小世界现象

1 题目描述

小世界现象（又称小世界效应），也称六度分隔理论（英文：Six Degrees of Separation）。假设世界上所有互不相识的人只需要很少中间人就能建立起联系。后来 1967 年哈佛大学的心理学教授斯坦利·米尔格拉姆根据这概念做过一次连锁信实验，尝试证明平均只需要 5 个中间人就可以联系任何两个互不相识的美国人。NowCoder 最近获得了社交网站 Footbook 的好友关系资料，请你帮忙分析一下某两个用户之间至少需要几个中间人才能建立联系？

1.1 输入描述：

输入第一行是一个整数 t，表示紧接着有 t 组数据。每组数据包含两部分：第一部分是好友关系资料；第二部分是待分析的用户数据。

好友资料部分第一行包含一个整数 n(5≤n≤50)，表示有 n 个用户，用户 id 用 1->n 表示。紧接着是一个只包含 0 和 1 的 n×n 矩阵，其中第 y 行第 x 列的值表示 id 是 y 的用户是否是 id 为 x 的用户的好友（1 代表是，0 代表不是）。假设好友关系是相互的，即 A 是 B 的好友意味着 B 也是 A 的好友。

待分析的用户数据第一行包含一个整数 m，紧接着有 m 行用户组数据。每组有两个用户 ID，A 和 B(1≤A，B≤n，A!=B)。

1.2 输出描述：

对于每组待分析的用户，输出用户 A 至少需要通过几个中间人才能认识用户 B。如果 A 无论如何也无法认识 B，输出“Sorry”。

1.3 输入例子：

```
2
5
1 0 1 0 1
0 1 1 1 0
1 1 1 0 0
0 1 0 1 0
1 0 0 0 1
3
1 2
```

```
2 4
3 5
6
1 1 0 0 1 0
1 1 0 1 0 1
0 0 1 0 0 1
0 1 0 1 0 1
1 0 0 0 1 0
0 1 1 1 0 1
4
2 3
3 6
5 1
4 2
```

1.4 输出例子：

```
1
0
1
1
0
2
1
```

2 解题思路

题目要求某两个人之间最少通过多少个中间人才能建立联系，人与人之间的关系用一个图进行表示，有直接关系的使用 1 表示，没有关系的使用 0 表示。可以对这个关系矩阵进行改进，将自身与身的关系计为 1，<v,w>存在直接关系记为 1，不存在直接关系的记为+∞。要求<x,y>最少通过多少个中间人可以取得联系，可以先计算<x,y>之间的最短路径，因为边的权权重都是 1，所以最短路径就是<x,y>所经过的最少的边的数目 e，而<x,y>最少的联系人数目就是<x,y>最少边所在线段中间的顶点数，即 e-1。

经过分析可以得，该题可以通过 Dijkstra、Bellman-Ford 或者 F loyd 算法进行处理。本题分析过程讲解 Floyd。Dijkstra 方法见 **016-回家过年**算法实现。

2.1 Floyd 算法

问题的提出：已知一个有向网（或无向网），对每一对顶点 $v_i \neq v_j$，要求求出 v_i 与 v_j 之间的最短路径和最短路径长度。

解决该问题的方法有：

1）轮流以每个顶点为源点，重复执行 Dijkstra 算法（或 Bellman-Ford 算法）n 次，就可求出每一对顶点之间的最短路径和最短路径长度，总的时间复杂度是 $O(n^3)$（或 $O(n^2+ne)$）。

2）采用 Floyd（弗洛伊德）算法。Floyd 算法的时间复杂度也是 $O(n^3)$，但 Floyd 算法形式更直接。

2.2 算法思想

Floyd（弗洛伊德）算法的基本思想是：对一个顶点个数为 n 的有向网（或无向网），设置一个 n×n 的方阵 A(k)，其中除对角线的矩阵元素都等于 0 外，其他元素 $A^{(k)}[i][j](i\neq j)$ 表示从顶点 v_i 到顶点 v_j 的有向路径长度，k 表示运算步骤，k=-1、0、1、2、…、n-1。

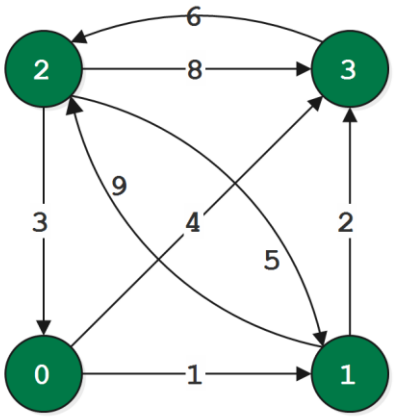
初始时：$A^{(-1)}$= Edge（图的邻接矩阵），即初始时，以任意两个顶点之间的直接有向边的权值作为最短路径长度：

