目录

[第十三章 DNA数术推导与RNA\_X\_THF\_DD元基芯片与肽逻辑 6](#_Toc23512)

[第一节 DNA数术的动机 6](#_Toc29409)

[第二节 DNA数术的应用需求 6](#_Toc25909)

[第三节 DNA数术的具体描述 6](#_Toc18906)

[元基数术, 活性, 腐蚀性排序表 6](#_Toc825)

[元基语义五行排序图 7](#_Toc4771)

[元基语义排序罗盘 7](#_Toc7551)

[元基语义肽展活性排序罗盘 7](#_Toc20108)

[元基肽展公式关系图 8](#_Toc4020)

[元基腐蚀性排序罗盘 8](#_Toc13248)

[语义生化双元基叠加罗盘 9](#_Toc9150)

[无机罗盘术数 9](#_Toc18256)

[生化钥匙罗盘 10](#_Toc10789)

[语义钥匙罗盘 11](#_Toc24695)

[第四节 DNA数术的应用实现 12](#_Toc2756)

[DecadeToPDS, 进制类 12](#_Toc9984)

[PDE\_PDS\_DL, 肽展类 22](#_Toc14353)

[第五节 全嘌呤的推导 48](#_Toc28162)

[DCPE THOS MAXF VIUQ 十六进制推导 58](#_Toc371)

[FU 全嘌呤变嘧啶数字锁存逻辑 61](#_Toc5448)

[十六元基进制的数字逻辑与离散数学发散 62](#_Toc25985)

[第十四章 DNA搜索与筛选应用 63](#_Toc12059)

[第一节 DNA搜索的动机 64](#_Toc19244)

[第二节 DNA搜索的应用需求 64](#_Toc21839)

[第三节 DNA搜索的具体描述 64](#_Toc273)

[ZhongYaoSearch, 搜索类15 64](#_Toc12222)

[DNA搜索的应用实现 64](#_Toc7306)

[第四节 DNA筛选的动机 64](#_Toc32665)

[第五节 DNA筛选的应用需求 64](#_Toc24132)

[第六节 DNA筛选的具体描述 64](#_Toc24113)

[味觉语义元基定义 64](#_Toc12369)

[味觉生化元基定义 64](#_Toc4908)

[双元筛选索引词库 64](#_Toc12826)

[第七节DNA筛选的应用实现 64](#_Toc17730)

[第八节DNN分词 词汇花函数源码 64](#_Toc10401)

[方剂森林花JOGL三维计算展示函数 64](#_Toc8689)

[药材功效花JOGL三维计算展示函数 64](#_Toc7653)

[药材禁忌花JOGL三维计算展示函数 64](#_Toc15312)

[花的筛选与观测 65](#_Toc4208)

[第十五章 元基模拟染色体新陈代谢催化编码 65](#_Toc7262)

[第一节 元基造字 65](#_Toc11660)

[Q\_OulerRing, 欧拉路径类 65](#_Toc4407)

[LYG9DWithDoubleTopSort4D, 极速排序算法 65](#_Toc82)

[LYG9DWithDoubleTopSort4D\_U, 极速排序算法 65](#_Toc17633)

[Top Sort 5D 65](#_Toc24833)

[第二节 最新笔记 包含十六元基造字 65](#_Toc12522)

[二次元基新陈代谢方式 65](#_Toc28480)

[LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类 66](#_Toc6457)

[LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类 66](#_Toc24274)

[LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类 67](#_Toc11310)

[第三节 图片识别 67](#_Toc28560)

[图片读脏能力 67](#_Toc20793)

[SkinPathDetectDis, 肽展图片处理类 67](#_Toc24131)

[GetICAStatisticRatio, 肽展图片处理类 67](#_Toc16583)

[GetColorRatioScore, 肽展图片处理类 67](#_Toc23874)

[SkinPathDetectTrip, 肽展图片处理类 67](#_Toc22767)

[MakeImag, 肽展图片处理类 67](#_Toc23291)

[第四节 元基枝与元基花及其在分词，排序，索引，加密上的应用 67](#_Toc31823)

[LYG9DWithDoubleTopSort4D，极速象契混合排序 67](#_Toc19925)

[LYG10DWCMSSort15D\_XCDX\_P\_U\_A, 象契字符排序类 67](#_Toc29937)

[LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类 67](#_Toc14345)

[LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C, 象契字符排序类 67](#_Toc25390)

[LYG10DWCMSSort13D\_XCDX\_S, 象契字符排序类 67](#_Toc27495)

[LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类 67](#_Toc16625)

[LYG10DWCMSSort13D\_XCDX\_P\_A\_C, 象契字符排序类 68](#_Toc5156)

[LYG10DWCMSSort13D\_XCDX\_S\_C, 象契字符排序类 68](#_Toc966)

[LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类 68](#_Toc2845)

[LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类 68](#_Toc31890)

[LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类 68](#_Toc13111)

[LYG10DWCMSSort13D\_XCDX\_C\_A\_C, 象契字符排序类 68](#_Toc18926)

[AE\_XCDX\_Map, 肽展中文分词类 68](#_Toc709)

[AE, 肽展中文分词类 68](#_Toc6778)

[A, 肽展中文分词类 68](#_Toc8477)

[A\_XCDX\_Map, 肽展中文分词类 68](#_Toc9890)

[BinaryForest, 肽展分词索引类 68](#_Toc442)

[BinaryForest\_A, 肽展分词索引类 68](#_Toc17832)

[CogsBinaryForest\_AE, 肽展分词索引类 68](#_Toc29096)

[CogsBinaryForest\_A, 肽展分词索引类 68](#_Toc24568)

[BinaryForest\_AE, 肽展分词索引类 68](#_Toc28936)

[Nlp\_CE\_XCDX\_A, 肽展分词索引类 68](#_Toc20744)

[Nlp\_C\_XCDX\_A, 肽展分词索引类 68](#_Toc16873)

[Nlp\_CE\_XCDX\_S, 肽展分词索引类 68](#_Toc7510)

[Nlp\_CE\_XCDX\_A, 肽展分词索引类 68](#_Toc7889)

[Nlp\_C\_XCDX\_S, 肽展分词索引类 68](#_Toc17733)

[Nlp\_CE\_XCDX, 肽展分词索引类 68](#_Toc24235)

[POS\_C\_Cognition\_E, 肽展分词索引类 68](#_Toc995)

[POS\_C, 肽展分词索引类 68](#_Toc14576)

[Pos\_CE\_XCDX\_E, 肽展分词索引类 68](#_Toc223)

[Pos\_CE\_XCDX\_O, 肽展分词索引类 68](#_Toc13377)

[Pos\_C\_XCDX\_E, 肽展分词索引类 68](#_Toc30523)

[Pos\_C\_XCDX\_O, 肽展分词索引类 68](#_Toc7805)

[Pos\_CE\_XCDX\_P, 肽展分词索引类 68](#_Toc27522)

[Pos\_CE\_XCDX\_E, 肽展分词索引类 68](#_Toc21095)

[Pos\_C\_XCDX\_P, 肽展分词索引类 68](#_Toc13006)

[Pos\_CE\_XCDX, 肽展分词索引类 68](#_Toc5508)

[第十六章 TinShell插件\_元基花模拟染色体组计算索引系统 68](#_Toc22545)

[第一节 软件介绍 68](#_Toc11532)

[软件开发动机 68](#_Toc550)

[软件开发目的 68](#_Toc18664)

[软件价值 68](#_Toc7138)

[软件主要功能 68](#_Toc226)

[软件开发系统环境 68](#_Toc32011)

[硬件开发系统环境 68](#_Toc30787)

[软件开发软件环境 68](#_Toc9209)

[软件开发硬件环境 68](#_Toc17256)

[软件部署软件环境 68](#_Toc19793)

[软件部署硬件环境 68](#_Toc18212)

[软件办公环境 68](#_Toc25205)

[软件使用方法 68](#_Toc18391)

[软件执行逻辑 69](#_Toc15520)

[软件注意细节 69](#_Toc20137)

[软件申明 69](#_Toc27086)

[软件大小 69](#_Toc32084)

[软件的设计思维 69](#_Toc18899)

[软件的架构理念 69](#_Toc27984)

[第二节 软件源码 69](#_Toc19803)

[RangePDI 70](#_Toc23913)

[LYG9DWithDoubleTopSort5D 70](#_Toc11679)

[StaticRootMap 70](#_Toc18698)

[StaticClassMap 70](#_Toc27415)

[StaticFunctionMap 70](#_Toc7862)

[StaticFunctionMapA\_VECS\_C 70](#_Toc4725)

[StaticFunctionMapA\_IDUQ\_C 70](#_Toc14225)

[StaticFunctionMapO\_VECS\_C 70](#_Toc676)

[StaticFunctionMapO\_IDUQ\_C 70](#_Toc32740)

[StaticFunctionMapP\_VECS\_C 70](#_Toc937)

[StaticFunctionMapP\_IDUQ\_C 70](#_Toc252)

[StaticFunctionMapM\_VECS\_C 70](#_Toc15760)

[StaticFunctionMapM\_IDUQ\_C 70](#_Toc16091)

[StaticFunctionMapA\_VECS\_E 70](#_Toc22483)

[StaticFunctionMapA\_IDUQ\_E 70](#_Toc12433)

[StaticFunctionMapO\_VECS\_E 70](#_Toc3288)

[StaticFunctionMapO\_IDUQ\_E 70](#_Toc4149)

[StaticFunctionMapP\_VECS\_E 70](#_Toc6774)

[StaticFunctionMapP\_IDUQ\_E 70](#_Toc26586)

[StaticFunctionMapM\_VECS\_E 71](#_Toc32682)

[StaticFunctionMapM\_IDUQ\_E 71](#_Toc10994)

[StaticFunctionMapV\_AOPM\_C 71](#_Toc14559)

[StaticFunctionMapV\_IDUQ\_C 71](#_Toc26619)

[StaticFunctionMapE\_AOPM\_C 71](#_Toc30575)

[StaticFunctionMapE\_IDUQ\_C 71](#_Toc15469)

[StaticFunctionMapC\_AOPM\_C 71](#_Toc30288)

[StaticFunctionMapC\_IDUQ\_C 71](#_Toc23817)

[StaticFunctionMapS\_AOPM\_C 71](#_Toc22670)

[StaticFunctionMapS\_IDUQ\_C 71](#_Toc27486)

[StaticFunctionMapV\_AOPM\_E 71](#_Toc30976)

[StaticFunctionMapV\_IDUQ\_E 71](#_Toc15193)

[StaticFunctionMapE\_AOPM\_E 71](#_Toc14914)

[StaticFunctionMapE\_IDUQ\_E 71](#_Toc17815)

[StaticFunctionMapC\_AOPM\_E 71](#_Toc25349)

[StaticFunctionMapC\_IDUQ\_E 71](#_Toc26664)

[StaticFunctionMapS\_AOPM\_E 71](#_Toc13417)

[StaticFunctionMapS\_IDUQ\_E 71](#_Toc8405)

[StaticFunctionMapI\_VECS\_C 71](#_Toc31608)

[StaticFunctionMapI\_AOPM\_C 71](#_Toc4494)

[StaticFunctionMapD\_VECS\_C 71](#_Toc3136)

[StaticFunctionMapD\_AOPM\_C 71](#_Toc6272)

[StaticFunctionMapU\_VECS\_C 71](#_Toc9514)

[StaticFunctionMapU\_AOPM\_C 71](#_Toc14235)

[StaticFunctionMapQ\_VECS\_C 71](#_Toc12525)

[StaticFunctionMapQ\_AOPM\_C 71](#_Toc25906)

[StaticFunctionMapI\_VECS\_E 71](#_Toc4385)

[StaticFunctionMapI\_AOPM\_E 71](#_Toc30413)

[StaticFunctionMapD\_VECS\_E 71](#_Toc32335)

[StaticFunctionMapD\_AOPM\_E 71](#_Toc8039)

[StaticFunctionMapU\_VECS\_E 71](#_Toc12728)

[StaticFunctionMapU\_AOPM\_E 71](#_Toc1191)

[StaticFunctionMapQ\_VECS\_E 71](#_Toc28204)

[StaticFunctionMapQ\_AOPM\_E 71](#_Toc19157)

[序列化索引调用真实示例 71](#_Toc26783)

[第十七章 后序DEMOS 71](#_Toc13731)

[登陆token 71](#_Toc22112)

[肽展session注册 71](#_Toc11260)

[登陆状态验证 71](#_Toc22051)

[PDESwapTestDemo 71](#_Toc8046)

[极速象契拼音笔画排序 71](#_Toc21050)

[精度中文搜索示例 72](#_Toc7165)

[人眼识别的方式 73](#_Toc11695)

[VPCS服务器部署 73](#_Toc11092)

[数字生命 73](#_Toc23331)

[引用 73](#_Toc20143)

[DNA元基催化与肽计算编码 73](#_Toc13203)

[DNA元基催化与肽计算肽展 73](#_Toc11513)

[DNA元基催化与肽计算解码 73](#_Toc21632)

[DNA元基催化与肽计算养疗经应用研究 73](#_Toc16112)

[德塔华瑞集养疗经软件工程类源码引用综合表 73](#_Toc20811)

1. DNA数术推导与RNA\_X\_THF\_DD元基芯片与肽逻辑

第一节 DNA数术的动机

为了方便中医与西医的结合, 辩证将 宏观医学与微观医学进行紧密的结合起来, 我一直在研究一种迅捷的有价值的元基观测角

度, 于是动机明显. 开始实践. 罗盘最大的作用是趋吉避凶，因果关联， 我之后的研发数据都会优化合并归纳在数据预测

API[5]最新版本中

1. DNA数术的应用需求

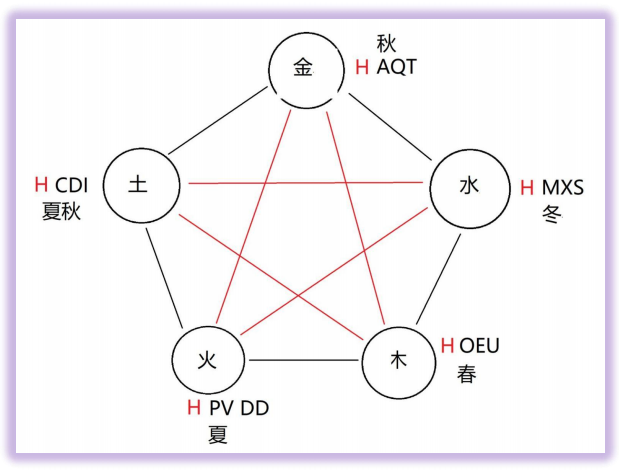
今天来仔细描述这一节. 看过我 CSDN, 知乎等媒体的相信一定有了解了. 我设计的模式很简单, 仅仅把元基解码的数据中 的酮基与甲基, 氨基 进行了计数, 形成了第一和第二排, 然后进行差值计算, 然后进行排序, 然后将元基图谱进行了离散的观测 排列到八卦罗盘形式进行了小简单变换然后连线排序了下活性, 于是生成了下面的各种基础元基罗盘图.

