第十四章 DNA 加密

第一节 DNA 加密的动机

DNA 加密的动机说起来有点意思, 很久以前, 我在传媒上看见国家南京大学一教授破解了 MD5 算法, 我思考了很久, 因为互联网的所有登录认证都是MD5 加密token, 如果MD5 破解, 还还有什么价值？于是我在做SOCKET 流 PLSQL 数据库的登录方式时候, 我想我一定要自己设计一种非对称加密 token 保护我的web 登录方式. 这是我的最早的动机. 现在token 的元基加密方式已经成功, 彻底打破了 MD5 的生态环境, 我将其思想开源目的不是毁掉整个加密界, 而是迅速的生产力优化更新.

第二节 DNA 加密的应用需求

目前我的需求有 3 个方向

1 满足我的 web 通讯 session 精简不能冗长.2 满足本地

数据的概率元基加密

3 满足物理无语义元基加密的迅捷化（极速加密）.

```
package OSI. OSU. SI. SD. SU. SQ. ASU. OSU. PSU. MSU. AVQ. ASQ. tin. catalytic. procedure. pde;

import java. util. HashMap;

import java. util. Map;

//注意: 该 文件对应的是罗瑶光先生 DNA 编码 与 计算的两本 国家软著作 思想的编码 实现.

//公安部 与 知识产权委员会 已经备案, 可阅读 相关 著作权 原文 进行逻辑辨别.

//作者: 罗瑶光

//思想: 罗瑶光

public class FullDNATokenPDI {

    public double[] key= new double[4];

    public String bys= "";

    public String pdw= "";
```

```
public String pds= "";
public String pde= "";
public String time= "";
public String session_key= "";
public String text;
public String cacheId;
public String code= "";
public String lock= "";
public String pdedeKey= "";
public String pdedsKey= "";
public String pdeieKey= "";
public String pdeisKey= "";

public void doKeyPress(String initons, FullDNATokenPDI pDE_RNA_FullFormular, boolean bYS)
    { Initon[] initon= new Initon[initons. length()];
    for(int i= 0; i< initons. length(); i++)
        { if(initon[i]== null) {
            initon[i]= new Initon();
        }
        initon[i]. setIniton(""+ initons. charAt(i));
        if(i+ 1< initons. length()) {
            if(initon[i+ 1]== null)
                { initon[i+ 1]= new
                Initon();
            }
            initon[i]. next= initon[i+ 1];
            initon[i+ 1]. prev= initon[i];
        }
    }
    if(null!= initons&& initons. length()> 0)
        { do_PDE_RNA_FullFormular(initon[0], pDE_RNA_FullFormular,
        bYS);
```

```
        }
    }
    public static void main(String[] argv) {
        FullDNATokenPDI pDE_RNA_FullFormular= new FullDNATokenPDI();
        @SuppressWarnings("unused")
        String initonKeys= "EIU/0. 6/EDU/0. 4/si/0. 3/sq/0. 7/EIU/0. 5/EDU/0. 5/si/0. 632/sq/0. 368";
        pDE_RNA_FullFormular. key[0]= 0. 6;
        pDE_RNA_FullFormular. key[1]= 0.3;
        pDE_RNA_FullFormular. key[2]= 0.5;
        pDE_RNA_FullFormular. key[3]= 0. 632;

        pDE_RNA_FullFormular. text= "控制吸收";
        pDE_RNA_FullFormular. pdw= pDE_RNA_FullFormular. initonSect(pDE_RNA_FullFormular. text);
        System. out. println("原文: " + pDE_RNA_FullFormular. text);
        //pDE_RNA_FullFormular. pdw= "字典保密: MSIOCUOCIPCUPCI";
        String[] lock= new String[12];
lock[0] = "A"; lock[3] = "O"; lock[6] = "P"; lock[9] = "M";
lock[1] = "V"; lock[4] = "E"; lock[7] = "C"; lock[10] = "S";
lock[2] = "I"; lock[5] = "D"; lock[8] = "U"; lock[11] = "Q";
int i= (int)(Math. random()* 12)% 12;
pDE_RNA_FullFormular. lock+= lock[i];
i= (int)(Math. random()* 12)% 12;
pDE_RNA_FullFormular. lock+=lock[i];
i= (int)(Math. random()* 12)% 12;
pDE_RNA_FullFormular. lock+=lock[i];
i= (int)(Math. random()* 12)% 12;
pDE_RNA_FullFormular. lock+=lock[i];

        for(i= 0; i< pDE_RNA_FullFormular. pdw. length(); i++) {
            pDE_RNA_FullFormular. code+= pDE_RNA_FullFormular. lock + pDE_RNA_FullFormular. pdw.
```

```
charAt(i);
        }
        System. out. println(" 肽 语 : "+ pDE_RNA_FullFormular. pdw);
        System. out. println(" 肽 锁 : "+ pDE_RNA_FullFormular. lock);
        System. out. println("散列肽语: "+ pDE_RNA_FullFormular. code);
        pDE_RNA_FullFormular. bys= "0. 6/0. 3/0. 5/0. 632";
        System. out. println("静态密钥: "+ pDE_RNA_FullFormular. bys);
        pDE_RNA_FullFormular. doKeyPress(pDE_RNA_FullFormular. code, pDE_RNA_FullFormular, false);
        System. out. println("静态肽展降元概率钥匙 E: "+ pDE_RNA_FullFormular. pdedeKey);
        System. out. println("静态肽展降元概率钥匙 S: "+ pDE_RNA_FullFormular. pdedsKey);
        System. out. println("静态肽展降元: "+ pDE_RNA_FullFormular. pds);

        System. out. println("静态肽展增元概率钥匙 E: "+ pDE_RNA_FullFormular. pdeieKey);
        System. out. println("静态肽展增元概率钥匙 S: "+ pDE_RNA_FullFormular. pdeisKey);
        System. out. println("静态肽展增元: "+ pDE_RNA_FullFormular. pde);

        pDE_RNA_FullFormular. time= "" + System. currentTimeMillis();
        pDE_RNA_FullFormular. cacheId= "ID" + Math. random() + ": " + Math. random();
        System. out. println("时间: " + pDE_RNA_FullFormular. time);
        System. out. println("账号随机缓存字符串: " + pDE_RNA_FullFormular. cacheId);
        pDE_RNA_FullFormular. session_key= pDE_RNA_FullFormular. pde;
        System. out. println("Session: " + pDE_RNA_FullFormular. session_key);
        System. out.
println("================================================================================");
        System. out. println("开始前序验证: ");
        System. out. println("开始Session 解析: " + pDE_RNA_FullFormular. session_key);
        System. out. println("开始概率钥匙解析: " + pDE_RNA_FullFormular. pdedeKey+
pDE_RNA_FullFormular. pdedsKey
                + pDE_RNA_FullFormular. pdeieKey+ pDE_RNA_FullFormular. pdeisKey);
```

```
        FullDNATokenPDI pDE_RNA_FullFormular1= new  FullDNATokenPDI(); pDE_RNA_FullFormular1.
        pdedeKey= pDE_RNA_FullFormular. pdedeKey. toString(); pDE_RNA_FullFormular1. pdedsKey=
        pDE_RNA_FullFormular. pdedsKey. toString(); pDE_RNA_FullFormular1. pdeieKey=
        pDE_RNA_FullFormular. pdeieKey. toString(); pDE_RNA_FullFormular1. pdeisKey=
        pDE_RNA_FullFormular. pdeisKey. toString(); pDE_RNA_FullFormular.
        doKeyUnPress(pDE_RNA_FullFormular. code, pDE_RNA_FullFormular1,
true);
        System. out. println("得到原降元元基 DNA 序列: "+ pDE_RNA_FullFormular. pds);
        System. out. println("得到新降元元基 DNA 序列: "+ pDE_RNA_FullFormular1. pds);
        System. out. println(" 得 到 原 元 基 DNA 序 列 : "+ pDE_RNA_FullFormular. pde);
        System. out. println(" 得 到 新 元 基 DNA 序 列 : "+ pDE_RNA_FullFormular1. pde);
        System. out. println("验证正确？");
        System. out. println(pDE_RNA_FullFormular. pde. equals(pDE_RNA_FullFormular1. pde)? "正确": "失
败");
        System. out.
println("==========================================================================");
        System. out. println("开始后序验证: ");
        FullDNATokenPDI pDE_RNA_FullFormular2= new FullDNATokenPDI();
        pDE_RNA_FullFormular2. pdeieKey= pDE_RNA_FullFormular. pdedeKey. toString();
        pDE_RNA_FullFormular2. pdeisKey= pDE_RNA_FullFormular. pdedsKey. toString();
        pDE_RNA_FullFormular2. pdedeKey= pDE_RNA_FullFormular. pdeieKey. toString();
        pDE_RNA_FullFormular2. pdedsKey= pDE_RNA_FullFormular. pdeisKey. toString();
        System. out. println("准备计算元基 DNA 序列: "+ pDE_RNA_FullFormular1. pde);
        pDE_RNA_FullFormular2. doSessionKeyUnPress(pDE_RNA_FullFormular1. pde,
pDE_RNA_FullFormular2, true);
        System. out. println("得到原续降元元基 DNA 序列: "+ pDE_RNA_FullFormular1. pds);
        System. out. println("得到后续降元元基 DNA 序列: "+ pDE_RNA_FullFormular2. pds);
        System. out. println("验证正确？");
        System. out. println(pDE_RNA_FullFormular1. pds. equals(pDE_RNA_FullFormular2. pds)? "正确": "失
败");
```

```
        System. out.
println("================================================================================");
        System. out. println("开始整序验证: ");
        FullDNATokenPDI pDE_RNA_FullFormular3= new FullDNATokenPDI();
        pDE_RNA_FullFormular3. pdeieKey= pDE_RNA_FullFormular. pdedeKey. toString();
        pDE_RNA_FullFormular3. pdeisKey= pDE_RNA_FullFormular. pdedsKey. toString();
        pDE_RNA_FullFormular3. pdedeKey= pDE_RNA_FullFormular. pdeieKey. toString();
        pDE_RNA_FullFormular3. pdedsKey= pDE_RNA_FullFormular. pdeisKey. toString();
        System. out. println("准备计算元基 DNA 序列: "+ pDE_RNA_FullFormular1. pde);
        pDE_RNA_FullFormular3. doFullSessionKeyUnPress(pDE_RNA_FullFormular1. pde,
pDE_RNA_FullFormular3, true);
        System. out. println("得到原续降元元基 DNA 序列: "+ pDE_RNA_FullFormular1. pds);
        System. out. println("得到后续降元元基 DNA 序列: "+ pDE_RNA_FullFormular3. pds);
        System. out. println("验证正确？");
        System. out. println(pDE_RNA_FullFormular1. pds. equals(pDE_RNA_FullFormular3. pds)? "正确": "失
败");
        System. out. println("准备整序计算元基 DNA 序列: "+ pDE_RNA_FullFormular. code);
        System. out. println("准备整序计算元基 DNA 序列: "+ pDE_RNA_FullFormular3. pde);
        System. out. println(pDE_RNA_FullFormular1. code. equals(pDE_RNA_FullFormular3. pde)? "正确": "
失败");
    }
    private void doFullSessionKeyUnPress(String initons, FullDNATokenPDI pDE_RNA_FullFormular3, boolean
bYS) {
        Initon[] initon= new Initon[initons. length()];
        for(int i= 0; i< initons. length(); i++) {
            if(initon[i]== null)
                { initon[i]= new
                Initon();
            }
            initon[i]. setIniton(""+ initons. charAt(i));
            if(i+ 1< initons. length()) {
```

```
                if(initon[i+ 1]== null)
                    { initon[i+ 1]= new
                    Initon();
                }
                initon[i]. next= initon[i+ 1];
                initon[i+ 1]. prev= initon[i];
            }
        }
        do_PDE_RNA_FullFormular_FullBack(initon[0], pDE_RNA_FullFormular3, bYS);
    }
    private void do_PDE_RNA_FullFormular_FullBack(Initon initon, FullDNATokenPDI
pDE_RNA_FullFormular3
            , boolean bYS)
        { Initon InitonPDE=
        initon;
        InitonLinkDNA initonLinkDNA= new InitonLinkDNA();
        //c->b
        Initon InitonPDEM= doDecrementM(InitonPDE, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDEP= doDecrementP(InitonPDEM, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDEO= doDecrementO(InitonPDEP, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDEA= doDecrementA(InitonPDEO, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDES= doDecrementS(InitonPDEA, initonLinkDNA, pDE_RNA_FullFormular3, bYS);
        Initon InitonPDEC= doDecrementC(InitonPDES, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDEE= doDecrementE(InitonPDEC, initonLinkDNA, pDE_RNA_FullFormular3, bYS);
        Initon InitonPDE1= doDecrementV(InitonPDEE, initonLinkDNA, pDE_RNA_FullFormular3);
        while(InitonPDE1. hasNext()) {
            pDE_RNA_FullFormular3. pds+= InitonPDE1. getStore();
            InitonPDE1= InitonPDE1. next;
        }
        pDE_RNA_FullFormular3. pds+= InitonPDE1. getStore();
        while(InitonPDE1. hasPrev()) {
            InitonPDE1= InitonPDE1. prev;
```

```
        }
        //VECS CSVE VCES CVES CEVS

        Initon InitonPDE1V= doIncrementV(InitonPDE1, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDE1E= doIncrementE(InitonPDE1V, initonLinkDNA, pDE_RNA_FullFormular3, bYS);
        Initon InitonPDE1C= doIncrementC(InitonPDE1E, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDE1S= doIncrementS(InitonPDE1C, initonLinkDNA, pDE_RNA_FullFormular3, bYS);

        Initon InitonPDE1A= doIncrementA(InitonPDE1S, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDE1O= doIncrementO(InitonPDE1A, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDE1P= doIncrementP(InitonPDE1O, initonLinkDNA, pDE_RNA_FullFormular3);
        Initon InitonPDE2= doIncrementM(InitonPDE1P, initonLinkDNA, pDE_RNA_FullFormular3);
        while(InitonPDE2. hasNext()) {
            pDE_RNA_FullFormular3. pde+= InitonPDE2. getStore();
            InitonPDE2= InitonPDE2. next;
        }
        pDE_RNA_FullFormular3. pde+= InitonPDE2. getStore();
        while(InitonPDE2. hasPrev()) {
            InitonPDE2= InitonPDE2. prev;
        }
    }

    private void doSessionKeyUnPress(String initons, FullDNATokenPDI pDE_RNA_FullFormular2, boolean bYS)
{
        Initon[] initon= new Initon[initons. length()];
        for(int i= 0; i< initons. length(); i++) {
            if(initon[i]== null)
                { initon[i]= new
                Initon();
            }
            initon[i]. setIniton(""+ initons. charAt(i));
```

```
            if(i+ 1< initons. length())
                { if(initon[i+ 1]== null)
                {
                    initon[i+ 1]= new Initon();
                }
                initon[i]. next= initon[i+ 1];
                initon[i+ 1]. prev= initon[i];
            }
        }
        do_PDE_RNA_FullFormular_Back(initon[0], pDE_RNA_FullFormular2, bYS);
    }

    private void do_PDE_RNA_FullFormular_Back(Initon initon, FullDNATokenPDI pDE_RNA_FullFormular
            , boolean bYS)
        { Initon InitonPDE=
        initon;
        InitonLinkDNA initonLinkDNA= new InitonLinkDNA();
        Initon InitonPDEM= doDecrementM(InitonPDE, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEP= doDecrementP(InitonPDEM, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEO= doDecrementO(InitonPDEP, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEA= doDecrementA(InitonPDEO, initonLinkDNA, pDE_RNA_FullFormular);

        Initon InitonPDES= doDecrementS(InitonPDEA, initonLinkDNA, pDE_RNA_FullFormular, bYS);
        Initon InitonPDEC= doDecrementC(InitonPDES, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEE= doDecrementE(InitonPDEC, initonLinkDNA, pDE_RNA_FullFormular, bYS);
        Initon InitonPDE1= doDecrementV(InitonPDEE, initonLinkDNA, pDE_RNA_FullFormular);
        while(InitonPDE1. hasNext()) {
            pDE_RNA_FullFormular. pds+= InitonPDE1. getStore();
            InitonPDE1= InitonPDE1. next;
        }
        pDE_RNA_FullFormular. pds+= InitonPDE1. getStore();
        while(InitonPDE1. hasPrev()) {
```

```
            InitonPDE1= InitonPDE1. prev;
        }
//      Initon InitonPDE1S= doIncrementS(InitonPDE1, initonLinkDNA, pDE_RNA_FullFormular, bYS);
//      Initon InitonPDE1C= doIncrementC(InitonPDE1S, initonLinkDNA, pDE_RNA_FullFormular);
//      Initon InitonPDE1E= doIncrementE(InitonPDE1C, initonLinkDNA, pDE_RNA_FullFormular, bYS);
//      Initon InitonPDE1V= doIncrementV(InitonPDE1E, initonLinkDNA, pDE_RNA_FullFormular);
//      Initon InitonPDE1M= doIncrementM(InitonPDE1V, initonLinkDNA, pDE_RNA_FullFormular);
//      Initon InitonPDE1P= doIncrementP(InitonPDE1M, initonLinkDNA, pDE_RNA_FullFormular);
//      Initon InitonPDE1O= doIncrementO(InitonPDE1P, initonLinkDNA, pDE_RNA_FullFormular);
//      Initon InitonPDE2= doIncrementA(InitonPDE1O, initonLinkDNA, pDE_RNA_FullFormular);
//      while(InitonPDE2. hasNext()) {
//          pDE_RNA_FullFormular. pde+= InitonPDE2. getStore();
//          InitonPDE2= InitonPDE2. next;
//      }
//      pDE_RNA_FullFormular. pde+= InitonPDE2. getStore();
//      while(InitonPDE2. hasPrev()) {
//          InitonPDE2= InitonPDE2. prev;
//      }
    }
    private void doKeyUnPress(String initons, FullDNATokenPDI pDE_RNA_FullFormular, boolean bYS)
        { Initon[] initon= new Initon[initons. length()];
        for(int i= 0; i< initons. length(); i++)
            { if(initon[i]== null) {
                initon[i]= new Initon();
            }
            initon[i]. setIniton(""+ initons. charAt(i));
            if(i+ 1< initons. length()) {
                if(initon[i+ 1]== null)
                    { initon[i+ 1]= new
                    Initon();
                }
```

```
                initon[i]. next= initon[i+ 1];
                initon[i+ 1]. prev= initon[i];
            }
        }
        do_PDE_RNA_FullFormular(initon[0], pDE_RNA_FullFormular, bYS);
    }

    public void do_PDE_RNA_FullFormular(Initon initon, FullDNATokenPDI pDE_RNA_FullFormular
            , boolean bYS)
        { Initon InitonPDE=
        initon;
        InitonLinkDNA initonLinkDNA= new InitonLinkDNA();
        //a->b
        Initon InitonPDEA= doDecrementA(InitonPDE, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEO= doDecrementO(InitonPDEA, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEP= doDecrementP(InitonPDEO, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEM= doDecrementM(InitonPDEP, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEV= doDecrementV(InitonPDEM, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDEE= doDecrementE(InitonPDEV, initonLinkDNA, pDE_RNA_FullFormular, bYS);
        Initon InitonPDEC= doDecrementC(InitonPDEE, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDE1= doDecrementS(InitonPDEC, initonLinkDNA, pDE_RNA_FullFormular, bYS);
        while(InitonPDE1. hasNext()) {
            pDE_RNA_FullFormular. pds+= InitonPDE1. getStore();
            InitonPDE1= InitonPDE1. next;
        }
        pDE_RNA_FullFormular. pds+= InitonPDE1. getStore();
        while(InitonPDE1. hasPrev()) {
            InitonPDE1= InitonPDE1. prev;
        }
        //b->c
        Initon InitonPDE1V= doIncrementV(InitonPDE1, initonLinkDNA, pDE_RNA_FullFormular);
```

```
        Initon InitonPDE1E= doIncrementE(InitonPDE1V, initonLinkDNA, pDE_RNA_FullFormular, bYS);
        Initon InitonPDE1C= doIncrementC(InitonPDE1E, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDE1S= doIncrementS(InitonPDE1C, initonLinkDNA, pDE_RNA_FullFormular, bYS);
        Initon InitonPDE1A= doIncrementA(InitonPDE1S, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDE1O= doIncrementO(InitonPDE1A, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDE1P= doIncrementP(InitonPDE1O, initonLinkDNA, pDE_RNA_FullFormular);
        Initon InitonPDE2= doIncrementM(InitonPDE1P, initonLinkDNA, pDE_RNA_FullFormular);
        while(InitonPDE2. hasNext()) {
            pDE_RNA_FullFormular. pde+= InitonPDE2. getStore();
            InitonPDE2= InitonPDE2. next;
        }
        pDE_RNA_FullFormular. pde+= InitonPDE2. getStore();
        while(InitonPDE2. hasPrev()) {
            InitonPDE2= InitonPDE2. prev;
        }
    }

/////INITONS SWAP
    private Initon doIncrementA(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_FullFormular(). PDE_IncrementA(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println();
        System. out. print("A->");
```

```
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementO(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_FullFormular(). PDE_IncrementO(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("O->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
```

```
private Initon doIncrementP(Initon InitonPDE, InitonLinkDNA initonLinkDNA
        , FullDNATokenPDI pDE_RNA_FullFormular)
    { while(InitonPDE. hasPrev()) {
        InitonPDE= InitonPDE. prev;
    }
    initonLinkDNA. setInitonLink(InitonPDE);
    InitonPDE= new PDE_Increment_FullFormular(). PDE_IncrementP(initonLinkDNA);
    while(InitonPDE. hasPrev()) {
        InitonPDE= InitonPDE. prev;
    }
    System. out. println(); System.
    out. print("P->");
    while(InitonPDE. hasNext()) {
        System. out. print(InitonPDE. getStore());
        InitonPDE= InitonPDE. next;
    }
    while(InitonPDE. hasPrev())
        { InitonPDE= InitonPDE. prev;
    }
    return InitonPDE;
}
private Initon doIncrementM(Initon InitonPDE, InitonLinkDNA initonLinkDNA
        , FullDNATokenPDI pDE_RNA_FullFormular)
    { while(InitonPDE. hasPrev()) {
        InitonPDE= InitonPDE. prev;
    }
    initonLinkDNA. setInitonLink(InitonPDE);
    InitonPDE= new PDE_Increment_FullFormular(). PDE_IncrementM(initonLinkDNA);
    while(InitonPDE. hasPrev()) {
        InitonPDE= InitonPDE. prev;
```

```
        }
        System. out. println(); System.
        out. print("M->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementV(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_FullFormular(). PDE_IncrementV(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("V->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
```

```
        }
        return InitonPDE;
    }
    private Initon doIncrementC(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_FullFormular(). PDE_IncrementC(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("C->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementE(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular, boolean bYS)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
```

```
        InitonPDE= new PDE_Increment_FullDNAFormular(). PDE_IncrementE_DU(initonLinkDNA
                , pDE_RNA_FullFormular, bYS);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("E->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementS(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular, boolean bYS)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_FullDNAFormular(). PDE_IncrementS_IQ(initonLinkDNA
                , pDE_RNA_FullFormular);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("S->");
        while(InitonPDE. hasNext()) {
```

```
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }

    private Initon doDecrementA(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_FullFormular(). PDE_DecrementA(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("A->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
```

```
private Initon doDecrementO(Initon InitonPDE, InitonLinkDNA initonLinkDNA
        , FullDNATokenPDI pDE_RNA_FullFormular)
    { while(InitonPDE. hasPrev()) {
        InitonPDE= InitonPDE. prev;
    }
    initonLinkDNA. setInitonLink(InitonPDE);
    InitonPDE= new PDE_Decrement_FullFormular(). PDE_DecrementO(initonLinkDNA);
    while(InitonPDE. hasPrev()) {
        InitonPDE= InitonPDE. prev;
    }
    System. out. println(); System.
    out. print("O->");
    while(InitonPDE. hasNext()) {
        System. out. print(InitonPDE. getStore());
        InitonPDE= InitonPDE. next;
    }
    while(InitonPDE. hasPrev())
        { InitonPDE= InitonPDE. prev;
    }
    return InitonPDE;
}
private Initon doDecrementP(Initon InitonPDE, InitonLinkDNA initonLinkDNA
        , FullDNATokenPDI pDE_RNA_FullFormular)
    { while(InitonPDE. hasPrev()) {
        InitonPDE= InitonPDE. prev;
    }
    initonLinkDNA. setInitonLink(InitonPDE);
    InitonPDE= new PDE_Decrement_FullFormular(). PDE_DecrementP(initonLinkDNA);
    while(InitonPDE. hasPrev()) {
```

