S. Kurtz, T. Schwabe,
G. Gonnella, D. Willrodt

Universität Hamburg
Zentrum für Bioinformatik

Programmierung für Naturwissenschaften
Sommersemester 2015
Übungen zur Vorlesung: Ausgabe am 21.05.2015

Punkteverteilung: Aufgabe 7.1: 5 Punkte, Aufgabe 7.2: 5 Punkte

Abgabe bis zum 03.06.2015, 10:00 Uhr.

Aufgabe 7.1 In der biologischen Sequenzanalyse gibt es viele „alle-gegen-alle"-Probleme, bei der für eine gegebene Menge M von Sequenzen alle Paare $s, s' \in M$ verglichen werden, um jeweils einen Distanzwert zu ermitteln. Der Vergleich zweier Sequenzen ist meist sehr aufwändig. So werden typischerweise Verfahren zum Sequenzvergleich angewendet, die für zwei Sequenzen s und s' eine Laufzeit von $O(|s| \cdot |s'|)$ haben. Dabei ist $|s|$ die Länge der Sequenz s. Diese Verfahren rechnen also viel auf relativ wenigen Daten und liefern wenige Ausgaben. Daher kann man diese „alle-gegen-alle"-Probleme gut mit Hilfe von Shared-Memory Multithreading Ansätzen lösen.

In dieser Aufgabe sollen Sie ein C-Programm `all_against_all.c` schreiben, das eine Datei im vereinfachten multiplen-Fasta Format einliest und für alle darin enthaltenen Sequenzpaare die sogenannte Edit-Distanz berechnet. Das ist ein Maß, das den Unterschied von zwei Sequenzen ausdrückt. Je geringer die Edit-Distanz, umso weniger Unterschiede gibt es in den Sequenzen. Für die Berechnung der Edit-Distanz gibt es in den Materialien zu dieser Übung bereits in der Datei `unit-edist.c` eine Implementierung einer Funktion

```
unsigned long eval_unit_edist(const unsigned char *u,
                              unsigned long m,
                              const unsigned char *v,
                              unsigned long n)
```

die für zwei Sequenzen die Edit-Distanz berechnet.

Dateien im vereinfachten multiplen Fasta-Format bestehen aus Kopfzeilen, die mit dem Zeichen `'>'` beginnen, gefolgt von genau einer Sequenz-Zeile, in der die Sequenzinformation steht.

Ihr Programm soll die k Paare von Sequenzen mit minimalen Distanzwerten ermittelt, wobei k ein durch den Benutzer definierter Wert ist, der als Option an das Programm übergeben werden soll.

Um diese k besten Sequenzpaare zu verwalten, verwenden Sie die beigefügte Datenstruktur `BestKVals`. Diese benötigt einen Funktionszeiger auf eine Vergleichsfunktion. Um Sequenzpaare zu unterscheiden, die identische Distanzwerte haben, vergleichen Sie zuerst die erste Sequenznummer und dann die zweite. Nehmen Sie auch hier an, dass kleinere Nummern „besser" sind, analog zu den Distanzwerten.

Lesen Sie die Eingabe-Datei komplett in den Speicher, da in diesem Fall zufällige Zugriffe auf den Dateiinhalt nötig sind. In der Dateien `multiseq.c` und `multiseq.h` implementieren Sie eine Datenstruktur, dafür geeignet, Informationen über die Eingabesequenzen zu speichern. Speichern Sie damit für jede Sequenz einen Zeiger auf den Anfang der Sequenz und die Länge der Sequenz.

Sei n die Anzahl der Sequenzen und seien $s_0, \ldots, s_{n-1}$ die Sequenzen. Implementieren Sie in `all_against_all.c` die Funktion `eval_seqrange`, die für gegebene Werte i, j, $0 \leq i \leq j \leq$

$n-1$, alle Sequenzen s_p, $i \leq p \leq j$ mit allen Sequenzen s_q, $p < q \leq n-1$ vergleicht und einen Distanzwert ermittelt. Die Funktion zum Vergleich und Berechnung des Distanzwertes soll als Funktionszeiger übergeben werden. Sie können davon ausgehen, dass die übergebene Funktion quadratische Zeit benötigt.

Sie sollen die Funktion `eval_seqrange` in $t \geq 1$ Threads gleichzeitig für verschiedene disjunkte Teilmengen der gegebenen Sequenzmenge aufrufen. Dazu unterteilen Sie den Indexbereich von 0 bis $n-1$ in t Abschnitte, so dass für jeden Abschnitt in etwa die gleiche Rechenzeit zu erwarten ist. Überlegen Sie sich, wie man diese gleichmäßige Aufteilung erreichen kann.

Neben dem Parameter k soll die Anzahl der Threads durch eine Option an das Programm übergeben werden. Der letzte Parameter Ihres Programms ist der Name der vereinfachten multiplen Fasta-Datei. Ein Aufruf soll also wie folgt aussehen:

```
all_against_all.x <k> <t> <Dateiname>
```

Dabei führt $k < 1$ und $t < 1$ zu einer Fehlermeldung und zum Abbruch des Programms.