第三节 DNA数术的具体描述

元基数术, 活性, 腐蚀性排序表

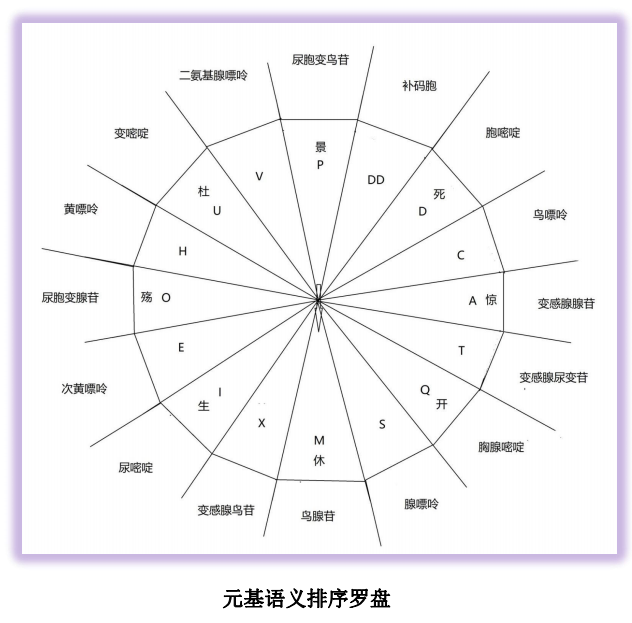


元基语义五行排序图



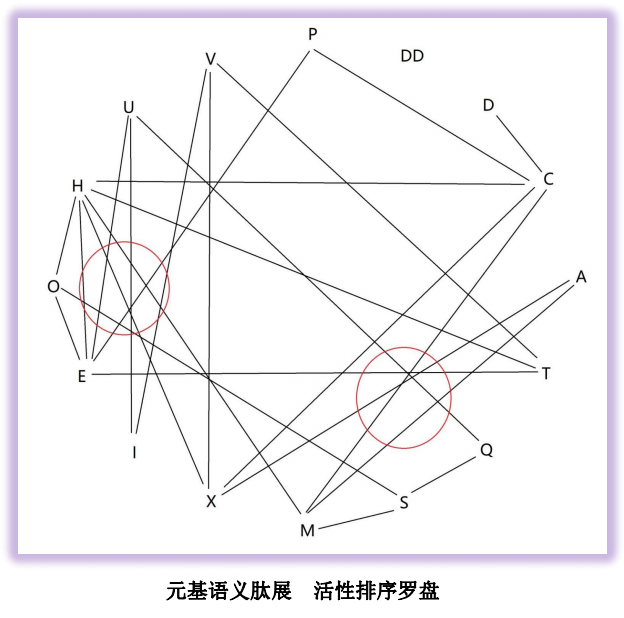
这里为什么我为什么会把 H 元基五行满排, 进行红色标注, 依据是它的活性. 与罗盘 7 个方位都有简洁变换逻辑. 其它元 基布局就好解释了, 按语义编码推导出来的.

元基语义排序罗盘



上面这个罗盘, 我是根据 DNA 解码一著进行了语义的罗盘排序, 没有变动, 一开始少了一个 DD, 后来我不妨大胆了一点, 把元基补码的 DD 按比值排序加进去, 计算发现也刚好在 PD 之间. 于是就不改了, 作为第一代元基罗盘放在第二卷里.

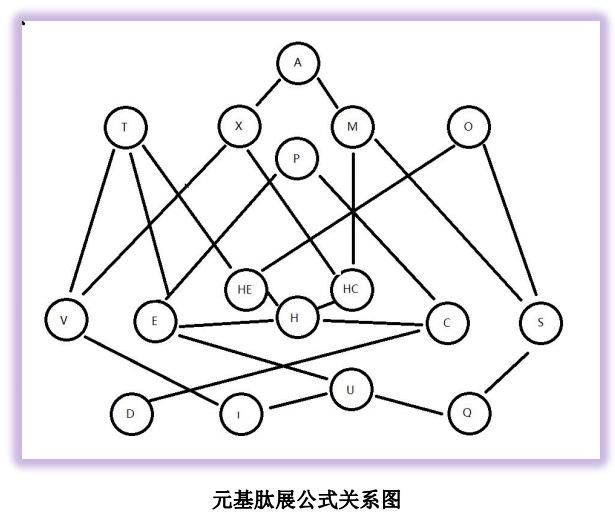
元基语义肽展活性排序罗盘



元基语义肽展 活性排序罗盘 是我将元基图谱进行了离散的观测排列到八卦罗盘形式进行了小简单变换然后连线排序了

下活性, 于是生成了下面的各种基础元基罗盘图, 我在这里举个例子怕大家不理解, 如下

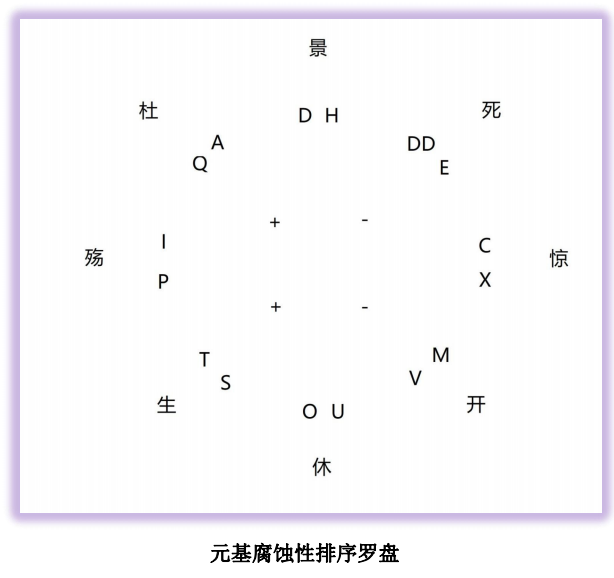
元基肽展公式关系图



上面这个图不是第二卷产物, 是第一卷已有的 DNA 解码文件. 根据肽展公式的生化计算解码推导出来的. 第一卷中我已

经描述的很详细了, 这里就停止介绍. 这里我要感谢 班加罗尔大学基督学院的维嘉斯神父, 08 年教我离散数学, 很认真, 谢谢他当年传授我这个离散观测变换的思维.元基语义肽展 活性排序罗盘是无向的, 而元基肽展公式关系图是有向的, 因为应用场景不同, 这里观测方式不同. 于是我准备结合肽展公式关系与酸碱分类进行生化模式排序如下:

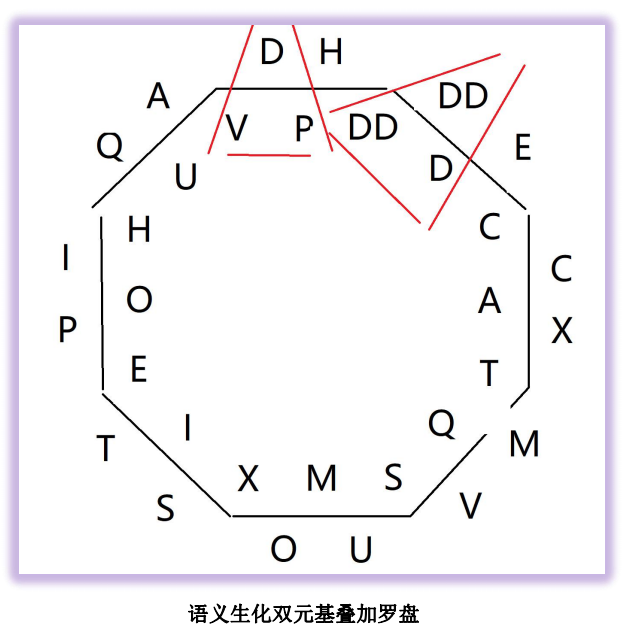
元基腐蚀性排序罗盘



上图元基是我计算了元基数术, 活性, 腐蚀性排序表 后将第六行进行腐蚀中性重新定义然后进行强度排序的. 效果不错,

录在第二卷里先, 稍后整稿

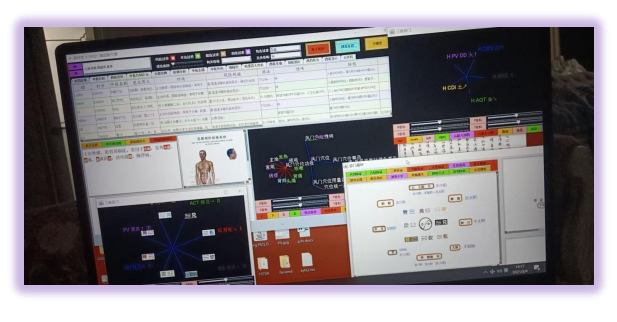
语义生化双元基叠加罗盘



这个图很好解释, 我将语义元基与腐蚀性排序元基 进行了无理级嵌套, 于是得到了这个无理级语义生化双元基叠加罗盘

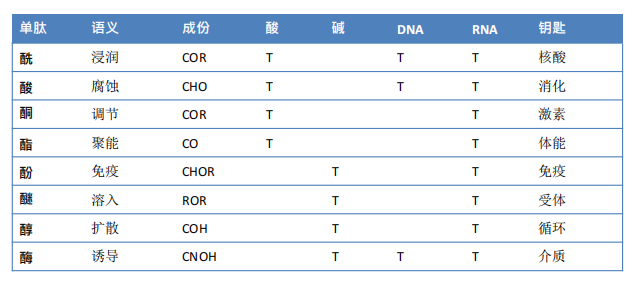
催化映射. 功能强大. 稍后描述. 这里右上角的 DD D DD 的双元催化双角, 胞嘧啶大堆, 这里是一个巨大的生化医学突破口, 第三卷 第四卷我会好好研究下. 先别想那么远, 我先把基础做好, 医学家有幸看到我的文章, 蠢蠢欲动, 让医学家去实验, 我按我的思路来, 求严谨, 不求跳跃. 另外, 语义肽展公式和生化肽展公式 开始有区别了, 我准备定个计划, 将语义肽展公式统一用生化肽展公式, 虽然会少几个, 但效率增加是必然.

无机罗盘术数



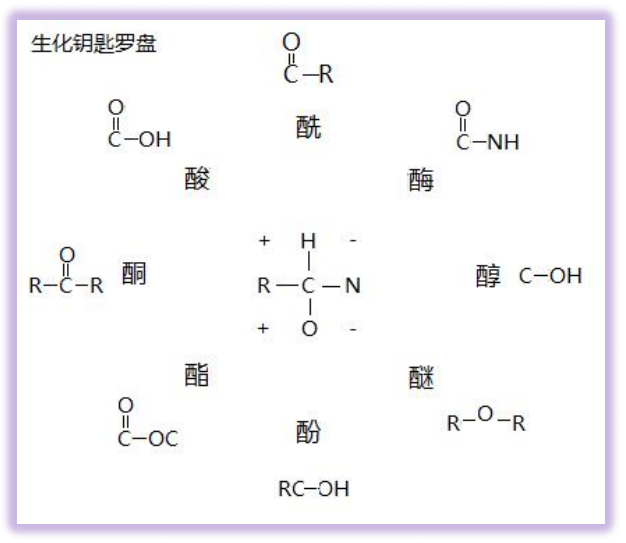
元基罗盘目前已经应用在德塔养疗经[17]医学大数据软件中, 用于中药数据的八纲辨证观测. 这只是一种简单应用, 我设

计了更多的实例. 我把这个图加入了养疗经[17]的奇门遁甲页面, 生成三维立体图, 用于环境, 中药. 为中医观测提供多种观测和 数据统计手段上面有了元基语义罗盘和元基生化罗盘后, DNA 催化的表达开始双元耦合, 于是第一能想到的是元基术数的 DNA 催化必定有钥匙来开启, 同时生化计算中催化钥匙同样可以用罗盘来表达, 于是我将下面八种非 DNA 单肽也通过酸碱活泼属性进行了方位归纳如下作为第一代钥匙参照物. 这酸醚醇酶 酚酯酰酮，八种肽键和其化学结构表达式来自人卫九[14] 等 生物化学教材书籍, 不是罗瑶光先生发明的, 卷头在前言中已经声明

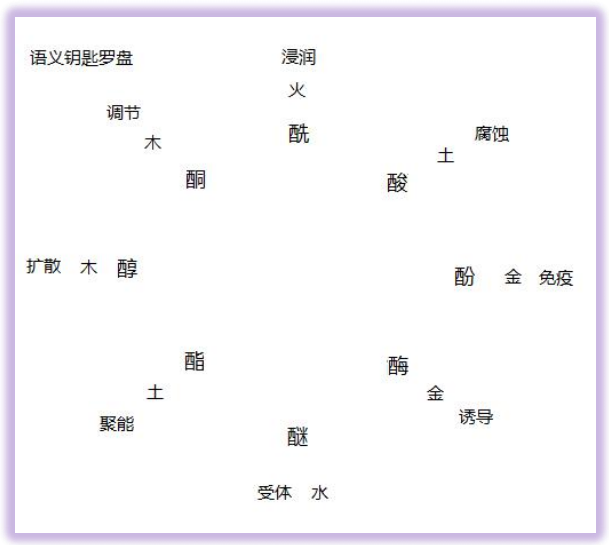


于是生成的语义钥匙和生化钥匙如下:

生化钥匙罗盘



语义钥匙罗盘





催化钥匙应用于 罗盘中观测展示, 我在这里描述下, 为什么我把笔记中的胺替换为醚, 因为胺的活性剧烈, 属于毒品原型,

在元基催化过程中已经类似于破坏的性质. 而醚类激素已经具备了其活性属性, 就去掉了

第四节 DNA数术的应用实现

DecadeToPDS, 进制类

package OSI.SSI.ASU.OSU.PSU.MSU.pde;

import java.util.HashMap;

import java.util.Map;

import OSI.SSI.ASU.OSU.PSU.MSU.pds.PDE\_PDS\_DL;

//这个函数集用于将常数变换成十七进制元基数字，

//这个函数集用于将十七进制元基数字进行元基变换

//这个函数集用于将元基变换进行肽展概率丝化展开

//这个函数用于将肽展丝化的肽增十七进制进行元基变换

//这个函数用于将肽展丝化的肽增十七进制变换成元基数字

//这个函数用于十七进制元基数字进行十进制还原。

public class DecadeToPDS{

//思想：肽展公式 1.2.2，元基数字逻辑； 十七进制元基组合数学；概率论

//算法：进制计算，肽展计算

//程序员： 罗瑶光，

public Map<String, String> initonsMap= new HashMap<>();

public Map<String, String> initonsCode= new HashMap<>();

public Map<String, String> initonsSet= new HashMap<>();

public Map<String, Integer> numberSet= new HashMap<>();

public static void main(String[] Args) {

DecadeToPDS decadeToPDS= new DecadeToPDS();

decadeToPDS.init(decadeToPDS);

int decade= (int)(Math.random()\*1000 % 256);//随便写一个数

double pDE\_KEY\_rate= 0.25;//随便模拟一个0-1之间的概率钥匙，假设 0~0.5为酸，0.5~1 为碱；

decadeToPDS.doPDS(decadeToPDS, decade, pDE\_KEY\_rate);

}

// //元基符号变元基数字

// //System.out.println("输入十进制数："+ decade);

// String seventeen= decadeToPDS.decadeToSeventeen(decade, decadeToPDS);

// //System.out.println("元基进制数为："+ seventeen);

// String initons= decadeToPDS.seventeenToIntons(seventeen, decadeToPDS);

// //System.out.println("变换为元基："+initons);

// //initons= "AOPMVE";

// //System.out.println("输入元基："+ initons);

// //System.out.println("输入概率："+ pDE\_KEY\_rate);

// String pDS= decadeToPDS.initonsToPDS(initons, pDE\_KEY\_rate, decadeToPDS);

// //System.out.println("输出肽丝:"+ pDS);

// pDS= pDS.replace(".", "");

// String pDSInitons= decadeToPDS.PDSToInitons(pDS, pDE\_KEY\_rate, decadeToPDS);

// //System.out.println("肽丝增元:"+ pDSInitons);

// //第二卷的肽展公式 可以用到了

// //String pDEInitons= decadeToPDS.PDSToPDE(pDSInitons, pDE\_KEY\_rate, decadeToPDS);

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

// String pDSSeventeen= decadeToPDS.initonsToSeventeen(pDSInitons, decadeToPDS);

// //System.out.println("元基数字:"+ pDSSeventeen);

// String pDSDecade= decadeToPDS.seventeenToDecade(pDSSeventeen, decadeToPDS);

// //System.out.println("输出十进制数:"+ pDSDecade);

// return Integer.valueOf(pDSDecade).intValue();

private int doPDS(DecadeToPDS decadeToPDS, int decade, double pDE\_KEY\_rate) {

String seventeen= decadeToPDS.decadeToSeventeen(decade, decadeToPDS);

String initons= decadeToPDS.seventeenToIntons(seventeen, decadeToPDS);

String pDS= decadeToPDS.initonsToPDS(initons, pDE\_KEY\_rate, decadeToPDS);

pDS= pDS.replace(".", "");

String pDSInitons= decadeToPDS.PDSToInitons(pDS, pDE\_KEY\_rate, decadeToPDS);

String pDSSeventeen= decadeToPDS.initonsToSeventeen(pDSInitons, decadeToPDS);

int pDSDecade= decadeToPDS.seventeenToDecade(pDSSeventeen, decadeToPDS);

return pDSDecade;

}

public void init(DecadeToPDS decadeToPDS) {

decadeToPDS.initonsMap.put("A", "7");

decadeToPDS.initonsMap.put("O", "A");

decadeToPDS.initonsMap.put("P", "2");

decadeToPDS.initonsMap.put("M", "8");

decadeToPDS.initonsMap.put("V", "D");

decadeToPDS.initonsMap.put("E", "3");

decadeToPDS.initonsMap.put("C", "1");

decadeToPDS.initonsMap.put("S", "9");

decadeToPDS.initonsMap.put("I", "E");

decadeToPDS.initonsMap.put("D", "0");

decadeToPDS.initonsMap.put("U", "F");

decadeToPDS.initonsMap.put("Q", "G");

decadeToPDS.initonsMap.put("T", "C");

decadeToPDS.initonsMap.put("X", "6");

decadeToPDS.initonsMap.put("+", "B");

decadeToPDS.initonsMap.put("-", "5");

decadeToPDS.initonsMap.put("H", "4");

