Dang Quynh Tram Nguyen

Studiengang: Bioinformatik

Matrikelnummer: 5311561

ProJEKT 1: Patternmatching

[Untertitel des Dokuments]

Inhaltverzeichnis

[1. Einleitung 2](#_Toc102600048)

[2. Algorithmen 2](#_Toc102600049)

[1. SimpleSearch Algorithmus 2](#_Toc102600050)

[2. Horspool Algorithmus 3](#_Toc102600051)

[3. Beispiele 4](#_Toc102600052)

[1. Text „Schneewittchen“ 4](#_Toc102600053)

[2. Text „ABAB“ 6](#_Toc102600054)

[4. Diskussion 7](#_Toc102600055)

# Einleitung

In Molekularbiologie beschäftigt man sich regelmäßig mit dem zentralen Dogma, das im Allgemeinen den Erbinformationsfluss vom DNA zu Proteinen besagt. Um das genauer zu verstehen werden 3 Hauptsynthesen dabei untersucht: Synthese von DNA zu DNA (Replikation), von DNA zu RNA (Transkription) und von RNA zu Proteinen (Translation). Bevor solche biologischen Prozesse starten, müssen bestimmten DNA- oder RNA-Abschnitte erkannt werden, an den bspw. die synthesefunktionellen Enzyme, Faktoren, usw. binden können. Die Frage wird gestellt, wie diese Abschnitte auf einem langen Erbinformationsstrang gefunden werden. Da folgt das Thema dieses Projekts: „Mustererkennung“, welche in diesem Fall eine Methode bzw. ein Algorithmus ist, einen bestimmten Pattern auf einer Sequenz zu suchen.

In diesem Projekt wird zunächst der Ablauf zweier Algorithmen „SimpleSearch“ und „Horspool“ mit Hilfe der Programmiersprache Python grundsätzlich dargestellt. Danach folgt die Anwendung eines der zwei Algorithmen auf einer großen Datei als Beispiel. Demnächst werden die zwei Algorithmen verglichen. Und zum Schluss kommt eine Zusammenfassung dieses ganzen Projekts.

# Algorithmen

## SimpleSearch Algorithmus

Wie oben erwähnt, geht es in diesem Algorithmus darum, Pattern auf einem bekannten Sequenz herauszufinden. Deutlicher ergibt er die Anzahl aller Treffer und die genauen Positionen, wo sie auf Sequenz liegen. Im ersten Blick kann man einen einfachen Modell der Methode vorstellen, wobei das Pattern auf jeder Position auf der Sequenz in einer Richtung bis zum Ende verschoben und mit dem auf der gleichen Position liegenden Teilsequenz verglichen wird. Das heißt, dass der Algorithmus alle Fälle versucht und am Ende die möglichste Anzahl der Hits zurückgibt. Dies erklärt den „SimpleSearch“ Algorithmus.

Demnächst wird die Implementierung dieses Algorithmus in Detail beschrieben. Die Funktion SimpleSearch nutzt eine for-Schleife für die zu vergleichende Position pos, die in einer bestimmten Grenze liegt, damit Teilsequenz teilstring die Länge der Sequenz nicht überschreitet. Wir fangen an der ersten Postion der Sequenz an. Da wird die Funktion Match\_SS angewendet, wobei es geprüft wird, ob die  Charakter in derselben Stelle der Teilsequenz und des Patterns gleich sind. Wenn sie miteinander übereinstimmen (Match\_SS ergibt True), wird die Position in der vordeklarierten Liste match gespeichert und die Anzahl der Vergleiche count werden gezählt. Danach wird die Position pos auf eine verschoben, an dieser Position nimmt die Teilsequenz einen neuen Wert und wird mit dem Pattern wieder verglichen. Der Algorithmus läuft weiter wieder so, bis die letzte Position der Teilsequenz dieselbe der Sequenz ist. Am Ende der Funktion wird die Liste der Treffer zurückgegeben.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Im Folgenden ist ein Beispiel dazu. Seien Pattern p als „tram“ und Sequenz s als „dangquynhtramnguyen“. Wenden wir das implementierten Skript an, bekommen wir folgendes Erbgebnis:





Der Pattern und die Sequenz werden zu Listen umgewandelt, dann hat jeder Charakter ein Index und diese werden in der Funktion Match\_SS geprüft. Die Trefferanzahl treffer entspricht die Länge der Liste der Positionen.

