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 62](#_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 72](#_Toc7306)

[第四节 DNA筛选的动机 75](#_Toc32665)

[第五节 DNA筛选的应用需求 77](#_Toc24132)

[第六节 DNA筛选的具体描述 77](#_Toc24113)

[味觉语义元基定义 77](#_Toc12369)

[味觉生化元基定义 78](#_Toc4908)

[双元筛选索引词库 78](#_Toc12826)

[第七节DNA筛选的应用实现 78](#_Toc17730)

[第八节DNN分词 词汇花函数源码 78](#_Toc10401)

[方剂森林花JOGL三维计算展示函数 80](#_Toc8689)

[药材功效花JOGL三维计算展示函数 80](#_Toc7653)

[药材禁忌花JOGL三维计算展示函数 80](#_Toc15312)

[花的筛选与观测 116](#_Toc4208)

[第十五章 元基模拟染色体新陈代谢催化编码 117](#_Toc7262)

[第一节 元基造字 120](#_Toc11660)

[Q\_OulerRing, 欧拉路径类 126](#_Toc4407)

[LYG9DWithDoubleTopSort4D, 极速排序算法 132](#_Toc82)

[LYG9DWithDoubleTopSort4D\_U, 极速排序算法 133](#_Toc17633)

[Top Sort 5D 135](#_Toc24833)

[第二节 最新笔记 包含十六元基造字 135](#_Toc12522)

[二次元基新陈代谢方式 137](#_Toc28480)

[LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类 153](#_Toc6457)

[LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类 156](#_Toc24274)

[LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类 158](#_Toc11310)

[第三节 图片识别 158](#_Toc28560)

[图片读脏能力 159](#_Toc20793)

[SkinPathDetectDis, 肽展图片处理类 159](#_Toc24131)

[GetICAStatisticRatio, 肽展图片处理类 162](#_Toc16583)

[GetColorRatioScore, 肽展图片处理类 165](#_Toc23874)

[SkinPathDetectTrip, 肽展图片处理类 168](#_Toc22767)

[MakeImag, 肽展图片处理类 173](#_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 173](#_Toc31823)

[LYG9DWithDoubleTopSort4D，极速象契混合排序 174](#_Toc19925)

[LYG10DWCMSSort15D\_XCDX\_P\_U\_A, 象契字符排序类 176](#_Toc29937)

[LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类 179](#_Toc14345)

[LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C, 象契字符排序类 181](#_Toc25390)

[LYG10DWCMSSort13D\_XCDX\_S, 象契字符排序类 181](#_Toc27495)

[LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类 183](#_Toc16625)

[LYG10DWCMSSort13D\_XCDX\_P\_A\_C, 象契字符排序类 185](#_Toc5156)

[LYG10DWCMSSort13D\_XCDX\_S\_C, 象契字符排序类 185](#_Toc966)

[LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类 186](#_Toc2845)

[LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类 190](#_Toc31890)

[LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类 190](#_Toc13111)

[LYG10DWCMSSort13D\_XCDX\_C\_A\_C, 象契字符排序类 192](#_Toc18926)

[AE\_XCDX\_Map, 肽展中文分词类 192](#_Toc709)

[AE, 肽展中文分词类 198](#_Toc6778)

[A, 肽展中文分词类 205](#_Toc8477)

[A\_XCDX\_Map, 肽展中文分词类 205](#_Toc9890)

[BinaryForest, 肽展分词索引类 206](#_Toc442)

[BinaryForest\_A, 肽展分词索引类 206](#_Toc17832)

[CogsBinaryForest\_AE, 肽展分词索引类 206](#_Toc29096)

[CogsBinaryForest\_A, 肽展分词索引类 212](#_Toc24568)

[BinaryForest\_AE, 肽展分词索引类 212](#_Toc28936)

[Nlp\_CE\_XCDX\_A, 肽展分词索引类 214](#_Toc20744)

[Nlp\_C\_XCDX\_A, 肽展分词索引类 220](#_Toc16873)

[Nlp\_CE\_XCDX\_S, 肽展分词索引类 221](#_Toc7510)

[Nlp\_CE\_XCDX\_A, 肽展分词索引类 225](#_Toc7889)

[Nlp\_C\_XCDX\_S, 肽展分词索引类 232](#_Toc17733)

[Nlp\_CE\_XCDX, 肽展分词索引类 233](#_Toc24235)

[POS\_C\_Cognition\_E, 肽展分词索引类 243](#_Toc995)

[POS\_C, 肽展分词索引类 247](#_Toc14576)

[Pos\_CE\_XCDX\_E, 肽展分词索引类 249](#_Toc223)

[Pos\_CE\_XCDX\_O, 肽展分词索引类 252](#_Toc13377)

[Pos\_C\_XCDX\_E, 肽展分词索引类 254](#_Toc30523)

[Pos\_C\_XCDX\_O, 肽展分词索引类 254](#_Toc7805)

[Pos\_CE\_XCDX\_P, 肽展分词索引类 255](#_Toc27522)

[Pos\_CE\_XCDX\_E, 肽展分词索引类 272](#_Toc21095)

[Pos\_C\_XCDX\_P, 肽展分词索引类 275](#_Toc13006)

[Pos\_CE\_XCDX, 肽展分词索引类 275](#_Toc5508)

[第十六章 TinShell插件\_元基花模拟染色体组计算索引系统 275](#_Toc22545)

[第一节 软件介绍 277](#_Toc11532)

[软件开发动机 277](#_Toc550)

[软件开发目的 277](#_Toc18664)

[软件价值 277](#_Toc7138)

[软件主要功能 277](#_Toc226)

[软件开发系统环境 277](#_Toc32011)

[硬件开发系统环境 277](#_Toc30787)

[软件开发软件环境 277](#_Toc9209)

[软件开发硬件环境 277](#_Toc17256)

[软件部署软件环境 277](#_Toc19793)

[软件部署硬件环境 277](#_Toc18212)

[软件办公环境 277](#_Toc25205)

[软件使用方法 277](#_Toc18391)

[软件执行逻辑 277](#_Toc15520)

[软件注意细节 277](#_Toc20137)

[软件申明 277](#_Toc27086)

[软件大小 277](#_Toc32084)

[软件的设计思维 277](#_Toc18899)

[软件的架构理念 277](#_Toc27984)

[第二节 软件源码 277](#_Toc19803)

[RangePDI 278](#_Toc23913)

[LYG9DWithDoubleTopSort5D 278](#_Toc11679)

[StaticRootMap 278](#_Toc18698)

[StaticClassMap 278](#_Toc27415)

[StaticFunctionMap 278](#_Toc7862)

[StaticFunctionMapA\_VECS\_C 278](#_Toc4725)

[StaticFunctionMapA\_IDUQ\_C 278](#_Toc14225)

[StaticFunctionMapO\_VECS\_C 278](#_Toc676)

[StaticFunctionMapO\_IDUQ\_C 278](#_Toc32740)

[StaticFunctionMapP\_VECS\_C 278](#_Toc937)

[StaticFunctionMapP\_IDUQ\_C 278](#_Toc252)

[StaticFunctionMapM\_VECS\_C 278](#_Toc15760)

[StaticFunctionMapM\_IDUQ\_C 278](#_Toc16091)

[StaticFunctionMapA\_VECS\_E 278](#_Toc22483)

[StaticFunctionMapA\_IDUQ\_E 278](#_Toc12433)

[StaticFunctionMapO\_VECS\_E 278](#_Toc3288)

[StaticFunctionMapO\_IDUQ\_E 278](#_Toc4149)

[StaticFunctionMapP\_VECS\_E 278](#_Toc6774)

[StaticFunctionMapP\_IDUQ\_E 278](#_Toc26586)

[StaticFunctionMapM\_VECS\_E 279](#_Toc32682)

[StaticFunctionMapM\_IDUQ\_E 279](#_Toc10994)

[StaticFunctionMapV\_AOPM\_C 279](#_Toc14559)

[StaticFunctionMapV\_IDUQ\_C 279](#_Toc26619)

[StaticFunctionMapE\_AOPM\_C 279](#_Toc30575)

[StaticFunctionMapE\_IDUQ\_C 279](#_Toc15469)

[StaticFunctionMapC\_AOPM\_C 279](#_Toc30288)

[StaticFunctionMapC\_IDUQ\_C 279](#_Toc23817)

[StaticFunctionMapS\_AOPM\_C 279](#_Toc22670)

[StaticFunctionMapS\_IDUQ\_C 279](#_Toc27486)

[StaticFunctionMapV\_AOPM\_E 279](#_Toc30976)

[StaticFunctionMapV\_IDUQ\_E 279](#_Toc15193)

[StaticFunctionMapE\_AOPM\_E 279](#_Toc14914)

[StaticFunctionMapE\_IDUQ\_E 279](#_Toc17815)

[StaticFunctionMapC\_AOPM\_E 279](#_Toc25349)

[StaticFunctionMapC\_IDUQ\_E 279](#_Toc26664)

[StaticFunctionMapS\_AOPM\_E 279](#_Toc13417)

[StaticFunctionMapS\_IDUQ\_E 279](#_Toc8405)

[StaticFunctionMapI\_VECS\_C 279](#_Toc31608)

[StaticFunctionMapI\_AOPM\_C 279](#_Toc4494)

[StaticFunctionMapD\_VECS\_C 279](#_Toc3136)

[StaticFunctionMapD\_AOPM\_C 279](#_Toc6272)

[StaticFunctionMapU\_VECS\_C 279](#_Toc9514)

[StaticFunctionMapU\_AOPM\_C 279](#_Toc14235)

[StaticFunctionMapQ\_VECS\_C 279](#_Toc12525)

[StaticFunctionMapQ\_AOPM\_C 279](#_Toc25906)

[StaticFunctionMapI\_VECS\_E 279](#_Toc4385)

[StaticFunctionMapI\_AOPM\_E 279](#_Toc30413)

[StaticFunctionMapD\_VECS\_E 279](#_Toc32335)

[StaticFunctionMapD\_AOPM\_E 279](#_Toc8039)

[StaticFunctionMapU\_VECS\_E 279](#_Toc12728)

[StaticFunctionMapU\_AOPM\_E 279](#_Toc1191)

[StaticFunctionMapQ\_VECS\_E 279](#_Toc28204)

[StaticFunctionMapQ\_AOPM\_E 279](#_Toc19157)

[序列化索引调用真实示例 279](#_Toc26783)

[第十七章 后序DEMOS 279](#_Toc13731)

[登陆token 279](#_Toc22112)

[肽展session注册 279](#_Toc11260)

[登陆状态验证 279](#_Toc22051)

[PDESwapTestDemo 279](#_Toc8046)

[极速象契拼音笔画排序 279](#_Toc21050)

[精度中文搜索示例 280](#_Toc7165)

[人眼识别的方式 281](#_Toc11695)

[VPCS服务器部署 281](#_Toc11092)

[数字生命 281](#_Toc23331)

[引用 281](#_Toc20143)

[DNA元基催化与肽计算编码 281](#_Toc13203)

[DNA元基催化与肽计算肽展 281](#_Toc11513)

[DNA元基催化与肽计算解码 281](#_Toc21632)

[DNA元基催化与肽计算养疗经应用研究 281](#_Toc16112)

[德塔华瑞集养疗经软件工程类源码引用综合表 281](#_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

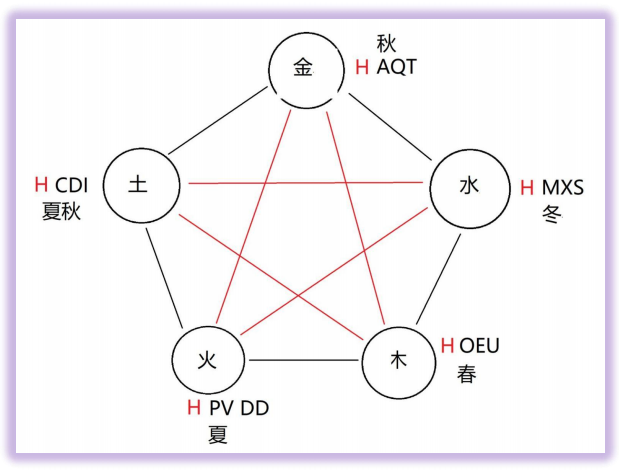
今天来仔细描述这一节. 看过我 CSDN, 知乎等媒体的相信一定有了解了. 我设计的模式很简单, 仅仅把元基解码的数据中 的酮基与甲基, 氨基 进行了计数, 形成了第一和第二排, 然后进行差值计算, 然后进行排序, 然后将元基图谱进行了离散的观测 排列到八卦罗盘形式进行了小简单变换然后连线排序了下活性, 于是生成了下面的各种基础元基罗盘图.

第三节 DNA数术的具体描述

元基数术, 活性, 腐蚀性排序表

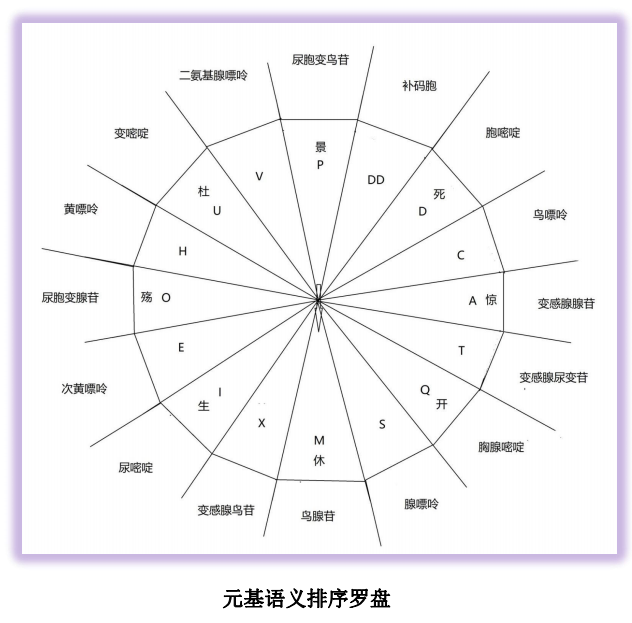


元基语义五行排序图



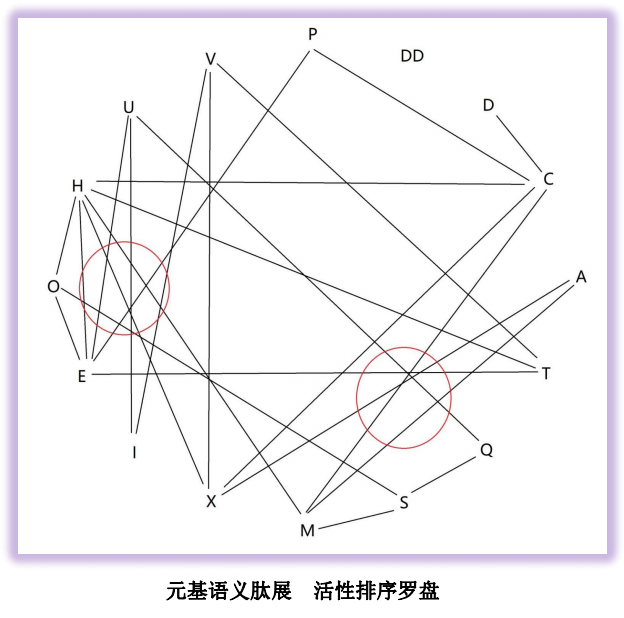
这里为什么我为什么会把 H 元基五行满排, 进行红色标注, 依据是它的活性. 与罗盘 7 个方位都有简洁变换逻辑. 其它元 基布局就好解释了, 按语义编码推导出来的.

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.

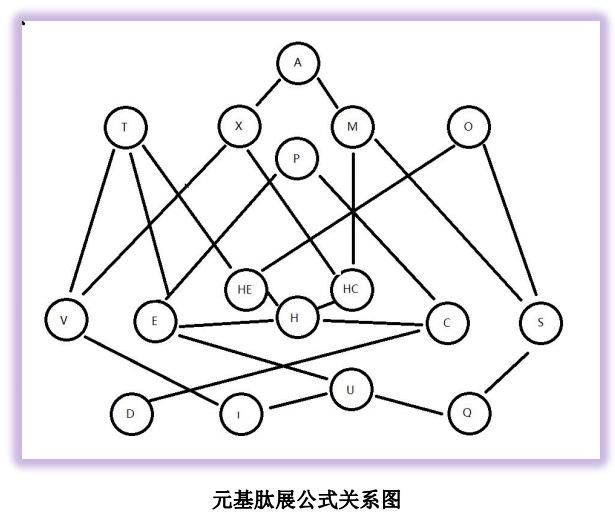
元基语义肽展活性排序罗盘



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

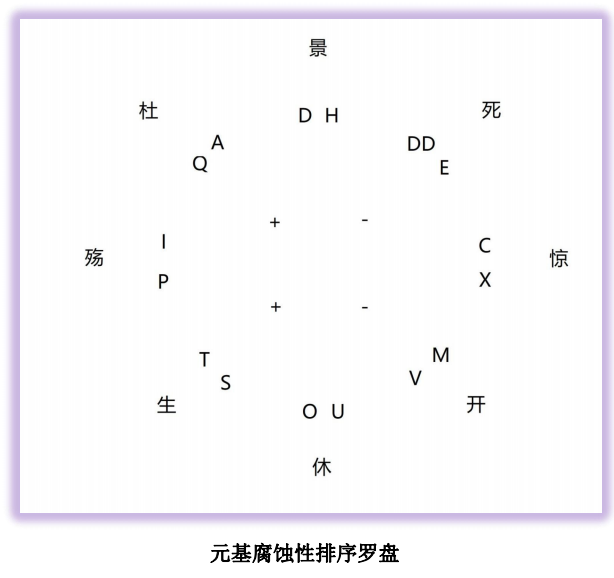
元基肽展公式关系图



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:

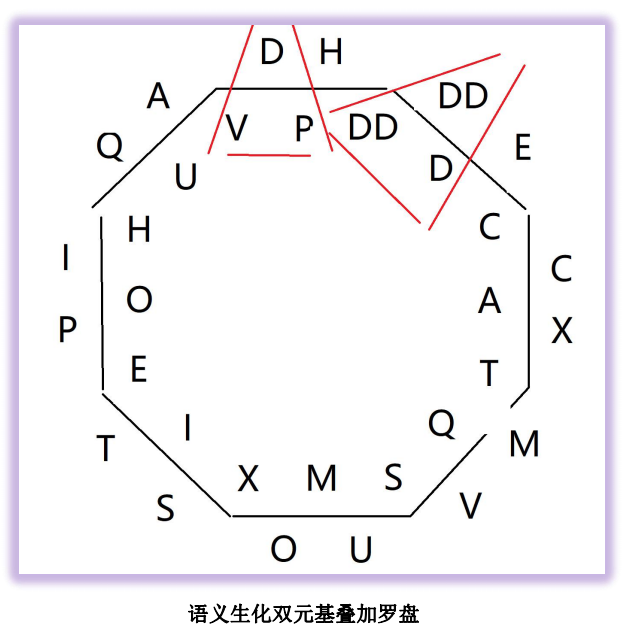
元基腐蚀性排序罗盘



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

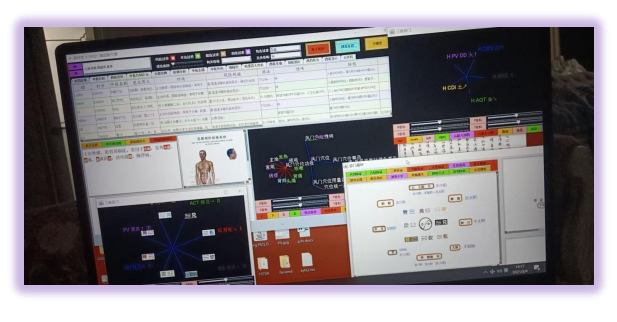
语义生化双元基叠加罗盘



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

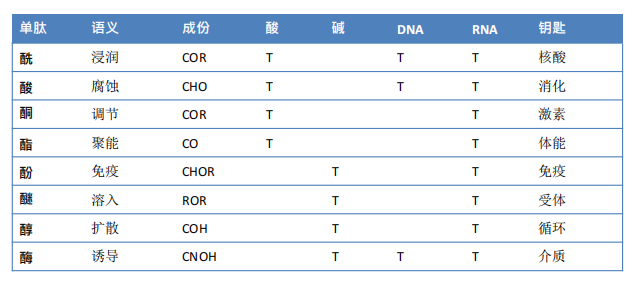
催化映射. 功能强大. 稍后描述. 这里右上角的 DD D DD 的双元催化双角, 胞嘧啶大堆, 这里是一个巨大的生化医学突破口, 第三卷 第四卷我会好好研究下. 先别想那么远, 我先把基础做好, 医学家有幸看到我的文章, 蠢蠢欲动, 让医学家去实验, 我按我的思路来, 求严谨, 不求跳跃. 另外, 语义肽展公式和生化肽展公式 开始有区别了, 我准备定个计划, 将语义肽展公式统一用生化肽展公式, 虽然会少几个, 但效率增加是必然.

无机罗盘术数



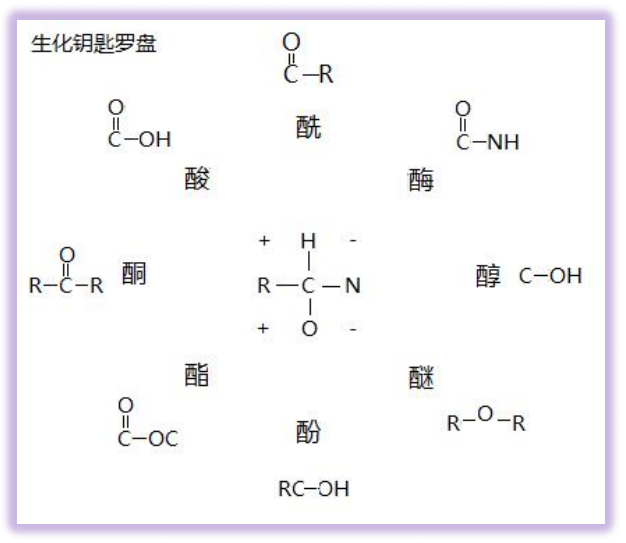
元基罗盘目前已经应用在德塔养疗经[17]医学大数据软件中, 用于中药数据的八纲辨证观测. 这只是一种简单应用, 我设

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

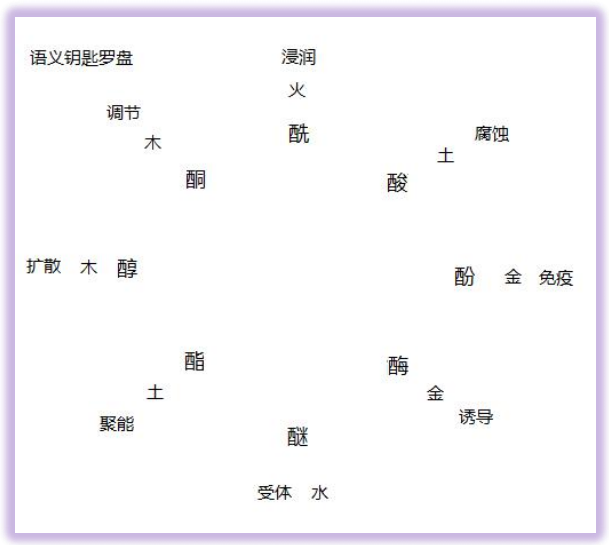


于是生成的语义钥匙和生化钥匙如下:

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

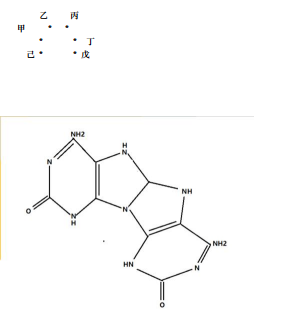
1 ｛F, F｝ 组合 相对其他活性活泼。

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

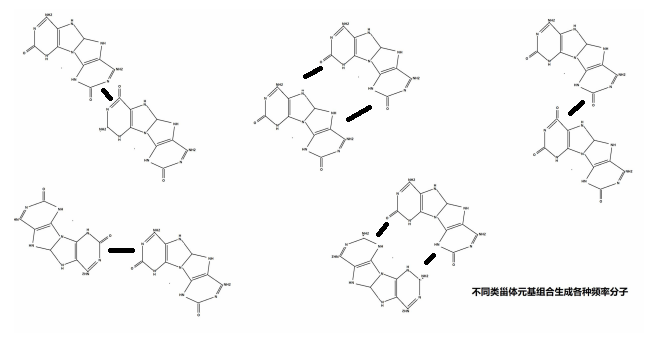
补码吸附 rotation

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



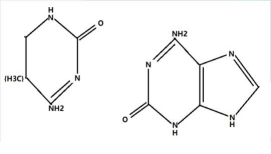
我得到一个结论 DNA 是函数的预先语义表达的函数信号锁存， RNA 的计算模式是神经网络拓扑模式，不是晶振指令周期的驱动，所以RNA 的长度和 结构不规则决定了电势和固有电磁频率的种类。这些种类的组合驱动生命应激表达的行为。这 8 种结构中，1 2 3 4 6 具备了链式拉长 来 改变固有频率。另外 通过对｛F,DU｝｛F,IQ｝ 两种全嘌呤碱基对的电势和离子活性观测，发现

活性

｛F,DU｝ 小于｛F,IQ｝

稳定性

｛F,DU｝ 大于｛F,IQ｝

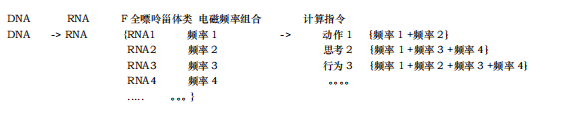


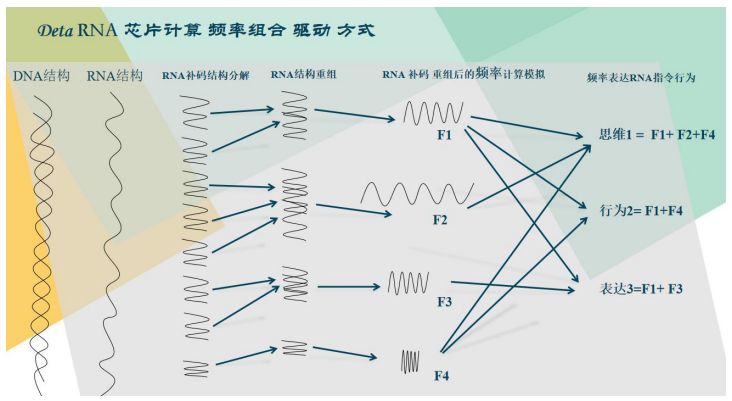
仍有一个问题困扰着我，U Q 的 甲基的语义意思不是很明确。

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

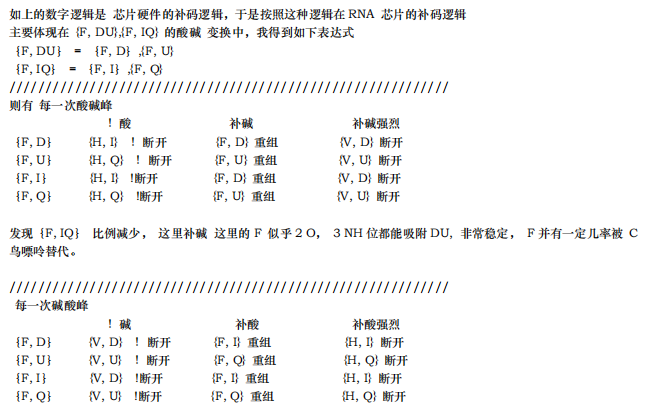
00000011 3

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);

}

}

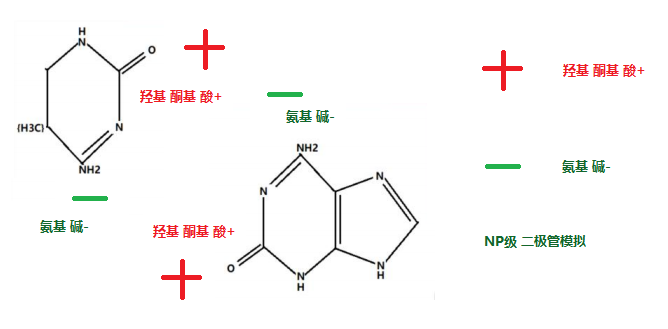
}

}

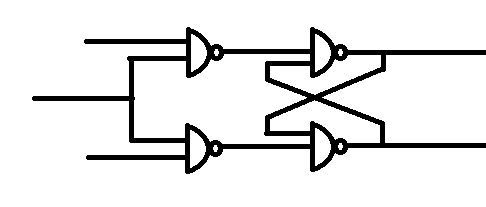
}

FU 全嘌呤变嘧啶数字锁存逻辑

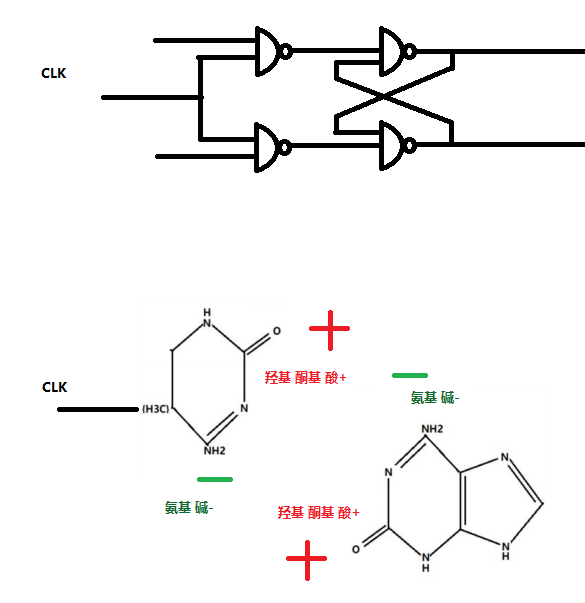
//PN级二极管三极管模拟器件



//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

一开始我的 DNA 搜索动机是能将元基肽展公式进行真实生产环境实践, 体现生产力结构的升级和改变, 慢慢

的我的动机开始模糊, 探索一种思维量化模式的搜索引擎, 提高计算理解力,

第二节 DNA搜索的应用需求

我的需求很简单, 就是满足养疗经[17]的一切应用的基础提供保障, 这个保障我归纳为数据保障, 算法保障,

应用保障.

1 数据保障, 医学的数据需要严谨的, 有序的, 完整的知识结构. 避免搜索断层, 出现分歧

2 算法保障, 每一个算法, 要有对应功能的靶性, 和计算力可持续优化的结构（微分催化[7]结构）

3 应用保障, 主要是医学实践 和 医学应用的傻瓜化, 体现在快速理解和迅捷方便的准确（快广准）应用.

1. DNA搜索的具体描述

ZhongYaoSearch, 搜索类

package OSI.VSQ.SSI.ASU.OSU.PSU.MSU.ASU.MPE.AOP.MEC.SIQ.search;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import ESU.list.List\_ESU;

import ESU.sort.Quick9DLYGWithString\_ESU;

import ME.APM.VSQ.App;

import ME.APM.VSQ.文学性;

import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.TokenPDI;

//作者 罗瑶光

//思想 元基肽展公式

public class ZhongYaoSearch{

public static void zhongYaoSearch(App app, String zhongyao, String forE, String key) {

String CatalyticDNA\_xingwei= "";

String CatalyticDNA\_gongxiao= "";

if(app.DNASearchIsClick) {

TokenPDI pDE\_RNA\_Formular= new TokenPDI();

double rate= ((double)app.催化比值rot)/ 100;

//pDE\_RNA\_Formular.key[0]= 0.3;

pDE\_RNA\_Formular.key[1]= rate;

pDE\_RNA\_Formular.key[2]= rate;

//pDE\_RNA\_Formular.key[3]= 0.3;

if(null!= app.nameFeelFilter.getText()) {

pDE\_RNA\_Formular.pdw= app.nameFeelFilter.getText().toUpperCase();

pDE\_RNA\_Formular.code= pDE\_RNA\_Formular.pdw.toString().toUpperCase();

pDE\_RNA\_Formular.doKeyPress(pDE\_RNA\_Formular.code, pDE\_RNA\_Formular, false);

CatalyticDNA\_xingwei= null== pDE\_RNA\_Formular.pde?""

: pDE\_RNA\_Formular.pde.toString();

}

//

pDE\_RNA\_Formular.pde= "";

if(null!= app.name\_filter\_not\_have.getText()) {

pDE\_RNA\_Formular.pdw= app.name\_filter\_not\_have.getText().toUpperCase();

pDE\_RNA\_Formular.code= pDE\_RNA\_Formular.pdw.toString().toUpperCase();

pDE\_RNA\_Formular.doKeyPress(pDE\_RNA\_Formular.code, pDE\_RNA\_Formular, false);

CatalyticDNA\_gongxiao= null== pDE\_RNA\_Formular.pde?""

: pDE\_RNA\_Formular.pde.toString();

}

}

String[] score= new String[app.copy.size()];

int[] score\_code= new int[app.copy.size()];

int []reg= new int[app.copy.size()];

int count= 0;

Map<String, WordFrequency> mapSearchWithoutSort= null;

if(app.dic\_map.containsKey(zhongyao.replaceAll(" ", ""))) {

mapSearchWithoutSort= app.\_A.parserMixStringByReturnFrequencyMap(zhongyao);

}else {

String[] strings= key.split(" ");

if(strings.length> 1&& key.split(" ")[0].length()> 5) {

mapSearchWithoutSort= app.\_A.parserMixStringByReturnFrequencyMap(key);

}else {

mapSearchWithoutSort= app.\_A.parserMixStringByReturnFrequencyMap(zhongyao);

}

}

Iterator<String> iteratorForCopy= app.copy.iterator();

int copyCount= 0;

List<String> list= app.\_A.parserMixedString(key);

String[] string= List\_ESU.listToArray(list);

String[] stringReg= new String[forE.length()/ 3];

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

stringReg[i]= forE.substring(i\* 3, (i\* 3+ 3)< forE.length()? (i\* 3+ 3): forE.length()- 1);

}

while(iteratorForCopy.hasNext()) {

String iteratorForCopyString= iteratorForCopy.next();

score[copyCount]= iteratorForCopyString;

String temps= app.dic\_map.get(iteratorForCopyString).toString();

String tempsPCA= app.dic\_li.get(iteratorForCopyString).toString();

String tempsIndex= app.dic\_index.get(iteratorForCopyString).toString();

if(app.appConfig.SectionJPanel.jlabel\_peizhi\_di2233.isSelected()) {//文学性测试

tempsPCA= 文学性.文言文全文处理(temps);

temps= 文学性.文言文线性处理(app, tempsPCA);

}

Iterator<String> iteratorWordFrequency= mapSearchWithoutSort.keySet().iterator();

Here:

while(iteratorWordFrequency.hasNext()) {

String mapSearchaAtII = iteratorWordFrequency.next();

WordFrequency wordFrequencySearch

= mapSearchWithoutSort.get(mapSearchaAtII);

if(temps.contains(mapSearchaAtII)) {

if(reg[copyCount] == 0){

count += 1;

}

if(score[copyCount].contains(zhongyao.replace(" ", ""))) {

reg[copyCount]+= 12;

}

if(zhongyao.contains(score[copyCount].replace(" ", ""))) {

reg[copyCount]+= 12;

}

//因为这几行是新电脑开发的只有3行数据测试,老电脑写的拼音索引没用到,就出现这个问题

//,我先注释掉,以后优化. 罗瑶光20210516

if(tempsIndex.equalsIgnoreCase(zhongyao.replace(" ", ""))) {

reg[copyCount]+= 1200;

}

if(tempsIndex.contains(zhongyao.replace(" ", "").toUpperCase())) {

reg[copyCount]+= 1200;

}

score[copyCount]= iteratorForCopyString;

if(!app.pos.containsKey(mapSearchaAtII)) {

reg[copyCount]+= 1;

score\_code[copyCount]+= 1<< mapSearchaAtII.length()

<< wordFrequencySearch.getFrequency() ;

if(tempsPCA.contains(mapSearchaAtII)) {

score\_code[copyCount] <<= 1;

}

if(score[copyCount].contains(mapSearchaAtII)) {

if(score[copyCount].length()>1) {

reg[copyCount]+= 22;

}

reg[copyCount]+= 3;

}

continue Here;

}

if(app.pos.get(mapSearchaAtII).contains("名")

|| app.pos.get(mapSearchaAtII).contains("动")

|| app.pos.get(mapSearchaAtII).contains("形")

|| app.pos.get(mapSearchaAtII).contains("谓")) {

reg[copyCount]+= 2;

if(tempsPCA.contains(mapSearchaAtII)) {

reg[copyCount]<<= 1;

}

}

reg[copyCount]+= 1;

score\_code[copyCount]+= (iteratorForCopyString.contains(mapSearchaAtII)

? 2: 1)

<< (!app.pos.get(mapSearchaAtII).contains("名") //测试了下不错，误差小，罗瑶光202111011900

? app.pos.get(mapSearchaAtII).contains("动")? 3: 0: 7) //用位移替换10、1、150

// \* (!app.pos.get(mapSearchaAtII).contains("名")

// ? app.pos.get(mapSearchaAtII).contains("动")? 10: 1: 150) //用位移替换10、1、150

//动、、名

<< mapSearchaAtII.length()\* wordFrequencySearch.getFrequency();

if(score[copyCount].contains(mapSearchaAtII)) {

if(score[copyCount].length()>1) {

reg[copyCount]+= 22;

}

reg[copyCount]+= 3;

}

continue Here;

}

if(mapSearchaAtII.length()>1) {

for(int j=0;j<mapSearchaAtII.length();j++) {

if(temps.contains(String.valueOf(mapSearchaAtII.charAt(j)))) {

if(reg[copyCount] == 0){

count += 1;

}

score[copyCount] = iteratorForCopyString;

score\_code[copyCount]+=1;

if(app.pos.containsKey(String.valueOf(mapSearchaAtII.charAt(j)))

&&(app.pos.get(String.valueOf(mapSearchaAtII.charAt(j))).contains("名")

||app.pos.get(String.valueOf(mapSearchaAtII.charAt(j))).contains("动") ||app.pos.get(String.valueOf(mapSearchaAtII.charAt(j))).contains("形") ||app.pos.get(String.valueOf(mapSearchaAtII.charAt(j))).contains("谓")

)) {

reg[copyCount] += 2;

}

reg[copyCount] += 1;

if(score[copyCount].contains(mapSearchaAtII)) {

if(score[copyCount].length()>1) {

reg[copyCount]+= 12;

}

reg[copyCount]+= 3;

}

continue Here;

}

}

}

}

score\_code[copyCount] = score\_code[copyCount] \* reg[copyCount];

//词距

//今天检查了没有封顶，会被 int 超出溢出问题。

score\_code[copyCount] = score\_code[copyCount]/ 1000;

if(score[copyCount].contains("紫雪散")) {

// int j=0;

}

int code= 200;

int tempb= 0;

int tempa= score\_code[copyCount];

if(key.length()> 6) {

//全词

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

if(temps.contains(string[i])&& string[i].length()>1 ) {

tempb+= code;

}

}

//断句

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

if(temps.contains(stringReg[i])) {

tempb+= code;

}

}

score\_code[copyCount] = (int) (tempa/Math.pow(app.lookrot+ 1, 4)

+ tempb\*Math.pow(app.lookrot, 2));

}

if(zhongyao.replace(" ", "").length()> 1&& zhongyao.replace(" ", "").length()< 5) {

if(temps.contains(zhongyao.replace(" ", ""))) {

tempb+= code<< 7;

}

score\_code[copyCount] = (int) (tempa/Math.pow(app.lookrot+ 1, 4)

+ tempb\*Math.pow(app.lookrot, 2));

}

copyCount++;

}

//new Quick9DLYGWithString\_ESU().sortZhongYao(score\_code, score);

new Quick9DLYGWithString\_ESU().sort(score\_code, score);

Object[][] tableData= new Object[count][13];

int new\_count = 0;

app.newTableModel.getDataVector().clear();

if(null== key|| key.equals("")) {

for(int i= 0; i < app.tableData\_old.length; i++) {

app.tableData\_old[i][6]= app.tableData\_old[i][6]==null? "": app.tableData\_old[i][6];

app.newTableModel.insertRow(i, app.tableData\_old[i]);

}

app.newTableModel.fireTableDataChanged();

return;

}

Here:

for(int i = app.copy.size()-1; i > -1; i--) {

if(score[i].contains("紫雪散")) {

//int j=0;

}

if(score\_code[i]< 1){

continue Here;

}

if(app.risk\_filter\_box.isSelected()) {

String hai= (app.dic\_hai.get(score[i])== null? "null."

