BIM – Übung 2

Aufgabe 1

Aufgabe 1.1

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * Klasse, welche das Greedy-Superstring-Verfahren simuliert
 */
public class GreedySuperString {
    //<editor-fold desc="Felder">
    private static String fragment1;
    private static String fragment2;
    private static int besteDeckung;
    private static int indexBesteDeckung;
    //</editor-fold>

    /**
     * Hauptprogramm.
     *
     * @param args Kommandozeilen-Argumente
     */
    public static void main(String[] args) throws IOException{
        File dir = new File(".");

        File unbekannterTextDatei = new File(dir.getCanonicalPath() +
              File.separator + "src" + File.separator + "unbekannterText.txt");
        List<String> unbekannterText = ladeFragmente(unbekannterTextDatei);
        List<String> unbekannterTextTeilstringfrei =
              entferneTeilstrings(unbekannterText);
        String superStringUnbekannterText =
              GreedySuperstring(unbekannterTextTeilstringfrei);
        ausgabedesSuperstrings(superStringUnbekannterText);

        File unbekannteDNA1Datei = new File(dir.getCanonicalPath() + File.separator
              + "src" + File.separator + "DNAFragmente1.txt");
        List<String> unbekannteDNA1 = ladeFragmente(unbekannteDNA1Datei);
        List<String> unbekannteDNA1Teilstringfrei =
              entferneTeilstrings(unbekannteDNA1);
        String superStringUnbekannteDNA1 =
              GreedySuperstring(unbekannteDNA1Teilstringfrei);
        ausgabedesSuperstrings(superStringUnbekannteDNA1);

        File unbekannteDNA2Datei = new File(dir.getCanonicalPath() + File.separator
```

```
            + "src" + File.separator + "DNAFragmente2.txt");
        List<String> unbekannteDNA2 = ladeFragmente(unbekannteDNA2Datei);
        List<String> unbekannteDNA2Teilstringfrei =
                entferneTeilstrings(unbekannteDNA2);
        String superStringUnbekannteDNA2 =
                GreedySuperstring(unbekannteDNA2Teilstringfrei);
        ausgabedesSuperstrings(superStringUnbekannteDNA2);

        File unbekannteDNA3Datei = new File(dir.getCanonicalPath() + File.separator
                + "src" + File.separator + "DNAFragmente3.txt");
        List<String> unbekannteDNA3 = ladeFragmente(unbekannteDNA3Datei);
        List<String> unbekannteDNA3Teilstringfrei =
                entferneTeilstrings(unbekannteDNA3);
        String superStringUnbekannteDNA3 =
                GreedySuperstring(unbekannteDNA3Teilstringfrei);
        ausgabedesSuperstrings(superStringUnbekannteDNA3);
    }

    //<editor-fold desc="Laden und Verarbeitung der Fragmente">
    /**
     * Liest eine Datei Zeile fuer Zeile ein.
     *
     * @param datei Datei, welche eingelesen werden soll
     * @throws IOException IOException, welche geworfen werden kann
     */
    private static List<String> ladeFragmente(File datei) throws IOException{
        List<String> fragmente = new ArrayList<String>();

        BufferedReader br = new BufferedReader(new FileReader(datei));

        String zeile = null;

        //Liest die Datei Zeile fuer Zeile ein
        while((zeile = br.readLine()) != null) {
            fragmente.add(zeile);
        }

        return fragmente;
    }

    /**
     * Entfernt alle Teilstrings aus der Fragmentliste
     *
     * @param fragmentListe Fragmentliste, aus welcher die Teilstrings entfernt
             werden muessen
     * @return Liste mit Teilstring-freien Fragmenten
     */
    private static List<String> entferneTeilstrings(List<String> fragmentListe){
        List<String> teilstringFreieListe = new ArrayList<String>();

        Set<String> uniqueFragmentListe = new HashSet<String>(fragmentListe);

        for(String fragment : uniqueFragmentListe){
            // Nur wenn das Fragment eine Laenge groesser 0 hat, muss es betrachtet
```

```
          werden
         if(fragment.length() > 0){
             List<String> fragmenteMitTeilstrings = new ArrayList<String>();

             for(String vergleichsFragment : uniqueFragmentListe){
                 // Wenn Vergleichsfragment eine Teilstring beinhaltet merken
                 if(vergleichsFragment.length() > 0 &&
                        vergleichsFragment.contains(fragment)){
                     fragmenteMitTeilstrings.add(vergleichsFragment);
                 }
             }

             // Wenn es keine weiteren Fragmente mit Teilstrings gibt, dann zu
              Teilstring-freien Liste hinzufuegen
             if(fragmenteMitTeilstrings.size() == 1){
                 teilstringFreieListe.add(fragmenteMitTeilstrings.get(0));
             }
         }
     }

     return teilstringFreieListe;
 }
 //</editor-fold>

 //<editor-fold desc="Der Greedy-Substring-Algorithmus">
 /**
  * Fuehrt den Greedy-Superstring-Algorithmus durch
  * @param fragmentListe Fragmentliste, die bearbeitet wird
  * @return Superstring
  */
 private static String GreedySuperstring(List<String> fragmentListe){
     List<Integer> fragmentIndizes;

