This project bases on the Extension project of DETA Socket PLSQL DB.

20210320 Initon Math Yaoguang Luo
20210320 元基数学 罗瑶光

自从有了 AOPM VECS IDUQ TXH DD ，16 个元基成分，我今天定义为 16 进制的数字，对应为

既然是严谨定义， 自然要用生化和语义双元基罗盘来进行推导开始。
我先设未知的为 X
A XXXX
O XXXX
P XXXX
M XXXX
V XXXX
E XXXX
C XXXX
S XXXX
I XXX1
D XXX0
U XXX2
Q XXX3
T XXXX
X XXXX
HE XXXX
HC XXXX
DD 补码

根据第一卷 和 第二卷 283 和 284 页，我能列出来的 新增 关系式 E -> HE, C -> HC.

根据 数字逻辑 和 离散数学 位列比 和 寄存法则 推导 VECS 为 ：
A XXXX
O XXXX
P XXXX
M XXXX

V XXX1
E XXX2
C XXX0
S XXX3

I 0001
D 0000
U 0002
Q 0003

T XXXX
X XXXX

HE　　XXX2
HC　　XXX0

DD　　补码

准备写个欧拉路径算法开始计算 。 第一卷的 数据预测 包 此时派上了用场。

方便大家理解。

```
package org.math.initon.ouler;

import java.util.HashMap;
import java.util.Map;

public class FindOulerRing{
    //这段函数用于观测元基映射的欧拉回路模型
    //思想 罗瑶光
    //算法 欧拉
    //程序员 罗瑶光
    //QUIVT+OSMAX-HEPCD 9 结果输出 有很多，我先任意选一种，就这个了 参考第 48 行。

//下面是所有的输出结果，没时间耗在这上面。
//    HEPCD3
//    2
//    1
//    UIVT+OSMAX-11
//    10
//    9
//    8
//    Q8
//    7
//    6
//    5
//    4
//    3
//    2
//    1
//    0
//
//    AMSO+HEPCD8
//    7
//    6
//    UIVT9
//    8
//    7
//    Q7
//    6
```

```
//    5
//    -X6
//    5
//    4
//    3
//    2
//    1
//    0
//
//、、、、、、、、、、、、、、、、、、、、、、、、 --》》》》》》》      OSMAX-HEPCD9
//    8
//    7
//    UIVT+11
//    10
//    9
//    8
//    Q8
//    7
//    6
//    5
//    4
//    3
//    2
//    1
//    0
//
//    PEHCD3
//    2
//    +OSMAX-8
//    7
//    6
//    5
//    QUIVT9
//    8
//    7
//    6
//    5
//    4
//    3
//    2
//    1
//    0
//
//    MAX-HEPCD7
//    6
//    5
//    UIVT+OSQ12
```

```
//      11
//      10
//      9
//      8
//      7
//      6
//      5
//      4
//      3
//      2
//      1
//      0
//
//      VIUEHCP5
//      D5
//      4
//      +OSMAX-10
//      9
//      8
//      7
//      Q7
//      6
//      5
//      T5
//      4
//      3
//      2
//      1
//      0
//
//      EHCP2
//      D2
//      1
//      +OSMAX-7
//      6
//      5
//      4
//      QUIVT8
//      7
//      6
//      5
//      4
//      3
//      2
//      1
//      0
//
```

```
//    CHEP2
//    UIVT+OSMAX-12
//    11
//    10
//    9
//    Q9
//    8
//    7
//    6
//    5
//    4
//    3
//    2
//    1
//    0
//    D0
//
//    SO+HEPCD6
//    5
//    4
//    UIVT7
//    6
//    5
//    Q5
//    4
//    3
//    -MAX6
//    5
//    4
//    3
//    2
//    1
//    0
//
//    IVTEHCP5
//    D5
//    4
//    +OSMAX-10
//    9
//    8
//    7
//    QU8
//    7
//    6
//    5
//    4
//    3
```

```
//     2
//     1
//     0
//
//     DCHEP3
//     UIVT+OSMAX-13
//     12
//     11
//     10
//     Q10
//     9
//     8
//     7
//     6
//     5
//     4
//     3
//     2
//     1
//     0
//
//     UEHCP3
//     D3
//     2
//     +OSMAX-8
//     7
//     6
//     5
//     Q5
//     4
//     3
//     TVI5
//     4
//     3
//     2
//     1
//     0
//
//     QSO+HEPCD7
//     6
//     5
//     UIVT8
//     7
//     6
//     5
//     4
//     -MAX7
```

```
//     6
//     5
//     4
//     3
//     2
//     1
//     0
//
//     TVIUEHCP6
//     D6
//     5
//     +OSMAX-11
//     10
//     9
//     8
//     Q8
//     7
//     6
//     5
//     4
//     3
//     2
//     1
//     0
//
//     XAMSO+HEPCD9
//     8
//     7
//     UIVT10
//     9
//     8
//     Q8
//     7
//     6
//     -6
//     5
//     4
//     3
//     2
//     1
//     0
//
//     +HEPCD4
//     3
//     2
//     UIVT5
//     4
```

```
//     3
//     QSO5
//     MAX-8
//     7
//     6
//     5
//     4
//     3
//     2
//     1
//     0
//
//     -HEPCD4
//     3
//     2
//     UIVT+OSMAX11
//     10
//     9
//     Q9
//     8
//     7
//     6
//     5
//     4
//     3
//     2
//     1
//     0
//
//     0

    public static void main(String[] args) {
        //init AOPM VECS IDUQ TXH DD
        //初始环路
        Map<String, Boolean> initonsLink= new HashMap<>();
        //环路探索
        Map<String, Boolean> didInitonsLink= new HashMap<>();
        initonsLink.put("DC", true);
        initonsLink.put("CD", true);
        initonsLink.put("IV", true);
        initonsLink.put("VI", true);
        initonsLink.put("IU", true);
        initonsLink.put("UI", true);
        initonsLink.put("UE", true);
        initonsLink.put("EU", true);
        initonsLink.put("UQ", true);
        initonsLink.put("QU", true);
```

```
initonsLink.put("QS", true);
initonsLink.put("SQ", true);

initonsLink.put("VT", true);
initonsLink.put("TV", true);
initonsLink.put("ET", true);
initonsLink.put("TE", true);
initonsLink.put("EH", true);
initonsLink.put("HE", true);
initonsLink.put("EP", true);
initonsLink.put("PE", true);
initonsLink.put("H+", true);//HE + HC -
initonsLink.put("+H", true);
initonsLink.put("H-", true);
initonsLink.put("-H", true);
initonsLink.put("HC", true);
initonsLink.put("CH", true);
initonsLink.put("CP", true);
initonsLink.put("PC", true);
initonsLink.put("SM", true);
initonsLink.put("MS", true);
initonsLink.put("SO", true);
initonsLink.put("OS", true);

initonsLink.put("XA", true);
initonsLink.put("AX", true);
initonsLink.put("MA", true);
initonsLink.put("AM", true);
initonsLink.put("X-", true);
initonsLink.put("-X", true);
initonsLink.put("M-", true);
initonsLink.put("-M", true);
initonsLink.put("T+", true);
initonsLink.put("+T", true);
initonsLink.put("O+", true);
initonsLink.put("+O", true);

String[] initons= new String[]{"H", "A", "O", "P", "M", "V", "E", "C", "S", "I", "D", "U", "Q", "T", "X", "+", "-"};
int[] initonsCount= new int[17];
//for loop
//开始计算 路径总数
//String didInitons= "";
int count= 0;
for(int i= 0; i< initons.length; i++) {
	//System.out.println(temp);
	System.out.print(initons[i]);
	initonsCount[i]++;
```

```
                recur(initons[i], initonsLink, didInitonsLink, initons, initonsCount, count, i);
                //System.out.println(count);
                //下一个
                count= 0;
                System.out.println();
                didInitonsLink.clear();
                initonsCount= new int[17];
            }
            // print loop initons
            //打印可能模式
            System.out.println(count);
        }
        //递归 继续修改。等会加 隔开观测。
        public static void recur(String firstChar, Map<String, Boolean> initonsLink
                , Map<String, Boolean> didInitonsLink, String[] initons, int[] initonsCount, int count, int i) {
            for(int j= 0; j< initons.length; j++) {
                if(!firstChar.equals(initons[j])) {
                    String temp= ""+ firstChar+ initons[j];
                    //有路径
//没有遍历                                                                                    //遍历了两次
                    if(initonsLink.containsKey(temp)&& !didInitonsLink.containsKey(temp)&& initonsCount[j]< 1) {
                        initonsCount[j]++;
                        didInitonsLink.put(""+ firstChar+ initons[j], true);
                        System.out.print(initons[j]);
                        recur(initons[j], initonsLink, didInitonsLink, initons, initonsCount, count+ 1, j);
                        System.out.println(count);
                    }
                }
            }
        }
}
```

