Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: 6th of November 2017
Manuscript Category

Algorithmische Bioinformatik

Projekt 1

Paul Vogler, Tobias Mechura & Franziska Rau

Abstract

Projekt 1 des Moduls Algorithmische Bioinformatik (Implementierung von k-mer Suchalgorithmen)

1 Einführung

Die Replikation beginnt in einer Region, die Replikationsursprung genannt wird (oriC). Sie wird von der DNA Polymerase ausgeführt und initiiert durch das Protein DnaA, das typischerweise an ein 9-mer bindet, das als DnaA Box bezeichnet wird. Damit DnaA die DnaA Box erkennt, muss die DnaA Box gehäuft im Bereich um den oriC vorkommen.

2 Ansatz

Das Ziel des Projektes ist es, aus 500er Fenstern um die oriC der Organismen Vibrio Cholerae und Thermotoga Petrophila die häufigsten 3- & 9-mere herauszufiltern. Dazu werden die Algorithmen FrequentWords, FastFrequentWords und FrequentWordsBySorting implementiert. Anschließend wurden deren Laufzeiten in Abhängigkeit von der Sequenzlänge und der Größe verglichen.
Zur Umsetzung der Algorithmen wurde die Programmiersprache Java eingesetzt, denn so können die Algorithmen auf einfache Weise auch in spätere Projekte integriert werden.

3 Methoden

Mit den Pseudocodes aus den Vorlesungsfolien konnten ausführbare Funktionen erstellt werden.

FrequentWords zählt für jeden k-mer im Text die Häufigkeit und gibt die häufigsten k-mere aus.
Laufzeit: $O(|Text|^2 * k)$

FastFrequentWords erstellt alle möglichen 4^k k-mere und konvertiert sie zu Zahlen zwischen 0 und 4^k - 1. Für jedes k-mer wird die Häufigkeit gezählt und die am meisten vorkommenden k-mere werden ausgegeben.
Laufzeit: $O(4^k + |Text| * k)$

FrequentWordsBySorting funktioniert ähnlich wie FastFrequentWords, mit dem Unterschied, dass nur für die auch wirklich auftretenden k-mere die Anzahl an Vorkommen gezählt wird, um die häufigsten k-mere zu ermitteln.
Laufzeit: $O(n * log(n) * k)$

4 Diskussion

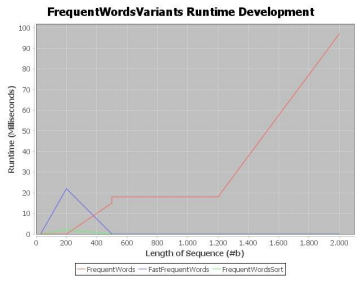
Ausgabe füer k = 3:

```
Algorithms for the oriC of V. Cholerae Chr.1:

[AAA]
The FrequentWords Algorithm took 5 Milliseconds to find a solution.

[AAA]
The FastFrequentWords Algorithm took 0 Milliseconds to find a solution.

[AAA]
The FrequentWordsSort Algorithm took 1 Milliseconds to find a solution.


Algorithms for the oriC of V. Cholerae Chr.2:

[AAA]
The FrequentWords Algorithm took 7 Milliseconds to find a solution.

[AAA]
The FastFrequentWords Algorithm took 0 Milliseconds to find a solution.

[AAA]
The FrequentWordsSort Algorithm took 1 Milliseconds to find a solution.


Algorithms for the oriC of T. Petrophila:

[GAA]
The FrequentWords Algorithm took 6 Milliseconds to find a solution.

[GAA]
The FastFrequentWords Algorithm took 0 Milliseconds to find a solution.

[GAA]
The FrequentWordsSort Algorithm took 1 Milliseconds to find a solution.
```

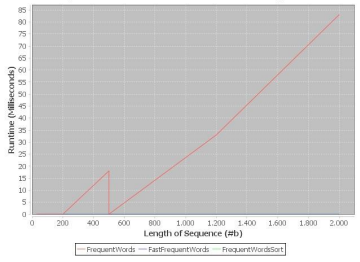
Ausgabe füer k = 9:

```
Algorithms for the oriC of T. Petrophila:

[GGAATCTTT]
The FrequentWords Algorithm took 5 Milliseconds to find a solution.

[GGAATCTTT]
The FastFrequentWords Algorithm took 4 Milliseconds to find a solution.

[GGAATCTTT]
The FrequentWordsSort Algorithm took 1 Milliseconds to find a solution.
```

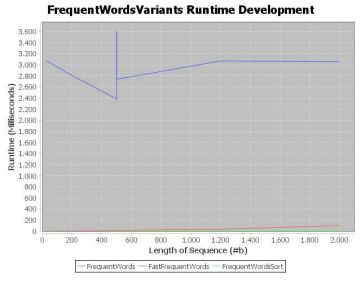
```
Algorithms for the oriC of V. Cholerae Chr.1:

[AGGGCTTTA]
The FrequentWords Algorithm took 6 Milliseconds to find a solution.

[AGGGCTTTA]
The FastFrequentWords Algorithm took 5 Milliseconds to find a solution.

[AGGGCTTTA]
The FrequentWordsSort Algorithm took 1 Milliseconds to find a solution.


Algorithms for the oriC of V. Cholerae Chr.2:

[AGGGCTTTA]
The FrequentWords Algorithm took 5 Milliseconds to find a solution.

[AGGGCTTTA]
The FastFrequentWords Algorithm took 4 Milliseconds to find a solution.

[AGGGCTTTA]
The FrequentWordsSort Algorithm took 0 Milliseconds to find a solution.
```



k = 3



k = 9



k = 13

Fig. 1. Algorithmenlaufzeit in Abhängigkeit der Länge der Inputsequenz und verschiedener k

4.1 Laufzeitanalyse

In der nebenstehenden Grafik 1 ist das Ergebnis, das in dieser Form zu erwarten war, zu sehen.

Während der FrequentWords-Algoritmus für kleine k eine erheblich längere Laufzeit als der FastFrequentWords-Algorithmus hat, erhöht sich seine Laufzeit fast gar nicht bei größer gewähltem k. Der FastFrequentWords-Algorithmus jedoch ist für große k ungeeignet (wegen der Erstellung aller 4^k k-mere) und braucht somit extrem lange, um das Ergebnis zu finden. Auch der FrequentWordsBySorting-Algorithmus entsprach den Annahmen und ist für beliebig gewählte k der schnellste der drei Algorithmen.

4.2 Probleme

Im Allgemeinen gab es während des Projektes keine großen Schwierigkeiten. Einzig die Suche nach den Genomen von T. Petrophila und V. Cholerae gestaltete sich als umständlich, da die Struktur der Webseite https://www.ncbi.nlm.nih.gov/ zunächst ungewohnt war.

4.3 Arbeitsaufteilung

Größtenteils wurde für das Projekt gemeinsam an einem PC gearbeitet. Lediglich kleinere Aufgaben wurden vereinzelt alleine bearbeitet.

5 Fazit

Das Projektziel wurde erfüllt (alle drei Algorithmen sind implementiert worden und liefern ein korrektes Ergebnis).

Wie bereits angenommen, ist der FrequentWordsBySorting-Algorithmus der schnellste der Algorithmen, FastFrequentWords ist nur schnell bei der Suche nach k-meren mit kleinem k und FrequentWords hat im Worst-Case-Szenario eine quadratische Laufzeit.