     // Laeuft solange mehrere Fragmente vorhanden sind
     while(fragmentListe.size() > 1){

         findeBesteDeckung(fragmentListe);

         String gemergetesFragment = mergeFragmente();

         // Loesche einzelfragmente
         fragmentListe.remove(fragment1);
         fragmentListe.remove(fragment2);

         // Fuege gemergetes, ganzes Fragment hinzu
         fragmentListe.add(gemergetesFragment);
     }

     return fragmentListe.get(0);
 }

 /**
  * Findet die beste Deckungsrate
  *
```

```
     * @param fragmentListe Fragmentliste
     */
    private static void findeBesteDeckung(List<String> fragmentListe){
        fragment1 = null;
        fragment2 = null;
        besteDeckung = 0;
        indexBesteDeckung = 0;

        for(String fragment : fragmentListe) {
            for (String vergleichsFragment : fragmentListe) {

                // Das gleiche Fragment darf nicht mit sich selbst verglichen
                werden
                if (!fragment.equals(vergleichsFragment)) {
                    int indexAktuelleDeckung = 0;
                    int indexVergleichsFragment = 0;

                    int aktuelleDeckung = 0;

                    // Prueft jeden Buchstaben auf Ueberlappung
                    for (int k = 0; k < fragment.length(); k++) {
                        if (fragment.charAt(k) ==
                           vergleichsFragment.charAt(indexVergleichsFragment)) {
                            // Nur der Start der Ueberlappung wird gespeichert
                            if(indexAktuelleDeckung == 0){
                                indexAktuelleDeckung = k;
                            }

                            aktuelleDeckung++;
                            indexVergleichsFragment++;
                        } else {
                            indexAktuelleDeckung = 0;
                            indexVergleichsFragment = 0;
                            aktuelleDeckung = 0;

                            //Da index von vergleichsFragment wieder auf 0 gesetzt
ist, muss nochmal geprueft werden
                            if(fragment.charAt(k) ==
                                 vergleichsFragment.charAt(indexVergleichsFragment)
                           ){
                                indexAktuelleDeckung = k;
                                aktuelleDeckung++;
                                indexVergleichsFragment++;
                            }
                        }
                    }

                    //Nur bessere Ueberlappungen merken
                    if (aktuelleDeckung >= besteDeckung) {
                        besteDeckung = aktuelleDeckung;
                        indexBesteDeckung = indexAktuelleDeckung;

                        fragment1 = new String(fragment);
                        fragment2 = new String(vergleichsFragment);
```

```
                    }
                }
            }
        }
    }

    /**
     * Merged das aktuelle Fragment mit dem optimalen Fragment
     *
     * @return Das gemergete Fragment
     */
    private static String mergeFragmente(){
        String gemergetesFragment = null;

        if(besteDeckung != 0){
            int indexRest = ((fragment1.length() - 1) - indexBesteDeckung) + 1;
            gemergetesFragment = fragment1 + fragment2.substring(indexRest);
        } else {
            //System.out.println("Wir raten.");

            //Wenn keine Ueberlappung ermittelt wurde einfach mergen
            gemergetesFragment = fragment1 + fragment2;
        }