刚计算了欧拉元基环路 QUIVT+OSMAX-HEPCD
我定义 17 进制的数据为
QUIVT+OSMAX-HEPCD DD
GFEDCBA9876543210 CARRY
我在思考怎么缩进成 16 进制。
先保证逻辑的严谨性，
我先用 17 进制走一段路程。

17 进制数据已经问世，我今天深入下，进行元基础加法 探索，wechat 已经发布了，在这里整理如下;
元基础数字 = 元基符号 = 生化名称
0 = D = 胞嘧啶
1 = C = 鸟嘌呤
2 = P = 尿胞变鸟苷
3 = E = 尿变嘌呤

4 = H = 黄嘌呤
5 = - =
6 = X = 变感腺鸟苷
7 = A = 变感腺腺苷
8 = M = 鸟腺苷
9 = S = 腺嘌呤
A = O = 尿胞变腺苷
B = + =
C = T = 变感腺尿变苷
D = V = 变感腺嘌呤
E = I = 尿嘧啶
F = U = 变嘧啶
G = Q = 胸腺嘧啶

数字逻辑的推导(C=U+D+D)
语义肽展公式推导
元基数字 =元基符号 = 肽展公式数字变换
0 = D = 0 + 0
1 = C = 0 + F
2 = P = 3 + 1
3 = E = F + 0
4 = H = 3 OR 1
5 = - =
6 = X = D + -
7 = A = D + 9
8 = M = - + 9
9 = S = G + E
A = O = 3 + 9
B = + =
C = T = D + 3
D = V = F + G
E = I =
F = U =
G = Q =

元基数学加法表 根据 4 的归纳完整推导如下
元基数字 = 元基符号= 肽展公式数字变换
0 = D = 0 + 0
1 = C = 0 + F
2 = P = 3 + 1
3 = E = F + 0
4 = H = 3 OR 1
5 = - = 4 + 1
6 = X = D + 5

7 = A = D + 9
8 = M = 5 + 9
9 = S = G + E
A = O = 3 + 9
B = + = 4 + 3
C = T = D + B
D = V = F + G
E = I = E
F = U = E++ OR G--
G = Q = G

20210322 今早把十七进制的元基组合数学变换 定义了，归纳整理如下：
我的思路是 元基稳定化 DEFG 变换
元基数字 = 元基符号= 肽展公式元基变换
0 = D = 00
1 = C = 02
2 = P = 2002
3 = E = 20
4 = H = 20, 02
5 = - = 2002, 0202
6 = X = 23(2002, 0202)
7 = A = 2331
8 = M = (2002, 0202)31
9 = S = 31
A = O = 2031
B = + = 2020, 0220
C = T = 23(2020, 0220)
D = V = 23
E = I = 1
F = U = 2
G = Q = 3
稳定化后于是元基替换为 0123-> DIUQ 如下

0 = D = D + D
1 = C = DU
2 = P = UDDU
3 = E = UD
4 = H = UD, DU
5 = - = (UD, DU)DU
6 = X = UQ(UD, DU)DU
7 = A = UQQI
8 = M = (UD, DU)DUQI
9 = S = QI
A = O = UDQI
B = + = (UD, DU)UD
C = T = UQ(UD, DU)UD
D = V = UQ

E = I = I
F = U = U
G = Q = Q

我在思考 这个括号内的元基如果进行之后计算的唯一化。
到现在 十进制常数进行元基码 变换的思路已经问世了，下一步，养疗经真实应用。
这里的 568B 我推测又是一组概率钥匙酸碱控制。我也会真实应用测试论证。

今天多做一点推导：我把 0 到 G 的欧拉顺序 改成 线性数学顺序观测如下：
0 = D = D + D
E = I = I
F = U = U
G = Q = Q
1 = C = DU
D = V = UQ
3 = E = UD
9 = S = QI
7 = A = UQQI
2 = P = UDDU
A = O = UDQI
4 = H = UD, DU
5 = - = (UD, DU)DU
B = + = (UD, DU)UD
6 = X = UQ(UD, DU)DU
C = T = UQ(UD, DU)UD
8 = M = (UD, DU)DUQI

我想这个顺序别有用途，先搁置。
下一步 H 化简 HE+, HC- , 然后重新线性排列如下
0 = D = D
E = I = I
F = U = U
G = Q = Q
1 = C = DU
3 = E = UD
4 = H = UD, DU
D = V = UQ
9 = S = QI
5 = - = DUDU
2 = P = UDDU
B = + = UDUD
A = O = UDQI
7 = A = UQQI
8 = M = DUDUQI
6 = X = UQDUDU
C = T = UQUDUD

修正后如下

0 = D = D
E = I = I
F = U = U
G = Q = Q
1 = C = DI
3 = E = UD
4 = H = UD, DU
D = V = UQ
9 = S = QI
5 = - = DUDU
2 = P = UDDU
B = + = UDUD
A = O = UDQI
7 = A = UQQI
8 = M = DUDUQI
6 = X = UQDUDU
C = T = UQUDUD

修正下 C=DU 改成 DI, 因为肽展公式(补码计算) C= DDU, DD 是补码

肽展公式的推导(肽展计算)(C=I+D)

