# 数据结构实验课程综合性报告

赵鹏 2127405037

## 一、 问题描述及要求

在社交网络实验问题(1).(2)的基础上自己创意添加更多的功能。

## 二、 概要设计

### 1. 问题分析

本问题的要求简单明了，即在实现社交网络模型构建以及求得两人之间最短联系路径的基础上添加其他的创意功能。

### 2. 程序界面设计

将实验中添加的各种功能通过序号的方式进行编号，在主程序内编写 ShowFunction 函数，引导用户输入相应的功能编号，调用 SocialGraph 类中的对应的公有方法。

程序的功能列表如下所示：

```
*****Functions*****
[0] Calculate Shortest Path between two people
[1] Calculate the people one can indirectly know
[2] Judge whether everyone can know each other
[3] Add new relationship
[4] Delete relationship
[5] Add new people to the social graph
[6] Delete people from the social graph
[7] Get the crucial people and crucial edge in the social graph
[8] Exit
*****End*****
```

### 3. 系统功能列表

在实现问题一、二要求功能的基础上，额外添加了以下的创意功能:

1. 计算一个人在这张社交网络中可以通过朋友间接认识的人有哪些
2. 判断在这张社交网络中所有人能否相互认识
3. 在社交网络中添加新关系
4. 在社交网络中删除一对关系
5. 在社交网络中添加新人
6. 从社交网络中删除一个人
7. 获得社交网络中的关键人物[i]以及关键关系[ii]

[i] 关键人物定义为从社交网络中删除此人后及其相关的关系，社交网络的不连通程度增加，即对应图中的连通分量个数增加。

[ii] 关键人物定义为从社交网络中删除此关系后，社交网络的不连通程度增加，即对应图中的连通分量个数增加。

## 4.总体设计思路

本次任务的重点是创意功能的添加。在前两问中，程序将人抽象为图模型中的点，将两个人之间相互认识的关系抽象为边，依次构建社交网络图，并采用邻接矩阵的方式进行存储。本问题是需要在此基础上添加创意方法即可。

在添加上述创意功能的过程中，为 SocialGraph 类添加了较多的方法，具体如下：

| SocialGraph | |
|---|---|
| 方法 | 功能 |
| void PeopleCanKnow(int id) | 计算某个用户可以间接认识的人有哪些 |
| bool judgeConnect() | 判断在这张社交网络中所有人能否相互认识 |
| void addEdge(int u, int v) | 在社交网络中添加新关系 |
| void deleteEdge(int u, int v) | 在社交网络中删除新关系 |
| void addPeople(std::string name) | 在社交网络中添加新人 |
| void deletePeople(int u) | 从社交网络中删除一个人 |
| int GetMin(int d[]) | 计算两个用户最短路径的辅助函数 |
| void getVDCC_EDCC(); | 获得关键人物与关键关系 |
| void DCC_DFS(int x); | 获得关键人物与关键关系的辅助函数 |

除了 SocialGraph 类以外，实验中还设计了名为 UnionSet 的命名空间，并在其中定义并实现了并查集相关功能，用于辅助 judgeConnect 函数的实现。

# 三、详细设计

### 1.寻找能够间接认识的人

找出能够间接认识的人首先可以找出所有能够认识的人，除去初始认识的人即可。

“间接认识”表达了认识关系具有传递性。因此，要找到所有人可以认识的关系，只需在可达性矩阵上利用 floyd 算法求解可达性矩阵的传递闭包即可。在本实验中，可达性矩阵即为邻接矩阵。邻接矩阵 A 的 n 次幂表达通过 n 个结点的可达关系，根据离散数学传递闭包的求解方法，只需要将 A 到 A 的 n 次幂的结果逻辑或起来即可。时间复杂度为$O(n^3)$

在实现过程中，可以使用如下的代码更加简洁的实现：

```
1.  int tempEdge[MaxSize][MaxSize]{};
2.  memset(tempEdge, 0, sizeof tempEdge);
3.  for (int k = 0; k < vertexNum; k++)
4.   for (int   i = 0; i < vertexNum; i++)
5.    for (int j = 0; j < vertexNum; j++)
6.     tempEdge[i][j] |= (edge[i][k]==1) * (edge[k][j]==1);
```

最后依次判断每个人，若在邻接矩阵中两人不认识，但在传递闭包所求得的矩阵中两人认识，即为可以间接认识的人。

### 2. 判断所有人能否相互认识

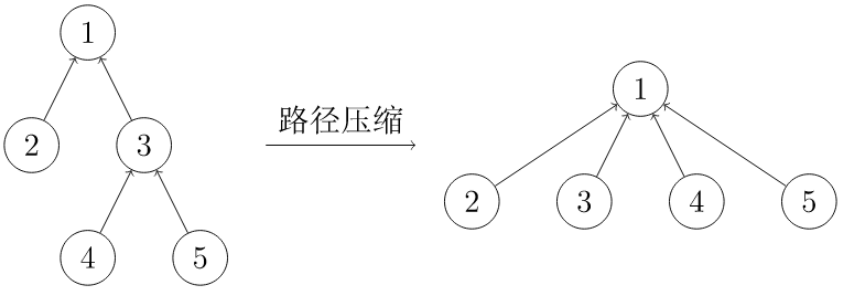
判断所有人能否相互认识，即判断这张图是否连通。我们可以通过判断是否存在最小生成树进行实现。时间复杂度为$O(n\log_2 n)$