        return gemergetesFragment;
    }
    //</editor-fold>

    //<editor-fold desc="Ausgabemethoden">
    /**
     * Gibt einen Superstring aus.
     * @param superString Der Superstring der ausgegeben werden soll.
     */
    private static void ausgabedesSuperstrings(String superString){
        System.out.println("Superstring: " + superString);
    }
    //</editor-fold>
}
```

Aufgabe 1.2

Ergebnis:

Das Wohltemperierte Klavier (BWV 846–893) ist eine Sammlung von Präludien und Fugen für ein Tasteninstrument von Johann Sebastian Bach in zwei Teilen. Teil I stellte Bach 1722, Teil II 1740/42 fertig. Jeder Teil enthält 24 Satzpaare aus je einem Präludium und einer Fuge in allen Dur- und Molltonarten, chromatisch aufsteigend angeordnet von C-Dur bis h-Moll. Mit dem Begriff Clavier, der alle damaligen Tasteninstrumente umfasste, ließ Bach die Wahl des Instruments für die Ausführung bewusst offen. Die Orgel scheidet in den meisten Fällen aus, da Bach keine separate Pedalstimme notierte oder als solche bezeichnete und die Orgeln seiner Zeit mitteltönig gestimmt waren. Der größte Teil des Werks ist offenbar für

Clavichord oder Cembalo konzipiert. Nach einer Äußerung Johann Nikolaus Forkels hatte Bach eine Vorliebe für das Clavichord. Im Nekrolog von 1754 steht dagegen über Bach: Die Clavicymbale wußte er, in der Stimmung, so rein und richtig zu temperiren, daß alle Tonarten schön und gefällig klangen. Das Werk wird heute sowohl auf dem Cembalo als auch auf dem modernen Klavier bzw. Flügel gespielt.

Erkenntnis:

Die Lösung ist eindeutig, bei jedem Durchgang konnte ein Overlap ermittelt werden.

Aufgabe 1.3

Ergebnis:

TGCTTTCCATTTTGAATATTAATATGACAGGAAATATCAGATGGAAATATTGACATCTGTA
ATGTATGTCAATAAATGAATTCTAAGTTAGTAGAGTTTGATGATTTTTTTTCTGTTCCTAA
AAAAAGGAAGGGAGAAGAGAGGAAAAGATGTTCAGGGAGCTACCATTTTGTTTCTAGCT
GTGATTTTATAAAATGATAGACACTTTTATCTTTGTGTTACGTTCCTACCCCCAGATGAG
AAAACTGAGACTCAAAAAATACAAGTGACCCGTCCACAGGCAGATAGTTAGGAATTGGG
TTTTTATTCAGAAGGGAGGGGCAGGAGCCGTTTCCTGAAACCTCCTGCATAGGGCATTT
TCGAGAGATTGCACCATCAAATAATATTAGTGATAAATAAGAAGGCAGGAAGAAACTTTT
GGAGGTGATGGATAGGTTTATGGTATAGATTGTGGCCTGACTTACTTTAGTAATAAAATT
GTCCAAGGACTAAATTTATAGATAAGATACCTCTTTGTCTCCTTATTGACAGAGTGAATG
GGGCAACTGTGGTGGAAGAAATGTGTGTATGTATGAGTGTGTGATGGAGCTAACTTTTC
TACAATGTCTACTAACATGTCCTAGCCTTTACTTCATTCGCCTGTTTCCTTCTCACAAAAA
CCCTGTATGGGAGTTTTTCTTTACTTTTTATTTGAGACAAAGTCTCGCTCTGTCTCCCAG
GCTGGAGTGCAGTGGCGCTATATCGGCTCACTGCAGCCTCCACCTCCCGGGTTCAAGC
GATTCTCCTGCCTCAGCCTCCTGAGTAGCTGGTACTACAGGCGTGCACCACCATGCCA
CTATTTTTTGTATTTTTAGTAGCCAGGAGTTTTTCTTATACTCATTTTATTCAGAAGGGAG
GGGCAGGAGGGAATGACAAGTGACTCACCTTGAATTCTTCCTCTAAGAAACTCACACCT