//元基数字变元基符号

decadeToPDS.initonsCode.put("0", "D");

decadeToPDS.initonsCode.put("1", "C");

decadeToPDS.initonsCode.put("2", "P");

decadeToPDS.initonsCode.put("3", "E");

decadeToPDS.initonsCode.put("4", "H");

decadeToPDS.initonsCode.put("5", "-");

decadeToPDS.initonsCode.put("6", "X");

decadeToPDS.initonsCode.put("7", "A");

decadeToPDS.initonsCode.put("8", "M");

decadeToPDS.initonsCode.put("9", "S");

decadeToPDS.initonsCode.put("A", "O");

decadeToPDS.initonsCode.put("B", "+");

decadeToPDS.initonsCode.put("C", "T");

decadeToPDS.initonsCode.put("D", "V");

decadeToPDS.initonsCode.put("E", "I");

decadeToPDS.initonsCode.put("F", "U");

decadeToPDS.initonsCode.put("G", "Q");

//阿拉伯数字变元基数字

decadeToPDS.initonsSet.put("0", "0");

decadeToPDS.initonsSet.put("1", "1");

decadeToPDS.initonsSet.put("2", "2");

decadeToPDS.initonsSet.put("3", "3");

decadeToPDS.initonsSet.put("4", "4");

decadeToPDS.initonsSet.put("5", "5");

decadeToPDS.initonsSet.put("6", "6");

decadeToPDS.initonsSet.put("7", "7");

decadeToPDS.initonsSet.put("8", "8");

decadeToPDS.initonsSet.put("9", "9");

decadeToPDS.initonsSet.put("10", "A");

decadeToPDS.initonsSet.put("11", "B");

decadeToPDS.initonsSet.put("12", "C");

decadeToPDS.initonsSet.put("13", "D");

decadeToPDS.initonsSet.put("14", "E");

decadeToPDS.initonsSet.put("15", "F");

decadeToPDS.initonsSet.put("16", "G");

//元基数字变阿拉伯数字

decadeToPDS.numberSet.put("0", 0);

decadeToPDS.numberSet.put("1", 1);

decadeToPDS.numberSet.put("2", 2);

decadeToPDS.numberSet.put("3", 3);

decadeToPDS.numberSet.put("4", 4);

decadeToPDS.numberSet.put("5", 5);

decadeToPDS.numberSet.put("6", 6);

decadeToPDS.numberSet.put("7", 7);

decadeToPDS.numberSet.put("8", 8);

decadeToPDS.numberSet.put("9", 9);

decadeToPDS.numberSet.put("A", 10);

decadeToPDS.numberSet.put("B", 11);

decadeToPDS.numberSet.put("C", 12);

decadeToPDS.numberSet.put("D", 13);

decadeToPDS.numberSet.put("E", 14);

decadeToPDS.numberSet.put("F", 15);

decadeToPDS.numberSet.put("G", 16);

}

// //准备集成第二卷的AOPM 级别 肽展公式 ，已经并入PDSToInitons 函数中

// private String PDSToPDE(String pds, double pDE\_KEY\_rate, DecadeToPDS decadeToPDS) {

////pds= pds.replace("UQ", "V");

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

////pds= pds.replace("IQ", "S");

////pds= pds.replace("VS", "A");

////pds= pds.replace("ES", "O");

////pds= pds.replace("EC", "P");

////pds= pds.replace("CS", "M");

////pds= pds.replace("VE", "T");

////pds= pds.replace("VC", "X");

////

// return pds;

// }

//这个函数集用于将常数变换成十七进制元基数字，

public String decadeToSeventeen(int decade, DecadeToPDS decadeToPDS) {

String seventeen= "";

int decad= decade;

while(0< decad/ 17) {

int seventeenth= decad% 17;

seventeen= decadeToPDS.initonsSet.get(""+ seventeenth)+ seventeen;

decad= decad/ 17;

}

seventeen= decadeToPDS.initonsSet.get(""+ decad)+ seventeen;

//

return seventeen;

}

//这个函数集用于将十七进制元基数字进行元基变换

public String seventeenToIntons(String seventeen, DecadeToPDS decadeToPDS) {

String initons= "";

for(int i= 0; i< seventeen.length(); i++) {

initons+= decadeToPDS.initonsCode.get(""+ seventeen.charAt(i));

}

//

return initons;

}

//这个函数集用于将元基变换进行肽展概率丝化展开

public String initonsToPDS(String initons, double pDE\_KEY\_rate, DecadeToPDS decadeToPDS) {

String PDS= "";

StringBuilder PDEKey= new StringBuilder("");

for(int i= 0; i< initons.length(); i++) {

PDS+= new PDE\_PDS\_DL().initonPDSwithBYS(""+ initons.charAt(i), pDE\_KEY\_rate, PDEKey, true)+ ".";

}

//System.out.println("生成钥匙："+ PDEKey);

//

return PDS;

}

//这个函数用于将肽展丝化的肽增十七进制进行元基变换

public String PDSToInitons(String pDS, double pDE\_KEY\_rate, DecadeToPDS decadeToPDS) {

String initons= "";

//initons= new PDE\_PDS\_DL().initonPDIwithBYS(pDS, 0, new StringBuilder(), false);

//initons= new PDE\_PDS\_DL().initonPDEwithBYS(pDS, pDE\_KEY\_rate, new StringBuilder(), true);

initons= new PDE\_PDS\_DL().initonPDE\_DCDLwithBYS(pDS, pDE\_KEY\_rate, new StringBuilder(), true);

return initons;

}

//这个函数用于将肽展丝化的肽增十七进制变换成元基数字

public String initonsToSeventeen(String initons, DecadeToPDS decadeToPDS) {

String seventeen= "";

//

for(int i= 0; i< initons.length(); i++) {

seventeen+= decadeToPDS.initonsMap.get(""+ initons.charAt(i));

}

return seventeen;

}

//这个函数用于十七进制元基数字进行十进制还原。

public int seventeenToDecade(String seventeen, DecadeToPDS decadeToPDS) {

int decade= 0;

//A11 10\*17\*17 + 1\*17 + 1

for(int i= 0; i< seventeen.length(); i++) {

int value= decadeToPDS.numberSet.get(""+ seventeen.charAt(i)).intValue();

decade+= value\* Math.pow(17, seventeen.length()- 1- i);

}

return decade;

}

//这个函数用于十七进制元基数字进行十进制矩阵变换。

public int[][] doPDSMatrix(DecadeToPDS decadeToPDS, int[][] rp, double facx) {

for(int i= 0; i< rp.length; i++) {

for(int j= 0; j< rp[0].length; j++) {

rp[i][j]= decadeToPDS.doPDS(decadeToPDS, rp[i][j], facx);

}

}

return rp;

}

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package OSI.SSI.ASU.OSU.PSU.MSU.pde;

import java.util.HashMap;

import java.util.Map;

import OSI.SSI.ASU.OSU.PSU.MSU.pds.PDE\_PDS\_DL16;

//这个函数集用于将常数变换成十六进制元基数字，

//这个函数集用于将十六进制元基数字进行元基变换

//这个函数集用于将元基变换进行肽展概率丝化展开

//这个函数用于将肽展丝化的肽增十六进制进行元基变换

//这个函数用于将肽展丝化的肽增十六进制变换成元基数字

//这个函数用于十六进制元基数字进行十进制还原。

public class DecadeToPDS16{

//思想：肽展公式 1.2.2，元基数字逻辑； 十六进制元基组合数学；概率论

//算法：进制计算，肽展计算

//程序员： 罗瑶光，

public Map<String, String> initonsMap= new HashMap<>();

public Map<String, String> initonsCode= new HashMap<>();

public Map<String, String> initonsSet= new HashMap<>();

public Map<String, Integer> numberSet= new HashMap<>();

public static void main(String[] Args) {

DecadeToPDS16 decadeToPDS= new DecadeToPDS16();

decadeToPDS.IV\_(decadeToPDS);

int decade= (int)(Math.random()\*1000 % 256);//随便写一个数

double pDE\_KEY\_rate= 0.25;//随便模拟一个0-1之间的概率钥匙，假设 0~0.5为酸

//，0.5~1 为碱；

decadeToPDS.doPDS(decadeToPDS, decade, pDE\_KEY\_rate);

System.out.println(1);

}

// //元基符号变元基数字

// //System.out.println("输入十进制数："+ decade);

// String seventeen= decadeToPDS.decadeToSeventeen(decade, decadeToPDS);

// //System.out.println("元基进制数为："+ seventeen);

// String initons= decadeToPDS.seventeenToIntons(seventeen, decadeToPDS);

// //System.out.println("变换为元基："+initons);

// //initons= "AOPMVE";

// //System.out.println("输入元基："+ initons);

// //System.out.println("输入概率："+ pDE\_KEY\_rate);

// String pDS= decadeToPDS.initonsToPDS(initons, pDE\_KEY\_rate, decadeToPDS);

// //System.out.println("输出肽丝:"+ pDS);

// pDS= pDS.replace(".", "");

// String pDSInitons= decadeToPDS.PDSToInitons(pDS, pDE\_KEY\_rate, decadeToPDS);

// //System.out.println("肽丝增元:"+ pDSInitons);

// //第二卷的肽展公式 可以用到了

// //String pDEInitons= decadeToPDS.PDSToPDE(pDSInitons, pDE\_KEY\_rate, decadeToPDS);

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

// String pDSSeventeen= decadeToPDS.initonsToSeventeen(pDSInitons, decadeToPDS);

// //System.out.println("元基数字:"+ pDSSeventeen);

// String pDSDecade= decadeToPDS.seventeenToDecade(pDSSeventeen, decadeToPDS);

// //System.out.println("输出十进制数:"+ pDSDecade);

// return Integer.valueOf(pDSDecade).intValue();

private int doPDS(DecadeToPDS16 decadeToPDS, int decade, double pDE\_KEY\_rate) {

String sixteen= decadeToPDS.decadeToSixteen(decade, decadeToPDS);

String initons= decadeToPDS.sixteenToIntons(sixteen, decadeToPDS);

String pDS= decadeToPDS.initonsToPDS(initons, pDE\_KEY\_rate, decadeToPDS);

pDS= pDS.replace(".", "");

String pDSInitons= decadeToPDS.PDSToInitons(pDS, pDE\_KEY\_rate, decadeToPDS);

String pDSSixteen= decadeToPDS.initonsToSixteen(pDSInitons, decadeToPDS);

int pDSDecade= decadeToPDS.sixteenToDecade(pDSSixteen, decadeToPDS);

return pDSDecade;

}

public void IV\_(DecadeToPDS16 decadeToPDS) {

decadeToPDS.initonsMap.put("A", "9");

decadeToPDS.initonsMap.put("O", "6");

decadeToPDS.initonsMap.put("P", "2");

decadeToPDS.initonsMap.put("M", "8");

decadeToPDS.initonsMap.put("V", "C");

decadeToPDS.initonsMap.put("E", "3");

decadeToPDS.initonsMap.put("C", "1");

decadeToPDS.initonsMap.put("S", "7");

decadeToPDS.initonsMap.put("I", "D");

decadeToPDS.initonsMap.put("D", "0");

decadeToPDS.initonsMap.put("U", "E");

decadeToPDS.initonsMap.put("Q", "F");

decadeToPDS.initonsMap.put("T", "4");

decadeToPDS.initonsMap.put("X", "A");

decadeToPDS.initonsMap.put("F", "B");

decadeToPDS.initonsMap.put("H", "5");

//元基数字变元基符号

decadeToPDS.initonsCode.put("0", "D");

decadeToPDS.initonsCode.put("1", "C");

decadeToPDS.initonsCode.put("2", "P");

decadeToPDS.initonsCode.put("3", "E");

decadeToPDS.initonsCode.put("4", "T");

decadeToPDS.initonsCode.put("5", "H");

decadeToPDS.initonsCode.put("6", "O");

decadeToPDS.initonsCode.put("7", "S");

decadeToPDS.initonsCode.put("8", "M");

decadeToPDS.initonsCode.put("9", "A");

decadeToPDS.initonsCode.put("A", "X");

decadeToPDS.initonsCode.put("B", "F");

decadeToPDS.initonsCode.put("C", "V");

decadeToPDS.initonsCode.put("D", "I");

decadeToPDS.initonsCode.put("E", "U");

decadeToPDS.initonsCode.put("F", "Q");

//阿拉伯数字变元基数字

decadeToPDS.initonsSet.put("0", "0");

decadeToPDS.initonsSet.put("1", "1");

decadeToPDS.initonsSet.put("2", "2");

decadeToPDS.initonsSet.put("3", "3");

decadeToPDS.initonsSet.put("4", "4");

decadeToPDS.initonsSet.put("5", "5");

decadeToPDS.initonsSet.put("6", "6");

decadeToPDS.initonsSet.put("7", "7");

decadeToPDS.initonsSet.put("8", "8");

decadeToPDS.initonsSet.put("9", "9");

decadeToPDS.initonsSet.put("10", "A");

decadeToPDS.initonsSet.put("11", "B");

decadeToPDS.initonsSet.put("12", "C");

decadeToPDS.initonsSet.put("13", "D");

decadeToPDS.initonsSet.put("14", "E");

decadeToPDS.initonsSet.put("15", "F");

//元基数字变阿拉伯数字

decadeToPDS.numberSet.put("0", 0);

decadeToPDS.numberSet.put("1", 1);

decadeToPDS.numberSet.put("2", 2);

decadeToPDS.numberSet.put("3", 3);

decadeToPDS.numberSet.put("4", 4);

decadeToPDS.numberSet.put("5", 5);

decadeToPDS.numberSet.put("6", 6);

decadeToPDS.numberSet.put("7", 7);

decadeToPDS.numberSet.put("8", 8);

decadeToPDS.numberSet.put("9", 9);

decadeToPDS.numberSet.put("A", 10);

decadeToPDS.numberSet.put("B", 11);

decadeToPDS.numberSet.put("C", 12);

decadeToPDS.numberSet.put("D", 13);

decadeToPDS.numberSet.put("E", 14);

decadeToPDS.numberSet.put("F", 15);

}

// //准备集成第二卷的AOPM 级别 肽展公式 ，已经并入PDSToInitons 函数中

// private String PDSToPDE(String pds, double pDE\_KEY\_rate, DecadeToPDS

//decadeToPDS) {

//

////pds= pds.replace("UQ", "V");

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

////pds= pds.replace("IQ", "S");

////pds= pds.replace("VS", "A");

////pds= pds.replace("ES", "O");

////pds= pds.replace("EC", "P");

////pds= pds.replace("CS", "M");

////pds= pds.replace("VE", "T");

////pds= pds.replace("VC", "X");

////

// return pds;

// }

//这个函数集用于将常数变换成十六进制元基数字，

public String decadeToSixteen(int decade, DecadeToPDS16 decadeToPDS) {

String sixteen= "";

int decad= decade;

while(0< decad/ 16) {

int sixteenth= decad% 16;

sixteen= decadeToPDS.initonsSet.get(""+ sixteenth)+ sixteen;

decad= decad/ 16;

}

sixteen= decadeToPDS.initonsSet.get(""+ decad)+ sixteen;

//

return sixteen;

}

//这个函数集用于将十六进制元基数字进行元基变换

public String sixteenToIntons(String sixteen, DecadeToPDS16 decadeToPDS) {

String initons= "";

for(int i= 0; i< sixteen.length(); i++) {

initons+= decadeToPDS.initonsCode.get(""+ sixteen.charAt(i));

}

//

return initons;

}

//这个函数集用于将元基变换进行肽展概率丝化展开

public String initonsToPDS(String initons, double pDE\_KEY\_rate, DecadeToPDS16

decadeToPDS) {

String PDS= "";

StringBuilder PDEKey= new StringBuilder("");

for(int i= 0; i< initons.length(); i++) {

PDS+= new PDE\_PDS\_DL16().initonPDSwithBYS(""+ initons.charAt(i)

, pDE\_KEY\_rate, PDEKey, true)+ ".";

}

//System.out.println("生成钥匙："+ PDEKey);

//

return PDS;

