**Proposal 0.0.2**

**DNA元基催化 与 肽计算**

**第三修订版**

**数字生命启蒙**

**To Christina**

**罗瑶光**

**浏阳**

**20210720**

**目录**

**前序**

**第一章 德塔语言图灵工程**

第一节 研发说明

第二节 研发笔记

第三节 研发源码

第四节 DNA元基索引版本

**第二章 德塔JAVA数据分析**

第一节 研发说明

第二节 研发笔记

第三节 研发源码

第四节 DNA元基索引版本

**第三章 德塔ETL**

第一节 研发说明

第二节 研发笔记

第三节 研发源码

第四节 DNA元基索引版本

**第四章 德塔Socket流PLSQL数据库**

第一节 研发说明

第二节 研发笔记

第三节 研发源码

第四节 DNA元基索引版本

**第五章 德塔数据变换**

第一节 第一节 研发说明

第二节 研发笔记

第三节 研发源码

第四节 DNA元基索引版本

**第六章 德塔数据预测**

第一节 研发说明

第二节 研发笔记

第三节 研发源码

第四节 DNA元基索引版本

**第七章 DNA AOPM-VECS-IDUQ元基语义编码**

**第八章 DNA 计算元基肽展公式PDE 推导**

**第九章 DNA 与 PDE催化计算元基 XT-H解码**

**第十章DNA 视觉**

第一节DNA 视觉的动机

第二节 DNA 视觉的应用需求

第三节 DNA 视觉的具体描述

第四节 DNA 视觉的应用实现

**第十一章 DNA 卷积**

第一节 DNA 卷积的动机

第二节 DNA 卷积的应用需求

第三节 DNA 卷积的具体描述

第四节 DNA 卷积的应用实现

**第十二章 DNA ETL**

第一节 DNA ETL 的动机

第二节 DNA ETL 的应用需求

第三节 DNA ETL 的具体描述

第四节 DNA ETL 的应用实现

**第十三章 DNA 语料库**

第一节 DNA 语料库的动机

第二节 DNA 语料库的应用需求

第三节 DNA 语料库的具体描述

第四节 DNA 语料库的应用实现

**第十四章 DNA 加密**

第一节 DNA 加密的动机

第二节 DNA 加密的应用需求

第三节 DNA 加密的具体描述

第四节 DNA 加密的应用实现

**第十五章 DNA 数据库**

第一节 DNA 数据库的动机

第二节 DNA 数据库的应用需求

第三节 DNA 数据库的具体描述

第四节 DNA 数据库的应用实现

**第十六章 DNA 数术**

第一节 DNA 数术的动机

第二节 DNA 数术的应用需求

第三节 DNA 数术的具体描述

第四节 DNA 数术的应用实现

**第十七章 DNA 搜索**

第一节 DNA 搜索的动机

第二节 DNA 搜索的应用需求

第三节 DNA 搜索的具体描述

第四节 DNA 搜索的应用实现

**第十八章 DNA 筛选**

第一节 DNA 筛选的动机

第二节 DNA 筛选的应用需求

第三节 DNA 筛选的具体描述

第四节 DNA 筛选的应用实现

**第十九章DNA 元基染色体**

第一节 元基造字

第二节 最新笔记

第三节 图片识别

第四节 元基枝与元基花

尾章 时间戳笔记

**后序 RNA X-TH-DD元基芯片 启蒙**

**引用**

**第十九章DNA 元基染色体**

**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.ED

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

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.ED

//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.O

//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.OEU

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.

1. 最新笔记：

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。

1. **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.

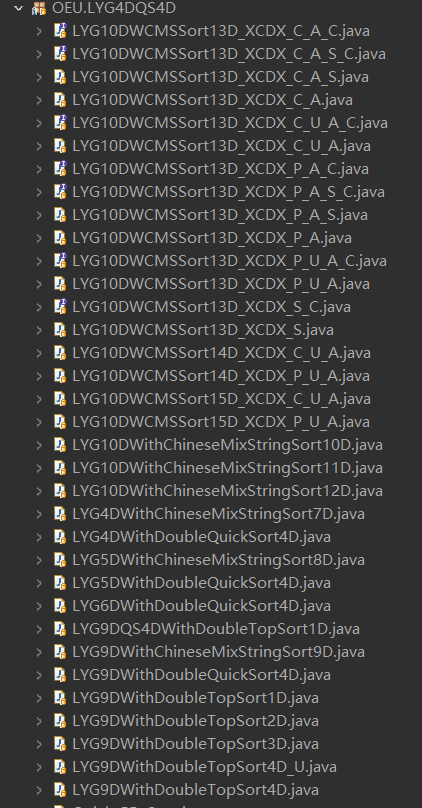
2 逐步的脱离养疗经华瑞集医学属性, 24组元基组 满足API的数据工程适用化,像java一样, 接口引用,参与计算.

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.