GAGCTTTGAGCTATAAAGAAATCTGATGCTGTTTCTGGGATCTGGAGAGGAAGACTCAG
TCCAGAATCCTCCCAGGGCCTTGAAAGTCCATCTCTGACCCAAAACAATCCAAGTAAGT
ACCTAATTCCTTTGGGAGTGGGTTGTGTATCTCACAGCAACAGAGAAAAAATAGTCACT
TAAAAGTTTCTCTTTGACATCTGTAATGTATGTCAATAAAAAAGAGAGAAACTAAAAAACA
AAAAGAAGCAGAAGCAAAAGTTAATGAGTCTTAACAGTTGCTTACCTATTGAAAACTTAT
TTAGAAATACTCTTTTAACATTGTGGTCACCTGAGTAAATCACTGGAGATAGTGCATTTC
AGAAATGTCTCCGTTCTGATTCCATAAACAATTTGACTTGTATAGTGTGCTATATTTTGGT
GATTTATCAAATCTTGATGTGAGTTTGGGAGTATTGCTAATGTCAGATGACTTGGGAACT
AAGAATAAGACATTTAACCTATGCTTAATTGAAATGAAATTTTTCCCAGTGTATGCCTATC
CCCAGACTCTCTCCTCCTCCTCACCTCATTGTCTCCCCGACTTATCCTAATGCGAAGAA
ATCTGATGCTGTTTCTGGTGCTGTCTTAGAATCACTTCAGGAGTATTGACAAGAGGGGT
AGGAACCCTTAGCGGGACCTGAACTTGAGGGCGGGTCTTTCTGACTCCAAAGCCTCTT
CCTGGCTACTCTGATATTGGCTATTGGCGGAGGCTGGGAAAACTTGAAATGGGGAATG
TATACATGGAATATGTAAAGCTTTTATATGTCAGTCACACCTCAGTAAAGTGGTTTACCT
ATCTATCTATCTATCTATCTATCTATCTAAATTTTTTTTTCTGTTCCTGAGGATGTTGCAAC
AAATACTGATGCAACTCCTGGTTAACTGATAAAGTACTGGCCAGGGACAAAGCTCTCTT

GCAGCAATTTCCCACCACGTACCTCTGCCCTCTCCTCACAGCTGGAGAGGGAAAGTCA
TGGAATCCTTGTCCTTCCTCTTGTTTCCACCTCTTCAAGATTGGGCCAATTGCAATGGAA
TATCCATTGGTTGTGAGGCCTTTGTACTCTGCAAGGAAAAGAAAAGAGACGGGGTTTCA
CTATGTTGGCCAGACTGGTCTCGAACTCTTGACCTCAGGTGATCCGCCCGCCTCGGCT
TCCCAGAGTGCTAGGATTACAGGCGTCCTGAGACCCTTCCTCCAGTCCTCCAAATTATG
GATCTGTGCCATTTGTACCGTGGACTTTTCTGTTTTCGCATTCCAGCCTGACAGGGGTG
ATTTGTAGCAAAATCGTGACTTCAGTTGTAAACTCTTCTATTATTATTATTTTTTTGAGACT
TTTTAAAAGATAGA

Erkenntnis:

Ab den letzten 6 Fragmenten gibt es keine Overlaps mehr -> keine Eindeutige Lösung kann gefunden werden. (In unserem Algorithmus werden die letzten zwei betrachteten Fragmente gemerged).

Aufgabe 1.4

Ergebnis:

TTGACATCTGTAATGTATGTCAATAAATGAATTCTAAGTTAGTAGAGTTTGATGTAAAGTC
CTGAAAATTAAAAAAGAGAGAAACTAAAAAACAAAAAGAAGCAGAAGCAAAAGTTAATGA
GTCTTAACAGTTGCTTACCTATTTGTAGCAAAATCGCTGGGATCTGGAGAGGAAGACTC
AGTCCAGAATCCTCCCAGGGCCTTGAAAGTCCATCTCTGACCCAAAACAATCCAAGTAA
GTACCTAATTCCTTTGGGAGTGGGTTGTGTATCTCACAGCAACAGAGAAAAAATAGTCA
CTTAAAAGTTTCTCTTTGACATCTGTAATGTATGTCAATAAAACAAAAAGAAGCAGAAGC
AAAAGTTAATGAGTCTTAACAGTTGCTTACCTATTGAAAACTTATTTAGAAATACTCTTTT
AACATTGTGGTCACCTGAGTAAATCACTGGAGATAGTGCATTTCAGAAATGTCTCCGTT
CTGATTCCATAAACAATTTGACTTGTATAGTGTGCTATATTTTGGTGATTTATCAAATCTT
GATGTGAGTTTGGGAGTATTGCTAATGTCAGATGACTTGGGAACTAAGAATAAGACATT
TAACCTATGCTTAATTGAAATGAAATTTTTCCCTAGAAGAAGAGTAGGTGGAAAAAGTCT
TCTTTCTTGACTTCAGTTGTAAACTCTTCTATTGCTTTCCATTTTGAATATTAATATGACA
GGAAATATCAGATGGAAATATTTTTAAAAGATAGAAATGTGAGTATGACGAAGAACTCTC
TCCTCCTCCTCACCTCATTGTCTCCCCGACTTATCCTAATGCGAAATTGGATATCTATCT
ATCTATCTATCTATCTATCTAAATTTTTTTTTCTGTTCCTCTTGTCCTGAGACCCTTCCTCA
AGATTTGCAGCAATTTCCCACCACGTACCTCTGCCCTCTCCTCACAGTGTATGCCTATC