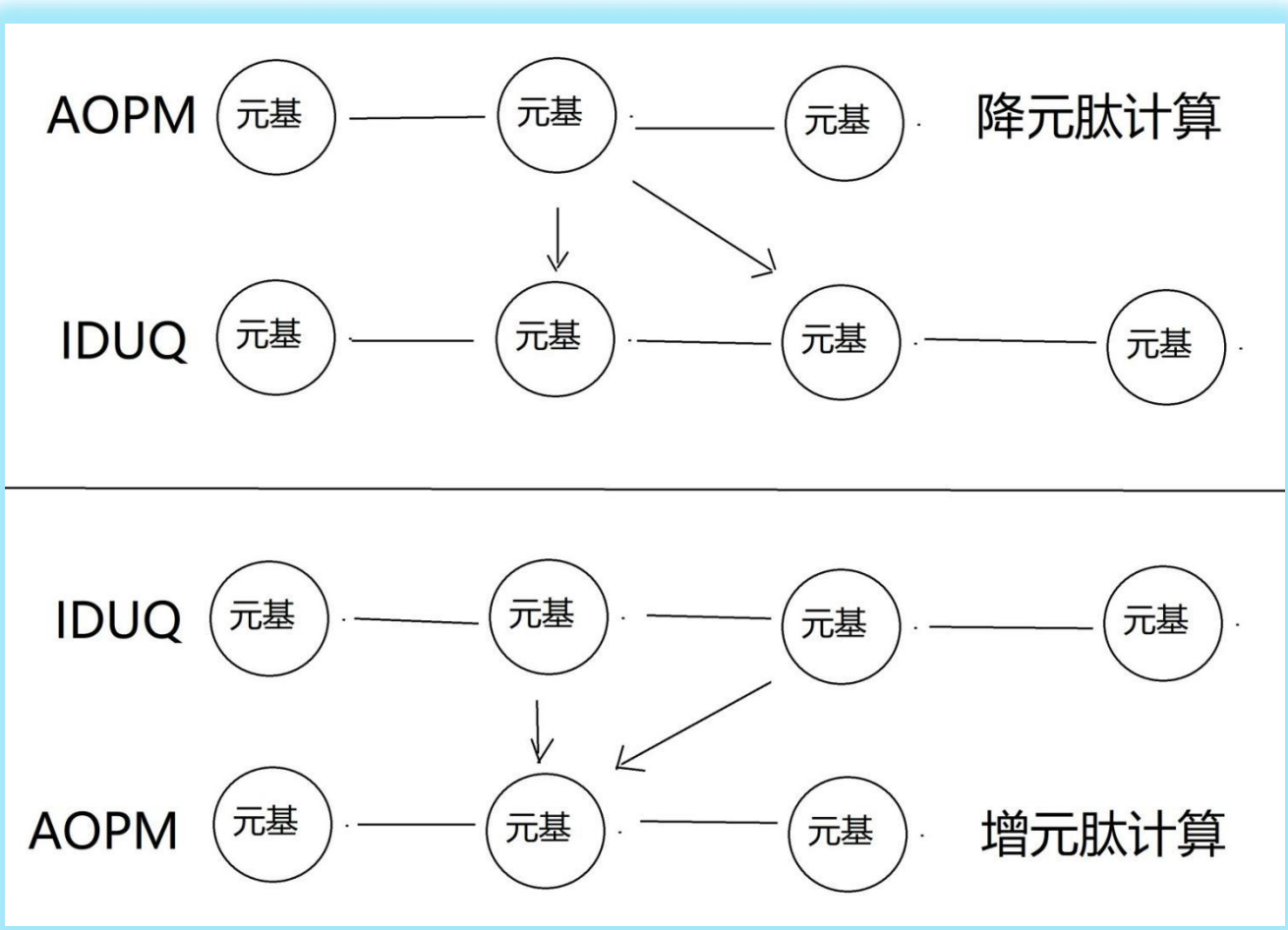
```
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("P->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementM(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_FullFormular(). PDE_DecrementM(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("M->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev()) {
```

```
            InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementV(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_FullFormular(). PDE_DecrementV(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("V->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementC(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
```

```
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_FullFormular(). PDE_DecrementC(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("C->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementE(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular, boolean bYS)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_FullDNAFormular(). PDE_DecrementE_DU(initonLinkDNA
                , pDE_RNA_FullFormular, bYS);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("E->");
        while(InitonPDE. hasNext()) {
```

```
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementS(Initon InitonPDE, InitonLinkDNA initonLinkDNA
            , FullDNATokenPDI pDE_RNA_FullFormular, boolean bYS)
        { while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_FullDNAFormular(). PDE_DecrementS_IQ(initonLinkDNA
                , pDE_RNA_FullFormular, bYS);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        System. out. println(); System.
        out. print("S->");
        while(InitonPDE. hasNext()) {
            System. out. print(InitonPDE. getStore());
            InitonPDE= InitonPDE. next;
        }
        System. out. println();
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
```

```
    }
    public String initonSect(String key)
        { String pdis= "";
        Map<String, String> map= new HashMap<>();
        map. put("0", "A");
        map. put("1", "O");
        map. put("2", "P");
        map. put("3", "M");
        map. put("4", "V");
        map. put("5", "E");
        map. put("6", "C");
        map. put("7", "S");
        map. put("8", "I");
        map. put("9", "D");
        map. put(". ", "U");
        for(int i= 0; i< key. length(); i++)
            { int c= key. charAt(i);
            String ints= ""+c;
            for(int j= 0; j< ints. length(); j++)
                { pdis+=map. get(""+ ints.
                charAt(j));
            }
            pdis+="U";
        }
        return pdis;
    }
}
```



肽展计算的逻辑起始非常简单，是一种线性的链表计算模型. 增元和降元的区别在第一卷的肽展公式推导论著已经描述了，这里就不多介绍.