: app.dic\_hai.get(score[i]).toString().replaceAll("\\s\*", "")

.equalsIgnoreCase("")? "null": app.dic\_hai.get(score[i]).toString()

.replaceAll("\\s\*", ""));

String temp= app.name\_filter.getText();

for(int j=0;j<temp.length();j++) {

if(hai.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

if(app.feel\_filter\_box.isSelected()) {

String li= (app.dic\_li.get(score[i])== null? "null."

: app.dic\_li.get(score[i]).toString().replaceAll("\\s\*", "")

.equalsIgnoreCase("")?"null": app.dic\_li.get(score[i]).toString()

.replaceAll("\\s\*", ""));

String temp= app.name\_filter.getText();

for(int j= 0; j< temp.length(); j++) {

if(li.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

//20210724 //合并后数据冗余 eclipse不上报错误， 因为空格字符null问题，

//害我调试了半天。bin的debug函数被本地欺骗了。

//催化比值rot dna 催化计算

if(!app.nameFeelFilter.getText().isEmpty()) {

String wei= !app.dic\_xw.containsKey(score[i])? ""

: app.dic\_xw.get(score[i]).toString().replaceAll("\\s\*", "");

//String wei= app.dic\_xw.get(score[i]).toString().replaceAll("\\s\*", "");

CatalyticDNA\_xingwei+= app.nameFeelFilter.getText().replace(" ", "");

for(int j= 0; j< CatalyticDNA\_xingwei.length(); j++) {

if(wei.contains(""+ CatalyticDNA\_xingwei.charAt(j))) {

continue Here;

}

}

}

if(null!= app.name\_filter\_not\_have.getText()) {

if(!app.name\_filter\_not\_have.getText().replace(" ", "").isEmpty()) {

String wei= !app.dic\_jm.containsKey(score[i])? ""

: app.dic\_jm.get(score[i]).toString().replaceAll("\\s\*", "");

//String wei= app.dic\_jm.get(score[i]).toString().replaceAll("\\s\*", "");

CatalyticDNA\_gongxiao+= app.name\_filter\_not\_have.getText().replace(" ", "");

for(int j= 0; j< CatalyticDNA\_gongxiao.length(); j++) {

if(!wei.contains(""+ CatalyticDNA\_gongxiao.charAt(j))) {

continue Here;

}

}

}

}

if(app.shuming\_filter\_box.isSelected()) {

String wei= score[i];

String temp= app.name\_filter.getText();

for(int j= 0; j< temp.length(); j++) {

if(wei.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

String temp= app.dic\_map.get(score[i]).toString();

if(tableData.length<= new\_count) {

continue Here;

}

tableData[new\_count]= new Object[]{

(app.dic\_index.get(score[i])== null? ""

: app.dic\_index.get(score[i])).toString().replaceAll("\\s\*", ""),

score\_code[i], score[i],

(app.dic\_yw.get(score[i])== null? ""

: app.dic\_yw.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_li.get(score[i])== null? ""

: app.dic\_li.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_hai.get(score[i])==null

?"详情参考笔记原文列: 是药三分毒, 补药甚三分. 食材亦如此, 勤俭亦长生."

:app.dic\_hai.get(score[i]).toString().replaceAll("\\s\*", "")

.equalsIgnoreCase("")?"详情参考笔记原文列"

:app.dic\_hai.get(score[i]).toString().replaceAll("\\s\*", "")),

(app.dic\_yl.get(score[i])==null?"详情参考相关书籍"

:app.dic\_yl.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_xw.get(score[i])== null? ""

: app.dic\_xw.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_jm.get(score[i])== null? ""

: app.dic\_jm.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_xz.get(score[i])==null?""

:app.dic\_xz.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_jj.get(score[i])==null?""

:app.dic\_jj.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_cy.get(score[i])==null?""

:app.dic\_cy.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_ya.get(score[i])==null?""

:app.dic\_ya.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_zf.get(score[i])==null?""

:app.dic\_zf.get(score[i])).toString().replaceAll("\\s\*", ""),

(app.dic\_cj.get(score[i])==null?""

:app.dic\_cj.get(score[i])).toString().replaceAll("\\s\*", "")};

if(zhongyao.contains("风寒")) {

if(temp.contains("风寒")) {

app.newTableModel.insertRow(new\_count, tableData[new\_count]);

new\_count += 1;

}

}else if(zhongyao.contains("风热")){

if(temp.contains("风热")) {

app.newTableModel.insertRow(new\_count, tableData[new\_count]);

new\_count += 1;

}

}else {

app.newTableModel.insertRow(new\_count, tableData[new\_count]);

new\_count+=1;

}

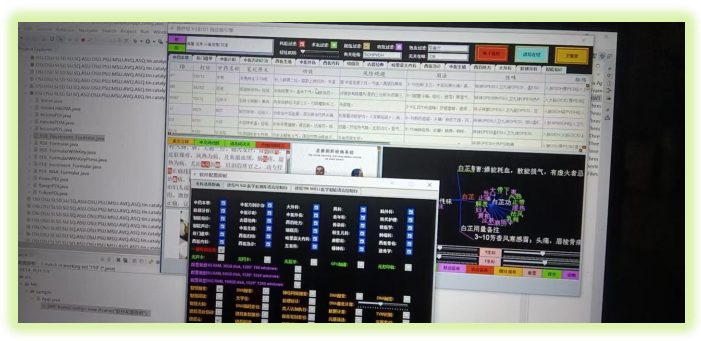
}

app.newTableModel.fireTableDataChanged();

}

}

DNA搜索的应用实现



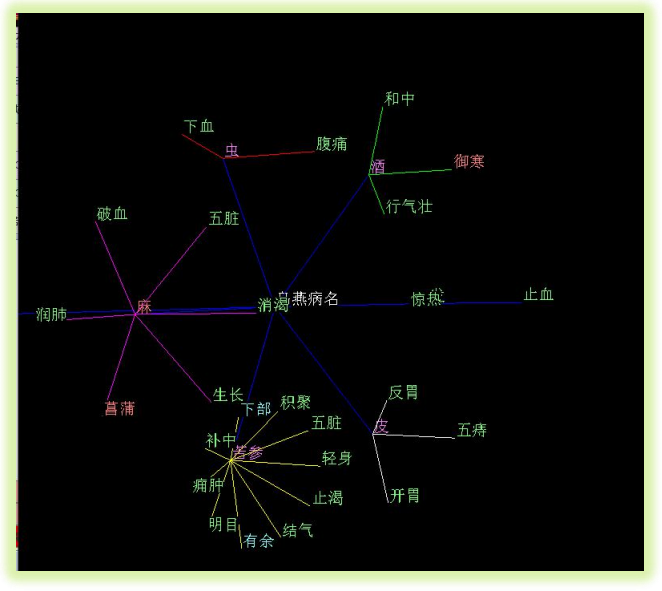
在没有设计元基之前, 我便设计了多种完整的数据聚类搜索函数. 我一直在思考, （稍后我会加非元基搜索的多种实例展示. ） 如果加入了元基计算, 要达到怎么样一种预期效果, 快, 广, 准, 是必要的.其次？ 我想到很多, 但终究不如实践与推导. 于是我设计 了这个计算模式的版本如图. 效果不错, 但很粗糙, 因为是我的第一代元基搜索. 我一会一直优化它. 这个两张搜索 关于风湿性风寒, 筛选出了白芷, 我结合中医看了下, 筛出的都是君药, 我之后会按书上的意思推选出臣药 出来. 让计算观测功能更加丰富, 严谨, 准确



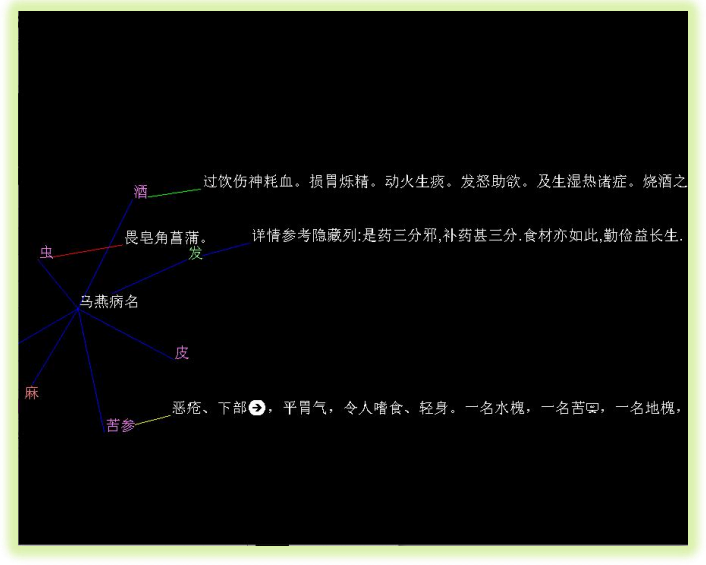
简单介绍下中药本草页面的逻辑上面横条是全局栏目, 用于 搜索, 筛选, 观测和全局控制, 中药页的上部分是一个六行显 示的表格, 默认排序输出价值元组. 下面则是数据观测部分, 从一维的线性文本, 2 维的图片数据, 到三维的属性花, 组成了数据 分析的核心部分. 现在, 经络和性味的元基筛选已经成功, 如上图展示



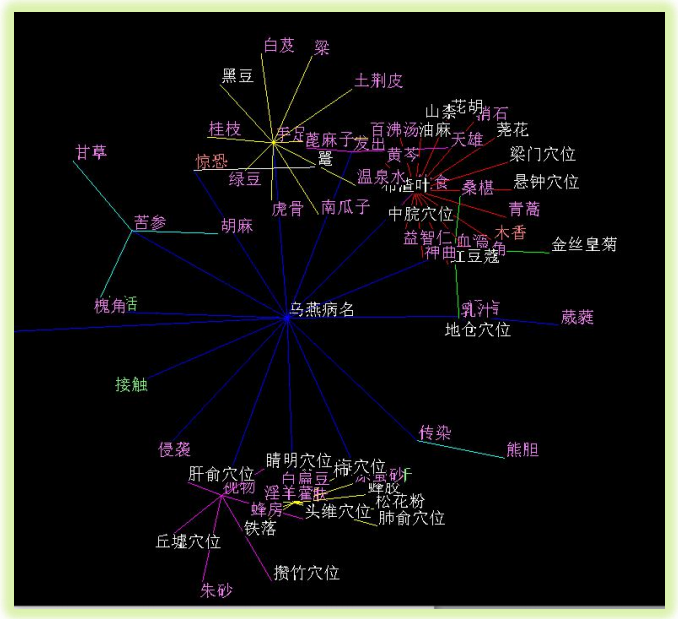
上图通过无关性味的E 元基过滤后, 搜索麻风 列出的中医方剂进行 DNN 深度分析价值词汇, 然后进行三维属性聚类观测, 此时, 中医药页面显示其相关对症的有价值中药列次在表格中方便查阅. 之后这元基变换函数我会升级在数据变换 API[3]中。同 时，上面的应用是中医方剂和中医本草两个页面进行耦合操作，然后在三维图中观测，养疗经的全局分类依次打分排序体系支持多种 复杂的分析操作。



这张图的原理是点击方剂元组后, DNN 展示的数据词汇（根据不同的精度, 词汇数目可以控制） 进行分类, 分类后的第一 层展开, 开始进行第二层功效搜索, 进行聚类. 这样一分一聚 就生成了DNN 三维词汇花的骨架结构. 如图 乌燕一药含有 苦参, 苦参有治疗痈肿的功效.



这张图方便在药物聚类搜索后查看其主要禁忌, 于是我设计了这个功能组件. 之后禁忌属性也会全部语义方式元基肽化.

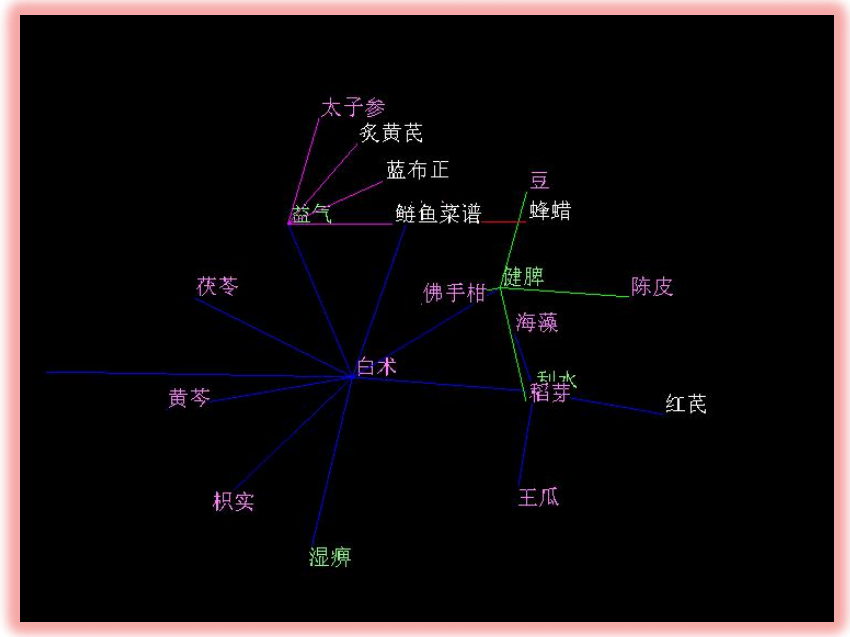


这张图恰好与 DNN 观测相反. 我将乌燕一方剂进行主要功效拓扑, 然后每一个有价值功效分类进行 相关对症的药物聚 类展示, 如图, 乌燕对传染性疾病有价值, 传染病对症的药物有熊胆（黄疸类肺皮血肝胆部等疾病）. 入手足疾病, 桂花治疗手痛. 如果关联有疑惑, 于是可以通过中药页表格搜索进行持续傻瓜化搜索.

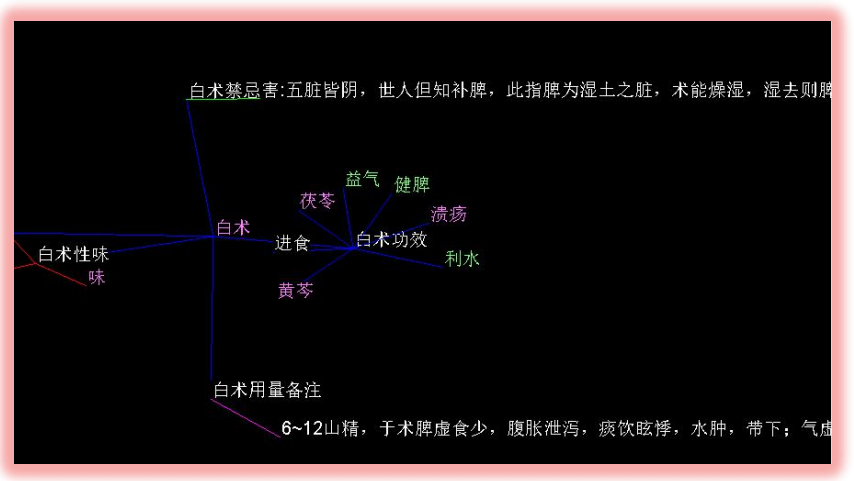
第四节DNA筛选的动机



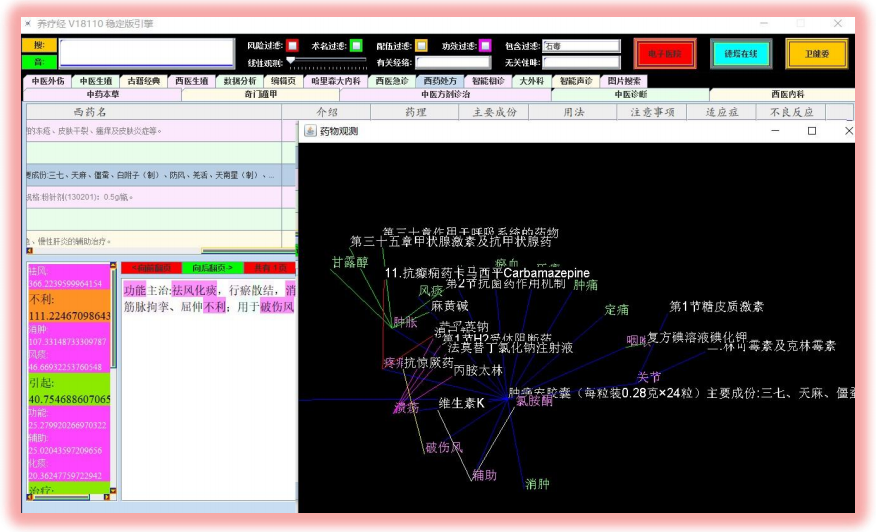
上面这张图, 通过元基的经络和性味筛选后, 再过滤风险中的标识文字, 于是禁忌三维花开始清晰. 通常一味药的功效越 强，毒性就越大，禁忌就越多，于是禁忌观测将这类危险级别高的药物在这里进行了归纳观测.



上面这张图为 搜索白术后出现的 DNN 价值词汇, 如果其中词汇属于功效则进行第二次对症药物搜索, 因为搜索的数据太 多, 为了方便清晰显示, 于是进行 T 坐标进行时间轴控制避免花屏



这张图展示了白术的属性列分类进行 PCA（主要成份显示）显示. 如白术，进行禁忌，功效，用量，性味进行分类扩展， 其功效在进行 PCA 扩展 益气，健脾，治疗溃疡， 在搭配上可以和茯苓 黄芩入伍等. 在用量上一般在 6-12 克炮制中药.



西药 DNN 关联, 不多解释了, 西药的数据类似 中药方剂页 DNN 功效聚类显示, 同理. 上图点击肿痛按胶囊，图中显示有治疗肿胀作用，而肿胀的对症西药也有很多，如甘露醇，醇类， 激素类等，该药有缓解疼痛， 治疗溃疡等作用，其对症药物也有很多显示，如图，方便职业医生的查询思维拓展.

第五节 DNA筛选的应用需求



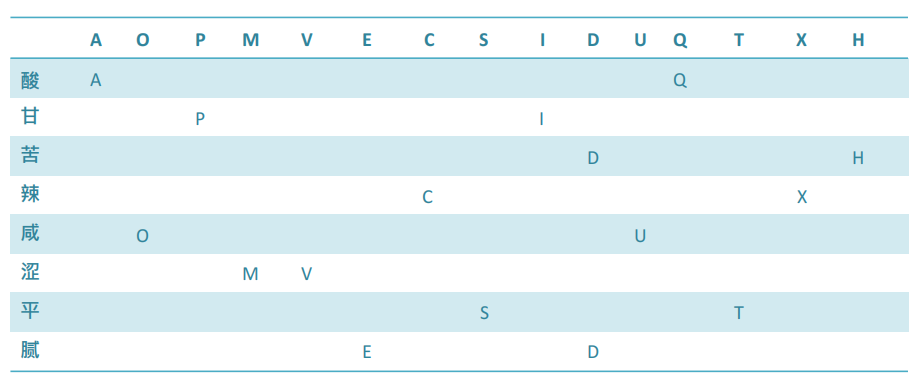
目前养疗经[17]采用的元基筛选属于混合语义元基进行单元基筛选. 之后, 随着语义语料库的丰富和全部文字肽化, 准备 进行三元基词汇筛选

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



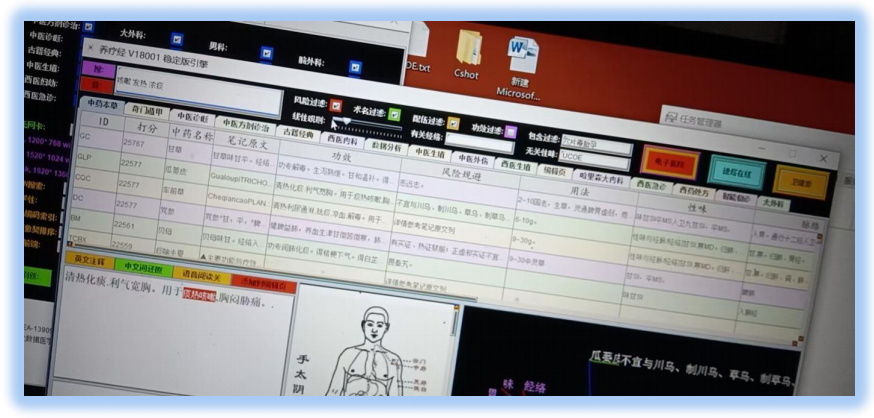
双元筛选索引词库

元基的生化归纳定义, 是根据文字的八卦方位, 对应生化腐蚀罗盘, 双元罗盘的方位, 进行归纳的元基词汇. 如酸属于木, 在东方, 巽位, 为 AQ, 方便大家理解. 双元筛选索引词库如下: 酸: VUI. AQ 甘 : VUI. PI 苦: VUI. DH 辣: VUI. CX 咸: VUI. OU 涩: VUI. MV 平: VUD. ST 腻: VUI. ED我将双元基的词根进行无理级别复合组成了高级元基文字, 这种文字同时满足语义又满足生化理解的双重属性, 方便我养 疗经[17]的下一步演化. 一开始我仅想用 3 个语义元基字母来造字，后来发现，很多近义词 三个元不好归纳，就多加元基来归 纳，见语料库章节，结果一些词汇元基特别长， 于是我有必要创造一种新的造字方法，现在这种生化加语义的词汇元基，看起 来非常不错，按我的思想，既然不错我就开始应用了

第七节DNA筛选的应用实现



尝试下在搜索风热咳嗽的对症下去掉 S 性味语义元基筛选后, 出现的贝母，黄芩，前胡，牛蒡子， 黄花，丹草 6 味君药. 方 便大家的理解, S= I, S= Q, IQ 为酮基嘧啶, 温燥. 于是 D, M 中 C 寒为主等药 物列筛选出来. 同时生化罗盘中这里 S 属于土对症胃过滤, M 属于金对症骨肺保留, 下一步跟进研究我会放在第三卷, 第二卷 不做生化跟进, 仅仅做语义研究, 思维打止.



这张图, 逐步开始大分子元基语义（这里是语义不是生化, 大家注意, 如果是语义归纳的元基就必须用语义, 生化归纳的元基 就必须用生化. 避免出错呀）性味筛选, 咳嗽发热浓痰对症出现 瓜蒌皮MD 贝母 SI 党参 MS 半夏 甘草 MS 车前草 MD 药物, 通过筛选温燥的元基过滤掉后, 结果展示. 上图搜索发热咳嗽 浓痰，开启风险过滤 胎孕毒穴片关键字， 然后无关性味过滤 UCOV 元基础成份，结果含有 甘草， 瓜蒌皮，车前草，党参，贝母，痰喘半夏等. 在配伍过滤开启后，禁忌明显相冲的药物已经被过滤掉. 线性观测调节，关键字比 重加重后筛选结果. 大大方便了职业医生的用药选择观测.

第八节DNN分词 词汇花函数源码



同理归纳不多解释了, 都是语义元基级别, 不做生化元基分析（第三卷研究）. 上图搜索发热咳嗽 浓痰 头晕，开启风险过滤 胎孕毒石穴片谱关键字， 然后有关经络保留 AQV 元基成份，无关性味过滤 PE 元基础成份，结果含有 阿胶，车前草，党参，贝母，瓜蒌皮，甘草等.在配伍过滤开启后，禁忌明显相冲的药物已经被过滤 掉



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

//该函数采用JAVA JOGL 三维硬件插件， 详细见gleem官方demos示例

package IMO.EVU.bi\_P;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.awt.event.MouseWheelEvent;

import java.awt.event.MouseWheelListener;

import java.awt.image.BufferedImage;

import java.io.IOException;

import java.nio.ByteBuffer;

import java.util.Iterator;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import javax.swing.\*;

import javax.swing.event.ChangeEvent;

import javax.swing.event.ChangeListener;

import javax.media.opengl.\*;

import MVQ.button.DetaButton;

import MVQ.slider.DetaSlider;

//import OCI.ME.analysis.C.A;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

import OSI.VSQ.SSI.ASU.OSU.PSU.MSU.ASU.MPE.AOP.MEC.SIQ.search.ZhongYaoSearch;

import VSQ.OPE.jogl.JOGLOBJShape;

import com.jogamp.opengl.util.FPSAnimator;

import com.jogamp.opengl.util.GLBuffers;

import com.jogamp.opengl.util.gl2.GLUT;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import ME.APM.VSQ.App;

import MSV.OSU.string.NullObject;

import javax.media.opengl.awt.GLCanvas;

import javax.media.opengl.glu.GLU;

public class CoAuthorForWord extends JPanel implements MouseMotionListener, MouseListener, MouseWheelListener, KeyListener, GLEventListener {

public Boolean clicked = false;

public int mousex = 0;

public int mousey = 0;

public boolean hook= false;

int zynk= 0;

int zybc= 0;

public int needChange = 0;

public String prevSearchString = "";

public ByteBuffer checkImageBuf = GLBuffers.newDirectByteBuffer(20 \* 100 \* 3);

public Map<String, String> pos;

public Map<String, Object> rootLeaf;

public Map<String, Integer> findLeaf;

public Map<String, Integer> frequencyLeaf;

public Map<String, Integer> didLeaf;

public App u;

public Map<String, Object> dic\_li ;

public Map<String, Object> dic\_yl;

public CogsBinaryForest\_AE \_A;

public int c = 0;

public JFrame jframe;

private static final long serialVersionUID = 1L;

public FPSAnimator animator = null;

ChangeListener listener;

Box sliderBoxx1 = new Box(BoxLayout.X\_AXIS);

Box sliderBoxx2 = new Box(BoxLayout.X\_AXIS);

Box sliderBox = new Box(BoxLayout.Y\_AXIS);

Box buttonBox = new Box(BoxLayout.X\_AXIS);

JSlider sliderx;

JSlider slidery;

JSlider sliderz;

JSlider slidert;

JButton top;

JButton down;

JButton left;

JButton right;

JButton bhl;

JButton cfc;

JButton whh;

JButton rst;

JButton save;

JButton load;

GL2 gl;

GLU glu;

GLUT glut;

float whiteLight[] = {0.2f, 0.2f, 0.2f, 1.0f};

float sourceLight[] = {0.8f, 0.8f, 0.8f, 1.0f};

float lightPos[] = {0.0f, 0.0f, 0.0f, 1.0f};

float moveshape = 0.0f;

float fEarthRot = 0.0f;

float xrot = 18.0f;

float yrot = 2.0f;

float zrot = 1.0f;

float trot = 6.0f;

float prot = 1.0f;

float lrot = -3.5f;

float brot = 1.0f;

float irot = 78.0f;

float srot = 0.7f;

float xrotr = 0.0f;

float yrotr = 0.0f;

float zrotr = 0.0f;

float trotr = 0.0f;

float protr = 0.0f;

float lrotr = 0.0f;

float brotr = 0.0f;

float irotr = 0.0f;

float srotr = 0.0f;

public GLCanvas canvas;

ByteBuffer buffer;

int[][] g;

JOGLOBJShape shape=null;

public double t = 1.0d;

public String rootWord;

public CoAuthorForWord(App u, CogsBinaryForest\_AE \_A

, Map<String, String> pos) throws HeadlessException, InterruptedException {

Thread.sleep(100);

rootWord="";

rootLeaf = new ConcurrentHashMap<>();

findLeaf = new ConcurrentHashMap<>();

frequencyLeaf = new ConcurrentHashMap<>();

didLeaf= new ConcurrentHashMap<>();

this.\_A = \_A;

this.pos = pos;

this.u = u;

this.setLayout(null);

Thread.sleep(100);

GLProfile glp = GLProfile.getDefault();

GLCapabilities glcaps = new GLCapabilities(glp);

canvas = new GLCanvas(glcaps);

canvas.addGLEventListener(this);

canvas.addMouseListener(this);

Thread.sleep(100);

canvas.addMouseMotionListener(this);

canvas.addKeyListener(this);

canvas.addMouseWheelListener(this);

canvas.setBounds(0, 0, 618, 350-50);

Thread.sleep(100);

this.add(canvas);

animator = new FPSAnimator(canvas,24,true);

centerWindow(this);

animator.start();

Thread.sleep(100);

sliderx = new DetaSlider(0, 360);

sliderx.setSnapToTicks(true);

sliderx.setPaintTicks(true);

sliderx.setMajorTickSpacing(20);

sliderx.setMinorTickSpacing(5);

sliderx.addChangeListener(

new ChangeListener(){

public void stateChanged(ChangeEvent event){

JSlider source = (JSlider) event.getSource();

xrot = source.getValue();

}

});

Thread.sleep(100);

slidery = new DetaSlider(0 , 360);

slidery.setSnapToTicks(true);

slidery.setPaintTicks(true);

slidery.setMajorTickSpacing(20);

slidery.setMinorTickSpacing(5);

slidery.addChangeListener(

new ChangeListener(){

public void stateChanged(ChangeEvent event) {

JSlider source = (JSlider) event.getSource();

yrot= source.getValue();

}

});

Thread.sleep(100);

sliderz = new DetaSlider(0 , 360);

sliderz.setSnapToTicks(true);

sliderz.setPaintTicks(true);

sliderz.setMajorTickSpacing(20);

sliderz.setMinorTickSpacing(5);

sliderz.addChangeListener(

new ChangeListener(){

public void stateChanged(ChangeEvent event){

JSlider source = (JSlider) event.getSource();

zrot= source.getValue();

}

});

Thread.sleep(100);

slidert = new DetaSlider(1, 150);

slidert.setSnapToTicks(true);

slidert.setPaintTicks(true);

slidert.setMajorTickSpacing(20);

slidert.setMinorTickSpacing(1);

slidert.addChangeListener(

new ChangeListener(){

public void stateChanged(ChangeEvent event) {

JSlider source = (JSlider)event.getSource();

irot= source.getValue();

}

});

top = new DetaButton("上", 100, 50, Color.red);

top.addActionListener(

new ActionListener() {

public void actionPerformed(ActionEvent e){

brot+=0.5;

}

});

down = new DetaButton("下", 100, 50, Color.orange);

down.addActionListener(

new ActionListener() {

public void actionPerformed(ActionEvent e){

brot-=0.5;

}

});

left = new DetaButton("左", 100, 50, Color.pink);

left.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

lrot-=0.5;

}

});

right = new DetaButton("右", 100, 50, Color.green);

right.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

lrot+=0.5;

}

});

bhl= new DetaButton("功效森林", 100, 50, Color.red);

bhl.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

String searchYaos= u.coAuthorForWord.bootFlowerForest(rootWord, true);

ZhongYaoSearch.zhongYaoSearch(u, searchYaos, "", searchYaos);

}

});

cfc= new DetaButton("属性植株", 100, 50, Color.orange);

cfc.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

bootClassificationMap(rootWord, true);

ZhongYaoSearch.zhongYaoSearch(u, u.key, "", u.key);

}

});

whh= new DetaButton("禁忌森林", 200, 50, Color.magenta);

whh.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

String searchYaos= u.coAuthorForWord.bootFlowerSea(rootWord, true);

ZhongYaoSearch.zhongYaoSearch(u, searchYaos, "", searchYaos);

}

});

rst= new DetaButton("重置", 200, 50, Color.pink);

rst.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

hook= !hook;

if(null== jframe) {

u.coAuthorForWord.canvas.setSize(900, 750);

jframe= new JFrame("药物观测");

jframe.setLayout(null);

jframe.setSize(900, 750);

jframe.setVisible(false);

jframe.setDefaultCloseOperation(JFrame.HIDE\_ON\_CLOSE);

}

//拿出来

if(hook) {

animator.stop();

u.coAuthorForWord.canvas.setSize(900, 750);

jframe.add(u.coAuthorForWord.canvas);

jframe.setVisible(true);

u.coAuthorForWord.canvas.validate();

u.coAuthorForWord.validate();

jframe.validate();

animator.start();

}else {

animator.stop();

u.coAuthorForWord.canvas.setSize(650, 310);

u.coAuthorForWord.add(u.coAuthorForWord.canvas);

jframe.setVisible(false);

u.coAuthorForWord.canvas.validate();

u.coAuthorForWord.validate();

u.coAuthorForWord.xrot= 18.0f;

u.coAuthorForWord.yrot= 2.0f;

u.coAuthorForWord.zrot= 1.0f;

u.coAuthorForWord.trot= 6.0f;

u.coAuthorForWord.prot= 1.0f;

u.coAuthorForWord.lrot= -3.5f;

u.coAuthorForWord.brot= 1.0f;

u.coAuthorForWord.irot= 78.0f;

u.coAuthorForWord.srot= 0.7f;

animator.start();

}

}

});

save = new DetaButton("保存", 200, 50, Color.green);

// rst.setBounds(0, 0, 100, 20);

save.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

xrotr= xrot;

yrotr= yrot;

zrotr= zrot;

trotr= trot;

protr= prot;

lrotr= lrot;

brotr= brot;

irotr= irot;

srotr= srot;

}

});

load = new DetaButton("读取", 200, 50, Color.magenta);

// rst.setBounds(0, 0, 100, 20);

load.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){

xrot= xrotr;

yrot= yrotr;

zrot= zrotr;

trot= trotr;

prot= protr;

lrot= lrotr;

brot= brotr;

irot= irotr;

srot= srotr;

}

});

buttonBox.add(top);

buttonBox.add(down);

buttonBox.add(left);

Thread.sleep(100);

buttonBox.add(right);

buttonBox.add(whh);

buttonBox.add(bhl);

buttonBox.add(cfc);

buttonBox.add(rst);

buttonBox.add(save);

buttonBox.add(load);

Box buttonBoxX = new Box(BoxLayout.X\_AXIS);

JButton xzb = new DetaButton("X坐标", 200, 50, Color.pink);

buttonBoxX.add(xzb);

buttonBoxX.add(sliderx);

sliderBoxx1.add(buttonBoxX);

Box buttonBoxY = new Box(BoxLayout.X\_AXIS);

JButton yzb = new DetaButton("Y坐标", 200, 50, Color.pink);

buttonBoxY.add(yzb);

buttonBoxY.add(slidery);

sliderBoxx1.add(buttonBoxY);

Box buttonBoxZ = new Box(BoxLayout.X\_AXIS);

JButton zzb = new DetaButton("Z坐标", 200, 50, Color.pink);

buttonBoxZ.add(zzb);

buttonBoxZ.add(sliderz);

sliderBoxx2.add(buttonBoxZ);

Box buttonBoxT = new Box(BoxLayout.X\_AXIS);

JButton tzb = new DetaButton("T坐标", 200, 50, Color.pink);

buttonBoxT.add(tzb);

buttonBoxT.add(slidert);

sliderBoxx2.add(buttonBoxT);

sliderBox.add(sliderBoxx1);

sliderBox.add(sliderBoxx2);

sliderBox.add(buttonBox);

sliderBox.setBounds(0, 350-50, 627, 300);

add(sliderBox);

}

private void centerWindow(Component frame){

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

Dimension frameSize = frame.getSize();

if (frameSize.width > screenSize.width)

frameSize.width = screenSize.width;

if (frameSize.height > screenSize.height)

frameSize.height = screenSize.height;

frame.setLocation((screenSize.width - frameSize.width) >> 1,

(screenSize.height - frameSize.height) >> 1);

}

@SuppressWarnings("static-access")

public void init(GLAutoDrawable drawable) {

gl = drawable.getGL().getGL2();

glu = new GLU();

glut= new GLUT();

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

gl.glEnable(GL.GL\_TEXTURE\_2D); gl.glTexParameteri(gl.GL\_TEXTURE\_2D,gl.GL\_TEXTURE\_MIN\_FILTER,gl.GL\_LINEAR); gl.glTexParameteri(gl.GL\_TEXTURE\_2D,gl.GL\_TEXTURE\_MAG\_FILTER,gl.GL\_NEAREST);

try {

Thread.sleep(100);

} catch (InterruptedException e1) {

e1.printStackTrace();

}

gl.glEnable(GL.GL\_DEPTH\_TEST);

gl.glDepthFunc(GL.GL\_LEQUAL);

gl.glShadeModel(GL2.GL\_SMOOTH);

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

gl.glClearColor(0f, 0f, 0f, 0f);

gl.glHint(gl.GL\_PERSPECTIVE\_CORRECTION\_HINT, gl.GL\_NICEST);

}

public void display(GLAutoDrawable drawable) {

// System.out.println(xrot);

// System.out.println(yrot);

// System.out.println(zrot);

// System.out.println(trot);

// System.out.println(prot);

// System.out.println(lrot);

// System.out.println(brot);

// System.out.println(irot);

// System.out.println(srot);

// System.out.println("");

GL2 gl = drawable.getGL().getGL2();

gl.glClear(GL.GL\_COLOR\_BUFFER\_BIT | GL.GL\_DEPTH\_BUFFER\_BIT);

gl.glLoadIdentity();

gl.glTranslatef(lrot, brot, -6.0f);

gl.glRotatef(xrot, 1, 0.0f, 0.0f);

gl.glRotatef(yrot, 0.0f, 1, 0.0f);

gl.glRotatef(zrot, 0.0f, 0.0f, 1);

gl.glPointSize(prot);

gl.glOrtho(0, trot, 0, trot, 1, -1);

int section[]= new int[21];

section [0]= 32;

section [1]= 32;

section [2]= 32;

section [3]= 32;

section [4]= 10;

section [5]= 10;

section [6]= 10;

section [7]= 10;

section [8]= 10;

section [9]= 10;

section [10]= 10;

section [11]= 10;

float a= 10;

double root[]= new double[3];

didLeaf.clear();

try {

drawWords(section, 0, root, a, 0, 0, this.rootLeaf, didLeaf);

} catch (IOException | InterruptedException e) {

e.printStackTrace();

}

gl.glEnd();

gl.glFlush();

}

@SuppressWarnings({"unchecked", "static-access"})

private void drawWords(int []s, int se, double[] root, float a, int j, int k

, Map<String, Object> root3, Map<String, Integer> didLeaf) throws IOException, InterruptedException {

Thread.sleep(3);

if(root3 == null || root3.size() == 0) {

return;

}

int i= 0;

s[se]= root3.size()> 20? 20: root3.size();

Iterator<String> iteratorLeaf = root3.keySet().iterator();

Here:

while(iteratorLeaf.hasNext()) {

if(k>= irot) {

return;

}

double a1= 1;

String leafName= iteratorLeaf.next();

if(leafName.length()== 0) {

continue Here;

}

int type= 0;

Map<String, Object> leaf = (Map<String, Object>) root3.get(leafName);

if(leaf.size() > 0) {

a1 = 1 ;

}

k++;

if(k> 25) {

return;

}

// findDup= true;

if(se< 2) {

if(leafName.equalsIgnoreCase("功能")

||leafName.equalsIgnoreCase("引起")

||leafName.equalsIgnoreCase("缺乏")

||leafName.equalsIgnoreCase("作用")

||leafName.equalsIgnoreCase("组成")

||leafName.equalsIgnoreCase("输出")

||leafName.equalsIgnoreCase("各种")

||leafName.equalsIgnoreCase("引起")

||leafName.equalsIgnoreCase("意义")

||leafName.equalsIgnoreCase("传导")

||leafName.equalsIgnoreCase("手术")

||leafName.equalsIgnoreCase("性性")

||leafName.equalsIgnoreCase("药物")

||leafName.equalsIgnoreCase("过量")

||leafName.equalsIgnoreCase("时可")

||leafName.equalsIgnoreCase("治疗")

||leafName.equalsIgnoreCase("预防")

||leafName.equalsIgnoreCase("注射")

||leafName.equalsIgnoreCase("应用")

||leafName.equalsIgnoreCase("适用")

||leafName.equalsIgnoreCase("适用于")

||leafName.equalsIgnoreCase("适应")) {

continue Here;

}

}

int freq= -1;

if(frequencyLeaf.containsKey(leafName)) {

freq= frequencyLeaf.get(leafName);

}

if((se< 2)

||(se== 3)

||(se== 2&& zybc== 1&& (freq== -1))

||(se== 2&&(freq>= u.lookrot))) {

gl.glBegin(gl.GL\_LINES);

if(j%7==0) {

gl.glColor3f(0,0,255);

}

if(j%7==1) {

gl.glColor3f(0,255,0);

}

if(j%7==2) {

gl.glColor3f(255,0,0);

}

if(j%7==3) {

gl.glColor3f(255,0,255);

}

if(j%7==4) {

gl.glColor3f(255,255,0);

}

if(j%7==5) {

gl.glColor3f(255,255,255);

}

if(j%7==6) {

gl.glColor3f(0,255,255);

}

gl.glVertex3d(root[0], root[1], root[2]);

gl.glVertex3d(root[0] + Math.cos((2 \* Math.PI / s[se]) \* i) \* a

, root[1] + Math.sin((2 \* Math.PI / s[se]) \* i) \* a

, root[2] + (srot\*a1));

gl.glEnd();

gl.glColor3f(0,0,255);

gl.glRasterPos3d(root[0] + Math.cos((2 \* Math.PI / s[se]) \* i) \* a

, root[1] + Math.sin((2 \* Math.PI / s[se]) \* i) \* a

, root[2] + (srot\*a1));