一些新的探索：

1 排序 新陈代谢



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");

}

}

2 笔画排序

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();

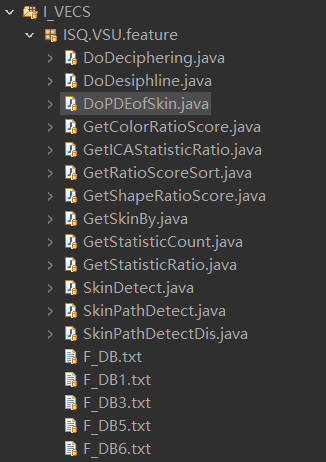
array[rightPosition]= x;

return rightPosition;

}

}

3 图片识别



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(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(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();

}

}

public int[][][] doPDEofSkin(int[][][] rgb, int ratio) {

int[][][] output= new int[3][][];

//int ratio= 50;

int[][] rp= new RangePDI().IPE(rgb[0], ratio);

int[][] gp= new RangePDI().IPE(rgb[1], ratio);

int[][] bp= new RangePDI().IPE(rgb[2], ratio);

output[0]= rp;

output[1]= gp;

output[2]= bp;

return output;

}

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;

}

}

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

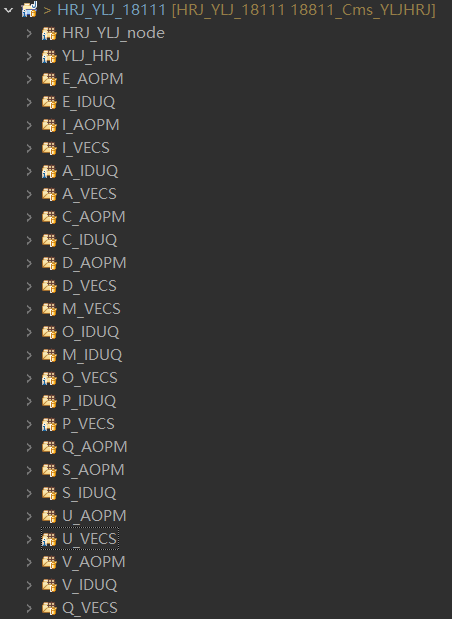
}

lineCount++;

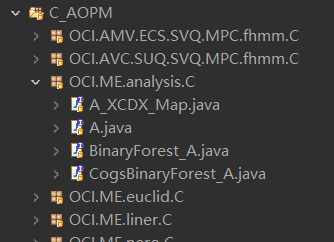
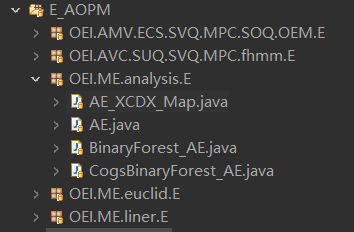
}

}

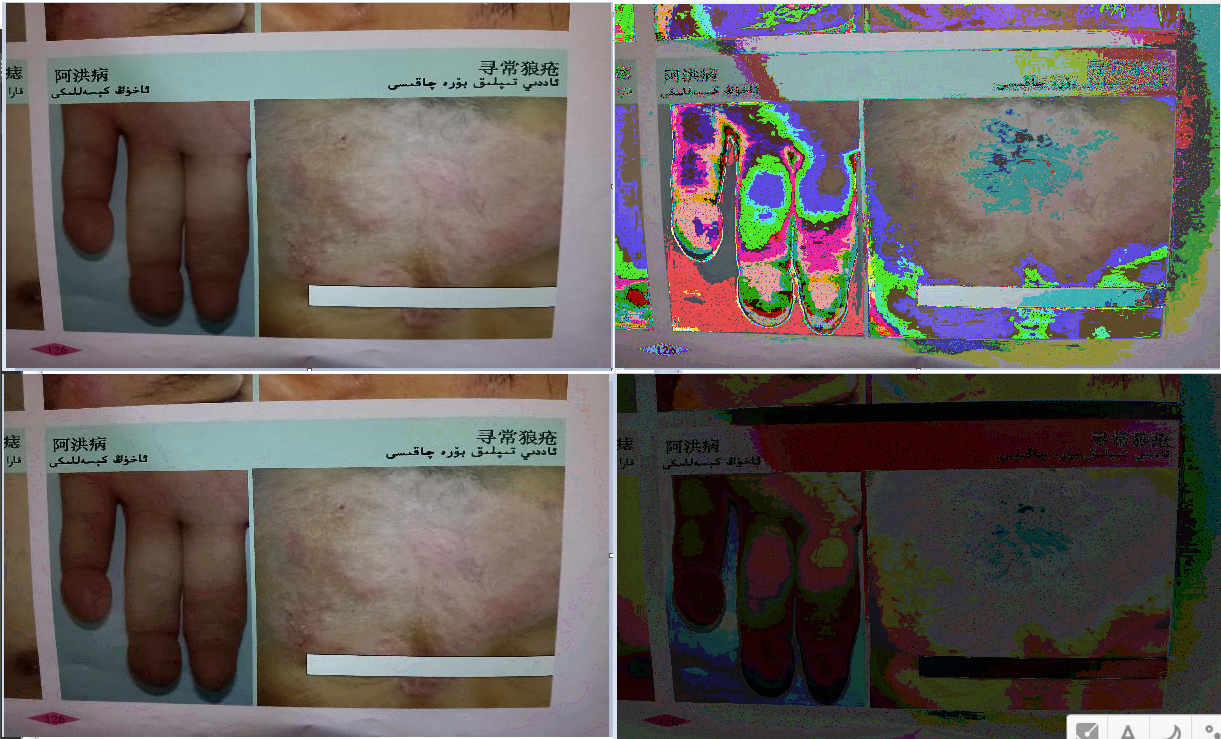
4元基枝与元基花



5 2 极速分词

6 四进制，十七进制肽腐蚀算法对比



左上原图 右上 十七进制八元腐蚀

左下十七进制四元腐蚀 右下 四进制四元腐蚀