Dijkstra Algorithm

@author: 朱虹翱，徐文欣
@description: dijkstra1.py 和 dijkstra2.py 应用 Dijkstra Algorithm 算法 @division: 朱虹翱完成环境、节点、路由、包裹等类的定义；徐文欣在此基础上应用 Dijkstra Algorithm 算法，完成 dijkstra1.py 和 dijkstra2.py。其中 dijkstra1 中根据提供的边缘信息和节点信息生成环境，计算不同包裹的时间，dijkstra2.py 中提供对于路径距离及选择的计算。

一、基础类定义：

Node: 表示物流环境中的节点，包括节点 ID、位置、吞吐量、延迟、成本等属性。

- id: 节点 ID，唯一标识一个节点。
- pos: 节点在环境中的位置坐标。
- throughput: 节点的吞吐量，表示节点能够处理包裹的能力。
- delay: 节点的延迟，表示包裹在节点处停留的时间。
- cost: 节点的成本，表示包裹在节点处停留的成本。
- is_station: 是否是 station
- buffer: 仓库，用来存放入站的包裹，用堆存储
- packages: 运输线，用来处理包裹，用堆存储
- dones: 完成仓库，用来存放到达终点的包裹，用列表存储
- history: 收到的包裹，用列表存储
- package_order: 最新处理的包裹 index（用于给堆排序，并不总是等于处理的包裹的总数）
- reset(): 重置节点状态，清空仓库和运输线上的包裹，清空历史记录
- add_package(package): 如果是包裹的终点，加入 done，否则加入 buffer；如果 buffer 中有其他包裹，判断新包裹的类型是否与 buffer 顶端的包裹类型一致（一致则插入到最底端，否则对于 express 包裹，插入到最顶端）。更新包裹的 delay 为当前节点的 delay
- process_packages(): 自动处理包裹，把 buffer 顶端的包裹输入到 packages 中处理
- remove_package(): 自动从 packages 中移除 delay<=0，即可以转移的包裹

Route: 表示物流环境中的路由，包括路由 ID、源节点、目的节点、延迟、成本等属性。

- src: 路由的源节点 ID。
- dst: 路由的目的节点 ID。
- id: 路由 ID，唯一标识一个路由。f"{src}- >{dst}"
- time: 路由的延迟，表示包裹在路由上传输的时间。
- cost: 路由的成本，表示包裹在路由上传输的成本。
- package_order: 最新处理的包裹 index（用于给堆排序，并不总是等于处理的包裹的总数）
- packages: 正在运输的包裹，用堆存储
- history: 收到的包裹，用列表存储
- reset(): 重置路由状态，清空运输线上的包裹，清空历史记录

- add_package(package): 将包裹加入到 packages 中，更新包裹的 delay 为当前路由的 time
- remove_package(): 自动从 packages 中移除 delay<=0，即可以转移的包裹

Package: 表示物流环境中的包裹，包括包裹 ID、延迟、源节点、当前节点、目的节点、路径、完成状态等属性。

- id: 包裹 ID，唯一标识一个包裹
- time_created: 创建时间
- time_arrived: 到达时间
- src: 起点
- dst: 终点
- category: 1 for 'Express' and 0 for 'Standard'
- history: 过去的状态及时间戳
- path: 规划的路线（当前节点不一定位于开头）
- delay: 允许状态转移的倒计时
- done: 是否到达终点

二、高级类定义（）：

通过使用 Dirlstra 算法模拟包裹配送系统。以下是对代码结构和组成部分的简要说明:

Location Class：

- Location Class 表示配送网络中的一个位置。
- 每个位置都有名称、距离、前一个位置和路由列表。

Initializing the Locations：

- 代码初始化了一个名为 "station_pos" 的 Location 实例列表。
- 每个位置被分配初始距离，并设置了起始位置和结束位置。

Dirkstra 算法：

- 函数 "dijkstra" 实现了 Dirkstra 算法来寻找最短路径。
- 它接受位置列表、起始位置和结束位置作为输入。
- 算法通过迭代选择距离最小的位置，并更新其邻居的距离。
- 最短路径通过从结束位置追溯到起始位置构建。

Helper Functions：

- 代码提供了辅助函数 "get_location_by_id" 和 "get_package_by_id"，用于根据 ID 获取位置和包裹对象。

Simulation:

- "simulate" 函数是主要的模拟函数。
- 它接受位置、路由和包裹的列表作为输入。
- 它调用 "dijkstra" 函数计算每个包裹的最短路径。
- 它模拟包裹配送过程，更新包裹的日志，记录每个位置的到达、处理和发送等事件。

- 记录每个包裹的配送时间，并根据配送时间对包裹进行排序。
- 最后，函数打印每个包裹的详细信息和日志。

Main Execution:

- "if **name** == '**main**':" 块是代码的入口点。
- 它生成模拟所需的数据，例如 "station_pos"、"station_prop"、"center_pos"、"center_prop"、"edges" 和 "packets"。
- 然后将数据传递给 "simulate" 函数来运行模拟。

使用这段代码，需要提供模拟所需的数据，例如位置、路由和包裹等。