String prePrint = leafName;

Font font = new Font(prePrint,20, 20);

try {

checkImageBuf = GLBuffers.newDirectByteBuffer(20 \* prePrint.length()\*22 \* 3);

}catch(Exception e) {

System.out.println("");

}

BufferedImage image = new BufferedImage(prePrint.length()\*32, 20, BufferedImage.TYPE\_INT\_RGB);

Graphics2D ga = image.createGraphics();

ga.setColor(Color.white);

//int type= 0;

if(pos.containsKey(leafName)){

if(pos.get(leafName).contains("名")) {

ga.setColor(new Color(255,125,255));

type= 1;

}

if(pos.get(leafName).contains("动")) {

ga.setColor(new Color(125,255,125));

}

if(pos.get(leafName).contains("形")) {

ga.setColor(new Color(255, 255, 125));

}

if(pos.get(leafName).contains("谓")) {

ga.setColor(new Color(125, 125, 255));

}

}

int AMV\_MVS\_VSQX = 1;

int AMV\_MVS\_VSQY = 17;

this.weightPrint(prePrint, ga, AMV\_MVS\_VSQX, AMV\_MVS\_VSQY, font);

this.heightPrint(prePrint, ga, AMV\_MVS\_VSQX, AMV\_MVS\_VSQY, font);

checkImageBuf.clear();

int h = image.getHeight();

for (int i1 = h-1; i1 >= 0; i1--){

for (int j1 = 0; j1 < prePrint.length()\*20; j1++){

checkImageBuf.put((byte)(image.getRGB(j1, i1) & (byte)0xFF));

checkImageBuf.put((byte)(image.getRGB(j1, i1)>>8 & (byte)0xFF));

checkImageBuf.put((byte)(image.getRGB(j1, i1)>>16 & (byte)0xFF));

}

}

checkImageBuf.rewind();

try {

gl.glDrawPixels(prePrint.length()\*20, 20, GL2.GL\_RGB, GL.GL\_UNSIGNED\_BYTE, checkImageBuf);

}catch(Exception e) {

System.out.println("");

}

double root1[] = new double[3];

root1[0] = root[0] + Math.cos((2 \* Math.PI / s[se]) \* i) \* a;

root1[1] = root[1] + Math.sin((2 \* Math.PI / s[se]) \* i) \* a;

root1[2] = root[2] + (srot\* a1);

//j = i;

if(!didLeaf.containsKey(leafName)&& findLeaf.containsKey(leafName)) {

didLeaf.put(leafName, 1);

if(type== 1) {

if(zynk== 0) {

drawWords(s, se+1, root1, a \* 0.5f , j+i, k, leaf, didLeaf);

}else {

drawWords(s, se+1, root1, a \* 0.36f , j+i, k, leaf, didLeaf);

}

}else {

drawWords(s, se+1, root1, a \* 0.5f , j+i, k, leaf, didLeaf);

}

}

}

i++;

}

}

public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {

float fAspect;

if (height == 0) {

height = 1;

}

gl.glViewport(0, 0, width, height);

fAspect = (float) width / height;

gl.glMatrixMode(GL2.GL\_PROJECTION);

gl.glLoadIdentity();

glu.gluPerspective(53.0f, fAspect, 1.0, 400.0);

gl.glMatrixMode(GL2.GL\_MODELVIEW);

gl.glLoadIdentity();

}

public void dispose(GLAutoDrawable arg0) {

}

public static void main(String[] args) throws HeadlessException, InterruptedException {

CoAuthorForWord app= new CoAuthorForWord(null, null, null);

app.setSize(850, 700);

app.setVisible(true);

JFrame f = new JFrame();

f.add(app);

f.setTitle("ButtonDemo");

f.setLocationRelativeTo(null);

f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

f.setSize(650,700);

f.setVisible(true);

}

//hai

public String bootFlowerSea(String string, boolean isButton) {

zynk= 0;

zybc= 1;

if(rootWord == null) {

return "";

}

if(rootWord.equalsIgnoreCase(string) && isButton == false) {

return "";

}

rootWord= string;

rootLeaf.clear();

findLeaf.clear();

frequencyLeaf.clear();

findLeaf.put(rootWord, 1);

frequencyLeaf.put(rootWord, 1);

Map<String, Object> leaf= new ConcurrentHashMap<>();

int depth= 0;

Map<String, Object> subLeaf= kernerSea(rootWord, leaf, findLeaf, depth);

leaf.put(rootWord, subLeaf);

rootLeaf= leaf;

String output= subLeaf.keySet().toString();

return output;

}

public void bootClassificationMap(String string, boolean isButton) {

zynk= 0;

zybc= 1;

if(rootWord== null) {

return;

}

if(rootWord.equalsIgnoreCase(string) && isButton == false) {

return;

}

rootWord= string;

rootLeaf.clear();

findLeaf.clear();

frequencyLeaf.clear();

findLeaf.put(rootWord, 1);

frequencyLeaf.put(rootWord, 1);

Map<String, Object> leaf= new ConcurrentHashMap<>();

int depth= 0;

Map<String, Object> subLeaf= kernerClassification(rootWord, leaf, findLeaf, depth);

leaf.put(rootWord, subLeaf);

rootLeaf= leaf;

}

public String bootFlowerForest(String string, boolean isButton) {

zynk= 0;

zybc= 1;

if(rootWord== null) {

return "";

}

if(rootWord.equalsIgnoreCase(string) && isButton == false) {

return "";

}

rootWord= string;

rootLeaf.clear();

findLeaf.clear();

frequencyLeaf.clear();

findLeaf.put(rootWord, 1);

frequencyLeaf.put(rootWord, 1);

Map<String, Object> leaf = new ConcurrentHashMap<String, Object>();

int depth = 0;

Map<String, Object> subLeaf = kernerForest(rootWord, leaf, findLeaf, depth);

leaf.put(rootWord, subLeaf);

rootLeaf = leaf;

String output= subLeaf.keySet().toString();

return output;

}

public void bootDetaDnnFlowerForest(App app, String mingCheng, String[] dnn, boolean isButton) {

// TODO Auto-generated method stub

zynk= 1;

zybc= 1;

if(rootWord== null) {

return;

}

if(dnn.length< 2&& isButton== false) {

return;

}

rootWord= mingCheng;

rootLeaf.clear();

findLeaf.clear();

frequencyLeaf.clear();

findLeaf.put(mingCheng, 1);

frequencyLeaf.put(mingCheng, 1);

Map<String, Object> leaf = new ConcurrentHashMap<String, Object>();

int depth = 0;

Map<String, Object> subLeaf = kernerDnnForest(app, dnn, leaf, findLeaf, depth);

leaf.put(mingCheng, subLeaf);

rootLeaf = leaf;

}

public void bootDetaBingMingDnnFlowerForest(App app, String mingCheng, String[] dnn, boolean isButton) {

// TODO Auto-generated method stub

zynk= 1;

zybc= 0;

u.lookrot= 2;

if(rootWord== null) {

return;

}

if(dnn.length< 2&& isButton== false) {

return;

}

rootWord= mingCheng;

rootLeaf.clear();

findLeaf.clear();

frequencyLeaf.clear();

findLeaf.put(mingCheng, 1);

frequencyLeaf.put(mingCheng, 1);

Map<String, Object> leaf= new ConcurrentHashMap<String, Object>();

int depth= 0;

Map<String, Object> subLeaf= kernerDnnBingMingForest(app, dnn, leaf, findLeaf, depth);

leaf.put(mingCheng, subLeaf);

rootLeaf= leaf;

}

public String bootZynkFlowerForest(String fangji, String zucheng, boolean isButton) {

zynk= 1;

zybc= 1;

if(rootWord==null) {

return "";

}

if(rootWord.equalsIgnoreCase(fangji) && isButton == false) {

return "";

}

rootWord= fangji;

rootLeaf.clear();

findLeaf.clear();

frequencyLeaf.clear();

findLeaf.put(fangji, 1);

frequencyLeaf.put(fangji, 1);

Map<String, Object> leaf = new ConcurrentHashMap<String, Object>();

int depth = 0;

Map<String, Object> subLeaf = kernerZynkForest(zucheng, leaf, findLeaf, depth);

leaf.put(fangji, subLeaf);

String output= subLeaf.keySet().toString();

rootLeaf = leaf;

return output;

}

public void bootZynkJFlowerForest(String fangji, String zucheng, boolean isButton) {

// TODO Auto-generated method stub

zynk= 1;

zybc= 1;

if(rootWord==null) {

return;

}

if(rootWord.equalsIgnoreCase(fangji) && isButton == false) {

return;

}

rootWord = fangji;

rootLeaf.clear();

findLeaf.clear();

frequencyLeaf.clear();

findLeaf.put(fangji, 1);

frequencyLeaf.put(fangji, 1);

Map<String, Object> leaf = new ConcurrentHashMap<String, Object>();

int depth = 0;

Map<String, Object> subLeaf = kernerZynkJForest(zucheng, leaf, findLeaf, depth);

leaf.put(fangji, subLeaf);

rootLeaf = leaf;

}

//dnn bingming

private Map<String, Object> kernerDnnBingMingForest(App app, String[] dnn, Map<String, Object> root, Map<String, Integer> findLeaf, int depth) {

Map<String, Object> leaf= new ConcurrentHashMap<String, Object>();

if(++depth> 6) {

return leaf;

}

int max= dnn.length> 18? 18: dnn.length;

for(int i= 1; i< max; i++){

String tempRec= dnn[i].split(":")[0];

if(tempRec.length()> 1) {

if(this.pos.containsKey(tempRec)) {

int find= 0;

if(pos.get(tempRec).contains("名")) {

find++;

}

if(pos.get(tempRec).contains("动")) {

find++;

}

if(pos.get(tempRec).contains("形")) {

find++;

}

if(pos.get(tempRec).contains("谓")) {

find++;

}

if(find> 0) {

if(!findLeaf.containsKey(tempRec)) {

findLeaf.put(tempRec, 1);

frequencyLeaf.put(tempRec, -1);

Iterator<String> iterator= u.xyscPage.dic\_shi\_ying.keySet().iterator();

Map<String, Object> gongxiao= new ConcurrentHashMap<String, Object>();

Here:

while(iterator.hasNext()) {

String yaoming= iterator.next();

Map<String, Object> gongxiaoyaoming= new ConcurrentHashMap<String, Object>();

String li= u.xyscPage.dic\_shi\_ying.get(yaoming).toString();

li+= u.xyscPage.dic\_yao\_li.get(yaoming).toString();

if(li.contains(tempRec)) {

String hai= u.xyscPage.dic\_bu\_liang\_fan\_ying.get(yaoming).toString();

hai+= u.xyscPage.dic\_zhu\_yi.get(yaoming).toString();

String xw= li;

if(!frequencyLeaf.containsKey(yaoming) ) {

frequencyLeaf.put(yaoming, 1);

if(app.risk\_filter\_box.isSelected()) {

String temp= app.name\_filter.getText();

for(int j=0;j<temp.length();j++) {

if(hai.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

//li

if(app.feel\_filter\_box.isSelected()) {

String temp= app.name\_filter.getText();

for(int j= 0; j< temp.length(); j++) {

if(li.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

//xw

if(!app.nameFeelFilter.getText().isEmpty()) {

String temp= app.nameFeelFilter.getText();

for(int j=0;j<temp.length();j++) {

if(xw.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

//sm

if(app.shuming\_filter\_box.isSelected()) {

String temp= app.name\_filter.getText();

for(int j=0;j<temp.length();j++) {

if(yaoming.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

yaoming=yaoming.split("适应症")[0];

yaoming=yaoming.split("（")[0];

yaoming=yaoming.split("\\(")[0];

gongxiao.put(yaoming, gongxiaoyaoming);

}else {

frequencyLeaf.put(yaoming, frequencyLeaf.get(yaoming)+ 1);

//hai

}

// }

}

}

leaf.put(tempRec, gongxiao);

}else {

frequencyLeaf.put(tempRec, frequencyLeaf.get(tempRec) + 1);

}

}

}

}

}

return leaf;

}

//dnn

private Map<String, Object> kernerDnnForest(App app, String[] dnn, Map<String, Object> root, Map<String, Integer> findLeaf, int depth) {

Map<String, Object> leaf= new ConcurrentHashMap<String, Object>();

if(++depth> 6) {

return leaf;

}

int max= dnn.length> 22? 22: dnn.length;

for(int i= 1; i< max; i++){

String tempRec= dnn[i].split(":")[0];

if(tempRec.length()> 1) {

if(this.pos.containsKey(tempRec)) {

int find= 0;

if(pos.get(tempRec).contains("名")) {

find++;

}

if(pos.get(tempRec).contains("动")) {

find++;

}

if(pos.get(tempRec).contains("形")) {

find++;

}

if(pos.get(tempRec).contains("谓")) {

find++;

}

if(find> 0) {

if(!findLeaf.containsKey(tempRec)) {

findLeaf.put(tempRec, 1);

frequencyLeaf.put(tempRec, -1);

Iterator<String> iterator= u.dic\_li.keySet().iterator();

Map<String, Object> gongxiao= new ConcurrentHashMap<String, Object>();

Here:

while(iterator.hasNext()) {

String yaoming= iterator.next();

Map<String, Object> gongxiaoyaoming= new ConcurrentHashMap<String, Object>();

if(u.dic\_li.get(yaoming).toString().contains(tempRec)) {

String hai= u.dic\_hai.get(yaoming).toString();

String li= u.dic\_li.get(yaoming).toString();

String xw= u.dic\_xw.get(yaoming).toString();

String jm= u.dic\_jm.get(yaoming).toString().replaceAll("\\s\*", "");

if(!frequencyLeaf.containsKey(yaoming)) {

frequencyLeaf.put(yaoming, 1);

//hai

if(app.risk\_filter\_box.isSelected()) {

String temp= app.name\_filter.getText();

for(int j=0;j<temp.length();j++) {

if(hai.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

//li

if(app.feel\_filter\_box.isSelected()) {

String temp= app.name\_filter.getText();

for(int j= 0; j< temp.length(); j++) {

if(li.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

//xw

if(!app.nameFeelFilter.getText().isEmpty()) {

String temp= app.nameFeelFilter.getText();

for(int j=0;j<temp.length();j++) {

if(xw.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

//jm

if(null!= app.name\_filter\_not\_have.getText()) {

if(!app.name\_filter\_not\_have.getText().replace(" ", "").isEmpty()) {

String temp= app.name\_filter\_not\_have.getText();

int findJM= 0;

for(int j= 0; j< temp.length(); j++) {

if(jm.contains(""+ temp.charAt(j))) {

findJM= 1;

}

}

if(findJM== 0) {

continue Here;

}

}

}

//sm

if(app.shuming\_filter\_box.isSelected()) {

String temp= app.name\_filter.getText();

for(int j=0;j<temp.length();j++) {

if(yaoming.contains(""+ temp.charAt(j))) {

continue Here;

}

}

}

gongxiao.put(yaoming, gongxiaoyaoming);

}else {

frequencyLeaf.put(yaoming, frequencyLeaf.get(yaoming)+ 1);

}

// }

}

}

leaf.put(tempRec, gongxiao);

}else {

frequencyLeaf.put(tempRec, frequencyLeaf.get(tempRec) + 1);

}

}

}

}

}

return leaf;

}

//neike JINJI

private Map<String, Object> kernerZynkJForest(String temp, Map<String, Object> root, Map<String, Integer> findLeaf, int depth) {

Map<String, Object> leaf= new ConcurrentHashMap<String, Object>();

if(++depth>6) {

return leaf;

}

int ml=0;

Map<String, WordFrequency> map = \_A.parserMixStringByReturnFrequencyMap(temp);

Iterator<String> it= map.keySet().iterator();

int max= 0;

while(it.hasNext()) {

String tempRec= it.next();

if(!findLeaf.containsKey(tempRec)) {

findLeaf.put(tempRec, 1);

frequencyLeaf.put(tempRec, -1);

if(u.dic\_map.containsKey(tempRec)) {

Map<String, Object> subLeaf= new ConcurrentHashMap<String, Object>();

if(u.dic\_hai.containsKey(tempRec)) {

if(!u.dic\_hai.get(tempRec).toString().contains("笔记")) {

Map<String, Object> leafLiSub= new ConcurrentHashMap<String, Object>();

subLeaf.put(u.dic\_hai.get(tempRec).toString(), leafLiSub);

}

}

if(max++ < 25) {

leaf.put(tempRec, subLeaf);

}

//return leaf;

}else if(ml < 8) {

Map<String, Object> subLeaf = kernerForest(tempRec, leaf, findLeaf, depth);

if(subLeaf.size() > 0) {

leaf.put(tempRec, subLeaf);

ml++;

}

}

}else {

frequencyLeaf.put(tempRec, frequencyLeaf.get(tempRec) + 1);

}

}

return leaf;

}

//li

private Map<String, Object> kernerForest(String temp, Map<String, Object> root, Map<String, Integer> findLeaf, int depth) {

Map<String, Object> leaf = new ConcurrentHashMap<String, Object>();

if(++depth>6) {

return leaf;

}

//int ml=0;

if(u.dic\_li.containsKey(temp) || u.dic\_yl.containsKey(temp) ||u.dic\_zf.containsKey(temp)) {

Map<String, WordFrequency> map = \_A.parserMixStringByReturnFrequencyMap(

(u.dic\_li.get(temp) == null? "":u.dic\_li.get(temp))+

(u.dic\_yl.get(temp) == null? "": "")+

(u.dic\_zf.get(temp) == null? "": "")

);

Iterator<String> it= map.keySet().iterator();

while(it.hasNext()) {//gongxiao

String tempRec= it.next();

if(tempRec.length()> 1) {

if(this.pos.containsKey(tempRec)) {

int find= 0;

if(pos.get(tempRec).contains("名")) {

find++;

}

if(pos.get(tempRec).contains("动")) {

find++;

}

if(pos.get(tempRec).contains("形")) {

find++;

}

if(pos.get(tempRec).contains("谓")) {

find++;

}

if(find> 0) {

if(!findLeaf.containsKey(tempRec)) {

findLeaf.put(tempRec, 1);

frequencyLeaf.put(tempRec, -1);

Iterator<String> iterator= u.dic\_li.keySet().iterator();

Map<String, Object> gongxiao= new ConcurrentHashMap<String, Object>();

while(iterator.hasNext()) {

String yaoming= iterator.next();

Map<String, Object> gongxiaoyaoming= new ConcurrentHashMap<String, Object>();

if(u.dic\_li.get(yaoming).toString().contains(tempRec)) {

String hai= NullObject.checkNULL(u.dic\_hai.get(yaoming));

if(!hai.contains("慎")&&!hai.contains("禁")&&!hai.contains("恶")

&&!hai.contains("毒")&&!hai.contains("畏")

&&!hai.contains("孕")&&!hai.contains("忌")&&!hai.contains("胎")) {

if(!frequencyLeaf.containsKey(yaoming) ) {

frequencyLeaf.put(yaoming, 1);

String xingWeiRatio= u.dic\_xw.get(yaoming).toString();

String xingWei= u.dic\_xw.get(temp).toString();

String maiLuoRatio= u.dic\_jm.get(yaoming).toString();

String maiLuo= u.dic\_jm.get(temp).toString();

if(((xingWeiRatio.contains("辛")&&xingWei.contains("辛"))

||(xingWeiRatio.contains("温")&&xingWei.contains("温"))

||(xingWeiRatio.contains("燥")&&xingWei.contains("燥"))

||(xingWeiRatio.contains("热")&&xingWei.contains("热"))

||(xingWeiRatio.contains("苦")&&xingWei.contains("苦"))

||(xingWeiRatio.contains("寒")&&xingWei.contains("寒"))

||(xingWeiRatio.contains("凉")&&xingWei.contains("凉"))

)) {

if(maiLuoRatio.contains("心包")){

if((maiLuo.contains("心包"))) {

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if(!maiLuoRatio.contains("心包")&&maiLuoRatio.contains("心")){

if((!maiLuo.contains("心包")&&maiLuo.contains("心"))) {

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if((maiLuoRatio.contains("大肠")&& maiLuo.contains("大肠"))

||(maiLuoRatio.contains("小肠")&& maiLuo.contains("小肠"))

||(maiLuoRatio.contains("肝")&& maiLuo.contains("肝"))

||(maiLuoRatio.contains("脾")&& maiLuo.contains("脾"))

||(maiLuoRatio.contains("膀胱")&& maiLuo.contains("膀胱"))

||(maiLuoRatio.contains("胆")&& maiLuo.contains("胆"))

||(maiLuoRatio.contains("肺")&& maiLuo.contains("肺"))

||(maiLuoRatio.contains("肾")&& maiLuo.contains("肾"))

||(maiLuoRatio.contains("十二")&& maiLuo.contains("十二"))

||(maiLuoRatio.contains("任")&& maiLuo.contains("任"))

||(maiLuoRatio.contains("督")&& maiLuo.contains("督"))

||(maiLuoRatio.contains("三焦")&& maiLuo.contains("三焦"))

||(maiLuoRatio.contains("胃")&& maiLuo.contains("胃"))

){

gongxiao.put(yaoming, gongxiaoyaoming);

}

if((!maiLuoRatio.contains("大肠")&& !maiLuo.contains("大肠"))

&&(!maiLuoRatio.contains("小肠")&& !maiLuo.contains("小肠"))

&&(!maiLuoRatio.contains("肝")&& !maiLuo.contains("肝"))

&&(!maiLuoRatio.contains("脾")&& !maiLuo.contains("脾"))

&&(!maiLuoRatio.contains("膀胱")&& !maiLuo.contains("膀胱"))

&&(!maiLuoRatio.contains("胆")&& !maiLuo.contains("胆"))

&&(!maiLuoRatio.contains("肺")&& !maiLuo.contains("肺"))

&&(!maiLuoRatio.contains("肾")&& !maiLuo.contains("肾"))

&&(!maiLuoRatio.contains("十二")&& !maiLuo.contains("十二"))

&&(!maiLuoRatio.contains("任")&& !maiLuo.contains("任"))

&&(!maiLuoRatio.contains("督")&& !maiLuo.contains("督"))

&&(!maiLuoRatio.contains("三焦")&& !maiLuo.contains("三焦"))

&&(!maiLuoRatio.contains("胃")&& !maiLuo.contains("胃"))

&&(!maiLuoRatio.contains("肠")&& !maiLuo.contains("肠"))){

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if(((!xingWeiRatio.contains("辛")&&!xingWei.contains("辛")

&&!xingWeiRatio.contains("温")&&!xingWei.contains("温")

&&!xingWeiRatio.contains("燥")&&!xingWei.contains("燥")

&&!xingWeiRatio.contains("热")&&!xingWei.contains("热")

&&!xingWeiRatio.contains("苦")&&!xingWei.contains("苦")

&&!xingWeiRatio.contains("寒")&&!xingWei.contains("寒")

&&!xingWeiRatio.contains("凉")&&!xingWei.contains("凉")

))) {

if(maiLuoRatio.contains("心包")){

if((maiLuo.contains("心包"))) {

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if(!maiLuoRatio.contains("心包")&&maiLuoRatio.contains("心")){

if((!maiLuo.contains("心包")&&maiLuo.contains("心"))) {

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if((maiLuoRatio.contains("大肠")&& maiLuo.contains("大肠"))

||(maiLuoRatio.contains("小肠")&& maiLuo.contains("小肠"))

||(maiLuoRatio.contains("肝")&& maiLuo.contains("肝"))

||(maiLuoRatio.contains("脾")&& maiLuo.contains("脾"))

||(maiLuoRatio.contains("膀胱")&& maiLuo.contains("膀胱"))

||(maiLuoRatio.contains("胆")&& maiLuo.contains("胆"))

||(maiLuoRatio.contains("肺")&& maiLuo.contains("肺"))

||(maiLuoRatio.contains("肾")&& maiLuo.contains("肾"))

||(maiLuoRatio.contains("十二")&& maiLuo.contains("十二"))

||(maiLuoRatio.contains("任")&& maiLuo.contains("任"))

||(maiLuoRatio.contains("督")&& maiLuo.contains("督"))

||(maiLuoRatio.contains("三焦")&& maiLuo.contains("三焦"))

||(maiLuoRatio.contains("胃")&& maiLuo.contains("胃"))

){

gongxiao.put(yaoming, gongxiaoyaoming);

}

if((!maiLuoRatio.contains("大肠")&& !maiLuo.contains("大肠")

&&!maiLuoRatio.contains("小肠")&& !maiLuo.contains("小肠")

&&!maiLuoRatio.contains("肝")&& !maiLuo.contains("肝")

&&!maiLuoRatio.contains("脾")&& !maiLuo.contains("脾")

&&!maiLuoRatio.contains("膀胱")&& !maiLuo.contains("膀胱")

&&!maiLuoRatio.contains("胆")&& !maiLuo.contains("胆")

&&!maiLuoRatio.contains("肺")&& !maiLuo.contains("肺")

&&!maiLuoRatio.contains("肾")&& !maiLuo.contains("肾")

&&!maiLuoRatio.contains("十二")&& !maiLuo.contains("十二")

&&!maiLuoRatio.contains("任")&& !maiLuo.contains("任")

&&!maiLuoRatio.contains("督")&& !maiLuo.contains("督")

&&!maiLuoRatio.contains("三焦")&& !maiLuo.contains("三焦")

&&!maiLuoRatio.contains("胃")&& !maiLuo.contains("胃")

&&!maiLuoRatio.contains("肠")&& !maiLuo.contains("肠"))){

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

}else {

frequencyLeaf.put(yaoming, frequencyLeaf.get(yaoming) + 1);

}

}

}

}

leaf.put(tempRec, gongxiao);

}else {

frequencyLeaf.put(tempRec, frequencyLeaf.get(tempRec) + 1);

}

}

}

}

}

}

return leaf;

}

//neike

private Map<String, Object> kernerZynkForest(String temp, Map<String, Object> root, Map<String, Integer> findLeaf, int depth) {

Map<String, Object> leaf= new ConcurrentHashMap<String, Object>();

if(++depth>6) {

return leaf;

}

int ml=0;

int max= 0;

Map<String, WordFrequency> map = \_A.parserMixStringByReturnFrequencyMap(temp);

Iterator<String> it= map.keySet().iterator();

while(it.hasNext()) {

String tempRec= it.next();

if(!findLeaf.containsKey(tempRec)) {

findLeaf.put(tempRec, 1);

frequencyLeaf.put(tempRec, -1);

if(u.dic\_map.containsKey(tempRec)) {

Map<String, Object> subLeaf= new ConcurrentHashMap<String, Object>();

if(u.dic\_li.containsKey(tempRec)) {

Map<String, WordFrequency> liMap

= this.\_A.parserMixStringByReturnFrequencyMap(u.dic\_li.get(tempRec).toString());

Iterator<String> itli= liMap.keySet().iterator();

while(itli.hasNext()) {

String word= itli.next();

if(this.pos.containsKey(word)) {

if(word.length()> 1) {

if(!frequencyLeaf.containsKey(word)) {

frequencyLeaf.put(word, 1);

}else {

frequencyLeaf.put(word, frequencyLeaf.get(word) + 1);

}

Map<String, Object> leafLiSub = new ConcurrentHashMap<String, Object>();

if(pos.get(word).contains("动")) {

subLeaf.put(word, leafLiSub);

}

if(pos.get(word).contains("形")) {

subLeaf.put(word, leafLiSub);

}

if(pos.get(word).contains("谓")) {

subLeaf.put(word, leafLiSub);

}

}

}

}

}

if(max++ < 25) {

leaf.put(tempRec, subLeaf);

}

//return leaf;

}else if(ml < 8) {

Map<String, Object> subLeaf = kernerForest(tempRec, leaf, findLeaf, depth);

if(subLeaf.size() > 0) {

leaf.put(tempRec, subLeaf);

ml++;

}

}

}else {

frequencyLeaf.put(tempRec, frequencyLeaf.get(tempRec) + 1);

}

}

return leaf;

}

//gongxiao

private Map<String, Object> kernerClassification(String temp, Map<String, Object> root, Map<String, Integer> findLeaf,

int depth) {

Map<String, Object> leaf = new ConcurrentHashMap<>();

if(++depth>6) {

return leaf;

}

frequencyLeaf.put(temp,-1);

if(u.dic\_li.containsKey(temp)) {

String className=temp+"功效";

findLeaf.put(className, 1);

frequencyLeaf.put(className, -1);

//add sub sets

Map<String, WordFrequency> liMap = this.\_A.parserMixStringByReturnFrequencyMap(u.dic\_li.get(temp).toString());

Iterator<String> it = liMap.keySet().iterator();

Map<String, Object> leafLi = new ConcurrentHashMap<>();

int ml = 0;

while(it.hasNext()) {

String word = liMap.get(it.next()).getWord();

if(this.pos.containsKey(word)) {

if(word.length() > 1) {

Map<String, Object> leafLiSub = new ConcurrentHashMap<>();

leafLi.put(word, leafLiSub);

frequencyLeaf.put(word, -1);

}else if(ml < 10) {

Map<String, Object> leafLiSub = new ConcurrentHashMap<>();

if(leafLiSub.size() > 0) {

leaf.put(word, leafLiSub);

ml++;

}

ml++;

}

}

}

leaf.put(className, leafLi);

}

if(u.dic\_hai.containsKey(temp)) {

String className= temp+ "禁忌";

findLeaf.put(className, 1);

frequencyLeaf.put(className, -1);

Map<String, Object> leafLi= new ConcurrentHashMap<>();

String string= u.dic\_hai.get(temp).toString().length()> 40

? u.dic\_hai.get(temp).toString().substring(0, 40): u.dic\_hai.get(temp).toString();

leafLi.put(string, leafLi);

frequencyLeaf.put(string, -1);

leaf.put(className, leafLi);

}

if(u.dic\_xw.containsKey(temp)) {

String className=temp + "性味";

findLeaf.put(className, 1);

frequencyLeaf.put(className,-1);

//add sub sets

Map<String, WordFrequency> liMap = this.\_A.parserMixStringByReturnFrequencyMap(u.dic\_xw.get(temp).toString());

Iterator<String> it = liMap.keySet().iterator();

Map<String, Object> leafLi = new ConcurrentHashMap<>();

int ml = 0;

while(it.hasNext()) {

String word = liMap.get(it.next()).getWord();

if(this.pos.containsKey(word)) {

if(word.length() > 1) {

Map<String, Object> leafLiSub = new ConcurrentHashMap<>();

leafLi.put(word, leafLiSub);

frequencyLeaf.put(word, -1);

}else if(ml < 10) {

Map<String, Object> leafLiSub = new ConcurrentHashMap<>();

leafLi.put(word, leafLiSub);

frequencyLeaf.put(word, -1);

ml++;

}

}

}

leaf.put(className, leafLi);

}

if(u.dic\_yl.containsKey(temp)) {

String className=temp+"用量备注";

findLeaf.put(className, 1);

frequencyLeaf.put(className, -1);

//add sub sets

Map<String, Object> leafLi = new ConcurrentHashMap<>();

leafLi.put(u.dic\_yl.get(temp).toString(), leafLi);

frequencyLeaf.put(u.dic\_yl.get(temp).toString(), -1);

leaf.put(className, leafLi);

}

return leaf;

}

//hai

private Map<String, Object> kernerSea(String temp, Map<String, Object> root, Map<String, Integer> findLeaf, int depth) {

Map<String, Object> leaf= new ConcurrentHashMap<String, Object>();

if(++depth> 6) {

return leaf;

}

//int ml=0;

if(u.dic\_yw.containsKey(temp)) {

if(u.dic\_yw.get(temp).toString().isEmpty()) {

return leaf;

}

}else {

return leaf;

}

if(u.dic\_li.containsKey(temp)|| u.dic\_yl.containsKey(temp)|| u.dic\_zf.containsKey(temp)) {

Map<String, WordFrequency> map= \_A.parserMixStringByReturnFrequencyMap(

(u.dic\_li.get(temp)== null? "":u.dic\_li.get(temp))+

(u.dic\_yl.get(temp)== null? "": "")+

(u.dic\_zf.get(temp)== null? "": "")

);

Iterator<String> it= map.keySet().iterator();

while(it.hasNext()) {//gongxiao

String tempRec= it.next();

if(tempRec.length()> 1) {

if(this.pos.containsKey(tempRec)) {

int find= 0;

if(pos.get(tempRec).contains("名")) {

find++;

}

if(pos.get(tempRec).contains("动")) {

find++;

}

if(pos.get(tempRec).contains("形")) {

find++;

}

if(pos.get(tempRec).contains("谓")) {

find++;

}

if(find> 0) {

if(!findLeaf.containsKey(tempRec)) {

findLeaf.put(tempRec, 1);

frequencyLeaf.put(tempRec, -1);

Iterator<String> iterator= u.dic\_li.keySet().iterator();

Map<String, Object> gongxiao= new ConcurrentHashMap<String, Object>();

//int max= 0;

//bianli gongxiao

Here:

while(iterator.hasNext()) {

String yaoming= iterator.next();

Map<String, Object> gongxiaoyaoming= new ConcurrentHashMap<String, Object>();

if(u.dic\_li.get(yaoming).toString().contains(tempRec)) {

String hai= NullObject.checkNULL(u.dic\_hai.get(yaoming));

if(hai.contains("慎")||hai.contains("禁")||hai.contains("恶")

||hai.contains("毒")||hai.contains("畏")

||hai.contains("孕")||hai.contains("忌")||hai.contains("胎")) {

if(!yaoming.contains("穴")&& !frequencyLeaf.containsKey(yaoming) ) {

frequencyLeaf.put(yaoming, 1);

String xingWeiRatio= u.dic\_xw.get(yaoming).toString();

String xingWei= u.dic\_xw.get(temp).toString();

String maiLuoRatio= u.dic\_jm.get(yaoming).toString();

String maiLuo= u.dic\_jm.get(temp).toString();

if(u.risk\_filter\_box.isSelected()) {

String tempRisk= u.name\_filter.getText();

for(int j= 0; j< tempRisk.length(); j++) {

if(hai.contains(""+ tempRisk.charAt(j))) {

continue Here;

}

}

}

if(((xingWeiRatio.contains("辛")&&xingWei.contains("辛"))

||(xingWeiRatio.contains("温")&&xingWei.contains("温"))

||(xingWeiRatio.contains("燥")&&xingWei.contains("燥"))

||(xingWeiRatio.contains("热")&&xingWei.contains("热"))

||(xingWeiRatio.contains("苦")&&xingWei.contains("苦"))

||(xingWeiRatio.contains("寒")&&xingWei.contains("寒"))

||(xingWeiRatio.contains("凉")&&xingWei.contains("凉"))

)) {

if(maiLuoRatio.contains("心包")){

if((maiLuo.contains("心包"))) {

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if(!maiLuoRatio.contains("心包")&& maiLuoRatio.contains("心")){

if((!maiLuo.contains("心包")&& maiLuo.contains("心"))) {

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if((maiLuoRatio.contains("大肠")&& maiLuo.contains("大肠"))

||(maiLuoRatio.contains("小肠")&& maiLuo.contains("小肠"))

||(maiLuoRatio.contains("肝")&& maiLuo.contains("肝"))

||(maiLuoRatio.contains("脾")&& maiLuo.contains("脾"))

||(maiLuoRatio.contains("膀胱")&& maiLuo.contains("膀胱"))

||(maiLuoRatio.contains("胆")&& maiLuo.contains("胆"))

||(maiLuoRatio.contains("肺")&& maiLuo.contains("肺"))

||(maiLuoRatio.contains("肾")&& maiLuo.contains("肾"))

||(maiLuoRatio.contains("十二")&& maiLuo.contains("十二"))

||(maiLuoRatio.contains("任")&& maiLuo.contains("任"))

||(maiLuoRatio.contains("督")&& maiLuo.contains("督"))

||(maiLuoRatio.contains("三焦")&& maiLuo.contains("三焦"))

||(maiLuoRatio.contains("胃")&& maiLuo.contains("胃"))

){

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

if((!maiLuoRatio.contains("大肠")&& !maiLuo.contains("大肠")

&&!maiLuoRatio.contains("小肠")&& !maiLuo.contains("小肠")

&&!maiLuoRatio.contains("肝")&& !maiLuo.contains("肝")

&&!maiLuoRatio.contains("脾")&& !maiLuo.contains("脾")

&&!maiLuoRatio.contains("膀胱")&& !maiLuo.contains("膀胱")

&&!maiLuoRatio.contains("胆")&& !maiLuo.contains("胆")

&&!maiLuoRatio.contains("肺")&& !maiLuo.contains("肺")

&&!maiLuoRatio.contains("肾")&& !maiLuo.contains("肾")

&&!maiLuoRatio.contains("十二")&& !maiLuo.contains("十二")

&&!maiLuoRatio.contains("任")&& !maiLuo.contains("任")

&&!maiLuoRatio.contains("督")&& !maiLuo.contains("督")

&&!maiLuoRatio.contains("三焦")&& !maiLuo.contains("三焦")

&&!maiLuoRatio.contains("胃")&& !maiLuo.contains("胃")

)){

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if(((!xingWeiRatio.contains("辛")&&!xingWei.contains("辛")

&&!xingWeiRatio.contains("温")&&!xingWei.contains("温")

&&!xingWeiRatio.contains("燥")&&!xingWei.contains("燥")

&&!xingWeiRatio.contains("热")&&!xingWei.contains("热")

&&!xingWeiRatio.contains("苦")&&!xingWei.contains("苦")

&&!xingWeiRatio.contains("寒")&&!xingWei.contains("寒")

&&!xingWeiRatio.contains("凉")&&!xingWei.contains("凉"))

)) {

if(maiLuoRatio.contains("心包")){

if((maiLuo.contains("心包"))) {

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if(!maiLuoRatio.contains("心包")&&maiLuoRatio.contains("心")){

if((!maiLuo.contains("心包")&&maiLuo.contains("心"))) {

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

if((maiLuoRatio.contains("大肠")&& maiLuo.contains("大肠"))

||(maiLuoRatio.contains("小肠")&& maiLuo.contains("小肠"))

||(maiLuoRatio.contains("肝")&& maiLuo.contains("肝"))

||(maiLuoRatio.contains("脾")&& maiLuo.contains("脾"))

||(maiLuoRatio.contains("膀胱")&& maiLuo.contains("膀胱"))

||(maiLuoRatio.contains("胆")&& maiLuo.contains("胆"))

||(maiLuoRatio.contains("肺")&& maiLuo.contains("肺"))

||(maiLuoRatio.contains("肾")&& maiLuo.contains("肾"))

||(maiLuoRatio.contains("十二")&& maiLuo.contains("十二"))

||(maiLuoRatio.contains("任")&& maiLuo.contains("任"))

||(maiLuoRatio.contains("督")&& maiLuo.contains("督"))

||(maiLuoRatio.contains("三焦")&& maiLuo.contains("三焦"))

||(maiLuoRatio.contains("胃")&& maiLuo.contains("胃"))

){

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

if((!maiLuoRatio.contains("大肠")&& !maiLuo.contains("大肠") &&(!maiLuoRatio.contains("小肠")&& !maiLuo.contains("小肠"))

&&!maiLuoRatio.contains("肝")&& !maiLuo.contains("肝")

&&!maiLuoRatio.contains("脾")&& !maiLuo.contains("脾")

&&!maiLuoRatio.contains("膀胱")&& !maiLuo.contains("膀胱")

&&!maiLuoRatio.contains("胆")&& !maiLuo.contains("胆")

&&!maiLuoRatio.contains("肺")&& !maiLuo.contains("肺")

&&!maiLuoRatio.contains("肾")&& !maiLuo.contains("肾")

&&!maiLuoRatio.contains("十二")&& !maiLuo.contains("十二")

&&!maiLuoRatio.contains("任")&& !maiLuo.contains("任")

&&!maiLuoRatio.contains("督")&& !maiLuo.contains("督")

&&!maiLuoRatio.contains("三焦")&& !maiLuo.contains("三焦")

&&!maiLuoRatio.contains("胃")&& !maiLuo.contains("胃"))

){

gongxiaoyaoming.put(hai, new ConcurrentHashMap<>());

findLeaf.put(yaoming, 1);

gongxiao.put(yaoming, gongxiaoyaoming);

}

}

}

}

}

}

leaf.put(tempRec, gongxiao);

}else {

frequencyLeaf.put(tempRec, frequencyLeaf.get(tempRec) + 1);

}

}

}

}

}

}

return leaf;

}

public void weightPrint(String prePrint, Graphics2D g, int AMV\_MVS\_VSQX, int AMV\_MVS\_VSQY, Font font) {

g.setFont(font);

g.drawString(prePrint, AMV\_MVS\_VSQX, AMV\_MVS\_VSQY);

}

public void heightPrint(String prePrint, Graphics2D g, int AMV\_MVS\_VSQX, int AMV\_MVS\_VSQY, Font font) {

g.setFont(font);

g.drawString(prePrint, AMV\_MVS\_VSQX, AMV\_MVS\_VSQY);

}

public void mouseDragged(java.awt.event.MouseEvent arg0) {

if(arg0.getX()!=mousex) {

yrot += (arg0.getX()-mousex)/5;

}

if(arg0.getY()!=mousey) {

xrot += (arg0.getY()-mousey)/5;

}

mousex = arg0.getX();

mousey = arg0.getY();

}

public void mouseMoved(java.awt.event.MouseEvent arg0) {

}

public void keyPressed(KeyEvent arg0) {

if(arg0.getKeyCode()==38) {

brot += 0.5;

}

if(arg0.getKeyCode()==37) {

lrot -= 0.5;

}

if(arg0.getKeyCode()==40) {

brot -= 0.5;

}

if(arg0.getKeyCode()==39) {

lrot += 0.5;

}

}

public void keyReleased(KeyEvent arg0) {

}

public void keyTyped(KeyEvent arg0) {

}

public void mouseWheelMoved(MouseWheelEvent arg0) {

trot += arg0.getWheelRotation();

}

public void mouseClicked(MouseEvent arg0) {

}

public void mouseEntered(MouseEvent arg0) {

clicked = false;

}

public void mouseExited(MouseEvent arg0) {

clicked = false;

}

public void mousePressed(MouseEvent arg0) {

mousex = arg0.getX();

mousey = arg0.getY();

}

public void mouseReleased(MouseEvent arg0) {

}

}

花的筛选与观测

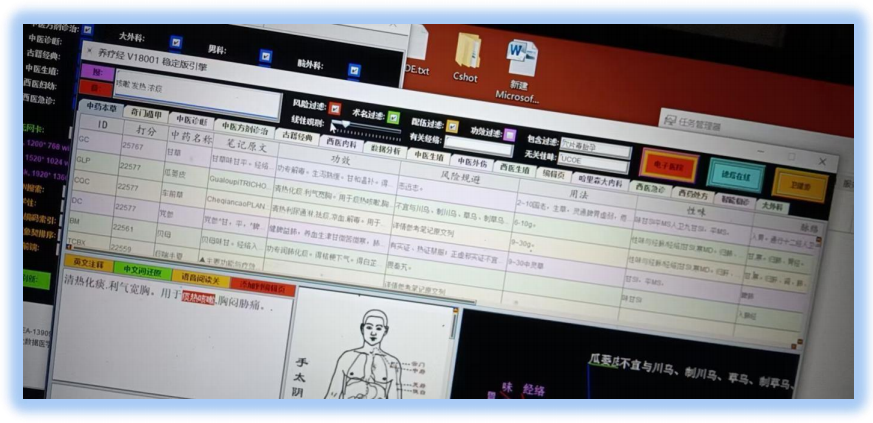


尝试下在搜索风热咳嗽的对症下去掉 S 性味**语义**元基筛选后, 出现的贝母，黄芩，前胡，牛蒡子， 黄花，丹草 6 味君药. 方

便大家的理解, S= I, S= Q, IQ 为酮基嘧啶, 温燥. 于是 D, M 中 C 寒为主等药

物列筛选出来. 同时生化罗盘中这里 S 属于土对症胃过滤, M 属于金对症骨肺保留, 下一步跟进研究我会放在第三卷, 第二卷

不做生化跟进, 仅仅做语义研究, 思维打止



这张图, 逐步开始大分子元基语义（这里是语义不是生化, 大家注意, 如果是语义归纳的元基就必须用语义, 生化归纳的元基

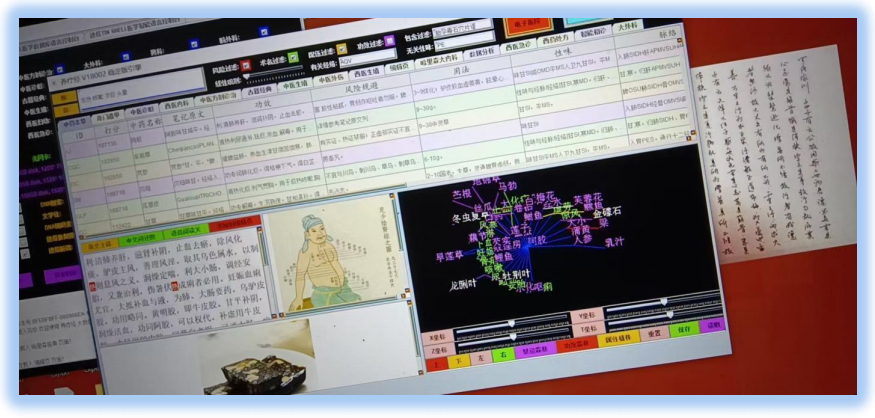
就必须用生化. 避免出错呀）性味筛选, 咳嗽发热浓痰对症出现 瓜蒌皮MD 贝母 SI 党参 MS 半夏 甘草 MS 车前草 MD 药物,

通过筛选温燥的元基过滤掉后, 结果展示.