}

//这个函数用于将肽展丝化的肽增十六进制进行元基变换， 之后利用下 PDEKey 在肽腐蚀中的作用。

public String PDSToInitons(String pDS, double pDE\_KEY\_rate, DecadeToPDS16

decadeToPDS) {

String initons= "";

//initons= new PDE\_PDS\_DL().initonPDIwithBYS(pDS, 0, new StringBuilder()

//, false);

//initons= new PDE\_PDS\_DL().initonPDEwithBYS(pDS, pDE\_KEY\_rate

//, new StringBuilder(), true);

initons= new PDE\_PDS\_DL16().initonPDE\_DCDLwithBYS(pDS, pDE\_KEY\_rate

, new StringBuilder(), true);

return initons;

}

//这个函数用于将肽展丝化的肽增十六进制变换成元基数字

public String initonsToSixteen(String initons, DecadeToPDS16 decadeToPDS) {

String sixteen= "";

//

for(int i= 0; i< initons.length(); i++) {

sixteen+= decadeToPDS.initonsMap.get(""+ initons.charAt(i));

}

return sixteen;

}

//这个函数用于十六进制元基数字进行十进制还原。

public int sixteenToDecade(String sixteen, DecadeToPDS16 decadeToPDS) {

int decade= 0;

//A11 10\*16\*16 + 1\*16 + 1

for(int i= 0; i< sixteen.length(); i++) {

int value= decadeToPDS.numberSet.get(""+ sixteen.charAt(i)).intValue();

decade+= value\* Math.pow(16, sixteen.length()- 1- i);

}

return decade;

}

//这个函数用于十六进制元基数字进行十进制矩阵变换。

public int[][] doPDSMatrix(DecadeToPDS16 decadeToPDS, int[][] rp, double facx) {

for(int i= 0; i< rp.length; i++) {

for(int j= 0; j< rp[0].length; j++) {

rp[i][j]= decadeToPDS.doPDS(decadeToPDS, rp[i][j], facx);

}

}

return rp;

}

}

PDE\_PDS\_DL, 肽展类

package OSI.SSI.ASU.OSU.PSU.MSU.pds;

//这个函数用于元基进行数字逻辑丝化变换

//思想：肽展公式，十七进制元基数字，元基数字逻辑

//作者：罗瑶光

//算法参考如下（肽展公式在离散数学中根据贝叶斯进行数字逻辑变换）

//#### 元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合)

//#### 0 = D = DD =(D, DD)

//#### E = I = I =(I)

//#### F = U = I++ OR Q-- =(I, Q)

//#### G = Q = Q =(Q)

//

//#### 1 = C = DI =(DI)

//#### 3 = E = IU, DU =(IU, DU)

//#### 4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)

//#### D = V = UQ =(UQ)

//#### 9 = S = QI =(QI)

//

//

//#### 2 = P = (IU, DU) + DI =(IUDI, DUDI)

//#### 5 = HC = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI) OR (IUDIDI

//, DUDIDI)

//#### B = HE = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU

//, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)

//#### A = O = (IU, DU) + QI =(IUQI, DUQI)

//#### 7 = A = UQQI =(UQQI)

//#### 8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI)

//OR (IUDIDIQI, DUDIDIQI)

//#### 6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI)

//OR (UQIUDIDI, UQDUDIDI)

//#### C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU,

//UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

public class PDE\_PDS\_DL {

public String initonPDSwithBYS(String initon, double bys, StringBuilder pDEKey

, boolean isBys) {

if(initon.equalsIgnoreCase("D")) {

return "D";

}

if(initon.equalsIgnoreCase("I")) {

return "I";

}

if(initon.equalsIgnoreCase("U")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "I";

}else {

pDEKey.append("1");

return "Q";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "I";

}else {

pDEKey.append("1");

return "Q";

}

}

}

if(initon.equalsIgnoreCase("Q")) {

return "Q";

}

if(initon.equalsIgnoreCase("C")) {

return "DI";

}

if(initon.equalsIgnoreCase("E")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IU";

}else {

pDEKey.append("1");

return "DU";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IU";

}else {

pDEKey.append("1");

return "DU";

}

}

}

//#### 4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)

if(initon.equalsIgnoreCase("H")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}

}

//+- 符号见 FindOulerRing 函数 的 332行。

//#### 5 = HC = ((IU, DU) OR DI) + DI

//=(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)

if(initon.equalsIgnoreCase("-")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDIDI";

}else {

pDEKey.append("1");

return "DUDIDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDIDI";

}else {

pDEKey.append("1");

return "DUDIDI";

}

}

}

//+- 符号见 FindOulerRing 函数 的 332行。

//#### B = HE = ((IU, DU) OR DI) + (IU, DU)

//=(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU

//, DUDIDU)

if(initon.equalsIgnoreCase("+")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDIIU";

}else {

pDEKey.append("1");

return "IUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "DUDIIU";

}else {

pDEKey.append("1");

return "DUDIDU";

}

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDIIU";

}else {

pDEKey.append("1");

return "IUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< bys) {

pDEKey.append("0");

return "DUDIIU";

}else {

pDEKey.append("1");

return "DUDIDU";

}

}

}

}

if(initon.equalsIgnoreCase("V")) {

return "UQ";

}

if(initon.equalsIgnoreCase("S")) {

return "QI";

}

//#### 2 = P = (IU, DU) + DI =(IUDI, DUDI)

if(initon.equalsIgnoreCase("P")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}

}

//#### A = O = (IU, DU) + QI =(IUQI, DUQI)

if(initon.equalsIgnoreCase("O")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUQI";

}else {

pDEKey.append("1");

return "DUQI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUQI";

}else {

pDEKey.append("1");

return "DUQI";

}

}

}

if(initon.equalsIgnoreCase("A")) {

return "UQQI";

}

//#### 8 = M = ((IU, DU) OR DI) + DI + QI

//=(IUDIQI, DUDIQI, DIDIQI) OR (IUDIDIQI, DUDIDIQI)

if(initon.equalsIgnoreCase("M")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDIDIQI";

}else {

pDEKey.append("1");

return "DUDIDIQI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDIDIQI";

}else {

pDEKey.append("1");

return "DUDIDIQI";

}

}

}

//#### 6 = X = UQ + ((IU, DU) OR DI) + DI

// =(UQIUDI, UQDUDI, UQDIDI) OR (UQIUDIDI, UQDUDIDI)

if(initon.equalsIgnoreCase("X")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "UQIUDIDI";

}else {

pDEKey.append("1");

return "UQDUDIDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "UQIUDIDI";

}else {

pDEKey.append("1");

return "UQDUDIDI";

}

}

}

//#### C = T = UQ + ((IU, DU) OR DI) + (IU, DU)

//=(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

if(initon.equalsIgnoreCase("T")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "UQIUDIIU";

}else {

pDEKey.append("1");

return "UQIUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "UQDUDIIU";

}else {

pDEKey.append("1");

return "UQDUDIDU";

}

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

if(Math.random()< bys) {

pDEKey.append("0");

return "UQIUDIIU";

}else {

pDEKey.append("1");

return "UQIUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< bys) {

pDEKey.append("0");

return "UQDUDIIU";

}else {

pDEKey.append("1");

return "UQDUDIDU";

}

}

}

}

return null;

}

//IUQ D CEVS H POA -+ MXT

//反向排列后如下

//TXM +- AOP H SVEC D QUI

public String initonPDIwithBYS(String pds, double bys, StringBuilder pDEKey

, boolean isBys) {

pds= pds.replace("UQIUDIIU", "T");

pds= pds.replace("UQIUDIDU", "T");

pds= pds.replace("UQDUDIIU", "T");

pds= pds.replace("UQDUDIDU", "T");

pds= pds.replace("UQIUDIDI", "X");

pds= pds.replace("UQDUDIDI", "X");

pds= pds.replace("IUDIDIQI", "M");

pds= pds.replace("DUDIDIQI", "M");

pds= pds.replace("IUDIIU", "+");

pds= pds.replace("IUDIDU", "+");

pds= pds.replace("DUDIIU", "+");

pds= pds.replace("DUDIDU", "+");

pds= pds.replace("IUDIDI", "-");

pds= pds.replace("DUDIDI", "-");

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

pds= pds.replace("IUQI", "O");

pds= pds.replace("DUQI", "O");

pds= pds.replace("IUDI", "P");

pds= pds.replace("DUDI", "P");

pds= pds.replace("IUDI", "H");

pds= pds.replace("DUDI", "H");

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

pds= pds.replace("IU", "E");

pds= pds.replace("DU", "E");

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

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

pds= pds.replace("Q", "Q");

pds= pds.replace("U", "U");

pds= pds.replace("I", "I");

return pds;

}

//用于肽展公式逐级变换

//QUI D SVEC H AOP +- TXM

public String initonPDEwithBYS(String pds, double pDE\_KEY\_rate, StringBuilder

pDEKey, boolean isBys) {

if(!isBys) {

pds= pds.replace("Q", "Q");

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

pds= pds.replace("I", "I");

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

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

pds= pds.replace("IU", "E");

pds= pds.replace("DU", "E");

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

pds= pds.replace("IUDI", "H");

pds= pds.replace("DUDI", "H");

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

pds= pds.replace("IUQI", "O");

pds= pds.replace("DUQI", "O");

pds= pds.replace("IUDI", "P");

pds= pds.replace("DUDI", "P");

pds= pds.replace("IUDIIU", "+");

pds= pds.replace("IUDIDU", "+");

pds= pds.replace("DUDIIU", "+");

pds= pds.replace("DUDIDU", "+");

pds= pds.replace("IUDIDI", "-");

pds= pds.replace("DUDIDI", "-");

pds= pds.replace("UQIUDIIU", "T");

pds= pds.replace("UQIUDIDU", "T");

pds= pds.replace("UQDUDIIU", "T");

pds= pds.replace("UQDUDIDU", "T");

pds= pds.replace("UQIUDIDI", "X");

pds= pds.replace("UQDUDIDI", "X");

pds= pds.replace("IUDIDIQI", "M");

pds= pds.replace("DUDIDIQI", "M");

return pds;

}

pds= pds.replace("Q", "Q");

pds= pds.replace("U", "U");

pds= pds.replace("I", "I");

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

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IU", "E");

}else {

pds= pds.replace("DU", "E");

}

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

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "H");

}else {

pds= pds.replace("DUDI", "H");

}

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

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUQI", "O");

}else {

pds= pds.replace("DUQI", "O");

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "P");

}else {

pds= pds.replace("DUDI", "P");

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIIU", "+");

}else {

pds= pds.replace("IUDIDU", "+");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("DUDIIU", "+");

}else {

pds= pds.replace("DUDIDU", "+");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDIDI", "-");

}else {

pds= pds.replace("DUDIDI", "-");

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIIU", "T");

}else {

pds= pds.replace("UQIUDIDU", "T");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQDUDIIU", "T");

}else {

pds= pds.replace("UQDUDIDU", "T");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIDI", "X");

}else {

pds= pds.replace("UQDUDIDI", "X");

}

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIDIQI", "M");

}else {

pds= pds.replace("DUDIDIQI", "M");

}

return pds;

}

//融合肽展公式，离散数学和数字逻辑 的元基变换

// pds= pds.replace("UQ", "V");

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

// pds= pds.replace("IQ", "S");

// pds= pds.replace("VS", "A");

// pds= pds.replace("ES", "O");

// pds= pds.replace("EC", "P");

// pds= pds.replace("CS", "M");

// pds= pds.replace("VE", "T");

// pds= pds.replace("VC", "X");

// 我的思维逻辑是先将PDS的数字逻辑和离散数学归纳识别，然后走肽展识别，最大缩短元基长度

public String initonPDE\_DCDLwithBYS(String pds, double pDE\_KEY\_rate

, StringBuilder pDEKey, boolean isBys) {

if(!isBys) {

pds= pds.replace("UQIUDIIU", "T");

pds= pds.replace("UQIUDIDU", "T");

pds= pds.replace("UQDUDIIU", "T");

pds= pds.replace("UQDUDIDU", "T");

pds= pds.replace("UQIUDIDI", "X");

pds= pds.replace("UQDUDIDI", "X");

pds= pds.replace("IUDIDIQI", "M");

pds= pds.replace("DUDIDIQI", "M");

pds= pds.replace("IUDIIU", "+");

pds= pds.replace("IUDIDU", "+");

pds= pds.replace("DUDIIU", "+");

pds= pds.replace("DUDIDU", "+");

pds= pds.replace("IUDIDI", "-");

pds= pds.replace("DUDIDI", "-");

pds= pds.replace("IUDI", "H");

pds= pds.replace("DUDI", "H");

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

pds= pds.replace("IUQI", "O");

pds= pds.replace("DUQI", "O");

pds= pds.replace("IUDI", "P");

pds= pds.replace("DUDI", "P");

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

pds= pds.replace("IU", "E");

pds= pds.replace("DU", "E");

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

pds= pds.replace("Q", "Q");

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

pds= pds.replace("I", "I");

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

//PDE

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

pds= pds.replace("ES", "O");

pds= pds.replace("EC", "P");

pds= pds.replace("HE", "+");

pds= pds.replace("HC", "-");

pds= pds.replace("VE", "T");

pds= pds.replace("VC", "X");

pds= pds.replace("CS", "M");

return pds;

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIIU", "T");

}else {

pds= pds.replace("UQIUDIDU", "T");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQDUDIIU", "T");

}else {

pds= pds.replace("UQDUDIDU", "T");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIDI", "X");

}else {

pds= pds.replace("UQDUDIDI", "X");

}

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIDIQI", "M");

}else {

pds= pds.replace("DUDIDIQI", "M");

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIIU", "+");

}else {

pds= pds.replace("IUDIDU", "+");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("DUDIIU", "+");

}else {

pds= pds.replace("DUDIDU", "+");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDIDI", "-");

}else {

pds= pds.replace("DUDIDI", "-");

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "H");

}else {

pds= pds.replace("DUDI", "H");

}

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

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUQI", "O");

}else {

pds= pds.replace("DUQI", "O");

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "P");

}else {

pds= pds.replace("DUDI", "P");

}

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IU", "E");

}else {

pds= pds.replace("DU", "E");

}

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

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

pds= pds.replace("Q", "Q");

pds= pds.replace("U", "U");

pds= pds.replace("I", "I");

//PDE

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

pds= pds.replace("ES", "O");

pds= pds.replace("EC", "P");

pds= pds.replace("HE", "+");

pds= pds.replace("HC", "-");

pds= pds.replace("VE", "T");

pds= pds.replace("VC", "X");

pds= pds.replace("CS", "M");

return pds;

}

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package OSI.SSI.ASU.OSU.PSU.MSU.pds;

//这个函数用于元基进行数字逻辑丝化变换

//思想：肽展公式，十六 十七进制元基数字，元基数字逻辑

//作者：罗瑶光

//算法参考如下（肽展公式在离散数学中根据贝叶斯进行数字逻辑变换）

//#### 元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合)

//#### 0 = D = DD =(D, DD)

//#### E = I = I =(I)

//#### F = U = I++ OR Q-- =(I, Q)

//#### G = Q = Q =(Q)

//#### 1 = C = DI =(DI)

//#### 3 = E = IU, DU =(IU, DU)

//#### 4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)

//#### D = V = UQ =(UQ)

//#### 9 = S = QI =(QI)

//#### 2 = P = (IU, DU) + DI =(IUDI, DUDI)

//#### 5 = HC = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI) OR (IUDIDI

//, DUDIDI)

//#### B = HE = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU

//, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)

//#### A = O = (IU, DU) + QI =(IUQI, DUQI)

//#### 7 = A = UQQI =(UQQI)

//

//

//#### 8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI)

//OR (IUDIDIQI, DUDIDIQI)

//#### 6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI)

//OR (UQIUDIDI, UQDUDIDI)

//#### C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU,

//UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

//根据结构式进行分析 全嘌呤肽展公式PDE PDS。罗瑶光20211201

//F全嘌呤 = E OR C OR S = E OR S OR C = (IU, DU) OR QI OR DI = (IUQIDI, DUQIDI)

// 排列方式为 活泼组合E + 稳定组合S + 中性组合C

public class PDE\_PDS\_DL16 {

public String initonPDSwithBYS(String initon, double bys, StringBuilder pDEKey

, boolean isBys) {

if(initon.equalsIgnoreCase("D")) {

return "D";

}

if(initon.equalsIgnoreCase("I")) {

return "I";

}

if(initon.equalsIgnoreCase("U")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "I";

}else {

pDEKey.append("1");

return "Q";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "I";

}else {

pDEKey.append("1");

return "Q";

