参赛密码 _____

（由组委会填写）

“华为杯”第十三届全国研究生
数学建模竞赛

学　　校	中国科学院上海高等研究院
参赛队号	80184001
队员姓名	1. 杨诗蝶
	2. 曹双迎
	3. 姜祥玉

参赛密码 _____

（由组委会填写）

“华为杯”第十三届全国研究生数学建模竞赛

题 目 具有遗传性疾病和性状的遗传位点分析

摘 要：

本文从研究 DNA 中引起人类性状差异的位点入手，基于单位点关联性的假设条件，建立了以计数为手段的百分比差值统计模型。依据所建立模型，统计每一个位点不同性状表现型的碱基对的样本分布，并进一步分析不同性状的样本的碱基组合，得到了具有高度致病性的遗传位点和基因。

问题 1：考虑到不同位点之间的差异性和相同点、计算机对于数字编码的识别方式和碱基组合的等位因素，将原来位点的 16 种碱基组合方式简化为 10 种，利用 C 语言程序对碱基对进行十进制的数字编码。

问题 2：我们首先分析了 GWAS 中几种常用的模型和算法，总结并分析这几种模型的可靠性。其次根据单位点关联性的假设条件，以及每一个位点都只有 3 种碱基组合的情况，对有病和无病两组样本进行统计计数，得出每一个位点的 10 种碱基对在样本中的分布情况。逐个对 10 种碱基对样本数量的百分比进行求差运算，然后对得到的差值进行排序，最后对这个结果设定一个阈值，将高于该阈值的位点筛选出来设为有效位点，不断改变阈值以得到更精确的结果。再对有效位点进行卡方检验，结果发现位点 rs2273298、rs7543405 和 rs7368252 是致病位点的可能性高达 99.9%。

问题 3：该题是在 300 个基因中找到哪些基因包含问题二中确定的致病位点，因此可通过遍历问题三中所有的位点组合（基因）来对致病位点进行有效的筛选，进而发现基因 gene_55，gene_102 和 gene_265 就是致病基因。

问题 4：先统计分析 1000 组样本的性状分布情况，考虑到 10 种性状都表现为显性或隐形的样本占比情况，将具有多个显性基因的样本标记为 1，多个隐形基因的样本标记为 0，重新对样本计数。基于问题二的方法分析出有效位点为 rs7538876，rs4073710 和 rs351617。

关键词：单位点相关、差值百分比、计数统计、卡方检验

目录

1 问题背景和问题描述

人体的每条染色体携带一个 DNA 分子。 DNA 是由分别带有 A，T，C，G 四种碱基的脱氧核苷酸链接组成的双螺旋长链分子。在这条双螺旋的长链中，共有约 30 亿个碱基对。在组成 DNA 的碱基对中，有一些特定位置的碱基经常发生变异引起 DNA 的多样性，我们称之为位点。而基因则是 DNA 长链中有遗传效应的一些片段。

大量研究表明，人体的许多表型性状差异以及对药物和疾病的易感性等都可能与某些位点相关联，或和包含有多个位点的基因相关联[1-3]。因此，定位与性状或疾病相关联的位点在染色体或基因中的位置，能帮助研究人员了解性状和一些疾病的遗传机理，也能使人们对致病位点加以干预，防止一些遗传病的发生。

研究人员大都采用全基因组的方法来确定致病位点或致病基因，具体做法是：选取具有某种遗传病的人和健康的人作为样本，通常用 1 表示病人，0 表示健康者。 对每个样本，采用碱基(A，T，C，G)的编码方式来获取每个位点的信息(因为染色体具有双螺旋结构，所以用两个碱基的组合表示一个位点的信息)；在同一个位点有且只有三种不同编码方式。研究人员可以通过对样本的健康状况和位点编码的对比分析来确定致病位点，从而发现遗传病或性状的遗传机理。

本文通过对样本进行 DNA 位点的关联性分析建立数学模型。解决以下几个问题：

问题一、请用适当的方法，把 genotype.dat 中每个位点的碱基(A，T，C，G)编码方式转化成数值编码方式，便于进行数据分析。

问题二、根据 1000 个样本的 9445 个位点的编码信息和样本患有遗传疾病 A 的信息。设计或采用一个方法，找出某种疾病最有可能的一个或几个致病位点，并给出相关的理论依据。

问题三、同上题中的样本患有遗传疾病 A 的信息。现有 300 个基因，每个基因所包含一定数量的位点。由于可以把基因理解为若干个位点组成的集合，遗传疾病与基因的关联性可以由基因中包含的位点的全集或其子集合表现出来请找出与疾病最有可能相关的一个或几个基因，并说明理由。

问题四、在问题二中，已知 9445 个位点。在实际的研究中，科研人员往往把相关的性状或疾病看成一个整体，然后来探寻与它们相关的位点或基因。试根据这 1000 个样本点 10 种不同的性状信息及其 9445 个位点的编码信息，找出与 10 个性状有关联的位点。对你得到的结果都应该进行适当的统计分析和检验，从而从理论上说明你所发现的致病位点和基因的合理性。

2 模型假设

假设 1：样本的缺失率为 0，没有剔除。

假设 2：研究个体之间是相互独立的，不存在家族血缘关系。

假设 3：忽略次等位基因频率，所有基因都是等位的，即 AT=TA，CT=TC 等。

假设 4：不考虑人群分层和种族混杂。

假设 5：致病位点是单位点关联性的，只有一个位点起作用。

假设 6：十个关联性状是等位的。

3 符号说明

i	碱基对编号变量
j	位点编号变量
TR	采取的样本容量
DVALUE	百分比差阈值
R	总碱基对数量（10）
$M_{i,j}$	有病样本各位点各个碱基对出现的百分比差值
$N_{i,j}$	无病样本各位点各个碱基对出现的百分比差值
$m_{i,j}$	有病样本各位点各个碱基对出现的次数
$n_{i,j}$	无病样本各位点各个碱基对出现的次数
S_j	各位点各个碱基对的有病计数值与无病计数值差值绝对值的和
a	致病位点中 AA 碱基对有病样本数量
b	致病位点中 Aa 碱基对有病样本数量
c	致病位点中 aa 碱基对有病样本数量
d	致病位点中 AA 碱基对无病样本数量
e	致病位点中 Aa 碱基对无病样本数量
f	致病位点中 aa 碱基对无病样本数量
A	检验实际频数
H0	假设条件
E	基于假设 H0 计算出的期望频数
T	基于期望 E 的理论频数
n	总的样本量
n_{case}	有病样本量
$n_{control}$	无病样本量
q	致病位点的个数

4 模型建立与求解

4.1 问题一

要求我们将每个位点碱基原本为(A，T，C，G) 编码方式转化成数值编码方式。我们采用了 C 语言的程序编码将本题中所含有的碱基对进行了编码。由于我们假设所有位点都是等位的，因此原本编码方式不同的 AC 和 CA，AT 和 TA，AG 和 GA，CG 和 GC，GT 和 TG，CT 和 TC，经过等位变换，可以写成 AC=CA，AG=GA，AT=TA，CG=GC，CT=TC，TG=GT 原本总共 4*4=16 种编码方式少去了 6 种，剩下 10 种，我们可以选择 10 进制的数值编码方式。具体编码方式如下：

