Data Structures and Algorithms
Exercise 1
iMyFace

You've been hired to help design a social networking site – iMyFace – which allows members to connect with each other in a manner similar (very similar) to the established players in the field. Users will request from each other the ability to push postings from their own page to other members' pages, and ask to receive updates from other members' pages. The business brains behind this startup believe the community will expand to around 400,000 members by the end of the year, and perhaps 2 million by the end of the following year. You're convinced this prediction is conservative. (You're paid mostly in stock options.)

You must design and build a simple application capable of providing the following operations. Note that you will not actually need to build the full system, just the parts that create and query the 'connections' within the community.

Required Operations:

Enroll

This allows a new user to provide basic information and become part of the community, which includes having a page on which information about the user is available. The following information is required:

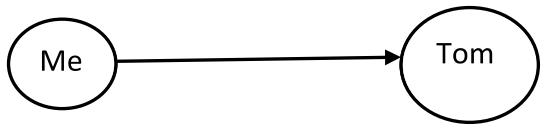
- Last Name
- First Name
- User ID (must be unique within the community in order to be accepted)
- Password
- Posts (Strings) – We will ignore these for now.

Once enrolled, other members of the community are called "faces".

OuttaMyFace

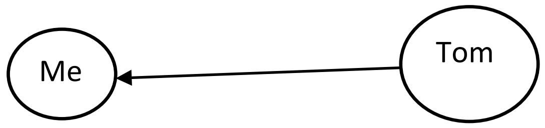
The user asks another member of the community ("the Face") to be "Outta My Face" – that is, to be "connected" – so If the Face accepts, then postings to the user's page will be broadcast to the Face's page and posted there too. Everyone "Outta" the Face's page will be asked if they want to be Outta the user's page too. For purposes of this exercise, assume the request will be accepted all the way down the line. For example,

```
>OuttaMyFace Tom
```



InMyFace
The user asks another member of the community to be "InMyFace" – that is, to have updates from the Face's page automatically sent to the user's page. This should work like the OuttaMyFace operation, except in reverse. So for example,

```
>InMyFace Tom
```



FacedUp
The user determines whether a Face is connected with them. In other words, will posts to the user's page be ultimately posted on the Face's page? The following information must be displayed.

- Whether the Face is currently a community member
- Whether the Face is InMyFace with respect to the user. So for example, if B is "OuttaMyFace" with respect to A, and C is "Outta" B, then updates to A will automatically be posted on B, and then automatically posted on C.
- The sequence(s) of Faces through which the user's posts will reach the Face?
- Whether the Face is InYourFace with respect to the user.
- The sequence(s) of Faces through which the Face's posts will reach the user.

FaceSpace
With which members are both user A and user B indirectly connected? In other words, who sees posts from both A and B?

Deliverables:

First deliverable:

- Design and implement the data structure(s) to support these operations. You may use any programming language you deem appropriate.
- Implement the Enroll operation.
- For each of the other operations, provide a description of the algorithm you will use, and its expected big-O performance. Clearly state any assumptions you've made including best case, average case, and worst case conditions.

- Provide a set of test cases to demonstrate that your enroll operation works as intended.
- Provide a transcript showing that the test cases have been applied successfully.

Second Deliverable:

- Implement the OuttaMyFace, InMyFace, and FacedUp operations.
- Provide a set of test cases to demonstrate that each operation works as intended.
- Provide a transcript showing that the test cases have been applied successfully.
- Use the provided test data to see whether the program's actual performance for each operation is in line with your prediction. Record the cpu time needed to complete each test case, and show the results in a table so you can compare the performance on the small data set with the performance on the large set. Explain what you see.

For an Extra Challenge…

If you want to really push yourself to a combinatorial explosion, try your hand (or face) at the following operations (after you've completed the others):

1. **MinimumFaces**
   What's the shortest number of faces through which user A and user B are connected?

2. **DegreesOfSeparation**
   Figure out the average path length between faced-up community members. What's the best performing algorithm (in terms of space or time) you can come up with?

3. **BackInYourFace**
   Which community members will wind up having their posts re-posted to their own page via face-ups with other community members? (This is obviously a situation we'd need to prevent from happening, but assume the OuttaMyFace and InMyFace operations don't check for it before creating the connection.)

4. **SaveFace**
   Store the current state of the entire community on disk so the application can be closed and re-opened without having to re-load the data.

5. **FaceList**
   If user A posts to her page, which other Faces will receive the post?

Test Data

You'll be provided with two sets of test data (small and large) you can use to help build and test your application. Each set will consist of three text files. The first file contains one line for each member of the community who should be enrolled. The member's data (including user id and password) is bang-separated for easy parsing. The second text file contains the name of an existing user, followed by the word “OuttaMyFace” or “InMyFace”, and the name of another existing user. Obviously there can be more than one entry for a user. The third file contains posts by individual members, where each line contains the name of the posting user, followed by the word “posts”, followed by a string.