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

Genominformatik
Sommersemester 2017
Übungen zur Vorlesung: Ausgabe am 23.05.2017

Punkteverteilung: Aufgabe 4.1: 6 Punkte, Aufgabe 4.2: 3 Punkte, Aufgabe 4.3: 6 Punkte

Abgabe bis zum 08.06.2017, 23:59 Uhr.

Aufgabe 4.1 Implementieren Sie in einem Programm `wotd` die *Write Only Top Down*-Konstruktion von Suffixbäumen. Die $-Kante soll immer die letzte Kante sein, die von einem Knoten ausgeht. Alle anderen Kanten sollen alphabetisch entsprechend des ersten Zeichens der Kantenmarkierung sortiert werden. Bitte repräsentieren Sie in Ihrem Programm die Suffixe durch ihre Startposition. Entsprechend können Mengen von Suffixen als Integer-Arrays repräsentiert werden.

Beachten Sie, dass es zur Lösung dieser Aufgabe nicht notwendig ist, den konstruierten Baum im Hauptspeicher zu repräsentieren. Es reicht, wenn die R-Mengen und die Gruppen berechnet werden, wobei die konstruierten Knoten und Kanten "on the fly" ausgegeben werden.

Als Argument des Programms soll die Sequenz ohne Sentinel übergeben werden. Den Sentinel müssen Sie selbst an die Sequenz anhängen. Verwenden Sie in Ihrem Programm intern das Zeichen mit dem ASCII-Code 255 als Sentinel. In der Ausgabe verwenden Sie statt dieses Zeichens das Symbol $. Ihr Programm soll mindestens zwei der drei folgenden Ausgabeformate unterstützen, das durch eine entsprechende Option ausgewählt wird:

Text Bei diesem Format werden nur die Kantenmarkierungen und Blattmarkierungen zeilenweise ausgeben, und zwar mit führenden Leerzeichen wie folgt: Wenn der Pfad zu einem verzweigenden Knoten i Kanten hat, dann sollen die Markierungen der Kanten mit $2i$ führenden Leerzeichen ausgegeben werden. Bei Blattkanten wird nach der Kantenmarkierung in der gleichen Zeile die Startposition des Suffixes, der zu dem Blatt korrespondiert, ausgegeben. Kantenmarkierung und Blattmarkierung werden durch einen Tabulator getrennt. Die Ausgabe der Kanten erfolgt in alphabetischer Reihenfolge von oben nach unten, siehe Abbildung 1(a).

dot Das dot-Format besteht aus der Liste aller Kanten des Baumes. Für jede Kante wird jeweils der Quellknoten, der Zielknoten und in eckigen Klammern die Kantenmarkierung angegeben. Für einen internen Knoten $\overline{w}$ kann man den Bezeichner w verwenden. Für ein Blatt, das zum Suffix $S[i \ldots n-1]\$$ korrespondiert, kann man i verwenden, Eine Ausgabe in diesem Format kann durch das Programm `dot` aus dem *Graphviz*-Paket (`http://www.graphviz.org`) visualisiert werden, siehe Abbildungen 1(b) und 1(c). Auf den ZBH-Rechnern ist `dot` unter `/usr/bin/dot` installiert.

tikz Das *tikz*-Format wird durch das *tikz*-Paket, das Teil von LaTeX ist, unterstützt. Hiermit wurden die Bäume aus dem Vorlesungsskript gezeichnet. Ein Baum besteht aus einem Wurzelknoten, der durch die `\node`-Befehl spezifiziert wird. Dieser wird durch

ein Semikolon am Ende abgeschlossen. Weiterhin besteht der Baum aus Kindern, für die jeweils das Schlüsselwort `child` verwendet wird. Alle Teile, die zu einem Kind gehören, werden durch `{` und `}` begrenzt. Ein Kind wird durch das Schlüsselwort `node` beschrieben, optional mit einer Annotation in `{}`-Klammern. Eine Kante wird durch das Schlüsselwort `edge from parent` beschrieben, optional mit einer Markierung in `{}`-Klammern nach dem Schlüsselwort `node`. Dieses kann mit Optionen versehen werden, um die Lage der Markierung relativ zur Kante selbst anzugeben, bzw. den Stil des Knotens, entsprechend der Stil-Definitionen in der Präambel des *tikz*-Formates. Ein Beispiel findet man in Abbildungen 1(d) und 1(e).

Die Textdateien für alle drei Formate finden Sie in den Materialien zu dieser Übung. Ausserdem finden sie darin noch entsprechende Dateien für den Suffixbaum der Sequenz $caccaccac\$$. Bitte verifizieren Sie, dass die Ausgabe Ihres Programms mit den Ausgabedateien aus den Materialien übereinstimmt.