AA=0，AC=CA=1，AG=GA=2，

AT=TA=3，CC=4，CG=GC=5，

CT=TC=6，GG=7，TG=GT=8，TT=9

如表 1 所示：

	A	C	G	T
A	0	1	2	3
C	1	4	5	6
G	2	5	7	8
T	3	6	8	9

表 1. 碱基对对应编码

利用 C 语言程序，将 ATCG 之外的字符用 X 进行输出。对 1000 个样本点，9000 多个位点表示成如下的数组 uh_rs[x][y]，h_rs[x][y]。其中 uh_rs、h_rs 分别表示有病样本和无病样本的数组名称。而 x，y 分别表示为样本的序号和位点序号。通过循环语句将 1000 个样本和 9545 个位点的碱基编码进行数字编码，存储到两个数组当中，过程图如图 1 所示，C 语言程序见附录 1。

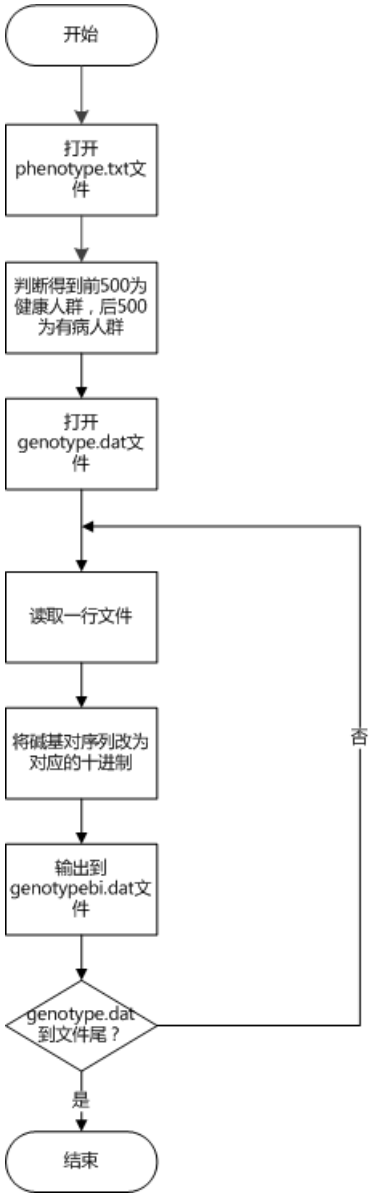

图 1. 碱基对数值编码流程图

4.2 问题二

4.2.1 几种常用的 GWAS 数学模型介绍：

模型一：随机森林方法识别致病 SNP 位点[4]

随机森林是一种集成学习算法，于 2001 年有 LeoBreiman 发明，简要思想是利用 Bootstrap 采样得到的多个数据集，训练多个分类器，然后将多个决策树的记过进行富欧式分类或者取均值。

高维的 SNP 数据需要高效的方法来识别致病 SNP 位点，随机森林的变量重

要性值被用来进行 SNP 数据的筛选和识别，一方面通过特征选择来降低维度，避免过拟合，提高数据的准确性，另一方面希望通过特征选择或者特征排名来识别致病基因的特征，也就是 SNP 位点。

早期的研究表明，随机森林特别适合于单个位点时边缘效应弱，但是和其他位点一起时交互作用强的情况。相关研究表明，SNP 的数目经常限制在一个较小的范围内，对数据的真实性有一定的局限性。

最近的研究虽然经随机森林应用到高维的 SNP 数据上，成千上万的 SNP 位点并且考虑到关联的 SNP 位点的影响。通过提出一种重要的测度概念来用于 GWAS 中进行特征选择。在较低的维度下随机森林成功的识别了交互作用，但是当 SNP 个数增加时，识别 SNP 交互作用的能力迅速下降。因此随机森林在高维数据下变量重要性值更适合检测单个位点的边缘效应。而无法检测到交互作用效应。

本次试验数据中共有 9000 多个位点，如果仅仅进行随机森林的单个位点检测来判断致病位点。不仅无法从少数样本中分析出主要特征，而且对于计算机来说算法过于复杂，不能进行高效率计算。

模型二：Relief 方法

Relief 由 Kira and Rendell 于 1994 年发明，是一种监督学习算法，通过对每个样本的寻找最邻近的样本来更新特征权重，是一种特征选择方法。Relief 在更新每个特征权重时，考虑到整个特征空间，具体做法是如果考虑每个样本 X 的重要性特征是被认为是重要特征，对样本有很强的区分度，会给这些特征赋一个很高的权重。

每一次迭代，每个特征的权重都会得到更新，并且归一化到-1 和 1 之间，输出是每个特征的权值，通过提出一个关联阈值 a，可以通过选择阈值>a 的权重对应的特征进行过滤特征，得到重要特征。

然而 ReliefF 本身对噪声比较敏感，所以需要结合其他方法来使用，ReliefF 的优势在于复杂度低，计算速度快。但是由于 ReliefF 算法采用随机选择样本，其采样个数 m 的大小对各特征权值有一定的影响，如果 m 取值太小会导致每组采样存在一定的差别，计算出的权值可能有较大的区别不能较好的反应特征权值；如果 m 取值较大，则明显增加算法的复杂度。随机选择样本会使小类别被选中进行权值计算的概率小，有时可能会被完全忽略。在无小类别参与的情形下，得到的特征权值是不科学的，从而对聚类的精度产生一定的负面影响。

样本中共有 9000 多个 SNP 位点，1000 多个样本，其中有病无病样本各占 50%。如果单纯的只用 ReliefF 是不可靠的。

模型三：传统的 GWAS 方法

目前的 GWAS 多采用两个阶段的设计：首先采用覆盖整个基因组的高通量的 SNP 分型芯片对一批样品进行分型和分析，然后筛选出最显著差异的 SNP 进行验证。GWAS 两个阶段研究设计减少了基因分型的工作量和花费，同时通过重复试验降低了研究的假阳性率。然而这两阶段的研究设计却存在着另一个问题：第一阶段通常在较少的样本（多为 1000 病例—1000 对照组）对全基因数量庞大的 SNPs

进行分析（可达 100 多万），发现的易感位点基本都是常见的（MAF>0.2）且具有中度效力（OR>=1.2）的变异；而对于频率低（MAF<0.5）、效力弱（OR 接近于 1）的 SNPs，由于统计把握度较低很难被检查出来，因此没有足够的检验效能发现所有可能与疾病关联的 SNPs。因此，为了发现更多的易感位点，目前常采用的方法就是在扩大 GWAS 样本量的同时，放宽第一阶段筛选 SNPs 的标准。扩大验证范围。

本实验中的 9000 多个位点 1000 多组样本，远远小于满足 GWAS 精度要求所需的量。因此我们重新建立了以下模型。

4.2.2 模型建立

我们假设致病位点是单位点关联性的，只有一个位点起作用。因为有 9000 多个位点，我们不能同时考虑所有的情况。由于不同位点碱基是等位的，理想情况下我们只考虑一个位点的情况，将有病没病的人群分别按照碱基的组合分为 3 种（由于程序要求只能先设定 10 种），总共是 6 组。其他位点与这个位点不同之处只是样本分布有所变化，各个样本的计数百分比有所不同。

利用统计学的方法，参照 GWAS 单阶段设计方法[5]，采用基于计数统计的反复抽样百分比差值运算，测试单个位点在有病样本和无病样本之间的差异，对差值进行排序，得到致病位点 SNP 的致病相关性，进而得到显著的致病 SNP 位点。依据该问题采用的模型，利用提供的 1000 个样本数据确定该疾病位点。