开始语义肽展公式验证### 元基数学加法表 根据 4 的归纳完整推导如下

元基数字 = 元基符号= 肽展公式元基变换

0 = D = D + D
1 = C = I + D
2 = P = E + C
3 = E = I + U, D + U
4 = H = E OR C
5 = - = H + C
6 = X = V + HC
7 = A = V + S
8 = M = HC + S

9 = S = Q + I
A = O = E + S
B = + = H + E

C = T = V + HE
D = V = U + Q
E = I = I
F = U = I++ OR Q--
G = Q = Q

###于是元基数字归纳

元基数字 = 元基符号= 肽展公式元基数字变换

0 = D = 0 + 0

1 = C = 1 + 0
2 = P = (12, 02) + 10
3 = E = 1 + 2, 0 + 2
4 = H = (12, 02) OR 10
5 = HC = (12, 02) OR 10 + 10
6 = X = 23 + (12, 02) OR 10 + 10
7 = A = 23 + 31
8 = M = (12, 02) OR 10 + 10 + 31

9 = S = 3 + 1
A = O = (12, 02) + 31
B = HE = (12, 02) OR 10 + 12, 02

C = T = 23 + (12, 02) OR 10 + (12, 02)
D = V = 2 + 3
E = I = 1
F = U = 1++ OR 3--
G = Q = 3

###于是元基肽展归纳如下
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = D + D
1 = C = I + D
2 = P = (IU, DU) + ID
3 = E = I + U, D + U
4 = H = (IU, DU) OR ID
5 = HC = (IU, DU) OR ID + ID
6 = X = UQ + (IU, DU) OR ID + ID
7 = A = UQ + QI
8 = M = (IU, DU) OR ID + ID + QI

9 = S = Q + I
A = O = (IU, DU) + QI
B = HE = (IU, DU) OR ID + (IU, DU)

C = T = UQ + (IU, DU) OR ID + (IU, DU)
D = V = U + Q
E = I = I
F = U = I++ OR Q--
G = Q = Q

###开始整理
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = DD
1 = C = ID
2 = P = IUID, DUID
3 = E = IU, DU

4 = H = (IU, DU) OR ID
5 = HC = (IU, DU) OR ID + ID
6 = X = UQ + (IU, DU) OR ID + ID
7 = A = UQQI
8 = M = (IU, DU) OR ID + ID + QI

9 = S = QI
A = O = (IU, DU) + QI
B = HE = (IU, DU) OR ID + (IU, DU)

C = T = UQ + (IU, DU) OR ID + (IU, DU)
D = V = UQ
E = I = I
F = U = I++ OR Q--
G = Q = Q

我得到一个结论，肽展公式的推导(C=I+D)比数字逻辑的推导(C=U+D+D)更准确。

###开始线性整理
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = DD
E = I = I
F = U = I++ OR Q--
G = Q = Q

1 = C = ID
3 = E = IU, DU
4 = H = (IU, DU) OR ID
D = V = UQ
9 = S = QI

2 = P = (IU, DU) + ID
A = O = (IU, DU) + QI
5 = HC = ((IU, DU) OR ID) + ID
B = HE = ((IU, DU) OR ID) + (IU, DU)
8 = M = ((IU, DU) OR ID) + ID + QI

7 = A = UQQI
6 = X = UQ + ((IU, DU) OR ID) + ID
C = T = UQ + ((IU, DU) OR ID) + (IU, DU)

###线性整理优化
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = DD

E = I = I
F = U = I++ OR Q--
G = Q = Q

1 = C = ID
3 = E = IU, DU
4 = H = (IU, DU) OR ID
5 = HC = ((IU, DU) OR ID) + ID
B = HE = ((IU, DU) OR ID) + (IU, DU)
D = V = UQ
9 = S = QI

2 = P = (IU, DU) + ID
A = O = (IU, DU) + QI
7 = A = UQQI

8 = M = ((IU, DU) OR ID) + ID + QI
6 = X = UQ + ((IU, DU) OR ID) + ID
C = T = UQ + ((IU, DU) OR ID) + (IU, DU)

我推导出语义元基的次序为
A O P M - T X H DD - V E C S - I D U Q
现在的元基数字逻辑次序为
M X T - P O A - C E H HC HE V S - D I U Q

###酸碱肽展开归纳如下
元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合)
0 = D = DD =(D, DD)
E = I = I =(I)
F = U = I++ OR Q-- =(I, Q)
G = Q = Q =(Q)

1 = C = ID =(ID)
3 = E = IU, DU =(IU, DU)
4 = H = (IU, DU) OR ID =(IU, DU, ID)
D = V = UQ =(UQ)
9 = S = QI =(QI)

2 = P = (IU, DU) + ID =(IUID, DUID)
5 = HC = ((IU, DU) OR ID) + ID =(IUID, DUID, IDID)
B = HE = ((IU, DU) OR ID) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, IDIU, IDDU)
A = O = (IU, DU) + QI =(IUQI, DUQI)

7 = A = UQQI =(UQQI)

8 = M = ((IU, DU) OR ID) + ID + QI =(IUIDQI, DUIDQI, IDIDQI)
6 = X = UQ + ((IU, DU) OR ID) + ID =(UQIUID, UQDUID, UQIDID)
C = T = UQ + ((IU, DU) OR ID) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQIDIU, UQIDDU)

归纳后的元基数字逻辑次序为
M X T - P HC HE O A - C E H V S - D I U Q

归纳后的元基数字活性次序为
T X M - HE HC O P A - H E C V S - U D I Q

准备应用于养疗经 DNA 视觉进行简单验证下，优化后用于 DNA 数据库的数字层计算。

在这次序表中 D 在 I 的前面，于是我准备修正 C=ID 为 DI，于是如下：

修正 C 后的最新肽展计算公式观测
元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合)
0 = D = DD =(D, DD)
E = I = I =(I)
F = U = I++ OR Q-- =(I, Q)
G = Q = Q =(Q)

1 = C = DI =(DI)
3 = E = IU, DU =(IU, DU)
4 = H = (IU, DU) OR DI =(IU, DU, DI)
D = V = UQ =(UQ)
9 = S = QI =(QI)

2 = P = (IU, DU) + DI =(IUDI, DUDI)
5 = HC = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI)
B = HE = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU)
A = O = (IU, DU) + QI =(IUQI, DUQI)
7 = A = UQQI =(UQQI)

8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI)
6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI)
C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU)

继续跟进了下在离散数学中 H = (IU, DU) OR DI= (IU, DU) + DI= IUDI, DUDI, 上面的肽展公式在 离散数学中可以继续展开如下

元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合)
0 = D = DD =(D, DD)
E = I = I =(I)
F = U = I++ OR Q-- =(I, Q)
G = Q = Q =(Q)

1 = C = DI =(DI)
3 = E = IU, DU =(IU, DU)
4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)
D = V = UQ =(UQ)
9 = S = QI =(QI)

2 = P = (IU, DU) + DI =(IUDI, DUDI)
5 = HC = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)
B = HE = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)
A = O = (IU, DU) + QI =(IUQI, DUQI)
7 = A = UQQI =(UQQI)

