Chair for Clinical Bioinformatics
Professor Dr. Andreas Keller
Dr. Christina Backes
Dipl.-Ing. Thomas Großmann
B. Sc. Yvonne Saara Gladbach

# *Bioinformatik 1 - Übungsblatt 3*

**Abgabe bis Dienstag 18.11.2014 um 11:59**

Vorname, Nachname:
Matrikelnummer:

**Anmerkungen** :
Die Übungen dürfen in Gruppen bestehend aus max. 2 Personen abgegeben werden. Der Quellcode ist als Quellcodepaket abzugeben, muss auf den Cip-Pool Rechnern kompilieren, gut dokumentiert und getestet sein (Testfälle sind mit abzugeben!). Schicken Sie Ihre Abgabe als tar.gz Paket per Mail an Ihre Tutorin Yvonne Gladbach (yvonne.gladbach@ccb.uni-saarland.de) mit Betreff: "Übung X, Name Y, Name Z". Das Paket darf keine temporären oder binären Dateien enthalten. Beim Entpacken des Paketes soll ein übergeordnetes Verzeichnis erstellt werden, welches den Quellcode enthält. Der Quellcode muss durch ein mitgeliefertes Makefile kompilierbar sein.

## Aufgabe 1 (30 P.):

Implementieren Sie den in der Vorlesung vorgestellten "Greedy-Algorithmus" unter Verwendung der Graph-Klasse. Gehen Sie dazu wie folgt vor:

(a) Erweitern Sie die Klasse `Graph` um eine Methode `void sortEdges()`. Diese Methode soll die Kanten nach absteigendem Kantengewicht mittels `CompEdge` sortieren. Dabei ist `CompEdge` ein Funktionsobjekt, das Objekte der Klasse `Edge` anhand der Kantengewichte geeignet vergleicht.

(b) Erweitern Sie die Klasse `Graph` um eine Methode `bool joinNodes(Edge& e)`. Diese Methode soll Start- und Zielknoten der Kante zu einem Knoten zusammenfassen, dem neuen Knoten die zusammengefügte Sequenz geben und die ursprünglichen Knoten und die aktuelle Kante löschen. Achten Sie darauf, dass alle überflüssigen Kanten gelöscht werden und alle Start- und Zielknoten der veränderten Kanten korrekt sind. Die Methode soll zurückliefern, ob zwei Knoten verschmolzen worden sind.

(c) Implementieren Sie eine Klasse `Assembler`. Diese Klasse soll neben einem vollständigen Interface eine Methode `readGraph(const std::string& filename)` enthalten, die einen Overlap-Graph einliest und zurückgibt, ob der Graph korrekt eingelesen wurde. Außerdem soll die Klasse eine Methode `Sequence assemble()` haben. Diese Methode soll, so lange es mehr als eine Kante gibt, die `joinNodes` Methode aufrufen und schließlich die assemblierte Sequenz zurückgeben.

Schreiben Sie für jeden Konstruktor, jeden Operator und jede Methode einen Testfall (z.B. in der Methode main). Testen Sie Ihren Assembler mit der Beispieldatei fragments.fta. *Hinweis: Der Standardheader algorithm enthält u.a. die Funktionen remove, replace und sort, die Sie verwenden können.*

**Aufgabe 2 (10 P.):**
In Aufgabe 1 haben Sie den Assembler implementiert. Nun sollen Sie den Overlap-Graph per Hand mit den folgenden Fragmenten konstruieren und die daraus entstehende Konsensus-Sequenz angeben:

- Fragmente: $\{CAATT,\ TGGCA,\ TGCAAT,\ ATTGAC,\ GCATTGCAA\}$

**Aufgabe 3 (10 P.):**
Ein ungerichteter Graph $G = (V, E)$ heißt *Hamilton-Graph* oder *hamiltonsch*, wenn es einen *Hamilton-Kreis* in ihm gibt, d.h. es gibt einen Kreis, der jeden Knoten in $V$ genau einmal enthält.

Zeigen Sie durch indirekten Beweis, dass jeder einfache Graph (ein ungerichteter Graph ohne Mehrfachkanten und Schleifen) $G = (V, E)$ mit $|V| \geq 3$ und $deg(v_1) + deg(v_2) \geq |V|$ für alle nicht adjazenten $v_1, v_2 \in V$ ein Hamilton-Graph ist. Dabei bezeichnet $deg(v)$ den Grad des Knoten $v$.

*Hinweis: Gehen Sie dazu von einem kantenmaximalen, nicht hamiltonschen Graph aus, der die Voraussetzungen erfüllt. Kantenmaximal bedeutet ein Graph, der durch Hinzunahme einer weiteren Kante hamiltonsch wird. Verwenden Sie dann das Schubfachprinzip um zu einem Widerspruch zu gelangen.*