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

Genominformatik
Sommersemester 2014
Übungen zur Vorlesung: Ausgabe am 02.06.2014

Es handelt sich hier um Übungsblatt mit vielen Aufgaben, damit die Studierenden, die noch keine Lösung vorgestellt haben, genug Auswahlmöglichkeiten haben.

Punkteverteilung: Aufgabe 5.1: 3 Punkte, Aufgabe 5.2: 3 Punkte, Aufgabe 5.3: 3 Punkte, Aufgabe 5.4: 4 Punkte, Aufgabe 5.5: 3 Punkte

Abgabe bis zum 19.6.2014, 23:59 Uhr.

Aufgabe 5.1 Implementieren Sie in C auf Basis der Software-Bibliothek `GtSuffixtree` (siehe Übungsaufgabe 4.4.) 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 realisiert. 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-match-mn.c` (siehe Materialien zur Übung). Diese enthält eine `main`-Funktion, die einen Suffixbaum für eine Sequenz $S\$$ einliest. Dazu werden zwei Argumente benötigt: Den Namen des Indexes, das den Suffixbaum repräsentiert und den Namen der Fasta-Datei, die die Muster jeweils in genau einer Zeile enthält. 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 4.4.. Falls Sie diese schon gelöst haben, benötigen Sie lediglich die neue Version von `gt_suffixtree.h`, in der die Funktionen am Ende der Datei vollständig dokumentiert sind. 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-match.sh` durchgeführt.

Aufgabe 5.2 Implementieren Sie in C auf Basis der Software-Bibliothek `GtSuffixtree` (siehe Übungsaufgabe 4.4.) 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 Skript) realisiert. Dabei gilt das folgende:

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

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 ℓ 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 4.4.. Falls Sie diese schon gelöst haben, benötigen Sie lediglich die neue Version von `gt_suffixtree.h`, in der die Funktionen am Ende der Datei vollständig dokumentiert sind.

Als Beispielsequenz zum Testen Ihres Programms verwenden Sie wie schon in Aufgabe 4.4. 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 Implementieren Sie in C auf Basis der Software-Bibliothek `GtSuffixtree` (siehe Übungsaufgabe 4.4.) in der Datei `stree-mum.c` eine Funktion

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

die den auf Suffixbäumen basierten Algorithmus zur Berechnung von Maximum Unique Matches (MUMs) (siehe Skript) realisiert. Dabei ist `indexname` der Name des Suffixbaum In-

dex, der genau zwei Sequenzen enthält und `minlength` die minimale Länge der auszugebenden MUMs. Falls der Parameter `withsequence` den Wert `true` hat, wird die Sequenz des MUMs jeweils mit ausgegeben. `err` ist das Fehlerobjekt, dass für die relevante Funktionen von `GtSuffixtree` benötigt wird, um Fehlermeldungen zu speichern. Beachten Sie, dass für ein MUM, bei dem die Match mit der kleineren Position nicht an Position 0 beginnen die linken Kontexte a und b verglichen werden müssen. Es mus entweder $a \neq b$ sein oder a ist ein Sonderzeichen. Letzeres wird mit dem Macro `ISSPECIAL` aus `match/chardef.h` überprüft.

Das Hauptprogramm, das Sie in den Materialien zu diesem Übungsblatt finden (siehe `stree-mum-mn.c`) hat zwei Argumente, nämlich den Namen des Suffixbaum Index sowie die minimale Länge ℓ der MUMs. Für jeden MUM werden die Startpositionen in den beiden Sequenzen S und S' sowie die 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 4.4.. Falls Sie diese schon gelöst haben, benötigen Sie lediglich die neue Version von `gt_suffixtree.h`, in der die Funktionen am Ende der Datei vollständig dokumentiert sind.

Als Beispielsequenz zum Testen Ihres Programms verwenden Sie die Genome von *Ecoli K12* und *Ecoli O127 H6* in den Dateien `Ecoli_K12.fna` und `Ecoli_O127_H6.fna`, die Sie durch den Aufruf von `./download.sh 1 2` erhalten. Durch den Aufruf

```
./index.sh Ecoli_K12_O127 Ecoli_K12.fna Ecoli_O127_H6.fna
```

erhalten Sie den Suffixbaum Index `Ecoli_K12_O127` aus beiden Genomen. Die genannten Aufrufe finden Sie auch im `Makefile` beim Ziel `test-mum`. Die erwartete Ausgabe für die Mindestlänge 400 finden Sie in der Datei `Ecoli_K12_O127_MUM.csv`. Durch Aufruf von `make test-mum` wird die Ausgabe Ihres Programms mit dieser Datei verglichen.

Bitte löschen Sie vor Abgabe der Lösungen die beiden Dateien mit den Genomen sowie den Index durch `rm -f *.fna Ecoli_K12_O127.*`.

Aufgabe 5.4 Ein lcp-Intervall $[\ell, r]$ mit lcp-Wert h ist ein lokales Maximum, wenn für alle q, $\ell + 1 \leq q \leq r$ die Eigenschaft $\mathsf{LCP}[q] = h$ gilt. Geben Sie in Form von Pseudocode einen Algorithmus an, der für eine gegebene LCP-Tabelle alle lokalen Maxima und ihre LCP-Werte ausgibt. Sie können nicht davon ausgehen, dass die lcp-Intervalle bereits bestimmt sind. Für eine LCP-Tabelle einer Sequenz der Länge n soll die Laufzeit Ihres Algorithmus $O(n)$ sein. Verifizieren und dokumentieren Sie, dass Ihr Algorithmus für die beiden Sequenzen $taaaaga\$$ und $ccttcgt\#ctgtcgt\$$ die korrekten Ergebnisse liefert. Hier sind die beiden Suffix Arrays sowie die lokalen Maxima:

i	SUF	LCP
0	1	
1	2	3
2	3	2
3	4	1
4	6	1
5	5	0
6	0	0
7	7	0

lok. Max.	LCP-Wert
[0,1]	3

i	SUF	LCP
0	0	
1	4	1
2	12	3
3	8	1
4	1	2
5	10	0
6	5	2
7	13	2
8	3	0
9	11	4
10	9	1
11	2	1
12	6	1
13	14	1
14	7	0
15	15	0

lok. Max.	LCP-Wert
[1,2]	3
[3,4]	2
[5,7]	2
[8,9]	4

Aufgabe 5.5 Im Greedy-Algorithmus zur Fragmentassemblierung wird die *Union-Find-Datenstruktur* verwendet. Diese wird auch als Datenstruktur zur Repräsentation von Disjunkten Mengen durch Wälder (disjoint-set forests) beschrieben.

Implementieren Sie in Ruby, oder C oder Python die *Union-Find-Datenstruktur* mit Hilfe eines Arrays und der Technik der „Pfadkomprimierung“, wie in Cormen et. al. 2009, oder in `http://en.wikipedia.org/wiki/Disjoint-set_data_structure` beschrieben.

Wenn Sie die Lösung vorstellen, sollen Sie auch die Technik der „Pfadkomprimierung“ erklären zu können.

Die Lösungen zu diesen Aufgaben werden am 23.06.2014 besprochen.