Institut für Informatik
Praktische Informatik und Bioinformatik
Prof. Dr. Ralf Zimmer

Wintersemester 2018/19

Übungen zum Bioinformatik-Tutorium

Blatt 9

Termin: Dienstag, 18.12.2018, 11 Uhr

1. Protein ID aus dem Fasta-Header extrahieren

Momentan speichert dein `Proteome`-Konstruktor für jedes Protein den gesamten Header (Zeilen die mit `>` beginnen) als ID. Modifiziere deinen Code so, dass nur die tatsächliche ID aus dem Header als Protein-ID abgespeichert wird.

In unserem Beispielheader (s.u.) ist `C09B8.7e` die ID. In unserem Fall ist die ID immer an der selben Position im Header. Überlege dir, wie du mit einem Regex-Ausdruck nur die ID matchst. Tipp: Nutze einen Regex-Tester wie `https://regexr.com/` um es auf ein paar Beispielen zu testen.

```
>C09B8.7e pep:known chromosome:WBcel235:X:6044312:6046161:-1 gene:C09B8.7 transcript: (...)
```

Wenn du das Pattern gefunden hast, überlege dir wo im Code beim Einlesen der Fasta-Datei die ID gespeichert wird. Das ist dein Ansatzpunkt um mit Hilfe des Patterns die ID zu extrahieren.

Ob du eine zusätzliche Methode schreibst die die ID aus dem Header extrahiert, oder ob du dies an der Stelle an der die ID eingelesen wird erledigst, bleibt dir überlassen.

Der Konstruktor bleibt hier nahezu identisch zu der vorherigen Version, nur an den beiden kommentierten Stellen wurden die besprochenen Regex-Methoden verwendet:

```
public Proteome(String fastaFile) {
    proteins = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader(fastaFile))) {
        String line;
        StringBuilder sb = new StringBuilder();
        String id = "";
        // Beispielsweise hier sein Pattern kompilieren
        Pattern p = Pattern.compile(">([\\S]+)");
        while ((line = br.readLine()) != null) {
            if (line.startsWith(">")) {
                if (sb.length() > 0) {
                    proteins.add(new Protein(id, sb.toString()));
                }
                Matcher m = p.matcher(line);
                // Und hier dann den Matcher die ID finden lassen
                if (m.find()) {
                    id = m.group(1);
                } else {
                    System.out.println("Protein hat keine ID");
                }
                sb = new StringBuilder();
            } else {
                sb.append(line);
            }
        }
        if (id.length() > 0) {
            proteins.add(new Protein(id, sb.toString()));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

2. Prosite-Pattern

Erweitere zunächst deine Klasse `Proteome` um eine Methode `searchPattern(Pattern pattern)`, um in allen geladenen Sequenzen nach Pattern zu suchen. Die Methode soll für jeden gefundenen Match eine Zeile mit folgenden Informationen im Terminal ausgeben:

- Protein-ID
- gematchtes Sequenzstück
- Position in der Protein-Sequenz

Die `for (Protein p : proteins)` Struktur ließe sich auch durch einen einfachen Loop über die `ArrayList` ersetzen:

```
public void searchPattern(Pattern pattern) {
    for (Protein p : proteins) {
        Matcher m = pattern.matcher(p.getSequence());
        while (m.find()) {
          System.out.println(p.getId() + " " + m.group(0) +
          " " + m.start(0) + "-" + m.end(0));
        }
    }
}
```

Informiere dich dann auf der Prosite-Website (`http://prosite.expasy.org/prosuser.html#conv_pa`) über Prosite-Pattern. Übersetze folgende Prosite-Pattern in reguläre Ausdrücke und teste `searchPattern` mit `C_elegans.pep.all.fa` aus dem Materialordner:

(a) `C-[VILMA]-x(5,6)-C-[DNH]-x(3)-[DENQHT]-C-x(3,4)-[STADEW]-[DEH]-[DE]-x(1,5)-C`

(b) `[LIVM]-[NST]-{T}-x-C-[SAGLI]-[ST]-[SAG]-[LIVMFYNS]-x-[STAG]-[LIVM]-x(6)-[LIVM]`

Pattern (a) sollte 154 mal matchen, Pattern (b) 12 mal.

```
C-[VILMA]-x(5,6)-C-[DNH]-x(3)-[DENQHT]-C-x(3,4)-[STADEW]-[DEH]-[DE]-x(1,5)-C
[LIVM]-[NST]-{T}-x-C-[SAGLI]-[ST]-[SAG]-[LIVMFYNS]-x-[STAG]-[LIVM]-x(6)-[LIVM]
```