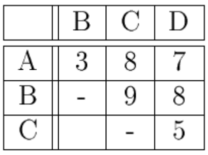
Miguel Caballero, 1414163 Dennis Czogalla, 1410116 Dustin Noah Young, 1412293

BIM - Übung 3

Aufgabe 1.1



	B	C	D
A	3	8	7
B	-	9	8
C		-	5

Tabelle 1: Abstandsmatrix.

a1 = dist(A,B) + dist(C,D) = 3 + 5 = 8

a2 = dist(A,C) + dist(B,D) = 8 + 8 = 16

a3 = dist(A,D) + dist(B,C) = 7 + 9 = 16

ist additiv, wenn gilt:

$$(a_1 = \min(a_1, a_2, a_3) \wedge a_2 = a_3) \vee$$
$$(a_2 = \min(a_1, a_2, a_3) \wedge a_1 = a_3) \vee$$
$$(a_3 = \min(a_1, a_2, a_3) \wedge a_1 = a_2)$$

8 = 8 ∧ 16 = 16 ∨

16 = 8 ∧ 8 = 16 ∨

16 = 18 ∧ 8 = 16

- ➔ Ist wahr; Matrix hat einen additiven Abstand

Aufgabe 1.2

Runde 1:

AB hat geringsten Abstand

Z = {C,D}

	B	Z
A	3	(8+7)/2=7,5
B	-	(9+8)/2=8,5

LGS:

I	a	b	0	3
II	a	0	z	7,5
III	0	b	z	8,5
IV:III-II	-a	b	0	1
V:IV+I	0	2b	0	4

Einsetzen:

b = 2 a = 1 z = 6,5

Runde 2:

AB werden zusammengefasst

	C	D
{A,B}	(8+9)/2= 8,5	(7+8)/2=7,5
C	-	5

CD hat geringsten Abstand, alle anderen Taxa werden zusammengefasst: Z = {A,B}

	D	Z
C	5	(8+9)/2= 8,5
D	-	(7+8)/2=7,5

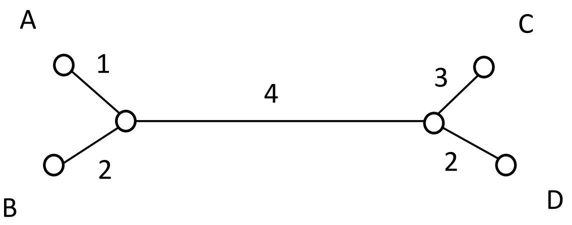
LGS:

I	c	d	0	5
II	c	0	z	8,5
III	0	d	z	7,5
IV:III-II	-c	d	0	-1
V:IV+I	0	2d	0	4

Einsetzen:

d = 2 c = 3 z = 5,5

z = dist(A,C) - c - a = 8 - 3 – 1 = 4



Aufgabe 1.3

Homologe Gene:

```
AB: ACTTA GCCAA TATCC GGGAA   (zur Hilfestellung)
CD: ACTGG ATCAA TATCC GGGAA   (zur Hilfestellung)
A:  CCTTA GCCAA TATCC GGGAA
B:  ATGTA GCCAA TATCC GGGAA
C:  ACTGG ATTGG TATCC GGGAA
D:  ACTGG ATCAA GTTCC GGGAA
```

Aufgabe 1.4

```
A:  CCTTA GCCAA TATCC GGGAA
B:  ATGTA GCCAA TATCC GGGAA
C:  TCTGG ATTGG TATCC GGGAA
D:  ACTGG ATCAA GTTCC GGGAA
```

	B	C	D
A	3	8	7
B	-	10	8
C		-	6

Begründung:

a1 = dist(A,B) + dist(C,D) = 3 + 6 = 9

a2 = dist(A,C) + dist(B,D) = 8 + 8 = 16

a3 = dist(A,D) + dist(B,C) = 7 + 10 = 17

ist additiv, wenn gilt:

$$(a_1 = \min(a_1, a_2, a_3) \wedge a_2 = a_3) \vee$$
$$(a_2 = \min(a_1, a_2, a_3) \wedge a_1 = a_3) \vee$$
$$(a_3 = \min(a_1, a_2, a_3) \wedge a_1 = a_2)$$

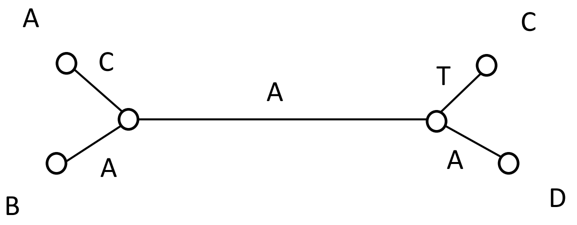
$9 = 9 \wedge 16 = 17 \vee$

$16 = 9 \wedge 9 = 17 \vee$

$17 = 9 \wedge 9 = 16$

➔ falsch, d. h. Matrix ist nicht additiv.