}

}

}

if(initon.equalsIgnoreCase("Q")) {

return "Q";

}

if(initon.equalsIgnoreCase("C")) {

return "DI";

}

if(initon.equalsIgnoreCase("E")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IU";

}else {

pDEKey.append("1");

return "DU";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IU";

}else {

pDEKey.append("1");

return "DU";

}

}

}

//#### 4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)

if(initon.equalsIgnoreCase("H")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}

}

//+- 符号见 FindOulerRing 函数 的 332行。

//#### 5 = HC = ((IU, DU) OR DI) + DI

//=(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)

if(initon.equalsIgnoreCase("-")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDIDI";

}else {

pDEKey.append("1");

return "DUDIDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDIDI";

}else {

pDEKey.append("1");

return "DUDIDI";

}

}

}

//+- 符号见 FindOulerRing 函数 的 332行。

//#### B = HE = ((IU, DU) OR DI) + (IU, DU)

//=(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU

//, DUDIDU)

if(initon.equalsIgnoreCase("+")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDIIU";

}else {

pDEKey.append("1");

return "IUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "DUDIIU";

}else {

pDEKey.append("1");

return "DUDIDU";

}

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDIIU";

}else {

pDEKey.append("1");

return "IUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< bys) {

pDEKey.append("0");

return "DUDIIU";

}else {

pDEKey.append("1");

return "DUDIDU";

}

}

}

}

if(initon.equalsIgnoreCase("V")) {

return "UQ";

}

if(initon.equalsIgnoreCase("S")) {

return "QI";

}

//#### 2 = P = (IU, DU) + DI =(IUDI, DUDI)

if(initon.equalsIgnoreCase("P")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDI";

}else {

pDEKey.append("1");

return "DUDI";

}

}

}

//#### A = O = (IU, DU) + QI =(IUQI, DUQI)

if(initon.equalsIgnoreCase("O")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUQI";

}else {

pDEKey.append("1");

return "DUQI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUQI";

}else {

pDEKey.append("1");

return "DUQI";

}

}

}

if(initon.equalsIgnoreCase("A")) {

return "UQQI";

}

//根据结构式进行分析 全嘌呤肽展公式PDE PDS。罗瑶光20211201

//F全嘌呤 = E OR C OR S = E OR S OR C = (IU, DU) OR QI OR DI = (IUQIDI, DUQIDI)

// 排列方式为 活泼组合E + 稳定组合S + 中性组合C

if(initon.equalsIgnoreCase("F")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUQIDI";

}else {

pDEKey.append("1");

return "DUQIDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUQIDI";

}else {

pDEKey.append("1");

return "DUQIDI";

}

}

}

//#### 8 = M = ((IU, DU) OR DI) + DI + QI

//=(IUDIQI, DUDIQI, DIDIQI) OR (IUDIDIQI, DUDIDIQI)

if(initon.equalsIgnoreCase("M")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "IUDIDIQI";

}else {

pDEKey.append("1");

return "DUDIDIQI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "IUDIDIQI";

}else {

pDEKey.append("1");

return "DUDIDIQI";

}

}

}

//#### 6 = X = UQ + ((IU, DU) OR DI) + DI

// =(UQIUDI, UQDUDI, UQDIDI) OR (UQIUDIDI, UQDUDIDI)

if(initon.equalsIgnoreCase("X")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

return "UQIUDIDI";

}else {

pDEKey.append("1");

return "UQDUDIDI";

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

return "UQIUDIDI";

}else {

pDEKey.append("1");

return "UQDUDIDI";

}

}

}

//#### C = T = UQ + ((IU, DU) OR DI) + (IU, DU)

//=(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

if(initon.equalsIgnoreCase("T")) {

if(!isBys) {

if(Math.random()< 0.5) {

pDEKey.append("0");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "UQIUDIIU";

}else {

pDEKey.append("1");

return "UQIUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< 0.5) {

pDEKey.append("0");

return "UQDUDIIU";

}else {

pDEKey.append("1");

return "UQDUDIDU";

}

}

}else {

if(Math.random()< bys) {

pDEKey.append("0");

if(Math.random()< bys) {

pDEKey.append("0");

return "UQIUDIIU";

}else {

pDEKey.append("1");

return "UQIUDIDU";

}

}else {

pDEKey.append("1");

if(Math.random()< bys) {

pDEKey.append("0");

return "UQDUDIIU";

}else {

pDEKey.append("1");

return "UQDUDIDU";

}

}

}

}

return initon;//出现null 的脏字符串，

}

//IUQ D CEVS H POA -+ MXT

//反向排列后如下

//TXM +- AOP H SVEC D QUI

public String initonPDIwithBYS(String pds, double bys, StringBuilder pDEKey

, boolean isBys) {

pds= pds.replace("UQIUDIIU", "T");

pds= pds.replace("UQIUDIDU", "T");

pds= pds.replace("UQDUDIIU", "T");

pds= pds.replace("UQDUDIDU", "T");

pds= pds.replace("UQIUDIDI", "X");

pds= pds.replace("UQDUDIDI", "X");

pds= pds.replace("IUDIDIQI", "M");

pds= pds.replace("DUDIDIQI", "M");

pds= pds.replace("IUDIIU", "+");

pds= pds.replace("IUDIDU", "+");

pds= pds.replace("DUDIIU", "+");

pds= pds.replace("DUDIDU", "+");

pds= pds.replace("IUDIDI", "-");

pds= pds.replace("IUQIDI", "F");//增加F PDE

pds= pds.replace("DUQIDI", "F");

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

pds= pds.replace("IUQI", "O");

pds= pds.replace("DUQI", "O");

pds= pds.replace("IUDI", "P");

pds= pds.replace("DUDI", "P");

pds= pds.replace("IUDI", "H");

pds= pds.replace("DUDI", "H");

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

pds= pds.replace("IU", "E");

pds= pds.replace("DU", "E");

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

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

pds= pds.replace("Q", "Q");

pds= pds.replace("U", "U");

pds= pds.replace("I", "I");

return pds;

}

//用于肽展公式逐级变换

//QUI D SVEC H AOP +- TXM

public String initonPDEwithBYS(String pds, double pDE\_KEY\_rate, StringBuilder

pDEKey, boolean isBys) {

if(!isBys) {

pds= pds.replace("Q", "Q");

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

pds= pds.replace("I", "I");

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

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

pds= pds.replace("IU", "E");

pds= pds.replace("DU", "E");

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

pds= pds.replace("IUDI", "H");

pds= pds.replace("DUDI", "H");

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

pds= pds.replace("IUQI", "O");

pds= pds.replace("DUQI", "O");

pds= pds.replace("IUDI", "P");

pds= pds.replace("DUDI", "P");

pds= pds.replace("IUDIIU", "+");

pds= pds.replace("IUDIDU", "+");

pds= pds.replace("DUDIIU", "+");

pds= pds.replace("DUDIDU", "+");

pds= pds.replace("IUDIDI", "-");

pds= pds.replace("DUDIDI", "-");

pds= pds.replace("UQIUDIIU", "T");

pds= pds.replace("UQIUDIDU", "T");

pds= pds.replace("UQDUDIIU", "T");

pds= pds.replace("UQDUDIDU", "T");

pds= pds.replace("UQIUDIDI", "X");

pds= pds.replace("UQDUDIDI", "X");

pds= pds.replace("IUDIDIQI", "M");

pds= pds.replace("DUDIDIQI", "M");

return pds;

}

pds= pds.replace("Q", "Q");

pds= pds.replace("U", "U");

pds= pds.replace("I", "I");

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

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IU", "E");

}else {

pds= pds.replace("DU", "E");

}

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

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "H");

}else {

pds= pds.replace("DUDI", "H");

}

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

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUQI", "O");

}else {

pds= pds.replace("DUQI", "O");

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "P");

}else {

pds= pds.replace("DUDI", "P");

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIIU", "+");

}else {

pds= pds.replace("IUDIDU", "+");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("DUDIIU", "+");

}else {

pds= pds.replace("DUDIDU", "+");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDIDI", "-");

}else {

pds= pds.replace("DUDIDI", "-");

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIIU", "T");

}else {

pds= pds.replace("UQIUDIDU", "T");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQDUDIIU", "T");

}else {

pds= pds.replace("UQDUDIDU", "T");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIDI", "X");

}else {

pds= pds.replace("UQDUDIDI", "X");

}

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIDIQI", "M");

}else {

pds= pds.replace("DUDIDIQI", "M");

}

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUQIDI", "F");//增加F PDE

}else {

pds= pds.replace("DUQIDI", "F");

}

return pds;

}

//融合肽展公式，离散数学和数字逻辑 的元基变换

// pds= pds.replace("UQ", "V");

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

// pds= pds.replace("IQ", "S");

// pds= pds.replace("VS", "A");

// pds= pds.replace("ES", "O");

// pds= pds.replace("EC", "P");

// pds= pds.replace("CS", "M");

// pds= pds.replace("VE", "T");

// pds= pds.replace("VC", "X");

// 我的思维逻辑是先将PDS的数字逻辑和离散数学归纳识别，然后走肽展识别，最大缩短元基长度

public String initonPDE\_DCDLwithBYS(String pds, double pDE\_KEY\_rate

, StringBuilder pDEKey, boolean isBys) {

if(!isBys) {

pds= pds.replace("UQIUDIIU", "T");//UQIUDIIU

pds= pds.replace("UQIUDIDU", "T");

pds= pds.replace("UQDUDIIU", "T");

pds= pds.replace("UQDUDIDU", "T");

pds= pds.replace("UQIUDIDI", "X");

pds= pds.replace("UQDUDIDI", "X");

pds= pds.replace("IUDIDIQI", "M");

pds= pds.replace("DUDIDIQI", "M");

pds= pds.replace("IUDIIU", "+");

pds= pds.replace("IUDIDU", "+");

pds= pds.replace("DUDIIU", "+");

pds= pds.replace("DUDIDU", "+");

pds= pds.replace("IUDIDI", "-");

pds= pds.replace("DUDIDI", "-");

pds= pds.replace("IUQIDI", "F");//增加F PDE

pds= pds.replace("DUQIDI", "F");

pds= pds.replace("IUDI", "H");

pds= pds.replace("DUDI", "H");

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

pds= pds.replace("IUQI", "O");

pds= pds.replace("DUQI", "O");

pds= pds.replace("IUDI", "P");

pds= pds.replace("DUDI", "P");

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

pds= pds.replace("IU", "E");

pds= pds.replace("DU", "E");

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

pds= pds.replace("Q", "Q");

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

pds= pds.replace("I", "I");

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

//PDE

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

pds= pds.replace("ES", "O");

pds= pds.replace("EC", "P");

pds= pds.replace("HE", "+");

pds= pds.replace("HC", "-");

pds= pds.replace("VE", "T");

pds= pds.replace("VC", "X");

pds= pds.replace("CS", "M");

return pds;

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIIU", "T");

}else {

pds= pds.replace("UQIUDIDU", "T");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQDUDIIU", "T");

}else {

pds= pds.replace("UQDUDIDU", "T");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("UQIUDIDI", "X");

}else {

pds= pds.replace("UQDUDIDI", "X");

}

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIDIQI", "M");

}else {

pds= pds.replace("DUDIDIQI", "M");

}

if(Math.random()< pDE\_KEY\_rate) {

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUDIIU", "+");

}else {

pds= pds.replace("IUDIDU", "+");

}

}else {

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("DUDIIU", "+");

}else {

pds= pds.replace("DUDIDU", "+");

}

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDIDI", "-");

}else {

pds= pds.replace("DUDIDI", "-");

}

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUQIDI", "F");//增加F PDE

}else {

pds= pds.replace("DUQIDI", "F");

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "H");

}else {

pds= pds.replace("DUDI", "H");

}

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

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IUQI", "O");

}else {

pds= pds.replace("DUQI", "O");

}

if(Math.random()< pDE\_KEY\_rate) {

pds= pds.replace("IUDI", "P");

}else {

pds= pds.replace("DUDI", "P");

}

pds= pds.replace("QI", "S");

pds= pds.replace("UQ", "V");

if(Math.random()<pDE\_KEY\_rate) {

pds= pds.replace("IU", "E");

}else {

pds= pds.replace("DU", "E");

}

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

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

pds= pds.replace("Q", "Q");

pds= pds.replace("U", "U");

pds= pds.replace("I", "I");

//PDE

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

pds= pds.replace("ES", "O");

pds= pds.replace("EC", "P");

pds= pds.replace("HE", "+");

pds= pds.replace("HC", "-");

pds= pds.replace("VE", "T");

pds= pds.replace("VC", "X");

pds= pds.replace("CS", "M");

return pds;

}

}

1. 全嘌呤的推导

今天开始写 RNA 芯片与肽逻辑 001，准备抛除一切杂念。

首先把昨天的 补码元基逐步简化的归纳 纸页笔记进行 电脑保存。

花了 2 年时间写完第三次修订版本的 DNA 元基催化与肽计算 一书， 我已经有一定的经验如下

1 肽展公式

S= I,

S= Q,

C= D

I= !D,

D= !I,

U= !Q,

Q= !U

I= ++D,

U= ++I,

Q= ++U,

DD= ++Q

V= U+ Q,

E= I+ U,

C= I+ D,

S= I+ Q

A= V+ S,

O= E+ S,

P= E+ C,

M= C+ S,

E= D+ U

D= DD,

U= E,

I= U,

E= I+ E,

P= P+ D,

C= U+ D+ D

2 十六元基

AOPM TXH VECS IDUQ DD

昨天基于这 1 和 2， 我分析了下 类嘌呤 和 类嘧啶的结构以及 类甾体结构，得到一个结论

关于 AOPMTX 的甾体弧结构，首先我开始统计其变化种类，得到如下归纳，

如 第三次修订版本的 DNA 元基催化与肽计算的 第 638 页所示 AOPM-X INITON

如果 NH2 为位置 1， O 为位置 2， NH2 对应的 H2N 为位置 3， O 对应的 NH2 为位置 4

则 X 的标记为 N O N N, 用这种归纳法，我把所有的类甾体可归纳为如下

NN

NO

OO

NNN

NN N

NNO

NN O

NON

NO N

NOO

NO O

OON

OO N

OOO

OO O

NNNN

NNNO

NNON

NNOONONN

NONO

NOON

NOOO

ONNN

ONNO

ONON

ONOO

OONN

OONO

OOON

OOOO

然后我进行了对称的过滤，把 OONO OOON, ONOO NOOO, NNON NNNO, ONNN NONN 四组对称的过滤一半，

然后把第 3 和 4 位为 O 的无意义过滤掉得到如下标记集合

NN

NO

OO

NNN

NN N

NON

NO N

OON

OO N

NNNN

NONN

OONN

再过滤掉对称的 NNN NN N, 得到如下标记

NN

NO

OO

NNN

NON

NO N

OON

OO N

NNNN

NONN

OONN

于是我开始分组

I， O O

D， N O

U， N O

Q， O O

V， N N

E， OC， O N

S， N

H， O O

P， OON

A， NNN

O， NO

？, OO

？, NN

M，NO N

T， NON

？, OONN

X， ONNN

？, NNNN

于是思路便清晰了，DD 补码的结构如 NnOo， 其关联的元基仅有 H, O O, 补码的逐步碱化可以转化为 V, N N ->

A, NNN -> ?, NNNN 我得到一个答案这四个 OONN , NNNN, OO, NN 属于 RNA 的计算过程元基产物。下一步谜题便揭开了帷幕。 上面是 20210905 的笔记，图片已经开源。今天要做的准备开始笔记研发。

在得知 DD 补码的结构如 NnOo， 我开始更进研究。 首先，我得到一些价值信息， 如 DD 补码在持续的碱化能得到 DD-> VVS -> A 的元基过程。 因为酸的 H, O O 不稳定无意义，我先展开 DD 的可探测的类型推导。 于是我得到下面 5 中 模型归纳 离子肽 对

1 D-D 嘧啶对

2 氨基黄嘌呤-氨基黄嘌呤 嘌呤对

3 氨基黄嘌呤-D 碱基对 分子肽

4 氨基黄嘌呤-D 分子

5 氨基黄嘌呤-氨基黄嘌呤 类似甾体分子

我推测这个 4 和 5 是一种不稳定的 RNA 中间过程 元基。

于是我根据这 5 种模型开始探索能模拟补码，二次补码的有效结构。

首先我用 D-D 嘧啶对做计算

常见的弱碱种类有 HCO3-, O-, CH3-, NH2-

补码的甲基化有效果，可是反码的实现就有问题。

常见的弱酸种类有 Na+，K+，H+, NO2+, 我得到一个信息：一些微量金属元素离子参与了酸化反应。

显然离子对 可以参与 RNA 计算过程，但不是有效的表达补码计算的主要化合物。于是我开始关注 氨基黄嘌呤-氨基黄嘌呤 类似甾体分子，准备画图观测。

今天下午在思考氨基黄嘌呤的执行补码过程， 我从两点开始行动

1 酸碱变化

2 肽展公式属性

有了行动点，我尝试找一种代号 来缩写这个嘌呤 首先我定义为氨黄，DNA 元基编码与催化计算已经有了 H 的 HE 和 HC 效用。 于是我开始观测氨黄， 氨黄的效用 能同理实现氨黄 V 和 氨黄 S， 既有感知 有又腺，静态的语义表达，我一开始定义为接触。Touch 又具备 H 的执行和控制语义表达，如酮基， 似乎很全面，我改为全嘌呤。Full 于是我定义氨基黄嘌呤 为 RNA -F 元基。 全嘌呤， 一开始我定义为补嘌呤，但是不好听， 还是定义全嘌呤 F 元基。

因为 F 元基在不同的环境能参与所有嘌呤的替代反应，我推断其必定是 RNA 的核心计算元基。 通过 DNA 元基编码与催化计算的第 639 页，可以发现 RNA F 元基能取代 DNA 的 E 元基 做补码计算 的碱基对表达锁存计算信号。

于是我得到 2 个论点

1 氨基黄嘌呤碱基对锁存计算信号。

F, DU = FD, FU

2 氨基黄嘌呤类的甾体分子 参与补码计算。

稍后准备开始论证.