这里采用 kruskal 算法来求解最小生成树。Kruskal 算法的基本思路是初始把具有 n 个点的点集 V 分为 n 个连通分量，然后将边按照权值从小到大进行排序，依次考察每条边。若考

察边的两个点不在同一个连通分量，则将两个连通分量连接为一个连通分量，否则舍去这条边以免造成回路，最终若只剩下一个连通分量则构建完成，也代表图连通，即所有人能够相互认识。

实现连通分量合并这一操作通常采取并查集进行实现。由于课本以及课上所示的并查集实现代码在某些特殊情况下效率较低，我借鉴算法竞赛中的并查集代码实现并查阅相关资料，对基础的并查集代码添加了两种优化："路径压缩"，"按秩合并"。

"路径压缩"优化基本思路是：由于查询过程中经过的每个元素都属于该集合，我们可以将其直接连到根节点以加快后续查询，如下图所示：




"按秩合并"优化的基本思路：是合并时，选择哪棵树的根节点作为新树的根节点会影响未来操作的复杂度。我们可以将节点较少或深度较小的树连到另一棵，以免发生退化。

在使用了上述两个优化后，并查集的每个操作的平均时间仅为$O(\alpha(n))$，其中$\alpha$为Ackermann 函数的反函数，其增长极为缓慢，因此可以认为单次操作的平均运行时间可以认为是一个很小的常数。

Ackermann 函数的定义$A(m,n)$如下：

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{otherwise} \end{cases}$$

**3.添加/删除关系**

由于关系利用邻接矩阵的值进行表达，edge[u][v]=1 表示存在 u,v 的关系,edge[u][v]=INF 表示不存在 u 到 v 的关系。故要添加和删除关系只需要直接修改邻接矩阵的值即可。加关系 u,v 只需要直接修改邻接矩阵 edge[u][v]=edge[v][u]=1 即可。删除关系 u,v 同理令 edge[u][v]=edge[v][u]=INF。时间复杂度为$O(1)$

**4.添加新人/删除某人**

对于添加新人的操作与初始化社交网络的过程中相类似。输入名称以及他具有的关系数和相应的关系，修改邻接矩阵的对应位置即可。平均时间复杂度为$O(n)$。

对于删除某人的操作，只需要在矩阵中断开他与其他人的所有关系即可。时间复杂度为$O(n)$

由于代码实现均较为简单，在此不再展示。

**5.求解关键人物与关键关系**

通过离散数学第七章图论的知识，在无向图中，删除某点或边后图的连通分量的个数增加，可以对应到求出无向图的割点和割边。通过查阅相关资料得知，可以利用 Robert Tarjan 提出的双连通分量算法对这个问题进行求解。

tarjan 算法求无向图双连通分量的基本思路如下：

对图进行深搜时，每个结点值只访问一次，被访问的结点与边构成搜索树。定义时间戳 dfn[x]表示结点 x 第一次被访问的顺序，定义追溯值 low[x]表示从节点 x 出发，所能访问到的最小时间戳。

**关键人物的求解**

根据割点判定法则：1.如果 x 不是根节点，当搜索树上存在 x 的一个 x 的一个子节点 y 满足 low[y]>=x，则 x 为割点。2.若 x 是根节点，当搜索树上存在两个子节点 y1,y2 满足上述条件，那么 x 为割点。

上述判定法则可以这样理解：low[y]≥dfn[x]，说明从 y 出发，在不通过 x 点的前提下，不管走哪条边，都无法到达比 x 更早访问的节点。故删除 x 点后，以 y 为根的子树 subtree(y)也就断开了。即环顶的点割得掉。反之，若 low[y]<dfn[x]，则说明 y 能绕行其他边到达比×更早访问的节点， x 就不是割点了。即环内内点割不棹

**关键关系的求解**

tarjan 算法求割边的基本思路如下：

根据割边判定法则：当搜索树上存在 x 的一个子节点 y,满足 low[y]>dfn[x]，则(x,y)这条边就是割边。

上述判定法则可以这样理解：low[y]>dfn[x]，说明从 y 出发，在不经过(x,y)这条边的前提下，不管走哪条边,都无法到达 x 或更早访问的节点。故删除(x,y)这条边，以 y 为根的子树 subtree(y)也就断开了。即环外的边割得断。