CCCAGACTCATCAAAGTGTATACATGGAATATGTAAAGCTTTTATATGTCAGTCACACCT
CAGTAAAGTGGTTTACCTATCTATCTATCTATCTATCTATCTATCTATCTAAGAAATGTGT
GTATGTATGAGTGTGTGATGGAGCTAACTTTTCTACAATGTCTACTAACATGTCCTAGCC
TTTACTTCATTCGCCTGTTTCCTTCTCACAAAAACCCTGTATGGGAGTTTTTCTTTACTTT
TTATTATTATTTTTTTGAGACAAAGTCTCGCTCTGTCTCCCAGGCTGGAGTGCAGTGGC
GCTATATCGGCTCACTGCAGCCTCCACCTCCCGGGTTCAAGCGATTCTCCTGCCTCAG
CCTCCTGAGTAGCTGGTACTACAGGCGTGCACCACCATGCCACTATTTTTTGTATTTTTA
GTAGAGACGGGGTTTCACTATGTTGGCCAGACTGGTCTCGAACTCTTGACCTCAGGTG
ATCCGCCCGCCTCGGCTTCCCAGAGTGCTAGGATTACAGGCGTGAGCCACTGCGCCC
AGCCAGGAGTTTTTCTTATACTCATTTTACAGATGAGAAAACTGAGACTCAAAAAATACA
AGTGACCCGTCCACAGGCAGATAGTTAGGAAGTAGCGGGACCTGAACTTGAGGGCGG

GTCTTTCTGACTCCAAAGCCTCTTCCTGGCTACTCTGATATTGGCTATTGGCGGAGGCT
GGGAAAACTTGAAATGGGGAATGATCGGGGAGCGGCGAGGGGGGACCAGCCGTTAAG
CATTCCAGCCTGACAGGGGTGATTTGTTAAACCCAGGAACTAGTTAGACGTTTCCTGAA
ACCTCCTGCATAGGGCATTTTCGAGAGATTGCACCATCACCCAGTCCTCCAAATTATGG
ATCTGTGCCATTTGTACCGTGGACTTTTCTGTTTTCTGAGGATGTTGCAACAAATACTGA
TGCAACTCCTGGTTAACTGATAAAGTACTGGCCAGGGACAAAGCTCTCTTGCAGCAATT
TCCCACCACGTACCTCTGCCCTCTCCTCACAGCTGGAGAGGGAAAGTCATGGAATCCT
TGTCCTTCCTCTTGTTTCCACCTCTTCAAGATTGGGCCAATTGCAATGGAATATCCATTG
GTTGTGAGGCCTTTGTACTCTGCAAGGAAAAGAAAAGAAATGTGTGTATGTATGAGTGT
GTTTTTCTGTTCCTAAAAAAAGGAAGGGAGAAGAGAGGAAAAGATGTTCAGGGAGCTAC
CATTTTGTTTCTAGCTGTGATTTTATAAAATGATAGACACTTTTATCTTTGTGTTACGTTC
CTACCCCCAGTTATTCAGAAGGGAGGGGCAGGAGGGAATGACAAGTGACTCACCTTGA
ATTCTTCCTCTAAGAAACTCACACCTGAGCTTTGAGCTATAAAGAAATCTGATGCTGTTT
CTGGTGCTGTCTTAGAATCACTTCAGGAGTATTGACAAGAGGGGTAGGAACCCTTAGAA
ATAATATTAGTGATAAATAAGAAGGCAGGAAGAAACTTTTGGAGGTGATGGATAGGTTTA
TGGTATAGATTGTGGTGATGATTTAATGATTAAAAGATAGAAATGTGAGTATGACGAAGA
ACTTTAGTAATAAAATTGTCCAAGGACTAAATTTATAGATAAGATACCTCTTTGTCTCCTT
ATTGACAGAGTGAATGGGGCAACTGTGGAGCCTGACTTACTTCTTTTAATTGGGTTTTTA
TTCAGAAGGGAGGGGCAGGAGGGAATGTTTTTTTTCTGTTCCTAAAAAAAGGAAGG

Erkenntnis:

Ab den letzten 5 Fragmenten gibt es keine Overlaps mehr -> keine Eindeutige Lösung kann gefunden werden. (In unserem Algorithmus werden die letzten zwei betrachteten Fragmente gemerged).

Aufgabe 1.5

Ergebnis:

TATCTATCTATCTATCTATCTATCTATCTAAATTTTTTTTTCTGTTCCTTGCAGCAATTTCC
CACCACGTACCTCTGCCCTCTCCTCACAGCTGGAGAGGGAAAGTCATGGAATCCTTGT
CCTTCCTCTTGTTTCCACCTCTTCAAGATTGGGCCAATTGCAATGGAATATCCATTGGTT
GTGAGGCCTTTGTACTCTGCAAGGAAAAGAAAAGAAATGTGTGTATGTATGAGTGTGTG
ATGGAGCTAACTTTTCTACAATGTTATTCAGAAGGGAGGGGCAGGAGGGAATGACAAGT
GACTCACCTTGAATTCTTCCTCTAAGAAACTCACACCTGAGCTTTGAGCTATAAAGAAAT
CTGATGCTGTTTCTGGTGCTGTCTTAGAATCACTTCAGGAGTATTGACAAGAGGGGTAG
GAACCCTTAGAAATAATATTAGTGATAAATAAGAAGGCAGGAAGAAACTTTTGGAGGTG
ATGGATAGGTTTATGGTATAGATTGTGGTGATGATTTAATGAGTGTATGCCTATCCCCAG
ACTCATCAAAGTGTATACATGGAATATGTAAAGCTTTTATATGTCAGTCACACCTCAGTA
