1 Input file

The input file has the following format:
userID of a user, followed by a TAB character, followed by the ids of the friends of the user separated by a comma. For example,
1 2,3
3 1,2
means that user 1 is friends with user 2 and 3, and user 3 is friends with user 2 and 1. Each user and their friends is on a different line (separated by a ENTER character).

2 Map using Hadoop (Java)

In the map() function, Hadoop automatically splits the input file into lines. The function responsible for this is
`MutualFriends.FriendOfFriendsMapper.map(Object key, Text value, Context context)`
This function receives a value containing a string in the format of `userID+TAB+list of friends separated by a comma`, for example `1TAB2,3,4`, or if the user didn't add anyone, it will look like `1TAB`. The map functions works by

1. Splitting the value by the TAB character, which will result an an array of strings in the format of `[characters before the TAB, characters after the TAB]`. If there are no characters following the TAB, the array will have the format of `[characters before the TAB]`. For example, if the value is `1TAB2,3,4`, the array will be `["1", "2,3,4"]`; if the value is `1TAB`, the array will be `["1"]`

2. if the array resulting from splitting by the TAB character have a length of 1, that means the user did not add any friends yet. The key and the sent to the reducer will be

   (a) key = user ID
   (b) value = user ID + TAB + null

   For example, if the value is "1TAB", the output to the reducer will be

   (a) key = 1
   (b) value = 1TABnull

3. if the array resulting from splitting by the TAB character has a length of more than 1, it means the user added at least some friends. The map function will then create a variable `friendList = user ID + TAB + friends of user ID`. Then, for every user `u` that the user with `user ID` has added (the list can be obtained by splitting the second value of the array by the character ,), the mapper will output to the reducer

(a) key = u
(b) value = friendList

For example, if the array is [”1”, ”2,3,4”], the mapper will output

(a) key = 2
(b) value = 1TAB2,3,4
(c) key = 3
(d) value = 1TAB2,3,4
(e) key = 4
(f) value = 1TAB2,3,4

This ensures that the reducer will have every user along with all their friends of friends, and from which friend they come from. For example, if we have

(a) 1 TAB 2,3,4,5
(b) 6 TAB 2,3,4,5,7

For the user 2, the reducer will see

(a) key = 2; value = 1TAB2,3,4,5
(b) key = 2; value = 6TAB2,3,4,5,7

From this, we see that 2 can reach 2,3,4,5 via 1, and 2 can reach 2,3,4,5,7 via 6.

3 Reduce with Hadoop (Java)

The reducer collects, for each user, all their friends of friends along with from which friend these come from. For example, if 2 can reach 3 via 1 and 4, then 2 and 3 have two mutual friends, namely 1 and 4. The reduce function basically counts how many times a user ID appears in this lists, and returns the ones that appear most often after removing the ones that the user have already added. If the friend list contains null, the reducer outputs an empty list of suggestions. If not, the following steps will be taken: for example, if we have

- key = 2
- value = 1TAB2,3,4,5

Let `alreadyAdded` be a hashset of hashsets of strings that contains all the users that 2 has already added and `mutualFriendsCount` be a HashMap that uses a HashSet of strings as a key and an integer as value. The steps executed by the reducer will be the following (same steps for each value corresponding to 2):

1. The string `1TAB2,3,4,5` is splitted by tab to get `["1", "2,3,4,5"]`, and the value `"2,3,4,5"` is splitted by `,` to get the array `potentialFriends = [2, 3, 4, 5]`
2. The reducer will add `HashSet[(2)]` and `HashSet[(2,1)]` in `alreadyAdded` since 2 cannot add 2 and 2 has already added 1.

3. For each user `u` in the list `potentialFriends`, a `HashSet h = HashSet([2,u])` will be created. If h is already in `mutualFriendsCount`, it will be added as a key with a value of 1. If it's already added, we will increase its value by 1.

After all the values associated with 2 have been processed, the keys contained in `alreadyAdded` will be removed from `mutualFriendsCount`. The rest will be stored in an array of `Pair` objects, with a `count` attribute that indicates the number of mutual friends and a `uid` attribute that indicates the user ID. The array is then sorted by `Pair.count` (descending order) then by `Pair.uid` (ascending order). The uid of the 10 first Pair objects (or the whole array, if less than 10 friends of friends) will be returned. Example:

- key = 2
- value = 1TAB2,3,4,5
- value = 6TAB4,3,7,5
- value = 8TAB6,3,4,5

After processing all the values, `alreadyAdded` will contain

```
HashSet([
    HashSet([2]),
    HashSet([2, 1]),
    HashSet([2, 6]),
    HashSet([2, 8])
])
```

And `mutualFriendsCount` will contain

```
HashMap([
    HashSet([2   ]): 1,
    HashSet([2, 1]): 1,
    HashSet([2, 3]): 3,
    HashSet([2, 4]): 2,
    HashSet([2, 5]): 3,
    HashSet([2, 6]): 2,
    HashSet([2, 7]): 1,
    HashSet([2, 8]): 1,
])
```

After removing the contents of `alreadyAdded` from `mutualFriendsCount`, `mutualFriendsCount` will contain

```
HashMap([
    HashSet([2, 3]): 3,
    HashSet([2, 4]): 2,
    HashSet([2, 5]): 3,
    HashSet([2, 7]): 1,
])
```

And an array containing

```
array = [
    Pair{uid="3", count= 3},
```

```
    Pair{uid="4", count= 2},
    Pair{uid="5", count= 3},
    Pair{uid="7", count= 1},
]
```

will be produced, then sorted first according to count, then according to uid (ex. ”3” comes before ”5” because 3 ¡ 5, even though they both have three mutual friends with 2):

```
array = [
    Pair{uid="3", count= 3},
    Pair{uid="5", count= 3},
    Pair{uid="4", count= 2},
    Pair{uid="7", count= 1},
]
```

The output will be

- key = 2
- value = 3,5,4,7