1000 个样本中有 500 个有病样本和 500 个无病样本。选取多次抽样的样本数量 TR（有病样本和无病样本各取 TR）分别为 300、400 和 500。对于每一次抽样统计，将有病样本的某一位点的三种碱基对出现的次数比例与对应的无病样本的同一位点三种碱基对的比例做差值。对于 9445 个位点，大部分有病样本和无病样本碱基对比例的差值都是很小的，这部分位点即可看作与该疾病无关。我们可以给差值设定一个阈值 DVALUE，当位点中某种碱基对比例的差值大于阈值时，该位点就被筛选出来，这部分位点与该疾病相关性很大。阈值的初次设定为任意的，根据筛选出的位点结果可做相应调整。通过三次不同样本容量的统计结果对比，最终确定哪几个位点与该疾病相关。

根据问题一的编码方式，对于 9445 个位点，每个位点都有 10 种可能的碱基对。实际上与每个位点对应的碱基对只有 3 种，考虑到各个样本在相同位点的碱基可能不同，以上处理方式可以使计算化简。

我们只考虑相对应的具体实现过程，建立了两种模型。一种是对单个位点内的十种碱基对逐个计算百分比差值，只要有一种的有无病百分比差值达到设定的阈值即被筛选出来；另一种是将位点内的十种碱基对的有无病计数差值绝对值之和计算出来，按照从高到低的顺序排列，最终选取差值绝对值之和大的几个位点作为致病位点。第一种模型采用多次抽样，用 C 语言实现；第二种模型将 500 组数据全部利用上，只做了一次统计，用 excel 实现。

4.2.2.1 模型一

对于两组样本，在容量为 TR 时，先处理有病样本，计算位点 1 的十种碱基对出现的次数 mi，1（i=0，1，2…9），然后得到位点 1 的十种碱基对出现的百分比 Mi，1=mi，1/TR×100%，（i=0，1，2…9），对于其余 9444 个位点做同样的处理。位点 rsj 的第 i 个碱基对出现次数百分比为 Mi，j，最终有病样本的结果形式如表 2 所示。无病样本的处理方式同有病样本的类似，位点 rsj 的第 i 个碱基对出现次数百分比为 Ni，j，处理结果的方式如表 3 所示。

位点 / 碱基对	rs1	rs2	…	rs_j	…	rs9444	rs9445
0	$M_{0,1}$	$M_{0,2}$	…	$M_{0,j}$	…	$M_{0,9444}$	$M_{0,9445}$
1	$M_{1,1}$	$M_{1,2}$	…	$M_{1,j}$	…	$M_{1,9444}$	$M_{1,9445}$
2							
3	⋮	⋮	…	⋮	…	⋮	⋮
4							
5	$M_{i,1}$	$M_{i,2}$	…	$M_{i,j}$	…	$M_{i,9444}$	$M_{i,9445}$
6	⋮	⋮	…	⋮	…	⋮	⋮
7							
8	$M_{8,1}$	$M_{8,2}$	…	$M_{8,j}$	…	$M_{8,9444}$	$M_{8,9445}$
9	$M_{9,1}$	$M_{9,2}$	…	$M_{9,j}$	…	$M_{9,9444}$	$M_{9,9445}$

表 2. 有病样本中各位点碱基对出现次数所占百分比

位点 / 碱基对	rs1	rs2	…	rs_j	…	rs9444	rs9445
0	$N_{0,1}$	$N_{0,2}$	…	$N_{0,j}$	…	$N_{0,9444}$	$N_{0,9445}$
1	$N_{1,1}$	$N_{1,2}$	…	$N_{1,j}$	…	$N_{1,9444}$	$N_{1,9445}$
2							
3	⋮	⋮	…	⋮	…	⋮	⋮
4							
5	$N_{i,1}$	$N_{i,2}$	…	$N_{i,j}$	…	$N_{i,9444}$	$N_{i,9445}$
6	⋮	⋮	…	⋮	…	⋮	⋮
7							
8	$N_{8,1}$	$N_{8,2}$	…	$N_{8,j}$	…	$N_{8,9444}$	$N_{8,9445}$
9	$N_{9,1}$	$N_{9,2}$	…	$N_{9,j}$	…	$N_{9,9444}$	$N_{9,9445}$

表 3. 无病样本中各位点碱基对出现次数所占百分比

位点的筛选方式用数学语言表示为：

对于$\forall rs_j$（$j = 1,2,3\cdots 9444,9445$），如果∃ i（$i = 0,1,2\cdots,9$）使得$\left|M_{i,j} - N_{i,j}\right| > DVALUE$，则提取该位点$rs_j$作为疾病相关位点。

DVALUE 初次试验值为 20%。

接下来通过 C 语言程序实现上述过程。图 2a 是模型一的流程图，对于每一个位点分别判断其是否为致病位点，该过程循环 9445 次。图 2b 是图 2a 的子程序流程图。C 语言程序见附录 2。

