Chris Bielow

Institut für Informatik
Bioinformatics Solution Center

Praktikum: Algorithmische Bioinf I und Numerik
3. Programmieraufgabe

Abgabe Montag, 01.12., 23:59 Uhr via GIT

In dieser Programmieraufgabe werden Sie die exakte Stringsuche für Strings aus dem ASCII-Alphabet (d.h. lexikographisch sortiert nach ASCII) mithilfe eines Suffixarrays implementieren. Dazu sind die Deklarationen zweier Funktionen `void construct(...);` und `void find(...);` in der Header-Datei aufgabe3.hpp vorgegeben.
Implementieren Sie diese zwei Funktionen in einer `aufgabe3.cpp`. Implementieren sie weiterhin eine `aufgabe3_main.cpp` welche eine Main-Funktion `int main(int argc, const char* argv[])` enthält und ihre Funktionen aus `aufgabe3.cpp` benutzt. Kompilierung erfolgt ähnlich zu Aufgabe 2: ihr Programm muss auf den Linux Poolrechnern mit
g++ -std=c++17 -Wall -pedantic -O3 -D_GLIBCXX_ASSERTIONS -g -fsanitize=address
kompilieren (siehe Makefile) und das korrekte Ergebnis liefern.

Hinweis: wenn Sie auf ihrem Rechner gcc/clang benutzen, aber das flag '-fsanitize=address' nicht unterstützt wird, dann upgraden Sie ihren Compiler oder lassen das Flag weg. In jedem Fall sollten Sie aber das Programm auf einem PC-Pool Rechner/Computeserver kompilieren und dort auf Korrektheit prüfen. Führen Sie KEINE COMPILE / RECHENJOBS auf Login-Nodes (Andorra, Lounge, etc) durch!

Aufbau Schreiben Sie eine `construct` Funktion – siehe aufgabe3.hpp, die ein zu füllendens Suffixarray sowie den Text erhält, das Suffixarray konstruiert und zurückgibt. Nutzen Sie die naive Konstruktionsmethode mit `std::sort`. Beachten Sie, dass ein Suffixarray niemals Strings speichert, sondern lediglich deren Startpositionen im Originaltext. Die Laufzeit der Konstruktionsmethode sollte also O(n*logn*c) sein, wobei c die Kosten für Vergleiche(!) von Strings abbildet.

- Anders als in den theoretischen Aufgaben benötigen sie kein extra $ am Ende des Textes. Warum?
- Wir definieren, dass ein Präfix eines Strings X kleiner ist als X.
- Um `std::sort` nutzen zu können, müssen Sie einen sog. Funktor anlegen, der den $<$ Operator für zwei Textpositionen definiert oder eine Lambda-Funktion verwenden. Beispiele hierfür gibt es im Netz, u.A. unter `http://en.cppreference.com/w/cpp/algorithm/sort`.

Suche Programmieren Sie die Binärsuche mit der *mlr*-Heuristik in der Funktion `find` (sonst Punktabzug). Die gefundenen Hits sollen aufsteigend nach Position im Text sortiert zurückgegeben werden!

Aufruf Ihr Programm `aufgabe3_main.cpp` soll in 2 Modi ausgeführt werden. Gibt man nur ein Argument - den Text - an, so soll das Suffixarray zeilenweise ausgegeben werden. Gibt man nach dem Text noch ein oder mehrere Suchwörter als weitere Kommandozeilenparameter an, dann sollen zeilenweise die Suchwörter, sowie, durch Leerzeichen getrennt, die Liste der Positionen der Treffer im Text ausgegeben werden. **Diese Liste von Positionen soll aufsteigend sortiert sein!** Bei nicht korrektem Aufruf soll das Programm `unexpected input` ausgeben und den return code 1 zurück geben.

Beispiele für beide Modi:

```
./aufgabe3 "banana"
5
3
1
0
4
2
```

```
./aufgabe3 "exact search using suffix arrays" "s" "horspool" "g suf"
s: 6 14 19 31
horspool:
g suf: 17
```

Abgabe Legen Sie die gesuchten Dateien `aufgabe3.cpp und aufgabe3_main.cpp` im Unterordner ./aufgabe3/ ihres Gruppenverzeichnisses an und checken Sie es ins GIT ein. Achten Sie auf korrekte Gross/Kleinschreibung (d.h. alles klein!).

Testen Sie, ob die URL (hier EXAMPLARISCH für Lab4) die notwendigen Datein anzeigt!

```
https://git.imp.fu-berlin.de/adp2025/group04/-/blob/main/aufgabe3/aufgabe3.cpp
https://git.imp.fu-berlin.de/adp2025/group04/-/blob/main/aufgabe3/aufgabe3_main.cpp
```

Praktikumshinweise **WICHTIG!** Achten Sie auf die korrekte Ausgabe. Die Ausgabe im 2. Modus entspricht dem Format: *suchwort: treffer_1 treffer_2 ....* Sollte kein Treffer gefunden werden, wird nur das Suchwort und der Doppelpunkt ausgegeben.

Für diese Aufgabe es insgesamt 10 Punkte, wenn das Programm korrekt funktioniert. Eine aufgabe3_test.cpp sagt ihnen, ob das der Fall ist (verfügbar ab dem Tutorium). Achten Sie deshalb darauf, dass ihr Programm gutartig auf folgende Eingaben reagiert: leerer Text, leere Query, Query=Text, überlappende Query (z.B. 'aaa' in Text 'aaaaaa'), Query kommt nicht im Text vor. Alle diese Fälle werden getestet.
Achtung: Verzichten Sie auf jegliche Nutzung von std::vector<string>, und string::substr() in der Konstruktion. Dadurch koennen Sie die Laufzeitanforderung von n*logn nicht mehr erfüllen! (Punktabzug)!
Auch in der Suche wird string::substr nicht benötigt!

Zusatzpunkte Es gibt 4 Zusatzpunkte wenn ihr Programm schneller ist, als die spätere Musterlösung auf einem recht großen Text (kleineres Genom: Candidatus.Solibacter.usitatusEllin6076.fa ca. 9MB) und vielen Queries (1 mio). Aufbau und die Suchzeit werden dabei addiert. Wie ihre Implementierung das intern macht, ist Ihnen überlassen, auch bessere Heuristiken als mlr sind erlaubt.
Das Programm aufgabe3_bench.cpp könnte ihnen helfen zu entscheiden, ob ihre iterativen Verbesserungen am Code sinnvoll sind. Eine Eingabedatei mit Text (Candidatus.Solibacter.usitatusEllin6076.fasta) liegt bei. Die zu schlagende Zeit für den Aufruf

```
./aufgabe3_bench Candidatus.Solibacter.usitatusEllin6076.fasta 1000000
```

liegt bei 11.5 Sekunden auf einem der Compute-Rechner (compute01 - compute05) am Fachbereich (`https://www.mi.fu-berlin.de/w/IT/ComputeServer#Compute_Server`).