PRO1-A Modulprojekt WiSe 2020/2021

Thema: String-Matching

Report

Name: Jia Jian Lee
Matrikelnummer: 803826

Abgabefrist: 22.03.2021

Inhaltsverzeichnis

1. Info

Der Code wurde auf Github in folgenden Repository hochgeladen.

Link: https://github.com/jjiajian/string-matching-algorithmus

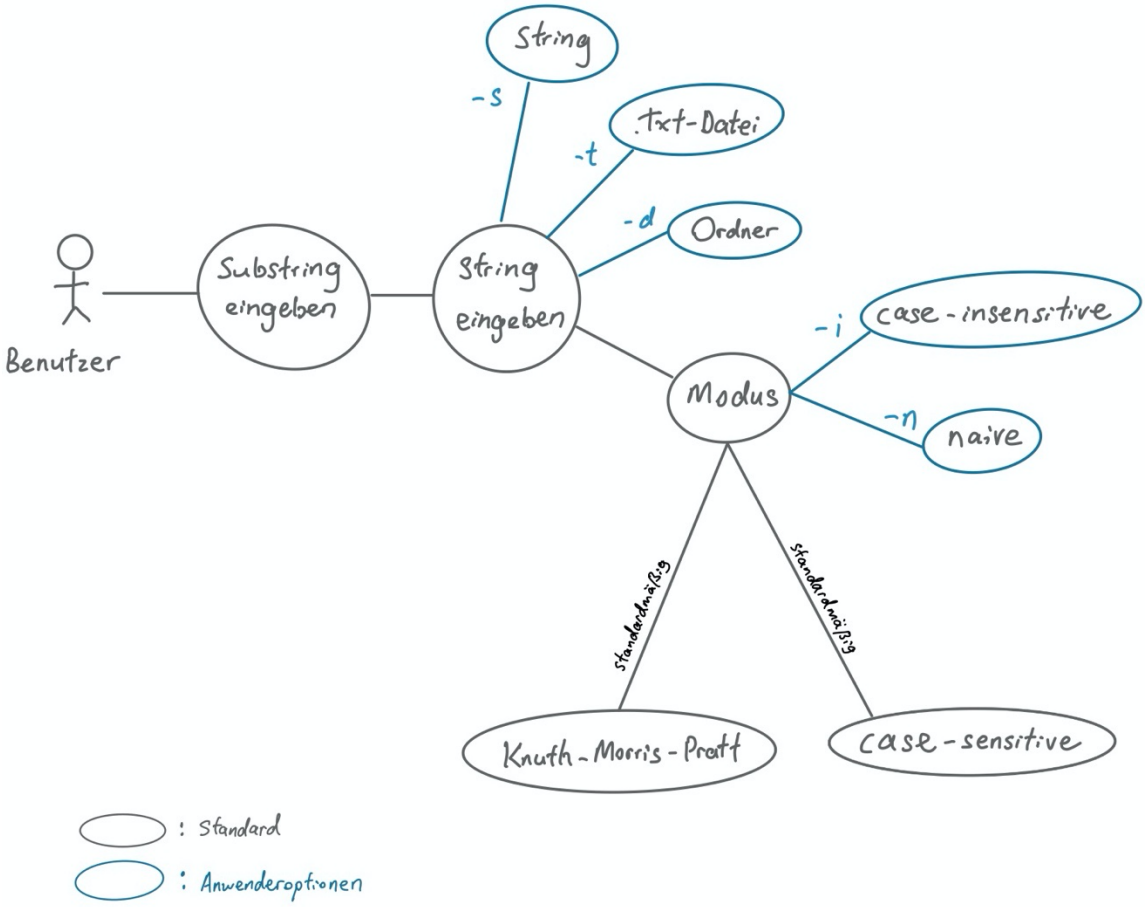
Username: jjiajian

2. Aufgabe

Ein String-Matching-Algorithmus, der von der Kommandozeile aufgerufen wird und die Indizes/Position(en) der Substring in einem Text zurückgibt. Zusätzlich zum naiven Algorithmus habe ich den Knuth-Morris-Pratt-Algorithmus gewählt und implementiert.