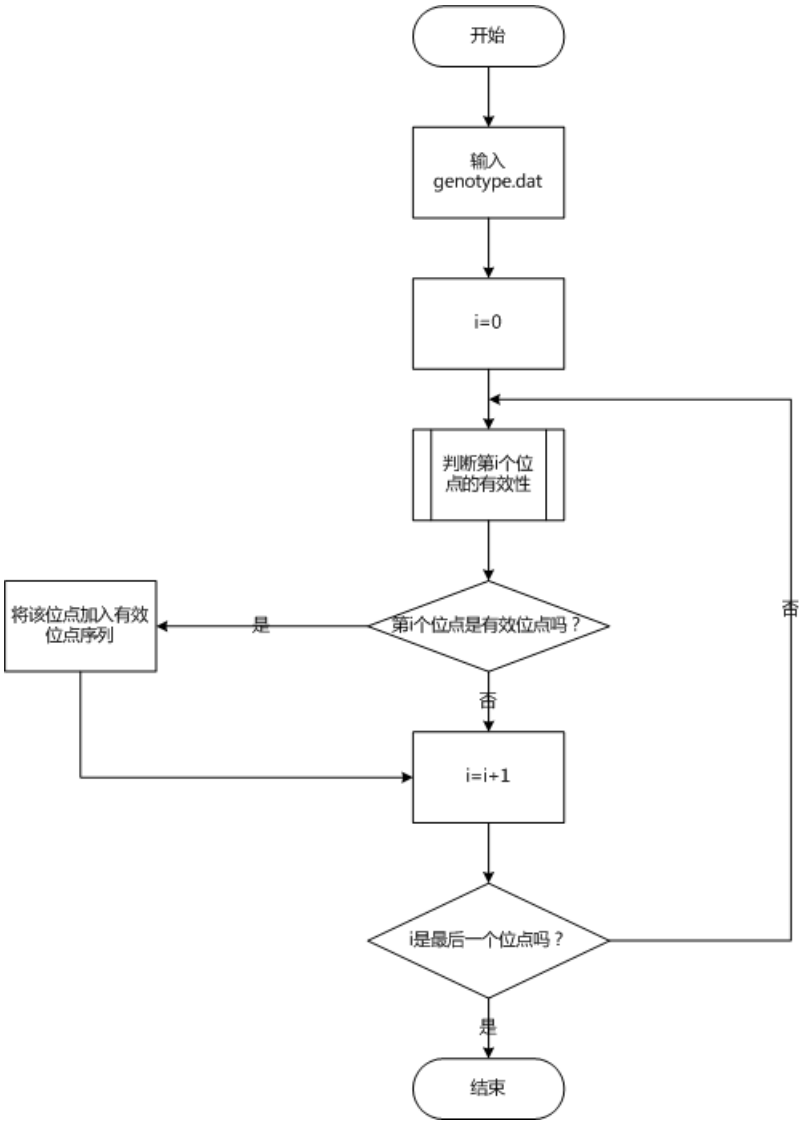

图 2a. 求所有致病位点的流程图

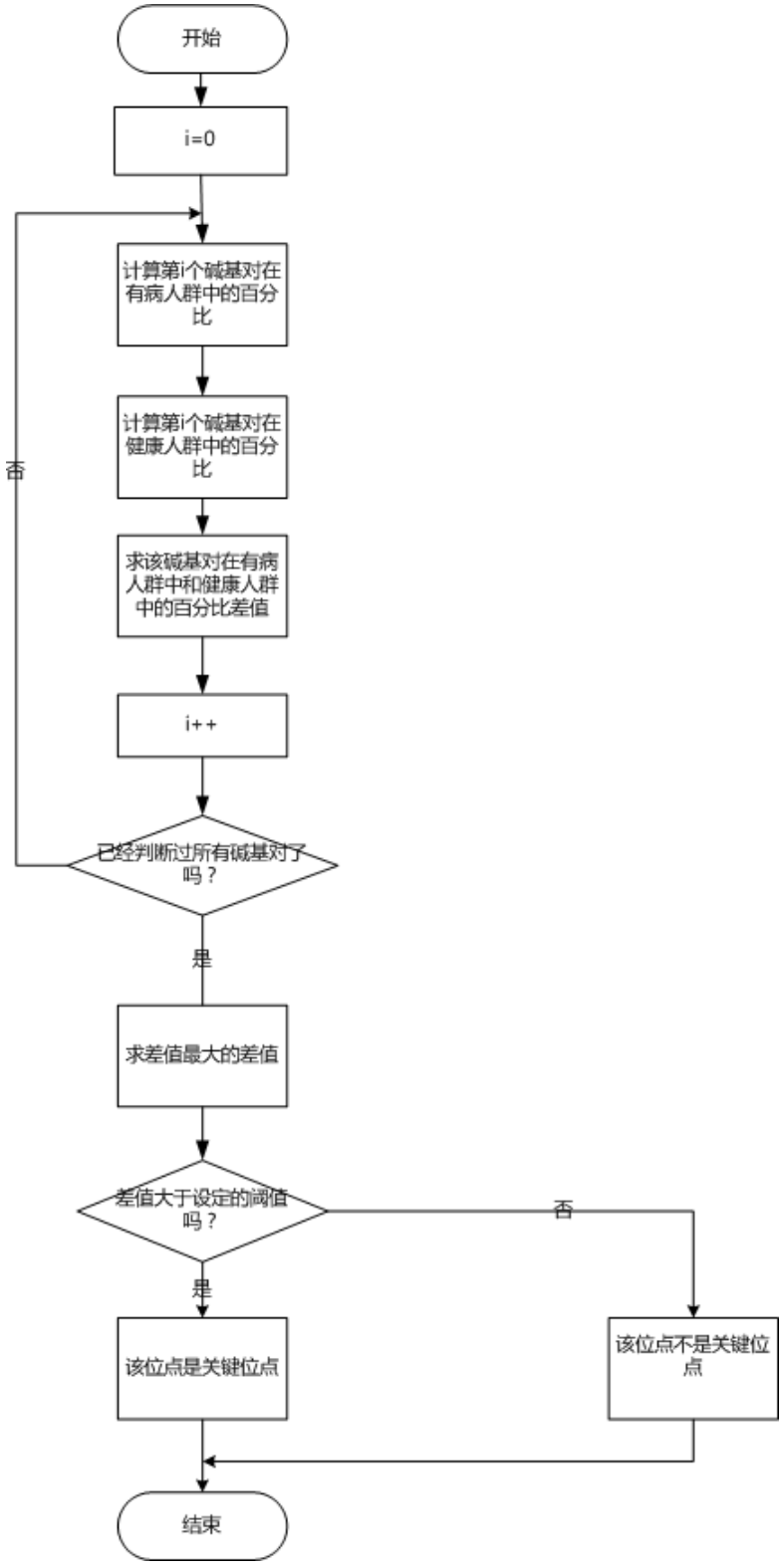

图 2b. 判断单个位点是否为致病位点流程图

当 DVALUE 为 20%、样本容量 TR 为 500 时，所有位点均不满足条件，因此第二次试验将 DVALUE 设为 10%。当样本容量 TR 分别为 300、400 和 500 时的数据提取结果如表 4。

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs7543405	0.173333	rs10903032	0.113333	rs11260562	0.103333
rs2273298	0.16	rs12752833	0.113333	rs10909901	0.103333
rs1541318	0.14	rs7514514	0.113333	rs7541288	0.103333
rs1849943	0.14	rs1317017	0.11	rs2076883	0.103333
rs10779765	0.14	rs10864304	0.11	rs12121369	0.103333
rs2244300	0.136667	rs6682376	0.11	rs10907214	0.103333
rs2236817	0.133333	rs198388	0.11	rs7522344	0.103333
rs2807345	0.133333	rs1033452	0.11	rs2281303	0.103333
rs4531246	0.126667	rs6690160	0.11	rs3923058	0.103333
rs2095518	0.126667	rs4278387	0.11	rs4908440	0.103333
rs1014988	0.123333	rs7533344	0.11	rs4908796	0.103333
rs2411894	0.123333	rs10917047	0.11	rs7417097	0.103333
rs11583605	0.123333	rs2473246	0.11	rs17032875	0.103333
rs5022242	0.123333	rs2182703	0.11	rs4845953	0.103333
rs4415555	0.123333	rs2811963	0.11	rs4845907	0.103333
rs4646092	0.123333	rs10903122	0.11	rs10927904	0.103333
rs1780316	0.123333	rs1702311	0.11	rs6697531	0.103333
rs17356177	0.123333	rs271321	0.11	rs10927982	0.103333
rs35047308	0.123333	rs6694632	0.106667	rs1883567	0.103333
rs28603108	0.12	rs10907194	0.106667	rs2229487	0.103333
rs2651935	0.12	rs10492941	0.106667	rs2473247	0.103333
rs12036552	0.12	rs1333190	0.106667	rs4649168	0.103333
rs6683017	0.12	rs3819967	0.106667	rs536577	0.103333
rs200447	0.12	rs4391636	0.106667	rs11589294	0.103333
rs1133398	0.12	rs623974	0.106667	rs17184982	0.103333
rs590368	0.12	rs11120917	0.106667	rs6598858	0.103333
rs3765380	0.12	rs226242	0.106667	rs7536088	0.103333
rs877648	0.12	rs4846195	0.106667	rs11247723	0.103333
rs1257163	0.12	rs586223	0.106667	rs1372372	0.103333
rs4648537	0.116667	rs11587326	0.106667	rs675696	0.103333
rs7527078	0.116667	rs1074078	0.106667	rs7520877	0.103333
rs10462021	0.116667	rs7513972	0.106667	rs399377	0.1
rs914994	0.116667	rs4845892	0.106667	rs6702564	0.1
rs7546786	0.116667	rs4240904	0.106667	rs7368252	0.1
rs6429695	0.116667	rs804136	0.106667	rs6698901	0.1
rs12727433	0.116667	rs10754876	0.106667	rs763161	0.1
rs3806310	0.116667	rs1883761	0.106667	rs707472	0.1
rs11247916	0.116667	rs2092324	0.106667	rs2982376	0.1
rs1372378	0.116667	rs12725198	0.106667	rs4661661	0.1

rs1868302	0.116667	rs17416878	0.106667	rs1009806	0.1
rs12145450	0.116667	rs2143810	0.106667	rs1569583	0.1
rs946758	0.113333	rs697760	0.106667	rs11580218	0.1
rs880801	0.113333	rs4920489	0.106667	rs6689530	0.1
rs164771	0.113333	rs2816065	0.106667	rs7553231	0.1
rs3010876	0.113333	rs2473808	0.106667	rs15045	0.1
rs7543486	0.113333	rs2301475	0.106667	rs1135251	0.1
rs2473356	0.113333	rs1256342	0.106667	rs11247771	0.1
rs35365738	0.113333	rs4648878	0.106667	rs11586865	0.1
rs873319	0.113333	rs11581460	0.106667	rs271383	0.1
rs648305	0.113333	rs4654355	0.106667	rs12142439	0.1
rs230182	0.113333	rs9286945	0.106667	rs12725881	0.1
rs12029258	0.113333	rs12036216	0.103333		

