**Arbeitsgruppe für Genominformatik** **Universität Hamburg**
**Stefan Kurtz** **Zentrum für Bioinformatik**

# Genominformatik
# Sommersemester 2017
# Übungen zur Vorlesung: Ausgabe am 13.06.2017

Punkteverteilung: Aufgabe 5.1: 4 Punkte, Aufgabe 5.2: 4 Punkte, Aufgabe 5.3: 2 Punkte.

Abgabe bis zum 22.06.2017, 23:59 Uhr.

**Aufgabe 5.1** Implementieren Sie in C auf Basis der Software-Bibliothek `GtSuffixtree` (siehe Übungsaufgabe 4.3.) in der Datei `stree-approx.c` eine Funktion

```
void stree_approx_pattern_match(const GtStree *stree,
                                const GtUchar *pattern,
                                GtUword len,
                                GtUword differences)
```

die den Algorithmus zur approximativen Mustersuche in Suffixbäumen implementiert. Dabei ist

- `stree` der Verweis auf den Suffixbaum,
- `pattern` das Muster der Länge `len` und
- `differences` die Anzahl von erlaubten Fehlern. Es muss `differences>0` gelten.

Verwenden Sie die Funktion `stree_approx_pattern_match` im Hauptprogramm der Datei `stree-approx-mn.c` (siehe Materialien zur Übung). Diese enthält eine `main`-Funktion, die einen Suffixbaum für eine Sequenz $S\$$ einliest. Dazu werden drei Argumente benötigt: Den Namen des Indexes, das den Suffixbaum repräsentiert, den Namen der Fasta-Datei, die die Muster jeweils in genau einer Zeile enthält und die maximale Anzahl von erlaubten Fehlern. Für das $i$-te Muster $p$ in der Musterdatei wird eine Zeile der Form

```
#<tab>i<tab>p
```

ausgegeben. Danach folgen jeweils in einer eigenen Zeile die Positionen des Vorkommens des Musters in $S$.

Damit Sie obiges Programm erfolgreich kompilieren und testen können, müssen Sie die gleichen Schritte durchführen wie in Aufgabe 4.3.. Auch die Datei `patternfile` und das Referenzgenom von *Ecoli K12* soll zum Testen wiederverwendet werden. Der Test wird durch `make test-approx-match` und mit Hilfe des Shell-Skriptes `check-approx-match.sh` durchgeführt.

**Aufgabe 5.2** Implementieren Sie in C auf Basis der Software-Bibliothek `GtSuffixtree` (siehe Übungsaufgabe 4.3.) in der Datei `stree-minunique.c` eine Funktion

```
int gt_stree_minunique(const char *indexname,
                       GtUword minlength,
                       bool withsequence,
                       GtError *err);
```

die den Algorithmus zur Berechnung von minimal eindeutigen Teilworten („minimal unique substrings“, siehe Vorlesungsskript) implementiert. Dabei gilt das folgende:

- `indexname` ist der Name des Suffixbaum Index.
- `minlength` ist die minimale Länge der auszugebenden eindeutigen Teilworte.
- Die Sequenz des minimal eindeutigen Teilworts wird jeweils mit ausgegeben, wenn der Parameter `withsequence` den Wert `true` hat.
- `err` ist das Fehlerobjekt, dass für die relevante Funktionen von `GtSuffixtree` benötigt wird, um Fehlermeldungen zu speichern.

Da obige Funktion in einem gegebenen `main`-Programm aufgerufen wird, brauchen Sie sich nicht darum kümmern, wie ein `err`-Objekt erzeugt wird. In dieser Anwendung von `GtSuffixtree` müssen Sie das `err`-Objekt an die Funktionen

- `gt_stree_loc_branch_iter_new` und
- `gt_stree_loc_branch_iter_apply`

übergeben. Falls eine dieser Funktionen den `NULL`-Zeiger bzw. eines von 0 verschiedenen Wert liefert, dann ist der Rückgabewert Ihree Implementierung $-1$. Falls kein Fehler auftritt ist der Rückgabewert Ihrer Funktion 0.

