Stefan Kurtz **Universität Hamburg**
Mathias von Behren **Zentrum für Bioinformatik**

Programmierung in der Bioinformatik
Wintersemester 2015
Übungen zur Vorlesung: Ausgabe am 23.11.2015

Punktevergabe:

- Aufgabe 7.1: 3 Punkte
- Aufgabe 7.2: 7 Punkte

Aufgabe 7.1 1. Definieren Sie mit **`typedef`** eine Struktur `Point`. Diese soll einen Punkt im zweidimensionalen Raum speichern.

Schreiben Sie anschließend eine Funktion **`double`** `distance(Point p1, Point p2)`, die die euklidische Distanz von `p1` und `p2` liefert.

Falls `p1`$= (x_1, y_1)$ und `p2`$= (x_2, y_2)$ dann wird die euklidische Distanz von `p1` und `p2` durch den folgenden Ausdruck berechnet:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Damit Sie die Funktionen **`double`** `sqrt(`**`double`** `x)` (zur Berechnung der Quadratwurzel) und **`double`** `pow(`**`double`** `x, `**`double`** `y)` (zur Berechnung von x^y) verwenden können, müssen Sie **`#include`** `<math.h>` am Anfang Ihrer C-Datei einfügen und beim Link-Schritt den Compiler mit der Option `-lm` aufrufen.

2. Nehmen Sie an, dass ein geschlossenes Polygon durch eine Folge von Punkten im zweidimensionalen Raum definiert ist.

   Schreiben Sie eine Deklaration für eine struct `Polygon`. Bedenken Sie, dass nicht jedes Polygon durch gleich viele Punkte definiert sein muss, die Anzahl der Punkte muss also Teil der Struktur sein.

   Schreiben Sie eine Funktion
   `void` `polygon_add_point(Polygon *poly, `**`const`** `Point *p)`. Diese soll einen weiteren Punkt zu einem bestehenden Polygon hinzufügen. Verwenden Sie hierbei `realloc` um den reservierten Speicher für das Array von Punkten im Polygon dynamisch zu erweitern.

   Schreiben Sie eine Funktion **`double`** `perimeter(`**`const`** `Polygon *shape)`. Diese soll den Umfang eines Polygons referenziert durch das Argument `shape` liefern.

3. Testen Sie in Ihrer `main`-Funktion die von Ihnen implementierten Funktionen. Hierfür laden Sie sich in STINE die Datei points.csv herunter. In dieser Datei stehen die Koordinaten von 1 000 Punkten, hierbei ist je ein Punkt pro Zeile angegeben und der x- und der y-Wert sind mit dem Tabulator-Zeichen getrennt. Lesen Sie alle Punkte in Ihrer `main`-Funktion ein und fügen Sie sie einem Polygon hinzu. Wenn alles richtig funktioniert sollte ihre Funktion einen Wert von 90.401289 für den Umfang berechnen. Um die mögliche Ungenauigkeit von **`float`** Variablen zu berücksichtigen erlauben Sie bei Ihrer Überprüfung eine Abweichung von 10^{-6}.

Aufgabe 7.2 Schreiben Sie eine C-Funktion `makepartition`, die das $Partitions$-Problem löst. Die Funktion soll eine Menge M von positiven ganzen Zahlen (gegeben als Array), in zwei disjunkte

Untermengen A und B zerlegen, so dass $M = A \cup B$ und $\sum_{i \in A} i = \sum_{i \in B} i$ gilt, d.h. die Summe der Zahlen in beiden Teilmengen ist identisch. Falls eine solche Zerlegung nicht existiert, dann soll eine entsprechende Meldung ausgegeben werden.

Das Problem lässt sich lösen, indem man alle möglichen Aufteilungen von M in A und B generiert und dabei jeweils die Summe der Elemente in A aktualisiert. Man muß also für alle noch nicht betrachteten Elemente beide Möglichkeiten der Zuordnung zu einer der beiden Mengen ausprobieren, und dabei jeweils testen, ob die Auswahl zum Erfolg führt. Man kann sich diese Idee veranschaulichen als binären Entscheidungsbaum mit $n = |M|$ Ebenen. In der Ebene i, $0 \leq i \leq n-1$ werden die Entscheidungen für das i-te Element von M dargestellt. Verfolgt man die linke Kante eines Knotens der Ebene i, dann ordnet man i der Menge A zu. Verfolgt man die rechte Kante eines Knotens der Ebene i, dann ordnet man i der Menge B zu.

