Übersicht über die Funktionalität von AaS-Benchmark mit Beispielen

Inhaltsverzeichnis

1 Einleitung

AaS-Benchmark ist ein Tool, das es ermöglicht, die Ausführungszeiten von Algorithmen aus der Vorlesung *Algorithmen auf Sequenzen* zu messen. Dabei können verschiedenste Parameter gewählt werden, um so beispielsweise Zusammenhänge von Musterlängen und Ausführungszeiten zu erkennen. Die Algorithmen aus der Vorlesung sowie dieses Tool wurden dazu in der Programmiersprache *Rust* implementiert.

In dieser Übersicht sollen dem Leser die verschiedenen Funktionen von AaS-Benchmark nähergebracht und mit einigen Beispielen demonstriert werden.

2 Algorithmen

Für Aas-Benchmark wurden elf Algorithmen aus der Vorlesung implementiert. Diese lassen sich wie folgt in drei Kategorien aufteilen:

2.1 Algorithmen für einzelne Muster

Diese Gruppe von Algorithmen findet alle Vorkommen eines einzelnen Musters in einem Text. Implementiert sind in dieser Gruppe die folgenden Algorithmen:

- Naiver Algorithmus
- Knuth-Morris-Pratt
- Shift-And
- BNDM

2.2 Algorithmen für mehrere Muster

Diese Gruppe enthält Algorithmen, die alle Vorkommen von Mustern aus einer gegebenen Menge in einem Text finden. Sie unterscheiden sich also insofern von den Algorithmen der letzten Gruppe, als dass sie nicht mit nur nach einem einzelnen Muster suchen, sondern nach mehreren Mustern gleichzeitig.

Hier sind ein naiver Algorithmus, sowie der Aho-Corasick Algorithmus implementiert.

2.3 Fehlertolerante Algorithmen

2.4 Volltextindizes

3 Das Ausgabeformat

Wurde Aas-Benchmark korrekt ausgeführt, erhält man eine Ausgabe, die der Form der in Listing 1 gezeigten Daten entspricht. Diese Ausgabe hat das CSV Format, es handelt sich also um eine Tabelle, bei der die Spalten je Zeile durch Kommata getrennt sind. Die erste Zeile enthält dabei den Tabellenkopf.

Jede Zeile steht für eine einzelne Ausführung des Algorithmus, der in der ersten Spalte der Zeile gegeben ist. Die Spalten enthalten die folgenden Informationen:

- `algorithm`: Der ausgeführte Algorithmus
- `text_length`: Die Länge des Textes bei dieser Ausführung
- `pattern_length`: Die Länge des Musters bei dieser Ausführung
- `execution`: Die wievielte Ausführung dies ist
- `matches`: Wie viele Vorkommen des Musters bei dieser Ausführung im Text gefunden wurden

[1] Bei Algorithmen, die keine Vorbereitungszeit benötigen immer 0

- `prep_time_ms`: Die benötigte Vorbereitungszeit in Millisekunden[1]
- `time_ms`: Die benötigte Zeit, um den Algorithmus mit den gegebenen Parametern auszuführen

4 Nutzung

4.1 Wahl der Algorithmen

4.2 Mehrere Ausführungen

4.3 Textquellen

5 Beispiele

Beginnen wir mit einem einfachen Beispiel. Wir möchten testen, wie sich die Laufzeit des naiven Algorithmus für die Mustersuche bei steigender Musterlänge verhält.

Der Befehl für die Ausführung von AaS-Benchmark könnte dafür zum Beispiel wie folgt aussehen:

```
./aas-benchmark naive -t 100000000 -p 1..10
```

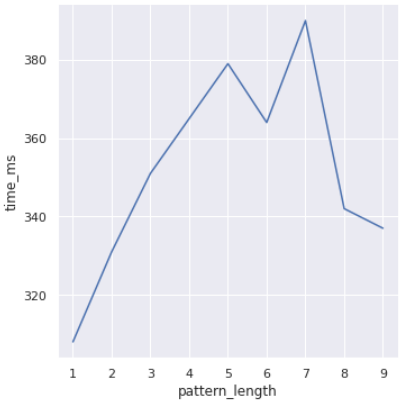

Abbildung 1: Laufzeit des naiven Algorithmus

Lässt man sich die Ausgabe dieses Befehls plotten, so erhält man das Diagramm in Abbildung 1. So kann man mit Aas-Benchmark das Verhalten der implementierten Algorithmen einfach betrachten und die verschiedenen Algorithmen miteinander vergleichen.

Als nächsten schauen wir uns ein einfaches Beispiel an, bei dem die Laufzeit zweier Algorithmen bei steigender Musterlänge geprüft wurde. Dazu wurde der folgende Befehl verwendet:

```
./aas-benchmark kmp,horspool -t 100000000 -p
1..10
```

Lässt man sich die Aufgabe erneut plotten, kann man nun erkennen, dass der *Horspool* Algorithmus mit steigender Musterlänge immer schneller ausgeführt werden kann, während die Laufzeit des *Shift-And* Algorithmus relativ konstant bleibt.

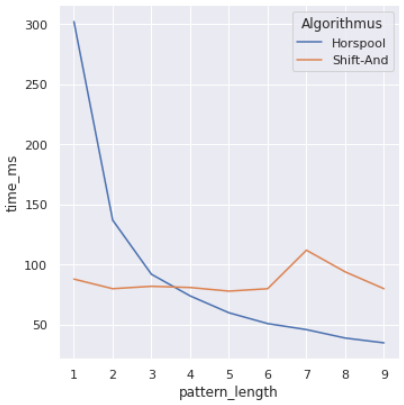

Abbildung 2: Laufzeit des naiven Algorithmus

```
algorithm,text_length,pattern_length,execution,matches,prep_time_ms,time_ms
Naive,1000000,1,0,4082,0,90
Naive,1000000,1,1,4082,0,90
Naive,1000000,2,0,14,0,90
Naive,1000000,2,1,14,0,90
Horspool,1000000,1,0,4082,0,79
Horspool,1000000,1,1,4082,0,79
Horspool,1000000,2,0,14,0,15
Horspool,1000000,2,1,14,0,15
KMP,1000000,1,0,4082,0,59
KMP,1000000,1,1,4082,0,59
KMP,1000000,2,0,14,0,56
KMP,1000000,2,1,14,0,56
```

Listing 1: Beispiel des Ausgabeformates

test