AAGTGGTTTACCTATCTATCTATCTATCTATCTATCTATCTATCTAAATTTTTTTTTCTGTT
CCTAAAAAAAGGAAGGGAGAAGAGAGGAAAAGATTAGTAATAAAATTGTCCAAGGACTA
AATTTATAGATAAGATACCTCTTTGTCTCCTTATTGACAGAGTGAATGGGGCAACTGTGG
AGCCTGACTTACTTCTTTTAATTGGGTTTTTATTCAGAAGGGAGGGGCAGGAGGGAATG
ACAAGAGTAGGTGGAAAAAGTCTTCTTTCTTGACTTCAGTTGTAAACTCTTCTATTGCTT
TCCATTTTGAATATTAATATGACAGGAAATATCAGATGGAAATATTTTTAAAAGATAGAAA

TGTGAGTATGACGAAGAACTCTCTCCTCCTCCTCACCTCATTGTCTCCCCGACTTATCCT
AATGCGAAATTGGATTCTGAGCATTTGTAGCAAAATCGCTGGGATCTGGAGAGGAAGAC
TCAGTCCAGAATCCTCCCAGGGCCTTGAAAGTCCATCTCTGACCCAAAACAATCCAAGT
AAGTACCTAATTCCTTTGGGAGTGGGTTGTGTATCTCACAGCAACAGAGAAAAAATAGT
CACTTAAAAGTTTCTCTTTGACATCTGTAATGTATGTCAATAAATGAATTCTAAGTTAGTA
GAGTTTGATGTAAAGTCCTGAAAATTAAAAAAGAGAGAAACTAAAAAACAAAAAGAAGCA
GAAGCAAAAGTTAATGAGTCTTAACAGTTGCTTACCTATTAAAAGATAGAAATGTGAGTA
TGACGAAGAACTTTAGTAATAAAATTGTCCAAGGACTAAATTTATAGATAAGAAATGTGT
GTATGTATGAGTGTGTGATGGAGCTAACTTTTCTACAATGTCTACTAACATGTCCTAGCC
TTTACTTCATTCGCCTGTTTCCTTCTCACAAAAACCCTGTATGGGAGTTTTTCTTTACTTT
TTATTATTATTTTTTTGAGACAAAGTCTCGCTCTGTCTCCCAGGCTGGAGTGCAGTGGC
GCTATATCGGCTCACTGCAGCCTCCACCTCCCGGGTTCAAGCGATTCTCCTGCCTCAG
CCTCCTGAGTAGCTGGTACTACAGGCGTGCACCACCATGCCACTATTTTTTGTATTTTTA
GTAGAGACGGGGTTTCACTATGTTGGCCAGACTGGTCTCGAACTCTTGACCTCAGGTG
ATCCGCCCGCCTCGGCTTCCCAGAGTGCTAGGATTACAGGCGTGAGCCACTGCGCCC
AGCCAGGAGTTTTTCTTATACTCATTTTACAGATGAGAAAACTGAGACTCAAAAAATACA
AGTGACCCGTCCACAGGCAGATAGTTAGGAAGTAGCGGGACCTGAACTTGAGGGCGG
GTCTTTCTGACTCCAAAGCCTCTTCCTGGCTACTCTGATATTGGCTATTGGCGGAGGCT
GGGAAAACTTGAAATGGGGAATGATCGGGGAGCGGCGAGGGGGGACCAGCCGTTAAG
CATTCCAGCCTGACAGGGGTGATTTGTTAAACCCAGGAACTAGTTAGACGTTTCCTGAA
ACCTCCTGCATAGGGCATTTTCGAGAGATTGCACCATCAAAACAAAAAGAAGCAGAAGC
AAAAGTTAATGAGTCTTAACAGTTGCTTACCTATTGAAAACTTATTTAGAAATACTCTTTT
AACATTGTGGTCACCTGAGTAAATCACTGGAGATAGTGCATTTCAGAAATGTCTCCGTT
CTGATTCCATAAACAATTTGACTTGTATAGTGTGCTATATTTTGGTGATTTATCAAATCTT
GATGTGAGTTTGGGAGTATTGCTAATGTCAGATGACTTGGGAACTAAGAATAAGACATT
TAACCTATGCTTAATTGAAATGAAATTTTTCCCTAGAAGAAGAGTAGGTGGAAAAAGTCT
TCTTTCTTGACTTCAGTTTTCTGTTCCTAAAAAAAGGAAGGGAGAAGAGAGGAAAAGAT
GTTCAGGGAGCTACCATTTTGTTTCTAGCTGTGATTTTATAAAATGATAGACACTTTTATC
TTTGTGTTACGTTCCTACCCCCAGTCCTCCAAATTATGGATCTGTGCCATTTGTACCGTG
GACTTTTCTGTTTTCTGAGGATGTTGCAACAAATACTGTTGTACCGTGGACTTTTCTGTT
TTCTGAGGATGTTGCAACAAATACTGATGCAACTCCTGGTTAACTGATAAAGTACTGGC
CAGGGACAAAGCTCTCTTGTCCTGAGACCCTTCCTCAAGATTTGCAGCAATTTCCCACC
ACGTACCTCTGCCCTCTCCTCACAGTTTATTCAGAAGGGAGGGGCAGGAGGGAATGAC
AAGTGACTATCTATCTATCTATCTATCTAAATTTTTTTTTCTGTTCCTAAAAAAAG

Erkenntnis:

Ab den letzten 6 Fragmenten gibt es keine Overlaps mehr -> keine Eindeutige Lösung kann gefunden werden. (In unserem Algorithmus werden die letzten zwei betrachteten Fragmente gemerged).

Aufgabe 2

Aufgabe 2.1

