Chris Bielow Institut für Informatik
Bioinformatics Solution Center

Praktikum: Algorithmen und Datenstrukturen in der Bioinformatik
6. Programmieraufgabe

Abgabe Montag, 29.01., 23:59 Uhr per GIT

Aho Corasick Trie (10 Punkte)

In dieser Programmieraufgabe werden Sie mit Hilfe eines AhoCorasick-Tries mehrere Patterns zeitgleich in einer Query (Haystack) suchen.

Lesen sie im Hauptprogramm *aufgabe6_ main.cpp* als erstes Argument von der Kommandozeile die Query und als folgende Argumente (mindestens ein) Pattern ein. Z.B.

```
./aufgabe6_main GATTACA  AT TA A
```

sucht im String *GATTACA* die drei Pattern *AT*, *TA* und *A*.

Geben Sie alle Hits sortiert nach dem Zeitpunkt des Findens im Text aus (längere Hits zuerst):

```
 ./aufgabe6_main  GATTACA TT ATT ATT A
Hits (position in query, pattern):
  pos 1, A
  pos 1, ATT
  pos 1, ATT
  pos 2, TT
  pos 4, A
  pos 6, A
```

Checken Sie *ACTrie.hpp, ACTrie.cpp und aufgabe6_ main.cpp* ins GIT in den Unterordner *./aufgabe6/* ein. Testen Sie, ob die URL (hier EXAMPLARISCH für Lab4) die notwendigen Datein anzeigt!

`https://git.imp.fu-berlin.de/adp2023/group04/-/blob/main/aufgabe6/ACTrie.hpp`
`https://git.imp.fu-berlin.de/adp2023/group04/-/blob/main/aufgabe6/ACTrie.cpp`
`https://git.imp.fu-berlin.de/adp2023/group04/-/blob/main/aufgabe6/aufgabe6_main.cpp`

Praktikumshinweise

- Benutzen und implementieren Sie Output-Links (d.h. folgen Sie nicht den Suffix-Links für die Ausgabe von Hits).
- Pattern könnten doppelt vorkommen und sollten dann entsprechend auch mehrere Hits erzeugen
- Im Header *ACTrie.hpp* sind die Funktionen genau beschrieben, auch evtl. exceptions die geworfen werden sollen.
- Compilieren sie ihr Programm mithilfe des Makefiles auf dem Poolrechner um Compile-Errors (und damit 0 Punkte) auszuschliessen
- Am Montag zum Tutorium wird eine Test-Klasse online gestellt, mit der Sie ihre Implementierung überprüfen können.

Zusatzaufgabe (4 Punkte)

Implementieren Sie einen *fully resolved* AC-trie, indem Sie beim Aufbau des Tries als letzten Schritt für jeden Knoten so viele zusätzliche ausgehende Kanten erzeugen bis die Grösse des Eingabealphabet erreicht ist. Diese neuen Kanten werden als Ersatz für den einen Suffixlink benutzt um beim Konsumieren eines Eingabezeichens aus der Query direkt (in einem Schritt) zum richtigen Folgeknoten zu springen (d.h. ob das Eingabezeichen das Pattern verlängert oder verkürzt macht semantisch keinen Unterschied mehr). Der Trie kann somit als Matrix verstanden werden (Zeilen sind Knoten, Spalten sind Zielknoten für jede mögliche Eingabe aus dem Alphabet). Verknüpfen Sie diese neuen Kanten indem Sie die Eingabe des entsprechenden Buchstaben simulieren und den Zielknoten direkt anspringen. Für die Ausgabe von Hits werden weiterhin Output-Links verwendet. Dokumentieren Sie Sie Ihre Abgabe/Code entsprechend.