1）对于任意两个顶点 v_i 和 v_j，若它们之间存在有向边，则以此边上的权值作为它们之间的最短路径长度；

2）若它们之间不存在有向边，则以 MAX 作为它们之间的最短路径。

以后逐步尝试在原路径中加入其他顶点作为中间顶点，如果增加中间顶点后，得到的路径比原来的最短路径长度减少了，则以此新路径代替原路径，修改矩阵元素，更新为新的更短的路径长度。

例如，在图 2-1 所示的有向网中，初始时，从顶点 v_2 到顶点 v_1 的最短路径距离为直接有向边<v_2,v_1>上的权值(=5)。加入中间顶点 v_0 之后，边<v_2,v_0>和<v_0,v_1>上的权值之和(=4)小于原来的最短路径长度，则以此新路径<v_2,v_0,v_1>的长度作为从顶点 v_2 到顶点 v_1 的最短路径距离 A[2][1]。



（a）有向网

$$Edge = \begin{bmatrix} 0 & 1 & 2 & 3 & \\ 0 & 1 & \infty & 4 & 0 \\ \infty & 0 & 9 & 2 & 1 \\ 3 & 5 & 0 & 8 & 2 \\ \infty & \infty & 6 & 0 & 3 \end{bmatrix}$$

（b）邻接矩阵

图 2-1　Floyd 算法：有向网及其邻接矩阵

将 v_0 作为中间顶点可能还会改变其他顶点之间的距离。例如，路径<v_2,v_0,v_3>的长度(=7)

小于原来的直接有向边<v_2,v_3>上的权值(=8)，矩阵元素 A[2][3]也要修改。

在下一步中又增加顶点 v_1 作为中间顶点，对于图中的每一条有向边<v_i,v_j>，要比较从 v_i 到 v_1 的最短路径长度加上从 v_1 到 v_j 的最短路径长度是否小于原来从 v_i 到 v_j 的最短路径长度，即判断 A[i][1]+A[1][j]< A[i][j]是否成立。如果成立，则需要用 A[i][1]+A[1][j]的值代替 A[i][j]的值。这时，从 v_i 到 v_1 的最短路径长度，以及从 v_1 到 v_j 的最短路径长度已经由于 v_0 作为中间顶点而修改过了，所以最新的 A[i][j]实际上是包含了顶点 v_i,v_0, v_1, v_j 的路径的长度。

如图 2.1 所示，A[2][3]在引入中间顶点 v_0 后，其值减为 7，再引入中间顶点 v_1 后，其值又减到 6。当然，有时加入中间顶点后的路径较原路径更长，这时就维持原来相应的矩阵元素的值不变。依此类推，可得到 Floyd 算法。

Floyd 算法的描述如下。

定义一个 n 阶方阵序列：$A^{(-1)}$,$A^{(0)}$,$A^{(1)}$, …,$A^{(n-1)}$，其中：

$A^{(-1)}$[i][j]表示顶点 v_i 到顶点 v_j 的直接边的长度，$A^{(-1)}$ 就是邻接矩阵 Edge[n][n]。

$A^{(0)}$[i][j]表示从顶点 v_i 到顶点 v_j，中间顶点（如果有，则）是 v_0 的最短路径长度。

$A^{(1)}$[i][j]表示从顶点 v_i 到顶点 v_j，中间顶点序号不大于 1 的最短路径长度。

……

$A^{(k)}$[i][j]表示从顶点 v_i 到顶点 v_j 的，中间顶点序号不大于 k 的最短路径长度。

……

$A^{(n-1)}$[i][j]是最终求得的从顶点 v_i 到顶点 v_j 的最短路径长度。

采用递推方式计算 $A^{(k)}$[i][j]：

增加顶点 v_k 作为中间顶点后，对于图中的每一对顶点 v_i 和 v_j，要比较从 v_i 到 v_k 的最短路径长度加上从 v_k 到 v_j 的最短路径长度是否小于原来从 v_i 到 v_j 的最短路径长度，即比较 $A^{(k-1)}$[i][k]+$A^{(k-1)}$[k][j]与 $A^{(k-1)}$[i][j]的大小，取较小者作为的 $A^{(k)}$[i][j]值。