## Horspool Algorithmus

Die Laufzeit des “SimpleSearch” Algorithmus mit der großen Sequenz ist aber ziemlich lang. Deswegen wird dieser Algorithmus später optimiert, damit sein Ablauf vielmals beschleunigt wird. Es ist dann als “Horspool” Algorithmus genannt. Bei diesem Algorithmus wird die zu vergleichende Position auf der Sequenz, wo die Teilsequenz gefunden ist, auf bestimmten Stellen verschoben, anstatt auf allen Stellen wie in “SimpleSearch” Algorithmus. Die Schritten zum Vergleich bleiben behalten.

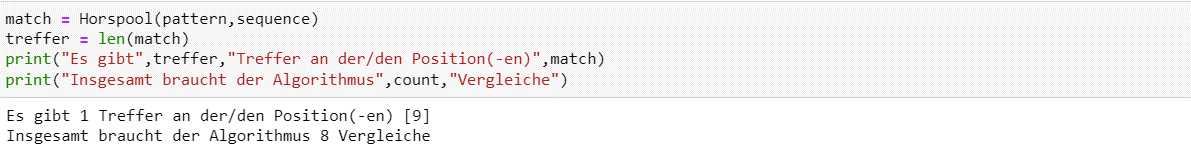
Zunächst beschreibe ich, wie dieser Algorithmus implementiert wird. Um die Verschiebung der  zu vergleichenden Position zu verdeutlichen, wird eine Shift-Tabelle (Verschiebung-Tabelle) unten einem Dictionary shift\_table erzeugt. Die Schlüssel von diesem Dictionary sind alle Charakter im angegebenen Pattern, außer dem letzten. Ihre entsprechenden Werte betragen die Differenz der Länge des Patterns und seine Position im Pattern. Welcher Charakter (als Schlüssel), der vielmals im Pattern getroffen ist, erhält als Wert das Ergebnis von der Subtraktion zwischen den Patternlänge und die Position im Pattern, die zuletzt gefunden wird.Wenn der Algorithmus läuft,  beginnt die zu vergleichende  Position an der ersten Position der Sequenz. Nach jedem Vergleich wird die Position auf den Betrag, was das Wert des letzten Charakters der Teilsequenz in der Shift-Tabelle entspricht, verschoben. Ist der letzter Charakter  kein Schlüssel in der Shift-Tabelle, nimmt der Betrag die Länge des Pattern.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Die Funktion Match\_H läuft ähnlich wie die Match\_SS vom oberen Algorithmus. Der Laufrichtung des Vergleichs startet aber an dem letzten Charakter und wird rückwärts fortgesetzt. Bei der Funktion werden die getätigten Vergleiche auch gezählt.

Ein Beispiel der Durchführung des Algorithmus wird (mit denselben Angaben wie “SimpleSearch” Algorithmus) wie folgt gegeben:



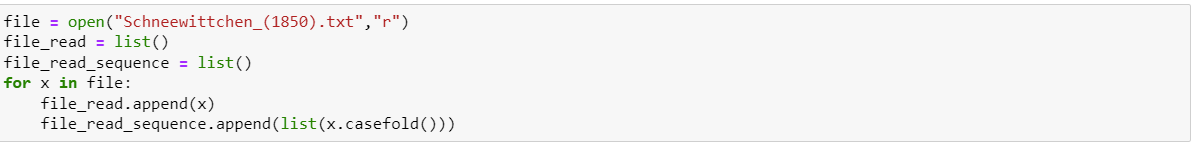
# Beispiele

2 folgende Beispiele erklären genauer, wie die Algorithmen bei den großen Dateien abläuft.