```
package uebung2.aufgabe2;

import net.gumbix.dynpro.DynProJava;
import net.gumbix.dynpro.Idx;
import net.gumbix.dynpro.PathEntry;
import scala.Function2;
import scala.Option;
import scala.Some;
import scala.collection.concurrent.Debug;

import java.util.Arrays;
import java.util.List;

/**
 * Created by Dennis on 29.11.2017.
 */
public class GlobalAlignment extends net.gumbix.dynpro.DynProJava<Integer> {

    //Nucleotid Sequences
    private String[] s = (" CGATCCTGT").split("");
    private String[] t = (" CATCGCCTT").split("");

    public static void main(String[] args) {

        GlobalAlignment ga = new GlobalAlignment();

        List<PathEntry<Integer>> solutionJava = ga.solutionAsList(new
Idx(ga.n() - 1, ga.m() - 1));
        System.out.println("Optimal Decisions:");
        for (PathEntry<Integer> entry : solutionJava) {
            System.out.print(entry.decision() + " ");

        }

        System.out.println("\n");
        System.out.println(ga.mkMatrixString(ga.solution(new Idx(ga.n() - 1,
ga.m() - 1))));
    }

    //TODO check why values of diagonal (idx.i() = idx.j()) are wrong
    @Override
    public Object decisions(Idx idx) {
        //Start
        if (idx.i() == 0 && idx.j() == 0) {
            System.out.println(idx.i() + " " + idx.j() + " START");
```

```
            return new Integer[]{0};
            //Insertion
        } else if (idx.i() == 0 && idx.j() > 0) {
            System.out.println(idx.i() + " " + idx.j() + " INSERT");
            return new Integer[]{1};
            //Deletion
        } else if (idx.j() == 0 && idx.i() > 0) {
            System.out.println(idx.i() + " " + idx.j() + " DELETION");
            return new Integer[]{2};
            //1 Match, -1 Missmatch
        } else {
            System.out.println(idx.i() + " " + idx.j() + " INSERT, DELETION,
BOTH");
            //TODO Check...
            return new Integer[]{1,2,3};
        }
    }

    //should be correct, just like on the slides "BIM-40" page 18
    @Override
    public Idx[] prevStates(Idx idx, Integer d) {

        //1 for Insert
        if(d == 1){
            return new Idx[]{new Idx(idx.i(), idx.j() - 1)};
            //2 for Deletion
        } else if(d == 2){
            return new Idx[]{new Idx(idx.i() - 1, idx.j())};
            //3 for Both
        } else if (d == 3){
            return new Idx[]{new Idx(idx.i() - 1, idx.j() - 1)};
            //0 for Start
        } else {
            return new Idx[]{};
        }
    }

    @Override
    public double value(Idx idx, Integer d) {

        //Start
        if (d == 0) {
            return 0;
            //Insert or Deletion
        } else if (d == 1 || d == 2) {
            return -2;
            //Both
        } else {
            return similarity(s[idx.i()], t[idx.j()]);
        }
    }
```

```
    /**
     * checks if sequences at given index is a match or a mismatch
     *
     * @param s   sequence s
     * @param t   sequence t
     * @return 1 for match, -1 for mismatch
     */
    private int similarity(String s, String t) {
        if (s.equals(t)) {
            return 1;
        } else {
            return -1;
        }
    }

    @Override
    public int n() {
        return s.length;
    }

    @Override
    public int m() {
        return t.length;
    }

    @Override
    public Function2 extremeFunction() {
        return this.MAX();
    }

    @Override
    public String[] rowLabels() {
        return s;
    }

    @Override
    public Option<String[]> columnLabels() {
        String[] cArray = new String[t.length];
        for (int i = 0; i < t.length; i++) {
            cArray[i] = "" + t[i];
        }
        return new Some(cArray);
    }

}
```

Aufgabe 2.2

Global Alignment (Needleman-Wunsch):

```
>> [Score2, Alignment2] = nwalign(aminoT, aminoS, 'Alphabet','AA','ScoringMatrix', aminoscore, 'GapOpen', 2);
>> Score2

Score2 =

   -16

>> Alignment2

Alignment2 =

  3×28 char array

    'KAQYRRE------CMI---FVWEI-NRL'
    '|  ||  ||        |   | |||  | '
    'KIQYKREPNIPSVSLINSLFAWEIRDRI'

>> [Score1, Alignment1] = nwalign(seqT, seqS, 'Alphabet','NT','ScoringMatrix', scorematrix, 'GapOpen', 2);
>> Score1

Score1 =

     0

>> Alignment1

Alignment1 =

  3×10 char array

    'C-ATCGCCTT'
    '| |||  |  |'
    'CGATC-CTGT'
```

Local Alignment (Smith/Waterman):

```
>> [Score1, Alignment1] = swalign(seqT, seqS, 'Alphabet','NT','ScoringMatrix', scorematrix, 'GapOpen', 2);
>> Score1

Score1 =

     3

>> Alignment1

Alignment1 =

  3×3 char array

    'ATC'
    '|||'
    'ATC'
```

```
>> [Score2, Alignment2] = swalign(aminoT, aminoS, 'Alphabet','AA','ScoringMatrix', aminoscore, 'GapOpen', 2);
>> Score2