//降元

```
package OSI. OSU. SI. SD. SU. SQ. ASU. OSU. PSU. MSU. AVQ. ASQ. tin. catalytic. procedure.pde;
//注意: 该 文件对应的是罗瑶光先生 DNA 编码 与 计算的两本 国家软著作 思想的编码 实现.
//公安部 与 知识产权委员会 已经备案，可阅读 相关 著作权 原文 进行逻辑辨别.
//作者：罗瑶光
//思想：罗瑶光
```

```
public class PDE_Decrement_FullDNAFormular {
    //A = V + S LINK 数据结构对象  (简单测试)
    //作用发现有A 的initon进行VPCS级展开. 按我在  印度  ANSI C代码风格进行编写方式.
    public Initon PDE_DecrementA(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("A"))
                { Initon initonIncrementV= new Initon();
                initonIncrementV. setV(); //新增一个数据V
                Initon initonIncrementS= new Initon();
                initonIncrementS. setS(); //新增一个数据S
                initonIncrementV. next= initonIncrementS; //V初始
                initonIncrementV. prev= initonLink. prev;
                if(null!= initonIncrementV. prev)  { initonIncrementV.
                    prev. next= initonIncrementV;
                }
                initonIncrementS. prev= initonIncrementV; //S初始
                initonIncrementS. next= initonLink. next;
                if(null!= initonIncrementS. next) {
                    initonIncrementS. next. prev= initonIncrementS;
                }
                initonLink= initonIncrementS; //最后S代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop  替增.
        }
        return initonLink;
    }
```

```
//O = E + S LINK 数据结构对象  (简单测试)
public Initon PDE_DecrementO(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("O"))
            { Initon initonIncrementE= new Initon();
            initonIncrementE. setE(); //新增一个数据E
            Initon initonIncrementS= new Initon();
            initonIncrementS. setS(); //新增一个数据S
            initonIncrementE. next= initonIncrementS; //E初始
            initonIncrementE. prev= initonLink. prev;
            if(null!= initonIncrementE. prev)  { initonIncrementE.
                prev. next= initonIncrementE;
            }
            initonIncrementS. prev= initonIncrementE; //S初始
            initonIncrementS. next= initonLink. next;
            if(null!= initonIncrementS. next) {
                initonIncrementS. next. prev= initonIncrementS;
            }
            initonLink= initonIncrementS; //最后S代替
        }
        if(!initonLink. hasNext()) {
            return initonLink;
        }
        initonLink= initonLink. next; //while loop  替增.
    }
    return initonLink;
}
//P = E + C LINK 数据结构对象  (简单测试)
public Initon PDE_DecrementP(InitonLinkDNA initonLinkDNA) {
```

```
        Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("P"))
                { Initon initonIncrementE= new Initon();
                initonIncrementE. setE(); //新增一个数据E
                Initon initonIncrementC= new Initon();
                initonIncrementC. setC(); //新增一个数据C
                initonIncrementE. next= initonIncrementC; //E初始
                initonIncrementE. prev= initonLink. prev;
                if(null!= initonIncrementE. prev)  { initonIncrementE.
                    prev. next= initonIncrementE;
                }
                initonIncrementC. prev= initonIncrementE; //C初始
                initonIncrementC. next= initonLink. next;
                if(null!= initonIncrementC. next) {
                    initonIncrementC. next. prev= initonIncrementC;
                }
                initonLink= initonIncrementC; //最后C代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop  替增.
        }
        return initonLink;
    }
    //M = C + S
    public Initon PDE_DecrementM(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
```

```
            if(initonLink. getStore(). equalsIgnoreCase("M"))
                { Initon initonIncrementC= new Initon();
                initonIncrementC. setC(); //新增一个数据C
                Initon initonIncrementS= new Initon();
                initonIncrementS. setS(); //新增一个数据S
                initonIncrementC. next= initonIncrementS; //C初始
                initonIncrementC. prev= initonLink. prev;
                if(null!= initonIncrementC. prev)  { initonIncrementC.
                    prev. next= initonIncrementC;
                }
                initonIncrementS. prev= initonIncrementC; //s初始
                initonIncrementS. next= initonLink. next;
                if(null!= initonIncrementS. next) {
                    initonIncrementS. next. prev= initonIncrementS;
                }
                initonLink= initonIncrementS; //最后S代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop  替增.
        }
        return initonLink;
    }
    //V = U + Q
    public Initon PDE_DecrementV(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("V"))
                { Initon initonIncrementU= new Initon();
```

```
                initonIncrementU. setU(); //新增一个数据U
                Initon initonIncrementQ= new Initon();
                initonIncrementQ. setQ(); //新增一个数据Q

                initonIncrementU. next= initonIncrementQ; //U初始
                initonIncrementU. prev= initonLink. prev;
                if(null!= initonIncrementU. prev)  { initonIncrementU.
                    prev. next= initonIncrementU;
                }
                initonIncrementQ. prev= initonIncrementU; //Q初始
                initonIncrementQ. next= initonLink. next;
                if(null!= initonIncrementQ. next) {
                    initonIncrementQ. next. prev= initonIncrementQ;
                }
                initonLink= initonIncrementQ; //最后Q代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop  替增.
        }
        return initonLink;
    }
    //E = D + U
    //肽展公式的概率问题, 先这样命名, 之后讨论  是IU还是DU
    //E = D + U
    //肽展公式的概率问题, 先这样命名, 之后讨论  是IU还是DU
    //2021年2月10, 优化次黄嘌呤  对应  D 和  U 弱碱强碱. 罗瑶光. 这里受酸碱的浓度控制  正如  S= I和  S= Q
强酸弱酸.
    //之后研究  u是否可以略去. 今改下E= D  和  E= U
```

```
    public Initon PDE_DecrementE_DU(InitonLinkDNA initonLinkDNA, FullDNATokenPDI pDE_RNA_Formular,
boolean bYS) {
        Initon initonLink= initonLinkDNA. getInitonLink();
        int index= 0;
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("E"))
                { Initon initonIncrementI= new Initon();
                if(bYS) {
                    try {
                        if(pDE_RNA_Formular. pdedeKey. charAt(index)== '0')
                            { initonIncrementI. setD(); //新增一个数据D
                        }if(pDE_RNA_Formular. pdedeKey. charAt(index)== '1') {
                            initonIncrementI. setU(); //新增一个数据U
                        }
                        //else {
                        //     initonIncrementI. setE(); //新增一个数据E
                        //}
                        index++;
                    }catch(Exception e) {
                        e. printStackTrace();
                    }
                }else {
                    if(Math. random()< pDE_RNA_Formular. key[1])
                        { pDE_RNA_Formular. pdedeKey+= "0";
                        initonIncrementI. setD();
                    }else {
                        pDE_RNA_Formular. pdedeKey+= "1";
                        initonIncrementI. setU();
                    }
                }
```

```
                initonIncrementI. prev= initonLink. prev;
                if(null!= initonIncrementI. prev) {
                    initonIncrementI. prev. next= initonIncrementI;
                }

                initonIncrementI. next= initonLink. next;
                if(null!= initonIncrementI. next) {
                    initonIncrementI. next. prev= initonIncrementI;
                }
                initonLink= initonIncrementI; //最后I代替
            }
            //else if(initonLink. getStore(). equalsIgnoreCase("D")) {
            //    pDE_RNA_Formular. pdedeKey+= "2";
            //}else if(initonLink. getStore(). equalsIgnoreCase("U")) {
            //    pDE_RNA_Formular. pdedeKey+= "2";
            //}
            if(!initonLink. hasNext()) {
                //System. out. println(pDE_RNA_Formular. pdedsKey);
                return initonLink;
            }
            initonLink= initonLink. next; //while loop  替增.
        }
        //System. out. println(pDE_RNA_Formular. pdedsKey);
        return initonLink;
    }
    //C = I + D
    public Initon PDE_DecrementC(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("C")) {
```

```
                Initon initonIncrementI= new Initon();
                initonIncrementI. setI(); //新增一个数据I
                Initon initonIncrementD= new Initon();
                initonIncrementD. setD(); //新增一个数据D

                initonIncrementI. next= initonIncrementD; //I初始
                initonIncrementI. prev= initonLink. prev;
                if(null!= initonIncrementI. prev) {
                    initonIncrementI. prev. next= initonIncrementI;
                }
                initonIncrementD. prev= initonIncrementI; //D初始
                initonIncrementD. next= initonLink. next;
                if(null!= initonIncrementD. next) {
                    initonIncrementD. next. prev= initonIncrementD;
                }
                initonLink= initonIncrementD; //最后D代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop  替增.
        }
        return initonLink;
    }
    //S = I
    //肽展公式的概率问题, 先这样命名, 之后讨论  是I, Q还是I + Q
    public Initon PDE_DecrementS_IQ(InitonLinkDNA initonLinkDNA, FullDNATokenPDI pDE_RNA_Formular,
boolean bYS) {
        Initon initonLink= initonLinkDNA. getInitonLink();
        int index= 0;
```

```
while(null!= initonLink) {
    if(initonLink. getStore(). equalsIgnoreCase("S"))
        { Initon initonIncrementI= new Initon();
        if(bYS) {
            try {
                if(pDE_RNA_Formular. pdedsKey. charAt(index)== '0')
                    { initonIncrementI. setI(); //新增一个数据I
                }if(pDE_RNA_Formular. pdedsKey. charAt(index)== '1') {
                    initonIncrementI. setQ(); //新增一个数据Q
                }
                index++;
            }catch(Exception e) {
                e. printStackTrace();
            }
        }else {
            if(Math. random()< pDE_RNA_Formular. key[1])
                { pDE_RNA_Formular. pdedsKey+= "0";
                initonIncrementI. setI();
            }else {
                pDE_RNA_Formular. pdedsKey+= "1";
                initonIncrementI. setQ();
            }
        }
        initonIncrementI. prev= initonLink. prev;
        if(null!= initonIncrementI. prev) {
            initonIncrementI. prev. next= initonIncrementI;
        }
        initonIncrementI. next= initonLink. next;
        if(null!= initonIncrementI. next) {
            initonIncrementI. next. prev= initonIncrementI;
```

```
                }
                initonLink= initonIncrementI; //最后I代替
            }
            if(!initonLink. hasNext()) {
                System. out. println(pDE_RNA_Formular. pdedsKey);
                return initonLink;
            }
            initonLink= initonLink. next; //while loop  替增.
        }
        //System. out. println(pDE_RNA_Formular. pdedsKey);
        return initonLink;
    }
}
```

//增元

package OSI. OSU. SI. SD. SU. SQ. ASU. OSU. PSU. MSU. AVQ. ASQ. tin. catalytic. procedure. pde;

import java. util. Iterator;

import java. util. List;

import java. util. concurrent. CopyOnWriteArrayList;

//注意: 该 文件对应的是罗瑶光先生 DNA 编码 与 计算的两本 国家软著作 思想的编码 实现.

//公安部 与 知识产权委员会 已经备案, 可阅读 相关 著作权 原文 进行逻辑辨别.

//作者: 罗瑶光

//思想: 罗瑶光

public class PDE_Increment_FullDNAFormular {

//A = V + S LINK 数据结构对象 (简单测试)

//这个函数我没有按 sonar 模式 修改, 因为我担心 sonar 会潜意识 改变我的写作模式,

//于是 按照我很多年的 ANSI C 代码风格进行编写, 因为这种方式是我的基础. 我个人觉

//得 一个人的研发能力来自于他的母语水平. 所以我是 C 基础, 我应该继续跟进这个天赋. 正如下面这函数.

```
public Initon PDE_IncrementA(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("V"))
            { if(initonLink. hasNext()) {
                Initon initonNext= initonLink. forwardNext();
                if(initonNext. getStore(). equalsIgnoreCase("S")) {
                    Initon initonIncrementA= new Initon();
                    initonIncrementA. setA(); //新增一个数据  A
                    if(initonNext. hasNext()) {
                        initonIncrementA. next= initonNext. next; //A 后序替换
                        initonIncrementA. next. prev= initonIncrementA; //A 后序前序恒等
                    }
                    if(null!= initonNext. prev. prev) {
                        initonIncrementA. prev= initonNext. prev. prev; //A 前序替换
                        initonIncrementA. prev. next= initonIncrementA; //A 前序后序恒等
                    }
                    initonLink= initonIncrementA; //最后  A 代替
                }
            }
        }
        if(!initonLink. hasNext())
            { return initonLink;
        }
        initonLink= initonLink. forwardNext(); //while loop  替增.
    }
    return initonLink;
}

//A = V + S LIST jdk util 对象, 下面函数是直接用JDK 的虚拟机函数  编写的, 逻辑比较清晰, 各有各的用
```