## Text „Schneewittchen“

Bei dem ersten Beispiel wird das Pattern „zwerg“ im Text „Schneewittchen“ gefunden und am Ende des Algorithmus sollten alle Positionen sowie die Anzahl der Treffer ergeben werden.

Zuerst wird der Text geöffnet und eine Liste, in der die Werte alle Zeilen im Text enthalten sind, wird erzeugt. Jeder dieser Werte ist eine Liste der Charakter derselben Zeile. Diese Charakterlisten gelten in den Algorithmen als die Sequenzen,worauf der angegebene Pattern gesucht wird.



Danach werden alle Charakter in jeder Liste geprüft, damit sich alle besondere Charakter wie Umlaut oder die Anführungszeichen in einer Python passenden Format umsetzen lassen. Dieser Schritt ist in der Funktion aendert\_umlaut\_und\_zeichen implementiert.



Weil jede Zeile als eine Sequenz gilt und die aufgerufene Funktion des Algorithmus endlich nur die Liste der gefundenen Positionen eine Zeile zurückgibt, ist ein Dictionary benötigt, um alle Listen zu sammeln. Dies wird in der Funktion dict\_positionen beschreibt. Die Zeilen, wo die Treffer liegen, sind die Schlüssel vom Dictionary und die Listen der genauen Position sind die Werte. Endlich wird der Dictionary zurückgegeben.

Für die Liste match\_position passt entweder die Funktion SimpleSearch oder Horspool. Hier wird Horspool benutzt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Der Pattern „zwerg“ wird im Text gefunden. Die Positionen, die Zeilen der Treffer und die Anzahl der Vergleiche werden ausgegeben. Das Ergebnis wird im Quellcode angezeigt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Es gibt noch ein Problem, dass alle Vorkommen von „zwerg“ im Text nicht zum Wort „zwerge“ gehören.

Die Idee dafür ist, dass alle Positionen von „zwerge“ im Dictionary von Treffer „zwerg“  gelöscht werden. Dafür wird ein Dictionary von Treffer „zwerge“, nämlich dict\_positionen\_zwerge auch erzeugt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ergebnis:

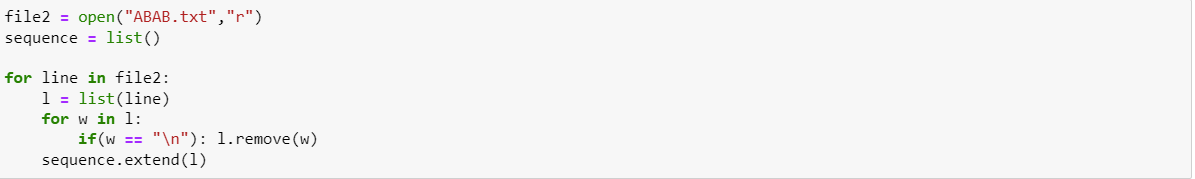
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Text „ABAB“

Das zweiter Beispiel zeigt die Suche nach 2 Pattern „ABA“ und „CCC“ im Text „ABAB“. Der Hauptpunkt in diesem Beispiel ist die gesamte Anzahl der Vergleiche in jedem Algorithmus.

Der Text wird zuerst geöffnet und eine Liste des Textes wie im ersten Beispiel mit dem Text „Schneewittchen“ erzeugt. Wichtig ist, dass alle Zeilenumbrüche „\n“ in der Liste entfernt werden, damit mehrere Vergleiche und Treffer gefunden werden. Damit kann man zwei Algorithmen vergleichen.



Mit dem Pattern „ABA“:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Mit dem Pattern „CCC“:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In zwei Versuche mit 2 Pattern kann man sehen, dass die Anzahl der Vergleiche vom *SimpleSearch* viel mehr als die von *Horspool* Algorithmus ist.

# Diskussion

Im zweiten Beispiel gibt es einen großen Unterschied zwischen der Anzahl der Vergleiche zweier Algorithmen. Das spiegelt die Laufzeit der Algorithmen ab. Der “Horspool” Algorithmus läuft deutlich schneller als der “SimpleSearch”, weil die zu vergleichenden Positionen im “Horspool” besser gewählt werden.