Praktikum Data-Mining und Sequenzanalyse
Aufgaben zur exakten Suche

Markus Fleischauer

Allgemeine Projektanforderungen

- Versionskontrolle mit Git
- Projektmanagement mit Gradle
- Implementierung der Klassen mit den entsprechenden Algorithmen
- API Dokumentation mit javadoc
- Kommandozeilenprogramm (CLI) inklusive Hilfe (–help)

1 Implementierung - Klassenstruktur

- Package: *sequences.model*
  - Definieren Sie die API für alle folgenden Sequenztypen mit dem Interface **Sequence**: [`getSequence()`, `getName()`, `charAt(int)`, `indexOf(char)`,`alphabetSize()`,`getAlphabetString()`]
  - `indexOf(char)` gibt die Position des Zeichens im Alphabet zurück.
  - Klassen fuer die folgenden Sequenztypen sollen implementiert werden. Versuchen Sie dabei redundanten Quelltext zu vermeiden (Abstrakte Klasse?). AsciiSequence nutzt ASCII als Alphabet, die beiden anderen Klassen die jeweiligen biol. Alphabete.
    - Klasse: **AsciiSequence** implementiert **Sequence**
    - Klasse: **NucleotideSequence** implementiert **Sequence** [`translate(int)`, `complement()`]
      - `translate(int)` translatiert eine **NucleotideSequence**, das Leseraster wird als `int` übergeben. (besser: `enum`)
      - `complement()` bildet die Komplementär-Sequenz zu dieser Sequenz.
    - Klasse: **AminoacidSequence** implementiert **Sequence**
- Package: *exactSearch.algo*
  Die Klassen nehmen Sequenz und Muster entgegen und erlauben die Suche mit der Methode `next()`, die die Position des nächsten Vorkommens des Musters in der Sequenz liefert.

  - API durch Interface **SearchAlgorithm** definieren:
    `[setPattern(Sequence),setText(Sequence),next(),count()]`
  - Die folgende Klassen überschreiben/implementieren `next()`, so dass die entsprechenden Suchalgorithmen verwendet werden. Versuchen Sie dabei redundanten Quelltext zu vermeiden (Abstrakte Klasse?):
    * Klasse: **NaiveSearch** implementiert **SearchAlgorithm**
    * Klasse: **KMPSearch** implementiert **SearchAlgorithm**
    * Klasse: **BMSearch** implementiert **SearchAlgorithm**

- Package: *exactSearch.io*
  Reader und Writer fuer *FASTA* Dateien
  - **FastaReader** Sequenz aus Datei im *FASTA* Format lesen
  - **FastaWriter** Sequenz in Datei im *FASTA* Format ausgeben

```
> Name
CTAGAGTGTAGTCATGTCAGTGCAGC
GTAGCGCGGTTTAAGTCT
```

- Package: *exactSearch.cli*
  Paket für den Kommandozeilenparser und die main Methode:
  - Klasse mit `main()` Methode
  - Parameter für Text- und Muster-Datei
  - Parameter für Typ der Sequenzen und Suchalgorithmus

2 Validierung der Algorithmen - Tests

Gewährleistet die Korrektheit eurer Algorithmen durch die Implementierung folgender (Unit)Tests. *Hinweis: Die frühzeitige Implementierung von Tests erleichtert den Debuggingprozess erheblich.*

1. Die folgenden Tests sollen für alle 3 (Naiv, KMP, BM) Implemetierungen der exakten Textsuche zur Verfuegung stehen.
   - Suche eines *nicht* leeren Patterns in einem leeren Text
   - Suche eines leeren Patterns in einem *nicht* leeren Text
   - Suche mit Vorkommen eines Patterns am Anfang des Textes
   - Suche mit Vorkommen eines Patterns am Ende des Textes
   - Suche mit mehreren *disjunkten* Vorkommen eines Patterns im Text
   - Suche mit mehreren *überlappenden* Vorkommen eines Patterns im Text
2. Testen sie eine zuffällig erzeugte DNA Sequenz der laenge 5000 gegen eine zuffällig erzeugtes Pattern der länge 5. Validieren Sie, dass alle 3 Implemetierungen (Naiv, KMP, BM) der exakten Textsuche das gleiche Ergebnis liefern.

3 Anwendung der Implemetierung

Allgemeine Hinweise:

- Die folgenden Aufgaben, sollen mit Hilfe ihres Kommandozeilenprogramms gelöst werden.
- Geben sie stets die Ausgefürten Befehle an, sodass die Ergebnisse reproduzierbar und nachvollziehbar sind.
- Geben sie das verwendete Testsystem an (CPU, RAM, HDD/SSD, OS(-Version), Java-Version, JVM Parameter).
- Erläutern Sie ihre Implementierung. Gab es Schwierigkeiten? Gibt es noch Fehler? Was könnte verbessert werden?

Daten: Folgende Testdaten liegen im `FASTA`-Format vor [1]

- `T.thermophilus.fasta`: komplettes Genom von T. thermophilus
- `SwinepoxVirus.fasta`: Genom des Schweinepockenvirus
- `CytochromeC.fasta`: Cytochrom c Oxidase-Gen von T. thermophilus
- `DNAPolyIIIalpha.fasta`: DNA-Polymerase III von T. thermophilus

Aufgaben:

1. Vergleicht die Algorithmen für die folgenden Aufgaben untereinander. Erweitert dazu eure Implementierung um das Zählen von Vergleichsoperationen und das Messen der Laufzeit einzelner Algorithmen. Zählt die Vorkommen jedes Patterns.
   (a) Sucht im Virusgenom nach `GTATTA`, `TTTCGAAA`, `AAATTGACG`. An welchen Positionen kommen sie vor?
   (b) Durchsucht das Bakteriengenom nach `GAATTC`, `GGATCC` und `ATTTAAAT`. Was ist das besondere an diesen drei Sequenzen? Warum sind sie biologisch relevant?
   (c) An welcher Position im Genom befindet sich das CytcOxidase-Gen?
   (d) Ist das Protein in einem der sechs Leseraster im Thermophilus-Genom zu finden? Ließe sich ein Protein auch im menschlichen Genom so leicht finden?
   (e) Konstruiert Beispiele, in denen die Algorithmen jeweils besonders gut, besonders schlecht geeignet sind.
2. Welche Schlüsse zieht ihr aus dem Vergleich? Für welche Anwendung in der Bioinformatik kann man welchen Algorithmus empfehlen?

[1] aus NCBI Nucleotide und Uniprot