-> Biologisch gesehen passieren zwei Mutationen pro Pfad



Aufgabe 1.5

MATLAB code:

```
sequencesA = 
{'CCTTAGCCAATATCCGGGAA','ATGTAGCCAATATCCGGGAA','ACTGGATTGGTATCCGGGAA','ACTG
GATCAAGTTCCGGGAA'};
sequencesNA = 
{'CCTTAGCCAATATCCGGGAA','ATGTAGCCAATATCCGGGAA','TCTGGATTGGTATCCGGGAA','ACTG
GATCAAGTTCCGGGAA'};

distancesA = [0 3 8 7;3 0 9 8;8 9 0 5;7 8 5 0];
distancesNA = [0 3 8 7;3 0 10 8;8 10 0 6;7 8 6 0];

treeA = seqlinkage(distancesA, 'single', sequencesA);
treeNA = seqlinkage(distancesNA, 'single', sequencesNA);

view(treeA)
view(treeNA)
```

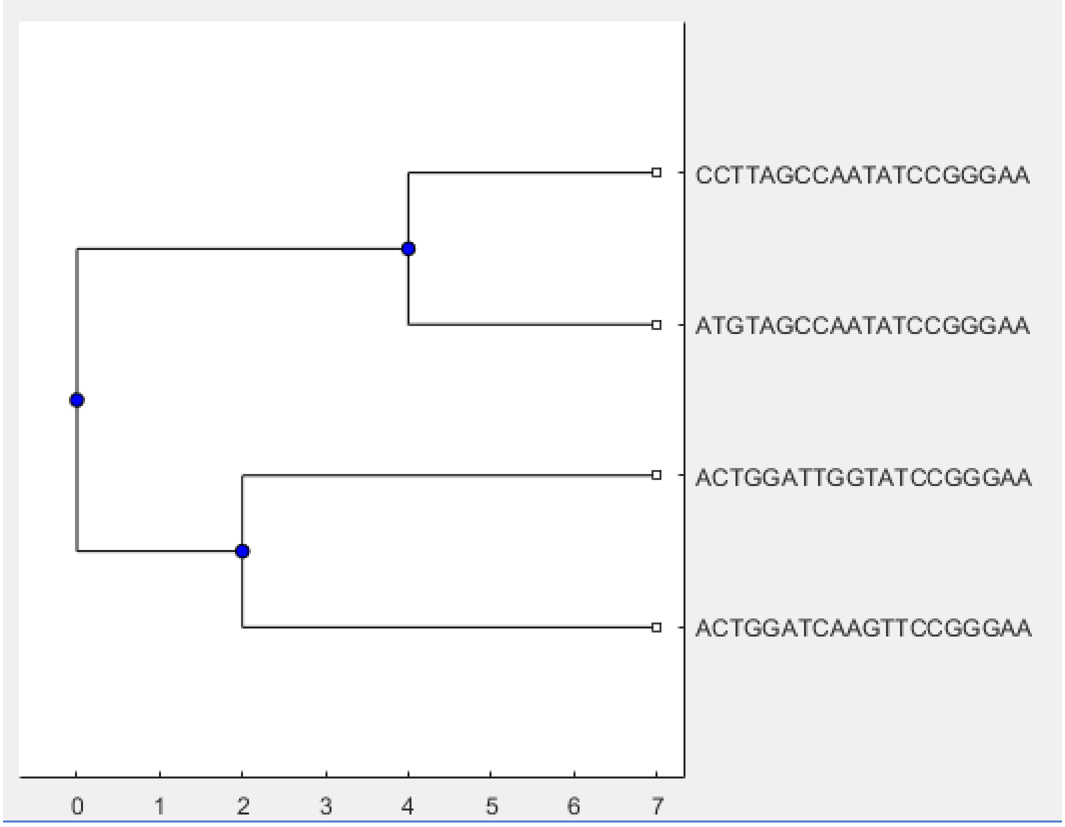


Abbildung 1 additiver phylogenetischer Baum

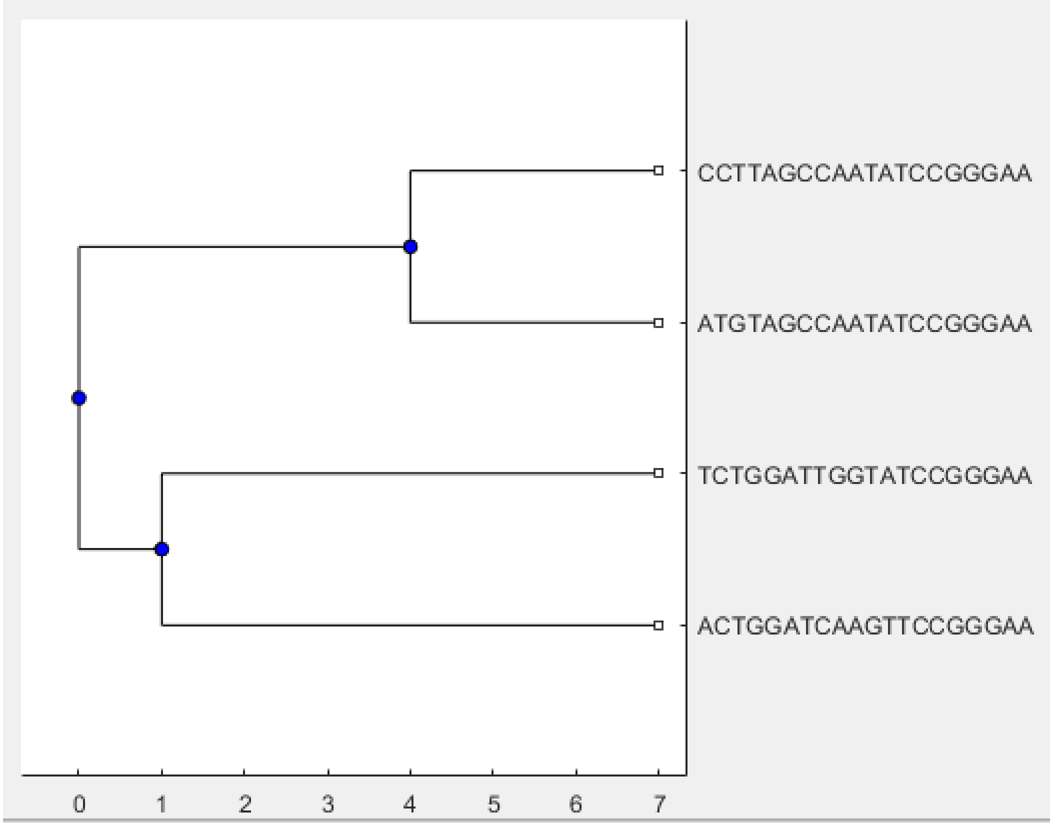


Abbildung 2 nicht-additiver phylogenetischer Baum

Aufgabe 2.1

JAVA Code:

```
package uebung3.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 java.util.List;

/**
 * The Viterbi problem solved with dynamic programming.
 *
 * @author Markus Gumbel (m.gumbel@hs-mannheim.de)
 */
public class Viterbi extends DynProJava<Integer> {

  public static void main(String[] args) {
        String[] rowLabels = {"1", "2", "3", "4", "5", "6"};
        String[] columnStatesLables = {"q0", "F", "U"};
        String[] hmmLabels = {"Wurf-Nr.", "Würfelzahl", "Zustand"};
        int[] states = {0, 1};
        double[][] emission = {{1d/6d, 1d/6d, 1d/6d, 1d/6d, 1d/6d, 1d/6d},{1d/10d, 1d/10d, 
1d/10d, 1d/10d, 1d/10d, 1d/2d}};
        double[][] transition = {{0.5,19d/20d,1d/20d}, {0.5,1d/20d,19d/20d}};

        String diceRoll = "6223153634131564636664355466552134666536666265315 2" +
                "1464516451551424332216461654325215554335255644631 6" +
                "4362443113253656263634162164646166664666335356141 5" +
                "6161554664122114633665653422565666666166464655324 6" +
                "6616661455236563266622152611664243464131436536136 6" +
                "5436154535624642435335626561342254661453662516143 5" +
                "4363361625664661166352334242616414666661412666416 5" +
                "2466655425442133555114266256466454134436563466524 1" +
                "3565353666332666662653666366253636645666645666165 5" +
                "2643446643446535111122141146664644233161353456622 64";


        Viterbi dp = new Viterbi(rowLabels, columnStatesLables, states, emission, transition, 
diceRoll);

        //check which j cell has the highest value and save index j
        double bestValue = -1 * Double.POSITIVE_INFINITY;
        int bestIndex = dp.m()-1;
        for (int j=0; j < dp.m(); j++) {
                double tempValue = 0;
                List<PathEntry<Integer>> entries = dp.solutionAsList(new Idx(dp.n() - 1, j));
