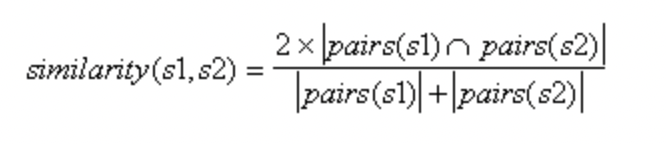
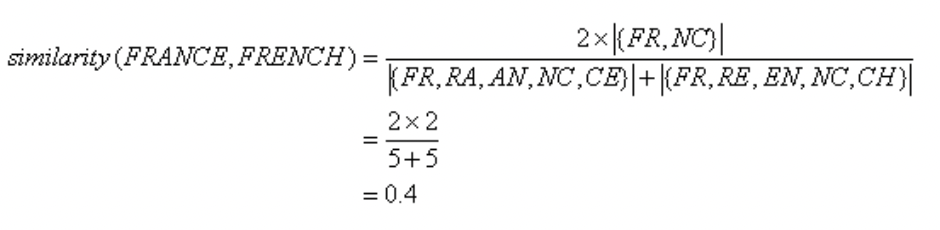
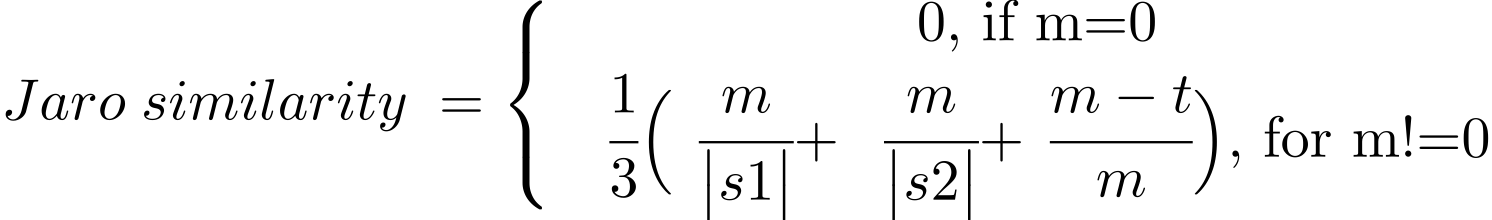
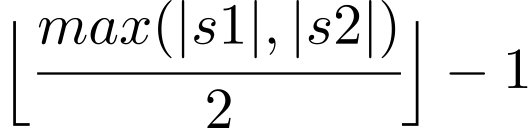
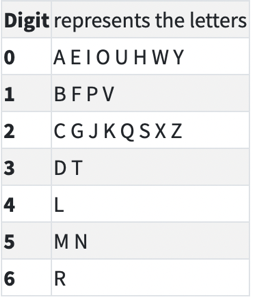
**Task:** Find appropriate string-matching algorithm for names expecting small mistakes and alternatives in the spellings.

There are over a dozen of string matching algorithms that come from as early as 1920s. Each of them has their drawbacks and advantages and are perfectly suitable for different scenarios. Below I will provide a general overview and analysis of the most popular and comparably more applicable matching algorithms that I have found, and also will provide an opinion of which one is the best to use for our task scenario.