8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI) OR (IUDIDIQI, DUDIDIQI)
6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI) OR (UQIUDIDI, UQDUDIDI)
C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

似乎开始完美。于是活性顺序又打乱了，再整理下如下：

元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合)
1 位
E = I = I =(I)
F = U = I++ OR Q-- =(I, Q)
G = Q = Q =(Q)
1~2 位
0 = D = DD =(D, DD)
2 位
1 = C = DI =(DI)
3 = E = IU, DU =(IU, DU)
D = V = UQ =(UQ)
9 = S = QI =(QI)
2~4 位
4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)
4 位
2 = P = (IU, DU) + DI =(IUDI, DUDI)

A = O = (IU, DU) + QI =(IUQI, DUQI)
7 = A = UQQI =(UQQI)
4~6 位
5 = HC- = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)
B = HE+ = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)
6~8 位
8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI) OR (IUDIDIQI, DUDIDIQI)
6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI) OR (UQIUDIDI, UQDUDIDI)
C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

C 还是 = DI 次序， 所以上一步的公式不用变动。
整理后：
元基活性次序为 <IUQ D CEVS H POA -+ MXT>

我得到一个结论：IDQ 是稳定元基， UH 是活性元基.（2021024 结论更新， IQ 是稳定元基， DUH 是活性元基）

归纳出核心
黄嘌呤：用于肽展换元 计算
变嘧啶：用于 血氧峰 计算
胞嘧啶：用于 补码 计算

```
package org.math.initon.pde;

import java.util.HashMap;
import java.util.Map;

import org.math.initon.pds.PDE_PDS_DL;

//这个函数集用于将常数变换成十七进制元基数字，
//这个函数集用于将十七进制元基数字进行元基变换
//这个函数集用于将元基变换进行肽展概率丝化展开
//这个函数用于将肽展丝化的肽增十七进制进行元基变换
//这个函数用于将肽展丝化的肽增十七进制变换成元基数字
//这个函数用于十七进制元基数字进行十进制还原。
public class DecadeToPDS{
    //思想：肽展公式 1.2.2，元基数字逻辑； 十七进制元基组合数学；概率论
    //算法：进制计算，肽展计算
    //程序员： 罗瑶光，
    public Map<String, String> initonsMap= new HashMap<>();
    public Map<String, String> initonsCode= new HashMap<>();
    public Map<String, String> initonsSet= new HashMap<>();
    public Map<String, Integer> numberSet= new HashMap<>();
    public static void main(String[] Args) {
        DecadeToPDS decadeToPDS= new DecadeToPDS();
```

```
//元基符号变元基数字
decadeToPDS.initonsMap.put("A", "7");
decadeToPDS.initonsMap.put("O", "A");
decadeToPDS.initonsMap.put("P", "2");
decadeToPDS.initonsMap.put("M", "8");
decadeToPDS.initonsMap.put("V", "D");
decadeToPDS.initonsMap.put("E", "3");
decadeToPDS.initonsMap.put("C", "1");
decadeToPDS.initonsMap.put("S", "9");
decadeToPDS.initonsMap.put("I", "E");
decadeToPDS.initonsMap.put("D", "0");
decadeToPDS.initonsMap.put("U", "F");
decadeToPDS.initonsMap.put("Q", "G");
decadeToPDS.initonsMap.put("T", "C");
decadeToPDS.initonsMap.put("X", "6");
decadeToPDS.initonsMap.put("+", "B");
decadeToPDS.initonsMap.put("-", "5");
decadeToPDS.initonsMap.put("H", "4");
//元基数字变元基符号
decadeToPDS.initonsCode.put("0", "D");
decadeToPDS.initonsCode.put("1", "C");
decadeToPDS.initonsCode.put("2", "P");
decadeToPDS.initonsCode.put("3", "E");
decadeToPDS.initonsCode.put("4", "H");
decadeToPDS.initonsCode.put("5", "-");
decadeToPDS.initonsCode.put("6", "X");
decadeToPDS.initonsCode.put("7", "A");
decadeToPDS.initonsCode.put("8", "M");
decadeToPDS.initonsCode.put("9", "S");
decadeToPDS.initonsCode.put("A", "O");
decadeToPDS.initonsCode.put("B", "+");
decadeToPDS.initonsCode.put("C", "T");
decadeToPDS.initonsCode.put("D", "V");
decadeToPDS.initonsCode.put("E", "I");
decadeToPDS.initonsCode.put("F", "U");
decadeToPDS.initonsCode.put("G", "Q");
//阿拉伯数字变元基数字
decadeToPDS.initonsSet.put("0", "0");
decadeToPDS.initonsSet.put("1", "1");
decadeToPDS.initonsSet.put("2", "2");
decadeToPDS.initonsSet.put("3", "3");
decadeToPDS.initonsSet.put("4", "4");
decadeToPDS.initonsSet.put("5", "5");
decadeToPDS.initonsSet.put("6", "6");
decadeToPDS.initonsSet.put("7", "7");
decadeToPDS.initonsSet.put("8", "8");
decadeToPDS.initonsSet.put("9", "9");
```

```
        decadeToPDS.initonsSet.put("10", "A");
        decadeToPDS.initonsSet.put("11", "B");
        decadeToPDS.initonsSet.put("12", "C");
        decadeToPDS.initonsSet.put("13", "D");
        decadeToPDS.initonsSet.put("14", "E");
        decadeToPDS.initonsSet.put("15", "F");
        decadeToPDS.initonsSet.put("16", "G");

        //元基数字变阿拉伯数字
        decadeToPDS.numberSet.put("0", 0);
        decadeToPDS.numberSet.put("1", 1);
        decadeToPDS.numberSet.put("2", 2);
        decadeToPDS.numberSet.put("3", 3);
        decadeToPDS.numberSet.put("4", 4);
        decadeToPDS.numberSet.put("5", 5);
        decadeToPDS.numberSet.put("6", 6);
        decadeToPDS.numberSet.put("7", 7);
        decadeToPDS.numberSet.put("8", 8);
        decadeToPDS.numberSet.put("9", 9);
        decadeToPDS.numberSet.put("A", 10);
        decadeToPDS.numberSet.put("B", 11);
        decadeToPDS.numberSet.put("C", 12);
        decadeToPDS.numberSet.put("D", 13);
        decadeToPDS.numberSet.put("E", 14);
        decadeToPDS.numberSet.put("F", 15);
        decadeToPDS.numberSet.put("G", 16);

        String decade= ""+ (int)(Math.random()*1000 % 256);//随便写一个数
        System.out.println("输入十进制数："+ decade);
        String seventeen= decadeToPDS.decadeToSeventeen(decade, decadeToPDS);
        System.out.println("元基进制数为："+ seventeen);

        String initons= decadeToPDS.seventeenToIntons(seventeen, decadeToPDS);
        System.out.println("变换为元基："+initons);

        double pDE_KEY_rate= 0.25;//随便模拟一个 0-1 之间的概率钥匙，假设 0~0.5 为酸，0.5~1 为碱；
        //initons= "AOPMVE";
        System.out.println("输入元基："+ initons);
        System.out.println("输入概率："+ pDE_KEY_rate);
        String pDS= decadeToPDS.initonsToPDS(initons, pDE_KEY_rate, decadeToPDS);
        System.out.println("输出肽丝:"+ pDS);
        pDS= pDS.replace(".", "");
        String pDSInitons= decadeToPDS.PDSToInitons(pDS, pDE_KEY_rate, decadeToPDS);
        System.out.println("肽丝增元:"+ pDSInitons);

        //第二卷的肽展公式 可以用到了
//      String pDEInitons= decadeToPDS.PDSToPDE(pDSInitons, pDE_KEY_rate, decadeToPDS);
```

```
//          System.out.println("肽展增元:"+ pDEInitons);


            String pDSSeventeen= decadeToPDS.initonsToSeventeen(pDSInitons, decadeToPDS);
            System.out.println("元基数字:"+ pDSSeventeen);

            String pDSDecade= decadeToPDS.seventeenToDecade(pDSSeventeen, decadeToPDS);
            System.out.println("输出十进制数:"+ pDSDecade);
        }

//      //准备集成第二卷的 AOPM 级别 肽展公式 ，已经并入 PDSToInitons 函数中
//      private String PDSToPDE(String pds, double pDE_KEY_rate, DecadeToPDS decadeToPDS) {
//
////          pds= pds.replace("UQ", "V");
////          pds= pds.replace("DI", "C");
////          pds= pds.replace("IQ", "S");
////          pds= pds.replace("VS", "A");
////          pds= pds.replace("ES", "O");
////          pds= pds.replace("EC", "P");
////          pds= pds.replace("CS", "M");
////          pds= pds.replace("VE", "T");
////          pds= pds.replace("VC", "X");
////
//          return pds;
//      }
        //这个函数集用于将常数变换成十七进制元基数字，