                //one cell holds one value, but not the accumulated one; so let's sum it up
                for (PathEntry<Integer> entry : entries){
                        tempValue += entry.value();
                }
                if(tempValue >= bestValue){
                        bestValue = tempValue;
                        bestIndex = entries.get(dp.n()-1).currCell().j();
                }
        }

        //print optimal decisions
        List<PathEntry<Integer>> solutionJava =
                dp.solutionAsList(new Idx(dp.n() - 1, bestIndex));
        System.out.println("Optimal Decisions:");
        for (PathEntry<Integer> entry : solutionJava) {
                System.out.print(entry.decision() + " ");
        }
		//print matrix
		System.out.println("\n");
		scala.collection.immutable.List<PathEntry<Integer>> solution = dp.solution(new 
Idx(dp.n() - 1, bestIndex));
		System.out.println(dp.mkMatrixString(solution));

		//print hidden-markov-modell
		dp.printHMM(solutionJava, hmmLabels);
	}

	/**
	 * prints results in Hidde-Markoc-Model format
	 * @param solution the solution containing the calculated states
	 * @param rowLabels the labels to label the three different rows
	 */
	private void printHMM(List<PathEntry<Integer>> solution, String[] rowLabels) {
		System.out.println();
		System.out.print(rowLabels[0] + "   |");
		for(int i = 0; i < this.n(); i++) {
			System.out.print(i+1 + "|");
		}
		System.out.println();
		System.out.print(rowLabels[1] + " |");
		char[] diceNumbers = this.path.toCharArray();
		for(int i = 0; i < this.path.length(); i++) {
			System.out.print(diceNumbers[i] + "|");
		}
		System.out.println();
		System.out.print(rowLabels[2] + "     |");
		for(PathEntry<Integer> entry : solution) {
			System.out.print(this.columnStatesLables[entry.decision()+1] + "|");
		}
	}

	private int[] states;
	private double [][] emission;
	private double [][] transition;
	private String[] alphabet;
	private String[] columnStatesLables;
	private String path;

	public Viterbi(String[] alphabet, String[] columnStatesLables, int[] states, 
double[][]emission, double[][] transition, String path) {
		this.states = states;
		this.alphabet = alphabet;
		this.emission = emission;
		this.transition = transition;
		this.columnStatesLables = columnStatesLables;
		this.path = path;
		// Defines how values are formatted in the console output.
		// Formatter are: INT, ENGINEER, DECIMAL
		this.formatter_$eq(this.ENGINEER());
	}

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

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