3. Programmteile

a. Programmstrukturdiagramm



b. Programmstruktur

Das Programm ist so konzipiert, dass es ein Substring gefolgt von einem Text aufnimmt, der ein String, eine Textdatei oder ein Ordner vom Benutzer sein kann (im Diagramm blau markiert). Wenn der Benutzer ein Ordner übergibt, prüft das Programm zunächst die Gültigkeit des Verzeichnispfads. Dann kann der Benutzer wählen, ob das Programm im Modus "case insensitive" oder im Modus "naive-Algorithmus" laufen soll. Wenn der Benutzer keine der Modusoptionen wählt, läuft das Programm im Modus "Groß-und Kleinschreibung/case sensitive" und verwendet die Knuth-Morris-Pratt-Algorithmus.

In dem Programm gibt es nur eine Klasse, nämlich StringMatcher. Die Klasse ist für den Hauptbetrieb des Programms verantwortlich und enthält folgende Funktionen:

- StringMatcher
    i. Parameters: *pattern (str), source (str)*

1. naive
    a. Parameters: *case_sensitive (bool)*
    b. Input: nimmt *pattern* und *source* auf
    c. Output: gibt Position von *pattern* im *source* zurück
    d. Pseudocode: Die Funktion verhält sich wie der Pseudocode. Es wurden Änderungen vorgenommen, so dass sie eine Liste zurückgibt, die erhebliche Erweiterungsmöglichkeiten bietet. „Groß- und Kleinbeschreibung“ Modus werden hier gecheckt.
2. lps_table
    a. Parameters: *pattern_length (int)*
    b. Input: nimmt *pattern* auf
    c. Output: gibt die Länge des LPS in einer Liste zurück
    d. Pseudocode: Die Funktion verhält sich wie der Pseudocode. Es wurden Änderungen vorgenommen, so dass sie das *pattern_length* als Argument aufnimmt.
3. kmp
    a. Parameters: *case_sensitive (bool)*
    b. Input: nimmt *pattern* und *source* auf
    c. Output: gibt Position von *pattern* im *source* zurück

d. Pseudocode: Die Funktion verhält sich wie der Pseudocode. Es wurden Änderungen vorgenommen, so dass sie eine Liste zurückgibt, die erhebliche Erweiterungsmöglichkeiten bietet. „Groß- und Kleinbeschreibung“ Modus werden hier gecheckt.

Es gibt eine weitere Funktion *dir_path*, die eine Verzeichnispfad aufnimmt und ihre Gültigkeit überprüft.

c. Beigefügte Beispieldateien zum Testen

dog2.txt, vaccine.txt

In sample_directory: berlin.txt, cat.txt, dog.txt, peloton.txt

d. Beispielaufrufe

Die Funktionsbeschreibungen entnehmen Sie bitte der README.md.

	Testbedingungen	Position/Index	Erwartungen erfüllt
1	"curious" -s 'curious and Curiouser!'	0	Ja
2	"curious" -s 'curious and Curiouser!' -i	0, 12	Ja
3	"curious" -s 'curious and Curiouser!' -n	0	Ja
4	"curious" -s 'curious and Curiouser!' -i -n	0, 12	Ja
5	"dog" -t "dog2.txt"	44, 46	Ja
6	"dog" -t "dog2.txt" -i	5, 44, 46	Ja
7	"dog" -t "dog2.txt" -n	44, 46	Ja
8	"dog" -t "dog2.txt" -i -n	5, 44, 46	Ja
9	"dog" -d "./sample_directory"	44, 82, 140, 171	Ja
10	"dog" -d "./sample_directory" -i	5, 44, 82, 140, 171	Ja
11	"dog" -d "./sample_directory" -n	44, 82, 140, 171	Ja
12	"dog" -d "./sample_directory" -i -n	5, 44, 82, 140, 171	Ja