Das Hauptprogramm, das Sie in den Materialien zu diesem Übungsblatt finden (siehe `stree-minunique-mn.c`) hat zwei Argumente, nämlich den Namen des Suffixbaum Index sowie die minimale Länge $\ell$ der eindeutigen Teilworte. Jedes minimal eindeutige Teilwort wird durch seine Startposition und Länge tabulator-separiert in einer eigenen Zeile ausgegeben. Wenn zusätzlich noch die Option `-s` vor den beiden Argumenten des Programms angegeben ist, dann hat der Parameter `withsequence` den Wert `true`.

Damit Sie obiges Programm erfolgreich kompilieren und Testen können, müssen Sie die gleichen Schritte durchführen wie in Aufgabe 4.3..

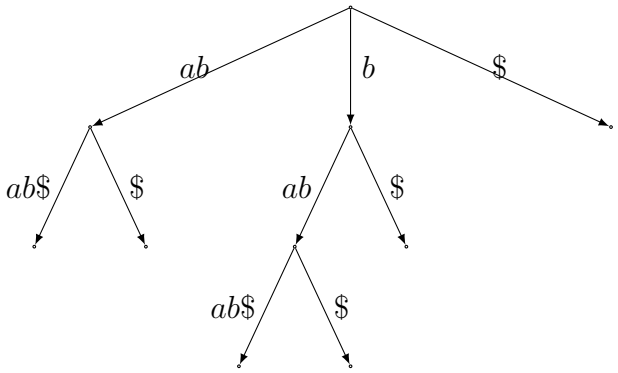
Als Beispielsequenz zum Testen Ihres Programms verwenden Sie wie schon in Aufgabe 4.3. das Genom von *Ecoli K12* in der Datei `Ecoli_K12.fna`. Dieses erhalten Sie durch den Aufruf `./download.sh 1`. und der entsprechende Index wird durch `./index.sh Ecoli_K12 Ecoli_K12.fna` erzeugt. Die erwartete Ausgabe für die Mindestlänge von 2700 finden Sie in der Datei `Ecoli_K12_MU_2700.csv`. Durch Aufruf von `make test-minunique` wird die Ausgabe Ihres Programms mit dieser Datei verglichen.

Bitte löschen Sie vor Abgabe der Lösungen die Datei `Ecoli_K12.fna` mit der Genomsequenz, sowie alle Indexdateien durch `rm -f Ecoli_K12.*`.

**Aufgabe 5.3** Wenn bei einem Suffixbaum ein Blatt zu einem Suffix von $S$ korrespondiert, das an der Stelle $i$ in $S$ beginnt, dann ist $i$ die Suffixnummer des Blattes. Für einige Anwendungen von Suffixbäumen benötigt man die Menge aller Suffixnummern in einem Unterbaum. Diese

kann man natürlich „on the fly“ berechnen, in dem man den Unterbaum durchläuft und dabei alle Suffixnummern aufsammelt. Wird die Menge der Suffixnummern für einen Knoten mehrfach benötigt, ist dieses Vorgehen nicht sinnvoll. Stattdessen möchte man an jedem Knoten die Menge der Suffixnummern im Unterbaum speichern und darauf in konstanter Zeit zugreifen. Überlegen Sie, wie man implizit die Menge aller Suffixnummern für jeden Knoten speichert, so dass man insgesamt für eine Eingabesequenz der Länge $n$ nur $O(n)$ Speicherplatz benötigt. Beschreiben Sie Ihren Ansatz.

Um ihre Idee zu entwickeln, ist es eventuell hilfreich, mit einem Beispiel zu arbeiten. Schreiben Sie auf, wie Ihre Repräsentation der Mengen von Suffixnummern für den folgenden Suffixbaum (für die Sequenz $babab\$$) aussehen würde:




**Die Lösungen zu diesen Aufgaben werden am 27.06.2017 besprochen.**