* **Levenshtein distance** 
  + Shows how many transformations (insertion, deletion or substitution) are necessary to obtain a word given another one. Used for string matching and for spell checks mostly.
    - The final obtained Levenshtein distance value must be converted to some percentage/coefficient in order to easier compare with other words. The less the distance, the similar are the words. But still having a percentage counter will be nicer.
    - This method is not taking into account the pronunciation. As our task is directly related to names and surnames, pronunciation is key as the same name can be written in a lot of different alternative ways.
      * For example: GILY and GEELY have distance value of 3. Which is a lot compared to the length of the words (less than 50% similarity). Seeing this value, one would deduce that the two words are not really matching, however if we take a look from the perspective of pronunciation and names, these two actually match pretty well.
  + I do believe that Levenshtein method is less applicable for names as it will be skipping a lot of name variants and aliases that actually sound the same and can match.
  + Additionally, this method does not take into account any semantic meaning and is inefficient for longer strings.
* **Bigram similarity**
  + 
  + With Bigram we are dividing the given words into pair of letters and using the above formula we are calculating their similarity value.
  + Example:
    - 
  + This method can be very useful in our scenario as we are taking into account all possible adjacent combinations of letters, however, just like the previous method, Bigram similarity does not take into account the sounding of the words which, as already mentioned, are crucial when working with name aliases.
* **Jaro distance**
  + The Jaro Similarity is calculated using the following formula
  + 
  + where:
    - m is the number of matching characters
    - t is half the number of transpositions
    - |s1| and |s2| are the lengths of strings s1 and s2 respectively.
  + The characters are said to be matching if they are the same and the characters are not further than
    - 
  + Calculation:
    - Let s1=”arnab”, s2=”raanb”, so the maximum distance to which each character is matched is 1.
    - It is evident that both the strings have 5 matching characters, but the order is not the same, so the number of characters that are not in order is 4, so the number of transpositions is 2.
    - Therefore, Jaro similarity can be calculated as follows:
    - Jaro Similarity = (1/3) \* {(5/5) + (5/5) + (5-2)/5 } = 0.86667
  + This method provides with a nice similarity value, takes into account the mispositioning of characters which can be the case in our scenario, however, again there is no implementation of pronunciation of words.
  + This method can be considered to be implemented as its calculation of similarity value is ambiguous with calculation of position fault, meaning that for our case the mistakes and alternatives of same names can be distinguished.
* **Soundex**
  + A Soundex code is a four-character string in the form of an initial letter followed by three digits, such as **Z452**.
  + The initial letter is the first letter of the surname, and the three digits are drawn from the sounds within the name using the following algorithm.
    - Discard all non-letter characters from surname: dashes, spaces, apostrophes, and so on.
    - Encode each letter as a digit using the table below.
    - Coalesce adjacent duplicate digits from code (e.g. **222025** becomes **2025**).
    - Replace the first digit of code with the first letter of the original name, converting to uppercase.
    - Remove all zeros from code.
    - Make the code exactly length 4 by padding with zeros or truncating the excess.
  + Through Soundex algorithm we are able to catch words that have similar sounding. As previously mentioned, pronunciation is a crucial point for our task as we are looking for matching forenames and surnames. Soundex provides us with such method, however the problem with Soundex is that it does not have a similarity value, so we cannot tell if a word is more similar or less similar. This could be a problem as part of our task is to rank from the most-likely match to less likely one but with just having the Soundex code we will not be able to do that.
  + In general, Soundex has many well-known limitations, including inability to handle different first letters with identical pronunciations (e.g., Soundex of “Kris” is K620, but Soundex of “Chris” is C620), truncation of long names, and bias towards English pronunciations.
* **Editex**
  + Editex is a phonetic distance measure that combines the properties of edit distances with the letter-grouping strat- egy used by Soundex and Phonix. In other words, Editex method combines Soundex with Levenshteindistance, meaning that we get matching with two properties: sounding(pronunciation) & character similarity.
  + Editex is basically a more developed version of Soundex which covers the limitations of the preceding.
  + I do believe that Editex is the most suitable option for our scenario as it takes into account the sounding of names and their similarity. We will be able to produce a correct similarity value and could sort the matching answers in order.

**Final Implementation suggestion:** I am suggesting to use Editex matching algorithm for names in the black list for forenames, surnames and third-names. Afterwards, we will filter out all the results that have generated a similarity value of less than 60-70%. In order to further increase our probabilities, we can also check other available data (date of birth or Passport Number) with the remaining matching candidates. In the end the results will be sorted in decreasing manner and presented to the user.