Humboldt-Universität zu Berlin

Institut für Informatik

Klassifikation und Benchmarking von String-Suche Bibliotheken

Exposé

von Franz Wilhelm Weiss

25. Jan. 2023

1. Einleitung

String-Suche-Algorithmen bezeichnen eine spezielle Gruppe von Algorithmen, die einen bestimmten String innerhalb einer Menge von Strings suchen. Als Strings werden Zeichenketten über einem Alphabet Σ bezeichnet. Die Länge eines Strings wird mit |s| bezeichnet. Der String, nachdem gesucht wird, wird als Query bezeichnet und die Menge der zu durchsuchenden Strings als Dictionary.
Anwendung findet diese Art von Algorithmen beispielsweise beim Finden von DNA in einem Genom oder bei der Rechtschreibprüfung von Microsoft Word [1]. Sie werden allgemeinhin in exakte String-Suche-Algorithmen und fuzzy String-Suche-Algorithmen unterteilt. Exakte String-Suche-Algorithmen suchen alle exakten Vorkommen der Query innerhalb des Dictionaries. Die Fuzzy String-Suche wird auch als Ähnlichkeitssuche oder “Similarity Search” bezeichnet. Sie ist folgendermaßen definiert:

Similarity Search
Gegeben sei eine Menge von Strings $S = \{s_1,...s_n\}$ und eine Query q. Dazu sind ein Schwellwert k und ein Ähnlichkeitsmaß bzw. Abstandsmaß definiert. Similarity Search gibt alle Strings aus S zurück, deren Abstand zu q unter dem Schwellwert k liegt [2].

Zwei bekannte Abstands-Metriken sind die Levenshtein-Distanz, auch Edit-Distanz genannt, und die Jaccard-Distanz.

Ein Algorithmus, der eine approximative String-Suche implementiert, ist der FastSS Algorithmus. Dieser sucht mithilfe einer Variante der Levenshtein-Distanz nach allen ähnlichen Vorkommen einer Query q. Die Levenshtein-Variante erlaubt hierbei nur das Löschen einzelner Zeichen [3].

2. Zielstellung

Es gibt Paper wie [1] “String similarity search and join: a survey”, die eine Übersicht zu verschiedenen Algorithmen, die die Suche in Dictionaries durchführen, geben. Allerdings gibt es kaum Übersichten über Bibliotheken, in denen diese Algorithmen unmittelbar verfügbar wären. Da Anwender allerdings aus Zeitgründen nicht lange an einer Implementierung dieser Algorithmen sitzen wollen, nutzen sie lieber fertig implementierte Algorithmen und integrieren diese in ihren Code.
Deswegen soll das Ziel dieser Arbeit sein, frei verfügbare Bibliotheken zur Ähnlichkeitssuche in Dictionaries aufzulisten und auf Speicherverbrauch und benötigte Zeit zu benchmarken. Die zu untersuchenden Bibliotheken sollen zur Suche entweder die Levenshtein-Distanz oder die Jaccard-Distanz als Ähnlichkeitsmaß verwenden.

3. Theoretische Grundlage

Die Jaccard-Distanz und die Levenshtein-Distanz sind unterschiedliche Maße dafür, wie ähnlich sich zwei Strings sind. Sie fallen in zwei unterschiedliche Kategorien:

3.1 Token basierende Ähnlichkeit

Jeder String s wird als Menge G seiner Q-Gramme dargestellt [1]. Die Menge G eines Strings s ist definiert als $G = \{ s' \mid s' \in s \wedge |s'| = q\}$.
Für den String s = “Baum” ist die Menge G der 3-Gramme {Bau, aum}. Die Ähnlichkeit zwischen String s1 und s2 wird so über die Anzahl der gemeinsamen Q-Gramme berechnet.
Die Jaccard-Distanz ist ein Verfahren, das auf so einer Zerlegung basiert und wird mit folgender Formel berechnet:

Jaccard-Distanz: $JAC(G1,G2) = \frac{|G1 \cap G2|}{|G1 \cup G2|} = \frac{|G1 \cap G2|}{|G1| + |G2| - |G1 \cap G2|}$

Wobei |G1|, |G2| die Anzahl der Q-Gramme in G1 bzw. G2 angibt [1].
So ist zum Beispiel bei den Strings s1 = “Raum” mit G1 = {Rau, aum} und
s2 = "Baum" mit den G2 = {Bau, aum} die Jaccard-Distanz bei JAC(s1,s2) = ⅓ .

3.2 Zeichen basierende Ähnlichkeit

Jeder String wird als eine Sequenz von Zeichen über einem Alphabet Σ dargestellt. Die Levenshtein-Distanz misst das Minimum an Operationen, die notwendig sind, um von S1 zu S2 zu gelangen. Als Operationen gelten nur Substitution, Einfügen und Löschen einzelner Zeichen [2].

So ist zum Beispiel die Levenshtein-Distanz der beiden Strings S1 = ”Baum” und S2 = ”Raum” 1, da in S1 nur das Zeichen B mit R ersetzt werden muss, um von S1 zu S2 zu kommen.

