# Bonus Problem

Benedikt Sosnowksy, Daniel Ulrich, Kharald Anager, Marcel Sinn

May 29, 2022

## 1 Einleitung

Im folgenden Bericht werden wir erklären, wie sich Mustererkennungsalgorithmen unterscheiden. Dafür werden wir insbesondere auf brute-force, KMP und Boyer-Moore eingehen und verschiedene Vor- und Nachteile aufzeigen.

Das Problem in unserem Beispiel der Mustererkennung ist, dass wir einen bestimmten String in Form eines Textes der Länge n haben und innerhalb dieses Strings einen Muster-String suchen, welcher Teil des Textes sein soll. Unsere Mustererkennungsalgorithmen haben nun die Aufgabe, möglichst schnell und effizient den gesuchten String zu finden. Diese Mustererkennung wird aktuell beispielsweise in PDFs oder verschiedenen Websites angewendet, wodurch man die Seite nach bestimmten Wörtern durchsuchen kann. Wie genau diese suche funktioniert und was die einzelnen Mustererkennungsalgorithmen machen, wird in den folgenden Teilen thematisiert.

## 2 Algorithmen

### 2.1 Brute-Force

Das Brute-Force-Entwurfsmuster für Algorithmen ist eine leistungsstarke Technik für den Entwurf von Algorithmen, wenn wir etwas suchen oder eine Funktion optimieren wollen. Bei der Anwendung dieser Technik in einer allgemeinen Situation werden normalerweise möglichen Konfigurationen der beteiligten Eingaben aufgezählt und die beste von allen dieser aufgezählten Konfigurationen ausgewählt.

#### 2.1.1 Laufzeit

Die Laufzeit des Brute-Force-Mustervergleichs ist im schlimmsten Fall nicht gut, da wir für jeden Kandidatenindex in T bis zu m Zeichenvergleiche durchführen können, um herauszufinden, dass P im aktuellen Index nicht mit T übereinstimmt. Die äußere for-Schleife wird höchstens n-m+1 Mal und die innere while-Schleife höchstens m Mal ausgeführt. Somit ist die Worst-Case Zeitkomplexität der Brute-Force-Methode ist also O(nm).

## 2.2 Knuth-Morris-Pratt

Der Knuth-Morris-Pratt (oder "KMP") Algorithmus vermeidet die Informationsverschwendung und erreicht dabei eine Zeitkomplexität von O(n+m), was asymptotisch optimal ist. Das heißt, dass im schlimmsten Fall jeder Mustervergleichsalgorithmus alle Zeichen des Textes und alle Zeichen des Musters mindestens einmal untersuchen. Die Hauptidee des KMP-Algorithmus ist die Vorberechnung von Selbstüberschneidungen zwischen Teilen des Musters vorzuberechnen, so dass wir, wenn eine Fehlübereinstimmung an einer Stelle auftritt, wissen wir sofort, um wie viel das Muster maximal verschoben werden muss bevor die Suche fortgesetzt wird.

### 2.2.1 Fehlerfunktion

Um den KMP-Algorithmus zu implementieren, berechnen wir im Voraus eine Fehlerfunktion, f, die die richtige Verschiebung von P bei einem fehlgeschlagenen Vergleich angibt. Konkret ist die Fehlerfunktion f (k) definiert als die Länge des längsten Präfixes von P, das ein Suffix von P[1:k+1] ist (beachten Sie, dass wir P[0] hier nicht berücksichtigt haben, da wir mindestens eine Einheit). Intuitiv ausgedrückt: Wenn wir eine Nichtübereinstimmung beim Zeichen P[k+1] finden, sagt uns die Funktion f (k) an, wie viele der unmittelbar vorangehenden Zeichen wiederverwendet werden können, um das Muster neu zu starten.

## 2.3 Boyer – Moore Algorithmus

Der Boyer-Moore Mustererkennungsalgorithmus kann Vergleiche zwischen einem Muster P und einem Bruchteil der Buchstaben in einem Text T vermeiden. In diesem Teil wird eine vereinfachte Version des Original Algorithmus von Boyer und Moore beschrieben. Die Hauptidee des Boyer-Moore Algorithmus ist die Verbesserung der Laufzeit des brute-force Algorithmus, indem zwei möglichst zeitsparende Heuristiken hinzugefügt werden.

Die erste Heuristik ist die Looking-Glass Heuristic. Bei dieser Methode wird auf eine mögliche Platzierung von P gegen T getestet. Der Vergleich wird mit dem Ende von P begonnen und läuft rückwärts an den Beginn von P.

Die zweite Heuristik ist die Character-Jump Heuristic. Während eine mögliche Platzierung von P in T getestet wird, wird eine Nichtübereinstimmung der Buchstaben T[i]=c mit dem korrespondierenden Muster P[k] wie folgt behandelt. Falls c nicht in P vorkommt, wird P komplett bis an T[i] vorbei geschoben. Andernfalls wird P solange verschoben, bis eine Übereinstimmung der Buchstaben c in P in T[i] auftritt.

# 3 Textbeschreibung

In dieser Untersuchung wurde ein dutzende Menge von Textdateien genutzt:

1. bigfile.txt ist eine 10.000.000 Zeichen große Datei an zufälligen Buchstaben und Ziffern.
2. shakespeare.txt enthält die gesammelten Werke desselben in altem Englisch, mit einigen Kommentaren an Beginn und Ende. Die gesamte Länge der Datei beträgt 5,7 Millionen Zeichen.
3. law.txt enthält die Richtlinien zur Durchsuchung und Beschlagnahmung von Computern des US Department of Justice. Das Dokument ist in modernen Englisch abgefasst und enthält ca. 300.000 Zeichen
4. In LI.txt ist ein computergenerierter, pseudolateinischer Text enthalten, der 750.000 Zeichen umfasst.
5. LI-large.txt ist im selben Stil wie LI.txt geschrieben und hat eine Länge von 2.000.000 Zeichen

# 4 Die Präsentation der Ergebnisse

| | bigfile | law | LI | LI-Large | Shakespear |
|---|---|---|---|---|---|
| brute-force | 746.259 ms | 745.166 ms | 1338.482 ms | 1279.951462 ms | 1279.49429 ms |
| KMP | 675.9925 ms | 1123.231 ms | 1160.338 ms | 1162.29496 ms | 1157.80158 ms |
| Boyer-Moore | 185.786 ms | 330.1709 ms | 288.56 ms | 288.25984 ms | 291.605437 ms |