表 4a. 样本容量 300、百分比差阈值 10%的筛选结果
（按百分比差降序排列，总共获取的数据点为 155 个）

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs2273298	0.1675	rs648305	0.11	rs15045	0.105
rs10779765	0.14	rs1883567	0.11	rs1257163	0.105
rs7543405	0.135	rs4649168	0.11	rs7555715	0.105
rs4646092	0.1275	rs1541318	0.1075	rs1188403	0.105
rs2236817	0.12	rs7368252	0.1075	rs3765380	0.1025
rs880801	0.1175	rs11587326	0.1075	rs2092324	0.1025
rs3806310	0.115	rs4073574	0.1075	rs1138333	0.1025
rs2807345	0.115	rs4845881	0.1075	rs9659647	0.1025
rs12145450	0.115	rs1262461	0.1075	rs9286945	0.1025
rs2477777	0.1125	rs1148455	0.1075	rs1201394	0.1025
rs1033867	0.1125	rs6661262	0.1075	rs12095517	0.1
rs28603108	0.11	rs7546786	0.1075	rs705579	0.1
rs4845892	0.11	rs12097284	0.1075	rs2473345	0.1
rs590368	0.11	rs10903032	0.1075	rs1924270	0.1
rs3010876	0.11	rs12133956	0.105		
rs2143810	0.11	rs1849943	0.105		

表 4b. 样本容量 400、百分比差阈值 10%的筛选结果
（按百分比差降序排列，总共获取的数据点为 46 个）

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs2273298	0.166	rs2143810	0.11	rs880801	0.104
rs7543405	0.13	rs12145450	0.11	rs2807345	0.104
rs4391636	0.114	rs1541318	0.108	rs9659647	0.104

rs7368252	0.112	rs3013045	0.108	rs4646092	0.102
rs1883567	0.112	rs932372	0.108		

表 4c. 样本容量 500、百分比差阈值 10%的筛选结果
（按百分比差降序排列，总共获取的数据点为 14 个）

通过以上三组统计结果对比可以发现，在给定某一百分比差阈值 DVALUE 的情况下，样本容量越大，筛选出来的结果越少。样本容量较小时，结果偶然性较大，结果的误差较大。当数据量较大时，各个位点的百分比差值越精细。而且还可以发现这三组结果之间并没有纯粹的包含关系，即表 4c 不完全包含于表 4b 且表 4b 也不完全包含于表 4a。

观察表 4c 的结果，可以看出前两个位点的百分比差值显著高于其他位点的。为了进一步筛选出相关性更高的位点，我们将百分比差阈值改为 12%，得到的结果明显减少了，如表 5 所示。

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs7543405	0.173333	rs7543486	0.126667	rs7536088	0.123333
rs2273298	0.16	rs2095518	0.126667	rs35047308	0.123333
rs1541318	0.14	rs12145450	0.126667	rs12036216	0.12
rs1849943	0.14	rs1014988	0.123333	rs28603108	0.12
rs10779765	0.14	rs2411894	0.123333	rs2651935	0.12
rs6429695	0.136667	rs11583605	0.123333	rs1333190	0.12
rs2244300	0.136667	rs5022242	0.123333	rs12036552	0.12
rs2236817	0.133333	rs4415555	0.123333	rs6683017	0.12
rs2807345	0.133333	rs4646092	0.123333	rs200447	0.12
rs6682376	0.13	rs1780316	0.123333	rs590368	0.12
rs7533344	0.13	rs17356177	0.123333	rs877648	0.12
rs4531246	0.126667	rs15045	0.123333	rs1257163	0.12

表 5a. 样本容量 300、百分比差阈值 12%的筛选结果
（按百分比差降序排列，总共获取的数据点为 36 个）

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs2273298	0.1675	rs7555715	0.1375	rs4646092	0.1275
rs10779765	0.14	rs7543405	0.135	rs28603108	0.125

表 5b. 样本容量 400、百分比差阈值 12%的筛选结果
（按百分比差降序排列，总共获取的数据点为 6 个）

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs2273298	0.166	rs7543405	0.13	rs7368252	0.11.6

表 5c. 样本容量 500、百分比差阈值 11.5%的筛选结果
（按百分比差降序排列，总共获取的数据点为 2 个）

表 5 中的结果都属于表 4 结果的前几位。需要注意的是，表 5 中 300、400 个样本的百分比差值结果与表 4 中的有所差别，这是由计算精度导致的。

通过以上分析，最终确定的与致病最相关的位点为 rs2273298。

4.2.2.2 模型二

我们利用另外一种筛选方式做了类似的尝试。这次的筛选方式是直接利用出现次数，而不是次数百分比。对各位点各个碱基对的有病计数值与无病计数值差值绝对值求和，记为 Sj，并且按由大到小的顺序排列，即对于 $\forall rs_j$（$j = 1,2,3\cdots 9444,9445$），求出 Sj=$\sum_{i}^{500}\left|m_{i,j} - n_{i,j}\right|$并排序。表 6 列出了按差值绝对值之和降序排列后，前几个位点的十个碱基对有病计数值与无病计数值。排在前三位的位点分别是 rs2273298、rs7543405 和 rs7368252，该结果与模型一的结果一致。

之所以在此选择前三个位点是因为我们并不确定这一疾病究竟与多少个位点有关，可能是 2 个或 3 个，也有可能是 4 个或者更多。我们先选择其中三个进行检验，并且分析这三个位点之间的联系来确定到底是 2 个还是 3 个位点决定疾病，或者是否要再加上后边第 4、第 5、…、第 k 个位点。

位点名	rs2273298	rs7543405	rs7368252	rs4391636	rs4646092	rs1883567	rs2143810	rs12145450
无病组 碱基对								
0	305	0	0	0	45	0	0	0
1	0	0	0	0	0	0	0	0
2	161	0	0	0	183	0	0	0
3	0	0	0	0	0	0	0	0
4	0	136	311	53	0	0	0	111
5	0	0	0	0	0	0	0	0
6	0	278	159	243	0	0	0	274
7	34	0	0	0	272	179	70	0
8	0	0	0	0	0	209	256	0
9	0	86	30	204	0	112	174	115
有病组 碱基对								
0	222	0	0	0	50	0	0	0
1	0	0	0	0	0	0	0	0
2	218	0	0	0	234	0	0	0
3	0	0	0	0	0	0	0	0
4	0	175	255	34	0	0	0	100
5	0	0	0	0	0	0	0	0
6	0	213	217	205	0	0	0	230

7	60	0	0	0	216	139	83	0
8	0	0	0	0	0	265	201	0
9	0	112	28	261	0	96	216	170
差值绝对值之和	**166**	**130**	**116**	**114**	**112**	**112**	**110**	**110**

