① $2891 = 1\,0\,1\,1\,0\,1\,0\,0\,1\,0\,1\,1$

$3927 = 1\,1\,1\,1\,0\,1\,0\,1\,0\,1\,1\,1$

$\underline{\quad 1 \quad\quad\quad 111 \quad} \Rightarrow$ Hamming Distanz $= 4$

② Der Genotyp hat die Länge $n = 46$, also 45 mögliche Crossover spaltungen. Angenommen ein Crossover findet zu 100% statt, ergeben sich folgende Stellen innerhalb der funktional zusammengehörender Gruppe:

1010010101000101101101**01**01010100101011100**10 11**1110111

Das bedeutet, wenn an 32 der 45 möglichen crossover Stellen gespalten wird, ist die Gruppe aufgebrochen.

$\frac{32}{45} = 0,7\overline{111} = 71,\overline{11}\%$

③ PfadCodierung $P = (1\ 2\ 7\ 4\ 6\ 5\ 3)$

Kantencodierung $K = (2\ 7\ 1\ 6\ 3\ 5\ 4)$

Originalcodierung $O = (1\ 1\ 5\ 2\ 3\ 2\ 1)$

Stelle	Liste	Orig.Codierung
1	1→2→3→4→5→6→7	1
2	2→3→4→5→6→7	1
7	3→4→5→6→7	5
4	3→4→5→6	2
6	3→5→6	3
5	3→5	2
3	3	1

④ 1. Absolute Gitterkoordinaten

P_0 (0,3)	P_3 (1,2)	P_6 (2,4)	P_9 (4,3)
P_1 (3,2)	P_4 (1,3)	P_7 (2,5)	P_{10} (4,4)
P_2 (2,3)	P_5 (2,4)	P_8 (3,5)	P_{11} (5,4)

2. Relative Gitterkoordinaten

P_0 (0/3)	P_{3+3} (L)	P_{6+6} (O)	P_{9+3} (R)
P_{0+1} (O)	P_{3+4} (O)	P_{6+3} (O)	P_{9+10} (U)
P_{1+2} (R)	P_{4+5} (R)	P_{3+8} (R)	P_{10+11} (R)

3. Relative Konformationskoordinaten

P_0 (0/3)	P_{3+3} (L)	P_{6+6} (LR)	P_{9+3} (L)
P_{0+1} (O)	P_{3+4} (RL)	P_{6+3} (L)	P_{9+10} (RL)
P_{1+2} (LR)	P_{4+5} (RR)	P_{3+8} (RR)	P_{10+11} (LR)

P_0 (0/3)	P_{3+3} (0/-1)	P_{6+6} (1/1)	P_{9+3} (0/1)
P_{0+1} (3/0)	P_{3+4} (-1/0)	P_{6+3} (1/0)	P_{9+10} (-1/0)
P_{1+2} (0/-1)	P_{4+5} (0/-1)	P_{3+8} (0/1)	P_{10+11} (0/-1)

Epitasie messen:

1. Wenn in absoluten Koordinaten ein Punkt verschoben wird, verändern sich alle anderen ohne weiteres erstmal gar nicht.
   → Es entsteht eine Lücke in der Proteinfaltung und eine Epitasie im Phänotyp ab der Position, an der die Verschiebung stattfindet.

   Das kann alle Positionen $n-2$ bis $n-(n-1)$ betreffen.
   Epitasie = Abhängigkeit (Anzahl) aller Gene nach verschieben.

2. In relativen Gitterkoordinaten kann bei einer Mutation keine Lücke mehr entstehen, da Folgekoordinaten entlang verlaufen und eine Umorientierung erzeugt werden. Jedoch besteht die Möglichkeit, dass ab dem Mutationspunkt durch die Folgekoordinaten mit fester Richtung, die sich nicht ändert, eine Streckung oder Faltung entsteht und somit doch Epitasie entsteht. Keine Umstände "Abknicken" oder anderes Layout.