昨晚搜了下百度， 氨基黄嘌呤有很多名称， 如 2 羟基腺嘌呤， 酮基腺嘌呤，6 氨基黄嘌呤，我取名为全嘌呤。

今天开始分析 全嘌呤碱基对 和 全嘌呤类甾体分子的 电势差，更好的确定 高电位和低电位，实现 2 进制的 1 和 0.

同时观测催化反应的 逻辑表达方式。 探索其触发器和锁存器的构建模式。 开始分析电势差，于是我设计了四种比较直白的可观测模式

1 {D, D} 嘧啶元基对

2 {F, DU} 碱基元基对

3 {F, F} 嘌呤元基对

4 {FF} 类甾体分子元基

通过观测可表观理解，可以得知 1 2 和 4 是相对比较稳定的结构。3 因为分子大，而离子键 相对 1 和 2 的引力要

弱。

于是我得到一些结论

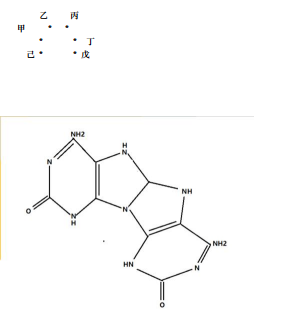
1 ｛F, F｝ 组合 相对其他活性活泼。

2 {FF} 类甾体分子元基 的离子组合 繁多，补码吸附逻辑复杂。

3 通过把{FF} 类甾体分子元基的 吸附面 定义为 甲乙丙丁戊己 6 个面， 发现类似一个马口蹄铁的磁石形状， 戊己

靠近，吸附力强度高。

甲乙丙丁散开，吸附力弱， 这里产生电势差倾斜， 可以有效的生成 高电位和低电位的表达方式。



可观测类型如下

甲｛ON,ON｝

乙｛ON｝

丙｛ON｝

丁｛ON,ON｝

戊｛ON｝

己｛ON｝

于是我 开始分开思考 马蹄的可吸附模式 和离子组合模式。今天是非常有意义的一天，我论证了 rna 芯片的计算实质: RNA 小分子肽团 的多种电磁频率 组合 方式 表达 驱动

计算信号。

根据昨天的 FF 甾体 进行结构观测， 和马口蹄铁 甾体 的 电磁吸附补码 推导。我设计了 8 个不同的甾体组合

根据 设计的甲乙丙丁戊己 6 个吸附面 我可以分析成

1 类甾体-嘧啶 补码 吸附

2 类甾体-嘌呤 补码 吸附

3 双类 甾体甲 补码吸附

4 双类 甾体甲

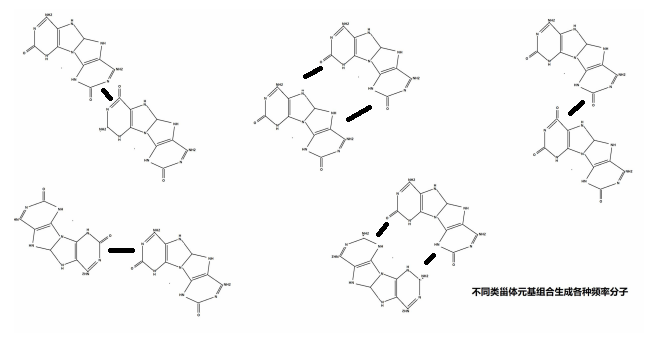
补码吸附 rotation

5 双类 甾体乙丙 背 补码吸附

6 双类 甾体丙 补码吸附

7 双类 甾体戊己 O 型吸附

8 双类 甾体丙 对称吸附



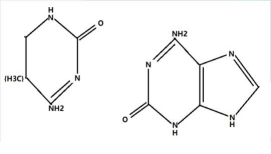
我得到一个结论 DNA 是函数的预先语义表达的函数信号锁存， RNA 的计算模式是神经网络拓扑模式，不是晶振指令周期的驱动，所以RNA 的长度和 结构不规则决定了电势和固有电磁频率的种类。这些种类的组合驱动生命应激表达的行为。这 8 种结构中，1 2 3 4 6 具备了链式拉长 来 改变固有频率。另外 通过对｛F,DU｝｛F,IQ｝ 两种全嘌呤碱基对的电势和离子活性观测，发现

活性

｛F,DU｝ 小于｛F,IQ｝

稳定性

｛F,DU｝ 大于｛F,IQ｝

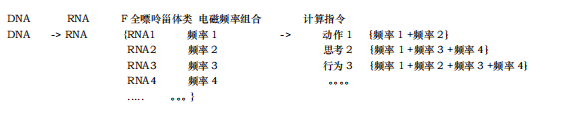


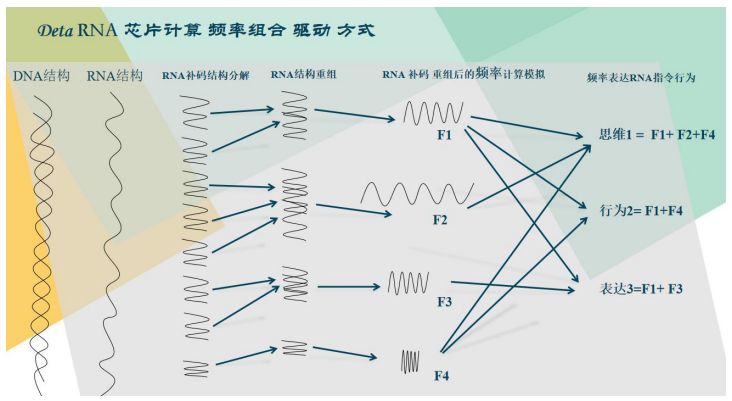
仍有一个问题困扰着我，U Q 的 甲基的语义意思不是很明确。

RNA 的吸附种类结构决定了电离电磁频率，这些频率组合方式系统完成不同的功能和应激表达。

rna 计算是一种神经网络的电磁频率组合综合应激表达计算，与市面的常规电脑芯片的晶振指令周期计算完全不同。

逻辑如下





我得到一些实质结论： DNA 的螺旋结构 只是一个函数的存储 和函数将要表达前的预先时序排列的信号描述。 一旦进行了 RNA 驱动计算， 计算过程中就无意义 RNA 的表达不稳定，因为电离不稳定，环境不稳定，很多因素不稳定，所以 RNA 的计算结果取值一定是一种神经元基网络 加概率论打分的计算过程。 神经元基网络见 DNA 元基催化与肽计算 039009 版本的第 695 页， 罗瑶光先生的思想很简单， 客观一切可推导，主观一切可描述的普遍存在的价值取向。这种思想可以和任何思维和逻辑观念耦合。非常方便罗瑶光先生的研发导向。举个例子 昨天设计了 RNA 小分子肽团 的多种电磁频率 组合 方式 表达 驱动计算信号。

按照客观一切可以推导的思维， 我开始设计推导逻辑如下

1 首先我已经掌握了数字逻辑结构的 补码计算 如下 4 个例子 1-3,3-1

1-3 补码逻辑

00000001 1

00000011 -3

00000001 1

11111100 +3！

00000001 +1 补码

11111110 0 carry 准备 2 次补码

00000001 ！

00000001 +1 补码

00000010

= -2

3-1 补码逻辑

00000011 3

00000001 -1

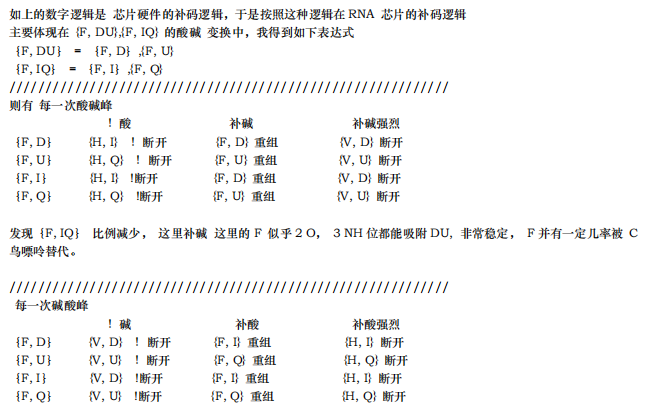
00000011 3

11111110 +1！

00000001 +1 补码

00000010 1 carry 不准备 2 次补码

= 2



发现｛F, DU｝ 比例减少 ， 这里补酸 这里的 F 似乎 是 3 NH 与 IQ 组合，不够稳定， F 并有一定几率被 C 鸟 嘌呤替代。

我得到一个 F 嘌呤 在酸碱峰来临时候会 全部断开的过程， 断开便意味着 饱和重组。 最后这些重组的大小规则长度都不一的新 RNA 拥有各自固有的电磁频率。 参与频率叠加组合相应的神经驱动钥匙， 完成相应的 RNA 计算逻辑。 前年在设计 养疗经的 智能声诊的 组件页 时候，我开源了 用傅里叶变换来提取人类发音的 元音 A O E I U 五个元音的训练集合 并生成 text 的训练文档， 分别包含了 5 个元音的 周期 频率对应的 振动叠加 信号 。

电磁与 物理简谐振动 通用这种叠加算法。我对叠加算法的应用描述是

1 声音的元音 等通过周期频率的 振动 分解 组合进行识别人类发音。

2 意识的元基同理通过周期频率的电磁信号分解组合识别人类的应激行为。

我记录声音的方法比较简单， 用素数来 描述训练集

如

周期 PI \*K\*｛1+ 1/2 + 1/3 + 1/5 + 1/7 + 1/11 + 1/13 + 1/17 + 1/19 + 1/23 + 1/29

+ 1/31 + 1/37 + 1/41 + 1/43 + 1/47 ....... ｝

我产生一个论点

元基公式在 RNA 芯片计算中是一种频率叠加组合计算公式

AOPM VECS IDUQ TXHF DD 十六元基有其各自内在的固有频率

fA fO fP fM fV fE fC fS fI fD fU fQ fT fX fH fF fDD

自然有其周期

TA TO TP TM TV TE TC TS TI TD TU TQ TT TX TH TF TDD

PDE 肽展公式 对应 这种周期的叠加 公式

TA = T{V + S}

TO = T{E + S}

TP = T{E + C}

TM = T{C + S}

TV = T{U + Q}

TE = T{I + U}

TC = T{I + D}

TS = T{I + Q}

TI = T{E - U}

TD = T{C - I}

TU = T{E - I}

TQ = T{S - I}

TDD = T{D + D} = T{FF}

TT = T{V + E}

TX = T{V + C}

TH = T{(HE + HC)/4} = T{(E + C)/2}

THE = T{H + E}

THC = T{H + C}

TII = T{I + I} = T{HH}

.......

我开始思考 长度不一， 规则不一，的各自固有频率的元基 周期叠加组合成 RNA 的频率怎么在养疗经的 常规计算机语言环境的逻辑中 进行集成这种 周期算法。 于是开始写算法包。 把笔记和 截图 进行了归纳一下，于是产生如下公式。

DNA

T{I} = T(羟基 + 羟基)

T{D} = T(羟基 + 氨基)

T{U} = T(羟基 + 氨基 + 甲基)

T{Q} = T(羟基 + 羟基 + 甲基)

T{V} = T{Q + U}

T{E} = T{I + U} = T{D + U}

T{C} = T{I + D}

T{S} = T{I + Q}

T{A} = T{V + S}

T{O} = T{E + S}

T{P} = T{E + C}

T{M} = T{C + S}

T{T} = T{V + E}

T{X} = T{V + C}

T{H} = T{(E + C) / 2}

T{HE} = T{H + E}T{HC} = T{H + C}

RNA

T{F} = T{D + D}

T{FF} = T{CC}

T{HH} = T{E + C} = T{P}

{} 为元基

（）为化合基

... ...=>T (氨基) = T (2\*羟基)

... ...=>T (甲基) = T ( ？)

f 频率 = 1 / T 周期

关于 T (氨基) = T (2\*羟基)的 推导 如下

T{HH} = T{E + C}

T(羟羟 羟羟)= T(羟 羟氨)

甲基的周期频率比，在肽展公式中没有可计算部分涉及，于是我产生 3 个论点

1 甲级作用 是掩码，用于计算。

2 甲基作用 是补码，用于负计算。

3 甲基作用 是 carry，用于正负区分。

再细看一遍，一目了然了， 甲基的作用是 carry。

准备论证， 1，2 见肽展公式 DNA 元基催化与肽计算第三修订版 039009 第 613 页

1 甲级作用 是掩码，用于计算。如果假设成立， 那么 I 的掩码等于 Q, 事实 I 的掩码是 D, 假设失败。

2 甲基作用 是补码，用于负计算。 如果假设成立， 那么 I 的补码等于 Q, 事实 I 的补码是 U, 假设失败。

3 甲基作用 是 carry，用于正负区分。没有逻辑错误。可以跟进论证。

... ...=>T (氨基) = T (2\*羟基)

... ...=>T (甲基) = f (carry 标识)

新的开始，准备整理下文档。

结论：

1 元基有其各自固有的频率公式。2 碱基对有其各自固有的频率公式。3 RNA 分子化合物有其各自 固有的频率公式。4 RNA 中全嘌呤可被鸟嘌呤替代， 因为鸟嘌呤更稳定。5 双 鸟嘌呤类甾体形成 的马口蹄磁铁与 双全嘌呤类甾体形成的马蹄口磁铁互补。6 首次出现PDE肽展公式适用的 减法 的 周期公式。7 全嘌呤的吸附面可以形成所有弧形rna 链，8 素数 K PI 叠加通用于频率的组合叠加公 式。9 甲基的意义已经确定周期频率中区别 氨基和羟基的第三种频率 标识， 目前准备测试做carry 用途。10 与或非RNA 与数字逻辑的与或非表达 完全不同， 碱基对的羟 氨 组合 类似 二 三极管的 PN 。11 元基的频率公式进行推导首先我会推导其 定义域与值域 的区间，方便更进研究。12 我得 到一个严谨的结论：元基是一种 拥有 语义表达，生化表达，电极表达，磁频率表达的 四重 基本 信号单位。同时元基也能代表一种 能量 的基本单位。

Refer

碱基对的 发现人是 J.D.Watson 和 P.H.C.Crick，

数字逻辑 的补码 思想 ，refer 其作者 冯诺依曼，作者在班加罗尔大学基督学院 数字逻辑课程系统的学习了基础。

傅里叶 频率周期叠加思想 refer 其作者 傅里叶

全嘌呤元基 TXHF-F 语义推导与定义者 罗瑶光

变嘧啶元基 IDUQ-U 语义推导与定义者 罗瑶光

RNA 的元基芯片 计算逻辑 频率组合叠加 驱动钥匙 发现者 罗瑶光

DNA 元基催化与肽计算 refer 罗瑶光

DCPE THOS MAXF VIUQ 十六进制推导

//欧拉进制

package OSI.SSI.ASU.OSU.PSU.MSU.ouler;

import java.util.HashMap;

import java.util.Map;

import SVQ.stable.StableCommon;

public class Q\_OulerRing16{

//这段函数用于观测元基映射的欧拉回路模型

//思想 罗瑶光

//算法 欧拉

//程序员 罗瑶光

//基于32行组 求接出 16进制元基为 DCPE THOS MAXF VIUQ

/\*\*

//AMSOHEPCD7 //基于该组数据求解 罗瑶光 20211129

//6

//5

//UIVT8

//FX9

//8

//7

//6

//Q6

//5

//4

//3

//2

//1

//0

\* \*/

public static void main(String[] args) {

//init AOPM VECS IDUQ TXH DD

//初始环路

Map<String, Boolean> initonsLink= new HashMap<>();