表 6. 按差值绝对值之和降序排列后前几个位点的十个碱基对有病计数值与无病计数值差值

4.2.2.3 两种模型对比

两种模型的筛选方式略微不同，各有优劣。第一种模型可以使计算次数相对减少，但是由于计算百分比时用到除法，会使计算结果精确度降低；但是由于是用 C 语言编程实现，便于处理后续题目的重复性操作。第二种模型直接利用计数结果，误差较小，而且便于用数理统计的检验方法进行检验，然而 excel 数据处理的重复性比较局限。实际上，第二种模型可以看作对第一种模型的验证与补充。

4.2.3 模型检验

利用上述模型二的数据，我们对选取的前三个位点进行了χ^2检验（又称卡方检验），用来验证所选位点是否合适。

χ^2检验的基本思想：

检验实际频数（A）和理论频数的（T）的差别是否由抽样误差所引起。也就是有样本率（或者样本构成比）来推断总体率（或者构成比），以及多个样本率（或多个构成比）比较的卡方检验以及分类资料的相关分析等。首先假设 H0 成立，基于此前提计算出χ^2值，它表示观察值与理论值之间的偏离程度。根据χ^2分布及自由度可以确定在 H0 假设成立的情况下获得当前统计量及更极端情况的概率 P。如果 P 值很小，说明观察值与理论值偏离程度太大，应当拒绝无效假设，表示比较资料之间有显著差异；否则就不能拒绝无效假设，尚不能认为样本所代表的实际情况和理论假设有差别。

卡方值的计算与意义：

χ^2值表示观察值与理论值之间的偏离程度。计算这种偏离程度的基本思路如下。

(1)设 A 代表某个类别的观察频数，E 代表基于 H0 计算出的期望频数，A 与 E 之差称为残差。

(2)显然，残差可以表示某一个类别观察值和理论值的偏离程度，但如果将残差简单相加以表示各类别观察频数与期望频数的差别，则有一定的不足之处。

因为残差有正有负，相加后会彼此抵消，总和仍然为 0，为此可以将残差平方后求和。

(3)另一方面，残差大小是一个相对的概念，相对于期望频数为 10 时，期望频数为 20 的残差非常大，但相对于期望频数为 1 000 时 20 的残差就很小了。考虑到这一点，人们又将残差平方除以期望频数再求和，以估计观察频数与期望频数的差别。

进行上述操作之后，就得到了常用的χ^2统计量，由于它最初是由英国统计学家 Karl Pearson 在 1900 年首次提出的，因此也称之为 Pearson χ^2，其计算公式为

$$\chi^2 = \sum \frac{(A-E)^2}{E} = \sum_{i=1}^{k} \frac{(A_i - E_i)^2}{E_i} = \sum_{i=1}^{k} \frac{(A_i - np_i)^2}{np_i}$$

（其中 i=1，2，3，…，k）

其中，Ai 为 i 水平的观察频数，Ei 为 i 水平的期望频数，n 为总频数，pi 为 i 水平的期望频率。i 水平的期望频数 Ei 等于总频数 n×i 水平的期望概率 pi，k 为单元格数。当 n 比较大时，χ^2统计量近似服从 k-1(计算 Ei 时用到的参数个数)个自由度的卡方分布。

由卡方的计算公式可知，当观察频数与期望频数完全一致时，χ^2值为 0；观察频数与期望频数越接近，两者之间的差异越小，χ^2值越小；反之，观察频数与期望频数差别越大，两者之间的差异越大，χ^2值越大。换言之，大的χ^2值表明观察频数远离期望频数，即表明远离假设。小的χ^2值表明观察频数接近期望频数，接近假设。因此，χ^2是观察频数与期望频数之间距离的一种度量指标，也是假设成立与否的度量指标。如果χ^2值“小”，研究者就倾向于不拒绝 H0；如果χ^2值大，就倾向于拒绝 H0。至于χ^2在每个具体研究中究竟要大到什么程度才能拒绝 H0，则要借助于卡方分布求出所对应的 P 值来确定。

在本次抽样调查当中，总共有 1000 多组样本，我们选择了 3 个位点进行χ^2检验。将已知的 SNP 数据分成有病样本和无病样本，将有病样本（Case）和无病样本（Controls）的数目分别记为$n_{control}$和n_{case}，对应的数据集类别分别标记为 1（病例样本）和 0（无病样本）。对于每个 SNP 位点，双等位基因可以将原来的 ATCG 这 4 个碱基对总共有 3 种组合，表示为 A，a，双等位基因碱基对类型样本统计

结果示例如表 7 所示。

	AA	Aa	aa	总计
有病样本	a	b	c	n_{case}
无病样本	d	e	f	$n_{control}$
总计	n_{AA}=a+d	n_{Aa}=b+e	n_{aa}=c+f	n

表 7. 双等位基因碱基对类型样本统计结果示例

根据表 7，可以通过统计学上自由度为（3-1）*（2-1）=2 的卡方检验（χ^2）来检测是否对应的 SNP 位点与疾病状态是否有关联，方法是检测表现型在有病样本和无病样本的频率差异是否显著，首先需要假设基因的表现性与疾病没有关联。基因型频率（F_{AA}，F_{Aa}，F_{aa}）可以通过下面的公式计算：

$$F_{AA} = \frac{a+d}{n}, \quad F_{Aa} = \frac{b+e}{n}, \quad F_{aa} = \frac{c+f}{n}$$

从这些基因频率可以分别得到有病样本和无病样本的期望出现次数，如下表 8 所示。

	AA	Aa	aa	总计
有病样本	$n_{case}F_{AA}$	$n_{case}F_{Aa}$	$n_{case}F_{aa}$	n_{case}
无病样本	$n_{control}F_{AA}$	$n_{control}F_{Aa}$	$n_{control}F_{aa}$	n_{case}
总计	a+d	b+c	c+f	n

表 8. 双等位基因碱基对类型样本期望结果示例

单位点致病作用的检测方法是判断独立假设条件下的观察值（表 7）与期望值（表 8）是否显著，可以通过卡方检验来测试独立性，计算以下公式：

$$\chi^2 = \sum \frac{(A-E)^2}{E} = \sum_{i=1}^{k} \frac{(A_i - E_i)^2}{E_i} = \sum_{i=1}^{k} \frac{(A_i - np_i)^2}{np_i}$$

其中，对于行 i 和列 j，Aij 表示观察样本出现的次数，而 Eij 是表示期望出现的次数，如果满足假设，则不存在关联为真，则公式计算得到的统计证近似满足卡方分布，自由度为 2，在我们的 SNP 数据集中。通过这个分布，可以得到一个

p 值，较少的 p 值表明基因型与表现型存在关联关系，对应的 SNP 位点可能是致病 SNP 位点。卡方检验的 p 值表示在假设条件下，观察到的当前值或者比当前值更极端的情况的概率和。

通过单变量的测试，可以对每个 SNP 位点得到一个 p 值，然后通过 p 值对 SNP 位点进行排名。通常设置 p 值的显著水平 α 为 0.05，小于当前这个值对应的 SNP 位点可以认为是致病 SNP 位点，但是对于全基因组测试存在一个多重检验的问题，检验的次数约多，得到显著结果的概率越大。

根据卡方检验结果，所选的前三个位点的卡方值分别为 28.83615 ，16.9097 ，14.55641。根据卡方检验表（表 9），当自由度为二时，置信概率为 0.001 时的卡方临界值为 13.82，这个卡方值大于我们所检测到的位点信息样本的卡方值。所以可以说明位点 rs2273298、rs7543405 和 rs7368252 与该种疾病相互独立，也就是说这个位点并非是致病位点的概率小于 0.001。换句话说该位点是致病位点的概率大于 99.9%。