        public String decadeToSeventeen(String decade, DecadeToPDS decadeToPDS) {
            String seventeen= "";
            int decad= Integer.valueOf(decade.toString());
            while(0< decad/ 17) {
                int seventeenth= decad% 17;
                seventeen= decadeToPDS.initonsSet.get(""+ seventeenth)+ seventeen;
                decad= decad/ 17;
            }
            seventeen= decadeToPDS.initonsSet.get(""+ decad)+ seventeen;
            //
            return seventeen;
        }
        //这个函数集用于将十七进制元基数字进行元基变换
        public String seventeenToIntons(String seventeen, DecadeToPDS decadeToPDS) {
            String initons= "";
            for(int i= 0; i< seventeen.length(); i++) {
                initons+= decadeToPDS.initonsCode.get(""+ seventeen.charAt(i));
            }
            //
            return initons;
        }
```

```
    //这个函数集用于将元基变换进行肽展概率丝化展开
    public String initonsToPDS(String initons, double pDE_KEY_rate, DecadeToPDS decadeToPDS) {
        String PDS= "";
        StringBuilder PDEKey= new StringBuilder("");
        for(int i= 0; i< initons.length(); i++) {
            PDS+=    new PDE_PDS_DL().initonPDSwithBYS(""+ initons.charAt(i), pDE_KEY_rate, PDEKey, true)+ ".";
        }
        System.out.println("生成钥匙："+ PDEKey);
        //
        return PDS;
    }
    //这个函数用于将肽展丝化的肽增十七进制进行元基变换
    public String PDSToInitons(String pDS, double pDE_KEY_rate, DecadeToPDS decadeToPDS) {
        String initons= "";
        //initons= new PDE_PDS_DL().initonPDIwithBYS(pDS, 0, new StringBuilder(), false);
        //initons= new PDE_PDS_DL().initonPDEwithBYS(pDS, pDE_KEY_rate, new StringBuilder(), true);
        initons= new PDE_PDS_DL().initonPDE_DCDLwithBYS(pDS, pDE_KEY_rate, new StringBuilder(), true);
        return initons;
    }
    //这个函数用于将肽展丝化的肽增十七进制变换成元基数字
    public String initonsToSeventeen(String initons, DecadeToPDS decadeToPDS) {
        String seventeen= "";
        //
        for(int i= 0; i< initons.length(); i++) {
            seventeen+= decadeToPDS.initonsMap.get(""+ initons.charAt(i));
        }
        return seventeen;
    }
    //这个函数用于十七进制元基数字进行十进制还原。
    public String seventeenToDecade(String seventeen, DecadeToPDS decadeToPDS) {
        int decade= 0;
        //A11                10*17*17 + 1*17 + 1
        for(int i= 0; i< seventeen.length(); i++) {
            int value= decadeToPDS.numberSet.get(""+ seventeen.charAt(i)).intValue();
            decade+= value* Math.pow(17,    seventeen.length()- 1- i);
        }
        return ""+ decade;
    }
}
```

```
package org.math.initon.pds;
//这个函数用于元基进行数字逻辑丝化变换
//思想：肽展公式，十七进制元基数字，元基数字逻辑
```

```
//作者：罗瑶光
//算法参考如下（肽展公式在离散数学中根据贝叶斯进行数字逻辑变换）
//#### 元基数字  =   元基符号=   肽展公式元基数字变换 = (肽概率展开数字逻辑集合)
//#### 0  =     D  =   DD =(D, DD)
//#### E  =     I  =   I  =(I)
//#### F  =     U  =   I++ OR Q--  =(I, Q)
//#### G  =     Q  =    Q  =(Q)
//
//#### 1  =     C  =   DI  =(DI)
//#### 3  =     E  =   IU, DU  =(IU, DU)
//#### 4  =     H  =   (IU, DU) OR DI  =(IU, DU, DI) OR (IUDI, DUDI)
//#### D  =     V  =   UQ  =(UQ)
//#### 9  =     S  =   QI  =(QI)
//
//
//#### 2  =     P  =   (IU, DU) + DI  =(IUDI, DUDI)
//#### 5  =     HC =    ((IU, DU) OR DI) + DI  =(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)
//#### B  =     HE =    ((IU, DU) OR DI) + (IU, DU)  =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU)
OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)
//#### A  =     O  =   (IU, DU) + QI  =(IUQI, DUQI)
//#### 7  =     A  =   UQQI  =(UQQI)
//
//
//#### 8  =     M  =   ((IU, DU) OR DI) + DI + QI  =(IUDIQI, DUDIQI, DIDIQI) OR (IUDIDIQI,
DUDIDIQI)
//#### 6  =     X  =   UQ + ((IU, DU) OR DI) + DI  =(UQIUDI, UQDUDI, UQDIDI) OR (UQIUDIDI,
UQDUDIDI)
//#### C  =     T  =   UQ + ((IU, DU) OR DI) + (IU, DU)  =(UQIUIU, UQIUDU, UQDUIU, UQDUDU,
UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

public class PDE_PDS_DL {
   public String initonPDSwithBYS(String initon, double bys, StringBuilder pDEKey,
boolean isBys) {
      if(initon.equalsIgnoreCase("D")) {
         return "D";
      }
      if(initon.equalsIgnoreCase("I")) {
         return "I";
      }
      if(initon.equalsIgnoreCase("U")) {
         if(!isBys) {
            if(Math.random()< 0.5) {
               pDEKey.append("0");
               return "I";
            }else {
               pDEKey.append("1");
               return "Q";
```

```
        }
    }else {
        if(Math.random()< bys) {
            pDEKey.append("0");
            return "I";
        }else {
            pDEKey.append("1");
            return "Q";
        }
    }
}
if(initon.equalsIgnoreCase("Q")) {
    return "Q";
}

if(initon.equalsIgnoreCase("C")) {
    return "DI";
}
if(initon.equalsIgnoreCase("E")) {
    if(!isBys) {
        if(Math.random()< 0.5) {
            pDEKey.append("0");
            return "IU";
        }else {
            pDEKey.append("1");
            return "DU";
        }
    }else {
        if(Math.random()< bys) {
            pDEKey.append("0");
            return "IU";
        }else {
            pDEKey.append("1");
            return "DU";
        }
    }
}
//#### 4  =      H  =   (IU, DU) OR DI  =(IU, DU, DI) OR (IUDI, DUDI)
if(initon.equalsIgnoreCase("H")) {
    if(!isBys) {
        if(Math.random()< 0.5) {
            pDEKey.append("0");
            return "IUDI";
        }else {
            pDEKey.append("1");
            return "DUDI";
        }
```

```
    }else {
        if(Math.random()< bys) {
            pDEKey.append("0");
            return "IUDI";
        }else {
            pDEKey.append("1");
            return "DUDI";
        }
    }
}

//+- 符号见 FindOulerRing 函数 的 332 行。
//#### 5  =      HC =    ((IU, DU) OR DI) + DI
//=(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)
if(initon.equalsIgnoreCase("-")) {
    if(!isBys) {
        if(Math.random()< 0.5) {
            pDEKey.append("0");
            return "IUDIDI";
        }else {
            pDEKey.append("1");
            return "DUDIDI";
        }
    }else {
        if(Math.random()< bys) {
            pDEKey.append("0");
            return "IUDIDI";
        }else {
            pDEKey.append("1");
            return "DUDIDI";
        }
    }
}
//+- 符号见 FindOulerRing 函数 的 332 行。
//#### B  =      HE =    ((IU, DU) OR DI) + (IU, DU)
//=(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)
if(initon.equalsIgnoreCase("+")) {
    if(!isBys) {
        if(Math.random()< 0.5) {
            pDEKey.append("0");
            if(Math.random()< 0.5) {
                pDEKey.append("0");
                return "IUDIIU";
            }else {
                pDEKey.append("1");
                return "IUDIDU";
            }
```

```
        }else {
            pDEKey.append("1");
