PRO1-A WiSe 2020/21: Projektreport
String Matching

Sara Derakhshani
Matrikelnummer: 792483

March 17, 2021

1 Aufgabenbeschreibung

Die grundlegende Aufgabe dieses Projekts war die Implementierung eines Kommandozeilenwerkzeugs, das Begriffe in einem Text sucht. Diese Suche sollte mithilfe eines String-Matching-Algorithmus durchgeführt werden und als Resultat alle Startpositionen des gesuchten Strings im Text zurückgeben.
Zur Aufgabe gehörte die Suche in direkten Text-Eingaben, in einzelnen .txt-Dateien und Ordnern mit mehreren Dateien zu ermöglichen. Außerdem sollte die Beachtung von Groß- und Kleinschreibung ausgeschaltet werden können und eine Auswahl aus zwei verschiedenen String-Matching-Methoden geboten werden.
Die Implementierung des naiven String-Matching-Algorithmus war obligatorisch und der zweite Algorithmus stand zur Auswahl. Ich habe mich für den Finite-State-Matcher nach Cormen et al. (1990) entschieden.

2 Programmstruktur

Mein Programm ist aufgeteilt in eine StringMatcher-Klasse, die die String-Matching-Algorithmen implementiert, und ein Script für das Kommandozeilenwerkzeug. Das Kommandozeilenwerkzeug verwendet die StringMatcher-Klasse.

2.1 String-Matching

Die StringMatcher-Klasse repräsentiert StringMatcher-Objekte, die mit einer der implementierten Methoden initialisiert werden und mit dieser Methode String-Matching durchführen können. Die Default-Methode ist, wie vorgeschrieben, der intelligentere Algorithmus, also der Finite-State-Matcher.
Die Klasse hat eine öffentliche Methode zum Matchen. Diese nimmt ein Muster und einen Text als Argumente und gibt eine Liste der Anfangspositionen der gefundenen Mustervorkommen zurück.
Die String-Matching-Algorithmen sind mit privaten statischen Methoden implementiert, die sich stark an den Pseudocodes aus Cormen et al. (1990) orientieren.
Der naive Algorithmus entspricht im Ablauf genau dem Pseudocode. Zu beachten war, dass Pythons range() mit einem halboffenen Intervall arbeitet. Außerdem enthält

der Pseudocode ein Print-Statement, sobald ein Auftreten des Musters gefunden wird. Da ein Print-Statement erst im Kommandozeilenwerkzeug formuliert werden soll, werden die gefundenen Positionen erst in einer Liste gespeichert und am Ende vollständig zurückgegeben.
Diese beiden Punkte sind auch für den Finite-State-Matcher-Algorithmus zutreffend. Die Implementierung dieses Algorithmus unterscheidet sich jedoch etwas mehr vom Pseudocode, da die Befehlsstruktur nicht genauso in Python geschrieben werden konnte. Übernommen habe ich die Aufteilung in zwei Methoden und deren Input-Parameter.
Die Methode, die mit dem Text den Automaten "durchläuft" und die Startpositionen speichert, wenn der akzeptierende Zustand erreicht wird, ist genau wie der Pseudocode aufgebaut. Die dafür benötigten Übergangsfunktionen werden aus einem Dict ausgelesen. Dieses Dict wird mit der zweite Methode erstellt.
Wie im Pseudocode wird für jede mögliche Position im Muster und jedes Zeichen des Automatenalphabets überprüft, ob das Zeichen an der jeweiligen Position vorkommen kann. Ist dies nicht der Fall, wird ein Übergang zu einem vorherigen Zustand erstellt. Zu diesem Zustand muss beim Matching zurückgegangen werden, falls das Zeichen in dieser Position auftritt.

2.2 Kommandozeilenwerkzeug

Das Kommandozeilenwerkzeug importiert die StringMatcher-Klasse und besteht aus einer Hauptfunktion, sowie einer Pretty-Print-Hilfsfunktion. Die benötigten Parameter der Hauptfunktion sind der Text und das Suchmuster. Das Format des Texts wird überprüft und mithilfe eines StringMatcher-Objekts wird nach den Vorkommen des Musters gesucht. Die optionalen Parameter sind ein Flag, um die Beachtung der Groß- und Kleinschreibung zu deaktivieren, sowie die Matching-Methode. Als Eingabe-Methode werden nur die beiden implementierten Matching-Methoden erlaubt. Zum Parsen der Eingabeparameter habe ich click verwendet.

3 Erweiterbarkeit

Eine sinnvolle Erweiterung der StringMatcher-Klasse und des Kommandozeilenwerkzeugs wäre die Implementierung weiterer String-Matching-Algorithmen. Dafür müsste die Liste der erlaubten Input-Methoden in der Klasse erweitert und ein Algorithmus so implementiert werden, dass dieser die gematchten Positionen in einer Liste zurückgibt. Dann könnte diese Matching-Methode den click-Optionen des Methoden-Parameters hinzugefügt werden.
Eine weitere Idee wäre das Kommandozeilenwerkzeug so zu erweitern, dass mehrere Begriffe gesucht werden können.

4 Reflektion

Zu Beginn der Projektbearbeitung habe ich mich mit der Literatur beschäftigt, um die Algorithmen zu verstehen und mich für einen zu entscheiden. Nachdem ich den Finite-State-Matcher nachvollzogen hatte, habe ich mich schnell dafür entschieden diesen zu implementieren, da ich (auch wegen des Pseudocodes) direkt eine sehr klare Vorstellung

von der Implementierung hatte.
Als Erstes habe ich die Algorithmen implementiert. Für den Aufbau der StringMatcher-Klasse habe ich mich an der im Kurs behandelten Tokenizer-Klasse orientiert. Dieses Konzept ließ sich sehr gut auf das Projekt übertragen.
Ein paar Schwierigkeiten hatte ich mit dem Finite-State-Matcher-Algorithmus. Um den genauen Ablauf des Algorithmus verstehen zu können, musste ich mehrmals handschriftlich den Ablauf anhand von Beispielen nachvollziehen.
Zum Testen des Codes habe ich unterschiedliche Inputs ausprobiert und bin dabei auf den Sonderfall des leeren Strings gestoßen. Ich habe mich dafür Entschieden diesen Fall in der StringMatcher-Klasse und im Kommandozeilenwerkzeug unterschiedlich zu bearbeiten.
Es schien mir im Falle der Matching-Methode der Klasse sinnvoll, dass jede Position des Texts zurückgegeben wird. Da der leere String ein Suffix von jedem String ist, wird auch die Position der Textlänge + 1 zurückgegeben. Somit ist der Matching-Prozess "richtig" und die Klasse kann besser weiterverwendet werden.
Dahingegen wird im Kommandozeilenwerkzeug ein leerer Muster-Input mit einem Print-Statement behandelt, welches darüber informiert, dass der Input nicht leer sein darf.
Beim Schreiben des Kommandozeilenwerkzeugs hat es mich außerdem einige Zeit gekostet click zu verstehen und sinnvoll anzuwenden. Generell habe ich aber mit allen Projektanteilen ähnlich viel Zeit verbracht.
Eine Sache, die ich jedes Mal wieder unterschätze ist wie viel Zeit die Dokumentation und Überarbeitung des Codes kosten kann. Das war nicht problematisch, da ich früh genug mit der Bearbeitung des Projekt begonnen habe, jedoch könnte in Zukunft mehr Zeit durch dokumentieren während der Entwicklung gespart werden.