处.

```
public List<Initon> PDE_IncrementA(List<Initon> Initons)
    { List<Initon> output= new CopyOnWriteArrayList<>();
    Iterator<Initon> iterator= Initons. iterator();
    HERE:
        while(iterator. hasNext()) {
            Initon initon= iterator. next();
            if(initon. getStore(). equalsIgnoreCase("V"))
                { if(iterator. hasNext()) {
                    Initon initonNext= iterator. next();
                    if(initonNext. getStore(). equalsIgnoreCase("S"))
                        { Initon initonIncrementA= new Initon();
                        initonIncrementA. setA();
                        output. add(initonIncrementA);
                        continue HERE;
                    }
                    Initon initonIncrementV= new Initon();
                    initonIncrementV. setV();
                    Initon initonIncrementS= new Initon();
                    initonIncrementS. setS();
                    output. add(initonIncrementV);
                    output.  add(initonIncrementS);
                    continue HERE;
                }
                Initon initonIncrementV= new Initon();
                initonIncrementV. setV();
                output. add(initonIncrementV);
                continue HERE;
            }
            output. add(initon);
```

```
            }
        return output;
    }
    //O = E + S LINK 数据结构对象 (简单测试)
    public Initon PDE_IncrementO(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("E"))
                { if(initonLink. hasNext()) {
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("S")) {
                        Initon initonIncrementO= new Initon();
                        initonIncrementO. setO(); //新增一个数据 O
                        if(initonNext. hasNext()) {
                            initonIncrementO. next= initonNext. next; //O 后序替换
                            initonIncrementO. next. prev= initonIncrementO; //O 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementO. prev= initonNext. prev. prev; //O 前序替换
                            initonIncrementO. prev. next= initonIncrementO; //O 前序后序恒等
                        }
                        initonLink= initonIncrementO; //最后 O 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop 替增.
        }
```

```
        return initonLink;
    }
    //P = E + C LINK 数据结构对象 (简单测试)
    public Initon PDE_IncrementP(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("E"))
                { if(initonLink. hasNext()) {
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("C")) {
                        Initon initonIncrementP= new Initon();
                        initonIncrementP. setP(); //新增一个数据 P
                        if(initonNext. hasNext()) {
                            initonIncrementP. next= initonNext. next; //P 后序替换
                            initonIncrementP. next. prev= initonIncrementP; //P 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementP. prev= initonNext. prev. prev; //P 前序替换
                            initonIncrementP. prev. next= initonIncrementP; //P 前序后序恒等
                        }
                        initonLink= initonIncrementP; //最后 P 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop 替增.
        }
        return initonLink;
```

```
}
//M = C + S
public Initon PDE_IncrementM(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("C"))
            { if(initonLink. hasNext()) {
                Initon initonNext= initonLink. forwardNext();
                if(initonNext. getStore(). equalsIgnoreCase("S")) {
                    Initon initonIncrementM= new Initon();
                    initonIncrementM. setM(); //新增一个数据M
                    if(initonNext. hasNext()) {
                        initonIncrementM. next= initonNext. next; //M 后序替换
                        initonIncrementM. next. prev= initonIncrementM; //M 后序前序恒等
                    }
                    if(null!= initonNext. prev. prev) {
                        initonIncrementM. prev= initonNext. prev. prev; //M 前序替换
                        initonIncrementM. prev. next= initonIncrementM; //M 前序后序恒等
                    }
                    initonLink= initonIncrementM; //最后  M 代替
                }
            }
        }
        if(!initonLink. hasNext())
            { return initonLink;
        }
        initonLink= initonLink. forwardNext(); //while loop  替增.
    }
    return initonLink;
}
```

```
//V = U + Q
public Initon PDE_IncrementV(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("U"))
            { if(initonLink. hasNext()) {
                Initon initonNext= initonLink. forwardNext();
                if(initonNext. getStore(). equalsIgnoreCase("Q")) {
                    Initon initonIncrementV= new Initon();
                    initonIncrementV. setV(); //新增一个数据  V
                    if(initonNext. hasNext()) {
                        initonIncrementV. next= initonNext. next; //V 后序替换
                        initonIncrementV. next. prev= initonIncrementV; //V 后序前序恒等
                    }
                    if(null!= initonNext. prev. prev) {
                        initonIncrementV. prev= initonNext. prev. prev; //V 前序替换
                        initonIncrementV. prev. next= initonIncrementV; //V 前序后序恒等
                    }
                    initonLink= initonIncrementV; //最后  V 代替
                }
            }
        }
        if(!initonLink. hasNext())
            { return initonLink;
        }
        initonLink= initonLink. forwardNext(); //while loop  替增.
    }
    return initonLink;
}
```

```
    //E = D + U
    //肽展公式的概率问题, 先这样命名, 之后讨论 是 IU 还是 DU
    public Initon PDE_IncrementE_DU(InitonLinkDNA initonLinkDNA, FullDNATokenPDI pDE_RNA_Formular,
boolean bYS) {
        Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            String initonString= initonLink. getStore();
            if(initonLink. getStore(). equalsIgnoreCase("D")
                    || initonLink. getStore(). equalsIgnoreCase("U"))
            { Initon initonIncrementE= new Initon();
            initonIncrementE. setE(); // 新 增 一 个 数 据 S
            if(initonString. equalsIgnoreCase("D")) {
                pDE_RNA_Formular. pdeieKey+= "0";
            }else if(initonString. equalsIgnoreCase("U"))
                { pDE_RNA_Formular. pdeieKey+= "1";
            }
            if(initonLink. hasNext()) {
                initonIncrementE. next= initonLink. next; //S 后序替换
                initonIncrementE. next. prev= initonIncrementE; //S 后序前序恒等
            }
            if(null!= initonLink. prev) {
                initonIncrementE. prev= initonLink. prev; //S 前序替换
                initonIncrementE. prev. next= initonIncrementE; //S 前序后序恒等
            }
            initonLink= initonIncrementE; //最后 E 代替
        }
        if(!initonLink. hasNext())
            { return initonLink;
        }
        initonLink= initonLink. forwardNext(); //while loop 替增.
```

```
        }
        return initonLink;
    }

    //C = I + D
    public Initon PDE_IncrementC(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("I"))
                { if(initonLink. hasNext()) {
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("D")) {
                        Initon initonIncrementC= new Initon();
                        initonIncrementC. setC(); //新增一个数据  C
                        if(initonNext. hasNext()) {
                            initonIncrementC. next= initonNext. next; //C 后序替换
                            initonIncrementC. next. prev= initonIncrementC; //C 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementC. prev= initonNext. prev. prev; //M 前序替换
                            initonIncrementC. prev. next= initonIncrementC; //M 前序后序恒等
                        }
                        initonLink= initonIncrementC; //最后  C 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop  替增.
```

```
        }
        return initonLink;
    }

    //S = I
    //肽展公式的概率问题, 先这样命名, 之后讨论 是 I, Q 还是 I + Q
    public Initon PDE_IncrementS_IQ(InitonLinkDNA initonLinkDNA, FullDNATokenPDI pDE_RNA_Formular)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        //int index= 0;
        while(null!= initonLink) {
            String initonString= initonLink. getStore();
            if(initonLink. getStore(). equalsIgnoreCase("I")
                    || initonLink. getStore(). equalsIgnoreCase("Q"))
                { Initon initonIncrementS= new Initon();
                initonIncrementS. setS(); // 新 增 一 个 数 据 S
                if(initonString. equalsIgnoreCase("I")) {
                    pDE_RNA_Formular. pdeisKey+= "0";
                }else if(initonString. equalsIgnoreCase("Q"))
                    { pDE_RNA_Formular. pdeisKey+= "1";
                }
                if(initonLink. hasNext()) {
                    initonIncrementS. next= initonLink. next; //S 后序替换
                    initonIncrementS. next. prev= initonIncrementS; //S 后序前序恒等
                }
                if(null!= initonLink. prev) {
                    initonIncrementS. prev= initonLink. prev; //S 前序替换
                    initonIncrementS. prev. next= initonIncrementS; //S 前序后序恒等
                }
                initonLink= initonIncrementS; //最后 S 代替
```

```
                }
                if(!initonLink. hasNext())
                    { return initonLink;
                }
                initonLink= initonLink. forwardNext(); //while loop 替增.
            }
            return initonLink;
        }
}
```

//测试输出结果原文:

控制吸收

肽语: PEEOOUPOAVCUPOECAUPEDOAU

肽锁: SVMD

散 列 肽 语 : SVMDPSVMDESVMDESVMDOSVMDOSVMDUSVMDPSVMDOSVMDASVMDVSVMDCSVMDUS
VMDPSVMDOSVMDESVMDCSVMDASVMDUSVMDPSVMDESVMDDSVMDOSVMDASVMDU

静态密钥: 0. 6/0. 3/0. 5/0. 632

A->SVMDPSVMDESVMDESVMDOSVMDOSVMDUSVMDPSVMDOSVMDVSSVMDVSVMDCSVM
DUSVMDPSVMDOSVMDESVMDCSVMDVSSVMDUSVMDPSVMDESVMDDSVMDOSVMDVSSV MD
O->SVMDPSVMDESVMDESVMDESSVMDESSVMDUSVMDPSVMDESSVMDVSSVMDVSVMDCS
VMDUSVMDPSVMDESSVMDESVMDCSVMDVSSVMDUSVMDPSVMDESVMDDSVMDESSVMD VSSVMD
P->SVMDECSVMDESVMDESVMDESSVMDESSVMDUSVMDECSVMDESSVMDVSSVMDVSVMD
CSVMDUSVMDECSVMDESSVMDESVMDCSVMDVSSVMDUSVMDECSVMDESVMDDSVMDESS VMDVSSVMD
M->SVCSDECSVCSDESVCSDESVCSDESSVCSDESSVCSDUSVCSDECSVCSDESSVCSDVSSVCSD
VSVCSDCSVCSDUSVCSDECSVCSDESSVCSDESVCSDCSVCSDVSSVCSDUSVCSDECSVCSDESV

CSDDSVCSDESSVCSDVSSVCSD

V->SUQCSDECSUQCSDESUQCSDESUQCSDESSUQCSDESSUQCSDUSUQCSDECSUQCSDESSUQ
CSDUQSSUQCSDUQSUQCSDCSUQCSDUSUQCSDECSUQCSDESSUQCSDESUQCSDCSUQCSDUQ
SSUQCSDUSUQCSDECSUQCSDESUQCSDDSUQCSDESSUQCSDUQSSUQCSD
E->SUQCSDUCSUQCSDUSUQCSDUSUQCSDUSSUQCSDDSSUQCSDUSUQCSDUCSUQCSDUSSU
QCSDUQSSUQCSDUQSUQCSDCSUQCSDUSUQCSDDCSUQCSDUSSUQCSDUSUQCSDCSUQCSD
UQSSUQCSDUSUQCSDUCSUQCSDUSUQCSDDSUQCSDUSSUQCSDUQSSUQCSD
C->SUQIDSDUIDSUQIDSDUSUQIDSDUSUQIDSDUSSUQIDSDDSSUQIDSDUSUQIDSDUIDSUQID
SDUSSUQIDSDUQSSUQIDSDUQSUQIDSDIDSUQIDSDUSUQIDSDDIDSUQIDSDUSSUQIDSDUSU
QIDSDIDSUQIDSDUQSSUQIDSDUSUQIDSDUIDSUQIDSDUSUQIDSDDSUQIDSDUSSUQIDSDUQ
SSUQIDSD1111101111101111111101100110111010111011101111010111010111

S->QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQU
QIDQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQD
UQUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQI DIDUQQQUQIDQD

V->QVIDQDUIDQVIDQDVVIDIDVVIDQDVQVIDQDDIQVIDQDVVIDQDUIDQVIDQDVQVIDIDV
QQVIDIDVIVIDQDIDQVIDIDVVIDQDDIDQVIDIDVIVIDQDVVIDQDIDIVIDQDVQQVIDIDVVIDQ
DUIDQVIDQDUIVIDQDDQVIDQDUIQVIDIDVQQVIDQD
E->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
C->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
S->SVSESEESESVSESEVVSESEVVSESEVSVSESEESSVSESEVVSESEESESVSESEVSVSESEVSSV
SESEVSVSESESESVSESEVVSESEESESVSESEVSVSESEVVSESESESVSESEVSSVSESEVVSESEES
ESVSESEESVSESEESVSESEESSVSESEVSSVSESE
A->SAESEESESAESEVAESEVAESEAAESEESSAESEVAESEESESAESEAAESEASAESEAAESESES

AESEVAESEESESAESEAAESEVAESESESAESEASAESEVAESEESESAESEESAESEESAESEESSAE SEASAESE

O->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO

EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE

P->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOE

AAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE

M->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO

EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE静态肽展降元概率钥匙E: 1111011011111

静态肽展降元概率钥匙S: 11111011111101111111101100110111010111011101111010111010111

静 态 肽 展 降 元 : QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQUQI

DQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQDU

QUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQIDI DUQQQUQIDQDU

静态肽展增元概率钥匙E: 00100000000000000100000000000000000000000000000010001000001000001

静 态 肽 展 增 元 概 率 钥 匙 S:

10101010001101010101010110011000010100010100001010001110001010100110101001101

静 态 肽 展 增 元 : SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA

OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

时间: 1613543667507

账号随机缓存字符串: ID0. 1367567645799036: 0. 5311761996535497

Session: SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA

OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

==========

开始前序验证:

开 始 Session 解 析 : SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA

OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

开 始 概 率 钥 匙 解 析 ：

1111011011111111110111110111111111011001101110101110111011110111010111001000000000000001

0000000000000000000000000000001000100000100001101010100011010101010101100110000101000101

0000101000111000101010011010100 1101

A->SVMDPSVMDESVMDESVMDOSVMDOSVMDUSVMDPSVMDOSVMDVSSVMDVSVMDCSVM

DUSVMDPSVMDOSVMDESVMDCSVMDVSSVMDUSVMDPSVMDESVMDDSVMDOSVMDVSSV MD

O->SVMDPSVMDESVMDESVMDESSVMDESSVMDUSVMDPSVMDESSVMDVSSVMDVSVMDCS

VMDUSVMDPSVMDESSVMDESVMDCSVMDVSSVMDUSVMDPSVMDESVMDDSVMDESSVMD VSSVMD

P->SVMDECSVMDESVMDESVMDESSVMDESSVMDUSVMDECSVMDESSVMDVSSVMDVSVMD

CSVMDUSVMDECSVMDESSVMDESVMDCSVMDVSSVMDUSVMDECSVMDESVMDDSVMDESS VMDVSSVMD

M->SVCSDECSVCSDESVCSDESVCSDESSVCSDESSVCSDUSVCSDECSVCSDESSVCSDVSSVCSD

VSVCSDCSVCSDUSVCSDECSVCSDESSVCSDESVCSDCSVCSDVSSVCSDUSVCSDECSVCSDESV

CSDDSVCSDESSVCSDVSSVCSD

V->SUQCSDECSUQCSDESUQCSDESUQCSDESSUQCSDESSUQCSDUSUQCSDECSUQCSDESSUQ

CSDUQSSUQCSDUQSUQCSDCSUQCSDUSUQCSDECSUQCSDESSUQCSDESUQCSDCSUQCSDUQ

SSUQCSDUSUQCSDECSUQCSDESUQCSDDSUQCSDESSUQCSDUQSSUQCSD

E->SUQCSDUCSUQCSDUSUQCSDUSUQCSDUSSUQCSDDSSUQCSDUSUQCSDUCSUQCSDUSSU

QCSDUQSSUQCSDUQSUQCSDCSUQCSDUSUQCSDDCSUQCSDUSSUQCSDUSUQCSDCSUQCSD

UQSSUQCSDUSUQCSDUCSUQCSDUSUQCSDDSUQCSDUSSUQCSDUQSSUQCSD

C->SUQIDSDUIDSUQIDSDUSUQIDSDUSUQIDSDUSSUQIDSDDSSUQIDSDUSUQIDSDUIDSUQID

SDUSSUQIDSDUQSSUQIDSDUQSUQIDSDIDSUQIDSDUSUQIDSDDIDSUQIDSDUSSUQIDSDUSU

QIDSDIDSUQIDSDUQSSUQIDSDUSUQIDSDUIDSUQIDSDUSUQIDSDDSUQIDSDUSSUQIDSDUQ

SSUQIDSD1111101111101111111101100110111010111011101111010111010111

S->QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQU

QIDQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQD
UQUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQI DIDUQQQUQIDQD