            if(Math.random()< 0.5) {
                pDEKey.append("0");
                return "DUDIIU";
            }else {
                pDEKey.append("1");
                return "DUDIDU";
            }
        }
    }else {
        if(Math.random()< bys) {
            pDEKey.append("0");
            if(Math.random()< bys) {
                pDEKey.append("0");
                return "IUDIIU";
            }else {
                pDEKey.append("1");
                return "IUDIDU";
            }
        }else {
            pDEKey.append("1");
            if(Math.random()< bys) {
                pDEKey.append("0");
                return "DUDIIU";
            }else {
                pDEKey.append("1");
                return "DUDIDU";
            }
        }
    }
}
if(initon.equalsIgnoreCase("V")) {
    return "UQ";
}
if(initon.equalsIgnoreCase("S")) {
    return "QI";
}
//#### 2  =      P  =   (IU, DU) + DI  =(IUDI, DUDI)
if(initon.equalsIgnoreCase("P")) {
    if(!isBys) {
        if(Math.random()< 0.5) {
            pDEKey.append("0");
            return "IUDI";
        }else {
            pDEKey.append("1");
            return "DUDI";
```

```
        }
    }else {
        if(Math.random()< bys) {
            pDEKey.append("0");
            return "IUDI";
        }else {
            pDEKey.append("1");
            return "DUDI";
        }
    }
}
//#### A  =     O  =   (IU, DU) + QI  =(IUQI, DUQI)
if(initon.equalsIgnoreCase("O")) {
    if(!isBys) {
        if(Math.random()< 0.5) {
            pDEKey.append("0");
            return "IUQI";
        }else {
            pDEKey.append("1");
            return "DUQI";
        }
    }else {
        if(Math.random()< bys) {
            pDEKey.append("0");
            return "IUQI";
        }else {
            pDEKey.append("1");
            return "DUQI";
        }
    }
}
if(initon.equalsIgnoreCase("A")) {
    return "UQQI";
}
//#### 8  =     M  =   ((IU, DU) OR DI) + DI + QI
//=(IUDIQI, DUDIQI, DIDIQI) OR (IUDIDIQI, DUDIDIQI)
if(initon.equalsIgnoreCase("M")) {
    if(!isBys) {
        if(Math.random()< 0.5) {
            pDEKey.append("0");
            return "IUDIDIQI";
        }else {
            pDEKey.append("1");
            return "DUDIDIQI";
        }
    }else {
        if(Math.random()< bys) {
```

```
                pDEKey.append("0");
                return "IUDIDIQI";
            }else {
                pDEKey.append("1");
                return "DUDIDIQI";
            }
        }

    }
    //#### 6  =      X  =   UQ + ((IU, DU) OR DI) + DI
    // =(UQIUDI, UQDUDI, UQDIDI) OR (UQIUDIDI, UQDUDIDI)
    if(initon.equalsIgnoreCase("X")) {
        if(!isBys) {
            if(Math.random()< 0.5) {
                pDEKey.append("0");
                return "UQIUDIDI";
            }else {
                pDEKey.append("1");
                return "UQDUDIDI";
            }
        }else {
            if(Math.random()< bys) {
                pDEKey.append("0");
                return "UQIUDIDI";
            }else {
                pDEKey.append("1");
                return "UQDUDIDI";
            }
        }

    }
    //#### C  =      T  =   UQ + ((IU, DU) OR DI) + (IU, DU)
    //=(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU,
UQDUDIIU, UQDUDIDU)
    if(initon.equalsIgnoreCase("T")) {
        if(!isBys) {
            if(Math.random()< 0.5) {
                pDEKey.append("0");
                if(Math.random()< 0.5) {
                    pDEKey.append("0");
                    return "UQIUDIIU";
                }else {
                    pDEKey.append("1");
                    return "UQIUDIDU";
                }
            }else {
                pDEKey.append("1");
```

```
                if(Math.random()< 0.5) {
                    pDEKey.append("0");
                    return "UQDUDIIU";
                }else {
                    pDEKey.append("1");
                    return "UQDUDIDU";
                }
            }
        }else {
            if(Math.random()< bys) {
                pDEKey.append("0");
                if(Math.random()< bys) {
                    pDEKey.append("0");
                    return "UQIUDIIU";
                }else {
                    pDEKey.append("1");
                    return "UQIUDIDU";
                }
            }else {
                pDEKey.append("1");
                if(Math.random()< bys) {
                    pDEKey.append("0");
                    return "UQDUDIIU";
                }else {
                    pDEKey.append("1");
                    return "UQDUDIDU";
                }
            }
        }
    }
    return null;
}
//IUQ D CEVS H POA -+ MXT
//反向排列后如下
//TXM +- AOP H SVEC D QUI
public String initonPDIwithBYS(String pds, double bys, StringBuilder pDEKey, boolean
isBys) {
    pds= pds.replace("UQIUDIIU", "T");
    pds= pds.replace("UQIUDIDU", "T");
    pds= pds.replace("UQDUDIIU", "T");
    pds= pds.replace("UQDUDIDU", "T");
    pds= pds.replace("UQIUDIDI", "X");
    pds= pds.replace("UQDUDIDI", "X");
    pds= pds.replace("IUDIDIQI", "M");
    pds= pds.replace("DUDIDIQI", "M");
    pds= pds.replace("IUDIIU", "+");
    pds= pds.replace("IUDIDU", "+");
```

```
        pds= pds.replace("DUDIIU", "+");
        pds= pds.replace("DUDIDU", "+");
        pds= pds.replace("IUDIDI", "-");
        pds= pds.replace("DUDIDI", "-");
        pds= pds.replace("UQQI", "A");
        pds= pds.replace("IUQI", "O");
        pds= pds.replace("DUQI", "O");
        pds= pds.replace("IUDI", "P");
        pds= pds.replace("DUDI", "P");
        pds= pds.replace("IUDI", "H");
        pds= pds.replace("DUDI", "H");
        pds= pds.replace("QI", "S");
        pds= pds.replace("UQ", "V");
        pds= pds.replace("IU", "E");
        pds= pds.replace("DU", "E");
        pds= pds.replace("DI", "C");
        pds= pds.replace("D", "D");
        pds= pds.replace("Q", "Q");
        pds= pds.replace("U", "U");
        pds= pds.replace("I", "I");
        return pds;
    }

    //用于肽展公式逐级变换
    //QUI  D  SVEC  H AOP +-  TXM
    public String initonPDEwithBYS(String pds, double pDE_KEY_rate, StringBuilder pDEKey,
boolean isBys) {
        if(!isBys) {
            pds= pds.replace("Q", "Q");
            pds= pds.replace("U", "D");
            pds= pds.replace("I", "I");
            pds= pds.replace("D", "D");
            pds= pds.replace("QI", "S");
            pds= pds.replace("UQ", "V");
            pds= pds.replace("IU", "E");
            pds= pds.replace("DU", "E");
            pds= pds.replace("DI", "C");
            pds= pds.replace("IUDI", "H");
            pds= pds.replace("DUDI", "H");
            pds= pds.replace("UQQI", "A");
