Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Sebastian Walter

Information Retrieval
WS 2023/2024

http://ad-wiki.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Exercise Sheet 7

Submit until Tuesday, December 12 at **12:00pm (noon)**

Exercise 1 (20 points)

Write a program for fuzzy prefix search with the following functionality. To save you some work, we have already implemented functions that efficiently compute the prefix edit distance between two strings and that merge multiple inverted lists. They are available in two versions: as pure Python code in *utils.py* and as a Rust-based Python package called *ad-freiburg-qgram-utils*. If you want to use the faster Rust-based version, install the package with *pip install ad-freiburg-qgram-utils* and it will be used automatically.

As usual, we provide a code template with unit tests on the Wiki.

1. Implement a method *build_from_file* that builds a q-gram index from the entity names in a given entity file. The file format is as follows: one line per entity with tab-separated columns, with the entity name in the first column, a popularity score in the second column, synonyms in the third column, and additional information like a short description in the remaining columns; the first line contains a header with a short description of each column. The scores are needed for ranking (see item 2), the use of synonyms is optional (see item 6), and the additional information is just for the output (see item 4). *Note: Before computing the q-grams, normalize each string by lowercasing and removing all non-word characters using the method* normalize *provided in the code template. Don't forget to later normalize the queries in the same way, see item 4 below.*

2. Implement a method *find_matches* that for a given string x, finds all entities y with $\text{PED}(x, y) \leq \delta$ for a given integer δ. Use the algorithm explained in the lecture: First use the q-gram index to exclude entities y that do not have a sufficient number of q-grams in common with x. Only for the remaining candidate entities compute the PED using the provided code (see above). The method should return for each entity y with $\text{PED}(x, y) \leq \delta$, its ID and the value of $\text{PED}(x, y)$. These pairs should be sorted in order of ascending $\text{PED}(x, y)$ and descending score. That is, all entities with $\text{PED} = 0$ should come before all entities with $\text{PED} = 1$, and so on, and the entities with the same PED should be sorted by score (higher score first). Additionally, in the method you should calculate the number of PED computations so that the correctness of that number can be verified for the given test cases.

[$$p, $pl, ple, lea, eas, ase, set, etu, tur]

3. Implement a method *get_infos* that returns the matched synonym, name, score, and additional info for an ID as returned by *find_matches*. If the q-gram index is built without synonyms, just set the synonym to the name of the entity (see item 6).

4. Implement a *main* function that builds a q-gram index from a given file, and then, in an infinite loop, lets the user type a query and shows the result for the *normalized* query (see item 1). The result should contain the (up to) top-5 matches computed with *find_matches*. Take $\delta = \lfloor |x|/(q+1) \rfloor$, where x is the normalized query. Understand that this value of δ ensures that a match needs to have at least one q-gram in common with x. For each match, output the original entity name (not the normalized name) and its description. Optionally, also output the Wikidata and Wikipedia URL of the entity (also available in the additional information). If there are more than 5 matches, also print the total number of matches. Also print the following information: the total time to process the query, the number of inverted lists that were merged, and the number of PED computations.

5. Run your system over the wikidata-entities.tsv dataset provided on the Wiki, with $q = 3$ (that is, using a 3-gram index). Add a row to the result table on the Wiki following the examples already given there.

6. Optionally (not needed for full points), support an option *--use-synonyms* (not used for the entry in the Wiki table) to also use the synonym information that is provided as part of the input file (see item 1), namely in the following way: When the query matches one of the synonyms, count this as a match for the original entity name. If an entity name gets several matches in this way (for its original name or one of its synonyms), rank it by the best of the corresponding PED values. The output should be just as described under item 4, except when the position in the ranking is due to a synonym match: in that case, additionally output the corresponding synonym (not all synonyms, because that can be quite many for popular entities). *Hint: You can implement this functionality by treating each synonym as a separate entity with a unique ID while also keeping track of the original entity each synonym refers to.*

Commit your code to our SVN, in a new subfolder *sheet-07*. The dataset files should not be committed.

As usual, in your *experiences.md*, provide a brief account of your experience with this sheet and the corresponding lecture. Make sure to add a statement asking for feedback. In this statement specify to which degree and on which parts of the sheet you want feedback. In addition, say how much time you invested and if you had major problems, and if yes, where.