因此，Floyd 算法的递推公式为：

$$A^{(k)}[i][j]=\begin{cases} Edge[i][j] & k=-1 \\ min\{A^{(k-1)}[i][j],A^{(k-1)}[i][k]+A^{(k-1)}[k][j]\} & k=0,1,2,\dots,n-1 \end{cases}$$

2.3 算法实现

Floyd 算法在实现时，需要使用两个数组：

1) 数组A:使用同一个数组A[i][j]来存放一系列的$A^{(k)}$[i][j],其中k=-1,0,1,…, n-1。

初始时，A[i][j]=Edge[i][j]，算法结束时 A[i][j]中存放的是从顶点 v_i 到顶点 v_j

的最短路径长度。

2) path 数组:path[i][j]是从顶点 v_i 到顶点 v_j 的最短路径上顶点 j 的前一顶点的序号。

Floyd 算法具体实现代码详见例 2.1。

例 2.1 利用 Floyd 算法求图 2-1(a)中各顶点间的最短路径长度,并输出对应的最短路径。

假设数据输入时采用如下的格式进行输入：首先输入顶点个数 n，然后输入每条边的数据。每条边的数据格式为：u v w，分别表示这条边的起点、终点和边上的权值。顶点序号从 0 开始计起。最后一行为-1 -1 -1，表示输入数据的结束。

分析：

如图 2-2 所示，初始时，数组 A 实际上就是邻接矩阵。path 数组的初始值：如果顶点 v_i 到顶点 v_j 有直接路径,则 path[i][j]初始为 i;如果顶点 v_i 到顶点 v_j 没有直接路径,则 path[i][j]初始为-1。在 Floyd 算法执行过程中，数组 A 和 path 各元素值的变化如图 2-2 所示。在该图中，如果数组元素的值有变化，则用粗体、下划线标明。

以从 $A^{(-1)}$ 推导到 $A^{(0)}$ 解释 $A^{(k)}$ 的推导。从 $A^{(-1)}$ 推导到 $A^{(0)}$，实际上是将 v_0 作为中间顶点。引入中间顶点 v_0 后,因为 $A^{(-1)}[2][0]+A^{(-1)}[0][1]=4$,小于 $A^{(-1)}[2][1]$,所以要将 $A^{(0)}[2][1]$ 修改成 $A^{(-1)}[2][0]+A^{(-1)}[0][1]$，为 4；同样 $A^{(0)}[2][3]$ 的值也要更新成 7。

当 Floyd 算法运算完毕，如何根据 path 数组确定顶点 v_i 到顶点 v_j 的最短路径？方法与 Dijkstra 算法和 Bellman-Ford 算法类似。以顶点 v_1 到顶点 v_0 的最短路径加以解释。如图 2-2 所示，从 $path^{(3)}[1][0]=2$ 可知，最短路径上 v_0 的前一个顶点是 v_2；从 $path^{(3)}[1][2]=3$ 可知，最短路径上 v_2 的前一个顶点是 v_3；从 $path^{(3)}[1][3]=1$ 可知，最短路径上 v_3 的前一个顶点是 v_1，就是最短路径的起点；因此，从顶点 1 到顶点 0 的最短路径为：$v_1 \rightarrow v_3 \rightarrow v_2 \rightarrow v0$，最短路径长度为 A[1][0]=11。

	$A^{(-1)}$				$A^{(0)}$				$A^{(1)}$				$A^{(2)}$				$A^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	1	∞	4	0	1	∞	4	0	1	**10**	**3**	0	1	10	3	0	1	**9**	3
1	∞	0	9	2	∞	0	9	2	∞	0	9	2	**12**	0	9	2	**11**	0	9	2
2	3	5	0	8	3	**4**	0	**7**	3	4	0	**6**	3	4	0	6	3	4	0	6
3	∞	∞	6	0	∞	∞	6	0	∞	∞	6	0	**9**	**10**	6	0	9	10	6	0
	$path^{(-1)}$				$path^{(0)}$				$path^{(1)}$				$path^{(2)}$				$path^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	-1	0	-1	0	-1	0	-1	0	-1	0	**1**	**1**	-1	0	1	1	-1	0	**3**	1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	**2**	-1	1	1	2	-1	**3**	1
2	2	2	-1	2	2	**0**	-1	**0**	2	0	-1	**1**	2	0	-1	1	2	0	-1	1
3	-1	-1	3	-1	-1	-1	3	-1	-1	-1	3	-1	**2**	**0**	3	-1	2	0	3	-1

注：在递推 $A^{(k)}[i][j]$ 和 $path^{(k)}[i][j]$ 时，有更新用粗体下划线标明。

图 2-2 Floyd 算法的求解过程中数组 A 和 path 的变化