反之，若 low[y]<=dfn[x]，则说明 y 能绕行其他边到达 x 或更早访问的节点，(x,y)就不是割边了。即环内的边割不断。

由于两个算法求解的基本思路相似，都是通过 dfs 并在过程中进行一系列判断求解出割点割边，故可以实验一个深搜函数同时求解割点与割边。时间复杂度$O(n+m)$。

由于代码较长，故不在此展示。

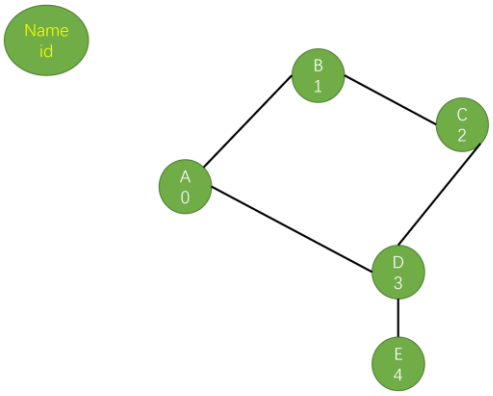
# 四、实验结果测试

使用一组数据，通过手动输入的方式检查各功能实现的正确性。

使用一组数据，通过手动输入的方式检查算法正确性：

```
Input the number of people in the socialnetwork:5
Input the name of people:A  B  C D E
People's id is numbered from 0 to 4
Input the number of A's friends:2
A's friends are (id):1 3

Input the number of B's friends:2
B's friends are (id):0 2

Input the number of C's friends:2
C's friends are (id):1 3

Input the number of D's friends:3
D's friends are (id):0 2 4

Input the number of E's friends:1
E's friends are (id):3
```

构建了如下图所示社交网络:




**间接认识测试：**

```
1
Input the id of people
0
C E
----------------------------
1
Input the id of people
2
A E
----------------------------
```

**添加删除关系及所有人相互认识测试：**

```
2
People can know each other from their friends
----------------------------
4
Input the id of two people you want to delete relationship
3 4
----------------------------
2
People can not know each other
----------------------------
3
Input the id of two people you want to add relationship
3 4
----------------------------
2
People can know each other from their friends
----------------------------
```

**测试添加和删除人物**

```
----------------------------
6
Input the id of people you want to delete from the graph
2
C has been deleted from the social graph
----------------------------
5
Input people's name you want to add to the social graph
F
F is numbered 5
Input the number of F's friends:1
F's friends are (id):4

----------------------------
0
Input the id of two people
1 5
Shortest Path:B>A>D>E>F
```

**求解关键人物与关键关系**

```
*****End*****
7
3 是这张社交网络图中的关键人物,去除其中任意一人这张图的不连通度都将增加
(4,3)是这张社交网络图中的关键关系，去除其中任意一个关系这张图的不连通度都将增加
----------------------------
```

**测试通过**

# 五、实验分析与探讨

通过上述测试，本实验较为顺利的实现了计划的各种功能。

对于所有人能否互相认识功能的实现，也存在另一种解法，即使用深度优先遍历或宽度优先遍历算法。任取一点作为起点，若所有点都能遍历到即说明图连通，所有人能够相互认识。

在实验过程中，也遇到了一部分问题，如：

在从图中删除人物后，由于编号保持不变，若在 kruskal 算法中仍然使用 vertexNum 作为连通块数量，会出现将连通判为不连通的问题。解决方法是额外存储图中实际点数。若直接修改 VertexNum 可能会导致在各种遍历操作中遍历不到最后一个未被删除的点。

# 六、小结

通过本次实验，我复习了图的邻接矩阵的存储结构，较好的设计了社交网络关系图模型并完成了各种创意功能的添加，同时也注意到了提升了程序的健壮性。我对 Floyd、Kruskal 算法也有了更加深刻的理解。通过查阅相关资料，我对并查集这一数据结构有了更进一步的了解。除此以外，我还通过自己查阅资料学习了 Robert Tarjan 提出的无向图的双连通分量算法。当然，这次实验也仍有很多不足，如添加的后几个功能都较为简单，没有更多新奇有趣的功能。通过此次机会，我有了丰富的收获。

## 参考文献

[1]Tarjan, R. E., & Van Leeuwen, J. (1984). Worst-case analysis of set union algorithms. Journal of the ACM (JACM), 31(2), 245-281.ResearchGate PDF
[2]Yao, A. C. (1985). On the expected performance of path compression algorithms.SIAM Journal on Computing, 14(1), 129-133.

[3]Gabow, H. N., & Tarjan, R. E. (1985). A Linear-Time Algorithm for a Special Case of Disjoint Set Union. JOURNAL OF COMPUTER AND SYSTEM SCIENCES, 30, 209-221.PDF
[4]dsu-complexity
[5] Robert Tarjan,Depth-First Search and Linear Graph Algorithms ResearchGate PDF