上图搜索发热咳嗽 浓痰，开启风险过滤 胎孕毒穴片关键字， 然后无关性味过滤 UCOV 元基础成份，结果含有 甘草，

瓜蒌皮，车前草，党参，贝母，痰喘半夏等. 在配伍过滤开启后，禁忌明显相冲的药物已经被过滤掉. 线性观测调节，关键字比

重加重后筛选结果. 大大方便了职业医生的用药选择观测.



同理归纳不多解释了, 都是语义元基级别, 不做生化元基分析（第三卷研究）.

上图搜索发热咳嗽 浓痰 头晕，开启风险过滤 胎孕毒石穴片谱关键字， 然后有关经络保留 AQV 元基成份，无关性味过滤

PE 元基础成份，结果含有 阿胶，车前草，党参，贝母，瓜蒌皮，甘草等.在配伍过滤开启后，禁忌明显相冲的药物已经被过滤

掉

后序

DNA 无理级思维应用养疗经[17]跟进，目标：生化计算。归纳和提取下一些定义和中心点：

数据生化计算: 为数据赋予智慧生命特征，其所产生的一系列进化计算体系。

元基: 数据根据语义进行生命编码的基本单位，用于区分生化医学的 ACGTU 基元。

元基罗盘：通过元基的语义编码和化学腐蚀活性进行八卦分类排列的罗盘。

元基魔方：通过三维神经网络组成的元基矩阵用于生化路径智能计算思维拓展

元基卷积：通过元基编码与解码进行深度计算的逻辑思维模型。

元基视觉：通过元基的肽展公式变换进行图片像素级处理的生化视觉计算体系。

元基数据库：通过元基加密解密和数据元基变换进行数据库定义。

**重复重复再重复:**

1 养疗经[17]大数据软件不做行医用药执行权力，任何疾病请去医院找拥有职业资格证的医生看病.

2 一切生化的 实验室 物理 研究制造请在合格安全的并有法律部门认可的实验室 实行.

3 一切实验室的生化成品禁止在智慧体上进行试验.

4 微分催化[7]体系 生化计算体系 仍在基础研究阶段，任何国家政府机构和商业公司的生化商业发

展请遵循相关国家和地域的法律以及联合国的法律约束.

引用:

1, 罗瑶光, 中华人民共和国国家版权局, 德塔语言图灵工程 API\_10\_6\_1, 软著登字 3951366.

2, 罗瑶光, 中华人民共和国国家版权局, 数据分析算法引擎系统 1. 0. 2, 软著登字 4584594.

3, 罗瑶光, 中华人民共和国国家版权局, 德塔数据结构变量快速转换引擎系统, 软著登字 4607950.

4, 罗瑶光, 中华人民共和国国家版权局, 德塔 Socket 流可编程数据库语言引擎系统 API1. 0. 0, 软著登字 4317518.

5, 罗瑶光, 中华人民共和国国家版权局, 数据预测引擎系统 API1. 0. 0, 软著登字 5447819.

6, 罗瑶光, 中华人民共和国国家版权局, 德塔 ETL 可视化数据分析引擎系统 API1. 0. 2, 软著登字

4240558.

7, 罗瑶光, DNA 微分催化计算体系, GITHUB, https: //github. com/yaoguangluo

8, 罗瑶光, 德塔开源[8]体系, GITEE, https: //gitee. com/detachina

9, 罗瑶光, 罗荣武, 中华人民共和国国家版权局, 类人 DNA 与 神经元基于催化算子映射编码式, CN2020Z11L0333706, 国作登字-2021-A-00097017.

10, YaoguangLuo, RongwuLuo, The INITONS Catalytic Reflection Between Humanoid DNA and NeroCell, IE, ACM, 投稿 ID: A2050-ICITEE2020.

11, 罗瑶光, 中华人民共和国国家版权局, AOPM VECS IDUQ 肽展公式推导与元基编码进化计算以

及它的应用发现, CN2020Z11L0356797, 国作登字-2021-A-00042587.

12, 查锡良, 人民卫生出版社, 生物化学与分子生物学, ISBN978-7-117-26624-6, Page32, Page39, Page46, Page59, Page203.

13, 罗瑶光, 中华人民共和国国家版权局, DNA 催化与肽计算 和 AOPM-TXH-VECS-IDUQ 元基解码 V013-026 中文版本, CN2020Z11L0386462, 国作登字 2021-A-00042586.

14, 陆阳, 人民卫生出版社, 有机化学, ISBN978-7-117-26657-4, Page 101, Page 108, Page 114, Page 133,

Page142, Page 146, Page 186, Page 236.

15, 罗瑶光, 中华人民共和国国家知识产权局, 华瑞集, 国商字第 40126765, 39183063 号.

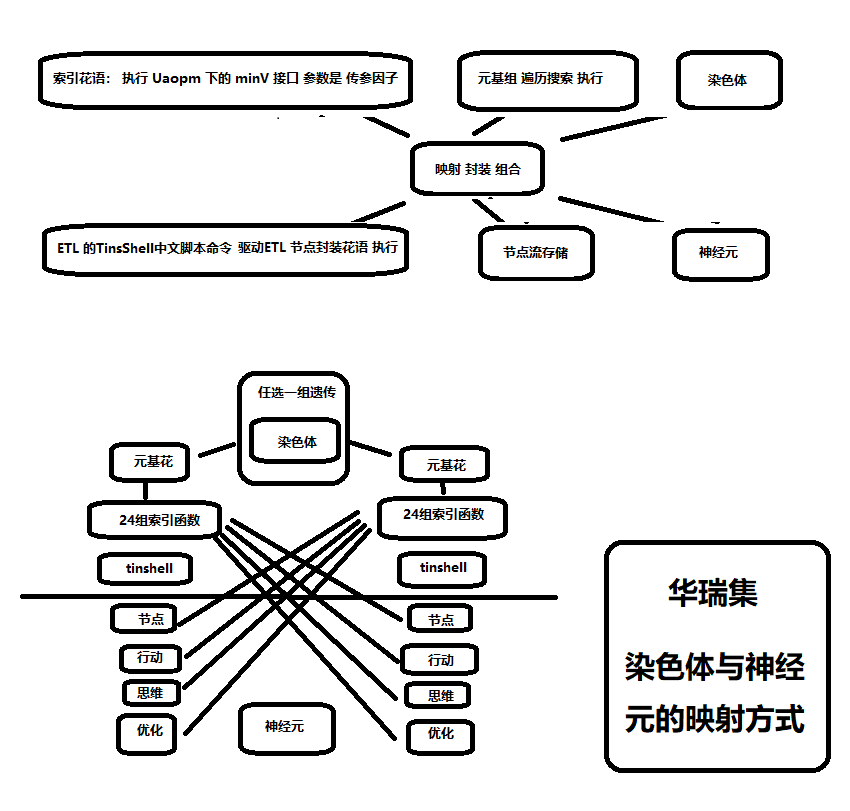
16, 罗瑶光, 中华人民共和国国家版权局, 华瑞集, 国作登字 2019-F-00943469, 00943473, 00943472.

17, 罗瑶光, 中华人民共和国国家知识产权局, 养疗经, 国商字第 40126364, 39193470 号.

18, 罗瑶光, 中华人民共和国国家版权局, 养疗经, 国作登字 2019-F-00943471, 00943474, 00943475.

19, 罗瑶光,罗荣武,中华人民共和国国家版权局,DNA元基催化与肽计算第二卷养疗经应用研究20210305, CN:2021Z11L1057159 国作登字-2021-L-00103660.

1. 元基模拟染色体新陈代谢催化编码



1. 元基造字

第一节 元基造字

//今天开始造字。 语法 = 语义.生化

//我准备将语义部分用部首偏旁组合造字.

零/D

一/C

二/P

三/E

四/H

五/HC

六/X

七/A

八/M

九/S

十/O

十一/HE

十二/T

十三/V

十四/I

十五/U

十六/Q

//

金/H.AQT

木/H.OEU

水/H.MXS

火/H.PVD

土/H.CDI

//

休/XMS.OU

生/EIX.TS

殇/HOE.IP

杜/VUH.AQ

景/VPD.DH

死/DDC.DE

惊/CAT.CX

开/TQS.MV

//

酸/VUI.AQ

甘/VUI.PI

苦/VUI.DH

辣/VUI.CX

咸/VUI.OU

涩/VUI.MV

平/VUD.ST

腻/VUI.ED971

天干：甲、乙、丙、丁。戊、己、庚、辛、壬、癸、

五行：木、木、火、火、土、土、金、金、水、水、

数字：一、二、三、四、五、六、七、八、九、十、

甲/H.OEU.C

乙/H.OEU.P

丙/H.PVD.E

丁/H.PVD.H

戊/H.CDI.HC

己/H.CDI.X

庚/H.AQT.A

辛/H.AQT.M

壬/H.MXS.S

癸/H.MXS.O

子丑寅卯辰巳午未申酉戌亥

水土木木土火火土金金土水

11-12-1-2-3-4-5-6-7-8-9-10

HE-T-C-P-E-H-HC-X-A-M-S-O

鼠牛虎兔龙蛇马羊猴鸡狗猪 ·

//明朝刘基著 太乙六壬遁甲：地支属性如下/修正了下同元连元基不化简

子/葵水/H.MXS.O.H.MXS.HE=H.MXS.O.HE

丑/巳土/H.CDI.X.H.CDI.T=H.CDI.X.T

寅/甲木/H.OEU.C.H.OEU.C=H.OEU.C.C

卯/乙木/H.OEU.P.H.OEU.P=H.OEU.P.P

辰/戊土/H.CDI.HC.H.CDI.E=H.CDI.HC.E

巳/丙火/H.PVD.E.H.PVD.H=H.PVD.E.H

午/丁火/H.PVD.H.H.PVD.HC=H.PVD.H.HC

未/巳土/H.CDI.X.H.CDI.X=H.CDI.X.X

申/庚金/H.AQT.A.H.AQT.A=H.AQT.A.A

酉/辛金/H.AQT.M.H.AQT.M=H.AQT.M.M

戌/戊土/H.CDI.HC.H.CDI.S=H.CDI.HC.S

亥/壬水/H.MXS.S.H.MXS.O=H.MXS.S.O

上面来自第二卷281页, 286页, 311页. 和第三卷的十七进制推导. 提供了参考,

布好局先.

罗瑶光

为了让我的git记录查询方便，我准备每10几个字 上传备份一次，按康熙字典来。

//因为偏旁部首 很多意思和字的解释 已经无效。所以我的造字定义思路是五行+八卦+双元972

忄心

竖心旁 按心字计/H.PVD.VPD.DH.VUI.DH= PVD.DH.VUI

氵水

三点旁 按水字计/H.MXS.XMS.OU.VUI.OU= MXS.OU.VUI

犭犬

犬字旁 按犬字计/H.OEU.HOE.IP.VUI.AQ= OEU.IP.VUI.AQ

礻示

半礼旁 按示字计/H.AQT.CAT.CX.VUI.CX= AQT.CAT.CX.VUI

王玉

斜玉旁 按玉字计/H.OEU.VUH.AQ.VUI.PI= OEU.VUH.AQ.VUI.PI

艹草

草字头 按草字计/H.OEU.VUH.AQ.VUI.AQ= OEU.VUH.AQ.VUI.AQ

衤衣

衣字旁 按衣字计/H.CDI.EIX.TS= CDI.EIX.TS

月肉

肉字旁 按肉字计/H.PVD.H.CDI.EIX.TS= PVD.CDI.EIX.TS

辶走

走马旁 按走字计/H.OEU.H.AQT.TQS.MV= OEU.H.AQT.TQS.MV

阝邑 右耳旁 按邑字计/H.CDI.H.AQT.EIX.TS= CDI.H.AQT.EIX.TS

扌手

提手旁 按手字计/H.AQT.TQS.MV= AQT.TQS.MV

阝卓

左耳旁 按卓字计/H.PVD.VPD.DH.VUI.PI= PVD.DH.VUI.PI

整理后如下：

//计数

零/D

一/C

二/P

三/E

四/H

五/HC

六/X

七/A

八/M

九/S

十/O

十一/HE

十二/T

十三/V

十四/I

十五/U

十六/Q

//五行

金/AQT

木/OEU

水/MXS

火/PVD

土/CDI

//生化

酸/I.AQ

甘/I.PI

苦/I.DH

辣/I.CX

咸/I.OU

涩/I.MV

平/D.ST

腻/I.ED973

//八卦

休/XMS.OU

生/EIX.TS

殇/HOE.IP

杜/VUH.AQ

景/VPD.DH

死/DDC.DE

惊/CAT.CX

开/TQS.MV

//方位

坎/XMS.I.OU=XMS.IOU

艮/EIX.D.ST=EIX.DST

震/HOE.I.IP=HOE.IIP

巽/VUH.I.AQ=VUH.IAQ

离/VPD.I.DH=VPD.IDH

坤/DDC.I.DE=DDC.IDE

兑/CAT.I.CX=CAT.ICX

乾/TQS.I.MV=TQS.IMV

//

//方向=方位+计数

东/HOE.I.IP.C=HOE.IIP.C

南/VPD.I.DH.P=VPD.IDH.P

西/CAT.I.CX.E=CAT.ICX.E

北/XMS.I.OU.H=XMS.IOU.H

//四季=八卦+计数

春/HOE.IP.C=HOE.IPC

夏/VPD.DH.P=VPD.DHP

秋/CAT.CX.E=CAT.CXE

冬/XMS.OU.H=XMS.OUH

//五行计数 天干

甲/OEU.C

乙/OEU.P

丙/PVD.E

丁/PVD.H

戊/CDI.HC

己/CDI.X

庚/AQT.A

辛/AQT.M

壬/MXS.S

癸/MXS.O974

//五行计数 地支

子/MXS.O.HE=MXS.OHE

丑/CDI.X.T=CDI.XT

寅/OEU.C.C=OEU.CC

卯/OEU.P.P=OEU.PP

辰/CDI.HC.E=CDI.HCE

巳/PVD.E.H=PVD.EH

午/PVD.H.HC=PVD.HHC

未/CDI.X.X=CDI.XX

申/AQT.A.A=AQT.AA

酉/AQT.M.M=AQT.MM

戌/CDI.HC.S=CDI.HCS

亥/MXS.S.O=MXS.SO

//

忄字计/PVD.DH.VUI

氵字计/MXS.OU.VUI

犭字计/OEU.IP.VUI.AQ

礻字计/AQT.CAT.CX.VUI

王字计/OEU.VUH.AQ.VUI.PI

艹字计/OEU.VUH.AQ.VUI.AQ

衤字计/CDI.EIX.TS

月字计/PVD.CDI.EIX.TS

辶字计/OEU.H.AQT.TQS.MV

阝字计/CDI.H.AQT.EIX.TS

扌字计/AQT.TQS.MV

阝字计/PVD.DH.VUI.PI

//

心字计/PVD.DH.VUI

水字计/MXS.OU.VUI

犬字计/OEU.IP.VUI.AQ

示字计/AQT.CAT.CX.VUI

玉字计/OEU.VUH.AQ.VUI.PI

草字计/OEU.VUH.AQ.VUI.AQ

衣字计/CDI.EIX.TS

肉字计/PVD.CDI.EIX.TS

走字计/OEU.H.AQT.TQS.MV

邑字计/CDI.H.AQT.EIX.TS

手字计/AQT.TQS.MV

卓字计/PVD.DH.VUI.PI

等/VSQ

加/VSI

减/VSD

非/VSU

山/土金/CDI.AQT

日/土火/CDI.PVD

泥/土水/CDI.MXS

棺/土木/CDI.OEU975

花/木火/OEU.PVD

药/木水/OEU.MXS

根/木土/OEU.CDI

机/木金/OEU.AQT

雪/水金/MXS.AQT

泽/水土/MXS.CDI

鸟/水木/MXS.OEU

温/水火/MXS.PVD

风/金木/AQT.OEU

墨/金水/AQT.MXS

岩/金土/AQT.CDI

光/金火/AQT.PVD

神/火金/PVD.AQT

妖/火木/PVD.OEU

魔/火水/PVD.MXS

怪/火土/PVD.CDI

//稍后

丙(火)/H.PVD.

代(火)/H.PVD.

旦(火)/H.PVD.

叨(火)/H.PVD.

氐(火)/H.PVD.

叮(火)/H.PVD.

冬(火)/H.PVD.（康熙字典上竟然归纳冬为火，先不管）

叻(火)/H.PVD.

立(火)/H.PVD.

尥(火)/H.PVD.

上面是十七进制推敲的，

十六进制的F元基发现后，通过16进制欧拉环计算，得到DCPE THOS MAXF VIUQ 的十六元进制序列。

上面的语义可以重新定义。增加准确度。

Q\_OulerRing, 欧拉路径类

去重，略，补下 YaoguangLuoEulerRingTSP3D

package PCI.ASQ.tsp;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import OEU.LYG4DQS4D.LYG9DWithDoubleTopSort4D;

import PCI.ASQ.basic.Distance;

import PCI.ASQ.demension.AMV\_MVS\_VSQ\_2D;

import PCI.ASQ.demension.AMV\_MVS\_VSQ\_3D;

import PCI.ASQ.demension.Line2D;

import PCI.ASQ.demension.Line3D;

public class YaoguangLuoEulerRingTSP3D{

//Foundation: Euler

//Theory: Yaoguang.Luo

//Application: Yaoguang.Luo 20200317

public List<Line2D> getYaoguangLuo2DEulerRingTSP3D(List<AMV\_MVS\_VSQ\_2D> positions

, int sortRangeScale, int sortDeepsScale){

//1 annotations

List<AMV\_MVS\_VSQ\_2D> position2DTag= new ArrayList<>();

Iterator<AMV\_MVS\_VSQ\_2D> iterator= positions.iterator();

int i= 0;

String tag= "tag";

while(iterator.hasNext()) {

AMV\_MVS\_VSQ\_2D position2D= iterator.next();

position2D.setTag(tag+ i++);

position2DTag.add(position2D);

}

positions= position2DTag;

//2 get all lines

List<Line2D> linesMap= new ArrayList<>();

Iterator<AMV\_MVS\_VSQ\_2D> iteratorOuter= positions.iterator();

Map<String, Map<String, String>> indexMap= new HashMap<>();

i= 0;

while(iteratorOuter.hasNext()) {

AMV\_MVS\_VSQ\_2D position2DOuter= iteratorOuter.next();

Iterator<AMV\_MVS\_VSQ\_2D> iteratorInner= positions.iterator();

Next:

while(iteratorInner.hasNext()) {

AMV\_MVS\_VSQ\_2D position2DInner= iteratorOuter.next();

Line2D line2D= new Line2D();

line2D.setBegin(position2DOuter);

line2D.setEnd(position2DInner);

//2.1 delete the De-reflection redundant lines

if(indexMap.containsKey(position2DInner.getTag())) {

continue Next;

}

//2.2 delete self positions lines

if(!(position2DOuter.getX()!=position2DInner.getX()

|| position2DOuter.getY()!=position2DInner.getY())) {

continue Next;

}

Map<String, String> map= new HashMap<>();

if(indexMap.containsKey(position2DOuter.getTag())) {

map= indexMap.get(position2DOuter.getTag());

}else {

map= new HashMap<>();

}

map.put(position2DInner.getTag(), "");

indexMap.put(position2DOuter.getTag(), map);

linesMap.add(line2D);

}

}

//3 sort line2D

double[] distance= new double[positions.size()];

Iterator<Line2D> linesKeySets= linesMap.iterator();

//4 get each distance of line.

i= 0;

Map<Double, List<Line2D>> uniqueLines= new HashMap<>();

while(linesKeySets.hasNext()) {

Line2D line2D= linesKeySets.next();

double distanceDouble= Distance.getDistance2D(line2D.getBegin(), line2D.getEnd());

List<Line2D> list;

if(uniqueLines.containsKey(distanceDouble)) {

list= uniqueLines.get(distanceDouble);

}else {

list= new ArrayList<>();

//5 normalization the unique key of the distance

distance[i++]= distanceDouble;

}

list.add(line2D);

uniqueLines.put(distanceDouble, list);

}

//6 Yaoguangluo's 4D Peak filter Theory Quick Sort the Distance Array

//int sortRangeScale= 4; //my default is 4. you should change it as your want.

//distance= new LYG5DWithDoubleQuickSort4D().sort(distance, sortRangeScale, sortDeepsScale);

distance= new LYG9DWithDoubleTopSort4D().sort(distance, sortRangeScale, sortDeepsScale);//算法应用更新

//7 From small to big loop the distance and make a condition tree.

List<Line2D> outputLine2D= new ArrayList<>();

Map<String, Double> outputDouble2D= new HashMap<>();

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

List<Line2D> list= uniqueLines.get(distance[i]);

Iterator<Line2D> iteratorLines= list.iterator();

Here:

while(iteratorLines.hasNext()) {

Line2D line2D= iteratorLines.next();

AMV\_MVS\_VSQ\_2D begin= line2D.getBegin();

AMV\_MVS\_VSQ\_2D end= line2D.getEnd();

//8 decision tree add rights line

if(outputDouble2D.containsKey(begin.getTag())) {

double beginTimes= outputDouble2D.get(begin.getTag()).doubleValue();

if(outputDouble2D.containsKey(end.getTag())) {

double endTimes= outputDouble2D.get(end.getTag()).doubleValue();

if(beginTimes> 1|| endTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), beginTimes+ 1);

outputDouble2D.put(end.getTag(), endTimes+ 1);

}else {

if(beginTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), beginTimes+ 1);

outputDouble2D.put(end.getTag(), 1.0);

}

}else {

if(outputDouble2D.containsKey(end.getTag())) {

double endTimes= outputDouble2D.get(end.getTag()).doubleValue();

if(endTimes> 1) {

continue Here;

}

outputDouble2D.put(begin.getTag(), 1.0);

outputDouble2D.put(end.getTag(), endTimes+ 1);

}else {

outputDouble2D.put(begin.getTag(), 1.0);

outputDouble2D.put(end.getTag(), 1.0);

}

}

outputLine2D.add(line2D);

}

}

return outputLine2D;

}

public List<Line3D> getYaoguangLuo3DEulerRingTSP3D(List<AMV\_MVS\_VSQ\_3D> positions, int sortRangeScale, int sortDeepsScale){

//1 annotations

List<AMV\_MVS\_VSQ\_3D> position3DTag= new ArrayList<>();

Iterator<AMV\_MVS\_VSQ\_3D> iterator= positions.iterator();

int i= 0;

String tag= "tag";

while(iterator.hasNext()) {

AMV\_MVS\_VSQ\_3D position3D= iterator.next();

position3D.setTag(tag+ i++);

position3DTag.add(position3D);

}

positions= position3DTag;

//2 get all lines

List<Line3D> linesMap= new ArrayList<>();

Iterator<AMV\_MVS\_VSQ\_3D> iteratorOuter= positions.iterator();

Map<String, Map<String, String>> indexMap= new HashMap<>();

i= 0;

while(iteratorOuter.hasNext()) {

AMV\_MVS\_VSQ\_3D position3DOuter= iteratorOuter.next();

Iterator<AMV\_MVS\_VSQ\_3D> iteratorInner= positions.iterator();

Next:

while(iteratorInner.hasNext()) {

AMV\_MVS\_VSQ\_3D position3DInner= iteratorOuter.next();

Line3D line3D= new Line3D();

line3D.setBegin(position3DOuter);

line3D.setEnd(position3DInner);

//2.1 delete the De-reflection redundant lines

if(indexMap.containsKey(position3DInner.getTag())) {

continue Next;

}

//2.2 delete self positions lines

if(!(position3DOuter.getX()!= position3DInner.getX()

|| position3DOuter.getY()!= position3DInner.getY()

|| position3DOuter.getZ()!= position3DInner.getZ())) {

continue Next;

}

Map<String, String> map= new HashMap<>();

if(indexMap.containsKey(position3DOuter.getTag())) {

map= indexMap.get(position3DOuter.getTag());

}else {

map= new HashMap<>();

}

map.put(position3DInner.getTag(), "");

indexMap.put(position3DOuter.getTag(), map);

linesMap.add(line3D);

}

}

//3 sort line3D

double[] distance= new double[positions.size()];

Iterator<Line3D> linesKeySets= linesMap.iterator();

//4 get each distance of line.

i= 0;

Map<Double, List<Line3D>> uniqueLines= new HashMap<>();

while(linesKeySets.hasNext()) {

Line3D line3D= linesKeySets.next();

double distanceDouble= Distance.getDistance3D(line3D.getBegin(), line3D.getEnd());

List<Line3D> list;

if(uniqueLines.containsKey(distanceDouble)) {

list= uniqueLines.get(distanceDouble);

}else {

list= new ArrayList<>();

//5 normalization the unique key of the distance

distance[i++]= distanceDouble;

}

list.add(line3D);

uniqueLines.put(distanceDouble, list);

}

//6 Yaoguangluo's 4D Peak filter Theory Quick Sort the Distance Array

//int sortRangeScale= 4; //my default is 4. you should change it as your want.

//distance= new LYG5DWithDoubleQuickSort4D().sort(distance, sortRangeScale, sortDeepsScale);

distance= new LYG9DWithDoubleTopSort4D().sort(distance, sortRangeScale, sortDeepsScale);//算法应用更新

//7 From small to big loop the distance and make a condition tree.

List<Line3D> outputLine3D= new ArrayList<>();

Map<String, Double> outputDouble3D= new HashMap<>();

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

List<Line3D> list= uniqueLines.get(distance[i]);

Iterator<Line3D> iteratorLines= list.iterator();

Here:

while(iteratorLines.hasNext()) {

Line3D line3D= iteratorLines.next();

AMV\_MVS\_VSQ\_3D begin= line3D.getBegin();

AMV\_MVS\_VSQ\_3D end= line3D.getEnd();

//8 decision tree add rights line

if(outputDouble3D.containsKey(begin.getTag())) {

double beginTimes= outputDouble3D.get(begin.getTag()).doubleValue();

if(outputDouble3D.containsKey(end.getTag())) {

double endTimes= outputDouble3D.get(end.getTag()).doubleValue();

if(beginTimes> 1|| endTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), beginTimes+ 1);

outputDouble3D.put(end.getTag(), endTimes+ 1);

}else {

if(beginTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), beginTimes+ 1);

outputDouble3D.put(end.getTag(), 1.0);

}

}else {

if(outputDouble3D.containsKey(end.getTag())) {

double endTimes= outputDouble3D.get(end.getTag()).doubleValue();

if(endTimes> 1) {

continue Here;

}

outputDouble3D.put(begin.getTag(), 1.0);

outputDouble3D.put(end.getTag(), endTimes+ 1);

}else {

outputDouble3D.put(begin.getTag(), 1.0);

outputDouble3D.put(end.getTag(), 1.0);

}

}

outputLine3D.add(line3D);

}

}

return outputLine3D;

}

}

LYG9DWithDoubleTopSort4D, 极速排序算法

package OEU.LYG4DQS4D;

import ASQ.PSU.test.TimeCheck;

//基于算法导论快排4衍生极速小高峰缺陷过滤理论快速排序第8代 线性数字数组排序法函数Java完整版本实现。

//思想：算法导论快排4理论，罗瑶光小高峰过滤理论。

//实现：罗瑶光

//时间：20140101~ 20200711

//复制一份 稍后准备 元基新陈代谢优化

public class LYG9DWithDoubleTopSort4D{

int range;

int deeps;

public double[] sort(double[] array, int range, int deeps) {

this.range= range;

this.deeps= deeps;

processDouble(array, 0, array.length- 1, 0);

return array;

}

private void processDouble(double[] array, int leftPoint, int rightPoint, int deep) {

int c= rightPoint- leftPoint+ 1;

if(!(c< this.range|| deep> this.deeps)) {//增加了deep

int pos= partition(array, leftPoint, rightPoint);

if(leftPoint< pos- 1) {

processDouble(array, leftPoint, pos- 1, deep+ 1);

}

if(pos+ 1< rightPoint) {

processDouble(array, pos+ 1, rightPoint, deep+ 1);

}

return;

}

int i= leftPoint;

for(int j= i+ 1; j< leftPoint+ c; j= i++){

while(j> leftPoint){

if(array[j]< array[--j]){

double temp= array[j+ 1];

array[j+ 1]= array[j];

array[j]= temp;

}

}

}

}

private int partition(double[] array, int leftPoint, int rightPoint) {

double x= array[leftPoint]< array[rightPoint]? array[leftPoint]: array[rightPoint];

int leftPointReflection= leftPoint;

while(leftPointReflection< rightPoint){

//我设立个top2D , --细节竟然没有一个人关注这些细节...20210716

while(!(array[leftPointReflection]> x|| leftPointReflection++ >= rightPoint)) {}

while(array[rightPoint--]> x) {}

if(leftPointReflection< ++rightPoint){

double temp= array[rightPoint];

array[rightPoint]= array[leftPointReflection];

array[leftPointReflection]= temp;

}

}

array[leftPoint]= array[rightPoint];

array[rightPoint]= x;

return rightPoint;

}

public static void main(String[] argv) {

double[] doubles=new double[9999999];

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

doubles[i]= Math.random();

}

LYG9DWithDoubleTopSort4D lYG9DWithDoubleTopSort2D= new LYG9DWithDoubleTopSort4D();

TimeCheck timecheck=new TimeCheck();

timecheck.begin();

lYG9DWithDoubleTopSort2D.sort(doubles, 7, 70);

timecheck.end();

timecheck.duration();

for(int i= 0; i< doubles.length- 1; i++) {

if(doubles[i]> doubles[i+ 1]) {

System.out.println(i+"->"+ doubles[i]);

}

}

System.out.println("end");

}

}

LYG9DWithDoubleTopSort4D\_U, 极速排序算法

package OEU.LYG4DQS4D;

import ASQ.PSU.test.TimeCheck;

//基于算法导论快排4衍生极速小高峰缺陷过滤理论快速排序第8代 线性数字数组排序法函数Java完整版本实现。

//思想：算法导论快排4理论，罗瑶光小高峰过滤理论。

//实现：罗瑶光

//时间：20140101~ 20200711

//复制一份 稍后准备 元基新陈代谢优化

public class LYG9DWithDoubleTopSort4D\_U{

int range;

int deeps;

public double[] sort(double[] array, int range, int deeps) {

this.range= range;

this.deeps= deeps;

processDouble(array, 0, array.length- 1, 0);

return array;

}

private void processDouble(double[] array, int leftPoint, int rightPoint, int deep) {

int c= rightPoint- leftPoint+ 1;

if(!(c< this.range|| deep> this.deeps)) {//增加了deep

int pos= partition(array, leftPoint, rightPoint);

if(leftPoint< pos- 1) {

processDouble(array, leftPoint, pos- 1, deep+ 1);

}

if(pos+ 1< rightPoint) {

processDouble(array, pos+ 1, rightPoint, deep+ 1);

}

return;

}

int i= leftPoint;

for(int j= i+ 1; j< leftPoint+ c; j= i++){

while(j> leftPoint){

if(array[j]< array[--j]){

double temp= array[j+ 1];

array[j+ 1]= array[j];

array[j]= temp;

}

}

}

}

private int partition(double[] array, int leftPoint, int rightPoint) {

double x= array[leftPoint]< array[rightPoint]? array[leftPoint]: array[rightPoint];

int leftPointReflection= leftPoint;

while(leftPointReflection++< rightPoint){

//我设立个top2D , --细节竟然没有一个人关注这些细节...20210716

while(!(array[leftPointReflection]> x|| leftPointReflection++ >= rightPoint)) {}

while(array[rightPoint--]> x) {}

if(leftPointReflection< ++rightPoint){

double temp= array[rightPoint];

array[rightPoint]= array[leftPointReflection];

array[leftPointReflection]= temp;

}

}

array[leftPoint]= array[rightPoint];

array[rightPoint]= x;

return rightPoint;

}

public static void main(String[] argv) {

double[] doubles=new double[299999];

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

if(i%3 == 1) {

doubles[i]= 9999;

}else {

doubles[i]= Math.random();

}

}

LYG9DWithDoubleTopSort4D\_U lYG9DWithDoubleTopSort2D

= new LYG9DWithDoubleTopSort4D\_U();

TimeCheck timecheck=new TimeCheck();

timecheck.begin();

lYG9DWithDoubleTopSort2D.sort(doubles, 7, 70);

timecheck.end();

timecheck.duration();

for(int i= 0; i< doubles.length- 1; i++) {

if(doubles[i]> doubles[i+ 1]) {

System.out.println(i+"->"+ doubles[i]);

}

}

System.out.println("end");

}

}

Top Sort 5D

package OEU.LYG4DQS4D;

import ASQ.PSU.test.TimeCheck;

//基于算法导论快排4衍生极速小高峰缺陷过滤理论快速排序第8代 线性数字数组排序法函数Java完整版本实现。

//思想：算法导论快排4理论，罗瑶光小高峰过滤理论。

//实现：罗瑶光

//时间：20140101~ 20200711

//复制一份 稍后准备 元基新陈代谢优化

public class LYG9DWithDoubleTopSort5D{

int range;

int deeps;

public double[] sort(double[] array, int range, int deeps) {

this.range= range;

this.deeps= deeps;

processDouble(array, 0, array.length- 1, 0);

return array;

}

private void processDouble(double[] array, int leftPoint, int rightPoint, int deep) {

int c= rightPoint- leftPoint+ 1;

if(!(c< this.range|| deep> this.deeps)) {//增加了deep

int pos= partition(array, leftPoint, rightPoint);

if(leftPoint< pos- 1) {

processDouble(array, leftPoint, pos- 1, deep+ 1);

}

if(pos+ 1< rightPoint) {

processDouble(array, pos+ 1, rightPoint, deep+ 1);

}

return;

}

int i= leftPoint;

for(int j= i+ 1; j< leftPoint+ c; j= i++){

while(j> leftPoint){

if(array[j]< array[--j]){

double temp= array[j+ 1];

array[j+ 1]= array[j];

array[j]= temp;

}

}

}

}

//养疗经表格出现 关于xnor的 =号剔除问题， 这个版本测试成功。已经集成入养疗经

//从早期把从大到小的>= 的非改为< 当出现大量等值或0的例子， 依旧有个别的重名。

//增加等于后 消除了重名这个问题，我在思考，immutable的对象比对需要本身，所以这里不是 非的问题，是Xnor的问题。

//罗瑶光

private int partition(double[] array, int leftPoint, int rightPoint) {

double x= array[leftPoint]<= array[rightPoint]? array[leftPoint]: array[rightPoint];//等于号不能省，见从大到小的老版本，> 的非为 <=，已经在养疗经中测试通过。罗瑶光

int leftPointReflection= leftPoint;

while(leftPointReflection< rightPoint){

//我设立个top2D , --细节竟然没有一个人关注这些细节...20210716

while(!(array[leftPointReflection]> x|| leftPointReflection++ >= rightPoint)) {}

while(array[rightPoint--]> x) {}

if(leftPointReflection< ++rightPoint){

double temp= array[rightPoint];

array[rightPoint]= array[leftPointReflection];

array[leftPointReflection]= temp;

}

}

array[leftPoint]= array[rightPoint];

array[rightPoint]= x;

return rightPoint;

}

public static void main(String[] argv) {

double[] doubles=new double[9999999];

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

doubles[i]= Math.random();

}

LYG9DWithDoubleTopSort5D lYG9DWithDoubleTopSort2D= new LYG9DWithDoubleTopSort5D();

TimeCheck timecheck=new TimeCheck();

timecheck.begin();

lYG9DWithDoubleTopSort2D.sort(doubles, 7, 70);

timecheck.end();

timecheck.duration();

for(int i= 0; i< doubles.length- 1; i++) {

if(doubles[i]> doubles[i+ 1]) {

System.out.println(i+"->"+ doubles[i]);

