INT102W7_floyd 算法

定义：**floyd** 算法又叫插点法，是一种求取多源最短路径的算法，通常用于稠密图（稠密图定义见 **prim** 算法）

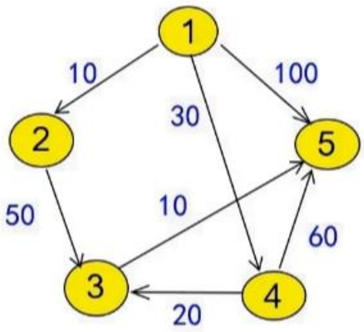
应用场景：**floyd** 算法性质上仍然是一种资源规划问题，在用途上还是求取最短路径，通常可以用在交通规划，网络路由等问题上

达成过程：该算法首先仍然是初始化非原点的距离为无穷然后更新，但是区别在于由于其多源的性质，我们是使用矩阵来存储数据。然后我们就根据矩阵（通常横坐标为结束点纵坐标为起始点）进行迭代，具体流程是计算 **Dik+Dij** 的值并与 **Dij** 进行比较，若是更小则更新 **Dij** 的值，此处 **i** 为出发点，**k** 为当前迭代次数对应的点，**j** 为结束点。通常对应 **n** 个数据我们会进行 **n** 次迭代，最后就可以得到对应当前纵坐标的点，我们的横坐标的最短路径是多少

存储方式：通常采用矩阵进行存储，在输入的时候也可以视作是采用二维数组进行输入，值得注意的是他需要的是两个矩阵，一个用来存储最短距离，通常标记为 **A**，另外一个则是用来记录对应

的中转点 k

例题：如图所示，求取每个点对应到其他点的最短距离，并且得到对应路径



A=

	1	2	3	4	5
1	0	10	∞	30	100
2	∞	0	50	∞	∞
3	∞	∞	0	∞	10
4	∞	∞	20	0	60
5	∞	∞	∞	∞	0

P=

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

对于这个题来说我们就可以首先初始化我们的矩阵，将现有的所有边对应植入到矩阵中，然后初始化我们的 p 矩阵为不相关的点表示从原点直接到达，并不一定是 0，任何你喜欢的数都可以，然后，我们就对应迭代顺序，从 1 开始迭代，对应 1 的时候，由于没有其他中转点，所以数据不变，对应数据为 2 的时候，把所有的值都写成 Di2 +D2j，得到对应的值后与表中对比，如果更小就会进行更新，在更新完成后就把 P 矩阵中对应的元素修改为 2 重复如上操作到 n=5，也就是我们目前的元素个数，就可以得到对应的最短路径了。然后由个例到全部，我们可以得到对于这类算法的模板就是首先建立二维数组存储矩阵，然后初始化矩阵的值，对应每一次迭代进行一次数据的更新，最后就可以得到路径

的长度。

他的伪代码如下

```
let V = number of vertices in graph
let dist = V × V array of minimum distances initialized to ∞
for each vertex v
  dist [v][v] ← 0
for each edge (u,v)
  dist [u][v] ← weight(u,v)
for k from 1 to V
  for i from 1 to V
    for j from 1 to V
      if dist [i][j] > dist [i][k] + dist [k][j]
        dist [i][j] ← dist [i][k] + dist [k][j]
      end if
```

由上图中代码我们可知，我们首先是建立一个 **v** 来存储我们的点，然后建立一个叫做 **dist** 的二维数组来存储我们的矩阵，然后，先把所有的距离设计为无穷，初始化我们的 **dist** 对角线为 **0**（因为数据本身到自己的位置肯定是 **0**），在这之后，我们将其余的点对应跟新为他的权值，在这之后我们几个开始我们的对比，因为我们要首先遍历所有到达点，所以 **j** 在最内层，因为我们在遍历到达点后还需要对应其他点寻找到达点（为了寻找多源最短路径），所以我们会把 **i** 放到第二层，最外层则是我们的迭代次数，所以放到最外层，然后我们看伪代码的核心部分，对应所有的点 **j**，如果说初始点到他的距离大于初始点对应中转点到达他的距离，那么我们执行一次替换。最终我们就可以得到他的最短路径长度