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

		double value = 0;
```

```
        char[] array = path.toCharArray();
        char currentNumber = array[idx.i()];
        int number = Integer.parseInt(currentNumber + "");
        if (idx.i() > 0) {
                value = Math.log10(emission[d][number - 1]) + Math.log10(transition[idx.j()][d
+ 1]);
        } else {
                value = Math.log10(emission[d][number - 1]) +
Math.log10(transition[idx.j()][d]);
        }
        return value;
  }

  /**
   * If the remaining capacity (idx.j) plus the weight that could be taken
   * is less than the overall capacity we could take it. Thus,  { 0, 1 }.
   * If not, we can only skip it (={0}).
   */
  @Override
  public Integer[] decisions(Idx idx) {
        if (idx.i() == 0){
                return new Integer[]{0};
        } else {
                Integer[] decisions = new Integer[this.m()];
                for (int i = 0; i < this.m(); i++){
                        decisions[i] = states[i];
                }
                return decisions;
        }
  }

  /**
   * The prev. state is the previous item (idx.i-1) and the prev. capacity.
   * The prev. capacity is the remaining capacity (idx.j) plus weight that was
   * taken (or plus 0 if it was skipped).
   */
  @Override
  public Idx[] prevStates(Idx idx, Integer d) {
        if (idx.i() > 0) {
                Idx pidx = new Idx(idx.i() - 1, d);
                return new Idx[]{pidx};
        } else {
                return new Idx[]{};
        }
  }

  /**
   * Defines whether the minimum or maximum is calculated.
   *
   * @return
   */
  @Override
  public Function2 extremeFunction() {
        return this.MAX(); // oder MIN()
  }

  /**
   * Provide row labels, i.e. each row gets a short description.
   *
   * @return Array of size n with the labels.
   */
  @Override
  public String[] rowLabels() {
        String[] rowLabels = new String[path.length()];
        char[] pathLabels = path.toCharArray();

        for (int i = 0; i < path.length(); i++) {
                rowLabels[i] =  pathLabels[i] + "";
        }

        return rowLabels;
  }

  /**
   * Provide column labels, i.e. each columns gets a short description.
   * In this case, the column labels are the same as the column index.
   *
   * @return Array of size m with the labels.
   */
  @Override
  public Option<String[]> columnLabels() {
        String[] cArray = new String[states.length];
        for (int i = 0; i < states.length; i++) {
                cArray[i] = columnStatesLables[i+1];
        }
        return new Some(cArray);
  }

}
```