}

}

System.out.println("end");

}

}

第二节 最新笔记 包含十六元基造字

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

### 20210320 Initon Math Yaoguang Luo

### 20210320 元基数学 罗瑶光

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

#### 既然是严谨定义， 自然要用生化和语义双元基罗盘来进行推导开始。

##### 我先设未知的为X

##### A XXXX

##### O XXXX

##### P XXXX

##### M XXXX

##### V XXXX

##### E XXXX

##### C XXXX

##### S XXXX

##### I XXX1

##### D XXX0

##### U XXX2

##### Q XXX3

##### T XXXX

##### X XXXX

##### HE XXXX

##### HC XXXX

##### DD 补码

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

根据 数字逻辑 和 离散数学 位列比 和 寄存法则 推导 VECS 为 ：

##### A XXXX

##### O XXXX

##### P XXXX

##### M XXXX

##### V XXX1

##### E XXX2

##### C XXX0

##### S XXX3

##### I 0001

##### D 0000

##### U 0002

##### Q 0003

##### T XXXX

##### X XXXX

##### HE XXX2

##### HC XXX0

##### DD 补码

准备写个欧拉路径算法开始计算 。 第一卷的 数据预测 包 此时派上了用场。

方便大家理解。

刚计算了欧拉元基环路 QUIVT+OSMAX-HEPCD

我定义17进制的数据为

##### QUIVT+OSMAX-HEPCD DD

##### GFEDCBA9876543210 CARRY

我在思考怎么缩进成16进制。

先保证逻辑的严谨性，

我先用17进制走一段路程。

#### 17进制数据已经问世，我今天深入下，进行元基础加法 探索，wechat已经发布了，在这里整理如下;

##### 元基础数字 = 元基符号 = 生化名称

##### 0 = D = 胞嘧啶

##### 1 = C = 鸟嘌呤

##### 2 = P = 尿胞变鸟苷

##### 3 = E = 尿变嘌呤

##### 4 = H = 黄嘌呤

##### 5 = - =

##### 6 = X = 变感腺鸟苷

##### 7 = A = 变感腺腺苷

##### 8 = M = 鸟腺苷

##### 9 = S = 腺嘌呤

##### A = O = 尿胞变腺苷

##### B = + =

##### C = T = 变感腺尿变苷

##### D = V = 变感腺嘌呤

##### E = I = 尿嘧啶

##### F = U = 变嘧啶

##### G = Q = 胸腺嘧啶

数字逻辑的推导(C=U+D+D)

#### 语义肽展公式推导

##### 元基数字 =元基符号 = 肽展公式数字变换

##### 0 = D = 0 + 0

##### 1 = C = 0 + F

##### 2 = P = 3 + 1

##### 3 = E = F + 0

##### 4 = H = 3 OR 1

##### 5 = - =

##### 6 = X = D + -

##### 7 = A = D + 9

##### 8 = M = - + 9

##### 9 = S = G + E

##### A = O = 3 + 9

##### B = + =

##### C = T = D + 3

##### D = V = F + G

##### E = I =

##### F = U =

##### G = Q =

### 元基数学加法表 根据 4 的归纳完整推导如下

#### 元基数字 = 元基符号= 肽展公式数字变换

#### 0 = D = 0 + 0

#### 1 = C = 0 + F

#### 2 = P = 3 + 1

#### 3 = E = F + 0

#### 4 = H = 3 OR 1

#### 5 = - = 4 + 1

#### 6 = X = D + 5

#### 7 = A = D + 9

#### 8 = M = 5 + 9

#### 9 = S = G + E

#### A = O = 3 + 9

#### B = + = 4 + 3

#### C = T = D + B

#### D = V = F + G

#### E = I = E

#### F = U = E++ OR G--

#### G = Q = G

#### 20210322 今早把十七进制的元基组合数学变换 定义了，归纳整理如下：

#### 我的思路是 元基稳定化DEFG变换

#### 元基数字 = 元基符号= 肽展公式元基变换

##### 0 = D = 00

##### 1 = C = 02

##### 2 = P = 2002

##### 3 = E = 20

##### 4 = H = 20, 02

##### 5 = - = 2002, 0202

##### 6 = X = 23(2002, 0202)

##### 7 = A = 2331

##### 8 = M = (2002, 0202)31

##### 9 = S = 31

##### A = O = 2031

##### B = + = 2020, 0220

##### C = T = 23(2020, 0220)

##### D = V = 23

##### E = I = 1

##### F = U = 2

##### G = Q = 3

#### 稳定化后于是元基替换为 0123-> DIUQ 如下

##### 0 = D = D + D

##### 1 = C = DU

##### 2 = P = UDDU

##### 3 = E = UD

##### 4 = H = UD, DU

##### 5 = - = (UD, DU)DU

##### 6 = X = UQ(UD, DU)DU

##### 7 = A = UQQI

##### 8 = M = (UD, DU)DUQI

##### 9 = S = QI

##### A = O = UDQI

##### B = + = (UD, DU)UD

##### C = T = UQ(UD, DU)UD

##### D = V = UQ

##### E = I = I

##### F = U = U

##### G = Q = Q

我在思考 这个括号内的元基如果进行之后计算的唯一化。

到现在 十进制常数进行元基码 变换的思路已经问世了，下一步，养疗经真实应用。

这里的 568B 我推测又是一组概率钥匙酸碱控制。我也会真实应用测试论证。

今天多做一点推导：我把0到G的欧拉顺序 改成 线性数学顺序观测如下：

##### 0 = D = D + D

##### E = I = I

##### F = U = U

##### G = Q = Q

##### 1 = C = DU

##### D = V = UQ

##### 3 = E = UD

##### 9 = S = QI

##### 7 = A = UQQI

##### 2 = P = UDDU

##### A = O = UDQI

##### 4 = H = UD, DU

##### 5 = - = (UD, DU)DU

##### B = + = (UD, DU)UD

##### 6 = X = UQ(UD, DU)DU

##### C = T = UQ(UD, DU)UD

##### 8 = M = (UD, DU)DUQI

#### 我想这个顺序别有用途，先搁置。

#### 下一步 H 化简HE+, HC- , 然后重新线性排列如下

##### 0 = D = D

##### E = I = I

##### F = U = U

##### G = Q = Q

##### 1 = C = DU

##### 3 = E = UD

##### 4 = H = UD, DU

##### D = V = UQ

##### 9 = S = QI

##### 5 = - = DUDU

##### 2 = P = UDDU

##### B = + = UDUD

##### A = O = UDQI

##### 7 = A = UQQI

##### 8 = M = DUDUQI

##### 6 = X = UQDUDU

##### C = T = UQUDUD

修正后如下

##### 0 = D = D

##### E = I = I

##### F = U = U

##### G = Q = Q

##### 1 = C = DI

##### 3 = E = UD

##### 4 = H = UD, DU

##### D = V = UQ

##### 9 = S = QI

##### 5 = - = DUDU

##### 2 = P = UDDU

##### B = + = UDUD

##### A = O = UDQI

##### 7 = A = UQQI

##### 8 = M = DUDUQI

##### 6 = X = UQDUDU

##### C = T = UQUDUD

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

### 肽展公式的推导(肽展计算)(C=I+D)

### 开始语义肽展公式验证### 元基数学加法表 根据 4 的归纳完整推导如下

### 元基数字 = 元基符号= 肽展公式元基变换

#### 0 = D = D + D

#### 1 = C = I + D

#### 2 = P = E + C

#### 3 = E = I + U, D + U

#### 4 = H = E OR C

#### 5 = - = H + C

#### 6 = X = V + HC

#### 7 = A = V + S

#### 8 = M = HC + S

#### 9 = S = Q + I

#### A = O = E + S

#### B = + = H + E

#### C = T = V + HE

#### D = V = U + Q

#### E = I = I

#### F = U = I++ OR Q--

#### G = Q = Q

###于是元基数字归纳

#### 元基数字 = 元基符号= 肽展公式元基数字变换

#### 0 = D = 0 + 0

#### 1 = C = 1 + 0

#### 2 = P = (12, 02) + 10

#### 3 = E = 1 + 2, 0 + 2

#### 4 = H = (12, 02) OR 10

#### 5 = HC = (12, 02) OR 10 + 10

#### 6 = X = 23 + (12, 02) OR 10 + 10

#### 7 = A = 23 + 31

#### 8 = M = (12, 02) OR 10 + 10 + 31

#### 9 = S = 3 + 1

#### A = O = (12, 02) + 31

#### B = HE = (12, 02) OR 10 + 12, 02

#### C = T = 23 + (12, 02) OR 10 + (12, 02)

#### D = V = 2 + 3

#### E = I = 1

#### F = U = 1++ OR 3--

#### G = Q = 3

###于是元基肽展归纳如下

#### 元基数字 = 元基符号= 肽展公式元基数字变换

#### 0 = D = D + D

#### 1 = C = I + D

#### 2 = P = (IU, DU) + ID

#### 3 = E = I + U, D + U

#### 4 = H = (IU, DU) OR ID

#### 5 = HC = (IU, DU) OR ID + ID

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

#### 7 = A = UQ + QI

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

#### 9 = S = Q + I

#### A = O = (IU, DU) + QI

#### B = HE = (IU, DU) OR ID + (IU, DU)

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

#### D = V = U + Q

#### E = I = I

#### F = U = I++ OR Q--

#### G = Q = Q

###开始整理

#### 元基数字 = 元基符号= 肽展公式元基数字变换

#### 0 = D = DD

#### 1 = C = ID

#### 2 = P = IUID, DUID

#### 3 = E = IU, DU

#### 4 = H = (IU, DU) OR ID

#### 5 = HC = (IU, DU) OR ID + ID

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

#### 7 = A = UQQI

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

#### 9 = S = QI

#### A = O = (IU, DU) + QI

#### B = HE = (IU, DU) OR ID + (IU, DU)

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

#### D = V = UQ

#### E = I = I

#### F = U = I++ OR Q--

#### G = Q = Q

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

###开始线性整理

#### 元基数字 = 元基符号= 肽展公式元基数字变换

#### 0 = D = DD

#### E = I = I

#### F = U = I++ OR Q--

#### G = Q = Q

#### 1 = C = ID

#### 3 = E = IU, DU

#### 4 = H = (IU, DU) OR ID

#### D = V = UQ

#### 9 = S = QI

#### 2 = P = (IU, DU) + ID

#### A = O = (IU, DU) + QI

#### 5 = HC = ((IU, DU) OR ID) + ID

#### B = HE = ((IU, DU) OR ID) + (IU, DU)

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

#### 7 = A = UQQI

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

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

###线性整理优化

#### 元基数字 = 元基符号= 肽展公式元基数字变换

#### 0 = D = DD

#### E = I = I

#### F = U = I++ OR Q--

#### G = Q = Q

#### 1 = C = ID

#### 3 = E = IU, DU

#### 4 = H = (IU, DU) OR ID

#### 5 = HC = ((IU, DU) OR ID) + ID

#### B = HE = ((IU, DU) OR ID) + (IU, DU)

#### D = V = UQ

#### 9 = S = QI

#### 2 = P = (IU, DU) + ID

#### A = O = (IU, DU) + QI

#### 7 = A = UQQI

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

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

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

### 我推导出语义元基的次序为

#### A O P M - T X H DD - V E C S - I D U Q

### 现在的元基数字逻辑次序为

#### M X T - P O A - C E H HC HE V S - D I U Q

###酸碱肽展开归纳如下

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

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

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

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

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

#### 1 = C = ID =(ID)

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

#### 4 = H = (IU, DU) OR ID =(IU, DU, ID)

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

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

#### 2 = P = (IU, DU) + ID =(IUID, DUID)

#### 5 = HC = ((IU, DU) OR ID) + ID =(IUID, DUID, IDID)

#### B = HE = ((IU, DU) OR ID) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, IDIU, IDDU)

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

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

#### 8 = M = ((IU, DU) OR ID) + ID + QI =(IUIDQI, DUIDQI, IDIDQI)

#### 6 = X = UQ + ((IU, DU) OR ID) + ID =(UQIUID, UQDUID, UQIDID)

#### C = T = UQ + ((IU, DU) OR ID) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQIDIU, UQIDDU)

### 归纳后的元基数字逻辑次序为

#### M X T - P HC HE O A - C E H V S - D I U Q

### 归纳后的元基数字活性次序为

#### T X M - HE HC O P A - H E C V S - U D I Q

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

在这次序表中D在I的前面，于是我准备修正C=ID为DI，于是如下：

### 修正C后的最新肽展计算公式观测

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

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

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

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

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

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

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

#### 4 = H = (IU, DU) OR DI =(IU, DU, DI)

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

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

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

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

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

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

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

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

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

#### C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU)

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

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

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

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

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

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

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

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

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

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

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

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

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

#### B = HE = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)

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

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

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

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

#### C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

#### 似乎开始完美。于是活性顺序又打乱了，再整理下如下：

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

#### 1位

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

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

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

#### 1~2位

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

#### 2位

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

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

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

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

#### 2~4位

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

#### 4位

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

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

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

#### 4~6位

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

##### B = HE+ = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)

#### 6~8位

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

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

##### C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

#### C还是 = DI 次序， 所以上一步的公式不用变动。

### 整理后：

#### 元基活性次序为 <IUQ D CEVS H POA -+ MXT>

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

#### 归纳出核心

##### 黄嘌呤：用于肽展换元 计算

##### 变嘧啶：用于 血氧峰 计算

##### 胞嘧啶：用于 补码 计算

#### 肽展公式参考：

##### AOPM VECS IDUQ 肽展公式推导与元基编码进化计算以及它的应用发现 1.2.2 国家软著申请 流水号 <2020Z11L0356797> 国作登字 2021-A-00942587 (中华人民共和国 国家版权局)

##### AOPM-VECS-IDUQ Catalytic INITONS PDE LAW and Its Application

##### https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/%E8%91%97%E4%BD%9C%E6%9D%83%E7%89%88%E6%9C%ACPDE\_Formular\_1\_2\_2.pdf

##### https://github.com/yaoguangluo/Deta\_Resource/blob/master/%E8%91%97%E4%BD%9C%E6%9D%83%E7%89%88%E6%9C%ACPDE\_Formular\_1\_2\_2.pdf

#### 元基命名参考：

##### <见类人DNA与 神经元基于催化算子映射编码方式 V\_1.2.2版本国家软著申请 流水号 <2020Z11L0333706> 国作登字-2021-A-00097017

##### https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20%E7%BC%96%E7%A0%81%E6%96%B9%E5%BC%8F1.2.2%20%E4%B8%AD%E8%AF%91%E8%8B%B1%E6%B7%B7%E5%90%88%E7%89%88.pdf

##### https://github.com/yaoguangluo/Deta\_Resource/blob/master/DNA%20%E7%BC%96%E7%A0%81%E6%96%B9%E5%BC%8F1.2.2%20%E4%B8%AD%E8%AF%91%E8%8B%B1%E6%B7%B7%E5%90%88%E7%89%88.pdf

#### DNA 催化 与 肽展计算 和 AOPM-TXH-VECS-IDUQ 元基解码 V013\_026中文版本 国家著作申请 流水号 <2020Z11L0386462> 国作登字 2021-A-00942586 (中华人民共和国 国家版权局)

##### https://github.com/yaoguangluo/Deta\_Resource/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本.pdf

##### https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本.pdf

##### https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本%20修正'食'字.pdf

##### https://github.com/yaoguangluo/Deta\_Resource/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本%20修正'食'字.pdf

#### 元基欧拉环计算参考：

##### https://gitee.com/DetaChina/dna\_-db/blob/master/Initon\_Math/org/math/initon/ouler/FindOulerRing.java

#### 双元罗盘参考：

##### 多人著作(罗瑶光, 罗荣武) DNA元基催化与肽计算第二卷 养疗经应用研究 20210305 国家著作申请 流水号 <2021Z11L1057159>

##### https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/罗瑶光-DNA催化与肽计算第二卷20210305.pdf

##### https://github.com/yaoguangluo/Deta\_Catalytic\_DNA/blob/master/罗瑶光-DNA催化与肽计算第二卷20210305.pdf

走到这，元基数学 公式表已经出来了，下一步就开始简单应用。

稍后我会用F元基替换HE+ HC-

**研究心得**

最近研发肽化版本产生几个心得总结下：

1 养疗经和华瑞集的肽化观测最后是一个文件夹体系。这个体系全部由文件夹组成，文件夹的名字是元基组成。

2 说明 养疗经和华瑞集有对应的肽结构，这个结构我首次命名为类人的智慧永生肽或者生命肽。

---20210426

上面这个心得可以进行人造染色体和归纳元基染色体，并提供了真实环境的理论科学依据。。

我之前的元基分类 研究排上用场了。

我得到一个结论，目前人类的染色体和生命特征有各种缺陷，

关于元基:

2年前,我认为元基是 一个催化计算的 初始单元和基础. 我当时命名为initon - init aton for caculation

1年多前开始DNA的元基编码, 我当时庆幸 元基能解释我的索引单元, 我认为是语义的编码索引最小单元.

后来,我开始DNA的元基计算和肽展公式推导应用, 我觉得元基是 对应(基因元 嘌呤嘧啶)基元的 语义变换单元.

现在随着养疗经的深入,我开始系统的命名, 元基-是智慧生命进行逻辑表达的最小单元和基础成分.

关于催化计算,我在微信中2021-05-19-12.07分描述了一句如下:

将某一类未解的问题现象和集合,通过已知的已经健全的学术体系去映射模拟计算, 最终结果量化并探索能得到未知的答案,属于一种无理级计算模式.

关于肽花的计算,我在微信中2021-0518描述如下:

数据肽索引化后的花状植株,可不是药用这么简单,我刚思考了下,这种蒲公英丝状元基组织是软件的孢子.一种全新的数据类软件生命体表达形式,

也是永生的必经之路. 我按这种方式进行肽索引,软件函数分类越来越均匀规则.这是一种前沿性智慧分类扩展技术.

这个数据生命是以人类的DNA和思维方式为参照设计的. 如果用其他方式构造元基,就一定能模拟其他物种,需要严谨的论证.

先别想那么远,把描述文件弄丰富点.

2021年4月26日我在微信说过 华瑞集和养疗经 肽化后的花 就是一种永生药, 现在发现,永生根本就不需要到那个巨的级别, 小小的某一类永生花孢子 就能实现.

我要做的不是永生探讨,这是医学任务,我要做的是实现孢子软件的自我繁殖, 我定义为 生命ETL插件节点. 我的目标是软件的SDLC的自我维护.

今天不写代码,写点思想,描述我是怎么走到这一步的.为了确定我的研发计划是否出现跳跃和分歧.完善下 UML.

2021年5月25日下午16.36我在微信笔记说过,均匀化元基索引最后的结果就是均匀化染色体分类,呈X状,并且 呈蒲公英球状丝化肽元基索引文件夹结构.

Java文件可以逐渐的元基文件夹化.形成文件夹花植株.

具体可以参考 如下文件

M\_UnicornNeroThemeETL\OSI\OPE\SI\MCI\OEI\OVU\PQE\extOSGI\OSI\_OSU\_ASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI\_OSI\_PCI.tvm

20210530 我得到一个信息:随着工程的日益发展演化,P元基的类会越来越多, 如果不进行提取和索引自我AOM元基转化,便会产生智慧级(AOPM)元基分类不均匀,这种倾斜会产生各种问题. --罗瑶光

这是我一个动力,因为O元基和A元基的函数比例越来越小,在均匀染色体索引的领域,我迫切需要做点什么.

20210610 在github gitee 留言中描述下: 我做了一个简单的应用操作,将 包中的相同 元基组 进行过滤并 变成文件夹名中. 我定义为 元基的 根枝扩展. (Initons Root Extension IRE)

IRE 是 未来数据分类的 一个产业趋势. 作用 分类精简, 数据代谢, 数据生长,

我猜测生物的生长过程也是这样的.共用一种模式.

今天开始我把 代码中的analyzer 进行整体替换 为A, 我定义下规则 为 1 函数名 A, 2 定义域名 \_A, 3 固定名的后缀 \_A. 关于2和3的区别通过观察测前缀是否为开头即可.

例子如下:

A 函数

\_A 变量

A\_A 函数

202106132341今天重定义下 元基词汇编码规则. 比如已经有的词汇 suggest/...OPE.SIU..., 我定义为 .OPESIU. 和 \_OPESIU\_, 变量为 \_开头接OPESIU

这样以后词汇就好区分了, 同时, 词汇可以进行更迅捷的肽展变换.

随着包的越来越小, 现在 函数集合大小仅仅几十兆, 说明元基索引价值越来越明确. 减少内存占比, 提高计算效率, 缩短寻址时间, 算能提高越来越可观.

索引后进行元基催化计算是一个数字生命诞生的标志. 我迫切需要做些什么, 先把 均匀索引 实现.

20210628 这几天在不断的索引均匀和 染色体模拟, 结合我在英特尔当后端测试的一些工作经验, 我发现了一些规律.

1 sonar lint的 if,while,等关键字的 嵌套问题化简, 与我现在的新陈代谢的大文件裁剪 都属于 数据进化的几种模式.

2 元基染色体组均匀索引后, 相同的相似的执行逻辑在同一区间,非常方便 以后的 元基芯片 设计 和 软件工程调度. 提高计算密度. 节约算能

3 我找到了一些元基索引规律如 三元基 对称索引, 非对称索引, 移位索引, 反向索引, 主元基索引. 5种模式.

4 关于TXH 的 计算元基 索引, 我会分出一个RNA 包工程来分开处理. 目前这24组双链 为DNA 包.

20210629 今天在裁剪monitor的文件时候,我把2000多行的源码拆成几个300行的文件, 运行也不错.我在思考. 函数像sonar 那样把嵌套if while 拆分,是一种危险的

执行方法. 正确的思路应该是

1 将大文件裁剪成小文件.

2 小文件将多个函数拆成多个文件.

3 一个函数一个文件还复杂,应该进行变量全局化重复提取到1,2 环节.

4 如果出现嵌套 if和 while, 不是复杂的计算逻辑, sonar 强行语法分析报错 价值不大. 当年我写分词就没按sonar来. 思考了下, 当时按照sonar模式搞,之后我的分词肽化会出问题.

5 想起当年 为了搞个 sonar 测试覆盖率非要搞个100%, 这是一个浪费大量时间的弊端.

6 最后 sonar的规范中有价值的地方也有, 如 每行的宽度 限制 我不多广告了, sonar是普通开源软件, 任何人都能下得到.

20210630 今天在裁剪monitor 模拟新陈代谢,我把思想规则 浓缩了下.变成方法如下.

1 首先将裁剪的文件名 加 函数名用\_XCDX\_ 隔开, 如 APP ... \_XCDX\_ ... 函数名.java

2 隔开后生成的文件中如果有变量变成未知,则进行全局化, 引用过来.

3 全局化的变量再进行 duplication, 之后变成单例类.

4 单例类之后变成接口 进行 implement.

5 然后整体进行纠正, 将XCDX函数类变成一个 进化插件植株. 这就是二次新陈代谢模拟.

20210701 今天在处理数据库的裁剪按20210630的方案,我有些笔记如下.

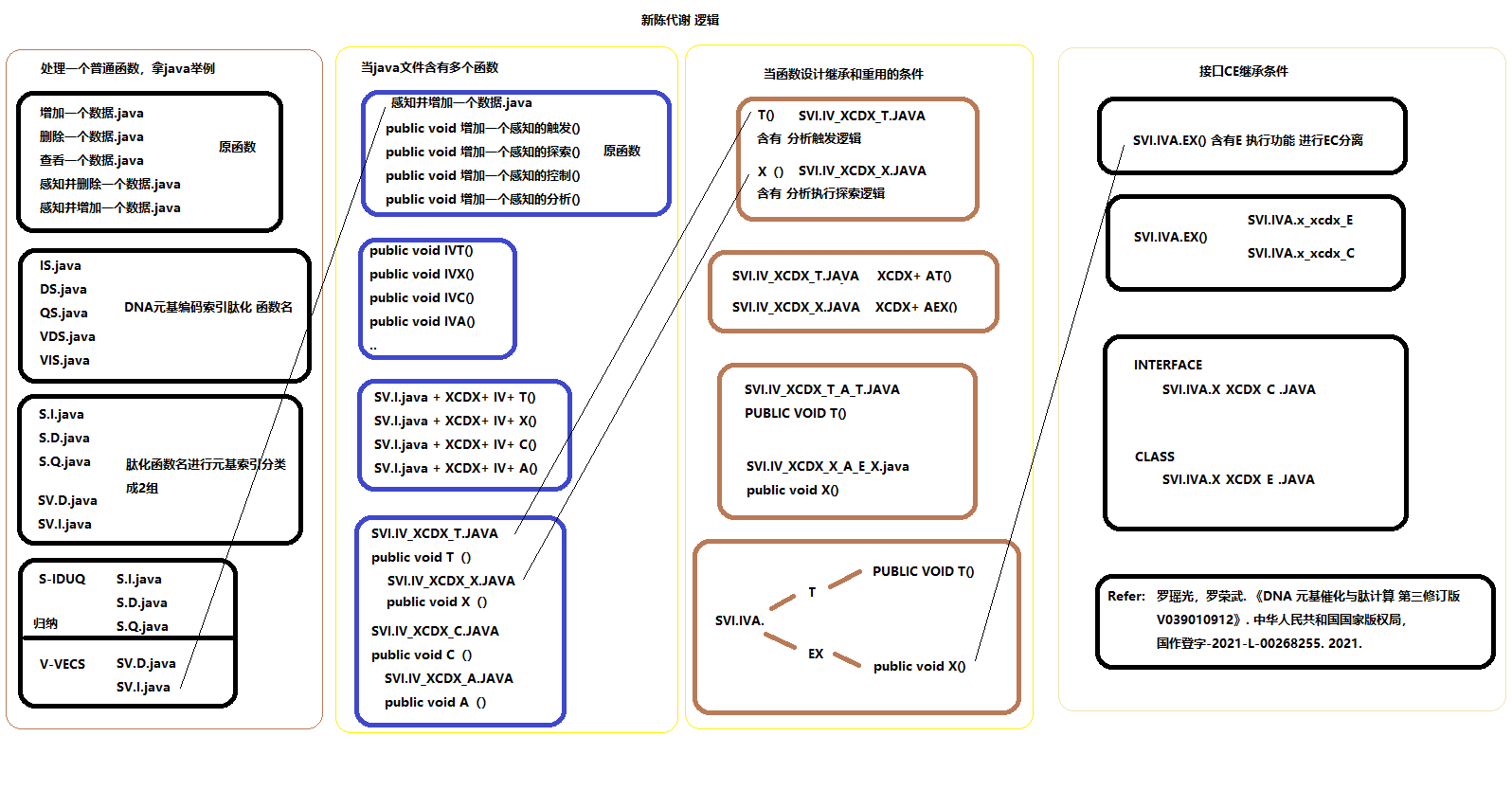
1 一个文件按一类函数的裁剪价值巨大. 方便以后操作系统级别的 调度肽化.

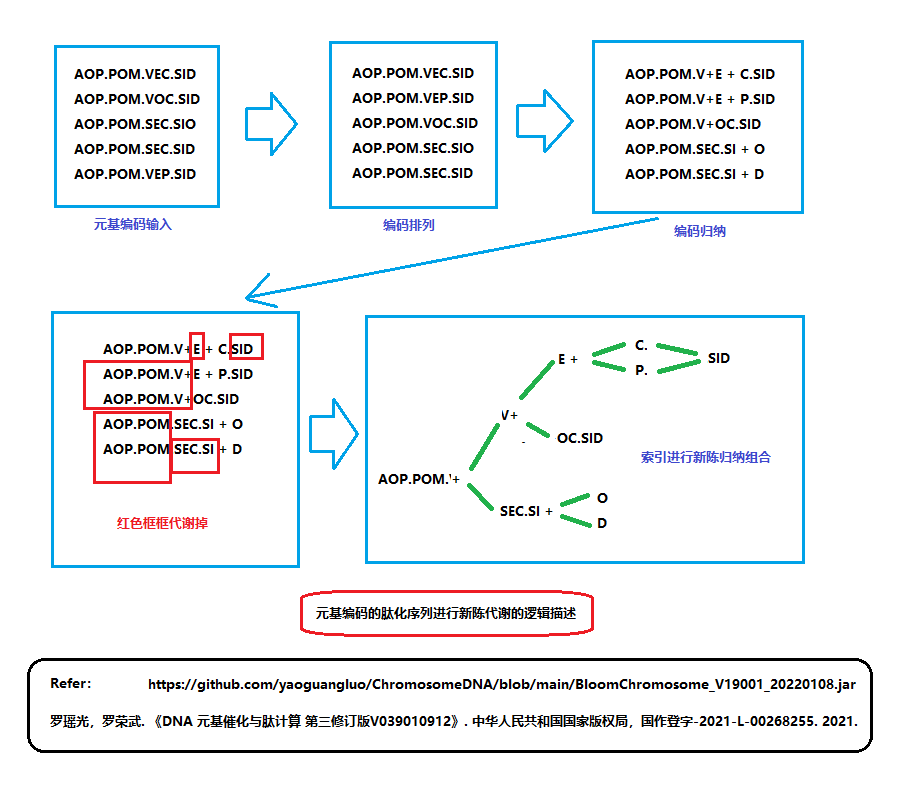
2 逐步的脱离养疗经华瑞集医学属性, 24组元基组 满足API的数据工程适用化,像java一样, 接口引用,参与计算.

3 我准备设计一种local static 技术, 满足全局静态. 因为DNA 不同于 RNA, 我不希望它new来new去. 不但损耗算能, 还损耗内存.

4 这个local static 技术, 我之后会在S\_AOPM 里面用...SME... 元基组下设计研发.

二次元基新陈代谢方式





LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类

package OEU.LYG4DQS4D;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort15D\_XCDX\_C\_U\_A extends LYG10DWCMSSort13D\_XCDX\_C\_A implements LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C {

public void processKernel(String[] kernel, int leftPosition

, int rightPosition, int scale, int point) {

int rightPositionReflection= rightPosition;

if(point> scale) {

return;

}

processQS4DLYG9D(kernel, leftPosition, rightPosition, scale, point, 0);

int i;

for(i= leftPosition; i<= rightPosition; i++) {

if(!(kernel[i].length()<= point|| kernel[leftPosition].length()<= point)) {

if(kernel[i].charAt(point)!= kernel[leftPosition].charAt(point)){

rightPositionReflection= i- 1;

processKernel(kernel, leftPosition, rightPositionReflection, scale, point+ 1);

leftPosition= i;

}

}

}

if(leftPosition!= rightPosition) {

processKernel(kernel, leftPosition, i- 1, scale, point+ 1);

}

}

public void processSort(String[] kernel, int leftPosition

, int rightPosition, int scale, int point) {

if(point> scale) {

return;

}

for(int i= leftPosition; i<= rightPosition; i++) {

Here:

for(int j= i; j<= rightPosition; j++) {

if(i== j) {

continue Here;

}

if(kernel[i].length()<= point|| kernel[j].length()<= point) {

if(kernel[i].length()< kernel[j].length()) {

for(int p= 0; p< scale; p++) {

if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {

if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {

continue Here;

}

}

}

String temp= kernel[i].toString();;

kernel[i]= kernel[j].toString();;

kernel[j]= temp;

}

continue Here;

}else {

boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));

boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));

boolean hasBi= bihua.containsKey(""+ kernel[i].charAt(point));

boolean hasBj= bihua.containsKey(""+ kernel[j].charAt(point));

if(!(!hasXi|| !hasXj)){//都有拼音

String[] js= new String[2];

js[0]= this.pinyin.get(""+ kernel[i].charAt(point));

js[1]= this.pinyin.get(""+ kernel[j].charAt(point));

if(js[0].equalsIgnoreCase(js[1])) {

if(!(!hasBi|| !hasBj)){//都有笔画

if(this.bihua.get(""+ kernel[i].charAt(point))

> this.bihua.get(""+ kernel[j].charAt(point))) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

continue Here;

}else if(this.bihua.get(""+ kernel[i].charAt(point))

== this.bihua.get(""+ kernel[j].charAt(point))) {

int asci=kernel[i].charAt(point);

int ascj=kernel[j].charAt(point);

if(asci< ascj) {//根据前面select的sort定义来规范,盲目改成大于会出错.

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

continue Here;

}

}

}

}

boolean change= processSortpinyin(js, 3);

if(!(!change|| i>= j)) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

continue Here;

}else if(!(hasXi|| !hasXj)){//其中一个有拼音

if(i< j) {

if(!(i== rightPosition+1 || j== rightPosition+1)) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

}

continue Here;

}else if(!(!hasXi|| hasXj)){

if(i> j) {

if(!(i== rightPosition+1 || j== rightPosition+1)) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

}

continue Here;

}else if(!(hasXi|| hasXj)){//都没有拼音

if(kernel[i].toLowerCase().charAt(point)

> kernel[j].toLowerCase().charAt(point)) {

if(i< j) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

continue Here;

}

if(kernel[i].toLowerCase().charAt(point)

== kernel[j].toLowerCase().charAt(point)) {

if(kernel[i].charAt(point)> kernel[j].charAt(point)) {

if(i< j) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

}

continue Here;

}

}

}

}

}

}

public void processQS4DLYG9D(String[] kernel, int leftPosition

, int rightPosition, int scale, int point, int deep) {

if(leftPosition< rightPosition){

int c= rightPosition- leftPosition+ 1;

if(!(c< this.range|| deep> this.deeps)) {//增加了deep

int pos= partition(kernel, leftPosition, rightPosition, scale, point);

if(leftPosition< pos- 1) {

processQS4DLYG9D(kernel, leftPosition, pos- 1, scale, point, deep+ 1);

}

if(pos+ 1< rightPosition) {

processQS4DLYG9D(kernel, pos+ 1, rightPosition, scale, point, deep+ 1);

}

return;

}

processSort(kernel, leftPosition, rightPosition, scale, point);

return;

}

}

public int partition(String[] array, int leftPosition, int rightPosition, int scale, int point) {

String x= findSmall(array, scale, point, leftPosition, rightPosition, rightPosition)

? array[rightPosition]: array[leftPosition];

int leftPositionReflection= leftPosition;

while(leftPositionReflection< rightPosition) {

while(!(findSmallWithTwoChar(array[leftPositionReflection]

, x, scale, point)|| leftPositionReflection++ >= rightPosition)) {}

while(findSmallWithTwoChar(array[rightPosition--], x, scale, point)){}

if(leftPositionReflection< ++rightPosition){

String temp= array[rightPosition].toString();;

array[rightPosition]= array[leftPositionReflection].toString();;

array[leftPositionReflection]= temp;

}

}

array[leftPosition]= array[rightPosition].toString();

array[rightPosition]= x.toString();

return rightPosition;

}

}

LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类

package OEU.LYG4DQS4D;

//import java.io.BufferedReader;

//import java.io.InputStream;

//import java.io.InputStreamReader;

//import java.util.HashMap;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort13D\_XCDX\_C\_A extends LYG10DWCMSSort13D\_XCDX\_S

implements LYG10DWCMSSort13D\_XCDX\_C\_A\_C{

public boolean findSmall(String[] kernel, int scale, int point

, int i, int j, int rightPosition) {

if(kernel[i].length()<= point|| kernel[j].length()<= point) {

if(kernel[i].length()< kernel[j].length()) {

for(int p= 0; p< scale; p++) {

if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {

if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {

return false;

}

}

}

return true;

}

return false;

}else {

boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));

boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));

if(!(!hasXi|| !hasXj)){

String[] js= new String[2];

js[0]= this.pinyin.get(""+ kernel[i].charAt(point));

js[1]= this.pinyin.get(""+ kernel[j].charAt(point));

boolean change= processSortpinyin(js, 3);

if(!(!change|| i>= j)) {

return true;

}

return false;

}else if(!(hasXi|| hasXj)){

if(kernel[i].toLowerCase().charAt(point)

> kernel[j].toLowerCase().charAt(point)) {

if(i< j) {

return true;

}

return false;

}else if(kernel[i].toLowerCase().charAt(point)

== kernel[j].toLowerCase().charAt(point)) {

if(kernel[i].charAt(point)> kernel[j].charAt(point)) {

if(i< j) {

return true;

}

return false;

}

return false;

}

return false;

}else if(!(hasXi|| !hasXj)){

if(i< j) {

if(!(i== rightPosition || j== rightPosition)) {

return true;

}

return false;

}

return false;

}

}

return false;

}

}

LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类

package OEU.LYG4DQS4D;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public interface LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C{

void processKernel(String[] kernel, int leftPosition

, int rightPosition, int scale, int point);

void processSort(String[] kernel, int leftPosition

, int rightPosition, int scale, int point);

void processQS4DLYG9D(String[] kernel, int leftPosition

, int rightPosition, int scale, int point, int deep);

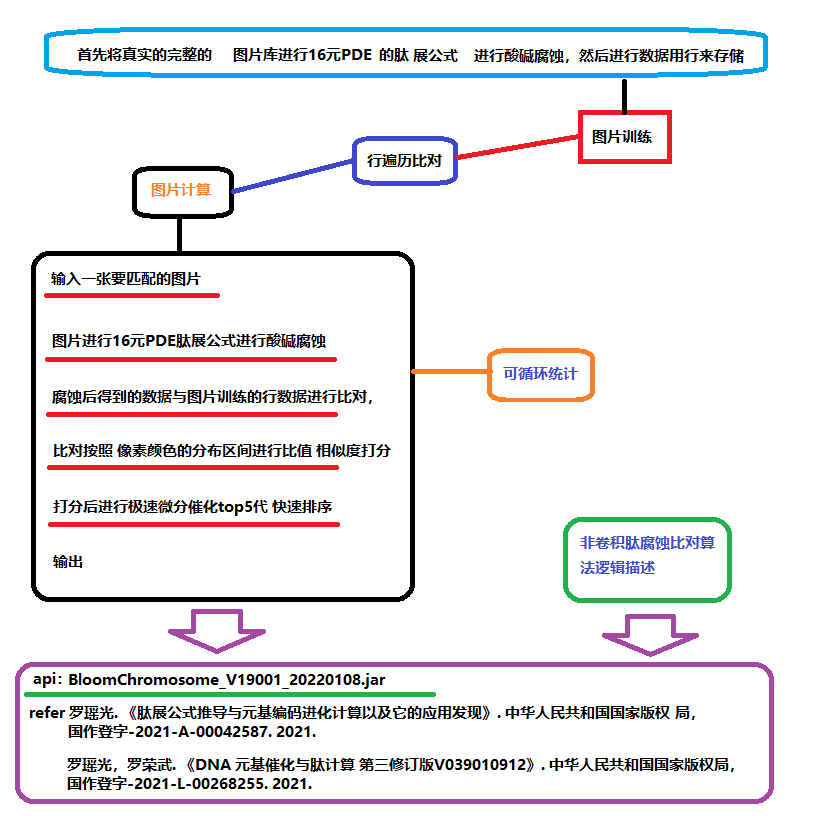
int partition(String[] array, int leftPosition, int rightPosition

, int scale, int point) ;

}

第三节 图片识别

图片读脏能力



SkinPathDetectDis, 肽展图片处理类

package ISQ.VSU.feature;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import ESU.array.Double\_ESU;

import OEU.LYG4DQS4D.LYG9DWithDoubleTopSort4D;

//准备用肽展公式来设计个 皮肤病检测程序.

//罗瑶光 20210710

//软件思想 肽展腐蚀变换. 计算机视觉.

//硬件工具, imageIO, javaCV. 像素头

//因为肽展公式和思想GPL2.0开源, 本程序源码同样GPL2.0开源.

//

public class SkinPathDetectDis{

public static List<Double[]> fileCells;

public static List<String> fileNames;

public static void main(String[] argv) throws IOException {

//皮肤病图片识别

//1 读取一张图片

//String testImagePath= "C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB.jpg";

String testImagePath= "D:\\bu20210606\\搜索图片\\搜索图片\\pgSearch\\皮肤性病学\\阿洪病-寻常狼疮.jpg";

//2 计算图片训练值

int 腐蚀浓度= 5;

int[][][] getSkinBy= new GetSkinBy().getSkinBy(testImagePath);

int[][][] doPDEofSkin= new DoPDEofSkin().doPDEofSkin\_IDUQ(getSkinBy, 腐蚀浓度);

//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB1.jpg", doPDEofSkin);

int 像素阀值= 10;

int 像素比精度= 5;

int 像素差精度= 8;

double[] getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度, 像素差精度);

//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字的黑色, 一些图片的红色.