V->QVIDQDUIDQVIDQDVVIDIDVVIDQDVQVIDQDDIQVIDQDVVIDQDUIDQVIDQDVQVIDIDV
QQVIDIDVIVIDQDIDQVIDIDVVIDQDDIDQVIDIDVIVIDQDVVIDQDIDIVIDQDVQQVIDIDVVIDQ
DUIDQVIDQDUIVIDQDDQVIDQDUIQVIDIDVQQVIDQD
E->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
C->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
S->SVSESEESESVSESEVVSESEVVSESEVSVSESEESSVSESEVVSESEESESVSESEVSVSESEVSSV
SESEVSVSESESESVSESEVVSESEESESVSESEVSVSESEVVSESESESVSESEVSSVSESEVVSESEES
ESVSESEESVSESEESVSESEESSVSESEVSSVSESE
A->SAESEESESAESEVAESEVAESEAAESEESSAESEVAESEESESAESEAAESEASAESEAAESESES
AESEVAESEESESAESEAAESEVAESESESAESEASAESEVAESEESESAESEESAESEESAESEESSAE SEASAESE
O->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO
EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
P->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOE
AAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
M->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO
EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE得到原降元元基DNA序列:
QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQUQI
DQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQDU
QUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQIDI DUQQQUQIDQDU
得到新降元元基DNA序列:

QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQUQI
DQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQDU
QUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQIDI DUQQQUQIDQDU

得到原元基DNA序列:

SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA
OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

得到新元基DNA序列:

SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA
OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

验证正确？

正确

==========

开始后序验证:

准 备 计 算 元 基 DNA 序 列 :

SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA
OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

M->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO
EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
P->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOE
AAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
O->SAESEESESAESEVAESEVAESEAAESEESSAESEVAESEESESAESEAAESEASAESEAAESESES
AESEVAESEESESAESEAAESEVAESESESAESEASAESEVAESEESESAESEESAESEESAESEESSAE SEASAESE
A->SVSESEESESVSESEVVSESEVVSESEVSVSESEESSVSESEVVSESEESESVSESEVSVSESEVSSV
SESEVSVSESESESVSESEVVSESEESESVSESEVSVSESEVVSESESESVSESEVSSVSESEVVSESEES
ESVSESEESVSESEESVSESEESSVSESEVSSVSESE10101010001101010101010110011000010100010 1
0000101000111000101010011010100 1101

S->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE

C->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
E->QVIDQDUIDQVIDQDVVIDIDVVIDQDVQVIDQDDIQVIDQDVVIDQDUIDQVIDQDVQVIDIDV
QQVIDIDVIVIDQDIDQVIDIDVVIDQDDIDQVIDIDVIVIDQDVVIDQDIDIVIDQDVQQVIDIDVVIDQ
DUIDQVIDQDUIVIDQDDQVIDQDUIQVIDIDVQQVIDQD
V->QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQU
QIDQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQD
UQUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQI
DIDUQQQUQIDQD得到原续降元元基DNA序列:
QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQUQI
DQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQDU
QUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQIDI DUQQQUQIDQDU
得到后续降元元基DNA序列:
QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQUQI
DQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQDU
QUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQIDI DUQQQUQIDQDU
验证正确？
正确
==========
开始整序验证:
准 备 计 算 元 基 DNA 序 列 :
SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA
OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

M->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO
EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
P->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOE
AAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
O->SAESEESESAESEVAESEVAESEAAESEESSAESEVAESEESESAESEAAESEASAESEAAESESES
AESEVAESEESESAESEAAESEVAESESESAESEASAESEVAESEESESAESEESAESEESAESEESSAE SEASAESE
A->SVSESEESESVSESEVVSESEVVSESEVSVSESEESSVSESEVVSESEESESVSESEVSVSESEVSSV
SESEVSVSESESESVSESEVVSESEESESVSESEVSVSESEVVSESESESVSESEVSSVSESEVVSESEES
ESVSESEESVSESEESVSESEESSVSESEVSSVSESE10101010001101010101010110011000010100010 1
000010100011100010101001101010011 01

S->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE

C->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
E->QVIDQDUIDQVIDQDVVIDIDVVIDQDVQVIDQDDIQVIDQDVVIDQDUIDQVIDQDVQVIDIDV
QQVIDIDVIVIDQDIDQVIDIDVVIDQDDIDQVIDIDVIVIDQDVVIDQDIDIVIDQDVQQVIDIDVVIDQ
DUIDQVIDQDUIVIDQDDQVIDQDUIQVIDIDVQQVIDQD
V->QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQU
QIDQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQD
UQUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQI DIDUQQQUQIDQD
V->QVIDQDUIDQVIDQDVVIDIDVVIDQDVQVIDQDDIQVIDQDVVIDQDUIDQVIDQDVQVIDIDV
QQVIDIDVIVIDQDIDQVIDIDVVIDQDDIDQVIDIDVIVIDQDVVIDQDIDIVIDQDVQQVIDIDVVIDQ
DUIDQVIDQDUIVIDQDDQVIDQDUIQVIDIDVQQVIDQD

E->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
C->QVIEQEEIEQVIEQEVVIEIEVVIEQEVQVIEQEEIQVIEQEVVIEQEEIEQVIEQEVQVIEIEVQQVIE
IEVIVIEQEIEQVIEIEVVIEQEEIEQVIEIEVIVIEQEVVIEQEIEIVIEQEVQQVIEIEVVIEQEEIEQVIEQ
EEIVIEQEEQVIEQEEIQVIEIEVQQVIEQE
S->SVSESEESESVSESEVVSESEVVSESEVSVSESEESSVSESEVVSESEESESVSESEVSVSESEVSSV
SESEVSVSESESESVSESEVVSESEESESVSESEVSVSESEVVSESESESVSESEVSSVSESEVVSESEES
ESVSESEESVSESEESVSESEESSVSESEVSSVSESE
A->SAESEESESAESEVAESEVAESEAAESEESSAESEVAESEESESAESEAAESEASAESEAAESESES
AESEVAESEESESAESEAAESEVAESESESAESEASAESEVAESEESESAESEESAESEESAESEESSAE SEASAESE
O->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO
EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
P->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOE
AAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE
M->SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAO
EAAOEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOE得到原续降元元基DNA序列:
QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQUQI
DQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQDU
QUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQIDI DUQQQUQIDQDU
得到后续降元元基DNA序列:
QUQIDQDUIDQUQIDQDUQUQIDIDUQUQIDQDUQQUQIDQDDIQUQIDQDUQUQIDQDUIDQUQI
DQDUQQUQIDIDUQQQUQIDIDUQIUQIDQDIDQUQIDIDUQUQIDQDDIDQUQIDIDUQIUQIDQDU
QUQIDQDIDIUQIDQDUQQQUQIDIDUQUQIDQDUIDQUQIDQDUIUQIDQDDQUQIDQDUIQUQIDI DUQQQUQIDQDU
验证正确？
正确
准备整序计算元基DNA序列:

SVMDPSVMDESVMDESVMDOSVMDOSVMDUSVMDPSVMDOSVMDASVMDVSVMDCSVMDUS

VMDPSVMDOSVMDESVMDCSVMDASVMDUSVMDPSVMDESVMDDSVMDOSVMDASVMDU

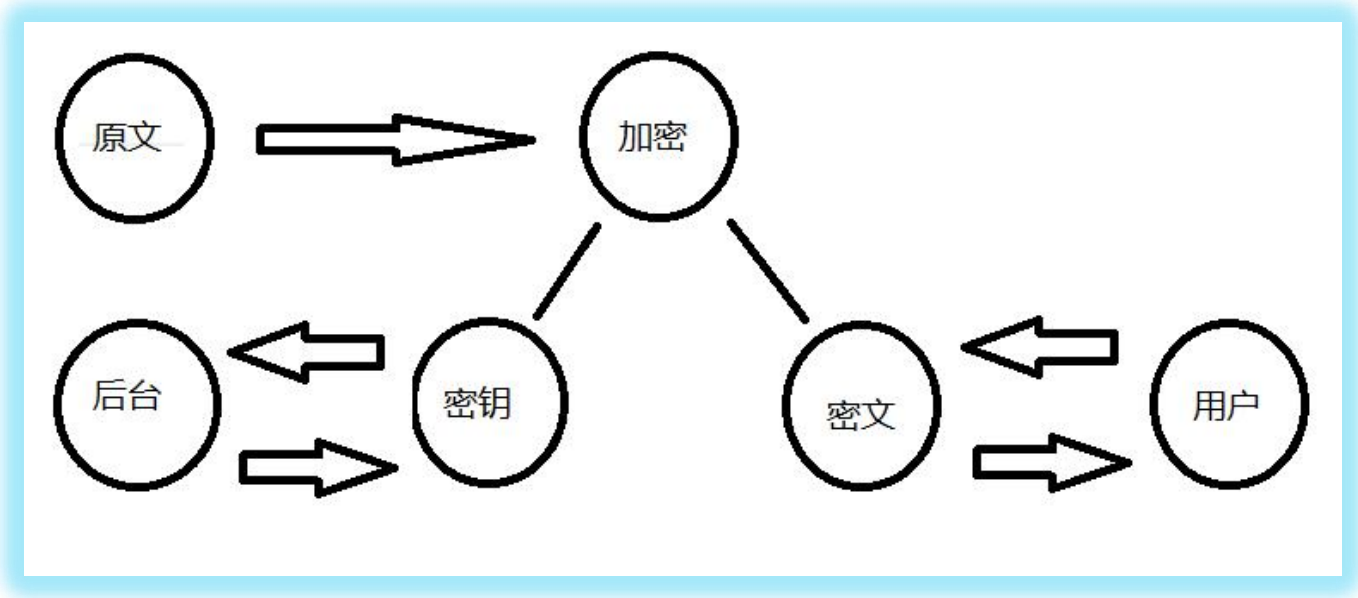
准备整序计算元基DNA序列:

SAOEOOAOEVAOEVAOEAAOEOSAOEVAOEOOAOEAAOEASAOEAAOOOAOEVAOEOOAOEAA

OEVAOOOAOEASAOEVAOEOOAOEOAOEOAOEOSAOEASAOEE

失败

第三节 DNA 加密的具体描述



如上图先将待加密的文字进行元基化, 然后进行肽展概率加密, 加密后保留概率记录的数字串, 然后把加密后的密文给用户做访问标识.

定义: 加密钥匙的逻辑: 将元基根据肽展公式的E 和S 变换成 DU 和 IQ 比率描述进行 0 和 1 记录, 这个记录串作为钥匙, 而概率变换后的元基串则为密文. 给用户端. 起到严谨的非对称加密效果.

第四节 DNA 加密的应用实现

```
//DNA token 生成调用
package org. plsql. common. utils;

import java. util. Date;
import MD5Processor. Token;
import MD5Processor. Usr;
import MD5Processor. UsrToken;
public class TokenUtil {
    //引用了中科大的筛子非对称加密论文思想, 这里标注下. ~20180701
    //今天已经全部替换为罗瑶光先生的肽元基计算加密思想. 20200103~.
    public static Token getNewTokenFromUsrAndUsrToken(Usr usr, UsrToken usrToken) throws Exception {
        //String key = String. valueOf(Double. valueOf(Math. random() * 10000000). intValue());
        //String mPassword = TokenUtil. getFirstMD5Password(key, usrToken. getuPassword());
        //替换如下:
        String key= "";
        String[] lock= new String[12];
  lock[0] = "A"; lock[3] = "O"; lock[6] = "P"; lock[9] = "M";
  lock[1] = "V"; lock[4] = "E"; lock[7] = "C"; lock[10] = "S";
  lock[2] = "I"; lock[5] = "D"; lock[8] = "U"; lock[11] = "Q";
  for(int loop= 0; loop< 4; loop++) {
     int i= (int)(Math. random()* 12)% 12;
    key+= lock[i];
  }
  Token token = new Token();
        String dnaPassword = TokenUtil. getFirstDNAPassword(key, usrToken. getuPassword(), token);
        token. setuEmail(usr. getuEmail());
        token. setuKey(key);
        token. setuTime(new Date(). getTime());
```

```
        //token.  setmPassword(mPassword);
        token. setmPassword(dnaPassword);
        return token;
    }
    public static String getSecondMD5Password(String uPassword) throws Exception
        { return StringUtil. EncoderByMd5("Author: Yaoguang Luo", uPassword, 8);
    }
    public static String getFirstMD5Password(String key, String uPassword) throws Exception
        { return StringUtil. EncoderByMd5(key, uPassword, 8);
    }
    public static String getFirstDNAPassword(String key, String uPassword, Token token)
        { return StringUtil. EncoderByDNA(key, uPassword, token);
    }
}

//DNA token 加密编码
public static String EncoderByDNA(String key, String uPassword, Token token)
        { TokenPDI pDE_RNA_Formular= new TokenPDI();
        @SuppressWarnings("unused")
        String initonKeys= "EIU/0. 6/EDU/0. 4/si/0. 3/sq/0. 7/EIU/0. 5/EDU/0. 5/si/0. 632/sq/0. 368";
        pDE_RNA_Formular. key[0]= 0. 6;
        pDE_RNA_Formular. key[1]= 0. 3;
        pDE_RNA_Formular. key[2]= 0. 5;
        pDE_RNA_Formular. key[3]= 0. 632;

        pDE_RNA_Formular. text= uPassword;
        System. out. println(" 原 文 : " + pDE_RNA_Formular. text);
        pDE_RNA_Formular. pdw= pDE_RNA_Formular. initonSect(key);
        for(int i= 0; i< pDE_RNA_Formular. pdw. length(); i++) {
            pDE_RNA_Formular. code+= pDE_RNA_Formular. lock + pDE_RNA_Formular. pdw. charAt(i);
```

```
        }
        System. out. println("肽语: " + pDE_RNA_Formular. pdw);
        System. out. println("肽锁: " + key);
        System. out. println(" 散 列 肽 语 : 保 密 " );
        pDE_RNA_Formular. bys= "0. 6/0. 3/0. 5/0. 632";
        System. out. println("静态密钥: " + pDE_RNA_Formular. bys);
        pDE_RNA_Formular. doKeyPress(pDE_RNA_Formular. code, pDE_RNA_Formular, false);
        token. setUpdsde( pDE_RNA_Formular. pdedeKey);
        token. setUpdsds( pDE_RNA_Formular. pdedsKey);
        token. setUpdsie( pDE_RNA_Formular. pdeieKey);
        token. setUpdsis( pDE_RNA_Formular. pdeisKey);
        pDE_RNA_Formular. time= "" + System. currentTimeMillis();
        pDE_RNA_Formular. cacheId= "ID" + Math. random() + ": " + Math. random();
        System. out. println("时间: " + pDE_RNA_Formular. time);
        System. out. println("账号随机缓存字符串: " + pDE_RNA_Formular. cacheId);
        pDE_RNA_Formular. session_key= pDE_RNA_Formular. pde;
        System. out. println("Session: " + pDE_RNA_Formular. session_key);
        return pDE_RNA_Formular. session_key;
    }
```