Aufgabe 2.2

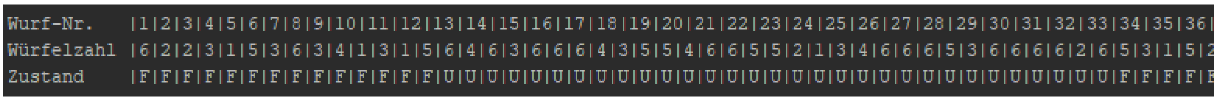
Wurf-Nr.
|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|37|38|39|40|41|42|43|44|45|46|47|48|49|50|51|52|53|54|55|56|57|58|59|60|61|62|63|64|65|66|67|68|69|70|71|72|73|74|75|76|77|78|79|80|81|82|83|84|85|86|87|88|89|90|91|92|93|94|95|96|97|98|99|100|101|102|103|104|105|106|107|108|109|110|111|112|113|114|115|116|117|118|119|120|121|122|123|124|125|126|127|128|129|130|131|132|133|134|135|136|137|138|139|140|141|142|143|144|145|146|147|148|149|150|151|152|153|154|155|156|157|158|159|160|161|162|163|164|165|166|167|168|169|170|171|172|173|174|175|176|177|178|179|180|181|182|183|184|185|186|187|188|189|190|191|192|193|194|195|196|197|198|199|200|201|202|203|204|205|206|207|208|209|210|211|212|213|214|215|216|217|218|219|220|221|222|223|224|225|226|227|228|229|230|231|232|233|234|235|236|237|238|239|240|241|242|243|244|245|246|247|248|249|250|251|252|253|254|255|256|257|258|259|260|261|262|263|264|265|266|267|268|269|270|271|272|273|274|275|276|277|278|279|280|281|282|283|284|285|286|287|288|289|290|291|292|293|294|295|296|297|298|299|300|301|302|303|304|305|306|307|308|309|310|311|312|313|314|315|316|317|318|319|320|321|322|323|324|325|326|327|328|329|330|331|332|333|334|335|336|337|338|339|340|341|342|343|344|345|346|347|348|349|350|351|352|353|354|355|356|357|358|359|360|361|362|363|364|365|366|367|368|369|370|371|372|373|374|375|376|377|378|379|380|381|382|383|384|385|386|387|388|389|390|391|392|393|394|395|396|397|398|399|400|401|402|403|404|405|406|407|408|409|410|411|412|413|414|415|416|417|418|419|420|421|422|423|424|425|426|427|428|429|430|431|432|433|434|435|436|437|438|439|440|441|442|443|444|445|446|447|448|449|450|451|452|453|454|455|456|457|458|459|460|461|462|463|464|465|466|467|468|469|470|471|472|473|474|475|476|477|478|479|480|481|482|483|484|485|486|487|488|489|490|491|492|493|494|495|496|497|498|499|500|