//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.

double[] getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);

//

//3 遍历疾病数据表.

initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB6.txt");

//4 打分

Double[] score= new Double[fileNames.size()];

String[] nameScore= new String[fileNames.size()];

int pcaScale= 25;

int upcaScale= 15;

int icaScale= 15;

int ecaScale= 20;

GetColorRatioScore.scoreDeciphring(score, nameScore, getStatisticRatio, fileCells, fileNames

, pcaScale, upcaScale, icaScale, ecaScale);

//

腐蚀浓度= 95;

getSkinBy= new GetSkinBy().getSkinBy(testImagePath);

doPDEofSkin= new DoPDEofSkin().doPDEofSkin\_IDUQ(getSkinBy, 腐蚀浓度);

//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB1.jpg", doPDEofSkin);

像素阀值= 10;

像素比精度= 8;

像素差精度= 8;

getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度, 像素差精度);

//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字的黑色, 一些图片的红色.

//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.

getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);

//

//3 遍历疾病数据表.

initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB5.txt");

//4 打分

Double[] score酸= new Double[fileNames.size()];

GetColorRatioScore.scoreDeciphring(score酸, nameScore, getStatisticRatio, fileCells, fileNames

, pcaScale, upcaScale, icaScale, ecaScale);

//融合

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

score[i]+= score酸[i];

}

//5 筛选

double[] scoreDouble= new double[score.length];

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

scoreDouble[i]= score[i];//

}

//改成map

Map<Double, Map<String, Boolean>> map= Double\_ESU.getMapFromDoubleStringArray(score, nameScore);

int 递归深度= 70; //避免同值冗余内存高峰

int 堆栈广度= 7; //避免堆栈浪费计算高峰

new LYG9DWithDoubleTopSort4D().sort(scoreDouble, 堆栈广度, 递归深度);

//new Quick9DLYGWithString\_ESU().sort(scoreInt, nameScore);

//6 推荐

Here:

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

if(!map.containsKey(scoreDouble[i])) {

i++;

continue Here;

}

Iterator<String> iterator= map.get(scoreDouble[i]).keySet().iterator();

while(iterator.hasNext()) {

String string= iterator.next();

if(string.contains("狼")) {

System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+ scoreDouble[i]);

}else{

if(i< 20) {

System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+ scoreDouble[i]);

}

}

}

map.remove(scoreDouble[i]);

// System.out.println("相似图片:"+ nameScore[i]+ "-----分数:"+ scoreInt[i]);

}

}

private static void initDeciphring(String string) throws NumberFormatException, IOException {

fileNames= new ArrayList<>();

fileCells= new ArrayList<>();

String ctempString;

InputStream inputStream= new FileInputStream(string);

BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, "GBK"));

while ((ctempString= cReader.readLine())!= null) {

String[] lineCell= ctempString.split(">d>");

if(lineCell.length> 1) {

fileNames.add(lineCell[0]);

Double[] lineDouble= new Double[lineCell.length- 1];

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

lineDouble[i]= Double.valueOf(lineCell[i+ 1]);

}

fileCells.add(lineDouble);

}

}

cReader.close();

}

}

GetICAStatisticRatio, 肽展图片处理类

package ISQ.VSU.feature;

//准备用肽展公式来设计个 皮肤病检测程序.

//罗瑶光 20210710

//软件思想 肽展腐蚀变换. 计算机视觉.

//硬件工具, imageIO, javaCV.

//因为肽展公式和思想GPL2.0开源,本程序源码同样GPL2.0开源.

public class GetICAStatisticRatio{

int corr= 15;

int scale= 7;

public GetICAStatisticRatio(int scale, int corr) {

this.scale= scale;

this.corr= corr;

}

public int getPinkStatisticCount(int[][][] rgb) {

int ri= 255;

int gi= 0;

int bi= 255;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getWhiteStatisticCount(int[][][] rgb) {

int ri= 255;

int gi= 255;

int bi= 255;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getPupilStatisticCount(int[][][] rgb) {

int ri= 255;

int gi= 0;

int bi= 255;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getYellowStatisticCount(int[][][] rgb) {

int ri= 255;

int gi= 255;

int bi= 0;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getRedStatisticCount(int[][][] rgb) {

int ri= 255;

int gi= 0;

int bi= 0;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getGreenStatisticCount(int[][][] rgb) {

int ri= 0;

int gi= 255;

int bi= 0;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getBlueStatisticCount(int[][][] rgb) {

int ri= 0;

int gi= 0;

int bi= 255;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getMidPinkStatisticCount(int[][][] rgb) {

int ri= 128;

int gi= 0;

int bi= 128;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getMidWhiteStatisticCount(int[][][] rgb) {

int ri= 128;

int gi= 128;

int bi= 128;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getMidPupilStatisticCount(int[][][] rgb) {

int ri= 128;

int gi= 0;

int bi= 128;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getMidYellowStatisticCount(int[][][] rgb) {

int ri= 128;

int gi= 128;

int bi= 0;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getMidRedStatisticCount(int[][][] rgb) {

int ri= 128;

int gi= 0;

int bi= 0;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getMidGreenStatisticCount(int[][][] rgb) {

int ri= 0;

int gi= 128;

int bi= 0;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getMidBlueStatisticCount(int[][][] rgb) {

int ri= 0;

int gi= 0;

int bi= 128;

return getCountOfKernel(rgb, ri, gi, bi);

}

public int getCountOfKernel(int[][][] rgb, int ri, int gi, int bi) {

int count= 0;

for(int i= 0; i< rgb[0].length; i++) {

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

if(rgb[0][i][j]\* scale> ri- corr&& rgb[0][i][j]\* scale< ri+ corr) {

if(rgb[1][i][j]\* scale> gi- corr&& rgb[1][i][j]\* scale< gi+ corr) {

if(rgb[2][i][j]\* scale> bi- corr&& rgb[2][i][j]\* scale< bi+ corr) {

count++;

}

}

}

}

}

return count;

}

}

GetColorRatioScore, 肽展图片处理类

package ISQ.VSU.feature;

import java.util.Iterator;

import java.util.List;

//准备用肽展公式来设计个 皮肤病检测程序.

//罗瑶光 20210710

//软件思想 肽展腐蚀变换. 计算机视觉.

//硬件工具, imageIO, javaCV.

//因为肽展公式和思想GPL2.0开源,本程序源码同样GPL2.0开源.

public class GetColorRatioScore{

//RGBY

public void getRatioScore() {

}

public void getRGRatioScore() {

}

public void getRBRatioScore() {

}

public void getGBRatioScore() {

}

public void getGYRatioScore() {

}

public void getBYRatioScore() {

}

public void getRYRatioScore() {

}

static void scoreDeiphring(Double[] score, String[] nameScore, double[] getStatisticRatio

, List<Double[]> fileCells, List<String> fileNames) {

int lineCount= 0;

Iterator<Double[]> iteratorCells= fileCells.iterator();

Iterator<String> iteratorNames= fileNames.iterator();

while(iteratorNames.hasNext()) {

String name= iteratorNames.next();

nameScore[lineCount++]= name.toString();

}

lineCount= 0;

while(iteratorCells.hasNext()) {

Double[] Doubles= iteratorCells.next();

score[lineCount]= new Double(0);

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

try {

if(0== Doubles[i]|| 0== getStatisticRatio[i]) {

score[lineCount]+= 10000;

}else {

Double ratio= (getStatisticRatio[i]\* 10000)/ Doubles[i];

if(ratio>2||ratio<0.5) {

score[lineCount]+= 10000;

}else {

score[lineCount]+= ratio;

}

}

}catch(Exception e) {

System.out.println(Doubles[i]);

score[lineCount]+= 0;

}

}

score[lineCount]= Math.abs(score[lineCount]/ 45 - 1);//nor 均值进行了优化 20210714

lineCount++;

}

}

//Deiphring 分解为 Deciphring + Desiphine

static void scoreDeciphring(Double[] score, String[] nameScore, double[] getStatisticRatio

, List<Double[]> fileCells, List<String> fileNames, int pcaScale, int upcaScale, int icaScale, int ecaScale) {

int lineCount= 0;

Iterator<Double[]> iteratorCells= fileCells.iterator();

Iterator<String> iteratorNames= fileNames.iterator();

while(iteratorNames.hasNext()) {

String name= iteratorNames.next();

nameScore[lineCount++]= name.toString();

}

lineCount= 0;

while(iteratorCells.hasNext()) {

Double[] Doubles= iteratorCells.next();

score[lineCount]= new Double(0);

int PCA= 0;

int UPCA= 0;

int ICA= 0;

int ECA= 0;

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

try {

if(0== Doubles[i]|| 0== getStatisticRatio[i]) {

score[lineCount]+= 10000;

ICA++;

UPCA++;

}else {

Double ratio= (getStatisticRatio[i]\* 10000)/ Doubles[i];

if(ratio> 2|| ratio< 0.5) {

UPCA++;

score[lineCount]+= 10000;

}else if(ratio< 1.5|| ratio> 0.75) {

score[lineCount]+= ratio;

PCA++;

if(i> 2) {

ICA++;

}

if(ratio< 1.1|| ratio> 0.9) {

ECA++;

}

}

}

}catch(Exception e) {

System.out.println(Doubles[i]);

score[lineCount]+= 0;

}

}

if(PCA> pcaScale&& UPCA< upcaScale&& ICA> icaScale&& ECA> ecaScale) {

score[lineCount]= Math.abs(score[lineCount]/ 45 - 1);//nor 均值进行了优化 20210714

}else {

score[lineCount]= 99999.0;//nor 均值进行了优化 20210714

}

lineCount++;

}

}

// static void scoreDesiphine(Double[] score, String[] nameScore, double[] getStatisticRatio

// , List<Double[]> fileCells, List<String> fileNames) {

// int lineCount= 0;

// int ECA= 0;

// Iterator<Double[]> iteratorCells= fileCells.iterator();

// Iterator<String> iteratorNames= fileNames.iterator();

// while(iteratorNames.hasNext()) {

// String name= iteratorNames.next();

// nameScore[lineCount++]= name.toString();

// }

// lineCount= 0;

// while(iteratorCells.hasNext()) {

// Double[] Doubles= iteratorCells.next();

// score[lineCount]= new Double(0);

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

// try {

// if(0== Doubles[i]|| 0== getStatisticRatio[i]) {

// score[lineCount]+= 10000;

// }else {

// Double ratio= (getStatisticRatio[i]\* 10000)/ Doubles[i];

// if(ratio>2||ratio<0.5) {

// score[lineCount]+= 10000;

// }else {

// score[lineCount]+= ratio;

// }

// }

//

// }catch(Exception e) {

// System.out.println(Doubles[i]);

// score[lineCount]+= 0;

// }

// }

// score[lineCount]= Math.abs(score[lineCount]/ 45 - 1);//nor 均值进行了优化 20210714

// lineCount++;

// }

// }

}

SkinPathDetectTrip, 肽展图片处理类

package ISQ.VSU.feature;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import ESU.array.Double\_ESU;

import OEU.LYG4DQS4D.LYG9DWithDoubleTopSort4D;

//准备用肽展公式来设计个 皮肤病检测程序.

//罗瑶光 20210710

//软件思想 肽展腐蚀变换. 计算机视觉.

//硬件工具, imageIO, javaCV. 像素头

//因为肽展公式和思想GPL2.0开源, 本程序源码同样GPL2.0开源.

//

public class SkinPathDetectTrip{

public static List<Double[]> fileCells;

public static List<String> fileNames;

public static void main(String[] argv) throws IOException {

//皮肤病图片识别

//1 读取一张图片

//String testImagePath= "C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB.jpg";

String testImagePath= "C:\\Users\\Lenovo\\Desktop\\deciphering\\一般脏\_白癜风伴白发-色素失禁症-毛发稀疏伴色素沉着.jpg";

//2 计算图片训练值

int 腐蚀浓度= 5;

int[][][] getSkinBy= new GetSkinBy().getSkinBy(testImagePath);

int[][][] doPDEofSkin= new DoPDEofSkin().doPDEofSkin\_IDUQ(getSkinBy, 腐蚀浓度);

//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB1.jpg", doPDEofSkin);

int 像素阀值= 10;

int 像素比精度= 5;

int 像素差精度= 8;

double[] getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度, 像素差精度);

//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字的黑色, 一些图片的红色.

//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.

double[] getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);

//

//3 遍历疾病数据表.

initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB6.txt");

//4 打分

Double[] score= new Double[fileNames.size()];

String[] nameScore= new String[fileNames.size()];

int pcaScale= 25;

int upcaScale= 15;

int icaScale= 15;

int ecaScale= 20;

GetColorRatioScore.scoreDeciphring(score, nameScore, getStatisticRatio, fileCells, fileNames

, pcaScale, upcaScale, icaScale, ecaScale);

//

腐蚀浓度= 95;

getSkinBy= new GetSkinBy().getSkinBy(testImagePath);

doPDEofSkin= new DoPDEofSkin().doPDEofSkin\_IDUQ(getSkinBy, 腐蚀浓度);

//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB1.jpg", doPDEofSkin);

像素阀值= 10;

像素比精度= 8;

像素差精度= 8;

getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度, 像素差精度);

//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字的黑色, 一些图片的红色.

//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.

getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);

//

//3 遍历疾病数据表.

initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB5.txt");

//4 打分

Double[] score酸= new Double[fileNames.size()];

GetColorRatioScore.scoreDeciphring(score酸, nameScore, getStatisticRatio, fileCells, fileNames, pcaScale, upcaScale, icaScale, ecaScale);

//腐蚀浓度= 5;

getSkinBy= new GetSkinBy().getSkinBy(testImagePath);

doPDEofSkin= new DoPDEofSkin().doPDEofSkin\_AOPM\_VECS\_IDUQ\_TXH\_AC(getSkinBy, 腐蚀浓度);

//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB\_17.jpg", doPDEofSkin);

像素阀值= 10;

像素比精度= 2;

像素差精度= 8;

getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度, 像素差精度);

//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字的黑色, 一些图片的红色.

//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.

getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);

//

//3 遍历疾病数据表.

initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB6\_jian\_17.txt");

//4 打分

Double[] score\_17= new Double[fileNames.size()];

pcaScale= 1;

upcaScale= 45;

icaScale= 1;

ecaScale= 1;

GetColorRatioScore.scoreDeciphring(score\_17, nameScore, getStatisticRatio, fileCells, fileNames, pcaScale, upcaScale, icaScale, ecaScale);

//

腐蚀浓度= 95;

getSkinBy= new GetSkinBy().getSkinBy(testImagePath);

doPDEofSkin= new DoPDEofSkin().doPDEofSkin\_AOPM\_VECS\_IDUQ\_TXH\_AC(getSkinBy, 腐蚀浓度);

//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB\_17.jpg", doPDEofSkin);

像素阀值= 10;

像素比精度= 3;//采样是5

像素差精度= 8;

getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度, 像素差精度);

//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字的黑色, 一些图片的红色.

//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.

getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);

//

//3 遍历疾病数据表.

initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F\_DB6\_suan\_17.txt");

//4 打分

Double[] score\_酸17= new Double[fileNames.size()];

pcaScale= 1;

upcaScale= 45;

icaScale= 1;

ecaScale= 1;

GetColorRatioScore.scoreDeciphring(score\_酸17, nameScore, getStatisticRatio, fileCells, fileNames, pcaScale, upcaScale, icaScale, ecaScale);

//融合

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

//score[i]+= score酸[i]+ score\_17[i]+ score\_酸17[i];

score[i]= score[i]\* 1+ score酸[i]\* 0.001+ score\_17[i]\* 1+ score\_酸17[i]\* 1;

}

//5 筛选

double[] scoreDouble= new double[score.length];

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

scoreDouble[i]= score[i];//

}

//改成map

Map<Double, Map<String, Boolean>> map= Double\_ESU.getMapFromDoubleStringArray(score, nameScore);

int 递归深度= 70; //避免同值冗余内存高峰

int 堆栈广度= 7; //避免堆栈浪费计算高峰

new LYG9DWithDoubleTopSort4D().sort(scoreDouble, 堆栈广度, 递归深度);

//new Quick9DLYGWithString\_ESU().sort(scoreInt, nameScore);

//6 推荐

Here:

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

if(!map.containsKey(scoreDouble[i])) {

i++;

continue Here;

}

Iterator<String> iterator= map.get(scoreDouble[i]).keySet().iterator();

while(iterator.hasNext()) {

String string= iterator.next();

if(string.contains("白癜风伴")) {

System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+ scoreDouble[i]);

}else{

if(i< 20) {

System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+ scoreDouble[i]);

}

}

}

map.remove(scoreDouble[i]);

// System.out.println("相似图片:"+ nameScore[i]+ "-----分数:"+ scoreInt[i]);

}

}

private static void initDeciphring(String string) throws NumberFormatException, IOException {

fileNames= new ArrayList<>();

fileCells= new ArrayList<>();

String ctempString;

InputStream inputStream= new FileInputStream(string);

BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, "GBK"));

while ((ctempString= cReader.readLine())!= null) {

String[] lineCell= ctempString.split(">d>");

if(lineCell.length> 1) {

fileNames.add(lineCell[0]);

Double[] lineDouble= new Double[lineCell.length- 1];

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

lineDouble[i]= Double.valueOf(lineCell[i+ 1]);

}

fileCells.add(lineDouble);

}

}

cReader.close();

}

}

MakeImag, 肽展图片处理类

package ASQ.PSU.test;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics2D;

import java.awt.image.BufferedImage;

//import java.io.File;

import java.io.IOException;

import SVQ.stable.StableCommon;

//import javax.imageio.ImageIO;

//著作权人+作者= 罗瑶光

public class MakeImage {

public static void main(String[] args) throws IOException {

MakeImage makeImage = new MakeImage();

int height = 600;

int weight = 800;

BufferedImage image = new BufferedImage(weight, height, BufferedImage.TYPE\_INT\_RGB);

Graphics2D g = image.createGraphics();

g.setColor(Color.white);

g.setFont(g.getFont());

String prePrint = "....";

Font font = new Font(prePrint,20, 20);

int stringW = g.getFontMetrics().stringWidth(prePrint);

int stringH = g.getFontMetrics().getHeight();

System.out.println(stringW + StableCommon.STRING\_SYMBOL\_PER + stringH);

int positionX = 10;

int positionY = 20;

makeImage.weightPrint(prePrint, g, stringW, stringH, positionX, positionY, font);

makeImage.heightPrint(prePrint,g, stringW, stringH, positionX, positionY, font);

//String pathBin = "C:\\Users\\Administrator\\Desktop\\涔\images\\output.jpg";//output path

//File outputBin = new File(pathBin);

//ImageIO.write(image, "jpeg", outputBin);

}

public void weightPrint(String prePrint, Graphics2D g, int stringW, int stringH, int positionX, int positionY, Font font) {

g.setFont(font);

g.drawString(prePrint, positionX, positionY);

}

public void heightPrint(String prePrint, Graphics2D g, int stringW, int stringH, int positionX, int positionY, Font font) {

g.setFont(font);

g.drawString(prePrint, positionX, positionY);

}

}

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

package OEU.LYG4DQS4D;

import ASQ.PSU.test.TimeCheck;

//基于算法导论快排4衍生极速小高峰缺陷过滤理论快速排序第8代 线性数字数组排序法函数Java完整版本实现。

//思想：算法导论快排4理论，罗瑶光小高峰过滤理论。

//实现：罗瑶光

//时间：20140101~ 20200711

//复制一份 稍后准备 元基新陈代谢优化

public class LYG9DWithDoubleTopSort4D{

int range;

int deeps;

public double[] sort(double[] array, int range, int deeps) {

this.range= range;

this.deeps= deeps;

processDouble(array, 0, array.length- 1, 0);

return array;

}

private void processDouble(double[] array, int leftPoint, int rightPoint, int deep) {

int c= rightPoint- leftPoint+ 1;

if(!(c< this.range|| deep> this.deeps)) {//增加了deep

int pos= partition(array, leftPoint, rightPoint);

if(leftPoint< pos- 1) {

processDouble(array, leftPoint, pos- 1, deep+ 1);

}

if(pos+ 1< rightPoint) {

processDouble(array, pos+ 1, rightPoint, deep+ 1);

}

return;

}

int i= leftPoint;

for(int j= i+ 1; j< leftPoint+ c; j= i++){

while(j> leftPoint){

if(array[j]< array[--j]){

double temp= array[j+ 1];

array[j+ 1]= array[j];

array[j]= temp;

}

}

}

}

private int partition(double[] array, int leftPoint, int rightPoint) {

double x= array[leftPoint]< array[rightPoint]? array[leftPoint]: array[rightPoint];

int leftPointReflection= leftPoint;

while(leftPointReflection< rightPoint){

//我设立个top2D , --细节竟然没有一个人关注这些细节...20210716

while(!(array[leftPointReflection]> x|| leftPointReflection++ >= rightPoint)) {}

while(array[rightPoint--]> x) {}

if(leftPointReflection< ++rightPoint){

double temp= array[rightPoint];

array[rightPoint]= array[leftPointReflection];

array[leftPointReflection]= temp;

}

}

array[leftPoint]= array[rightPoint];

array[rightPoint]= x;

return rightPoint;

}

public static void main(String[] argv) {

double[] doubles=new double[9999999];

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

doubles[i]= Math.random();

}

LYG9DWithDoubleTopSort4D lYG9DWithDoubleTopSort2D= new LYG9DWithDoubleTopSort4D();

TimeCheck timecheck=new TimeCheck();

timecheck.begin();

lYG9DWithDoubleTopSort2D.sort(doubles, 7, 70);

timecheck.end();

timecheck.duration();

for(int i= 0; i< doubles.length- 1; i++) {

if(doubles[i]> doubles[i+ 1]) {

System.out.println(i+"->"+ doubles[i]);

}

}

System.out.println("end");

}

}

LYG10DWCMSSort15D\_XCDX\_P\_U\_A, 象契字符排序类

package OEU.LYG4DQS4D;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort15D\_XCDX\_P\_U\_A extends LYG10DWCMSSort13D\_XCDX\_P\_A

implements LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C{

public void processKernel2D(String[][] kernel, int leftPosition

, int rightPosition, int scale, int point, int culumn) {

int rightPositionReflection= rightPosition;

if(point> scale) {

return;

}

processQS4DLYG9D2D(kernel, leftPosition, rightPosition, scale, point, 0, culumn);

int i;

for(i= leftPosition; i<= rightPosition; i++) {

if(!(kernel[i][culumn].length()<= point

|| kernel[leftPosition][culumn].length()<= point)) {

if(kernel[i][culumn].charAt(point)

!= kernel[leftPosition][culumn].charAt(point)){

rightPositionReflection= i- 1;

processKernel2D(kernel, leftPosition, rightPositionReflection, scale

, point+ 1, culumn);

leftPosition= i;

}

}

}

if(leftPosition!= rightPosition) {

processKernel2D(kernel, leftPosition, i- 1, scale, point+ 1, culumn);

}

}

public void processQS4DLYG9D2D(String[][] kernel, int leftPosition

, int rightPosition, int scale, int point, int deep, int culumn) {

if(leftPosition< rightPosition){

int c= rightPosition- leftPosition+ 1;

if(!(c< this.range|| deep> this.deeps)) {//增加了deep

int pos= partition2D(kernel, leftPosition, rightPosition, scale, point

, culumn);

if(leftPosition< pos- 1) {

processQS4DLYG9D2D(kernel, leftPosition, pos- 1, scale, point

, deep+ 1, culumn);

}

if(pos+ 1< rightPosition) {

processQS4DLYG9D2D(kernel, pos+ 1, rightPosition, scale, point

, deep+ 1, culumn);

}

return;

}

processSort2D(kernel, leftPosition, rightPosition, scale, point, culumn);

return;

}

}

public int partition2D(String[][] array, int leftPosition, int rightPosition

, int scale, int point, int culumn) {

String[] x= findSmall2D(array, scale, point, leftPosition, rightPosition

, rightPosition, culumn)

? array[rightPosition]: array[leftPosition];

int leftPositionReflection= leftPosition;

while(leftPositionReflection< rightPosition) {

while(!(findSmallWithTwoChar(array[leftPositionReflection][culumn]

, x[culumn], scale, point)

|| leftPositionReflection++ >= rightPosition)) {}

while(findSmallWithTwoChar(array[rightPosition--][culumn]

, x[culumn], scale, point)){}

if(leftPositionReflection< ++rightPosition){

String[] temp= array[rightPosition].clone();;

array[rightPosition]= array[leftPositionReflection].clone();;

array[leftPositionReflection]= temp.clone();

}

}

array[leftPosition]= array[rightPosition].clone();

array[rightPosition]= x.clone();

return rightPosition;

}

public void processSort2D(String[][] kernel, int leftPosition

, int rightPosition, int scale, int point, int culumn) {

if(point> scale) {

return;

}

for(int i= leftPosition; i<= rightPosition; i++) {

Here:

for(int j= i; j<= rightPosition; j++) {

if(i== j) {

continue Here;

}

if(kernel[i][culumn].length()<= point

|| kernel[j][culumn].length()<= point) {

if(kernel[i][culumn].length()< kernel[j][culumn].length()) {

for(int p= 0; p< scale; p++) {

if(!(kernel[i][culumn].length()<= p

|| kernel[j][culumn].length()<= p)) {

if(kernel[i][culumn].charAt(p)

!= kernel[j][culumn].charAt(p)) {

continue Here;

}

}

}

String[] temp= kernel[i].clone();;

kernel[i]= kernel[j].clone();;

kernel[j]= temp.clone();

}

continue Here;

}else {

boolean hasXi= pinyin.containsKey(""

+ kernel[i][culumn].charAt(point));

boolean hasXj= pinyin.containsKey(""

+ kernel[j][culumn].charAt(point));

if(!(!hasXi|| !hasXj)){

String[] js= new String[2];

js[0]= this.pinyin.get(""+ kernel[i][culumn].charAt(point));

js[1]= this.pinyin.get(""+ kernel[j][culumn].charAt(point));

boolean change= processSortpinyin(js, 3);

if(!(!change|| i>= j)) {

String[] temp= kernel[i].clone();

kernel[i]= kernel[j].clone();

kernel[j]= temp.clone();

}

continue Here;

}else if(!(hasXi|| !hasXj)){

if(i< j) {

if(!(i== rightPosition+1 || j== rightPosition+1)) {

String[] temp= kernel[i].clone();

kernel[i]= kernel[j].clone();

kernel[j]= temp.clone();

}

}

continue Here;

}else if(!(hasXi|| hasXj)){

if(kernel[i][culumn].toLowerCase().charAt(point)

> kernel[j][culumn].toLowerCase().charAt(point)) {

if(i< j) {

String[] temp= kernel[i].clone();

kernel[i]= kernel[j].clone();

kernel[j]= temp.clone();

}

continue Here;

}

if(kernel[i][culumn].toLowerCase().charAt(point)

== kernel[j][culumn].toLowerCase().charAt(point)) {

if(kernel[i][culumn].charAt(point)> kernel[j][culumn].charAt(point)) {

if(i< j) {

String[] temp= kernel[i].clone();

kernel[i]= kernel[j].clone();

kernel[j]= temp.clone();

}

}

}

}

}

}

}

}

}

LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类

package OEU.LYG4DQS4D;

////import java.io.BufferedReader;

////import java.io.InputStream;

////import java.io.InputStreamReader;

////import java.util.HashMap;

////20200314 集成了最新的小高峰过滤催化排序5代思想。

//

////20200818 集成了最新的小高峰过滤催化排序9代思想。

////增加同拼音同笔画的字按char的int大小区分20210529

////罗瑶光

//import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort13D\_XCDX\_P\_A extends LYG10DWCMSSort13D\_XCDX\_S

implements LYG10DWCMSSort13D\_XCDX\_P\_A\_C{

public boolean findSmall2D(String[][] kernel, int scale, int point

, int i, int j, int rightPosition, int culumn) {

if(kernel[i][culumn].length()<= point

|| kernel[j][culumn].length()<= point) {

if(kernel[i][culumn].length()< kernel[j][culumn].length()) {

for(int p= 0; p< scale; p++) {

if(!(kernel[i][culumn].length()<= p|| kernel[j][culumn].length()<= p)) {

if(kernel[i][culumn].charAt(p)!= kernel[j][culumn].charAt(p)) {

return false;

}

}

}

return true;

}

return false;

}else {

boolean hasXi= pinyin.containsKey(""+ kernel[i][culumn].charAt(point));

boolean hasXj= pinyin.containsKey(""+ kernel[j][culumn].charAt(point));

if(!(!hasXi|| !hasXj)){

String[] js= new String[2];

js[0]= this.pinyin.get(""+ kernel[i][culumn].charAt(point));

js[1]= this.pinyin.get(""+ kernel[j][culumn].charAt(point));

boolean change= processSortpinyin(js, 3);

if(!(!change|| i>= j)) {

return true;

}

return false;

}else if(!(hasXi|| hasXj)){

if(kernel[i][culumn].toLowerCase().charAt(point)

> kernel[j][culumn].toLowerCase().charAt(point)) {

if(i< j) {

return true;

}

return false;

}else if(kernel[i][culumn].toLowerCase().charAt(point)

== kernel[j][culumn].toLowerCase().charAt(point)) {

if(kernel[i][culumn].charAt(point)

> kernel[j][culumn].charAt(point)) {

if(i< j) {

return true;

}

return false;

}

return false;

}

return false;

}else if(!(hasXi|| !hasXj)){

if(i< j) {

if(!(i== rightPosition || j== rightPosition)) {

return true;

}

return false;

}

return false;

}

}

return false;

}

}

LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C, 象契字符排序类

package OEU.LYG4DQS4D;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public interface LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C{

void processKernel2D(String[][] kernel, int leftPosition

, int rightPosition, int scale, int point, int culumn);

void processQS4DLYG9D2D(String[][] kernel, int leftPosition

, int rightPosition, int scale, int point, int deep, int culumn);

int partition2D(String[][] array, int leftPosition, int rightPosition

, int scale, int point, int culumn) ;

void processSort2D(String[][] kernel, int leftPosition

, int rightPosition, int scale, int point, int culumn);

}

LYG10DWCMSSort13D\_XCDX\_S, 象契字符排序类

package OEU.LYG4DQS4D;

//import java.io.BufferedReader;

//import java.io.InputStream;

//import java.io.InputStreamReader;

import java.util.HashMap;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort13D\_XCDX\_S implements LYG10DWCMSSort13D\_XCDX\_S\_C{

Map<String, Boolean> find= new HashMap<>();

public Map<String, String> pinyin;

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

int range;

int deeps;

String numbers= "0123456789.Ee";

public Map<String, Integer> bihua;

public boolean findSmallWithTwoChar(String x1, String x2

, int scale, int point) {

if(x1.length()<= point|| x2.length()<= point) {

if(x1.length()< x2.length()) {

for(int p= 0; p< scale; p++) {

if(!(x1.length()<= p|| x2.length()<= p)) {

if(x1.charAt(p)!= x2.charAt(p)) {

return false;

}

}

}

return true;

}

return false;

}else {

boolean hasX1= pinyin.containsKey(""+ x1.charAt(point));

boolean hasX2= pinyin.containsKey(""+ x2.charAt(point));

if(!(!hasX1|| !hasX2)){

String[] js= new String[2];

js[0]= this.pinyin.get(""+ x1.charAt(point));

js[1]= this.pinyin.get(""+ x2.charAt(point));

boolean changepinyin= processSortpinyin(js, 3);

if(changepinyin) {

return true;

}

return false;

}else if(!(hasX1|| hasX2)){

if(x1.toLowerCase().charAt(point)> x2.toLowerCase().charAt(point)) {

return true;

}else if(x1.toLowerCase().charAt(point)== x2.toLowerCase().charAt(point)) {

if(x1.charAt(point)> x2.charAt(point)) {

return true;

}

return false;

}

return false;

}else if(!(hasX1|| !hasX2)){

return true;

}

}

return false;

}

public boolean processSortpinyin(String[] kernel, int scale) {

for(int k= 0; k< scale; k++) {

if(kernel[0].length()<= k|| kernel[1].length()<= k) {

if(kernel[0].length()< kernel[1].length()) {

return true;

}

return false;

}

if(kernel[0].toLowerCase().charAt(k)

> kernel[1].toLowerCase().charAt(k)) {

return true;

}

if(kernel[0].toLowerCase().charAt(k)

< kernel[1].toLowerCase().charAt(k)) {

return false;

}

}

if(kernel[0].length()< kernel[1].length()) {

return true;

}

return false;

}

}

LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类

package OEU.LYG4DQS4D;

////import java.io.BufferedReader;

////import java.io.InputStream;

////import java.io.InputStreamReader;

////import java.util.HashMap;

////20200314 集成了最新的小高峰过滤催化排序5代思想。

////20200818 集成了最新的小高峰过滤催化排序9代思想。

////增加同拼音同笔画的字按char的int大小区分20210529

////罗瑶光

//import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort13D\_XCDX\_P\_A extends LYG10DWCMSSort13D\_XCDX\_S

implements LYG10DWCMSSort13D\_XCDX\_P\_A\_C{

public boolean findSmall2D(String[][] kernel, int scale, int point

, int i, int j, int rightPosition, int culumn) {

if(kernel[i][culumn].length()<= point

|| kernel[j][culumn].length()<= point) {

if(kernel[i][culumn].length()< kernel[j][culumn].length()) {

for(int p= 0; p< scale; p++) {

if(!(kernel[i][culumn].length()<= p|| kernel[j][culumn].length()<= p)) {

if(kernel[i][culumn].charAt(p)!= kernel[j][culumn].charAt(p)) {

return false;

}

}

}

return true;

}

return false;

}else {

boolean hasXi= pinyin.containsKey(""+ kernel[i][culumn].charAt(point));

boolean hasXj= pinyin.containsKey(""+ kernel[j][culumn].charAt(point));

if(!(!hasXi|| !hasXj)){

String[] js= new String[2];

js[0]= this.pinyin.get(""+ kernel[i][culumn].charAt(point));

js[1]= this.pinyin.get(""+ kernel[j][culumn].charAt(point));

boolean change= processSortpinyin(js, 3);

if(!(!change|| i>= j)) {

return true;

}

return false;

}else if(!(hasXi|| hasXj)){

if(kernel[i][culumn].toLowerCase().charAt(point)

> kernel[j][culumn].toLowerCase().charAt(point)) {

if(i< j) {

return true;

}

return false;

}else if(kernel[i][culumn].toLowerCase().charAt(point)

== kernel[j][culumn].toLowerCase().charAt(point)) {

if(kernel[i][culumn].charAt(point)

> kernel[j][culumn].charAt(point)) {

if(i< j) {

return true;

}

return false;

}

return false;

}

return false;

}else if(!(hasXi|| !hasXj)){

if(i< j) {

if(!(i== rightPosition || j== rightPosition)) {

return true;

}

return false;

}

return false;

}

}

return false;

}

}

LYG10DWCMSSort13D\_XCDX\_P\_A\_C, 象契字符排序类

package OEU.LYG4DQS4D;

////import java.io.BufferedReader;

////import java.io.InputStream;

////import java.io.InputStreamReader;

////import java.util.HashMap;

////20200314 集成了最新的小高峰过滤催化排序5代思想。

//

////20200818 集成了最新的小高峰过滤催化排序9代思想。

////增加同拼音同笔画的字按char的int大小区分20210529

////罗瑶光

//import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public interface LYG10DWCMSSort13D\_XCDX\_P\_A\_C{

boolean findSmall2D(String[][] kernel, int scale, int point

, int i, int j, int rightPosition, int culumn);

}

LYG10DWCMSSort13D\_XCDX\_S\_C, 象契字符排序类

package OEU.LYG4DQS4D;

//import java.io.BufferedReader;

//import java.io.InputStream;

//import java.io.InputStreamReader;

//import java.util.HashMap;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public interface LYG10DWCMSSort13D\_XCDX\_S\_C{

boolean findSmallWithTwoChar(String x1, String x2

, int scale, int point);

boolean processSortpinyin(String[] kernel, int scale);

}

LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类

package OEU.LYG4DQS4D;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort15D\_XCDX\_C\_U\_A extends LYG10DWCMSSort13D\_XCDX\_C\_A implements LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C {

public void processKernel(String[] kernel, int leftPosition

, int rightPosition, int scale, int point) {

int rightPositionReflection= rightPosition;

if(point> scale) {

return;

}

processQS4DLYG9D(kernel, leftPosition, rightPosition, scale, point, 0);

int i;

for(i= leftPosition; i<= rightPosition; i++) {

if(!(kernel[i].length()<= point|| kernel[leftPosition].length()<= point)) {

if(kernel[i].charAt(point)!= kernel[leftPosition].charAt(point)){

rightPositionReflection= i- 1;

processKernel(kernel, leftPosition, rightPositionReflection, scale, point+ 1);

leftPosition= i;

}

}

}

if(leftPosition!= rightPosition) {

processKernel(kernel, leftPosition, i- 1, scale, point+ 1);

}

}

public void processSort(String[] kernel, int leftPosition

, int rightPosition, int scale, int point) {

if(point> scale) {

return;

}

for(int i= leftPosition; i<= rightPosition; i++) {

Here:

for(int j= i; j<= rightPosition; j++) {

if(i== j) {

continue Here;

}

if(kernel[i].length()<= point|| kernel[j].length()<= point) {

if(kernel[i].length()< kernel[j].length()) {

for(int p= 0; p< scale; p++) {

if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {

if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {

continue Here;

}

}

}

String temp= kernel[i].toString();;

kernel[i]= kernel[j].toString();;

kernel[j]= temp;

}

continue Here;

}else {

boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));

boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));

boolean hasBi= bihua.containsKey(""+ kernel[i].charAt(point));

boolean hasBj= bihua.containsKey(""+ kernel[j].charAt(point));

if(!(!hasXi|| !hasXj)){//都有拼音

String[] js= new String[2];

js[0]= this.pinyin.get(""+ kernel[i].charAt(point));

js[1]= this.pinyin.get(""+ kernel[j].charAt(point));

if(js[0].equalsIgnoreCase(js[1])) {

if(!(!hasBi|| !hasBj)){//都有笔画

if(this.bihua.get(""+ kernel[i].charAt(point))

> this.bihua.get(""+ kernel[j].charAt(point))) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

continue Here;

}else if(this.bihua.get(""+ kernel[i].charAt(point))

== this.bihua.get(""+ kernel[j].charAt(point))) {

int asci=kernel[i].charAt(point);

int ascj=kernel[j].charAt(point);

if(asci< ascj) {//根据前面select的sort定义来规范,盲目改成大于会出错.