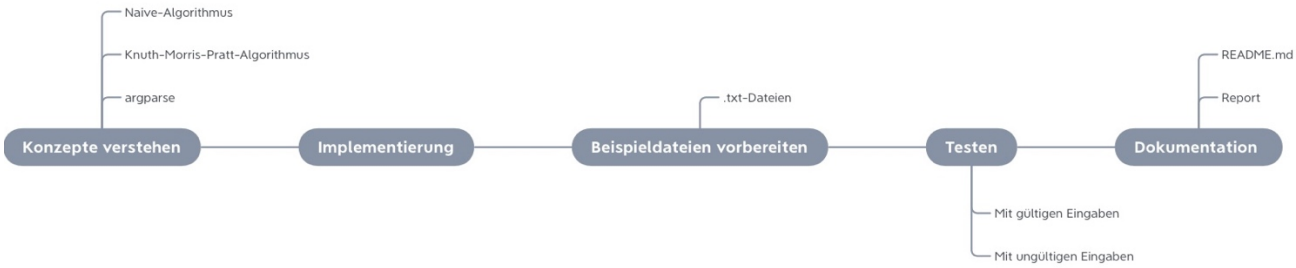
* Die Ergebnisse von *directory* (Ordner) enthalten nur die Ergebnisse aus dog.txt

e. Mögliche Erweiterungen des Projekts

Da die Positionen des Substrings in einer Liste anstelle eines Strings angegeben werden, kann der Benutzer die Positionen im Text verwenden z. B., um die Abstände zwischen den einzelnen Substrings zu finden und zu vergleichen. Eine weitere Möglichkeit wäre, das Programm so zu ändern, dass es mehr als ein Substring in einem Programmaufruf aufnimmt und durchsucht (momentan nimmt das Programm immer nur ein Substring auf einmal auf) und deren Positionen zurückgibt.

4. Reflektionen

a. Workflow und Planung



Ich habe meinem Workflow in fünf Phasen eingeteilt:

Phase 1: Konzepte verstehen

Mein erster Schritt in diesem Projekt ist es, das Konzept der Algorithmen zu verstehen, die ich später implementieren muss. Ich verbrachte ziemlich viel Zeit in dieser Phase, um sicherzustellen, dass ich vollständig verstehe, was ich tun muss, und um bei der Implementierungsphase etwas Zeit zu sparen. Außerdem habe ich die python Dokumentation von *argparse* aufwendig gelesen, die ebenfalls eine große Rolle in diesem Projekt spielt.

Phase 2: Implementierung

In dieser Phase habe ich die meiste Zeit verbracht, da sie meiner Meinung nach einer der kompliziertesten Teile des Projekts war. Da ich die *argparse library* zum ersten Mal verwendete, musste ich die Dokumentation immer wieder lesen und ging auf Websites wie Stackoverflow, um meine Unklarheiten zu lösen.

Phase 3: Beispieldateien vorbereiten

Diese Phase war der am wenigsten zeitaufwendige Teil des Projekts und alles lief gut. Von den sechs Beispieldateien sind drei von mir geschrieben und die anderen drei sind von Wikipedia übernommen (Der Link zum originalen Wikipedia-Beitrag ist in den jeweiligen txt-Dateien zu finden).

Phase 4: Testen

Mit der Erzeugung der ersten vollständigen Version des Codes kamen die Beispieldateien zum Einsatz. Nach einiger *Debugging* und *code-tuning* entsprachen die Testergebnisse meinen Erwartungen. Danach habe ich nochmal das Programm mit gültigen und ungültigen Eingaben getestet.

Phase 5: Dokumentation

Der letzte Teil des Projekts, der nicht der schwierigste war, aber genauso zeitaufwendig wie die erste Phase. Ich beschloss, zuerst das Readme zu erstellen, bevor ich das Projekt mit meinem Report abschloss.

b. Was wurden Sie beim nächsten Mal anders machen?

Insgesamt bin ich mit meinem derzeitigen Arbeitsablauf und der Planung zufrieden. Nur eine Sache wäre besser, wenn ich das Projekt früher beginnen würde, was mir mehr Zeit und Flexibilität beim Schreiben des Berichts lässt.

5. Literatur

- Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). Introduction to Algorithms. In McGraw-Hill. MIT Press. Chapter 34 String Matching.
- https://home.cse.ust.hk/~dekai/271/notes/L16/L16.pdf