1.内容总结

1.1 定义类（只列举部分变量和方法）

1.1.1Assignment

重点变量：agent_types, demand_periods, demands, network, spnetworks, memory_blocks,
重点方法：

setup_spnetwork(self)——设置 spnetwork，具体作用不详

_convert_mode(self, mode)——使输入的 agent type 无论是 id 还是 name 都可以被识别

find_shortest_path(self, from_node_id, to_node_id, mode, seq_type='node')——根据输入的 mode 设置 agent type，调用 find_shortest_path 函数

1.1.2Network

重点变量：node_list, link_list, agent_list, zones
重点方法：

allocate_for_CAPI (self)——作用不详

1.1.3Node

重点变量：seq_no, id, outgoing_link_list, incoming_link_list, zone_id, x_coord, y_coord

1.1.4Link

重点变量：id, seq_no, from_node, to_node, length, free_speed

1.1.5 其他类

Agenttype, DemandPeriod, Demand, VDFPeriod, SPNetwork, UI

1.2 定义方法

1）_update_orig_zone(oz_id)和_update_dest_zone(dz_id)——用_zone_degrees 表示 zone 的连接性，0 表示无入无出，1 表示有出无入，2 表示有入无出， 3 表示有入有出
2）read_network:

read_settings(input_dir, assignment)——将 settings 文件中的参数添加到 assignment 中
read_nodes——读取 node.csv 文件中的参数
read_links——读取 link.csv 文件中的参数

3）find_shortest_path(G, from_node_id, to_node_id, seq_type='node'):

_single_source_shortest_path：三种寻找单源最短路的方法，FIFO，deque 和 dijkstra，三种方式的不同主要体现在两点：一是对 status 的设置，二是 SEList 的存储方式

FIFO：

status：初始将所有点的 status 设置为 0，当点被添加到 SEList 中时 status 变为 1，防止点被重复添加

SEList 的存储方式为 list

deque：

status：初始将所有点的 status 设置为 0，当有点被作为标签更新的源点使用时将该点的 status 改为 2，当有点被添加到 SEList 中时进行判断：0→将该点添加到队尾最后使用；1→该点已在 SEList 中，不需要重复添加；2→该点被作为标签更新源点使用过，添加到队首首先使用

SEList 的存储方式为 deque

dijstra：

status：初始将所有点的 status 设置为 0，当有点被作为标签更新的源点使用时将该点的 status 改为 1，在寻找被更新的点时：0→该点没有被作为标签更新源点使用过，添加到堆中，由于堆有排序的功能，无需在意位置；1→该点被作为标签更新源点使用过，无需再次判断（dijstra 算法默认无负弧）

SEList 的存储方式为二叉树

_get_path_cost(G, to_node_id)：获得路径的 cost

output_path_sequence(G, to_node_id, type='node')：获得具体的路径

2.问题总结

1) 所有关于 Spnetworks 的定义和方法的作用不详，在 find_shortest_path 中暂时没有用到，后续继续关注

2）Assignment 类中的 memory_blocks 作用不详，在 find_shortest_path 中暂时没有用到，后续继续关注

3）Network 类中 allocate_for_CAPI（）无返回值，用法不明：

```
def allocate_for_CAPI(self):
    # execute only on the first call
    if self.has_capi_allocated:
        return
```