Score2 =

     3

>> Alignment2

Alignment2 =

  3×7 char array

    'KAQYRRE'
    '| || ||'
    'KIQYKRE'
```

Aufgabe 2.3

Local Alignment (Smith/Waterman):

```
>> [Score3, Alignment3] = swalign(aminoT, aminoS, 'Alphabet','AA','ScoringMatrix', 'Blosum62', 'GapOpen', 5);
>> Score3

Score3 =

   15.5000

>> Alignment3

Alignment3 =

  3×19 char array

    'KAQYRRECMI-FVWEINRL'
    '| ||:||  |  |  || |'
    'KIQYKREPNIPSVSLINSL'
```

Global Alignment (Needleman-Wunsch):

```
>> [Score3, Alignment3] = nwalign(aminoT, aminoS, 'Alphabet','AA','ScoringMatrix', 'Blosum62', 'GapOpen', 5);
>> Score3

Score3 =

    8.5000

>> Alignment3

Alignment3 =

  3×28 char array

    'KAQYRRE------CMI---FVWEI-NRL'
    '| ||:||       :|   |:||| :|:'
    'KIQYKREPNIPSVSLINSLFAWEIRDRI'
```

Gap Penalty 5: bestes Ergebnis für lokales Alignment, da die Sequenz nicht in verschiedene kleine Einzelteile zerlegt wird. Für Globales Alignment spielt es in diesem Fall keine Rolle, da immer auf die gesamte Sequenz ausgerichtet wird. Bei kurzen Sequenzen gibt es kaum andere Möglichkeiten für Alignments.

Das lokale Alignment ist mit Hilfe der Blosum62 Matrix effektiver, als mit der selbst erstellen Scoringmatrix in Aufgabe 2.2

Aufgabe 3

Aufgabe 3.1

		s1	s2	s3	s4	
		GCTTATA	GCTATA	GTTATA	GCTTAGA	Σ
s1	GCTTATA	7	4	4	5	**20**
s2	GCTATA	4	6	4	2	16
s3	GTTATA	4	4	6	2	16
s4	GCTTAGA	5	2	2	7	16

Zentrum c = s1
Blätter B = {s2,s3,s4}

```
MA1 (c und s2):
GCTTATA
|| ||||
GC-TATA

-------------------------

A (c und s3):
GCTTATA
| |||||
G-TTATA

MA2:
GCTTATA
GC-TATA
G-TTATA

-------------------------

A (c und s4):
GCTTATA
||||| |
GCTTAGA
```

```
MA3 (finales Alignment):
GCTTATA
GC-TATA
G-TTATA
GCTTAGA
-------
GCTTATA (Consensus-Sequenz)

Summe der Paare Bewertung = 6+(-3)+(-3)+6+6+0+6 = 18
Consensus Bewertung = 3
```

Aufgabe 3.2

MATLAB code:

```
s1 = 'GCTTATA';
s2 = 'GCTATA';
s3 = 'GTTATA';
s4 = 'GCTTAGA';

scorematrix = eye(4);
scorematrix(scorematrix==0) = -1;

S = {s1,s2,s3,s4};
output = multialign(S,'ScoringMatrix',scorematrix,'GapOpen',2);
seqalignviewer(output)
```

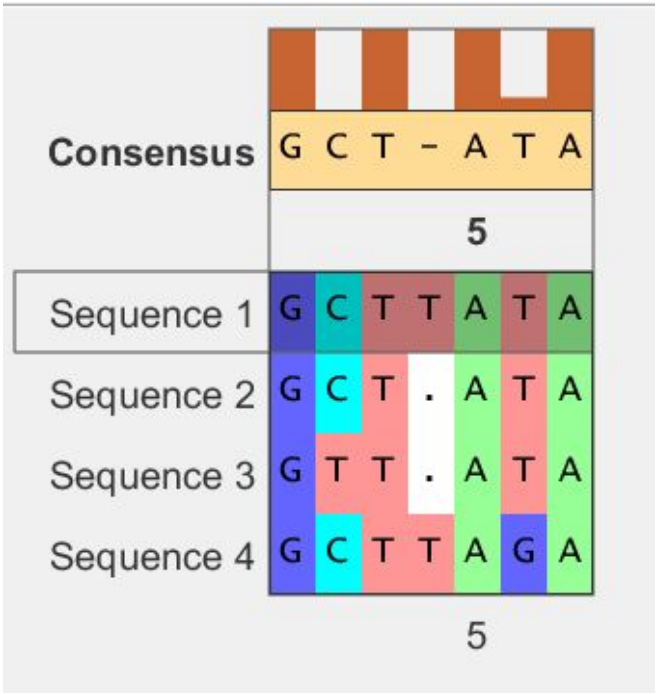

Abbildung 5: Sequence Align Viewer Output