Djkstra算法原理

目的：求解在强连通有向赋权图中，从任意起始点出发到其他所有节点的最短路径

输入参数：有向图结构参数，起始点，

重要参数（影响求解效率的重要因素）：每次寻找距离dv最小的点的策略，如果遍历其他known表为未知的节点，那么会耗费相当多的时间。一个方法就是采用优先队列，将known表中值为未知的节点按照优先队列方式取出来，但是这又有一个问题，优先队列不支持find操作，因为取出来节点后，要对这个节点相邻的known表中值为未知的节点进行距离判断更新，那么这意味着要对优先队列中的部分节点需要更新距离并更新优先队列的空间结构，而这个优先队列恰是根据距离dv来生成和调整的，但是很遗憾优先队列不支持高效的find操作。

前期准备：起始点，有向图结构（采用邻结表或者邻结矩阵）

算法原理：

**1：初始化：**

输入起始点、采用邻结表或者邻结矩阵记录的有向图结构。

算法运行过程中应当维护的三个表：

1：记录节点是否已经访问的列表（is\_visited\_list）。

2：记录从起始点到各节点的最短距离的列表（distance\_visited\_list）。

3：记录对于所有节点而言，最短路径中连接当前节点的上级节点ID（best\_path\_pre\_node\_list）。

最终算法输出的也是这三个表，distance\_visited\_list和best\_path\_pre\_node\_list分别记录了从初始点出发到这个有向赋权图中其他任一点的最短路径（可以对best\_path\_pre\_node\_list进行反向追溯得到完整最短路径）和最短路径的距离。

初始化过程中:

1、对于is\_visited\_list，所有节点全设为false。

2、对于distance\_visited\_list，除了起始点v0对应的位置应设为0以外，其他所有节点全设为无穷（python 中用sys.maxsize代替）

3、对于best\_path\_pre\_node\_list，所有节点的对应位置全部设置为-1。

**2、维护一个过程列表node\_container(存放**is\_visited\_list值为false并且distance\_visited\_list不为无穷大的备选访问节点**),将初始节点ID加入node\_container。**

3、如果node\_container不为空，那么重复以下步骤：

3.1 从node\_container查找dv值最小的节点ID，并将其弹出

3.2 将这个节点（设为node\_base）在is\_visited\_list中对应的值设为True

3.3 对于这个节点node\_base相邻的并且在is\_visited\_list对应位置为false的全部节点，执行如下步骤：

3.3.1 查找node\_base在distance\_visited\_list中的距离dv1，相邻节点跟node\_base的距离dv2,

3.3.2 计算dv1 + dv2 是否小于相邻节点在distance\_visited\_list中的距离，**如果小于**则更新这个对应位置的值为dv1 + dv2并且更新best\_path\_pre\_node\_list对应位置的值为node\_base的ID，然后如果该相邻节点的ID没在**node\_container中就**放进去就行**。**