            pds= pds.replace("IUQI", "O");
            pds= pds.replace("DUQI", "O");
            pds= pds.replace("IUDI", "P");
            pds= pds.replace("DUDI", "P");
            pds= pds.replace("IUDIIU", "+");
            pds= pds.replace("IUDIDU", "+");
            pds= pds.replace("DUDIIU", "+");
```

```
    pds= pds.replace("DUDIDU", "+");
    pds= pds.replace("IUDIDI", "-");
    pds= pds.replace("DUDIDI", "-");
    pds= pds.replace("UQIUDIIU", "T");
    pds= pds.replace("UQIUDIDU", "T");
    pds= pds.replace("UQDUDIIU", "T");
    pds= pds.replace("UQDUDIDU", "T");
    pds= pds.replace("UQIUDIDI", "X");
    pds= pds.replace("UQDUDIDI", "X");
    pds= pds.replace("IUDIDIQI", "M");
    pds= pds.replace("DUDIDIQI", "M");
    return pds;
}
pds= pds.replace("Q", "Q");
pds= pds.replace("U", "U");
pds= pds.replace("I", "I");
pds= pds.replace("D", "D");
pds= pds.replace("QI", "S");
pds= pds.replace("UQ", "V");

if(Math.random()<pDE_KEY_rate) {
    pds= pds.replace("IU", "E");
}else {
    pds= pds.replace("DU", "E");
}

pds= pds.replace("DI", "C");

if(Math.random()< pDE_KEY_rate) {
    pds= pds.replace("IUDI", "H");
}else {
    pds= pds.replace("DUDI", "H");
}

pds= pds.replace("UQQI", "A");

if(Math.random()<pDE_KEY_rate) {
    pds= pds.replace("IUQI", "O");
}else {
    pds= pds.replace("DUQI", "O");
}

if(Math.random()< pDE_KEY_rate) {
    pds= pds.replace("IUDI", "P");
}else {
    pds= pds.replace("DUDI", "P");
}
```

```
if(Math.random()< pDE_KEY_rate) {
    if(Math.random()<pDE_KEY_rate) {
        pds= pds.replace("IUDIIU", "+");
    }else {
        pds= pds.replace("IUDIDU", "+");
    }
}else {
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("DUDIIU", "+");
    }else {
        pds= pds.replace("DUDIDU", "+");
    }
}

if(Math.random()< pDE_KEY_rate) {
    pds= pds.replace("IUDIDI", "-");
}else {
    pds= pds.replace("DUDIDI", "-");
}


if(Math.random()< pDE_KEY_rate) {
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("UQIUDIIU", "T");
    }else {
        pds= pds.replace("UQIUDIDU", "T");
    }
}else {
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("UQDUDIIU", "T");
    }else {
        pds= pds.replace("UQDUDIDU", "T");
    }
}

if(Math.random()< pDE_KEY_rate) {
    pds= pds.replace("UQIUDIDI", "X");
}else {
    pds= pds.replace("UQDUDIDI", "X");
}

if(Math.random()<pDE_KEY_rate) {
    pds= pds.replace("IUDIDIQI", "M");
}else {
    pds= pds.replace("DUDIDIQI", "M");
}
```

```
        return pds;
    }

    //融合肽展公式，离散数学和数字逻辑 的元基变换
    // pds= pds.replace("UQ", "V");
    // pds= pds.replace("DI", "C");
    // pds= pds.replace("IQ", "S");
    // pds= pds.replace("VS", "A");
    // pds= pds.replace("ES", "O");
    // pds= pds.replace("EC", "P");
    // pds= pds.replace("CS", "M");
    // pds= pds.replace("VE", "T");
    // pds= pds.replace("VC", "X");
    // 我的思维逻辑是先将 PDS 的数字逻辑和离散数学归纳识别，然后走肽展识别，最大缩短元基长度
    public String initonPDE_DCDLwithBYS(String pds, double pDE_KEY_rate, StringBuilder
pDEKey, boolean isBys) {
        if(!isBys) {
            pds= pds.replace("UQIUDIIU", "T");
            pds= pds.replace("UQIUDIDU", "T");
            pds= pds.replace("UQDUDIIU", "T");
            pds= pds.replace("UQDUDIDU", "T");
            pds= pds.replace("UQIUDIDI", "X");
            pds= pds.replace("UQDUDIDI", "X");
            pds= pds.replace("IUDIDIQI", "M");
            pds= pds.replace("DUDIDIQI", "M");
            pds= pds.replace("IUDIIU", "+");
            pds= pds.replace("IUDIDU", "+");
            pds= pds.replace("DUDIIU", "+");
            pds= pds.replace("DUDIDU", "+");
            pds= pds.replace("IUDIDI", "-");
            pds= pds.replace("DUDIDI", "-");
            pds= pds.replace("IUDI", "H");
            pds= pds.replace("DUDI", "H");
            pds= pds.replace("UQQI", "A");
            pds= pds.replace("IUQI", "O");
            pds= pds.replace("DUQI", "O");
            pds= pds.replace("IUDI", "P");
            pds= pds.replace("DUDI", "P");
            pds= pds.replace("QI", "S");
            pds= pds.replace("UQ", "V");
            pds= pds.replace("IU", "E");
            pds= pds.replace("DU", "E");
            pds= pds.replace("DI", "C");
            pds= pds.replace("Q", "Q");
            pds= pds.replace("U", "D");
            pds= pds.replace("I", "I");
```

```
    pds= pds.replace("D", "D");

    //PDE
    pds= pds.replace("VS", "A");
    pds= pds.replace("ES", "O");
    pds= pds.replace("EC", "P");
    pds= pds.replace("HE", "+");
    pds= pds.replace("HC", "-");
    pds= pds.replace("VE", "T");
    pds= pds.replace("VC", "X");
    pds= pds.replace("CS", "M");

    return pds;
}
if(Math.random()< pDE_KEY_rate) {
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("UQIUDIIU", "T");
    }else {
        pds= pds.replace("UQIUDIDU", "T");
    }
}else {
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("UQDUDIIU", "T");
    }else {
        pds= pds.replace("UQDUDIDU", "T");
    }
}
if(Math.random()< pDE_KEY_rate) {
    pds= pds.replace("UQIUDIDI", "X");
}else {
    pds= pds.replace("UQDUDIDI", "X");
}
if(Math.random()<pDE_KEY_rate) {
    pds= pds.replace("IUDIDIQI", "M");
}else {
    pds= pds.replace("DUDIDIQI", "M");
}
if(Math.random()< pDE_KEY_rate) {
    if(Math.random()<pDE_KEY_rate) {
        pds= pds.replace("IUDIIU", "+");
    }else {
        pds= pds.replace("IUDIDU", "+");
    }
}else {
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("DUDIIU", "+");
    }else {
```

```
            pds= pds.replace("DUDIDU", "+");
        }
    }
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("IUDIDI", "-");
    }else {
        pds= pds.replace("DUDIDI", "-");
    }
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("IUDI", "H");
    }else {
        pds= pds.replace("DUDI", "H");
    }
    pds= pds.replace("UQQI", "A");
    if(Math.random()<pDE_KEY_rate) {
        pds= pds.replace("IUQI", "O");
    }else {
        pds= pds.replace("DUQI", "O");
    }
    if(Math.random()< pDE_KEY_rate) {
        pds= pds.replace("IUDI", "P");
    }else {
        pds= pds.replace("DUDI", "P");
    }
    pds= pds.replace("QI", "S");
    pds= pds.replace("UQ", "V");
    if(Math.random()<pDE_KEY_rate) {
        pds= pds.replace("IU", "E");
    }else {
        pds= pds.replace("DU", "E");
    }
    pds= pds.replace("DI", "C");
    pds= pds.replace("D", "D");
    pds= pds.replace("Q", "Q");
    pds= pds.replace("U", "U");
    pds= pds.replace("I", "I");
    //PDE
    pds= pds.replace("VS", "A");
    pds= pds.replace("ES", "O");
    pds= pds.replace("EC", "P");
    pds= pds.replace("HE", "+");
    pds= pds.replace("HC", "-");
    pds= pds.replace("VE", "T");
    pds= pds.replace("VC", "X");
    pds= pds.replace("CS", "M");
    return pds;
}
```