//环路探索

Map<String, Boolean> didInitonsLink= new HashMap<>();

initonsLink.put("DC", true);

initonsLink.put("CD", true);

initonsLink.put("IV", true);

initonsLink.put("VI", true);

initonsLink.put("IU", true);

initonsLink.put("UI", true);

initonsLink.put("UE", true);

initonsLink.put("EU", true);

initonsLink.put("UQ", true);

initonsLink.put("QU", true);

initonsLink.put("QS", true);

initonsLink.put("SQ", true);

initonsLink.put("VT", true);

initonsLink.put("TV", true);

initonsLink.put("ET", true);

initonsLink.put("TE", true);

initonsLink.put("EH", true);

initonsLink.put("HE", true);

initonsLink.put("EP", true);

initonsLink.put("PE", true);

// initonsLink.put("H+", true);//HE + HC -

// initonsLink.put("+H", true);

// initonsLink.put("H-", true);

// initonsLink.put("-H", true);

initonsLink.put("HC", true);

initonsLink.put("CH", true);

initonsLink.put("CP", true);

initonsLink.put("PC", true);

initonsLink.put("SM", true);

initonsLink.put("MS", true);

initonsLink.put("SO", true);

initonsLink.put("OS", true);

initonsLink.put("HF", true);

initonsLink.put("FH", true);

initonsLink.put("VF", true);

initonsLink.put("FV", true);

initonsLink.put("XA", true);

initonsLink.put("AX", true);

initonsLink.put("MA", true);

initonsLink.put("AM", true);

// initonsLink.put("X-", true);

// initonsLink.put("-X", true);

// initonsLink.put("M-", true);

// initonsLink.put("-M", true);

initonsLink.put("XF", true);

initonsLink.put("FX", true);

initonsLink.put("MF", true);

initonsLink.put("FM", true);

// initonsLink.put("T+", true);

// initonsLink.put("+T", true);

// initonsLink.put("O+", true);

// initonsLink.put("+O", true);

initonsLink.put("TH", true);

initonsLink.put("HT", true);

initonsLink.put("OH", true);

initonsLink.put("HO", true);

String[] initons= new String[]{"H", "A", "O", "P", "M", "V", "E", "C", "S"

, "I", "D", "U", "Q", "T", "X", "+", "F"};

int[] initonsCount= new int[17];

//for loop

//开始计算 路径总数

//String didInitons= "";

int count= 0;

for(int i= 0; i< initons.length; i++) {

//System.out.println(temp);

System.out.print(initons[i]);

initonsCount[i]++;

recur(initons[i], initonsLink, didInitonsLink, initons, initonsCount

, count, i);

//System.out.println(count);

//下一个

count= 0;

System.out.println();

didInitonsLink.clear();

initonsCount= new int[17];

}

// print loop initons

//打印可能模式

System.out.println(count);

}

//递归 继续修改。等会加 隔开观测。

public static void recur(String firstChar, Map<String, Boolean> initonsLink

, Map<String, Boolean> didInitonsLink, String[] initons

, int[] initonsCount, int count, int i) {

for(int j= 0; j< initons.length; j++) {

if(!firstChar.equals(initons[j])) {

String temp= StableCommon.STRING\_EMPTY+ firstChar+ initons[j];

//有路径

//没有遍历 //遍历了两次

if(initonsLink.containsKey(temp)&& !didInitonsLink.containsKey(temp)

&& initonsCount[j]< 1) {

initonsCount[j]++;

didInitonsLink.put(""+ firstChar+ initons[j], true);

System.out.print(initons[j]);

recur(initons[j], initonsLink, didInitonsLink, initons

, initonsCount, count+ 1, j);

System.out.println(count);

}

}

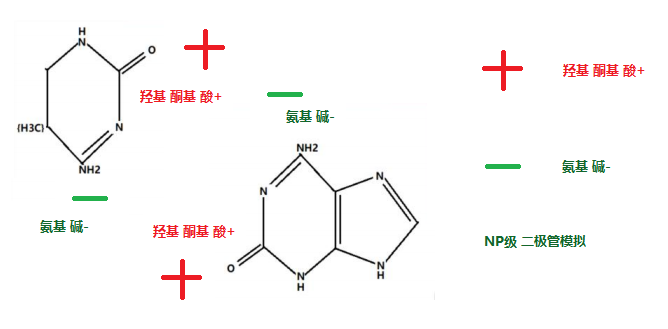
}

}

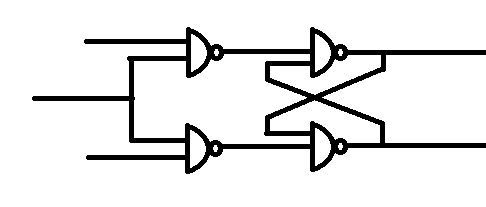
}

FU 全嘌呤变嘧啶数字锁存逻辑

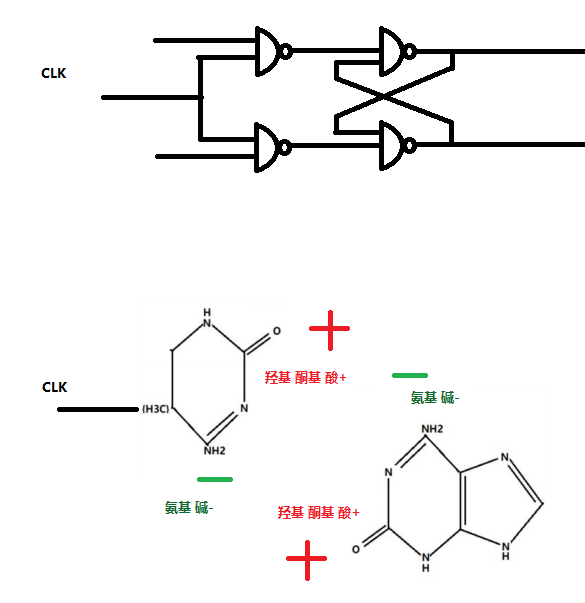
//PN级二极管三极管模拟器件



//FU锁存器件



十六元基进制的数字逻辑与离散数学发散



作者在这里有很多想法。如Cmos ， nMOS, 等逻辑图的 元基表达方式。

第十四章 DNA搜索与筛选应用

第一节 DNA搜索的动机

第二节 DNA搜索的应用需求

第三节 DNA搜索的具体描述

ZhongYaoSearch, 搜索类15

DNA搜索的应用实现

第四节 DNA筛选的动机

第五节 DNA筛选的应用需求

第六节 DNA筛选的具体描述

味觉语义元基定义

味觉生化元基定义

双元筛选索引词库

第七节DNA筛选的应用实现

第八节DNN分词 词汇花函数源码



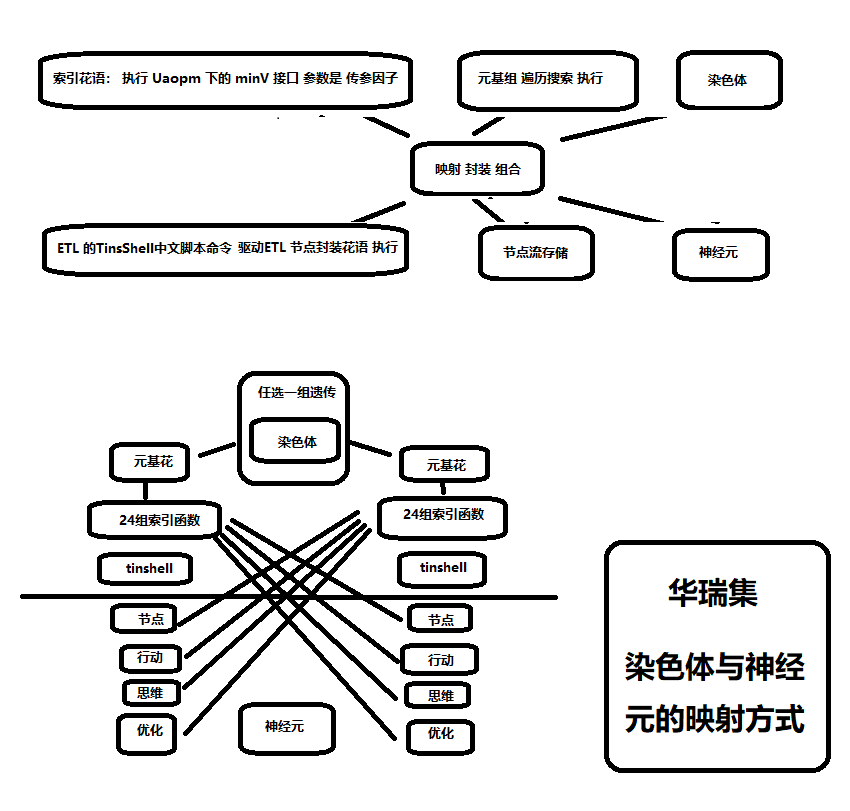
方剂森林花JOGL三维计算展示函数

药材功效花JOGL三维计算展示函数

药材禁忌花JOGL三维计算展示函数

花的筛选与观测

1. 元基模拟染色体新陈代谢催化编码



第一节 元基造字

Q\_OulerRing, 欧拉路径类

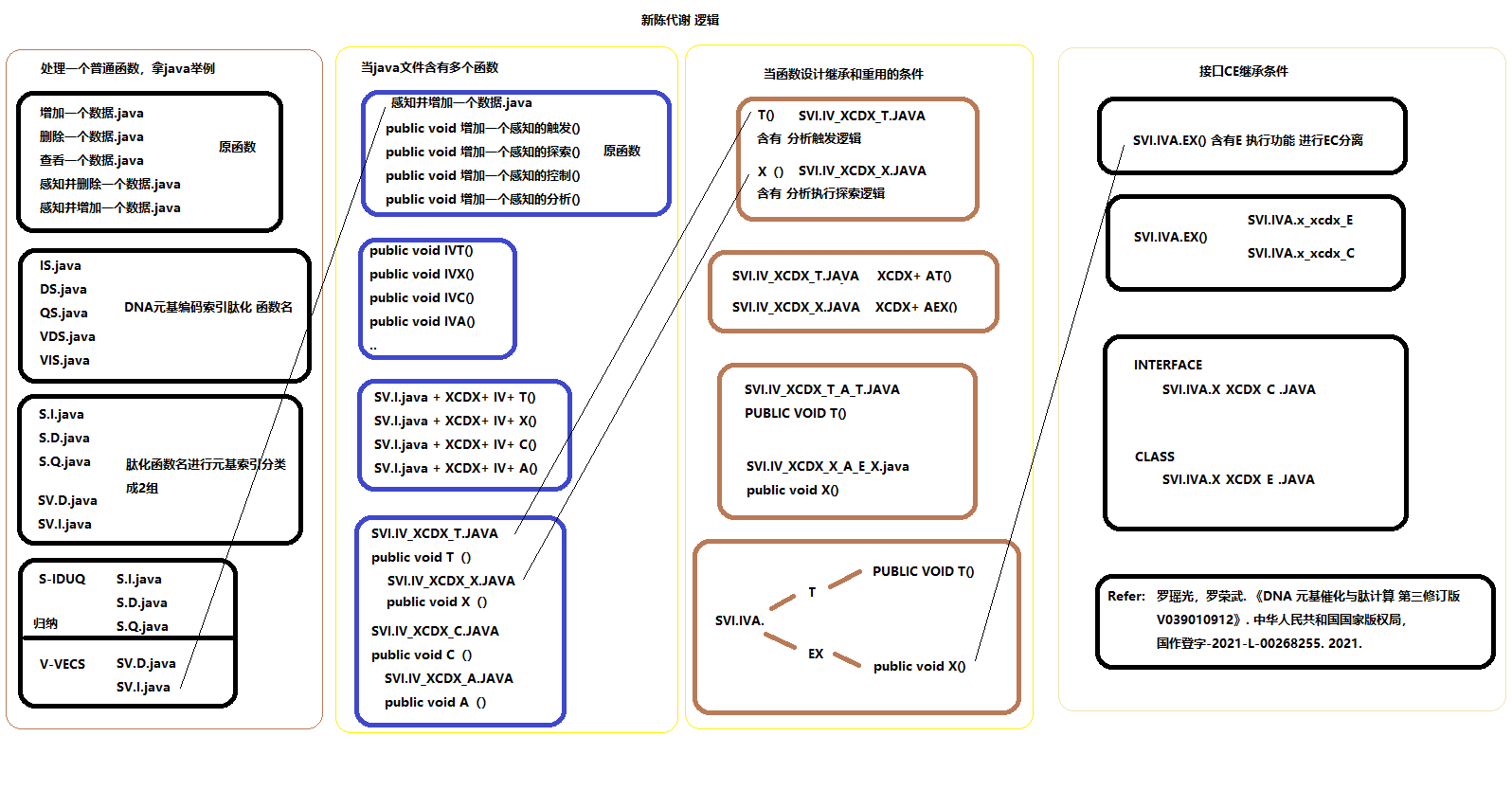
LYG9DWithDoubleTopSort4D, 极速排序算法

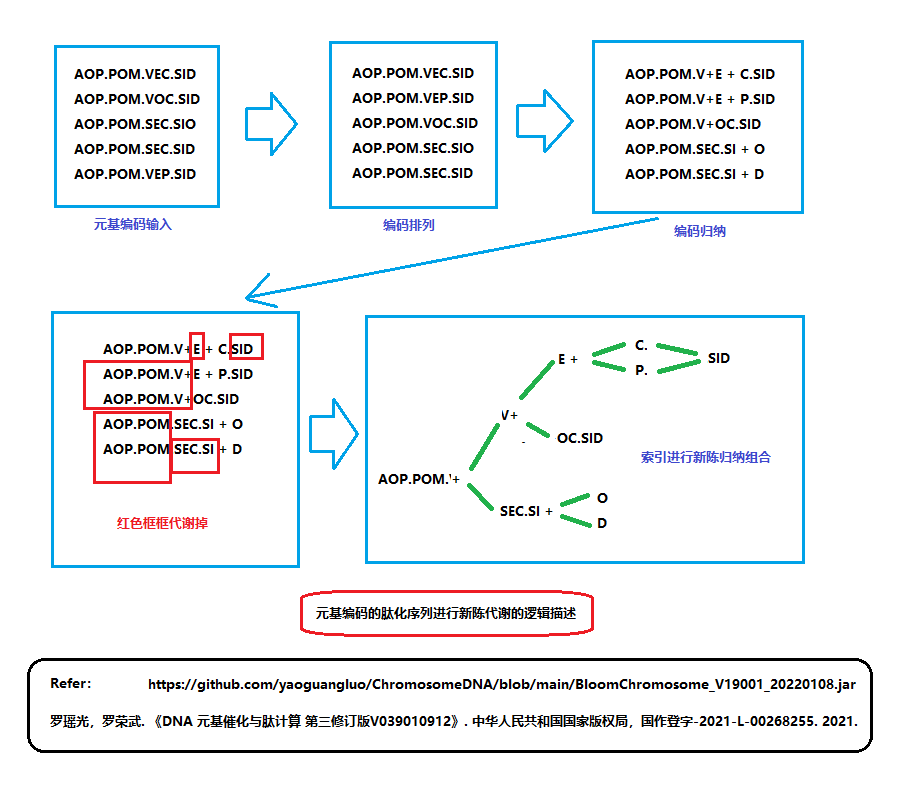
LYG9DWithDoubleTopSort4D\_U, 极速排序算法

Top Sort 5D

第二节 最新笔记 包含十六元基造字

二次元基新陈代谢方式





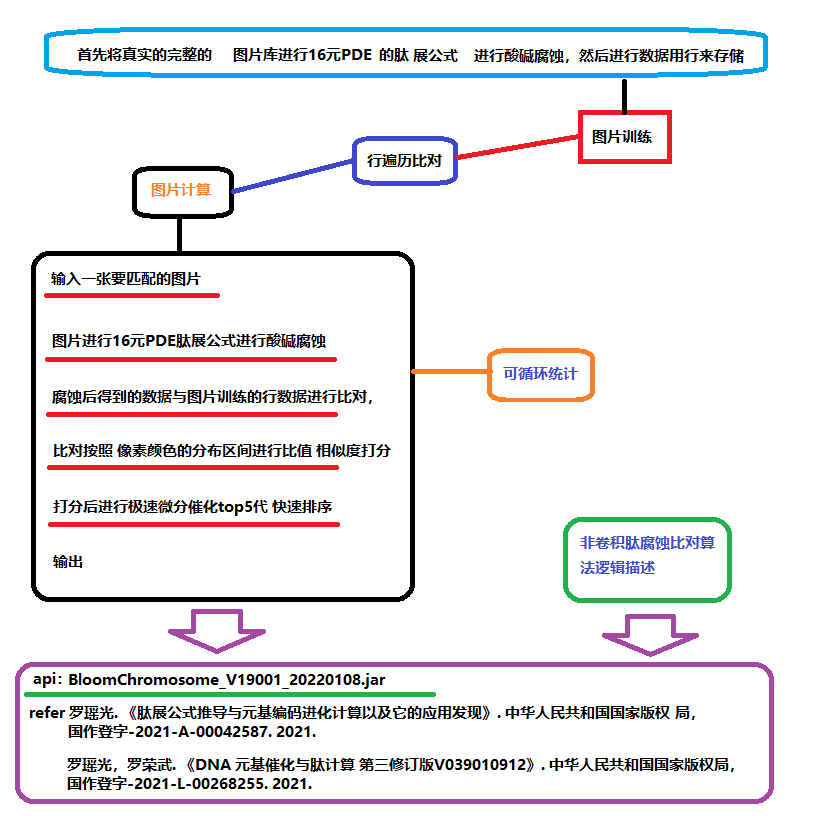
LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类