Um das Programm zu testen, finden Sie in der Materialien zur Übung eine Sequenzdatei mit 175 Proteinsequenzen: der Aufruf von `make test` soll fehlerfrei beenden.

Aufgabe 7.2 Schreiben Sie ein Modul `sorting.c` mit den folgenden zwei Sortierfunktionen:

(`sorting.h`, siehe STiNE)

```
 /* Bibliothek mit Funktionen zum Sortieren von Zahlen
    (Source-code file) */
#ifndef SORTING_H
#define SORTING_H

/* Sortiert das Array <values> mit <nofelements> Elementen. Mit Hilfe der
   insertionsort Methode */
void insertionsort(unsigned int *values, unsigned long nofelements);

/* Sortiert das Array <values> mit <nofelements> Elementen. Mit Hilfe der
   countingsort Methode */
void countingsort(unsigned int *values, unsigned long nofelements);
#endif
```

Verwenden Sie dabei die aus „Algorithmen und Datenstrukturen“ bekannten Algorithmen „Insertion Sort“ und „Counting Sort“ (siehe auch: Cormen, Leiserson, Rivest: „Introduction to Algorithms“, MIT Press). Der Parameter `values` zeigt jeweils auf das Array der zu sortierenden Zahlen; `nofelements` gibt an, aus wievielen Elementen das Array besteht.

Implementieren Sie nun in `sorting_main.c` ein Testprogramm, das folgende Konstanten verwendet:

```
#define MAXARRAYELEMENTS 1000000
#define MAXSORTVALUE 1000U
```

und wie folgt arbeitet:

- Der Benutzer übergibt die gewünschte Anzahl an Zufallszahlen per Komandozeilenparameter. Diese soll größer 0 und kleiner gleich `MAXARRAYELEMENTS` sein. Dann wird die gewünschte Anzahl von Zufallszahlen mit Werten zwischen 0 und `MAXSORTVALUE` erzeugt und in einem Array der passenden Größe abgelegt. Dieses kann durch die folgenden Anweisungen realisiert werden:

```
srand48(42349421);
for (i = 0; i < nofelements; i++)
  values[i] = (unsigned int) (drand48() * (MAXSORTVALUE+1));
```

- Danach werden die Zufallszahlen mit `insertionsort` sortiert.
- Nun werden erneut die gleichen Zufallszahlen in einem neuen Array erzeugt und mit `countingsort` sortiert (man kann auch vorher eine Kopie anlegen).
- Beide sortierten Arrays sollen verglichen werden und im Falle eines Unterschiedes soll eine Fehlermeldung ausgegeben werden.
- Schließlich wird die Dezentilverteilung ausgegeben. Dazu wird der Zahlenbereich zwischen kleinster und grösster erzeugter Zufallszahl in 10 gleich große Zahlenbereiche (Dezentile) aufgeteilt und die Anzahl der Zufallszahlen je Dezentil bestimmt.

Testen Sie Ihre Sortierfunktionen für die Arraygrössen 1000, 10000, 100000, 1000000. Lassen Sie sich, wie unten im Beispiel, anzeigen, wann Ihr Testprogramm welche Befehle ausführt. Welche Aussagen können Sie daraus über die relative Laufzeit von `insertionsort` und `countingsort` ableiten?

Unten sehen Sie ein Beispiel für die gewünschte Ausgabe des Testprogramms.

```
# Erzeugung der Zufallszahlen...
# kopieren der gleichen Zufallszahlen
# Sortieren der Zufallszahlen mit Insertion Sort...
# Sortieren der Zufallszahlen mit Counting Sort...
# Vergleich der Arrays...
# Arrays identisch!
Kleinste erzeugte Zufallszahl: 0
Groesste erzeugte Zufallszahl: 1000
Dezentilverteilung:

 Dezentil | #Werte |
----------+--------+
    0-100 |    103 |
  101-199 |    108 |
  204-300 |    105 |
  303-400 |    113 |
  401-499 |     82 |
  501-600 |     95 |
  601-700 |    105 |
  703-800 |     89 |
  802-898 |    101 |
 901-1000 |     99 |
```

Gestalten Sie die Ausgabe Ihres Programmes so, dass die Spaltenbreite der Dezentiltabelle wie im Beispiel immer optimal ist (u.a. auch bei Änderung der Konstanten `MAXSORTVALUE` bzw. bei mehr oder weniger erzeugten Zufallszahlen). Nutzen Sie dazu die Möglichkeit, im `printf()`-Befehl die Breite der Ausgabefelder dynamisch anzupassen, z.B.

```
printf("...%*s...", fieldwidth, string);
```

Für die dynamische Bestimmung der erforderlichen Feldbreite können Sie die Funktion `log10` verwenden. Binden Sie dazu die Datei `math.h` ein. Schauen Sie in das Makefile das wir ihnen mitgeben und Überprüfen Sie ob die Option `-lm` gesetzt ist.

Die Lösungen zu diesen Aufgaben werden am 04.06.2015 besprochen.