3. Außer aufeinander überlappen, wie auch bei 1 & 2 möglich, behält die Folgefaltung ab dem Mutationspunkt ihre Struktur und hat somit die geringsten Epitasieschäden, bzw. gar keine Änderung im Phänotypabschnitt nach dem Mutationspunkt.

Geringste Epitasie

Die relativen Konformationskoordinaten ∇

Actuel Lösung aus P
Per Abstandsmatrix

	1	2	3	4	5	6
1	ξ	1	$\sqrt{2}$	$\sqrt{5}$	2	$\sqrt{5}$
2		ξ	1	$\sqrt{2}$	1	$\sqrt{2}$
3			ξ	1	$\sqrt{2}$	$\sqrt{5}$
4				ξ	1	$\sqrt{2}$
5					ξ	1
6						ξ

$d = 7$

	1	2	3	4	5	6
1	ξ	1	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{2}$
2		ξ	1	$\sqrt{2}$	$\sqrt{5}$	$\sqrt{5}$
3			ξ	1	2	$\sqrt{2}$
4				ξ	1	
5					ξ	

$d = 5$

5 Unterschied

Unterschied: 2   1

$\sqrt{2^2 + 1^2} = \sqrt{5}$

Algorithmus
- Liste aller Koordinaten erstellen
- Distanzvergleich zu nächsten Punkten:

$1 - m \rightarrow m. overlap$

als

rel. Gitter
OOOS

rel. Konf.
GGRR

Algorithmus für Überlapp bei relativen Konformationskoordinaten ohne Koordinaten?

1 Funktion checkOverlap (Count (ListeDirection) d)

Algorithmus für Überlappung bei relativen Koordinaten ohne Koordinaten 🗹

```
function checkOverlap (List<Directions> d)      // use Stack to check overlaps for rel. better:
{                                                   if we move North 2 times and South 2 times one pos.
    Array<Directions> a = [];
    int overlaps = 0;
    bool started = false;

    for (int i=0; i<d.Count(); i++)
    {
        if (a.Count == 0)
            a.push(d[i]);
        if (started)                    //we started and found an overlap due to opposing direction
            overlaps++;
        else
        {                 // if we move in opposing direction remove → same place
            if (a[a.Length] == NORTH && d[i] == SOUTH
            || a[a.Length] == EAST && d[i] == WEST
            || a[a.Length] == SOUTH && d[i] == NORTH
            || a[a.Length] == WEST && d[i] == EAST)
                a.remove(a.count());

            else a.push(d[i]);     //other direction : push
        }
        started = true;
    }
    return overlaps;
}
```

Coding ohne overlaps

Koordinatensystem Klasse mit Feldern und bool besetzt.
Viel speicherverbrauch 🗹

Class Tabelle   ☐ besetzt = true.

		7	8	9		
		6		10	11	
4	5					
3	2	1				
		0				

→ oder absolute Koordinaten
→ wo gehört eine Mutation hin?

Aufgaben in der Klausur kommen für haer 3* dran ;-)

Algorithmus
- Liste aller Koordinaten erstellen
- Distanzvergleich zu nächsten Punkten:

```
function checkOverlapByCounts (List<Directions> d)
{
    List<Points> visited = [];
    int overlaps = 0;
    currPoint = (0,0)

    for (int i=0; i<d.Count(); i++)
    {
        if (currPoint in visited)
            overlaps++;
        else
            visited.add(currPoint);

        if d.at(i) == Direction:North
            currPoint = (currPoint[0], currPoint[1]+1);
        if d.at(i) == Direction: East
            currPoint = (currPoint[0]+1, currPoint[1]);
        if d.at(i) == Direction: South
            currPoint = (currPoint[0], currPoint[1]-1);
        if d.at(i) == Direction: West
            currPoint = (currPoint[0]-1, currPoint[1]);
    }
    return overlaps;
}
```

- d = 0 → overlap
- d > j → break, weil für diesen Punkt schaffen wir in j schritten nicht mehr zum Punkt zurück zu kehren 🗹

~ j = i-2 → vorbei, die beiden Vorletzten können vielleicht curl.