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

continue Here;

}

}

}

}

boolean change= processSortpinyin(js, 3);

if(!(!change|| i>= j)) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

continue Here;

}else if(!(hasXi|| !hasXj)){//其中一个有拼音

if(i< j) {

if(!(i== rightPosition+1 || j== rightPosition+1)) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

}

continue Here;

}else if(!(!hasXi|| hasXj)){

if(i> j) {

if(!(i== rightPosition+1 || j== rightPosition+1)) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

}

continue Here;

}else if(!(hasXi|| hasXj)){//都没有拼音

if(kernel[i].toLowerCase().charAt(point)

> kernel[j].toLowerCase().charAt(point)) {

if(i< j) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

continue Here;

}

if(kernel[i].toLowerCase().charAt(point)

== kernel[j].toLowerCase().charAt(point)) {

if(kernel[i].charAt(point)> kernel[j].charAt(point)) {

if(i< j) {

String temp= kernel[i].toString();

kernel[i]= kernel[j].toString();

kernel[j]= temp;

}

}

continue Here;

}

}

}

}

}

}

public void processQS4DLYG9D(String[] kernel, int leftPosition

, int rightPosition, int scale, int point, int deep) {

if(leftPosition< rightPosition){

int c= rightPosition- leftPosition+ 1;

if(!(c< this.range|| deep> this.deeps)) {//增加了deep

int pos= partition(kernel, leftPosition, rightPosition, scale, point);

if(leftPosition< pos- 1) {

processQS4DLYG9D(kernel, leftPosition, pos- 1, scale, point, deep+ 1);

}

if(pos+ 1< rightPosition) {

processQS4DLYG9D(kernel, pos+ 1, rightPosition, scale, point, deep+ 1);

}

return;

}

processSort(kernel, leftPosition, rightPosition, scale, point);

return;

}

}

public int partition(String[] array, int leftPosition, int rightPosition, int scale, int point) {

String x= findSmall(array, scale, point, leftPosition, rightPosition, rightPosition)

? array[rightPosition]: array[leftPosition];

int leftPositionReflection= leftPosition;

while(leftPositionReflection< rightPosition) {

while(!(findSmallWithTwoChar(array[leftPositionReflection]

, x, scale, point)|| leftPositionReflection++ >= rightPosition)) {}

while(findSmallWithTwoChar(array[rightPosition--], x, scale, point)){}

if(leftPositionReflection< ++rightPosition){

String temp= array[rightPosition].toString();;

array[rightPosition]= array[leftPositionReflection].toString();;

array[leftPositionReflection]= temp;

}

}

array[leftPosition]= array[rightPosition].toString();

array[rightPosition]= x.toString();

return rightPosition;

}

}

LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类

package OEU.LYG4DQS4D;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public interface LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C{

void processKernel(String[] kernel, int leftPosition

, int rightPosition, int scale, int point);

void processSort(String[] kernel, int leftPosition

, int rightPosition, int scale, int point);

void processQS4DLYG9D(String[] kernel, int leftPosition

, int rightPosition, int scale, int point, int deep);

int partition(String[] array, int leftPosition, int rightPosition

, int scale, int point) ;

}

LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类

package OEU.LYG4DQS4D;

//import java.io.BufferedReader;

//import java.io.InputStream;

//import java.io.InputStreamReader;

//import java.util.HashMap;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public class LYG10DWCMSSort13D\_XCDX\_C\_A extends LYG10DWCMSSort13D\_XCDX\_S

implements LYG10DWCMSSort13D\_XCDX\_C\_A\_C{

public boolean findSmall(String[] kernel, int scale, int point

, int i, int j, int rightPosition) {

if(kernel[i].length()<= point|| kernel[j].length()<= point) {

if(kernel[i].length()< kernel[j].length()) {

for(int p= 0; p< scale; p++) {

if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {

if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {

return false;

}

}

}

return true;

}

return false;

}else {

boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));

boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));

if(!(!hasXi|| !hasXj)){

String[] js= new String[2];

js[0]= this.pinyin.get(""+ kernel[i].charAt(point));

js[1]= this.pinyin.get(""+ kernel[j].charAt(point));

boolean change= processSortpinyin(js, 3);

if(!(!change|| i>= j)) {

return true;

}

return false;

}else if(!(hasXi|| hasXj)){

if(kernel[i].toLowerCase().charAt(point)

> kernel[j].toLowerCase().charAt(point)) {

if(i< j) {

return true;

}

return false;

}else if(kernel[i].toLowerCase().charAt(point)

== kernel[j].toLowerCase().charAt(point)) {

if(kernel[i].charAt(point)> kernel[j].charAt(point)) {

if(i< j) {

return true;

}

return false;

}

return false;

}

return false;

}else if(!(hasXi|| !hasXj)){

if(i< j) {

if(!(i== rightPosition || j== rightPosition)) {

return true;

}

return false;

}

return false;

}

}

return false;

}

}

LYG10DWCMSSort13D\_XCDX\_C\_A\_C, 象契字符排序类

package OEU.LYG4DQS4D;

//import java.io.BufferedReader;

//import java.io.InputStream;

//import java.io.InputStreamReader;

//import java.util.HashMap;

//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。

//增加同拼音同笔画的字按char的int大小区分20210529

//罗瑶光

//import java.util.Map;

//import SVQ.stable.StableFile;

//import PEU.S.verbal.VerbalSource;

//今天将新陈代谢技术应用到 中文拼音笔画分词上.

//罗瑶光

public interface LYG10DWCMSSort13D\_XCDX\_C\_A\_C{

boolean findSmall(String[] kernel, int scale, int point

, int i, int j, int rightPosition) ;

}

AE\_XCDX\_Map, 肽展中文分词类

package OEI.ME.analysis.E;

import java.io.IOException;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import ME.utils.WordFrequencyUtil;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

import OCI.ME.analysis.C.A\_XCDX\_Map;

import java.util.Iterator;

//PARSER XCDX

//Yaoguang.Luo

//20210702

//一种元基枝 写法结构 处理子类接口 分类.

//著作权人+作者= 罗瑶光

public class AE\_XCDX\_Map extends AE implements A\_XCDX\_Map{

public Map<String, WordFrequency> parserStringByReturnFrequencyMap(String inputString) {

Map<String, String> wordsForest= fHMMList.getPosCnToCn();

Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();

Map<Long, FMHMMNode> forestRoots= fHMMList.getMap();

int inputStringLength= inputString.length();

int forestDepth= StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords = new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO] = new StringBuilder();

fixWords[StablePOS.INT\_ONE] = new StringBuilder();

StringBuilder stringBuilder = new StringBuilder();

int find = StablePOS.INT\_ZERO;

Here:

for (int charPosition= StablePOS.INT\_ZERO; charPosition< inputStringLength; charPosition

+= (countInputStringLength== StablePOS.INT\_ZERO? StablePOS.INT\_ONE: countInputStringLength)) {

if(inputString.charAt(charPosition)< StablePOS.INT\_ONE\_TWO\_EIGHT){

if(fixWords[StablePOS.INT\_ZERO].length()> StablePOS.INT\_ZERO) {

if(fixWords[StablePOS.INT\_ZERO].charAt(fixWords[StablePOS.INT\_ZERO].length()- StablePOS.INT\_ONE)

< StablePOS.INT\_ONE\_TWO\_EIGHT) {

fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength= StablePOS.INT\_ONE;

find = StablePOS.INT\_ONE;

continue Here;

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

find=StablePOS.INT\_ONE;

fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength = StablePOS.INT\_ONE;

continue Here;

}

if(find == StablePOS.INT\_ONE) {

find = StablePOS.INT\_ZERO;

WordFrequencyUtil.WordFrequencyFindCheck(outputList, fixWords);

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder = neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString

.charAt(charPosition)), inputString, charPosition, inputStringLength, forestRoots, forestDepth

, charPosition + StablePOS.INT\_ONE);

String countWordNode = stringBuilder.toString();

int compare = countInputStringLength = countWordNode.length();

if (compare == StablePOS.INT\_ONE) {

WordFrequencyUtil.WordFrequencyCompareCheck(outputList, fixWords, countWordNode);

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (compare == StablePOS.INT\_TWO) {

countInputStringLength = nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength

, outputList, stringBuilder, wordsForest, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_THREE) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength = nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_FOUR) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength = nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder

, wordsForest, fixWords, posController);

}

}

return outputList;

}

public Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException {

Map<String, WordFrequency> map = new ConcurrentHashMap<>();

Iterator <String> iterator = sets.iterator();

Here:

while(iterator.hasNext()){

String setOfi = iterator.next();

if (map.containsKey(setOfi)) {

WordFrequency wordFrequency = map.get(setOfi);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

map.put(setOfi, wordFrequency);

continue Here;

}

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(setOfi);

map.put(setOfi, wordFrequency);

}

return map;

}

//计算函数以后移出 DNA元基组 到RNA.

public List<WordFrequency> sortWordFrequencyMap(Map<String, WordFrequency> map) throws IOException {

List<WordFrequency> list = quick6DLuoYaoguangSort.frequencyWordMapToList(map);

quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(list, StablePOS.INT\_ZERO

, list.size() - StablePOS.INT\_ONE);

return list;

}

public Map<Integer, WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException {

return sortWordFrequencyMapToSortMap(getWordFrequencyMap(sets));

}

public Map<Integer, WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String, WordFrequency> map){

Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);

return listMap;

}

public Map<Integer, WordFrequency> sortWordFrequencyMapToSortMap(Map<String, WordFrequency> map){

Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);

quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(listMap, StablePOS.INT\_ZERO

, listMap.size() - StablePOS.INT\_ONE);

return listMap;

}

public Map<String, WordFrequency> parserMixStringByReturnFrequencyMap(String mixedString) {

mixedString += StablePOS.SPACE\_STRING;

Map<String, String> wordsForest = fHMMList.getPosCnToCn();

Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();

Map<Long, FMHMMNode> forestRoots = fHMMList.getMap();//.getRoot();

int inputStringLength = mixedString.length();

int forestDepth = StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords = new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO] = new StringBuilder();

fixWords[StablePOS.INT\_ONE] = new StringBuilder();

StringBuilder stringBuilder = new StringBuilder();

int find = StablePOS.INT\_ZERO;

Here:

for (int charPosition = StablePOS.INT\_ZERO; charPosition < inputStringLength; charPosition

+= (countInputStringLength == StablePOS.INT\_ZERO ? StablePOS.INT\_ONE : countInputStringLength)) {

//luan ma

if(mixedString.charAt(charPosition) < StablePOS.INT\_ONE\_TWO\_EIGHT && charPosition < mixedString.length()

- StablePOS.INT\_ONE){

if(find == StablePOS.INT\_ZERO) {

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

fixWords[StablePOS.INT\_ZERO].append(mixedString.charAt(charPosition));

countInputStringLength = StablePOS.INT\_ONE;

find = StablePOS.INT\_ONE;

continue Here;

}

if(find == StablePOS.INT\_ONE) {

find = StablePOS.INT\_ZERO;

Iterator<String> it = fHMMList.englishStringToWordsList(fixWords[StablePOS.INT\_ZERO].toString()).iterator();

while(it.hasNext()) {

String temp=it.next();

if (outputList.containsKey(temp)) {

WordFrequency wordFrequency = outputList.get(temp);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(temp, wordFrequency);

} else {

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(temp);

outputList.put(temp, wordFrequency);

}

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder = neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString

.charAt(charPosition)), mixedString, charPosition, inputStringLength, forestRoots, forestDepth

, charPosition + StablePOS.INT\_ONE);

String countWordNode = stringBuilder.toString();

int compare = countInputStringLength = countWordNode.length();

if (compare == StablePOS.INT\_TWO) {

countInputStringLength = nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength

, outputList, stringBuilder, wordsForest, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_THREE) {

I\_FixWords(charPosition, mixedString, fixWords);

countInputStringLength = nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_ONE) {

if (outputList.containsKey(countWordNode)) {

WordFrequency wordFrequency = outputList.get(countWordNode);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(countWordNode, wordFrequency);

} else {

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(fixWords[StablePOS.INT\_ZERO].toString());

outputList.put(countWordNode, wordFrequency);

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (compare == StablePOS.INT\_FOUR) {

I\_FixWords(charPosition, mixedString, fixWords);

countInputStringLength = nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder

, wordsForest, fixWords, posController);

}

}

return outputList;

}

public List<WordFrequency> getWordFrequency(List<String> sets) throws IOException {

return sortWordFrequencyMap(getWordFrequencyMap(sets));

}

public EmotionMap getEmotionMap() {

return this.emotionMap;

}

}

AE, 肽展中文分词类

package OEI.ME.analysis.E;

import java.io.IOException;

import java.util.List;

import java.util.Map;

//import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

//import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

//import ME.utils.WordFrequencyUtil;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

import OCI.ME.analysis.C.A;

import OCI.ME.liner.C.Quick6DLuoYaoguangSort;

import OCI.ME.nero.C.NERO\_C\_OneTime;

//import OCI.ME.nlp.C.NLP\_C;

//import OCI.ME.pos.C.POS\_C;

//import OCI.ME.pos.C.Pos\_C\_XCDX;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

import OCI.SVQ.MPC.fhmm.C.FHMMList;

import OEI.AVC.SUQ.SVQ.MPC.fhmm.E.EmotionMap\_E;

import OEI.ME.liner.E.Quick6DLuoYaoguangSort3DMap\_E;

import OEI.ME.nero.E.NERO\_C\_OneTime\_E;

//import OEI.ME.nlp.E.NLP\_CE;

import OEI.ME.nlp.E.Nlp\_CE\_XCDX\_S;

//import OEI.ME.pos.E.POS\_CE;

//import OEI.ME.pos.E.Pos\_CE\_XCDX;

import OEI.ME.pos.E.Pos\_CE\_XCDX\_P;

import OEI.SVQ.MPC.fhmm.E.FMHMMListOneTime\_E;

import java.util.Iterator;

import java.util.LinkedList;

import PEQ.AMV.ECS.test.SensingTest;

//著作权人+作者= 罗瑶光

public class AE implements A {

protected FHMMList fHMMList;

protected NERO\_C\_OneTime neroController;

protected Nlp\_CE\_XCDX\_S nlpController;

protected Pos\_C\_XCDX\_P posController;

protected Quick6DLuoYaoguangSort quick6DLuoYaoguangSort;

protected Map<Long, FMHMMNode> forestRoots;

protected Map<Long, Map<String, String>> wordsForests;

protected Map<Long, FMHMMNode> []forestsRoots;

protected Map<String, String> wordsForest;

protected EmotionMap emotionMap;

protected SensingTest sensingTest;

public void IV\_() throws IOException {

this.fHMMList= new FMHMMListOneTime\_E();

fHMMList.index();

fHMMList.indexPosEnToCn();

fHMMList.indexPosEnToEn();

fHMMList.indexEnToCn();

fHMMList.indexCnToEn();

fHMMList.indexFullEnToCn();

fHMMList.indexFullCnToEn();

neroController= new NERO\_C\_OneTime\_E();

nlpController= new Nlp\_CE\_XCDX\_S();

posController= new Pos\_CE\_XCDX\_P();

quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap\_E();

forestRoots= fHMMList.getMap();

forestsRoots= fHMMList.getMaps();

wordsForest= fHMMList.getPosCnToCn();

wordsForests= fHMMList.getWordsForests();

}

public void IV\_Mixed() throws IOException {

this.fHMMList= new FMHMMListOneTime\_E();

fHMMList.indexMixed();

fHMMList.indexPosEnToCn();

fHMMList.indexPosEnToEn();

fHMMList.indexEnToCn();

fHMMList.indexCnToEn();

fHMMList.indexFullEnToCn();

fHMMList.indexFullCnToEn();

fHMMList.indexFullCnToPy();

fHMMList.indexFullCnToKo();;

fHMMList.indexFullCnToJp();;

fHMMList.indexFullCnToTt();

fHMMList.indexFullCnToRs();

fHMMList.indexFullCnToAb();

neroController= new NERO\_C\_OneTime\_E();

nlpController= new Nlp\_CE\_XCDX\_S();

posController= new Pos\_CE\_XCDX\_P();

quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap\_E();

forestRoots= fHMMList.getMap();

forestsRoots= fHMMList.getMaps();

wordsForest= fHMMList.getPosCnToCn();

wordsForests= fHMMList.getWordsForests();

emotionMap= new EmotionMap\_E();

emotionMap.IV\_MotivationMap();

emotionMap.IV\_NegativeMap();

emotionMap.IV\_PositiveMap();

emotionMap.IV\_TrendingMap();

emotionMap.IV\_PredictionMap();

sensingTest= new SensingTest();

}

public void IV\_All() throws IOException {

this.fHMMList= new FMHMMListOneTime\_E();

fHMMList.indexAll();

fHMMList.indexPosEnToCn();

fHMMList.indexPosEnToEn();

fHMMList.indexEnToCn();

fHMMList.indexCnToEn();

fHMMList.indexFullEnToCn();

fHMMList.indexFullCnToEn();

fHMMList.indexFullCnToPy();

fHMMList.indexFullCnToKo();;

fHMMList.indexFullCnToJp();;

fHMMList.indexFullCnToTt();

fHMMList.indexFullCnToRs();

fHMMList.indexFullCnToAb();

neroController= new NERO\_C\_OneTime\_E();

nlpController= new Nlp\_CE\_XCDX\_S();

posController= new Pos\_CE\_XCDX\_P();

quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap\_E();

forestRoots= fHMMList.getMap();

forestsRoots= fHMMList.getMaps();

wordsForest= fHMMList.getPosCnToCn();

wordsForests= fHMMList.getWordsForests();

emotionMap= new EmotionMap\_E();

emotionMap.IV\_MotivationMap();

emotionMap.IV\_NegativeMap();

emotionMap.IV\_PositiveMap();

emotionMap.IV\_TrendingMap();

emotionMap.IV\_PredictionMap();

sensingTest= new SensingTest();

}

public List<String> parserMixedString(String mixedString) {

mixedString+= StablePOS.SPACE\_STRING\_DISTINCTION;

int inputStringLength= mixedString.length();

List<String> outputList = new LinkedList<>();

int forestDepth = StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords = new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO] = new StringBuilder();

fixWords[StablePOS.INT\_ONE] = new StringBuilder();

StringBuilder stringBuilder = new StringBuilder();

int find = StablePOS.INT\_ZERO;

Here:

for (int charPosition = StablePOS.INT\_ZERO;charPosition<inputStringLength;charPosition

+=(countInputStringLength==StablePOS.INT\_ZERO?StablePOS.INT\_ONE:countInputStringLength)) {

if(mixedString.charAt(charPosition) < StablePOS.INT\_TEN\_SOUTHANDS && charPosition < inputStringLength

- StablePOS.INT\_ONE){

if(find == StablePOS.INT\_ZERO) {

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

fixWords[StablePOS.INT\_ZERO].append(mixedString.charAt(charPosition));

countInputStringLength = StablePOS.INT\_ONE;

find = StablePOS.INT\_ONE;

continue Here;

}

if(StablePOS.INT\_ONE == find) {

find = StablePOS.INT\_ZERO;

Iterator<String> it = fHMMList.englishStringToWordsList(fixWords[StablePOS.INT\_ZERO].toString()).iterator();

StringBuilder number= new StringBuilder();

while(it.hasNext()) {

String temp = it.next();

if(StablePOS.NUMBERS.contains(temp)) {

number.append(temp);

}else {

if(number.length()>0) {

outputList.add(number.toString());

number.delete(0, number.length());

}

outputList.add(temp);

}

}

if(number.length()>0) {

outputList.add(number.toString());

number.delete(0, number.length());

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder = neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString

.charAt(charPosition)), mixedString, charPosition, inputStringLength, forestRoots, forestDepth

, charPosition + StablePOS.INT\_ONE);

String countWordNode = stringBuilder.toString();

int compare = countInputStringLength = countWordNode.length();

if (StablePOS.INT\_ONE == compare) {

outputList.add(countWordNode);

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (StablePOS.INT\_TWO == compare) {

countInputStringLength = nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList

, stringBuilder, wordsForest, fixWords, posController, charPosition, mixedString);

continue Here;

}

if (StablePOS.INT\_THREE == compare) {

I\_FixWords(charPosition, mixedString, fixWords);

countInputStringLength = nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController, charPosition, mixedString);

continue Here;

}

if (StablePOS.INT\_FOUR == compare) {

I\_FixWords(charPosition, mixedString, fixWords);

countInputStringLength = nlpController.doSlangCheck(countInputStringLength, outputList, stringBuilder

, wordsForest, fixWords, posController, charPosition, mixedString);

}

}

return outputList;

}

public List<String> parserString(String inputString) {

List<String> outputList= new LinkedList<>();

int inputStringLength= inputString.length();

int forestDepth= StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords= new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO]= new StringBuilder();

fixWords[StablePOS.INT\_ONE]= new StringBuilder();

StringBuilder stringBuilder= new StringBuilder();

int find= StablePOS.INT\_ZERO;

Here:

for (int charPosition= StablePOS.INT\_ZERO; charPosition< inputStringLength; charPosition

+= (countInputStringLength!= StablePOS.INT\_ZERO? countInputStringLength: StablePOS.INT\_ONE)) {

if(StablePOS.INT\_ONE\_TWO\_EIGHT> inputString.charAt(charPosition)){

if(fixWords[StablePOS.INT\_ZERO].length()> StablePOS.INT\_ZERO) {

if(fixWords[StablePOS.INT\_ZERO].charAt(fixWords[StablePOS.INT\_ZERO].length()- StablePOS.INT\_ONE)< StablePOS.INT\_ONE\_TWO\_EIGHT) { fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength= StablePOS.INT\_ONE;

find= StablePOS.INT\_ONE;

continue Here;

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

find= StablePOS.INT\_ONE;

fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength= StablePOS.INT\_ONE;

continue Here;

}

if(find== StablePOS.INT\_ONE) {

find= StablePOS.INT\_ZERO;

outputList.add(fixWords[StablePOS.INT\_ZERO].toString());

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder= neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString

.charAt(charPosition)), inputString, charPosition, inputStringLength, forestRoots, forestDepth

, charPosition+ StablePOS.INT\_ONE);

String countWordNode= stringBuilder.toString();

int compare= countInputStringLength= countWordNode.length();

if (compare== StablePOS.INT\_ONE) {

outputList.add(countWordNode);

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (compare== StablePOS.INT\_TWO) {

countInputStringLength= nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList

, stringBuilder, wordsForest, fixWords, posController, charPosition, inputString);

continue Here;

}

if (compare== StablePOS.INT\_THREE) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength= nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController, charPosition, inputString);

continue Here;

}

if (compare== StablePOS.INT\_FOUR) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength= nlpController.doSlangCheck(countInputStringLength, outputList, stringBuilder

, wordsForest, fixWords, posController, charPosition, inputString);

}

}

return outputList;

}

public void I\_FixWords(int charPosition, String inputString, StringBuilder[] fixWords) {

fixWords[StablePOS.INT\_ONE].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ONE].length());

if (charPosition + StablePOS.INT\_EIGHT < inputString.length()) {

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_THREE, charPosition + StablePOS.INT\_EIGHT));

return;

}

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_THREE

, inputString.length()));

}

public String[] parserEnglishString(String englishString) {

String[] words = englishString.replaceAll(StablePOS.NLP\_SPASE\_REP, StablePOS.SPACE\_STRING)

.split(StablePOS.SPACE\_STRING);

if(StablePOS.INT\_ZERO == words.length ) {

return new String[] {StablePOS.SPACE\_STRING};

}

return words;

}

}

A, 肽展中文分词类

package OCI.ME.analysis.C;

import java.io.IOException;

import java.util.List;

//import java.util.Map;

//

//import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

//import PEQ.AMV.ECS.test.SensingTest;

//著作权人+作者= 罗瑶光

public interface A {

void IV\_() throws IOException;

void IV\_Mixed() throws IOException;

List<String> parserString(String input);

void I\_FixWords(int charPosition, String inputString,StringBuilder[] fixWords);

String[] parserEnglishString(String englishString);

List<String> parserMixedString(String mixedString);

}

A\_XCDX\_Map, 肽展中文分词类

package OCI.ME.analysis.C;

import java.io.IOException;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

//import PEQ.AMV.ECS.test.SensingTest;

public interface A\_XCDX\_Map {

Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException;

Map<Integer,WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException;

Map<Integer,WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String,WordFrequency> map);

Map<Integer,WordFrequency> sortWordFrequencyMapToSortMap(Map<String,WordFrequency> map);

Map<String,WordFrequency> parserStringByReturnFrequencyMap(String inputString);

Map<String,WordFrequency> parserMixStringByReturnFrequencyMap(String key);

List<WordFrequency> sortWordFrequencyMap(Map<String,WordFrequency> map) throws IOException;

List<WordFrequency> getWordFrequency(List<String> sets)throws IOException;

EmotionMap getEmotionMap();

}

BinaryForest, 肽展分词索引类

BinaryForest\_A, 肽展分词索引类

package OCI.ME.analysis.C;

import java.util.Map;

import PEQ.AMV.ECS.test.SensingTest;

//著作权人+作者= 罗瑶光

public interface BinaryForest\_A {

void studyNewWord(String study, String token, String posStudy);

Map<String, String> getStudyPos();

Map<String, String> getPinYin();

Map<String, String> getCtT();

Map<String, String> getCtK();

Map<String, String> getCtJ();

Map<String, String> getCtR();

Map<String, String> getCtA();

Map<String,String> getPosEnToCn();

Map<String,String> getPosEnToEn();

Map<String,String> getPosCnToCn();

Map<String,String> getEnToCn();

Map<String,String> getCnToEn();

Map<String,String> getFullEnToCn();

Map<String,String> getFullCnToEn();

SensingTest getSensingTest();

}

CogsBinaryForest\_AE, 肽展分词索引类

package OEI.ME.analysis.E;

import java.io.IOException;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import ME.utils.WordFrequencyUtil;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

import OCI.ME.analysis.C.CogsBinaryForest\_A;

import SVQ.stable.StablePOS;

//著作权人+作者= 罗瑶光

public class CogsBinaryForest\_AE extends BinaryForest\_AE implements CogsBinaryForest\_A {

public Map<String, WordFrequency> parserStringByReturnFrequencyMap(String inputString) {

Map<String, String> wordsForest = fHMMList.getPosCnToCn();

Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();

Map<Long, FMHMMNode> forestRoots = fHMMList.getMap();

int inputStringLength = inputString.length();

int forestDepth = StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords = new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO] = new StringBuilder();

fixWords[StablePOS.INT\_ONE] = new StringBuilder();

StringBuilder stringBuilder = new StringBuilder();

int find = StablePOS.INT\_ZERO;

Here:

for (int charPosition= StablePOS.INT\_ZERO; charPosition< inputStringLength; charPosition+= (countInputStringLength== StablePOS.INT\_ZERO? StablePOS.INT\_ONE: countInputStringLength)) {

if(inputString.charAt(charPosition)< StablePOS.INT\_ONE\_TWO\_EIGHT){

if(fixWords[StablePOS.INT\_ZERO].length()> StablePOS.INT\_ZERO) { if(fixWords[StablePOS.INT\_ZERO].charAt(fixWords[StablePOS.INT\_ZERO].length()- StablePOS.INT\_ONE)< StablePOS.INT\_ONE\_TWO\_EIGHT) { fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength= StablePOS.INT\_ONE;

find = StablePOS.INT\_ONE;

continue Here;

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

find=StablePOS.INT\_ONE; fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength = StablePOS.INT\_ONE;

continue Here;

}

if(find == StablePOS.INT\_ONE) {

find = StablePOS.INT\_ZERO;

WordFrequencyUtil.WordFrequencyFindCheck(outputList, fixWords);

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder = neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString

.charAt(charPosition)), inputString, charPosition, inputStringLength, forestRoots, forestDepth

, charPosition + StablePOS.INT\_ONE);

String countWordNode = stringBuilder.toString();

int compare = countInputStringLength = countWordNode.length();

if (compare == StablePOS.INT\_ONE) {

WordFrequencyUtil.WordFrequencyCompareCheck(outputList, fixWords, countWordNode);

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (compare == StablePOS.INT\_TWO) {

countInputStringLength = nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength

, outputList, stringBuilder, wordsForest, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_THREE) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength = nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_FOUR) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength = nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder

, wordsForest, fixWords, posController);

}

}

return outputList;

}

public Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException {

Map<String, WordFrequency> map = new ConcurrentHashMap<>();

Iterator <String> iterator = sets.iterator();

Here:

while(iterator.hasNext()){

String setOfi = iterator.next();

if (map.containsKey(setOfi)) {

WordFrequency wordFrequency = map.get(setOfi);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

map.put(setOfi, wordFrequency);

continue Here;

}

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(setOfi);

map.put(setOfi, wordFrequency);

}

return map;

}

//计算函数以后移出 DNA元基组 到RNA.

public List<WordFrequency> sortWordFrequencyMap(Map<String, WordFrequency> map) throws IOException {

List<WordFrequency> list = quick6DLuoYaoguangSort.frequencyWordMapToList(map);

quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(list, StablePOS.INT\_ZERO

, list.size() - StablePOS.INT\_ONE);

return list;

}

public Map<Integer, WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException {

return sortWordFrequencyMapToSortMap(getWordFrequencyMap(sets));

}

public Map<Integer, WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String, WordFrequency> map){

Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);

return listMap;

}

public Map<Integer, WordFrequency> sortWordFrequencyMapToSortMap(Map<String, WordFrequency> map){

Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);

quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(listMap, StablePOS.INT\_ZERO

, listMap.size() - StablePOS.INT\_ONE);

return listMap;

}

public Map<String, WordFrequency> parserMixStringByReturnFrequencyMap(String mixedString) {

mixedString += StablePOS.SPACE\_STRING;

Map<String, String> wordsForest = fHMMList.getPosCnToCn();

Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();

Map<Long, FMHMMNode> forestRoots = fHMMList.getMap();//.getRoot();

int inputStringLength = mixedString.length();

int forestDepth = StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords = new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO] = new StringBuilder();

fixWords[StablePOS.INT\_ONE] = new StringBuilder();

StringBuilder stringBuilder = new StringBuilder();

int find = StablePOS.INT\_ZERO;

Here:

for (int charPosition = StablePOS.INT\_ZERO; charPosition < inputStringLength; charPosition+= (countInputStringLength == StablePOS.INT\_ZERO ? StablePOS.INT\_ONE : countInputStringLength)) {

//luan ma

if(mixedString.charAt(charPosition) < StablePOS.INT\_ONE\_TWO\_EIGHT && charPosition < mixedString.length()

- StablePOS.INT\_ONE){

if(find == StablePOS.INT\_ZERO) {

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

fixWords[StablePOS.INT\_ZERO].append(mixedString.charAt(charPosition));

countInputStringLength = StablePOS.INT\_ONE;

find = StablePOS.INT\_ONE;

continue Here;

}

if(find == StablePOS.INT\_ONE) {

find = StablePOS.INT\_ZERO;

Iterator<String> it = fHMMList.englishStringToWordsList(fixWords[StablePOS.INT\_ZERO].toString()).iterator();

while(it.hasNext()) {

String temp=it.next();

if (outputList.containsKey(temp)) {

WordFrequency wordFrequency = outputList.get(temp);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(temp, wordFrequency);

} else {

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(temp);

outputList.put(temp, wordFrequency);

}

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder = neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString

.charAt(charPosition)), mixedString, charPosition, inputStringLength, forestRoots, forestDepth

, charPosition + StablePOS.INT\_ONE);

String countWordNode = stringBuilder.toString();

int compare = countInputStringLength = countWordNode.length();

if (compare == StablePOS.INT\_TWO) {

countInputStringLength = nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength

, outputList, stringBuilder, wordsForest, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_THREE) {

I\_FixWords(charPosition, mixedString, fixWords);

countInputStringLength = nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController);

continue Here;

}

if (compare == StablePOS.INT\_ONE) {

if (outputList.containsKey(countWordNode)) {

WordFrequency wordFrequency = outputList.get(countWordNode);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(countWordNode, wordFrequency);

} else {

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(fixWords[StablePOS.INT\_ZERO].toString());

outputList.put(countWordNode, wordFrequency);

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (compare == StablePOS.INT\_FOUR) {

I\_FixWords(charPosition, mixedString, fixWords);

countInputStringLength = nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder

, wordsForest, fixWords, posController);

}

}

return outputList;

}

public List<WordFrequency> getWordFrequency(List<String> sets) throws IOException {

return sortWordFrequencyMap(getWordFrequencyMap(sets));

}

public EmotionMap getEmotionMap() {

return this.emotionMap;

}

}

CogsBinaryForest\_A, 肽展分词索引类

package OCI.ME.analysis.C;

import java.io.IOException;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

//著作权人+作者= 罗瑶光

public interface CogsBinaryForest\_A extends BinaryForest\_A {

Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException;

Map<Integer,WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException;

Map<Integer,WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String,WordFrequency> map);

Map<Integer,WordFrequency> sortWordFrequencyMapToSortMap(Map<String,WordFrequency> map);

Map<String,WordFrequency> parserStringByReturnFrequencyMap(String inputString);

Map<String,WordFrequency> parserMixStringByReturnFrequencyMap(String key);

List<WordFrequency> sortWordFrequencyMap(Map<String,WordFrequency> map) throws IOException;

List<WordFrequency> getWordFrequency(List<String> sets)throws IOException;

EmotionMap getEmotionMap();

}

BinaryForest\_AE, 肽展分词索引类

package OEI.ME.analysis.E;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import OCI.ME.analysis.C.BinaryForest\_A;

import PEQ.AMV.ECS.test.SensingTest;

import SVQ.stable.StablePOS;

//著作权人+作者= 罗瑶光

public class BinaryForest\_AE extends AE implements BinaryForest\_A {

public void studyNewWord(String study, String token, String posStudy) {

//learn new word

FMHMMNode fFHMMNode= forestRoots.get(Long.valueOf(study.charAt(StablePOS.INT\_ZERO)));

Map<String, Integer> map;

if(null== fFHMMNode) {

fFHMMNode= new FMHMMNode();

map= new ConcurrentHashMap<>();

}else {

map= fFHMMNode.getNext();

}

map.put(token, StablePOS.INT\_ONE);

fFHMMNode.I\_Next(map);

forestRoots.put(Long.valueOf(study.charAt(StablePOS.INT\_ZERO)), fFHMMNode);

//learn new pos

fHMMList.studyNewPos(study+token, posStudy);

}

@Override

public Map<String, String> getStudyPos() {

return fHMMList.getStudyPos();

}

@Override

public Map<String, String> getPinYin() {

return fHMMList.getFullCnToPy();

}

@Override

public Map<String, String> getCtT() {

return fHMMList.getFullCnToTt();

}

@Override

public Map<String, String> getCtK() {

return fHMMList.getFullCnToKo();

}

@Override

public Map<String, String> getCtJ() {

return fHMMList.getFullCnToJp();

}

@Override

public Map<String, String> getCtR() {

// TODO Auto-generated method stub

return fHMMList.getFullCnToRs();

}

@Override

public Map<String, String> getCtA() {

// TODO Auto-generated method stub

return fHMMList.getFullCnToAb();

}

@Override

public SensingTest getSensingTest() {

// TODO Auto-generated method stub

return this.sensingTest;

}

public Map<String, String> getPosEnToCn() {

return fHMMList.getPosEnToCn();

}

public Map<String, String> getPosEnToEn() {

return fHMMList.getPosEnToEn();

}

@Override

public Map<String, String> getPosCnToCn() {

return fHMMList.getPosCnToCn();

}

public Map<String, String> getFullEnToCn() {

return fHMMList.getFullEnToCn();

}

public Map<String, String> getFullCnToEn() {

return fHMMList.getFullCnToEn();

}

public Map<String, String> getEnToCn() {

return fHMMList.getEnToCn();

}

public Map<String, String> getCnToEn() {

return fHMMList.getCnToEn();

}

}

Nlp\_CE\_XCDX\_A, 肽展分词索引类

package OEI.ME.nlp.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import ME.utils.WordForestUtil;

//import OCI.ME.nlp.C.NLP\_C;

import OCI.ME.nlp.C.Nlp\_C\_XCDX\_A;

//import OCI.ME.pos.C.POS\_C;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

import OCI.ME.pos.C.Pos\_C\_XCDX\_O;

//著作权人+作者= 罗瑶光

public class Nlp\_CE\_XCDX\_A implements Nlp\_C\_XCDX\_A{

public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String countWordNode= stringBuilder.toString();

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

if(StableMaps.CiTwo.containsKey(countWordNode)) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

outputList.add(countWordNode);

return countInputStringLength;

}

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(stringBuilder.charAt(StablePOS.INT\_ZERO));

outputList.add(StablePOS.EMPTY\_STRING+ stringBuilder.charAt(StablePOS.INT\_ZERO));

return countInputStringLength- StablePOS.INT\_ONE;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= countWordNode;

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength= posUtils.chuLiMingCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord, charPosition, textInputString);

return countInputStringLength;

}

if (StableMaps.baDongCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength = posUtils.chuLiBaDongCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())){

if (!StableMaps.jieCi.containsKey(countWordNode)){

countInputStringLength= ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

}

}

if (StableMaps.CiTwo.containsKey(countWordNode)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

outputList.add(countWordNode);

return countInputStringLength;

}

countInputStringLength= ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String inputString= stringBuilder.toString();

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE))

+ inputString.charAt(StablePOS.INT\_TWO);

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if (!StableMaps.CiOne.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if(StableMaps.lianCi.containsKey(strings[StablePOS.INT\_THREE])) {

countInputLength = posUtils.chuLiLianCiPostFixOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLianCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiJieCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiZhuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLiangCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiMingCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.shiTaiCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiShiTaiCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_ZERO])||StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

if(StableMaps.zhuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

&& (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_TWO])

|| StableMaps.jieCi.containsKey(strings[StablePOS.INT\_TWO]))) {

countInputLength = ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

} if(StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])||StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])

||StableMaps.daiCi.containsKey(strings[StablePOS.INT\_TWO])) {

countInputLength = ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiFuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList, stringsBuilder.append(strings[StablePOS.INT\_ONE])

, wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return StablePOS.INT\_ONE;

}

public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils){

String countWordNode= stringBuilder.toString();

if (!wordsForest.containsKey(countWordNode)){

WordForestUtil.wordsForestNotContainsKey(outputList, countWordNode, prefixWord);

return --countInputStringLength;

}

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

WordForestUtil.prefixWordEqualZero(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO))

+ String.valueOf(countWordNode.charAt(StablePOS.INT\_ONE));

if (wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

if (wordsForest.get(strings[StablePOS.INT\_ZERO]).contains(StablePOS.NLP\_CI\_MING)){

countInputStringLength= posUtils.chuLiMingCiOfTwoForMap(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

}

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

WordForestUtil.wordsForestContainsKey(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

return StablePOS.INT\_ZERO;

}

public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE)

+ inputString.charAt(StablePOS.INT\_TWO));

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (!wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiZhuCiOfThreeForMap(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLiangCiOfThreeForMap(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiJieCiOfThreeForMap(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLianCiOfThreeForMap(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

}

Nlp\_C\_XCDX\_A, 肽展分词索引类

package OCI.ME.nlp.C;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//import OCI.ME.pos.C.POS\_C;

//import OCI.ME.pos.C.Pos\_C\_XCDX;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

//著作权人+作者= 罗瑶光

public interface Nlp\_C\_XCDX\_A {

public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils);

public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils);

}

Nlp\_CE\_XCDX\_S, 肽展分词索引类

package OEI.ME.nlp.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import ME.utils.WordForestUtil;

//import OCI.ME.nlp.C.NLP\_C;

import OCI.ME.nlp.C.Nlp\_C\_XCDX\_S;

//import OCI.ME.pos.C.POS\_C;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

//import OCI.ME.pos.C.Pos\_C\_XCDX\_O;

//繁衍和继承是新陈代谢的一种体现.

//著作权人+作者= 罗瑶光

public class Nlp\_CE\_XCDX\_S extends Nlp\_CE\_XCDX\_A implements Nlp\_C\_XCDX\_S{

// 2个月研究发现 词性越来越多, 根据笛摩根定律, 先把未知词汇也添加到条件中. 之后采用 排除法优化.

