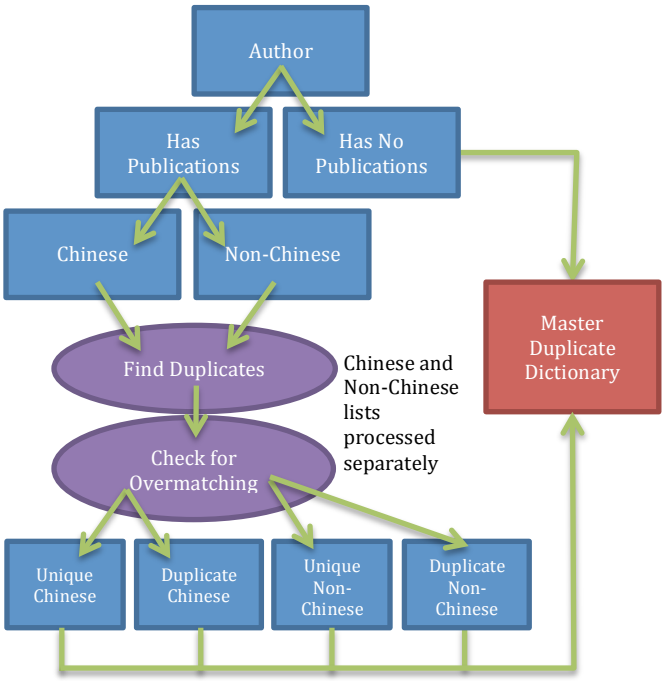
Project 2

Bryant Moquist

Algorithm Description:

My Python program begins by constructing a set of the 100 most common surnames of Chinese origin across regional and dialect variants (Mandarin, Cantonese, Korean, etc). totaling 689 unique names. Next, the program constructs a set of 22 banned names that are commonly used in English and Chinese such as "tom" and "long". Then, the program reads in PaperAuthor.csv and constructs a set of the file's 2,143,148 unique author ids.

Algorithm Diagram:



With these sets constructed, my program initializes two dictionaries: *author* and *master*. *Author*'s key-value pair is comprised of the author id (key) and then the list of strings corresponding to the author's name. *Master*'s key-value pair is the author id (key) and the corresponding list of duplicate author ids (including the current key). Reading in Author.csv, each author is checked to see if they have published works listed in PaperAuthor.csv using his or her author id. If the author has not published any papers, the author is assumed to have no duplicates, and the corresponding id is hashed into *master*. If the author has published papers, the author's name string is standardized using regular expressions, converted to lowercase, and hashed into *author* using the author's id. Following this process, 150,689 names are categorized as unique and placed in *master*, and the remaining 96,515 in *author* will be processed further.

Next, every author in *author* is categorized as either Chinese or not using the Chinese and banned word sets. To do this, the algorithm examines each author's name counting the number of words in the name and Chinese strings not in the banned set. For example, the list: [['john'] , ['ming'] , ['lin']] has 2 Chinese strings and 3 strings in total. The algorithm categorizes a name as Chinese if it has at least one Chinese string and satisfies the inequality: (total strings – Chinese strings) < 2. When appended to either the Chinese or non-Chinese lists, names are stored in a name string object called *nstr* that holds both the author's name and id. After all of the names have been processed, the result is a Chinese list of 7,097 names and a non-Chinese list of 89,418 names.

Both the Chinese and non-Chinese lists are examined for duplicates separately using the same procedure. First, an empty list for unique names and an empty list for duplicates are created. Names are compared versus one another using the *compare* function. This function checks if two names are identical (e.g. [['john'] , ['doe']] and [['john'] , ['doe']]), if one name contains prefixes but is otherwise identical (e.g. [['j'] , ['doe']] and [['john'] , ['doe']]), or if one is a subset of the other (e.g. [['john'] , ['doe']] and [['john'] , [' turner '] , ['doe']]). If any of the above occur, the two names are grouped as duplicates and placed in the duplicate list. If no duplicates are found, the name is placed in the unique list. Each name is checked against the duplicate list and the remaining names in the original list. However, once a name is placed on the unique list, names are no longer checked against it, which improves the procedure's running time.

Once the duplicate lists are found, they are reprocessed by a splitting procedure that breaks apart duplicates that are overmatched. To do this, the algorithm considers each group of duplicates as an undirected graph and finds the longest common extended name ("LCEN") for each word. For example, given the names on the right, "john david turner doe" would be considered the LCEN of "john d turner doe" whereas "john chris turner doe" would be considered the LCEN of "john c turner doe". Consequently, this duplicate collection would be split into two separate duplicate collections. Because "john turner doe" and "john doe" could belong to either collection, they are arbitrarily placed in one of the two collections.

Name Collection Example:
john david turner doe
john chris turner doe
john d turner doe
john c turner doe
john turner doe
john doe

After the Chinese and non-Chinese name lists have been checked for overmatching, both the unique lists and duplicate lists are written into *master* using the author ids. Finally, each entry in *master* is written to an output CSV file.

Strategy:
My strategy focused on using name matching to identify duplicates and attempting to reduce the search space whenever possible to improve the algorithm's running time. First, by considering any author in Author.csv to be unique if they had no published papers in PaperAuthor.csv reduced the number of names with potential duplicates by 60.9%. Intuitively if an author hasn't published any papers, his or her name will likely appear less often in the corpus so his or her name will likely be unique or have fewer duplicates than someone who actively publishes. This assumption may not hold for all datasets but seemed to work well with this one.

By splitting the remaining names into Chinese and non-Chinese lists, the search space was reduced further as the Chinese names did not need to be exhaustively checked versus the non-Chinese names and vice versa. Furthermore, the categorization provided a better grouping to look for duplicate names and likely improved accuracy.

Last, overmatching was a problem particularly among Chinese names. Consequently, the algorithm includes a splitting procedure to ensure names in duplicate collections make sense in the context of one another by checking each word's longest common extended name. If names didn't share a LCEN, the algorithm split the duplicate into different collections to reflect their differences.

Offline Evaluation:
My offline evaluation was conducted by constructing small datasets and testing individual functions to ensure that my program processed the data as expected. My test datasets included subsets of the real data and hand constructed situations where name disambiguation would be difficult. Once each aspect of my program was tested, I began testing the program on larger datasets. With the full dataset, my algorithm takes around 2.5-3 hours to complete all of its calculations and produce an output file. When the algorithm was submitted online, it resulted in a score of 0.97400.

References:

- "Effective String Processing and Matching for Author Disambiguation" Chin, Juan, Zhuang, et al. National Taiwan University, 2013. http://www.kdd.org/kddcup2013/sites/default/files/papers/track2place1.pdf
- List of the 100 most common Chinese surnames: http://en.wikipedia.org/wiki/List_of_common_Chinese_surnames