Schreiben Sie eine rekursive Funktion `makepartition`, welche den Entscheidungsbaum durchsucht. Sie brauchen natürlich nicht explizit einen Baum mit Knoten und Kanten aufbauen. Jedoch sollte die Aufrufstruktur Ihrer Funktion der Struktur des Baumes folgen. Muss der Entscheidungsbaum immer vollständig durchlaufen werden? Nach welchem Kriterium können einzelne Äste des Entscheidungsbaumes vorzeitig *abgeschnitten* werden (Branch and Bound)?

Deklarieren Sie für Ihre Funktion einen eigenen **`struct`**-Typ `Partition` mit mindestens den Komponenten `values`, `nofvalues`, `totalsum`, `isinA`. Dabei ist `values` ein Array mit den Elementen in M. `nofvalues` ist die Anzahl der Elemente in M. `totalsum` ist die Summe der Elemente in M. `isinA` ist ein Array, das die Zuordnung zur Untermenge A spezifiziert. D.h `isinA[i]` hat den Wert `true` gdw. das i-te Element zu A gehört.

Ihre Funktion zur Zerlegung ist dann wie folgt deklariert:

```
int makepartition(Partition *part, int nextelem)
```

Dabei ist `part` ein Zeiger auf die aktuelle generierte Partition und `nextelem` der Index des nächsten Elementes von M, das ausprobiert werden soll. Die Funktion liefert 0 zurück, wenn eine korrekte Zerlegung gefunden wurde, sonst -1. Welche Zeit- und Speicherplatz-Effizienz hat Ihre Funktion?

Testen Sie Ihre Funktion, indem Sie ein Array von Zufallszahlen generieren und es dann versuchen zu zerlegen. Nutzen Sie in Ihrem Programm `partition.x` bitte die folgenden Konstanten:

```
#define MAXNOFELEMENTS      5000
#define SEEDVALUE       42349421
#define MAXVALUE           10000
```

Die erste Konstante gibt an, dass maximal 5000 Zufallszahlen erzeugt werden. Benutzen Sie `SEEDVALUE` bei der Initialisierung des Zufallsgenerators mit der C-Funktion `srand48()`. Sie können die Zufallszahlen dann mit der C-Funktion `drand48()` generieren. (Siehe hierzu die Online-Manual Einträge, die Sie durch `man drand48` erhalten.) Nutzen Sie `MAXVALUE`, um die erzeugten Zufallszahlen auf den (ganzzahligen) Wertebereich von 1 bis `MAXVALUE` zu skalieren.

Unten sehen Sie zwei mögliche Ausgaben des Programms, das mit einem Argument (nämlich der Grösse der Zahlenmenge) aufgerufen wird. Sie können auf dieses Argument über die Parameter `argv` und `argc` der `main`-Funktion zugreifen.

Beispiel 1: Aufteilung in Untermengen mit gleicher Summe ist möglich:

```
partition.x 20
Die Zahlenmenge der Groesse 20 kann folgendermassen in zwei
```

```
summengleiche Untermengen mit der Summe 42943 aufgeteilt werden:
Zahlen in Untermenge A:
7483 9527 1101 3460 1460 5043 1774 2637 3508 2638 1383 2929
Zahlen in Untermenge B:
8684 2392 3979 9682 3982 6434 2045 5745
```

Beispiel 2: Aufteilung in Untermengen mit gleicher Summe ist nicht möglich:

```
partition.x 13
Die Zahlenmenge
7483 9527 1101 3460 1460 8684 5043 1774 2637 2392 3979 9682 3508
der Groesse 13 kann nicht in zwei summengleiche Untermengen
aufgeteilt werden!
```

Die Lösungen zu diesen Aufgaben werden am 07.12.2015 besprochen.