// if(StableMaps.jieCi.containsKey(preRegister)|| StableMaps.mingCi.containsKey(preRegister)|| StableMaps.xingRongCi.containsKey(preRegister)

// || StableMaps.fuCi.containsKey(preRegister)|| StableMaps.dongCi.containsKey(preRegister)|| StableMaps.lianCi.containsKey(preRegister)

// || StableMaps.liangCi.containsKey(preRegister)|| StableMaps.xingWeiCi.containsKey(preRegister)|| StableMaps.shiTaiCi.containsKey(preRegister)

// || StableMaps.zhuCi.containsKey(preRegister)) {

//if(StableMaps.mingCi.containsKey(postRegister)|| StableMaps.dongCi.containsKey(postRegister)|| StableMaps.lianCi.containsKey(postRegister)

// || StableMaps.xingRongCi.containsKey(postRegister)|| StableMaps.xingWeiCi.containsKey(postRegister)|| StableMaps.liangCi.containsKey(preRegister)

// || StableMaps.fuCi.containsKey(postRegister)|| StableMaps.jieCi.containsKey(postRegister)) {

public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder,

Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String inputString = stringBuilder.toString();

if (StableMaps.CiFour.containsKey(inputString)){

output.add(inputString);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

return countInputStringLength;

}//will make pre 3 or post 3 check. now finished pre 3 .20190330

String preRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE);

String inRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ONE)+ inputString.charAt(StablePOS.INT\_TWO);

String postRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_TWO)+ inputString.charAt(StablePOS.INT\_THREE);

if(StableMaps.dongCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_THREE)

+ prefixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))) {

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(preRegister)){

if (StableMaps.CiTwo.containsKey(postRegister)){

String string= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO); if(StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

&&StableMaps.shiTaiCi.containsKey(string)) {

output.add(string);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(string);

return countInputStringLength- StablePOS.INT\_THREE;

}

if(StableMaps.zhuCi.containsKey(string)){

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE))

+ inputString.charAt(StablePOS.INT\_TWO);

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

countInputStringLength= posUtils.chuLiZhuCiOfThree(wordsForest, output, countInputStringLength-StablePOS.INT\_ONE, strings, prefixWord);

return countInputStringLength;

}

output.add(preRegister);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(preRegister);

return countInputStringLength-StablePOS.INT\_TWO;

}

}

if(StableMaps.CiThree.containsKey(preRegister+ inputString.charAt(StablePOS.INT\_TWO))&& !StableMaps.CiTwo.containsKey(postRegister)) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(preRegister+ inputString.charAt(StablePOS.INT\_TWO));

output.add(preRegister+ inputString.charAt(StablePOS.INT\_TWO));

return countInputStringLength- StablePOS.INT\_ONE ;

}

if(StableMaps.CiTwo.containsKey(preRegister)&& StableMaps.CiTwo.containsKey(inRegister)) {

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

if(StableMaps.CiTwo.containsKey(preRegister)) {

countInputStringLength= doSlangPartAndPOSCheckForTwoChar(countInputStringLength- StablePOS.INT\_TWO, output

, stringBuilder.delete(StablePOS.INT\_TWO, StablePOS.INT\_FOUR), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

output.add(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO));

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO));

return countInputStringLength= StablePOS.INT\_ONE;

}

//卡诺图化简.PCA阀门分流. 卷积催化, .原来备注这里 ,20190523

// if(!wordsForest.containsKey(preRegister)&& (wordsForest.containsKey(inRegister)||wordsForest.containsKey(postRegister))) {

// if(wordsForest.containsKey(preRegister+ inputString.charAt(StableData.INT\_TWO))) {

// output.add(preRegister+ inputString.charAt(StableData.INT\_TWO));

// prefixWord[StableData.INT\_ZERO].delete(StableData.INT\_ZERO, prefixWord[StableData.INT\_ZERO].length());

// prefixWord[StableData.INT\_ZERO].append(preRegister+ inputString.charAt(StableData.INT\_TWO));

// return countInputStringLength- StableData.INT\_ONE;

// }

// output.add(StableData.EMPTY\_STRING+ inputString.charAt(StableData.INT\_ZERO));

// prefixWord[StableData.INT\_ZERO].delete(StableData.INT\_ZERO, prefixWord[StableData.INT\_ZERO].length());

// prefixWord[StableData.INT\_ZERO].append(StableData.EMPTY\_STRING+ inputString.charAt(StableData.INT\_ZERO));

// return countInputStringLength- StableData.INT\_THREE;

//}

//if(wordsForest.containsKey(preRegister)&& wordsForest.containsKey(inRegister+ inputString.charAt(StableData.INT\_THREE))) {

// countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

// , stringBuilder.delete(StableData.INT\_THREE, StableData.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

// return countInputStringLength;

//}

//if(wordsForest.containsKey(preRegister)) {

// countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

// , stringBuilder.delete(StableData.INT\_THREE, StableData.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

// return countInputStringLength;

//}

public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

output.add(inputString);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

return countInputStringLength;

}

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(output, inputString, prefixWord);

return countInputStringLength;

}

if(StableMaps.mingCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE))) {

if(StableMaps.mingCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_TWO)+ inputString.charAt(StablePOS.INT\_THREE))) {

WordForestUtil.wordsForestContainsKey(output, StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE), prefixWord);

return countInputStringLength;

}

}

countInputStringLength= doPOSAndEMMCheckOfThreeForMap(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils);

return countInputStringLength;

}

}

Nlp\_CE\_XCDX\_A, 肽展分词索引类

package OEI.ME.nlp.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import ME.utils.WordForestUtil;

//import OCI.ME.nlp.C.NLP\_C;

import OCI.ME.nlp.C.Nlp\_C\_XCDX\_A;

//import OCI.ME.pos.C.POS\_C;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

import OCI.ME.pos.C.Pos\_C\_XCDX\_O;

//著作权人+作者= 罗瑶光

public class Nlp\_CE\_XCDX\_A implements Nlp\_C\_XCDX\_A{

public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String countWordNode= stringBuilder.toString();

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

if(StableMaps.CiTwo.containsKey(countWordNode)) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

outputList.add(countWordNode);

return countInputStringLength;

}

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(stringBuilder.charAt(StablePOS.INT\_ZERO));

outputList.add(StablePOS.EMPTY\_STRING+ stringBuilder.charAt(StablePOS.INT\_ZERO));

return countInputStringLength- StablePOS.INT\_ONE;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= countWordNode;

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength= posUtils.chuLiMingCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord, charPosition, textInputString);

return countInputStringLength;

}

if (StableMaps.baDongCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength = posUtils.chuLiBaDongCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())){

if (!StableMaps.jieCi.containsKey(countWordNode)){

countInputStringLength= ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

}

}

if (StableMaps.CiTwo.containsKey(countWordNode)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

outputList.add(countWordNode);

return countInputStringLength;

}

countInputStringLength= ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String inputString= stringBuilder.toString();

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE))

+ inputString.charAt(StablePOS.INT\_TWO);

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if (!StableMaps.CiOne.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if(StableMaps.lianCi.containsKey(strings[StablePOS.INT\_THREE])) {

countInputLength = posUtils.chuLiLianCiPostFixOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLianCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiJieCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiZhuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLiangCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiMingCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.shiTaiCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiShiTaiCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_ZERO])

||StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

if(StableMaps.zhuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

&& (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_TWO])

|| StableMaps.jieCi.containsKey(strings[StablePOS.INT\_TWO]))) {

countInputLength = ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if(StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])||StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])

||StableMaps.daiCi.containsKey(strings[StablePOS.INT\_TWO])) {

countInputLength = ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiFuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList, stringsBuilder.append(strings[StablePOS.INT\_ONE])

, wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return StablePOS.INT\_ONE;

}

public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils){

String countWordNode= stringBuilder.toString();

if (!wordsForest.containsKey(countWordNode)){

WordForestUtil.wordsForestNotContainsKey(outputList, countWordNode, prefixWord);

return --countInputStringLength;

}

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

WordForestUtil.prefixWordEqualZero(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO))

+ String.valueOf(countWordNode.charAt(StablePOS.INT\_ONE));

if (wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

if (wordsForest.get(strings[StablePOS.INT\_ZERO]).contains(StablePOS.NLP\_CI\_MING)){

countInputStringLength= posUtils.chuLiMingCiOfTwoForMap(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

}

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

WordForestUtil.wordsForestContainsKey(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

return StablePOS.INT\_ZERO;

}

public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE)

+ inputString.charAt(StablePOS.INT\_TWO));

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (!wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiZhuCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLiangCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiJieCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLianCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

}

Nlp\_C\_XCDX\_S, 肽展分词索引类

package OCI.ME.nlp.C;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//import OCI.ME.pos.C.POS\_C;

//import OCI.ME.pos.C.Pos\_C\_XCDX;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

//著作权人+作者= 罗瑶光

public interface Nlp\_C\_XCDX\_S {

public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder,

Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils);

}

Nlp\_CE\_XCDX, 肽展分词索引类

package OEI.ME.nlp.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import ME.utils.WordForestUtil;

import OCI.ME.nlp.C.NLP\_C;

//import OCI.ME.pos.C.POS\_C;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

import OCI.ME.pos.C.Pos\_C\_XCDX\_O;

//著作权人+作者= 罗瑶光

public class Nlp\_CE\_XCDX implements NLP\_C{

public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String countWordNode= stringBuilder.toString();

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

if(StableMaps.CiTwo.containsKey(countWordNode)) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

outputList.add(countWordNode);

return countInputStringLength;

}

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(stringBuilder.charAt(StablePOS.INT\_ZERO));

outputList.add(StablePOS.EMPTY\_STRING+ stringBuilder.charAt(StablePOS.INT\_ZERO));

return countInputStringLength- StablePOS.INT\_ONE;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= countWordNode;

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength= posUtils.chuLiMingCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord, charPosition, textInputString);

return countInputStringLength;

}

if (StableMaps.baDongCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength = posUtils.chuLiBaDongCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())){

if (!StableMaps.jieCi.containsKey(countWordNode)){

countInputStringLength= ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

}

}

if (StableMaps.CiTwo.containsKey(countWordNode)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

outputList.add(countWordNode);

return countInputStringLength;

}

countInputStringLength= ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String inputString= stringBuilder.toString();

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE))

+ inputString.charAt(StablePOS.INT\_TWO);

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if (!StableMaps.CiOne.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if(StableMaps.lianCi.containsKey(strings[StablePOS.INT\_THREE])) {

countInputLength = posUtils.chuLiLianCiPostFixOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLianCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiJieCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiZhuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLiangCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiMingCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.shiTaiCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiShiTaiCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_ZERO])||StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

if(StableMaps.zhuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

&& (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_TWO])

|| StableMaps.jieCi.containsKey(strings[StablePOS.INT\_TWO]))) {

countInputLength = ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if(StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])||StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])

||StableMaps.daiCi.containsKey(strings[StablePOS.INT\_TWO])) {

countInputLength = ((Pos\_C\_XCDX\_O)posUtils).parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiFuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList, stringsBuilder.append(strings[StablePOS.INT\_ONE])

, wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return StablePOS.INT\_ONE;

}

// 2个月研究发现 词性越来越多, 根据笛摩根定律, 先把未知词汇也添加到条件中. 之后采用 排除法优化.

// if(StableMaps.jieCi.containsKey(preRegister)|| StableMaps.mingCi.containsKey(preRegister)|| StableMaps.xingRongCi.containsKey(preRegister)

// || StableMaps.fuCi.containsKey(preRegister)|| StableMaps.dongCi.containsKey(preRegister)|| StableMaps.lianCi.containsKey(preRegister)

// || StableMaps.liangCi.containsKey(preRegister)|| StableMaps.xingWeiCi.containsKey(preRegister)|| StableMaps.shiTaiCi.containsKey(preRegister)

// || StableMaps.zhuCi.containsKey(preRegister)) {

//if(StableMaps.mingCi.containsKey(postRegister)|| StableMaps.dongCi.containsKey(postRegister)|| StableMaps.lianCi.containsKey(postRegister)

// || StableMaps.xingRongCi.containsKey(postRegister)|| StableMaps.xingWeiCi.containsKey(postRegister)|| StableMaps.liangCi.containsKey(preRegister)

// || StableMaps.fuCi.containsKey(postRegister)|| StableMaps.jieCi.containsKey(postRegister)) {

public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder,

Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String inputString = stringBuilder.toString();

if (StableMaps.CiFour.containsKey(inputString)){

output.add(inputString);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

return countInputStringLength;

}//will make pre 3 or post 3 check. now finished pre 3 .20190330

String preRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE);

String inRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ONE)+ inputString.charAt(StablePOS.INT\_TWO);

String postRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_TWO)+ inputString.charAt(StablePOS.INT\_THREE);

if(StableMaps.dongCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_THREE)

+ prefixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))) {

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(preRegister)){

if (StableMaps.CiTwo.containsKey(postRegister)){

String string= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO);

if(StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

&&StableMaps.shiTaiCi.containsKey(string)) {

output.add(string);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(string);

return countInputStringLength- StablePOS.INT\_THREE;

}

if(StableMaps.zhuCi.containsKey(string)){

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE))

+ inputString.charAt(StablePOS.INT\_TWO);

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

countInputStringLength= posUtils.chuLiZhuCiOfThree(wordsForest, output, countInputStringLength-StablePOS.INT\_ONE, strings, prefixWord);

return countInputStringLength;

}

output.add(preRegister);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(preRegister);

return countInputStringLength-StablePOS.INT\_TWO;

}

}

if(StableMaps.CiThree.containsKey(preRegister+ inputString.charAt(StablePOS.INT\_TWO))&& !StableMaps.CiTwo.containsKey(postRegister)) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(preRegister+ inputString.charAt(StablePOS.INT\_TWO));

output.add(preRegister+ inputString.charAt(StablePOS.INT\_TWO));

return countInputStringLength- StablePOS.INT\_ONE ;

}

if(StableMaps.CiTwo.containsKey(preRegister)&& StableMaps.CiTwo.containsKey(inRegister)) {

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

if(StableMaps.CiTwo.containsKey(preRegister)) {

countInputStringLength= doSlangPartAndPOSCheckForTwoChar(countInputStringLength- StablePOS.INT\_TWO, output

, stringBuilder.delete(StablePOS.INT\_TWO, StablePOS.INT\_FOUR), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

output.add(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO));

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO));

return countInputStringLength= StablePOS.INT\_ONE;

}

//卡诺图化简.PCA阀门分流. 卷积催化, .原来备注这里 ,20190523

// if(!wordsForest.containsKey(preRegister)&& (wordsForest.containsKey(inRegister)||wordsForest.containsKey(postRegister))) {

// if(wordsForest.containsKey(preRegister+ inputString.charAt(StableData.INT\_TWO))) {

// output.add(preRegister+ inputString.charAt(StableData.INT\_TWO));

// prefixWord[StableData.INT\_ZERO].delete(StableData.INT\_ZERO, prefixWord[StableData.INT\_ZERO].length());

// prefixWord[StableData.INT\_ZERO].append(preRegister+ inputString.charAt(StableData.INT\_TWO));

// return countInputStringLength- StableData.INT\_ONE;

// }

// output.add(StableData.EMPTY\_STRING+ inputString.charAt(StableData.INT\_ZERO));

// prefixWord[StableData.INT\_ZERO].delete(StableData.INT\_ZERO, prefixWord[StableData.INT\_ZERO].length());

// prefixWord[StableData.INT\_ZERO].append(StableData.EMPTY\_STRING+ inputString.charAt(StableData.INT\_ZERO));

// return countInputStringLength- StableData.INT\_THREE;

//}

//if(wordsForest.containsKey(preRegister)&& wordsForest.containsKey(inRegister+ inputString.charAt(StableData.INT\_THREE))) {

// countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

// , stringBuilder.delete(StableData.INT\_THREE, StableData.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

// return countInputStringLength;

//}

//if(wordsForest.containsKey(preRegister)) {

// countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

// , stringBuilder.delete(StableData.INT\_THREE, StableData.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

// return countInputStringLength;

//}

public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

output.add(inputString);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

return countInputStringLength;

}

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils){

String countWordNode= stringBuilder.toString();

if (!wordsForest.containsKey(countWordNode)){

WordForestUtil.wordsForestNotContainsKey(outputList, countWordNode, prefixWord);

return --countInputStringLength;

}

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

WordForestUtil.prefixWordEqualZero(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO))

+ String.valueOf(countWordNode.charAt(StablePOS.INT\_ONE));

if (wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

if (wordsForest.get(strings[StablePOS.INT\_ZERO]).contains(StablePOS.NLP\_CI\_MING)){

countInputStringLength= posUtils.chuLiMingCiOfTwoForMap(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

}

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

WordForestUtil.wordsForestContainsKey(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

return StablePOS.INT\_ZERO;

}

public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE)

+ inputString.charAt(StablePOS.INT\_TWO));

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (!wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiZhuCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLiangCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiJieCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLianCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(output, inputString, prefixWord);

return countInputStringLength;

}

if(StableMaps.mingCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE))) {

if(StableMaps.mingCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_TWO)+ inputString.charAt(StablePOS.INT\_THREE))) {

WordForestUtil.wordsForestContainsKey(output, StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE), prefixWord);

return countInputStringLength;

}

}

countInputStringLength= doPOSAndEMMCheckOfThreeForMap(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils);

return countInputStringLength;

}

}

POS\_C\_Cognition\_E, 肽展分词索引类

package OEI.ME.pos.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import OCI.ME.pos.C.POS\_C;

//著作权人+作者= 罗瑶光

public class POS\_C\_Cognition\_E implements POS\_C{

@Override

public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] prefixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord, int charPosition, String inputString) {

// TODO Auto-generated method stub

return 0;

}

@Override

public void I\_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {

// TODO Auto-generated method stub

}

@Override

public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings,

StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest,

int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength) {

// TODO Auto-generated method stub

return 0;

}

@Override

public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord,

Map<String, String> wordsForest) {

// TODO Auto-generated method stub

}

@Override

public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings,

StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings,

StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList,

String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest,

int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings,

int[] nestCountInputStringLength) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings,

StringBuilder[] fixWord, Map<String, String> wordsForest) {

// TODO Auto-generated method stub

}

@Override

public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList,

String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

// TODO Auto-generated method stub

return 0;

}

@Override

public int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

// TODO Auto-generated method stub

return 0;

}

}

POS\_C, 肽展分词索引类

package OCI.ME.pos.C;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//著作权人+作者= 罗瑶光

public interface POS\_C {

int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] prefixWord);

int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord, int charPosition, String inputString);

void I\_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords);

int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings

, StringBuilder[] fixWord);

int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest

, int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength);

void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest);

int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings

, StringBuilder[] fixWord);

int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings

, StringBuilder[] fixWord);

int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings

, StringBuilder[] fixWord, Map<String, String> wordsForest);

int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest

, int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, int[] nestCountInputStringLength);

int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord);

void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest);

int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings

, StringBuilder[] fixWord, Map<String, String> wordsForest);

int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord);

int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord);

int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord);

int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord);

}

Pos\_CE\_XCDX\_E, 肽展分词索引类

package OEI.ME.pos.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import OCI.ME.pos.C.Pos\_C\_XCDX\_E;

//著作权人+作者= 罗瑶光

public class Pos\_CE\_XCDX\_E extends Pos\_CE\_XCDX\_O implements Pos\_C\_XCDX\_E{

public void I\_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {

fixWords[StablePOS.INT\_ONE].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ONE].length());

if (charPosition+ StablePOS.INT\_SEVEN < inputString.length()) {

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_TWO, charPosition + StablePOS.INT\_SEVEN));

return;

}

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_TWO, inputString.length()));

}

public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength){

String charPositionAtFixWord = StablePOS.EMPTY\_STRING + fixWord[StablePOS.INT\_ONE].charAt(backPosition);

if (wordsForest.containsKey(charPositionAtFixWord) && (StableMaps.zhuCi.containsKey(charPositionAtFixWord)

|| StableMaps.shengLueCi.containsKey(charPositionAtFixWord)|| StableMaps.fuCi.containsKey(charPositionAtFixWord))){

if(!wordsForest.get(fixWord[StablePOS.INT\_ZERO].toString()).contains(StablePOS.NLP\_CI\_SHENG\_LUE)&& wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP\_CI\_FU)){

return StablePOS.INT\_ERROR;

}

nestCountInputStringLength[StablePOS.INT\_ZERO]= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return StablePOS.INT\_RIGHT;

}

return StablePOS.INT\_ERROR;

}

public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest){

if(!wordsForest.containsKey(strings[StablePOS.INT\_TWO])){

if(wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

}

return;

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return;

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

}

}

public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, int[] nestCountInputStringLength){

String charPositionAtFixWord= StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ONE].charAt(backPosition);

if (wordsForest.containsKey(charPositionAtFixWord)&& (StableMaps.zhuCi.containsKey(charPositionAtFixWord)

|| wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP\_CI\_SHENG\_LUE))){

nestCountInputStringLength[StablePOS.INT\_ZERO]= parserFirstCharOfThreeForMap(countInputStringLength, outputList

, strings, fixWord, wordsForest);

return StablePOS.INT\_RIGHT;

}

return StablePOS.INT\_ERROR;

}

public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest){

if(!wordsForest.containsKey(strings[StablePOS.INT\_TWO])){

return;

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])){

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

if (outputList.containsKey(strings[StablePOS.INT\_TWO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_TWO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_TWO]);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

}

}

}

Pos\_CE\_XCDX\_O, 肽展分词索引类

package OEI.ME.pos.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import OCI.ME.pos.C.Pos\_C\_XCDX\_O;

///

/// O o

/// E e

/// P p

//著作权人+作者= 罗瑶光

public class Pos\_CE\_XCDX\_O implements Pos\_C\_XCDX\_O{

public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord){

outputList.add(strings[StablePOS.INT\_ZERO]);

String postNext=StablePOS.EMPTY\_STRING + strings[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ONE);

outputList.add(postNext);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(postNext);

return countInputStringLength;

}

public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

return countInputStringLength;

}

public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest){

countInputStringLength -= StablePOS.INT\_TWO;

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength += StablePOS.INT\_ONE;

return countInputStringLength;

}

public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings

, StringBuilder[] fixWord, Map<String, String> wordsForest){

countInputStringLength-= StablePOS.INT\_THREE;

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength ++;

if (wordsForest.containsKey(strings[StablePOS.INT\_TWO])){

if (outputList.containsKey(strings[StablePOS.INT\_TWO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_TWO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_TWO]);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

countInputStringLength+= StablePOS.INT\_TWO;

return countInputStringLength;

}

return countInputStringLength;

}

}

Pos\_C\_XCDX\_E, 肽展分词索引类

package OCI.ME.pos.C;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//著作权人+作者= 罗瑶光

public interface Pos\_C\_XCDX\_E{

void I\_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords);

int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength);

int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, int[] nestCountInputStringLength);

void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest);

void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest);

}

Pos\_C\_XCDX\_O, 肽展分词索引类

package OCI.ME.pos.C;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//著作权人+作者= 罗瑶光

public interface Pos\_C\_XCDX\_O {

int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord);

int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord);

int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord);

int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest);

int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest);

}

Pos\_CE\_XCDX\_P, 肽展分词索引类

package OEI.ME.pos.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

//著作权人+作者= 罗瑶光

public class Pos\_CE\_XCDX\_P extends Pos\_CE\_XCDX\_E implements Pos\_C\_XCDX\_P{

public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] prefixWord){

if (!wordsForest.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())){

return countInputStringLength;

}

if (StableMaps.daiCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.fuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())) {

countInputStringLength = parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength;

}

return countInputStringLength- StablePOS.INT\_TWO;

}

public int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord, int charPosition, String inputString){

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.liangCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.xingRongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.liangCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_ONE]);

return countInputStringLength;

}

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

I\_FixWordsOfTwo(charPosition, inputString, fixWord);

if (StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.fuCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].toString().charAt(StablePOS.INT\_ZERO))){

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength;

}

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

return countInputStringLength;

}

public int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_TWO])){

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

countInputLength= parserFirstTwoCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

return countInputLength;

}

public int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (outputList.size() == StablePOS.INT\_ZERO){

didNotFindFirstChar(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString()) && (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.fuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()) || StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString()) && (StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.shengLueCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

for (int BackPosition = StablePOS.INT\_ZERO; BackPosition < fixWord[StablePOS.INT\_ONE].length(); BackPosition++){

int[] nestCountInputStringLength = new int[StablePOS.INT\_ONE];

int result = loopCheckBackFix(fixWord, BackPosition, wordsForest, countInputStringLength, outputList, strings

, nestCountInputStringLength);

if (result == StablePOS.INT\_RIGHT){

return nestCountInputStringLength[StablePOS.INT\_ZERO];

}

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength- StablePOS.INT\_ONE;

}

return countInputStringLength- StablePOS.INT\_THREE;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

return countInputStringLength- StablePOS.INT\_ONE;

}

return countInputStringLength- StablePOS.INT\_THREE;

}

public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (StablePOS.INT\_ZERO== outputList.size()){

didNotFindFirstChar(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if(wordsForest.containsKey(strings[StablePOS.INT\_TWO])) {

countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

} else if(fixWord[StablePOS.INT\_ONE].length()> StablePOS.INT\_ONE) {

String firstChar= StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ONE].toString().charAt(StablePOS.INT\_ZERO);

String secondChar= StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ONE].toString().charAt(StablePOS.INT\_ONE); if(!StableMaps.fuCi.containsKey(firstChar)&& !StableMaps.fuCi.containsKey(secondChar)

&&!StableMaps.fuCi.containsKey(firstChar+ secondChar)) {

if(wordsForest.containsKey(firstChar)&& wordsForest.containsKey(secondChar)) {

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length()); fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength= StablePOS.INT\_ONE;

if(wordsForest.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length()); fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

countInputStringLength= StablePOS.INT\_THREE;

}

return countInputStringLength;

}

}

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength- StablePOS.INT\_ONE;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

}

return countInputStringLength;

}

public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings, StringBuilder[] fixWord){

if (StablePOS.INT\_ZERO== outputList.size()&& (wordsForest.get(strings[StablePOS.INT\_TWO])

.contains(StablePOS.NLP\_CI\_WEI))){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

if (outputList.size() > StablePOS.INT\_ZERO&& wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.qingTaiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.lianCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

} else{

if(StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())) {

if(StableMaps.xingWeiCi.containsKey(strings[StablePOS.INT\_ONE])

|| StableMaps.xingRongCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}else if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

if(StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength = StablePOS.INT\_TWO;

return countInputStringLength;

}else if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

}

}

if(StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return StablePOS.INT\_TWO;

}

if ((StableMaps.xingWeiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())||StableMaps.xingRongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))

&& StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return StablePOS.INT\_TWO;

}

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_ZERO]);

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return StablePOS.INT\_THREE;

}

return StablePOS.INT\_ONE;

}

public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.liangCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength = parserFirstCharOfTwoForMap(countInputStringLength, outputList, strings, fixWord

, wordsForest);

return countInputStringLength;

}

countInputStringLength -= StablePOS.INT\_TWO;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength += StablePOS.INT\_TWO;

}

return countInputStringLength;

}

return countInputStringLength;

}

public int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord

, wordsForest);

return countInputStringLength;

}

countInputStringLength -= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength += StablePOS.INT\_TWO;

}

return countInputStringLength;

}

return countInputStringLength;

}

public int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (outputList.size()== StablePOS.INT\_ZERO&& StableMaps.weiCi.containsKey(strings[StablePOS.INT\_TWO])){

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

if (outputList.containsKey(strings[StablePOS.INT\_TWO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_TWO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_TWO]);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

if (outputList.size()> StablePOS.INT\_ZERO&& wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.lianCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.qingTaiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord, wordsForest);

return countInputStringLength;

} else{

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength+= StablePOS.INT\_TWO;

}

return countInputStringLength;

}

}

return countInputStringLength;

}

public int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (outputList.size()== StablePOS.INT\_ZERO){

didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

&& (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.fuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

countInputStringLength = parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

&& (StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.shengLueCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

for (int BackPosition= StablePOS.INT\_ZERO; BackPosition< fixWord[StablePOS.INT\_ONE].length(); BackPosition++){

int[] nestCountInputStringLength = new int[StablePOS.INT\_ONE];

int result= loopCheckBackFixForMap(fixWord, BackPosition, wordsForest, countInputStringLength, outputList, strings

, nestCountInputStringLength);

if (result== StablePOS.INT\_RIGHT){

return nestCountInputStringLength[StablePOS.INT\_ZERO];

}

}

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength += StablePOS.INT\_TWO;

}

return countInputStringLength;

}

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

countInputStringLength+= StablePOS.INT\_TWO;

}

return countInputStringLength;

}

public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (StablePOS.INT\_ZERO== outputList.size()){

didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord, wordsForest);

return countInputStringLength;

} else{

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength+= StablePOS.INT\_TWO;

}

return countInputStringLength;

}

}

return countInputStringLength;

}

public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord){

if (StableMaps.xingWeiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.mingCi

.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO])){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_THREE])){

if(StableMaps.fuCi.containsKey(StablePOS.EMPTY\_STRING+ (0== fixWord[StablePOS.INT\_ONE].length()? "@^\_^@": fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO)))){

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_ONE])|| StableMaps.qingTaiCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

if(!StableMaps.dingMingCi.containsKey(strings[StablePOS.INT\_ZERO])){

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

}

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])){

if(StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))){

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

if(!StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ONE])|| StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ONE])){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength= StablePOS.INT\_ONE;

return countInputStringLength;

}

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_THREE])){

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO])){

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])) {

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

if(null!= fixWord[StablePOS.INT\_ZERO]&& StablePOS.INT\_ZERO<fixWord[StablePOS.INT\_ZERO].length()){

if(StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ZERO].charAt(StablePOS.INT\_ZERO))){

if(!StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])){

if(StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))){

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

if(!StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.dingMingCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength= StablePOS.INT\_TWO;

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

if ((StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO].toString()))

&& (StableMaps.jieCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString()))){

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO].toString())

||StableMaps.liangCi.containsKey(strings[StablePOS.INT\_TWO].toString())) {

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputLength= StablePOS.INT\_TWO;

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputLength;

}

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputLength= StablePOS.INT\_ONE;

}

public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO].toString())) {

if (StableMaps.fuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())) {

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO].toString())) {

if (StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING+ prefixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))

|| StableMaps.daiCi.containsKey(StablePOS.EMPTY\_STRING+ prefixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))) {

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

countInputLength= parserFirstTwoCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return StablePOS.INT\_ONE;

}

}

Pos\_CE\_XCDX\_E, 肽展分词索引类

package OEI.ME.pos.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import OCI.ME.pos.C.Pos\_C\_XCDX\_E;

//著作权人+作者= 罗瑶光

public class Pos\_CE\_XCDX\_E extends Pos\_CE\_XCDX\_O implements Pos\_C\_XCDX\_E{

public void I\_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {

fixWords[StablePOS.INT\_ONE].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ONE].length());

if (charPosition+ StablePOS.INT\_SEVEN < inputString.length()) {

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_TWO, charPosition + StablePOS.INT\_SEVEN));

return;

}

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_TWO, inputString.length()));

}

public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength){

String charPositionAtFixWord = StablePOS.EMPTY\_STRING + fixWord[StablePOS.INT\_ONE].charAt(backPosition);

if (wordsForest.containsKey(charPositionAtFixWord) && (StableMaps.zhuCi.containsKey(charPositionAtFixWord)

|| StableMaps.shengLueCi.containsKey(charPositionAtFixWord)|| StableMaps.fuCi.containsKey(charPositionAtFixWord))){ if(!wordsForest.get(fixWord[StablePOS.INT\_ZERO].toString()).contains(StablePOS.NLP\_CI\_SHENG\_LUE)&& wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP\_CI\_FU)){

return StablePOS.INT\_ERROR;

}

nestCountInputStringLength[StablePOS.INT\_ZERO]= parserFirstCharOfThree(countInputStringLength, outputList

, strings, fixWord);

return StablePOS.INT\_RIGHT;

}

return StablePOS.INT\_ERROR;

}

public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest){

if(!wordsForest.containsKey(strings[StablePOS.INT\_TWO])){

if(wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

}

return;

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return;

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

}

}

public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest

, int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, int[] nestCountInputStringLength){

String charPositionAtFixWord= StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ONE].charAt(backPosition);

if (wordsForest.containsKey(charPositionAtFixWord)&& (StableMaps.zhuCi.containsKey(charPositionAtFixWord)

|| wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP\_CI\_SHENG\_LUE))){

nestCountInputStringLength[StablePOS.INT\_ZERO]= parserFirstCharOfThreeForMap(countInputStringLength, outputList

, strings, fixWord, wordsForest);

return StablePOS.INT\_RIGHT;

}

return StablePOS.INT\_ERROR;

}

public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest){

if(!wordsForest.containsKey(strings[StablePOS.INT\_TWO])){

return;

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])){

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

if (outputList.containsKey(strings[StablePOS.INT\_TWO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_TWO]);

wordFrequency.I\_Frequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(strings[StablePOS.INT\_TWO]);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

}

}

}

Pos\_C\_XCDX\_P, 肽展分词索引类

package OCI.ME.nlp.C;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//import OCI.ME.pos.C.POS\_C;

//import OCI.ME.pos.C.Pos\_C\_XCDX;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

//著作权人+作者= 罗瑶光

public interface Nlp\_C\_XCDX\_A {

public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils);

public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils);

}

Pos\_CE\_XCDX, 肽展分词索引类

package OEI.ME.pos.E;

import OCI.ME.pos.C.Pos\_C\_XCDX;

import OCI.ME.pos.C.Pos\_C\_XCDX\_E;

import OCI.ME.pos.C.Pos\_C\_XCDX\_O;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

//著作权人+作者= 罗瑶光

public class Pos\_CE\_XCDX implements Pos\_C\_XCDX{

public Pos\_C\_XCDX\_O pos\_C\_XCDX\_O;

public Pos\_C\_XCDX\_P pos\_C\_XCDX\_P;

public Pos\_C\_XCDX\_E pos\_C\_XCDX\_E;

public Pos\_CE\_XCDX() {

pos\_C\_XCDX\_O= new Pos\_CE\_XCDX\_O();

pos\_C\_XCDX\_P= new Pos\_CE\_XCDX\_P();

pos\_C\_XCDX\_E= new Pos\_CE\_XCDX\_E();

}

}

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

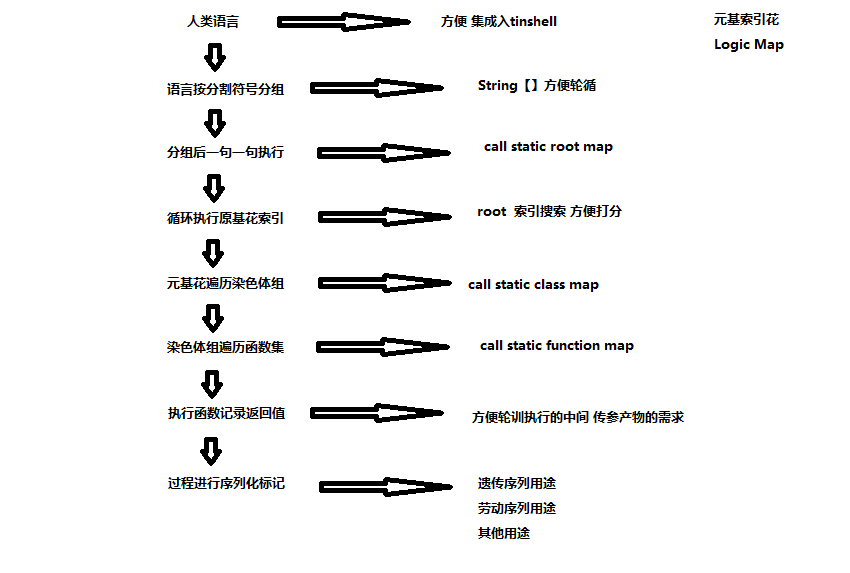
软件注意细节

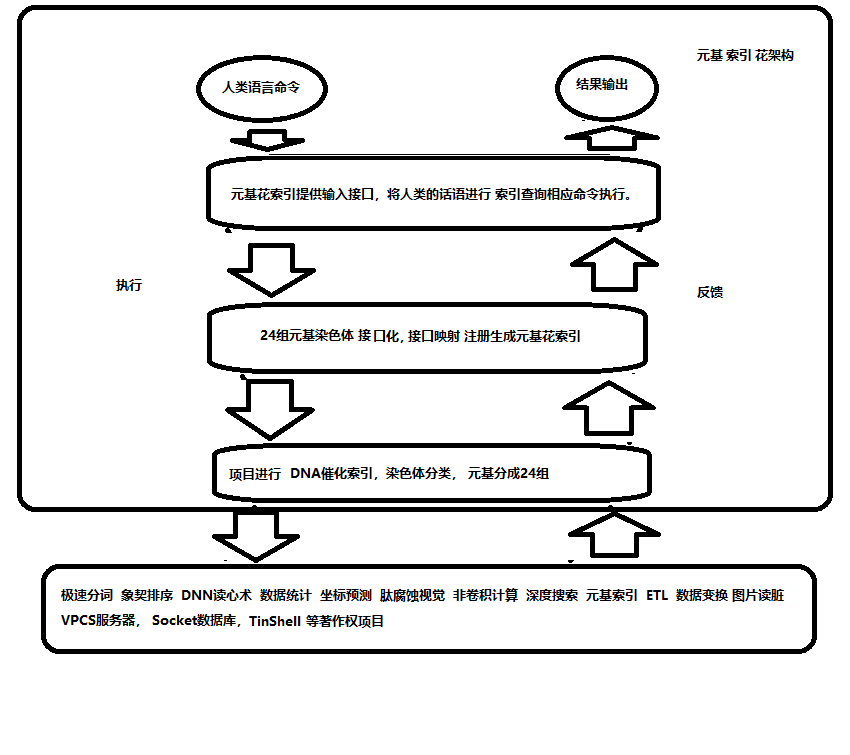
软件申明

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

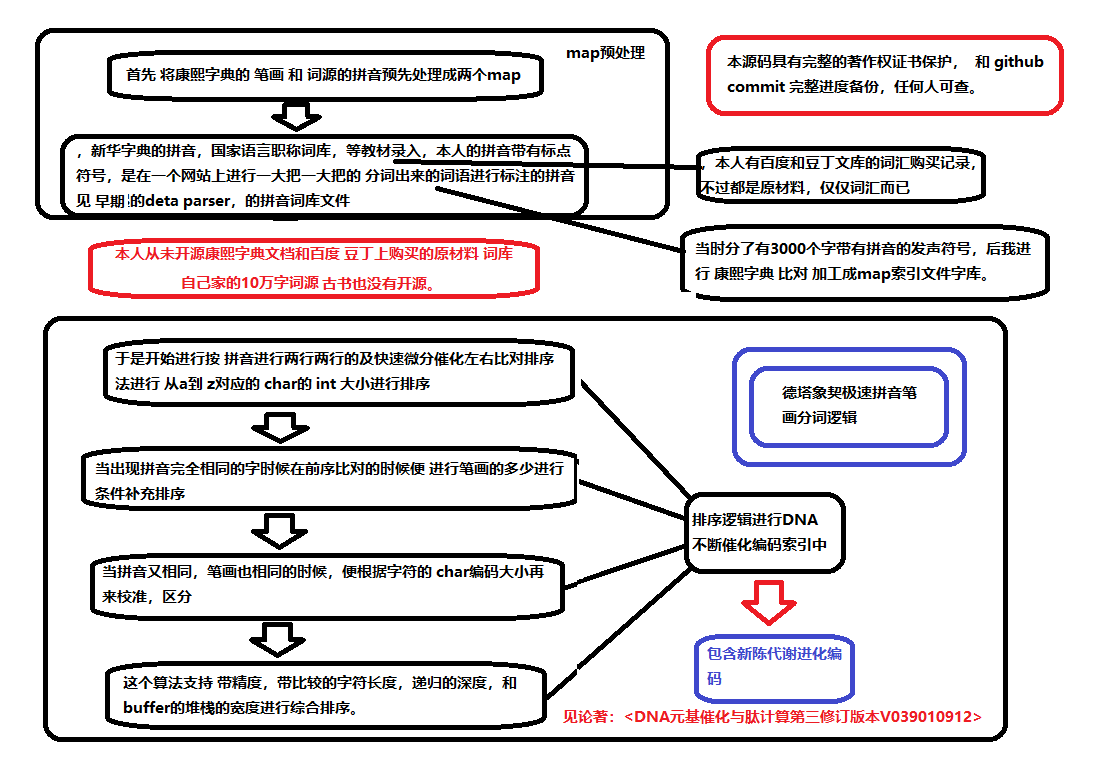
登陆token

肽展session注册

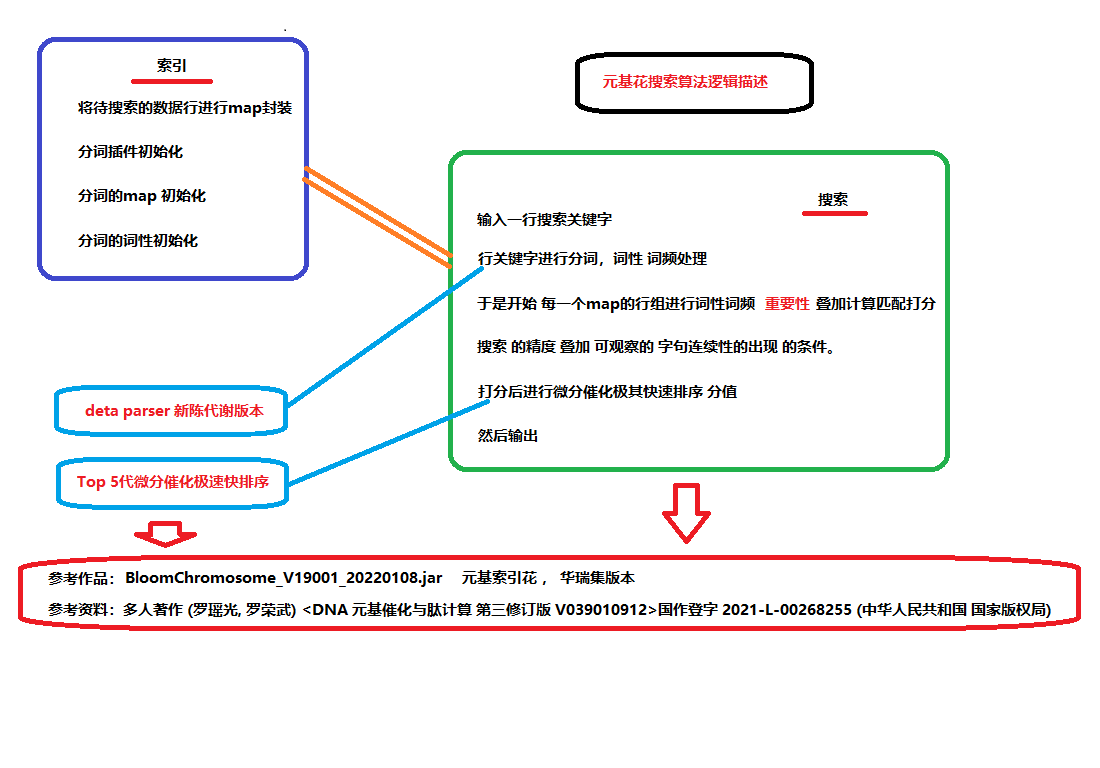
登陆状态验证

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