
  - ¿Utilizar alguna estrategia de programación ?

Conclusión

I. Organización:
   - División de Tareas
   - Trabajo compartiendo pantalla

II. ¡Aprendiendo Python!

¡Muchas Gracias!