自由度k \ P value （概率）	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.60	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.82	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88
10	3.94	4.86	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59

表 9. 卡方不同自由度下临界值

为了进一步验证该结论，我们又对位点 rs2273298、rs7543405 和 rs7368252 进行了两两碱基对组合，分别统计各个碱基对组合在有病样本和无病样本中出现的次数，然后求差值的平方。结果如表 9a、9b、9c 所示。为了凸显结果，我们随机选取了两组位点组合，进行了同样的操作，结果如表 9d、9e 所示。五组结果中标黄的差方和有很大差异，a、b、c 三组的差方和都很大，达到几千，而随机抽取的两组位点的差方和较小，这说明我们选择的三个位点是致病位点的可能性极大。

通过仔细对比表 10a、b、c 中各个碱基对组合之间的差异，可以发现 rs2273298（排名 1）和 rs7543405（排名 2）组合碱基对中 06 组合在有无病样本的差异非常突出，rs2273298（排名 1）和 rs7368252（排名 3）组合碱基对中 04 组合在有无病样本的差异非常突出，rs7543405（排名 2）和 rs7368252（排名 3）组合碱基对中 64 组合在有无病样本的差异非常突出。不难看出 rs2273298（排名 1）的 0 号碱基对 AA、rs7543405（排名 2）的 6 号碱基对 CT/TC 和 rs7368252（排名 3）的 4 号碱基对 CC 在疾病有无上的影响很大。所以可以说在该模型下，疾病是这

三个位点同时决定的。

rs2273298（排名 1）和 rs7543405（排名 2）碱基对组合的有病与无病样本数量										合计
组合	04	06	09	24	26	29	74	76	79	
有病样本个数	82	97	43	74	89	55	19	27	14	500
无病样本个数	79	174	52	48	84	29	9	20	5	500
差方	9	5929	81	676	25	676	100	49	81	7626

表 10a. rs2273298（排名 1）和 rs7543405（排名 2）碱基对组合的有病与无病样本数量差方

rs2273298（排名 1）和 rs7368252（排名 3）碱基对组合的有病与无病样本数量										合计
组合	04	06	09	24	26	29	74	76	79	
有病样本个数	112	96	14	112	94	12	31	27	2	500
无病样本个数	185	105	15	105	43	13	21	11	2	500
差方	5329	81	121	49	2601	1	100	256	0	8538

表 10b. rs2273298（排名 1）和 rs7368252（排名 3）碱基对组合的有病与无病样本数量差方

rs7543405（排名 2）和 rs7368252（排名 3）碱基对组合的有病与无病样本数量										合计
组合	44	46	49	64	66	69	94	96	99	
有病样本个数	74	89	12	121	79	13	60	49	3	500
无病样本个数	73	54	9	177	84	17	61	21	4	500
差方	1	1225	9	3136	25	16	1	784	1	5198

表 10c. rs7543405（排名 2）和 rs7368252（排名 3）碱基对组合的有病与无病样本数量差方

rs3094315（随机）和 rs3131972（随机）碱基对组合的有病与无病样本数量										合计
组合	44	46	49	64	66	69	94	96	99	
有病样本个数	5	7	5	51	68	27	122	157	58	500
无病样本个数	4	19	3	45	78	24	105	154	68	500
差方	1	144	4	36	100	9	289	9	100	692

表 10d. rs3094315（随机）和 rs3131972（随机）碱基对组合的有病与无病样本数量差方

rs510504（随机）和 rs7521204（随机）碱基对组合的有病与无病样本数量										合计
组合	04	06	09	24	26	29	74	76	79	
有病样本个数	7	66	100	9	66	163	4	32	53	500
无病样本个数	7	70	121	14	65	147	3	20	53	500
差方	0	16	441	25	1	256	1	144	0	884

表 10e. rs510504（随机）和 rs7521204（随机）碱基对组合的有病与无病样本数量差方

4.3 问题三

前面我们通过建立基于计数统计的单位点差值运算模型，从 1000 个样本中筛选出 rs2273298、rs7543405 和 rs7368252 致病位点。在满足假设条件下，我们通过卡方检验，发现这三种致病位点与疾病相关的可能性大于 99.9%。

问题三中要求我们利用遗传疾病与基因的关联性从 300 多组基因数据中找出与疾病最有可能相关的一个或几个基因。如果在原假设条件成立的情况下，可以假设每组基因对于致病的可能性相互独立，并且 001 是致病基因不影响 002 也是致病基因。这样我们就可以通过在已知致病位点的情况下将致病位点从所属基因中筛选出来。

如果说位点是一个元素，那么基因就是一个子集。我们可以将最有可能的几个致病位点作为目标，遍历 300 多个基因数据库进行搜索。这样就会产生多个含有致病位点信息的基因，再将含有致病位点的基因通过 C 语言算法的形式输出成文件名的形式，这样致病基因就通过筛选产生了。流程图如图 3 所示。C 语言程序见附录 3。

最终找到的含有 rs2273298、rs7543405 和 rs7368252 致病位点的基因有：gene_55，gene_102 和 gene_265。

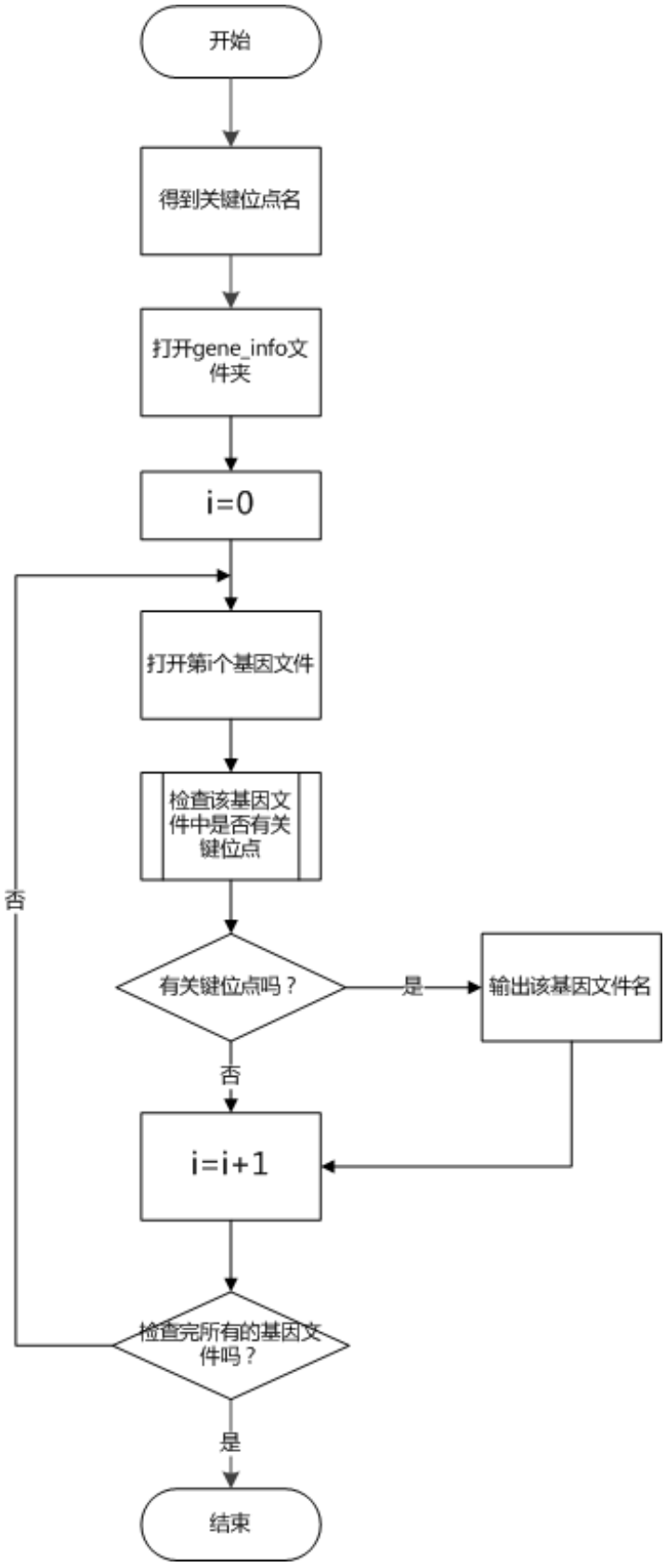
开始
得到关键位点名
打开gene_info文件夹
i=0
打开第i个基因文件
检查该基因文件中是否有关键位点
有关键位点吗？
是
输出该基因文件名
否
i=i+1
检查完所有的基因文件吗？
否
是
结束