}

肽展公式参考：
AOPM VECS IDUQ 肽展公式推导与元基编码进化计算以及它的应用发现 1.2.2 国家软著申请 流水号 <2020Z11L0356797> 国作登字 2021-A-00942587 (中华人民共和国 国家版权局)
AOPM-VECS-IDUQ Catalytic INITONS PDE LAW and Its Application
#####
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/%E8%91%97%E4%BD%9C%E6%9D%83%E7%89%88%E6%9C%ACPDE_Formular_1_2_2.pdf
#####
https://github.com/yaoguangluo/Deta_Resource/blob/master/%E8%91%97%E4%BD%9C%E6%9D%83%E7%89%88%E6%9C%ACPDE_Formular_1_2_2.pdf

元基命名参考：
<见类人 DNA 与 神经元基于催化算子映射编码方式 V_1.2.2 版本国家软著申请 流水号 <2020Z11L0333706>
#####
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20%E7%BC%96%E7%A0%81%E6%96%B9%E5%BC%8F1.2.2%20%E4%B8%AD%E8%AF%91%E8%8B%B1%E6%B7%B7%E5%90%88%E7%89%88.pdf
#####
https://github.com/yaoguangluo/Deta_Resource/blob/master/DNA%20%E7%BC%96%E7%A0%81%E6%96%B9%E5%BC%8F1.2.2%20%E4%B8%AD%E8%AF%91%E8%8B%B1%E6%B7%B7%E5%90%88%E7%89%88.pdf

DNA 催化 与 肽展计算 和 AOPM-TXH-VECS-IDUQ 元基解码 V013_026 中文版本 国家著作申请 流水号 <2020Z11L0386462> 国作登字 2021-A-00942586 (中华人民共和国 国家版权局)
https://github.com/yaoguangluo/Deta_Resource/blob/master/DNA%20Initon%20 解码%20013026 软著申请中文最终版本.pdf
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20Initon%20 解码%20013026 软著申请中文最终版本.pdf
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20Initon%20 解码%20013026 软著申请中文最终版本%20 修正'食'字.pdf
https://github.com/yaoguangluo/Deta_Resource/blob/master/DNA%20Initon%20 解码%20013026 软著申请中文最终版本%20 修正'食'字.pdf

元基欧拉环计算参考：
https://gitee.com/DetaChina/dna_-db/blob/master/Initon_Math/org/math/initon/ouler/FindOulerRing.java

双元罗盘参考：
多人著作(罗瑶光, 罗荣武) DNA 元基催化与肽计算第二卷 养疗经应用研究 20210305 国家著作申请 流水号 <2021Z11L1057159>
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/罗瑶光-DNA 催化与肽计算第二卷 20210305.pdf
https://github.com/yaoguangluo/Deta_Catalytic_DNA/blob/master/罗瑶光-DNA 催化与肽计算第二卷 20210305.pdf

走到这，元基数学 公式表已经出来了，下一步就开始简单应用。

Yaoguang.Luo
罗瑶光

正在 将 eclipse 的工程往 idea 上 部署调试。看了下 license， 教育版是侵权版本。 又改回 eclipse