第三节 图片识别

图片读脏能力



SkinPathDetectDis, 肽展图片处理类

GetICAStatisticRatio, 肽展图片处理类

GetColorRatioScore, 肽展图片处理类

SkinPathDetectTrip, 肽展图片处理类

MakeImag, 肽展图片处理类

第四节 元基枝与元基花及其在分词，排序，索引，加密上的应用

LYG9DWithDoubleTopSort4D，极速象契混合排序

LYG10DWCMSSort15D\_XCDX\_P\_U\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_S, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_A\_C, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_S\_C, 象契字符排序类

LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_A\_C, 象契字符排序类

AE\_XCDX\_Map, 肽展中文分词类

AE, 肽展中文分词类

A, 肽展中文分词类

A\_XCDX\_Map, 肽展中文分词类

BinaryForest, 肽展分词索引类

BinaryForest\_A, 肽展分词索引类

CogsBinaryForest\_AE, 肽展分词索引类

CogsBinaryForest\_A, 肽展分词索引类

BinaryForest\_AE, 肽展分词索引类

Nlp\_CE\_XCDX\_A, 肽展分词索引类

Nlp\_C\_XCDX\_A, 肽展分词索引类

Nlp\_CE\_XCDX\_S, 肽展分词索引类

Nlp\_CE\_XCDX\_A, 肽展分词索引类

Nlp\_C\_XCDX\_S, 肽展分词索引类

Nlp\_CE\_XCDX, 肽展分词索引类

POS\_C\_Cognition\_E, 肽展分词索引类

POS\_C, 肽展分词索引类

Pos\_CE\_XCDX\_E, 肽展分词索引类

Pos\_CE\_XCDX\_O, 肽展分词索引类

Pos\_C\_XCDX\_E, 肽展分词索引类

Pos\_C\_XCDX\_O, 肽展分词索引类

Pos\_CE\_XCDX\_P, 肽展分词索引类

Pos\_CE\_XCDX\_E, 肽展分词索引类

Pos\_C\_XCDX\_P, 肽展分词索引类

Pos\_CE\_XCDX, 肽展分词索引类

1. TinShell插件\_元基花模拟染色体组计算索引系统

第一节 软件介绍

软件开发动机

软件开发目的

软件价值

软件主要功能

软件开发系统环境

硬件开发系统环境

软件开发软件环境

软件开发硬件环境

软件部署软件环境

软件部署硬件环境

软件办公环境

软件使用方法

软件执行逻辑

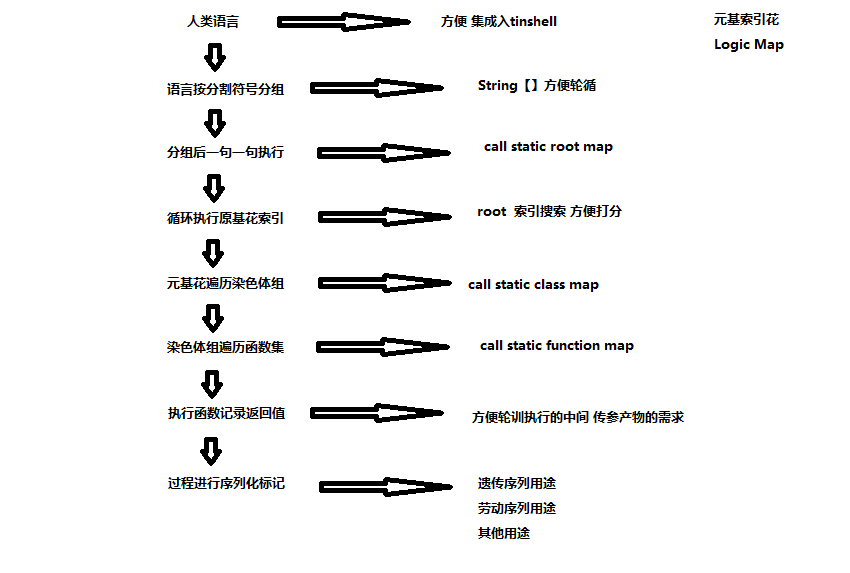
软件注意细节

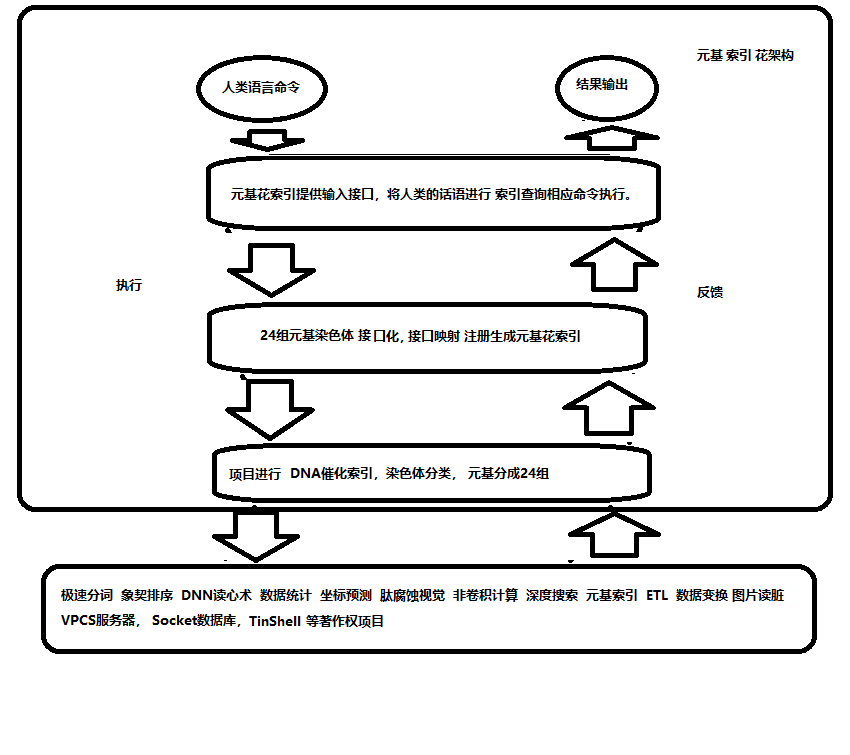
软件申明

软件大小

软件的设计思维

软件的架构理念





第二节 软件源码

RangePDI

LYG9DWithDoubleTopSort5D

StaticRootMap

StaticClassMap

StaticFunctionMap

StaticFunctionMapA\_VECS\_C

StaticFunctionMapA\_IDUQ\_C

StaticFunctionMapO\_VECS\_C

StaticFunctionMapO\_IDUQ\_C

StaticFunctionMapP\_VECS\_C

StaticFunctionMapP\_IDUQ\_C

StaticFunctionMapM\_VECS\_C

StaticFunctionMapM\_IDUQ\_C

StaticFunctionMapA\_VECS\_E

StaticFunctionMapA\_IDUQ\_E

StaticFunctionMapO\_VECS\_E

StaticFunctionMapO\_IDUQ\_E

StaticFunctionMapP\_VECS\_E

StaticFunctionMapP\_IDUQ\_E

StaticFunctionMapM\_VECS\_E

StaticFunctionMapM\_IDUQ\_E

StaticFunctionMapV\_AOPM\_C

StaticFunctionMapV\_IDUQ\_C

StaticFunctionMapE\_AOPM\_C

StaticFunctionMapE\_IDUQ\_C

StaticFunctionMapC\_AOPM\_C

StaticFunctionMapC\_IDUQ\_C

StaticFunctionMapS\_AOPM\_C

StaticFunctionMapS\_IDUQ\_C

StaticFunctionMapV\_AOPM\_E

StaticFunctionMapV\_IDUQ\_E

StaticFunctionMapE\_AOPM\_E

StaticFunctionMapE\_IDUQ\_E

StaticFunctionMapC\_AOPM\_E

StaticFunctionMapC\_IDUQ\_E

StaticFunctionMapS\_AOPM\_E

StaticFunctionMapS\_IDUQ\_E

StaticFunctionMapI\_VECS\_C

StaticFunctionMapI\_AOPM\_C

StaticFunctionMapD\_VECS\_C

StaticFunctionMapD\_AOPM\_C

StaticFunctionMapU\_VECS\_C

StaticFunctionMapU\_AOPM\_C

StaticFunctionMapQ\_VECS\_C

StaticFunctionMapQ\_AOPM\_C

StaticFunctionMapI\_VECS\_E

StaticFunctionMapI\_AOPM\_E

StaticFunctionMapD\_VECS\_E

StaticFunctionMapD\_AOPM\_E

StaticFunctionMapU\_VECS\_E

StaticFunctionMapU\_AOPM\_E

StaticFunctionMapQ\_VECS\_E

StaticFunctionMapQ\_AOPM\_E

序列化索引调用真实示例

第十七章 后序DEMOS

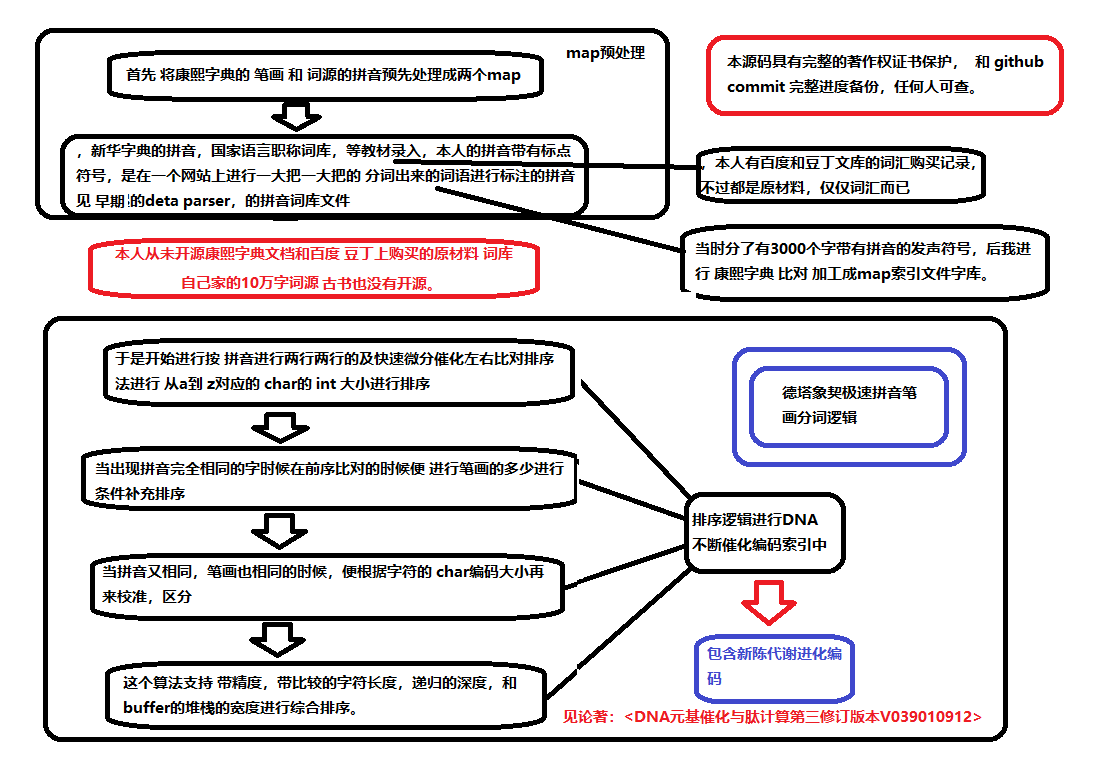
登陆token

肽展session注册

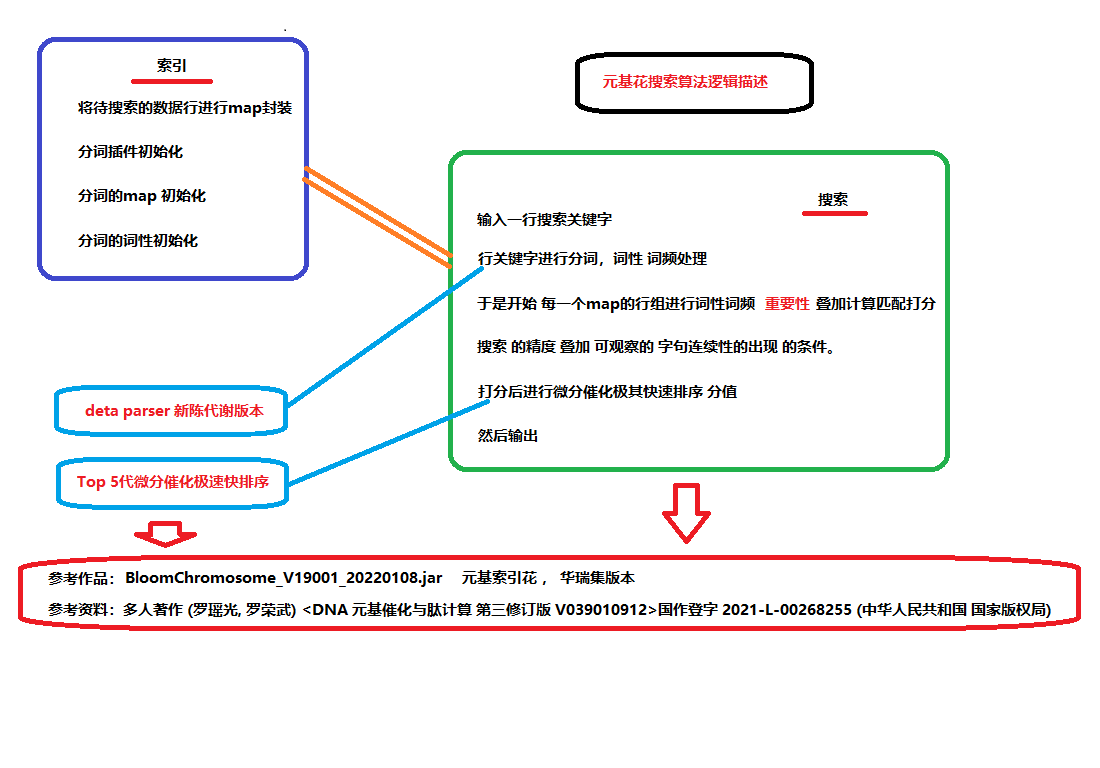
登陆状态验证

PDESwapTestDemo

极速象契拼音笔画排序



精度中文搜索示例



人眼识别的方式

VPCS服务器部署

数字生命



引用

DNA元基催化与肽计算编码

DNA元基催化与肽计算肽展

DNA元基催化与肽计算解码

DNA元基催化与肽计算养疗经应用研究

德塔华瑞集养疗经软件工程类源码引用综合表