4. Vorgehen

4.1 Kandidaten

Ein Kriterium, nach dem die Bibliotheken ausgewählt werden, ist die Programmiersprache, in der sie geschrieben wurden. So werden nur Bibliotheken, die in Java, C++ oder Python geschrieben wurden, betrachtet. Des Weiteren werden, um Aktualität zu berücksichtigen, keine Bibliotheken betrachtet, an denen länger als 5 Jahre nicht mehr gearbeitet wurde. Stichtag hierfür ist der 1.1.2018. Außerdem werden alle Bibliotheken ausgeschlossen, die verteilte Algorithmen implementieren, also die Suche über verschiedene Geräte verteilen. Zudem ist wichtig, dass alle Bibliotheken in der Lage sind, ein Dictionary entgegenzunehmen und eigenständig nach Vorkommen der Query in diesem zu suchen.
Mögliche Kandidaten werden über die Plattformen Github, PyPI und über Anfragen an die Suchmaschinen Google und DuckDuckGo gesucht.
Ein Beispiel für einen Kandidaten ist die in Java geschriebene Bibliothek Apache Lucene der Apache Software Foundation. Apache Lucene umfasst viele Features zur Analyse und Suche innerhalb von Texten, Dokumenten und Dictionaries[1]. Ein weiterer Kandidat ist Boost. Boost ist eine Ansammlung von in C++ geschriebenen Bibliotheken und beinhaltet eine Funktion zum Suchen in Dictionaries mit der Levenshtein-Distanz als Fehlermaß[2]. Eine in Python geschriebene Bibliothek ist zum Beispiel thefuzz. Diese ist das Nachfolgeprojekt von FuzzyWuzzy einer häufig genutzten Bibliothek für die fuzzy String-Suche[3]. Eine Bibliothek, die in C++ geschrieben ist und einen Port nach Python anbietet, ist die RapidFuzz Bibliothek[4].

4.2 Daten

Die zu untersuchenden Bibliotheken sollen auf Laufzeit und Speicherverbrauch untersucht werden. Hierfür sollen die Bibliotheken einen Datensatz von etwa 100.000 Medikamentennamen durchsuchen. Diese stammt aus dem PREDICT-Projekt [4]. Die Namen sind Strings über einem Alphabet von etwa 60 Zeichen und haben eine durchschnittliche Länge von 22 Zeichen.
Für die Suche werden 1000 Queries aus den Goldstandards, die für die Python Bibliothek preon verwendet wurden, entnommen [5]. Die Suche soll für unterschiedliche Thresholds k durchgeführt werden.

4.3 Benchmarking

Für das Benchmarking wird die Bibliothek preon als Baseline genutzt. Preon enthält eine Methode zur Suche in Dictionaries, welche für die Daten aus 4.2 entwickelt wurde und gibt somit eine Referenzzeit für die anderen Bibliotheken [5].
Für die Laufzeit wird die Clock-Zeit über die Standardbibliotheken der einzelnen Programmiersprachen gemessen.

---

[1] https://lucene.apache.org/ Abrufdatum: 8.1.2023
[2] https://erikerlandson.github.io/algorithm/libs/algorithm/doc/html/algorithm/Sequence.html Abrufdatum: 24.1.2023
[3] https://github.com/seatgeek/thefuzz Abrufdatum: 24.1.2023
[4] https://github.com/maxbachmann/RapidFuzz Abrufdatum: 24.1.2023

Der Speicherverbrauch wird über die Datei “status” des "/proc" -Filesystems erfasst. Diese enthält unter anderem Informationen zum virtuellen und physischen Speicherverbrauch eines laufenden Prozesses und kann über die “Process ID” erfasst werden[5]. Dazu soll ein Programm geschrieben werden, das automatisch die zu benchmarkenden Programme startet, von diesen ihre Process ID erfragt und anschließend regelmäßig die Status-Datei abruft, um so Veränderungen im Speicherverbrauch zu messen.

Die Bibliotheken sollen sprachübergreifend miteinander verglichen werden, auch wenn Python, aufgrund seiner Eigenschaft eine interpretierte Programmiersprache zu sein, langsamer ist, als es die kompilierten Sprachen Java und C++ sind. Dadurch soll eine bessere Gesamtübersicht über die Bibliotheken entstehen.

4.4 Endergebnis

Am Ende soll eine Gesamtübersicht über die verschiedenen Bibliotheken erstellt werden, in denen diese, ihrer Leistung im Benchmark nach, absteigend aufgelistet werden.

5. Literaturverzeichnis

[1] Yu, M., Li, G., Deng, D. *et al.* String similarity search and join: a survey. *Front. Comput. Sci.* 10, 399–417 (2016). https://doi.org/10.1007/s11704-015-5900-5

[2] Sebastian Wandelt, Dong Deng, Stefan Gerdjikov, Shashwat Mishra, Petar Mitankin, Manish Patil, Enrico Siragusa, Alexander Tiskin, Wei Wang, Jiaying Wang, and Ulf Leser. 2014. State-of-the-art in string similarity search and join. SIGMOD Rec. 43, 1 (March 2014), 64–76. https://doi.org/10.1145/2627692.2627706

[3] T. Bocek, E. Hunt, D. Hausheer and B. Stiller, "Fast similarity search in peer-to-peer networks," NOMS 2008 - 2008 IEEE Network Operations and Management Symposium, 2008, S240-S247,

[4] Ulf Leser, Nils Blüthgen, Ulrich Keilholz, Christine Sers, Reinhold Schäfe, Marius Kloft. Predict.Https://pathologie-ccm.charite.de/forschung/arbeitsgruppen/ag_sers_molekulare_tumorpathologie/predict/, Abrufdatum: 24.12.2022

[5] Arik Ermshaus, Michael Piechotta, Gina Rüter, Ulrich Keilholz, Ulf Leser and Manuela Benary. 2022, preon: Fast and accurate entity normalization for drug names and cancer types in precision oncology, unpublished manuscript

[5] https://linux.die.net/man/5/proc Abrufdatum: 8.1.2023