

Praktikum: Algorithmen und Datenstrukturen

Zweite Programmieraufgabe

Abgabe Donnerstag, 26.11., 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 `aufgabe2.hpp` vorgegeben.

Implementieren Sie diese zwei Funktionen in einer `aufgabe2.cpp`. Implementieren sie weiterhin eine `aufgabe2_main.cpp` welche eine Main-Funktion `int main(int argc, const char* argv[])` enthält und ihre Funktionen aus `aufgabe2.cpp` benutzt. Kompilierung erfolgt ähnlich zu Aufgabe 1: ihr Programm muss auf den Linux Poolrechnern mit `g++ -Wall -pedantic -O3 -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 die Programm auf einem PC-Pool Rechner kompilieren und dort auf Korrektheit prüfen.

Aufbau Schreiben Sie eine `construct` Funktion – siehe `aufgabe2.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.

- 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 2 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 `aufgabe2_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:

```
./aufgabe2 "banana"
```

```
5
3
1
0
4
2
```

```
./aufgabe2 "exact search using suffix arrays" "s" "horspool" "g suf"
```

```
s: 6 14 19 31
```

```
horspool:
```

```
g suf: 17
```

Abgabe Legen Sie die gesuchten Dateien `aufgabe2.cpp` und `aufgabe2_main.cpp` im Unterordner `./aufgabe2/` ihres Gruppenverzeichnis an und checken Sie es ins GIT ein. Achten Sie auf korrekte Gross/Kleinschreibung (d.h. alles klein!).

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 `aufgabe2_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. Punktabzug bei Fehlverhalten.

Achtung: Verzichten Sie auf jegliche Nutzung von `std::vector<string>` (Punktabzug)!

Es gibt 4 Zusatzpunkte wenn ihr Programm schneller ist, als die spätere Musterlösung auf einem recht großen Text (kleineres Genom: `Candidatus.Solibacter.usitatusEllin6076.fasta`, 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. Die zu schlagende Zeit ist ihnen nicht bekannt, liegt aber weit unter 30 Sekunden (mit -O3) auf einem ihnen unbekannten Rechner. Hinweis: das Programm `aufgabe2_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.