Aufgabe 4.2 Gegeben sei ein Alphabet $\mathcal{A}$ und ein String $S \in \mathcal{A}^+$. Für jedes S definieren wir eine Menge $Absent(S)$:

$$Absent(S) = \left\{ w \in \mathcal{A}^+ \mid w \text{ ist } \underline{\text{kein}} \text{ Substring von } S \right\}$$

Die Elemente von $Absent(S)$ heißen ***abwesende Worte***. $Absent(S)$ ist offensichtlich unendlich gross, denn es gibt keine Begrenzung der Wortlänge für die abwesenden Worte in S. Wir sind aber daran interessiert, wie lang ein abwesendes Wort mindestens ist. Daher sei

$$\mu = \min \{|w| \mid w \in Absent(S)\}$$

die minimale Länge eines abwesenden Wortes in S.

Entwickeln und beschreiben Sie einen Algorithmus, der den Wert μ mit Hilfe des Suffixbaums von $S\$$ bestimmt und alle abwesenden Worte der Länge μ aufzählt. Dabei soll die Struktur des Suffixbaumes genutzt werden. Eine Lösung, die alle Strings u über dem gegebenen Alphabet sortiert nach der Länge aufzählt und den Suffixbaum nur nutzt, um festzustellen, ob u in S vorkommt, ist nicht gewünscht. Geben Sie den Algorithmus in Pseudocode an. Welche Laufzeit hat Ihr Algorithmus? Falls Sie für den Pseudocode LaTeX verwenden wollen, sollten Sie ihn wie im Vorlesungsskript formatieren. In den Materialien finden Sie eine Datei mit einem Beispiel.

Aufgabe 4.3 Implementieren Sie in C auf Basis der Software-Bibliothek `GtSuffixtree` in der Datei `stree-exact.c` eine Funktion

```
const GtUword *stree_exact_pattern_match(GtUword *numofmatches,
                                         const GtStree *stree,
                                         const GtUchar *pattern,
                                         GtUword len)
```

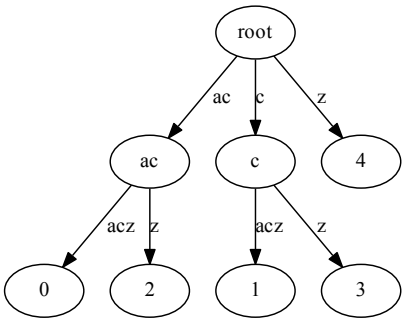
die in einem Suffixbaum `stree` ein Muster `pattern` sucht. Falls das Muster im Suffixbaum nicht vorkommt, liefert die Funktion den `NULL`-Zeiger und der Speicherplatz auf den `numofmatches` zeigt hat den Wert 0. Falls das Muster im Suffixbaum vorkommt, liefert die Funktion einen

```
ac
  ac$ 0
  $ 2
c
  ac$ 1
  $ 3
$ 4
```

(a) Text-Format für $ST(acac\$)$

```
digraph suffixtree_acacz {
  root -> ac [label=ac];
  ac -> 0 [label=acz];
  ac -> 2 [label=z];
  root -> c [label=c];
  c -> 1 [label=acz];
  c -> 3 [label=z];
  root -> 4 [label=z];
}
```

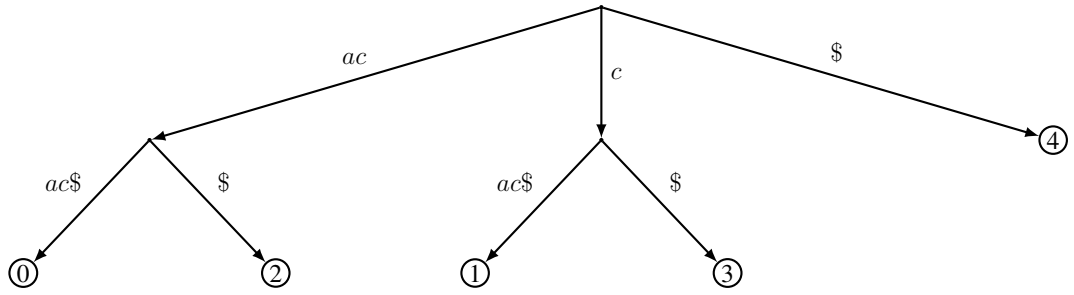
(b) dot-Format für $ST(acac\$)$. Beachten Sie, dass als Sentinel das Zeichen `z` verwendet wird, da $ im dot-Format nicht in einem `label` verwendet werden darf.



(c) Visualisierung des Dotfiles

```
\begin{tikzpicture}[
  >=stealth', thick, level distance=50pt,
  level 1/.style={sibling distance=170pt},
  level 2/.style={sibling distance=95pt},
  level 3/.style={sibling distance=55pt},
  level 4/.style={sibling distance=15pt},
  every node/.style={scale=.8},
  edge from parent/.style={draw,-latex},
  root node/.style={circle,draw,inner sep=0pt},
  leaf node/.style={circle,draw,inner sep=1pt},
  inner node/.style={circle,draw,inner sep=0pt}
]
\node[root node] {}
child {
  node[inner node] {}
  child {
    node[leaf node] {0}
    edge from parent node[auto=right] {$ac\$$}
  }
  child {
    node[leaf node] {2}
    edge from parent node[auto=left] {$\$$}
  }
  edge from parent node[auto=right] {$ac$}
}
child {
  node[inner node] {}
  child {
    node[leaf node] {1}
    edge from parent node[auto=right] {$ac\$$}
  }
  child {
    node[leaf node] {3}
    edge from parent node[auto=left] {$\$$}
  }
  edge from parent node[auto=left] {$c$}
}
child {
  node[leaf node] {4}
  edge from parent node[auto=left] {$\$$}
}
;
\end{tikzpicture}
```

(d) tikz-Format für $ST(acac\$)$



(e) tikz-Ausgabe für $ST(acac\$)$

Abbildung 1: Visualisierung von Suffixbäumen

Zeiger auf ein Array von allen $k > 0$ Positionen, an denen das Muster vorkommt. Der Speicherplatz auf den `numofmatches` zeigt hat den Wert k.

