#1 寻找两点之间的最短路

母方法：——path.py

`find_shortest_path(G,from_node_id,to_node_id,mode='all',seq_type='node')`

输入起始点与目标点的 id，输出起始点与目标点之间的最短路.

G——属于 network 类

mode——settings.yml 中定义的目标运输模式，可以是 agent_type 或 name，如'w'或'walk'，默认是'all'，即所有的 link 都对所有运输模式开放.

seq_type——'node'或'link'，选择按 node 还是 link 顺序排列输出的内容.

计算两点之间最短路径由以下逻辑实现：——path.py

`single_source_shortest_path(G, from_node_id, engine_type='c', sp_algm='deque')`

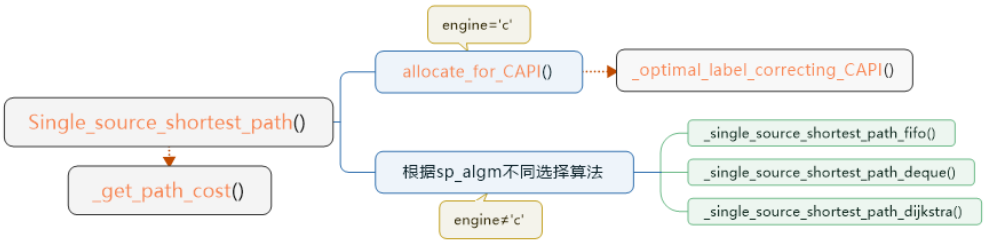
engine_type——指定求解引擎，可选 C++，Python.

sp_algm——指定最短路算法类型，可选 deque，fifo，Dijkstra.

输入起始点，调用 C++程序利用 deque 方法计算该点出发到所有可达点的最短距离.

`_get_path_cost(G, to_node_id)`

在上一步计算完成的基础上，输入目标点，输出起始点到目标点的最短路.



G.allocate_for_CAPI()——classes.py

作用：将现有的数据进行汇总与格式转换，为下一步传入 C++程序做准备.

`_optimal_label_correcting_CAPI(G, origin_node_no)`

调用 C++程序计算最短路.

`output_path_sequence(G, to_node_id, type='node')`

选择将最短路按照所经过的 node 或 link 输出.

核心算法复习：

Label-correcting Algorithm——

while some arc (i, j) satisfies $d(j) > d(i) + c_{ij}$ **do**
begin
$d(j) := d(i) + c_{ij}$;
pred$(j) := i$;

FIFO 实现——引入 SE-List，并将违规弧(i,j)的目标点 j 添加到 List 中，按照先进先出的顺序以最短路条件扫描从对应点出发的所有弧.

Dequeue 实现——若 j 曾在 List 中出现过，则添加到 List 最前面，否则添加到最后面.

Dijkstra Algorithm——设起点标号为 0，其余点为∞:将标号最小的点转为永久标号，

用$min(d(i) + C_{ij}, d(j))$更新其余点;重复，直至所有点都有永久标号.