Würfelzahl
|6|2|2|3|1|5|3|6|3|4|1|3|1|5|6|4|6|3|6|6|6|4|3|5|5|4|6|6|5|5|2|1|3|4|6|6|6|5|3|6|6|6|6|2|6|5|3|1|5|2|1|4|6|4|5|1|6|4|5|1|5|5|1|4|2|4|3|3|2|2|1|6|4|6|1|6|5|4|3|2|5|2|1|5|5|5|4|3|3|5|2|5|5|6|4|4|6|3|1|6|4|3|6|2|4|4|3|1|1|3|2|5|3|6|5|6|2|6|3|6|3|4|1|6|2|1|6|4|6|4|6|1|6|6|6|6|4|6|6|6|3|3|5|3|5|6|1|4|1|5|6|1|6|1|5|5|4|6|6|4|1|2|2|1|1|4|6|3|3|6|6|5|6|5|3|4|2|2|5|6|5|6|6|6|6|6|6|1|6|6|4|6|4|6|5|5|3|2|4|6|6|6|1|6|6|6|1|4|5|5|2|3|6|5|6|3|2|6|6|6|2|2|1|5|2|6|1|1|6|6|4|2|4|3|4|6|4|1|3|1|4|3|6|5|3|6|1|3|6|6|5|4|3|6|1

|5|4|5|3|5|6|2|4|6|4|2|4|3|5|3|3|5|6|2|6|5|6|1|3|4|2|2|5|4|6|6|1|4|5|3|6|6|2|5|1|6|1|
4|3|5|4|3|6|3|3|6|1|6|2|5|6|6|4|6|6|1|1|6|6|3|5|2|3|3|4|2|4|2|6|1|6|4|1|4|6|6|6|6|6|1
|4|1|2|6|6|6|4|1|6|5|2|4|6|6|6|5|5|4|2|5|4|4|2|1|3|3|5|5|5|1|1|4|2|6|6|2|5|6|4|6|6|4|
5|4|1|3|4|4|3|6|5|6|3|4|6|6|5|2|4|1|3|5|6|5|3|5|3|6|6|6|3|3|2|6|6|6|6|6|2|6|5|3|6|6|6
|3|6|6|2|5|3|6|3|6|6|4|5|6|6|6|6|4|5|6|6|6|1|6|5|5|2|6|4|3|4|4|6|6|4|3|4|4|6|5|3|5|1|
1|1|1|2|2|1|4|1|1|4|6|6|4|6|4|4|2|3|3|1|6|1|3|5|3|4|5|6|6|2|2|6|4|

Zustand
|F|F|F|F|F|F|F|F|F|F|F|F|F|F|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|
U|U|U|U|U|U|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F
|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|U|U|U|U|U|U|U|U|U|U|U|U
|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|
U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|
U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F
|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|
F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|
U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|F|F|F|F|F|F|F|F
|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|
U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|
U|U|U|U|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|F|
F|F|F|F|F|F|F|F|F|F|F|F|

In der Konsole ist die Ausgabe nur 3 Zeilen lang, wofür hier allerdings kein Platz ist. Der Anfang der Ausgabe sieht so aus:



```
Wurf-Nr.    |1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|
Würfelzahl  |6|2|2|3|1|5|3|6|3|4|1|3|1|5|6|4|6|3|6|6|6|4|3|5|5|4|6|6|5|5|2|1|3|4|6|6|6|5|3|6|6|6|6|2|6|5|3|1|5|2
Zustand     |F|F|F|F|F|F|F|F|F|F|F|F|F|F|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|U|F|F|F|F|F
```

Aufgabe 3 (https://www.ensembl.org/index.html)

Aufgabe 3.1

Verfahren: **TBLASTX**

→ Gen: NR1H4 (Höchste Übereinstimmung und höchster Score, niedrigster E-Wert)
Ort: Chromosome 12: 100,473,708-100,564,413

Verfahren: **BLASTN**

→ Gen: NR1H4 (Höchste Übereinstimmung und höchster Score, niedrigster E-Wert)
Ort: Chromosome 12: 100,473,708-100,564,413

weitere Gene:
Gen: FSTL5
Ort: Chromosome 4: 161,383,897-162,164,035

Gen: IFT88
Ort: Chromosome 13: 20,567,069-20,691,437

Gen: ROR1
Ort: Chromosome 1: 63,774,022-64,181,498

Aufgabe 3.2

Verfahren: **TBLASTN**

Gen: IFNL3
Ort: Chromosome 19: 39,243,553-39,245,129

Verfahren: **BLASTP**

Gen: IFNL3
Ort: Chromosome 19:39244065-39244157

Gen: IFNL2
Ort: Chromosome 19:39269488-39269580

Gen: IFNL1
Ort Chromosome 19:39297964-39298050

Welches Gen haben sie entdeckt?
--> ***IFNL3***