Prof. Dr. Knut Reinert
Enrico Siragusa
Sascha Meiers
Christoph Hartmann

Institut für Informatik
AG Algorithmische Bioinformatik

Algorithmen und Datenstrukturen in der Bioinformatik
Siebentes Übungsblatt WS 10/11

Abgabe Montag, 5.12., 15:00 Uhr

Name: Übungsgruppe: A☐ B☐ C☐

Matrikelnummer:

Niveau I

Aufgabe 1 + 2: FastA

In dieser Aufgabe soll mit dem **FastA**-Algorithmus ein lokales Alignment der Sequenzen $S_1 =$ `IALEQIAQDI` and $S_2 =$ `TIALDIAWEADI` gefunden werden. Halten Sie sich dabei an das Skript und führen Sie die folgenden 6 Schritte aus:

a) Erstellen Sie eine *Hash-Tabelle* für die Sequenz S_1 mit $ktup = 2$ als Parameter.

b) Finden Sie mit Hilfe dieser Hash-Tabelle die *Hot-spots* in Sequenz S_2

c) Tragen Sie jetzt diese Hot-spots in einen *Dot-plot* ein und finden Sie die *Diagonal runs*. Hierbei dürfen Diagonale Lücken der Länge 1 (entspricht einem Mismatch) zugelassen werden.

d) *Rescoring*: Bilden Sie den Score aller ihrer Diagonal runs gemäß der PAM 70 Scoringmatrix.

e) Erzeugen Sie durch *topologische Sortierung* einen *DAG* aus den gefundenen Hot-spots (Es sollten 5 Runs mit Länge >= 2 entstehen)

f) Modifizieren Sie diesen Graph, sodass der **DAG shortest paths**-Algorithmus anwendbar ist und lösen Sie anschließend das *chaining-Problem*.

Niveau II

Aufgabe 3: Verständnis

a) An welchen Stellen des FastA-Algorithmus wird deutlich, dass er eine *Heuristik* ist? Unter welchen Umständen wird ein optimales lokales Alignment nicht gefunden?

b) Wie wirken sich groß e und kleine $k-tup$-Werte auf *Sensitivität* und *Spezifität* des Algorithmus aus?

Aufgabe 3: Vorbereitung

Bereiten Sie sich auf das Review vor. Es findet im SR 031, Arnimallee 6, am Mittwoch, dem 7.12. von 12-14 Uhr statt. Es werden die bisherigen Vorlesungsinhalte bis auf den **FastA**-Algorithmus abgefragt.

Programmieraufgabe (Abgabe Montag, 12.12.2010, 15:00)

P-Aufgabe 4: Parallelize Horspool algorithm using OpenMP.

As in P-Aufgabe 2, your program has to take as first argument a filename containing the text. The second argument is one single pattern. The third argument is the number of threads to use. The fourth and last argument is a results filename where start positions of all found occurrences have to be written (in ascending order).

The program has to output the time elapsed in milliseconds (without measuring file input/output and sorting). The function `omp_get_wtime` can be used to this purpose.

Example:

```
user@tetrahymena:~$ ./aufgabe4 english.50MB whatever 1 results.txt
1000, 687
user@tetrahymena:~$ ./aufgabe4 english.50MB whatever 2 results.txt
500, 687
user@tetrahymena:~$ ./aufgabe4 english.50MB whatever 4 results.txt
250, 687
user@tetrahymena:~$
```

Hints: While searching, you can use a `std::vector<unsigned long>` to store start positions of occurrences found in the text (remember, 0 is the first position). At the end of the search, these positions must be sorted (with `std::sort`) and written into a file (see `std::ofstream`).

Example:

```
user@tetrahymena:~$ head results.txt
46909
60671
71667
74688
```

```
96565
248812
251671
252341
260710
260784
user@tetrahymena:~$
```

Remember the Praktikum material at `https://www.mi.fu-berlin.de/w/ABI/AlDaBiWS11`.