目前上面这段加密函数在养疗经[17]中使用非常稳定, 其逻辑结构已经具体应用于养疗经[17]的加密, 解密, Rest Token 组件中. 关于之后的走向, 我会让它逐渐轻量级优化, 轻巧, 高效, 迅捷.

```
//TOKENPDI 的语义解析函数
package OSI. OSU. SI. SD. SU. SQ. ASU. OSU. PSU. MSU. AVQ. ASQ. tin. catalytic. procedure. pde;

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

//注意: 该 文件对应的是罗瑶光先生 DNA 编码 与 计算的两本 国家软著作 思想的编码 实现.
//公安部 与 知识产权委员会 已经备案, 可阅读 相关 著作权 原文 进行逻辑辨别.
public class TokenPDI {
    public double[] key= new double[4];
    public String bys= "";
    public String pdw= "";
    public String pds= "";
    public String pde= "";
    public String time= "";
    public String session_key= "";
    public String text;
    public String cacheId;
    public String code= "";
    public String lock= "";
    public String pdedeKey= "";
    public String pdedsKey= "";
    public String pdeieKey= "";
    public String pdeisKey= "";

    public void doKeyPress(String initons, TokenPDI pDE_RNA_Formular, boolean bYS)
        { Initon[] initon= new Initon[initons. length()];
        for(int i= 0; i< initons. length(); i++)
            { if(initon[i]== null) {
```

```
                initon[i]= new Initon();
            }
            initon[i]. setIniton(""+ initons. charAt(i));
            if(i+ 1< initons. length()) {
                if(initon[i+ 1]== null)
                    { initon[i+ 1]= new
                    Initon();
                }
                initon[i]. next= initon[i+ 1];
                initon[i+ 1]. prev= initon[i];
            }
        }
        if(null!= initons&& initons. length()> 0)
            { do_PDE_RNA_Formular(initon[0], pDE_RNA_Formular,
            bYS);
        }
    }

    public static void main(String[] argv) {
        TokenPDI pDE_RNA_Formular= new TokenPDI();
        @SuppressWarnings("unused")
        String initonKeys= "EIU/0. 6/EDU/0. 4/si/0. 3/sq/0. 7/EIU/0. 5/EDU/0. 5/si/0. 632/sq/0. 368";
        pDE_RNA_Formular. key[0]= 0. 6;
        pDE_RNA_Formular. key[1]= 0. 3;
        pDE_RNA_Formular. key[2]= 0. 5;
        pDE_RNA_Formular. key[3]= 0. 632;
        pDE_RNA_Formular. text= "控制吸收";
        pDE_RNA_Formular. pdw= pDE_RNA_Formular. initonSect(pDE_RNA_Formular. text);
        System. out. println("原文: " + pDE_RNA_Formular. text);
        //pDE_RNA_Formular. pdw= "字典保密: MSIOCUOCIPCUPCI";
        String[] lock= new String[12];
lock[0] = "A"; lock[3] = "O"; lock[6] = "P"; lock[9] = "M";
```

```
lock[1] = "V"; lock[4] = "E"; lock[7] = "C"; lock[10] = "S";
lock[2] = "I"; lock[5] = "D"; lock[8] = "U"; lock[11] = "Q";
int i= (int)(Math. random()* 12)% 12;
pDE_RNA_Formular. lock+= lock[i];
i= (int)(Math. random()* 12)% 12;
pDE_RNA_Formular. lock+= lock[i];
i= (int)(Math. random()* 12)% 12;
pDE_RNA_Formular. lock+= lock[i];
i= (int)(Math. random()* 12)% 12;
pDE_RNA_Formular. lock+= lock[i];

        for(i= 0; i< pDE_RNA_Formular. pdw. length(); i++) {
            pDE_RNA_Formular. code+= pDE_RNA_Formular. lock + pDE_RNA_Formular. pdw. charAt(i);
        }
        System. out. println("肽语: " + pDE_RNA_Formular. pdw);
        System. out. println("肽锁: " + pDE_RNA_Formular. lock);
        System. out. println(" 散 列 肽 语 : 保 密 " );
        pDE_RNA_Formular. bys= "0. 6/0. 3/0. 5/0. 632";
        System. out. println("静态密钥: " + pDE_RNA_Formular. bys);
        pDE_RNA_Formular. doKeyPress(pDE_RNA_Formular. code, pDE_RNA_Formular, false);
        System. out. println("静态肽展降元概率钥匙  E: "+ pDE_RNA_Formular. pdedeKey);
        System. out. println("静态肽展降元概率钥匙  S: "+ pDE_RNA_Formular. pdedsKey);
        System. out. println("静态肽展降元: "+ pDE_RNA_Formular. pds);

        System. out. println("静态肽展增元概率钥匙  E: "+ pDE_RNA_Formular. pdeieKey);
        System. out. println("静态肽展增元概率钥匙  S: "+ pDE_RNA_Formular. pdeisKey);
        System. out. println("静态肽展增元: "+ pDE_RNA_Formular. pde);

        pDE_RNA_Formular. time= "" + System. currentTimeMillis();
        pDE_RNA_Formular. cacheId= "ID" + Math. random() + ": " + Math. random();
```

```
        System. out. println("时间: " + pDE_RNA_Formular. time);
        System. out. println("账号随机缓存字符串: " + pDE_RNA_Formular. cacheId);
        pDE_RNA_Formular. session_key= pDE_RNA_Formular. pde;
        System. out. println("Session: " + pDE_RNA_Formular. session_key);
        System. out.
println("================================================================================
==========");
        System. out. println("开始前序验证: ");
        System. out. println("开始  Session 解析: " + pDE_RNA_Formular. session_key);
        System. out. println("开始概率钥匙解析: " + pDE_RNA_Formular. pdedeKey+ pDE_RNA_Formular.
pdedsKey+ pDE_RNA_Formular. pdeieKey+ pDE_RNA_Formular. pdeisKey);

        TokenPDI pDE_RNA_Formular1= new TokenPDI();
        pDE_RNA_Formular1. pdedeKey= pDE_RNA_Formular. pdedeKey. toString();
        pDE_RNA_Formular1. pdedsKey= pDE_RNA_Formular. pdedsKey. toString();
        pDE_RNA_Formular1. pdeieKey= pDE_RNA_Formular. pdeieKey. toString();
        pDE_RNA_Formular1. pdeisKey= pDE_RNA_Formular. pdeisKey. toString();

        pDE_RNA_Formular. doKeyUnPress(pDE_RNA_Formular. code, pDE_RNA_Formular1, true);
        System. out. println("得到原降元元基  DNA 序列: "+ pDE_RNA_Formular. pds);
        System. out. println("得到新降元元基  DNA 序列: "+ pDE_RNA_Formular1. pds);
        System. out. println("得到原元基  DNA 序列: "+ pDE_RNA_Formular. pde);
        System. out. println("得到新元基  DNA 序列: "+ pDE_RNA_Formular1. pde);
        System. out. println("验证正确？ ");
        System. out. println(pDE_RNA_Formular. pde. equals(pDE_RNA_Formular1. pde)? "正确": "失败");
        System. out.
println("============================================================");
        System. out. println("开始后序验证: ");
        TokenPDI pDE_RNA_Formular2= new TokenPDI();
        pDE_RNA_Formular2. pdeieKey= pDE_RNA_Formular. pdedeKey. toString();
```

```
        pDE_RNA_Formular2. pdeisKey= pDE_RNA_Formular. pdedsKey. toString();
        pDE_RNA_Formular2. pdedeKey= pDE_RNA_Formular. pdeieKey. toString();
        pDE_RNA_Formular2. pdedsKey= pDE_RNA_Formular. pdeisKey. toString();
        System. out. println("准备计算元基 DNA 序列: "+ pDE_RNA_Formular1. pde);
        pDE_RNA_Formular2. doSessionKeyUnPress(pDE_RNA_Formular1. pde, pDE_RNA_Formular2, true);
        System. out. println("得到原续降元元基 DNA 序列: "+ pDE_RNA_Formular1. pds);
        System. out. println("得到后续降元元基 DNA 序列: "+ pDE_RNA_Formular2. pds);
        System. out. println("验证正确？ ");
        System. out. println(pDE_RNA_Formular1. pds. equals(pDE_RNA_Formular2. pds)? "正确": "失败");

    }

    private void doSessionKeyUnPress(String initons, TokenPDI pDE_RNA_Formular2, boolean bYS)
        { Initon[] initon= new Initon[initons. length()];
        for(int i= 0; i< initons. length(); i++)
            { if(initon[i]== null) {
                initon[i]= new Initon();
            }
            initon[i]. setIniton(""+ initons. charAt(i));
            if(i+ 1< initons. length()) {
                if(initon[i+ 1]== null)
                    { initon[i+ 1]= new
                    Initon();
                }
                initon[i]. next= initon[i+ 1];
                initon[i+ 1]. prev= initon[i];
            }
        }
        do_PDE_RNA_Formular_Back(initon[0], pDE_RNA_Formular2, bYS);
    }
```

```
    private void do_PDE_RNA_Formular_Back(Initon initon, TokenPDI pDE_RNA_Formular, boolean bYS)
        { Initon InitonPDE= initon;
        InitonLinkDNA initonLinkDNA= new InitonLinkDNA();
        Initon InitonPDEM= doDecrementM(InitonPDE, initonLinkDNA, pDE_RNA_Formular);
        Initon InitonPDEP= doDecrementP(InitonPDEM, initonLinkDNA, pDE_RNA_Formular);
        Initon InitonPDEO= doDecrementO(InitonPDEP, initonLinkDNA, pDE_RNA_Formular);
        Initon InitonPDEA= doDecrementA(InitonPDEO, initonLinkDNA, pDE_RNA_Formular);
        Initon InitonPDES= doDecrementS(InitonPDEA, initonLinkDNA, pDE_RNA_Formular, bYS);
        Initon InitonPDEC= doDecrementC(InitonPDES, initonLinkDNA, pDE_RNA_Formular);
        Initon InitonPDEE= doDecrementE(InitonPDEC, initonLinkDNA, pDE_RNA_Formular, bYS);
        Initon InitonPDE1= doDecrementV(InitonPDEE, initonLinkDNA, pDE_RNA_Formular);
        while(InitonPDE1. hasNext()) {
            pDE_RNA_Formular. pds+= InitonPDE1. getStore();
            InitonPDE1= InitonPDE1. next;
        }
        pDE_RNA_Formular. pds+= InitonPDE1. getStore();
        while(InitonPDE1. hasPrev()) {
            InitonPDE1= InitonPDE1. prev;
        }

//      Initon InitonPDE1S= doIncrementS(InitonPDE1, initonLinkDNA, pDE_RNA_Formular, bYS);
//      Initon InitonPDE1C= doIncrementC(InitonPDE1S, initonLinkDNA, pDE_RNA_Formular);
//      Initon InitonPDE1E= doIncrementE(InitonPDE1C, initonLinkDNA, pDE_RNA_Formular, bYS);
//      Initon InitonPDE1V= doIncrementV(InitonPDE1E, initonLinkDNA, pDE_RNA_Formular);
//      Initon InitonPDE1M= doIncrementM(InitonPDE1V, initonLinkDNA, pDE_RNA_Formular);
//      Initon InitonPDE1P= doIncrementP(InitonPDE1M, initonLinkDNA, pDE_RNA_Formular);
//      Initon InitonPDE1O= doIncrementO(InitonPDE1P, initonLinkDNA, pDE_RNA_Formular);
//      Initon InitonPDE2= doIncrementA(InitonPDE1O, initonLinkDNA, pDE_RNA_Formular);
//      while(InitonPDE2. hasNext()) {
```

```
//                pDE_RNA_Formular. pde+= InitonPDE2. getStore(); InitonPDE2=
//                InitonPDE2. next;
//          }
//          pDE_RNA_Formular. pde+= InitonPDE2. getStore();
//          while(InitonPDE2. hasPrev()) {
//                InitonPDE2= InitonPDE2. prev;
//          }
      }

      private void doKeyUnPress(String initons, TokenPDI pDE_RNA_Formular, boolean bYS)
            { Initon[] initon= new Initon[initons. length()];
            for(int i= 0; i< initons. length(); i++)
                  { if(initon[i]== null) {
                        initon[i]= new Initon();
                  }
                  initon[i]. setIniton(""+ initons. charAt(i));
                  if(i+ 1< initons. length()) {
                        if(initon[i+ 1]== null)
                              { initon[i+ 1]= new
                              Initon();
                        }
                        initon[i]. next= initon[i+ 1];
                        initon[i+ 1]. prev= initon[i];
                  }
            }
            do_PDE_RNA_Formular(initon[0], pDE_RNA_Formular, bYS);
      }
      public void do_PDE_RNA_Formular(Initon initon, TokenPDI pDE_RNA_Formular, boolean bYS)
            { Initon InitonPDE= initon;
            InitonLinkDNA initonLinkDNA= new InitonLinkDNA();
            Initon InitonPDEA= doDecrementA(InitonPDE, initonLinkDNA, pDE_RNA_Formular);
```

```
Initon InitonPDEO= doDecrementO(InitonPDEA, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDEP= doDecrementP(InitonPDEO, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDEM= doDecrementM(InitonPDEP, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDEV= doDecrementV(InitonPDEM, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDEE= doDecrementE(InitonPDEV, initonLinkDNA, pDE_RNA_Formular, bYS);
Initon InitonPDEC= doDecrementC(InitonPDEE, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDE1= doDecrementS(InitonPDEC, initonLinkDNA, pDE_RNA_Formular, bYS);
while(InitonPDE1. hasNext()) {
    pDE_RNA_Formular. pds+= InitonPDE1. getStore();
    InitonPDE1= InitonPDE1. next;
}
pDE_RNA_Formular. pds+= InitonPDE1. getStore();
while(InitonPDE1. hasPrev()) {
    InitonPDE1= InitonPDE1. prev;
}
Initon InitonPDE1V= doIncrementV(InitonPDE1, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDE1E= doIncrementE(InitonPDE1V, initonLinkDNA, pDE_RNA_Formular, bYS);
Initon InitonPDE1C= doIncrementC(InitonPDE1E, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDE1S= doIncrementS(InitonPDE1C, initonLinkDNA, pDE_RNA_Formular, bYS);
Initon InitonPDE1A= doIncrementA(InitonPDE1S, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDE1O= doIncrementO(InitonPDE1A, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDE1P= doIncrementP(InitonPDE1O, initonLinkDNA, pDE_RNA_Formular);
Initon InitonPDE2= doIncrementM(InitonPDE1P, initonLinkDNA, pDE_RNA_Formular);
while(InitonPDE2. hasNext()) {
    pDE_RNA_Formular. pde+= InitonPDE2. getStore();
    InitonPDE2= InitonPDE2. next;
}
pDE_RNA_Formular. pde+= InitonPDE2. getStore();
while(InitonPDE2. hasPrev()) {
    InitonPDE2= InitonPDE2. prev;
```

```
        }
    }

/////INITONS SWAP

    private Initon doIncrementA(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementA(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementO(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
```

```
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementO(initonLinkDNA);
```

```
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementP(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular)
{
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementP(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
```

```
            prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementM(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
```

```
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementM(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementV(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementV(initonLinkDNA);
```

```
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
```

```
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementC(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular)
{
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementC(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
```

```
    }
    private Initon doIncrementE(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular,
boolean bYS) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementE_IU(initonLinkDNA, pDE_RNA_Formular,
bYS);
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
```

```
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doIncrementS(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular,
boolean bYS) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Increment_Formular(). PDE_IncrementS_I(initonLinkDNA, pDE_RNA_Formular);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
```

```
    private Initon doDecrementA(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementA(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }

    private Initon doDecrementO(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementO(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
```

```
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementP(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementP(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementM(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
```

```
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
```

```
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementM(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementV(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementV(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
```

```
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
```

```
        return InitonPDE;
    }
    private Initon doDecrementC(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementC(initonLinkDNA);
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementE(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular, boolean bYS) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
```

```
            prev;
        }
        initonLinkDNA. setInitonLink(InitonPDE);
        InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementE_IU(initonLinkDNA,
pDE_RNA_Formular, bYS);
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
```

```
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
        }
        return InitonPDE;
    }
    private Initon doDecrementS(Initon InitonPDE, InitonLinkDNA initonLinkDNA, TokenPDI
pDE_RNA_Formular, boolean bYS) {
        while(InitonPDE. hasPrev())
            { InitonPDE= InitonPDE.
            prev;
```

```
bYS);
}
initonLinkDNA. setInitonLink(InitonPDE);
InitonPDE= new PDE_Decrement_Formular(). PDE_DecrementS_I(initonLinkDNA, pDE_RNA_Formular,

while(InitonPDE. hasPrev()) { InitonPDE=
    InitonPDE. prev;
        }
        while(InitonPDE. hasNext())
            { InitonPDE= InitonPDE.
            next;
        }
        System. out. println();
        while(InitonPDE. hasPrev()) {
            InitonPDE= InitonPDE. prev;
        }
        return InitonPDE;
    }
```

```
    public String initonSect(String key)
        { String pdis= "";
        Map<String, String> map= new HashMap<>();
        map. put("0", "A");
        map. put("1", "O");
        map. put("2", "P");
        map. put("3", "M");
        map. put("4", "V");
        map. put("5", "E");
        map. put("6", "C");
        map. put("7", "S");
        map. put("8", "I");
        map. put("9", "D");
```

```
        map. put(". ", "U");
        for(int i= 0; i< key. length(); i++)
            { int c= key. charAt(i);
            String ints= ""+c;
            for(int j= 0; j< ints. length(); j++)
                { pdis+=map. get(""+ ints.
                charAt(j));
            }
            pdis+="U";
        }
        return pdis;
    }
}
//tokenpdi 的 pde 降元
//注意: 该 文件对应的是罗瑶光先生 DNA 编码 与 计算的两本 国家软著作 思想的编码 实现.
//公安部 与 知识产权委员会 已经备案, 可阅读 相关 著作权 原文 进行逻辑辨别.
public class PDE_Decrement_Formular {
    //A = V + S LINK 数据结构对象 (简单测试)
-------------------------------------------------------------------------------
    //作用发现有A 的initon进行VPCS级展开. 按我在 印度 ANSI C代码风格进行编写方式.
    public Initon PDE_DecrementA(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("A"))
                { Initon initonIncrementV= new Initon();
                initonIncrementV. setV(); //新增一个数据V
                Initon initonIncrementS= new Initon();
                initonIncrementS. setS(); //新增一个数据S
                initonIncrementV. next= initonIncrementS; //V初始
                initonIncrementV. prev= initonLink. prev;
                if(null!= initonIncrementV. prev) { initonIncrementV.
                    prev. next= initonIncrementV;
                }
                initonIncrementS. prev= initonIncrementV; //S初始
```

```
                initonIncrementS. next= initonLink. next;
                if(null!= initonIncrementS. next) { initonIncrementS.
                    next. prev= initonIncrementS;
                }
                initonLink= initonIncrementS; //最后S代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop 替增.
        }
        return initonLink;
    }

    //O = E + S LINK 数据结构对象  (简单测试)
-----------------------------------------------------------------------
    public Initon PDE_DecrementO(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("O"))
                { Initon initonIncrementE= new Initon();
                initonIncrementE. setE(); //新增一个数据E
                Initon initonIncrementS= new Initon();
                initonIncrementS. setS(); //新增一个数据S

                initonIncrementE. next= initonIncrementS; //E初始
                initonIncrementE. prev= initonLink. prev;
                if(null!= initonIncrementE. prev) { initonIncrementE.
                    prev. next= initonIncrementE;
                }
                initonIncrementS. prev= initonIncrementE; //S初始
                initonIncrementS. next= initonLink. next;
                if(null!= initonIncrementS. next) { initonIncrementS.
                    next. prev= initonIncrementS;
```

```
                }
                initonLink= initonIncrementS; //最后S代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop 替增.
        }
        return initonLink;
    }
    //P = E + C LINK 数据结构对象 (简单测试)
    public Initon PDE_DecrementP(InitonLinkDNA initonLinkDNA) {
---------------------------------------------------------------------------
        Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("P"))
                { Initon initonIncrementE= new Initon();
                initonIncrementE. setE(); //新增一个数据E
                Initon initonIncrementC= new Initon();
                initonIncrementC. setC(); //新增一个数据C

                initonIncrementE. next= initonIncrementC; //E初始
                initonIncrementE. prev= initonLink. prev;
                if(null!= initonIncrementE. prev) { initonIncrementE.
                    prev. next= initonIncrementE;
                }
                initonIncrementC. prev= initonIncrementE; //C初始
                initonIncrementC. next= initonLink. next;
                if(null!= initonIncrementC. next) { initonIncrementC.
                    next. prev= initonIncrementC;
                }
                initonLink= initonIncrementC; //最后C代替
            }
            if(!initonLink. hasNext()) {
```

```
            return initonLink;
        }
        initonLink= initonLink. next; //while loop 替增.
    }
    return initonLink;
}
//M = C + S
public Initon PDE_DecrementM(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
```

```
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("M"))
            { Initon initonIncrementC= new Initon();
            initonIncrementC. setC(); //新增一个数据C
            Initon initonIncrementS= new Initon();
            initonIncrementS. setS(); //新增一个数据S

            initonIncrementC. next= initonIncrementS; //C初始
            initonIncrementC. prev= initonLink. prev;
            if(null!= initonIncrementC. prev) { initonIncrementC.
                prev. next= initonIncrementC;
            }
            initonIncrementS. prev= initonIncrementC; //s初始
            initonIncrementS. next= initonLink. next;
            if(null!= initonIncrementS. next) { initonIncrementS.
                next. prev= initonIncrementS;
            }
            initonLink= initonIncrementS; //最后S代替
        }
        if(!initonLink. hasNext()) {
            return initonLink;
        }
        initonLink= initonLink. next; //while loop  替增.
```

```
        }
        return initonLink;
    }
    //V = U + Q
    public Initon PDE_DecrementV(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
```

```
            if(initonLink. getStore(). equalsIgnoreCase("V"))
                { Initon initonIncrementU= new Initon();
                initonIncrementU. setU(); //新增一个数据U
                Initon initonIncrementQ= new Initon();
                initonIncrementQ. setQ(); //新增一个数据Q

                initonIncrementU. next= initonIncrementQ; //U初始
                initonIncrementU. prev= initonLink. prev;
                if(null!= initonIncrementU. prev) { initonIncrementU.
                    prev. next= initonIncrementU;
                }
                initonIncrementQ. prev= initonIncrementU; //Q初始
                initonIncrementQ. next= initonLink. next;
                if(null!= initonIncrementQ. next) { initonIncrementQ.
                    next. prev= initonIncrementQ;
                }
                initonLink= initonIncrementQ; //最后Q代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop 替增.
        }
        return initonLink;
    }
```

```
    //E = I + U
    //肽展公式的概率问题, 先这样命名, 之后讨论  是IU还是DU
    public Initon PDE_DecrementE_IU(InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular, boolean
bYS) {
        Initon initonLink= initonLinkDNA. getInitonLink();
---------------------------------------------------------------------------------------------------
        int index= 0;
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("E"))
                { Initon initonIncrementI= new Initon();
                if(bYS) {
                    if(pDE_RNA_Formular. pdedeKey. charAt(index++)== '0')
                        { initonIncrementI. setI(); //新增一个数据I
                    }else {
                        initonIncrementI. setD(); //新增一个数据D
                    }
                }else {
                    if(Math. random()< pDE_RNA_Formular. key[2])
                        { pDE_RNA_Formular. pdedeKey+= "0";
                        initonIncrementI. setI(); //新增一个数据I
                    }else {
                        pDE_RNA_Formular. pdedeKey+= "1";
                        initonIncrementI. setD(); //新增一个数据D
                    }
                }
                Initon initonIncrementU= new Initon(); //u
                initonIncrementU. setU(); //新增一个数据u
                initonIncrementI. next= initonIncrementU; //I初始
                initonIncrementI. prev= initonLink. prev;
                if(null!= initonIncrementI. prev) { initonIncrementI.
                    prev. next= initonIncrementI;
                }
                initonIncrementU. prev= initonIncrementI; //U初始
                initonIncrementU. next= initonLink. next;
```

```
                if(null!= initonIncrementU. next) { initonIncrementU.
                    next. prev= initonIncrementU;
```

```
                }
                initonLink= initonIncrementU; //最后U代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop 替增.
        }
        return initonLink;
    }
    //C = I + D
    public Initon PDE_DecrementC(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("C"))
                { Initon initonIncrementI= new Initon();
                initonIncrementI. setI(); //新增一个数据I
                Initon initonIncrementD= new Initon();
                initonIncrementD. setD(); //新增一个数据D
                initonIncrementI. next= initonIncrementD; //I初始
                initonIncrementI. prev= initonLink. prev;
                if(null!= initonIncrementI. prev) { initonIncrementI.
                    prev. next= initonIncrementI;
                }
                initonIncrementD. prev= initonIncrementI; //D初始
                initonIncrementD. next= initonLink. next;
                if(null!= initonIncrementD. next) { initonIncrementD.
                    next. prev= initonIncrementD;
                }
                initonLink= initonIncrementD; //最后D代替
```

```
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop 替增.
        }
        return initonLink;
    }
    //S = I + Q
    public Initon PDE_DecrementS(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("S"))
                { Initon initonIncrementI= new Initon();
                initonIncrementI. setI(); //新增一个数据I
                Initon initonIncrementQ= new Initon();
                initonIncrementQ. setQ(); //新增一个数据Q
                initonIncrementI. next= initonIncrementQ; //I初始
                initonIncrementI. prev= initonLink. prev;
                if(null!= initonIncrementI. prev) { initonIncrementI.
                    prev. next= initonIncrementI;
                }
                initonIncrementQ. prev= initonIncrementI; //Q初始
                initonIncrementQ. next= initonLink. next;
                if(null!= initonIncrementQ. next) { initonIncrementQ.
                    next. prev= initonIncrementQ;
                }
                initonLink= initonIncrementQ; //最后Q代替
            }
            if(!initonLink. hasNext()) {
```

```
                return initonLink;
            }
            initonLink= initonLink. next; //while loop 替增.
```

```
        }
        return initonLink;
    }
    //E = D + U
    //肽展公式的概率问题, 先这样命名, 之后讨论  是IU还是DU
    public Initon PDE_DecrementE_DU(InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("E"))
                { Initon initonIncrementD= new Initon();
                if(Math. random()< pDE_RNA_Formular. key[2])
                    { initonIncrementD. setI(); //新增一个数据I
                }else {
                    initonIncrementD. setD(); //新增一个数据D
                }
                Initon initonIncrementU= new Initon();
                initonIncrementU. setU(); //新增一个数据U
                initonIncrementD. next= initonIncrementU; //D初始
                initonIncrementD. prev= initonLink. prev;
                if(null!= initonIncrementD. prev) { initonIncrementD.
                    prev. next= initonIncrementD;
                }
                initonIncrementU. prev= initonIncrementD; //U初始
                initonIncrementU. next= initonLink. next;
                if(null!= initonIncrementU. next) { initonIncrementU.
                    next. prev= initonIncrementU;
                }
```

```
                    initonLink= initonIncrementU; //最后U代替
                }
                if(!initonLink. hasNext()) {
                    return initonLink;
                }
                initonLink= initonLink. next; //while loop 替增.
            }
            return initonLink;
        }
        //S = I
        //肽展公式的概率问题, 先这样命名, 之后讨论  是I, Q还是I + Q
        public Initon PDE_DecrementS_I(InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular, boolean
bYS) {
            Initon initonLink= initonLinkDNA. getInitonLink();
            int index= 0;
            while(null!= initonLink) {
                if(initonLink. getStore(). equalsIgnoreCase("S"))
                    { Initon initonIncrementI= new Initon();
                    if(bYS) {
                        try {
                            if(pDE_RNA_Formular. pdedsKey. charAt(index++)== '0')
                                { initonIncrementI. setI(); //新增一个数据I
                            }else {
                                initonIncrementI. setQ(); //新增一个数据Q
                            }
                        }catch(Exception e) {
                            e. printStackTrace();
                        }
                    }else {
                        if(Math. random()< pDE_RNA_Formular. key[1]) {
```

```
                    pDE_RNA_Formular. pdedsKey+= "0";
                    initonIncrementI. setI();
                }else {
                    pDE_RNA_Formular. pdedsKey+= "1";
                    initonIncrementI. setQ();
                }
            }
            initonIncrementI. prev= initonLink. prev;
            if(null!= initonIncrementI. prev) { initonIncrementI.
                prev. next= initonIncrementI;
            }
            initonIncrementI. next= initonLink. next;
            if(null!= initonIncrementI. next) { initonIncrementI.
                next. prev= initonIncrementI;
            }
            initonLink= initonIncrementI; //最后I代替
        }
        if(!initonLink. hasNext()) {
            //System. out. println(pDE_RNA_Formular. pdedsKey);
            return initonLink;
        }
        initonLink= initonLink. next; //while loop 替增.
    }
    //System. out. println(pDE_RNA_Formular. pdedsKey);
    return initonLink;
}
//S = Q
//肽展公式的概率问题, 先这样命名, 之后讨论  是I, Q还是I + Q
public Initon PDE_DecrementS_Q(InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular)
    { Initon initonLink= initonLinkDNA. getInitonLink();
```

```
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("S"))
                { Initon initonIncrementQ= new Initon();
                if(Math. random()< pDE_RNA_Formular. key[1])
                    { initonIncrementQ. setI();
                }else {
                    initonIncrementQ. setQ();
                }