Die Klassen, d.h. Typen und Funktionen von `GtSuffixtree` sind in der Datei `gt_suffixtree.h` dokumentiert und werden in der Vorlesung oder Übung ërläutert. Sie brauchen für diese Aufgabe nur einige der Funktionen. Überlegen Sie genau welche, bevor Sie mit der Implementierung beginnen.

Verwenden Sie die Funktion `stree_exact_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, und alle Vorkommen von angegebenen Mustern in S ausgibt. Dazu werden zwei Argumente benötigt: Den Namen des Indexes, der den Suffixbaum repräsentiert sowie 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 folgende Schritte durchgeführt werden:

1. Die Suffixbaum-Bibliothek `GtSuffixtree` basiert auf den GenomeTools, einer in C implementierten Open-Source Software, die in der Arbeitsgruppe Genominformatik entwickelt wurde. Die GenomeTools findet man unter `http://genometools.org`. Bitte verwenden das Shell-Skript `gt-download.sh` um die aktuelle "unstable" Version der Genometools zu kopieren. Nach Auspacken des Tar-Archives sollte

   ```
   make errorcheck=no with-sqlite=no cairo=no 64bit=yes
   ```

   im Wurzelverzeichnis `genometools-unstable` funktionieren.

2. Sobald die GenomeTools erfolgreich compiliert sind, definieren Sie (passend zu Ihrer Linux-Shell) in der Datei `.bashrc` oder `.profile` Umgebungsvariablen `GTDIR` und `CC` und führen diese mit `. ~/.bashrc` oder `. ~/.profile` aus. Alternativ können Sie ein neues Login durchführen. `GTDIR` gibt den Pfad zu Ihrer GenomeTools Installation an und `CC` den Namen des C-Compilers. Diese Definitionen könnten so aussehen:

   ```
   export GTDIR=${HOME}/genometools-unstable
   export CC=clang
   ```

   falls Sie die GenomeTools in Ihrem Home-Verzeichnis liegen. Falls Sie nicht die `bash` benutzen (sondern z.B. die `tcsh`), dann ergänzen Sie in der passenden Datei (z.B. `.tcshrc`) die Definition von `GTDIR` und `CC` in der entsprechnden Syntax und führen die entsprechende Datei einmal aus (z.B. durch `source ~/.tcshrc`).

3. Welchseln Sie nun in das Verzeichnis `materialStreeExact`. Durch `make stree-exact-match.x` können Sie das entsprechende ausführbares Programm kompilieren.

4. Die Materialien zur Übung enthalten eine Datei `patternfile` mit 100 Mustern sowie die Skripte zum Generieren der Ergebnisse mit einem anderen Programm.
5. Ausserdem benötigen Sie noch das Genom von *E. coli K12*. Wenn Sie eine Internet-Verbindung haben und das Programm `curl` installiert ist, dann erhalten Sie dieses Genom durch den Aufruf von

   ```
   ./download.sh 1
   ```
6. Sie können nun in Ihrem Entwicklungsverzeichnis einen Suffixbaum-Index durch den Aufruf

   ```
   ./index.sh Ecoli_K12 Ecoli_K12.fna
   ```

   erzeugen. Der Index besteht aus mehreren Dateien, die mit dem Präfix `Ecoli_K12` beginnen.
7. Wenn alle Schritte erfolgreich abgeschlossen sind, dann testen Sie `stree-exact-match.x` durch den Aufruf von `make test-exact-match`. Der Test wird durch das Shell-Skript `check-exact-match.sh` realisiert, das überprüft, ob Ihr Programm die gleiche Ausgabe erzeugt wie das Programm `tagerator` aus den GenomeTools. Um bei der Entwicklung mögliche Fehler zu finden, müssen Sie `check-exact-match.sh` ggf. für ein kürzeres Muster und eine kleinere Sequenz aufrufen.

Das README in den Materialien enthält weitere Informationen über die einzelnen Schritte. Ihre Abgabe sollte alle benötigten Dateien (außer die Genomsequenz) enthalten, so dass `make test-exact-match` erfolgreich ist. Bitte löschen Sie vor Abgabe der Lösungen die Datei `Ecoli_K12.fna` mit der Genomsequenz, sowie alle Indexdateien durch `rm -f Ecoli_K12.*`. Abgaben, die diese Dateien noch enthalten, werden nicht angenommen.

Die Lösungen zu diesen Aufgaben werden am 13.06.2017 besprochen.