                initonIncrementQ. prev= initonLink. prev;
                if(null!= initonIncrementQ. prev) { initonIncrementQ.
                    prev. next= initonIncrementQ;
                }
                initonIncrementQ. next= initonLink. next;
                if(null!= initonIncrementQ. next) { initonIncrementQ.
                    next. prev= initonIncrementQ;
                }
                initonLink= initonIncrementQ; //最后Q代替
            }
            if(!initonLink. hasNext()) {
                return initonLink;
            }
            initonLink= initonLink. next; //while loop 替增.
        }
        return initonLink;
    }
}
```

```
//tokenpdi 的 pde 增元
package OSI. OSU. SI. SD. SU. SQ. ASU. OSU. PSU. MSU. AVQ. ASQ. tin. catalytic. procedure. pde;
import java. util. Iterator;
import java. util. List;
import java. util. concurrent. CopyOnWriteArrayList;
//注意: 该 文件对应的是罗瑶光先生 DNA 编码 与 计算的两本 国家软著作 思想的编码 实现.
//公安部 与 知识产权委员会 已经备案, 可阅读 相关 著作权 原文 进行逻辑辨别.
public class PDE_Increment_Formular {
    //A = V + S LINK 数据结构对象 (简单测试)
    //这个函数我没有按 sonar 模式 修改, 因为我担心 sonar 会潜意识 改变我的写作模式,
    //于是 按照我很多年的 ANSI C 代码风格进行编写, 因为这种方式是我的基础. 我个人觉
    //得 一个人的研发能力来自于他的母语水平. 所以我是 C 基础, 我应该继续跟进这个天赋. 正如下面这函
数.
    public Initon PDE_IncrementA(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("V"))
                { if(initonLink. hasNext()) {
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("S")) {
                        Initon initonIncrementA= new Initon();
                        initonIncrementA. setA(); //新增一个数据 A
                        if(initonNext. hasNext()) {
                            initonIncrementA. next= initonNext. next; //A 后序替换
                            initonIncrementA. next. prev= initonIncrementA; //A 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementA. prev= initonNext. prev. prev; //A 前序替换
                            initonIncrementA. prev. next= initonIncrementA; //A 前序后序恒等
                        }
```

```
                    initonLink= initonIncrementA; //最后  A 代替
                }
            }
        }
        if(!initonLink. hasNext())
            { return initonLink;
        }
        initonLink= initonLink. forwardNext(); //while loop  替增.
    }
    return initonLink;
}

//A = V + S LIST jdk util 对象, 下面函数是直接用JDK 的虚拟机函数  编写的, 逻辑比较清晰, 各有各的用处.
public List<Initon> PDE_IncrementA(List<Initon> Initons)
    { List<Initon> output= new CopyOnWriteArrayList<>();
    Iterator<Initon> iterator= Initons. iterator();
    HERE:
        while(iterator. hasNext()) {
            Initon initon= iterator. next();
            if(initon. getStore(). equalsIgnoreCase("V"))
                { if(iterator. hasNext()) {
                    Initon initonNext= iterator. next();
                    if(initonNext. getStore(). equalsIgnoreCase("S"))
                        { Initon initonIncrementA= new Initon();
                        initonIncrementA. setA();
                        output. add(initonIncrementA);
                        continue HERE;
                    }
                    Initon initonIncrementV= new Initon();
```

```
                    initonIncrementV. setV();
                    Initon initonIncrementS= new Initon();
                    initonIncrementS. setS();
                    output. add(initonIncrementV);
                    output.  add(initonIncrementS);
                    continue HERE;
                }
                Initon initonIncrementV= new Initon();
                initonIncrementV. setV();
                output. add(initonIncrementV);
                continue HERE;
            }
            output. add(initon);
        }
    return output;
}

//O = E + S LINK 数据结构对象  (简单测试)
public Initon PDE_IncrementO(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("E"))
            { if(initonLink. hasNext()) {
                Initon initonNext= initonLink. forwardNext();
                if(initonNext. getStore(). equalsIgnoreCase("S")) {
                    Initon initonIncrementO= new Initon();
                    initonIncrementO. setO(); //新增一个数据  O
                    if(initonNext. hasNext()) {
                        initonIncrementO. next= initonNext. next; //O 后序替换
                        initonIncrementO. next. prev= initonIncrementO; //O 后序前序恒等
```

```
                    }
                    if(null!= initonNext. prev. prev) {
                        initonIncrementO. prev= initonNext. prev. prev; //O 前序替换
                        initonIncrementO. prev. next= initonIncrementO; //O 前序后序恒等
                    }
                    initonLink= initonIncrementO; //最后  O 代替
                }
            }
        }
        if(!initonLink. hasNext())
            { return initonLink;
        }
        initonLink= initonLink. forwardNext(); //while loop  替增.
    }
    return initonLink;
}

//P = E + C LINK 数据结构对象  (简单测试)
public Initon PDE_IncrementP(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("E"))
            { if(initonLink. hasNext()) {
                Initon initonNext= initonLink. forwardNext();
                if(initonNext. getStore(). equalsIgnoreCase("C")) {
                    Initon initonIncrementP= new Initon();
                    initonIncrementP. setP(); //新增一个数据  P
                    if(initonNext. hasNext()) {
                        initonIncrementP. next= initonNext. next; //P 后序替换
                        initonIncrementP. next. prev= initonIncrementP; //P 后序前序恒等
```

```
                    }
                    if(null!= initonNext. prev. prev) {
                        initonIncrementP. prev= initonNext. prev. prev; //P 前序替换
                        initonIncrementP. prev. next= initonIncrementP; //P 前序后序恒等
                    }
                    initonLink= initonIncrementP; //最后  P 代替
                }
            }
        }
        if(!initonLink. hasNext())
            { return initonLink;
        }
        initonLink= initonLink. forwardNext(); //while loop  替增.
    }
    return initonLink;
}
//M = C + S
public Initon PDE_IncrementM(InitonLinkDNA initonLinkDNA)
    { Initon initonLink= initonLinkDNA. getInitonLink();
    while(null!= initonLink) {
        if(initonLink. getStore(). equalsIgnoreCase("C"))
            { if(initonLink. hasNext()) {
                Initon initonNext= initonLink. forwardNext();
                if(initonNext. getStore(). equalsIgnoreCase("S")) {
                    Initon initonIncrementM= new Initon();
                    initonIncrementM. setM(); //新增一个数据M
                    if(initonNext. hasNext()) {
                        initonIncrementM. next= initonNext. next; //M 后序替换
                        initonIncrementM. next. prev= initonIncrementM; //M 后序前序恒等
                    }
```

```
                        if(null!= initonNext. prev. prev) {
                            initonIncrementM. prev= initonNext. prev. prev; //M 前序替换
                            initonIncrementM. prev. next= initonIncrementM; //M 前序后序恒等
                        }
                        initonLink= initonIncrementM; //最后  M 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop  替增.
        }
        return initonLink;
}

//V = U + Q
public Initon PDE_IncrementV(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("U"))
                { if(initonLink. hasNext()) {
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("Q")) {
                        Initon initonIncrementV= new Initon();
                        initonIncrementV. setV(); //新增一个数据  V
                        if(initonNext. hasNext()) {
                            initonIncrementV. next= initonNext. next; //V 后序替换
                            initonIncrementV. next. prev= initonIncrementV; //V 后序前序恒等
                        }
```

```
                        if(null!= initonNext. prev. prev) {
                            initonIncrementV. prev= initonNext. prev. prev; //V 前序替换
                            initonIncrementV. prev. next= initonIncrementV; //V 前序后序恒等
                        }
                        initonLink= initonIncrementV; //最后  V 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop  替增.
        }
        return initonLink;
    }

    //E = I + U
    //肽展公式的概率问题, 先这样命名, 之后讨论  是  IU 还是  DU
    public Initon PDE_IncrementE_IU(InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular, boolean
bYS) {
        Initon initonLink= initonLinkDNA. getInitonLink();
        //int index= 0;
        while(null!= initonLink) {
            String initonString= initonLink. getStore();
            if(initonLink. getStore(). equalsIgnoreCase("I")|| initonLink. getStore(). equalsIgnoreCase("D"))
                { if(initonLink. hasNext()) {
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("U")) {
                        Initon initonIncrementE= new Initon();
                        initonIncrementE. setE(); //新增一个数据  E
```

```
                        if(initonString. equalsIgnoreCase("I"))
                            { pDE_RNA_Formular. pdeieKey+=
                            "0";
                        }else if(initonString. equalsIgnoreCase("D"))
                            { pDE_RNA_Formular. pdeieKey+= "1";
                        }
                        if(initonNext. hasNext()) {
                            initonIncrementE. next= initonNext. next; //E 后序替换
                            initonIncrementE. next. prev= initonIncrementE; //E 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementE. prev= initonNext. prev. prev; //E 前序替换
                            initonIncrementE. prev. next= initonIncrementE; //E 前序后序恒等
                        }
                        initonLink= initonIncrementE; //最后  E 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop  替增.
        }
        return initonLink;
    }

    //C = I + D
    public Initon PDE_IncrementC(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("I")) {
```

```
                if(initonLink. hasNext()) {
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("D")) {
                        Initon initonIncrementC= new Initon();
                        initonIncrementC. setC(); //新增一个数据 C
                        if(initonNext. hasNext()) {
                            initonIncrementC. next= initonNext. next; //C 后序替换
                            initonIncrementC. next. prev= initonIncrementC; //C 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementC. prev= initonNext. prev. prev; //M 前序替换
                            initonIncrementC. prev. next= initonIncrementC; //M 前序后序恒等
                        }
                        initonLink= initonIncrementC; //最后 C 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop 替增.
        }
        return initonLink;
    }
    //S = I + Q
    public Initon PDE_IncrementS(InitonLinkDNA initonLinkDNA)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("I"))
                { if(initonLink. hasNext()) {
```

```
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("Q")) {
                        Initon initonIncrementS= new Initon();
                        initonIncrementS. setS(); //新增一个数据 S
                        if(initonNext. hasNext()) {
                            initonIncrementS. next= initonNext. next; //S 后序替换
                            initonIncrementS. next. prev= initonIncrementS; //S 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementS. prev= initonNext. prev. prev; //S 前序替换
                            initonIncrementS. prev. next= initonIncrementS; //S 前序后序恒等
                        }
                        initonLink= initonIncrementS; //最后 S 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop 替增.
        }
        return initonLink;
    }
    //E = D + U
    //肽展公式的概率问题, 先这样命名, 之后讨论 是 IU 还是 DU
    public Initon PDE_IncrementE_DU(InitonLinkDNA initonLinkDNA) {
        Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("D"))
                { if(initonLink. hasNext()) {
```

```
                    Initon initonNext= initonLink. forwardNext();
                    if(initonNext. getStore(). equalsIgnoreCase("U")) {
                        Initon initonIncrementE= new Initon();
                        initonIncrementE. setE(); //新增一个数据  E
                        if(initonNext. hasNext()) {
                            initonIncrementE. next= initonNext. next; //E 后序替换
                            initonIncrementE. next. prev= initonIncrementE; //E 后序前序恒等
                        }
                        if(null!= initonNext. prev. prev) {
                            initonIncrementE. prev= initonNext. prev. prev; //E 前序替换
                            initonIncrementE. prev. next= initonIncrementE; //E 前序后序恒等
                        }
                        initonLink= initonIncrementE; //最后  E 代替
                    }
                }
            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop  替增.
        }
        return initonLink;
    }
    //S = I
    //肽展公式的概率问题, 先这样命名, 之后讨论  是  I, Q 还是  I + Q
    public Initon PDE_IncrementS_I(InitonLinkDNA initonLinkDNA, TokenPDI pDE_RNA_Formular)
        { Initon initonLink= initonLinkDNA. getInitonLink();
        while(null!= initonLink) {
            String initonString= initonLink. getStore();
            if(initonLink. getStore(). equalsIgnoreCase("I")|| initonLink. getStore(). equalsIgnoreCase("Q")) {
```

```
                Initon initonIncrementS= new Initon();
                initonIncrementS. setS(); //新增一个数据  S
                if(initonString. equalsIgnoreCase("I")) {
                    pDE_RNA_Formular. pdeisKey+= "0";
                }else if(initonString. equalsIgnoreCase("Q"))
                    { pDE_RNA_Formular. pdeisKey+= "1";
                }
                if(initonLink. hasNext()) {
                    initonIncrementS. next= initonLink. next; //S 后序替换
                    initonIncrementS. next. prev= initonIncrementS; //S 后序前序恒等
                }
                if(null!= initonLink. prev) {
                    initonIncrementS. prev= initonLink. prev; //S 前序替换
                    initonIncrementS. prev. next= initonIncrementS; //S 前序后序恒等
                }
                initonLink= initonIncrementS; //最后  S 代替

            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop  替增.
        }
        return initonLink;
    }

    //S = Q
    //肽展公式的概率问题, 先这样命名, 之后讨论  是  I, Q 还是  I + Q
    public Initon PDE_IncrementS_Q(InitonLinkDNA initonLinkDNA) {
        Initon initonLink= initonLinkDNA. getInitonLink();
```

```
        while(null!= initonLink) {
            if(initonLink. getStore(). equalsIgnoreCase("Q"))
                { Initon initonIncrementS= new Initon();
                initonIncrementS. setS(); //新增一个数据  S
                if(initonLink. hasNext()) {
                    initonIncrementS. next= initonLink. next; //S 后序替换
                    initonIncrementS. next. prev= initonIncrementS; //S 后序前序恒等
                }
                if(null!= initonLink. prev) {
                    initonIncrementS. prev= initonLink. prev; //S 前序替换
                    initonIncrementS. prev. next= initonIncrementS; //S 前序后序恒等
                }
                initonLink= initonIncrementS; //最后  S 代替

            }
            if(!initonLink. hasNext())
                { return initonLink;
            }
            initonLink= initonLink. forwardNext(); //while loop  替增.
        }
        return initonLink;
    }
}
```