图 3. 求关键基因流程图

4.4 问题四

前面的问题都是基于假设，假设这个致病位点是单位点关联性的，只有一个位点起作用。然而在实际的研究中，科研人员往往把相关的性状或疾病看成一个整体，然后来探寻与它们相关的位点或基因。题目中要求我们将原来 1000 个样本，按照性状与位点的相关性进行筛选，筛选出和 10 个性状或者多个性状都具有高度相关性位点的位点信息。如果考虑性状都是单位点关联性的，可以按照第二个问题的模型，通过分别统计不同位点的性状表现型的不同来统计样本。将有表现型的样本设为 1 无表现型的样本设为 0。通过百分比差值的大小进行排序筛选出 10 对与样本性状表现型具有高度相关性的位点信息。

然而由于 10 种性状可能是由于一个或者多个位点共同控制的。如果单纯地依靠上述方法是无法准确获取我们希望得到的与这 10 种性状都高度关联的位点，仅仅能得出 10 种与其他性状没有关联性的不同的位点信息。我们需要对 10 种性状表现型不同的 1000 组样本进行处理，我们采取了以下方案。

首先将 10 种性状按照表现型的多少进行统计，设 10 个相关性状的 1 作为显性表现型，0 作为隐形表现型。将 1000 组样本按照 10 种性状的出现显性表现型的数量从 0 到 10 进行排序，统计结果如下表 11 所示。

表现型量数	0	1	2	3	4	5	6	7	8	9	10
样本数量	300	76	48	32	27	30	27	36	50	80	294
百分比	30%	7.6%	4.8%	3.2%	2.7%	3%	2.7%	3.6%	5%	8%	29.4%

表 11. 不同性状表现型的样本数量及百分比

我们发现 1000 组位点信息中，0 个显性表现型和 10 个显性表现型的样本数量占据了整个样本数量的 59.4%，而 2 个显性表现型到 8 个显性表现型样本数量占据了总的样本数量的 25%。而且各种显性表现型次数 0 与 10、1 与 9、2 与 8…的样本个数基本对称，可以认为这 10 个关联性状也都是等位的。为此，假设剔除掉 2 个显性表现型到 8 个显性表现型样本对总的样本空间没有很大的影响，并且对致病位点的选取也没有较大的变化。

我们建立这样一个模型，将 2 种到 8 种总共 7 种表现型的样本在总的样本空间中进行剔除。筛选出剩余的 750 个样本然后分别将具有 9 个显性表现型和 10 个显性表现型的样本标记为 1，将具有 0 个显性表现型和 1 个显性表现型的样本标记为 0。然后分别依据样本的标记信息将原有样本的位点信息进行对应筛选，筛选出 750 个具有位点信息的样本信息，同时标记上 1 和 0（其中具有 9 个显性表现型和 10 个显性表现型的样本标记为 1，将具有 0 种表现型和 1 种表现性的样本标记为 0）。选用的样本中显性的个数为 374 个，隐性的个数为 376，计算时各从中挑选 370 个。

表现型量数	0	1	2	3	4	5	6	7	8	9	10
样本数量	300	76	48	32	27	30	27	36	50	80	294
百分比	30%	7.6%	25%							8%	29.4%

表 12. 不同性状表现型的样本数量及百分比及所选的样本（标粉色）

同样，由于不同位点碱基是等位的，理想情况下我们只考虑一个位点的情况。

分别统计 9000 多个位点中每一个位点的 0 表现型和 1 表现型的 3 种碱基对的人数分布。剩余步骤参照问题二。

利用模型二按照各个碱基对计数差值绝对值之和降序排列，筛选的结果前 8 位的位点分别是 rs12722898，rs10737913，rs2501430，rs4360511，rs4073710，rs2038095，rs12758112，rs716325，rs387232。前几个位点的差值之和相差不大，都能通过卡方检验，很难确定前几位到底哪几个位点决定这 10 个关联性状。利用模型二的程序得到的结果类似，前几位的区分度不大。我们分析原因主要是数据较少，因此要考虑改变所采用的样本量。

位点名	rs12722898	rs10737913	rs2501430	rs4360511	rs4073710	rs2038095	rs12758112	rs716325
隐性组 碱基对								
0	77	0	161	130	80	0	42	9
1	0	0	158	0	0	0	0	0
2	208	0	0	196	160	0	141	103
3	0	0	0	0	0	0	0	0
4	0	58	51	0	0	60	0	0
5	0	0	0	0	0	0	0	0
6	0	188	0	0	0	207	0	0
7	85	0	0	44	130	0	187	258
8	0	0	0	0	0	0	0	0
9	0	124	0	0	0	103	0	0
显性组 碱基对								
0	126	0	115	176	67	0	53	9
1	0	0	186	0	0	0	0	0
2	174	0	0	156	205	0	175	148
3	0	0	0	0	0	0	0	0
4	0	53	69	0	0	86	0	0
5	0	0	0	0	0	0	0	0
6	0	147	0	0	0	162	0	0
7	70	0	0	38	98	0	142	213
8	0	0	0	0	0	0	0	0
9	0	170	0	0	0	122	0	0
差值绝对值之和	98	92	92	92	90	90	90	90

表 13. 按差值绝对值之和降序排列后前几个位点的十个碱基对
显性计数值与隐性计数值差值（370 个样本）

再次选择样本，将 3 种到 7 种总共 5 种表现型的样本在总的样本空间中进行剔除。筛选出剩余的 848 个样本，然后分别将具有 8 个、9 个和 10 个显性表现型的样本标记为 1，将具有 0 个、1 个和 2 个显性表现型的样本标记为 0（如表 14）。然后分别依据样本的标记信息将原有样本的位点信息进行对应筛选，筛选出 848 个具有位点信息的样本信息，同时标记上 1 和 0（其中具有 8 个、9 个和 10 个显性表现型的样本标记为 1，将具有 0 个、1 个和 2 种表现性的样本标记为 0）。总共的显性样本和隐性样本各 424 个。重复上次试验的计算过程。

表现型量数	0	1	2	3	4	5	6	7	8	9	10
样本数量	300	76	48	32	27	30	27	36	50	80	294
百分比	30%	7.6%	4.8%	15.2%					5%	8%	29.4%

表 14. 不同性状表现型的样本数量及百分比及再次选择的样本（标粉色）

利用问题二中的模型一（流程图如图 4，C 语言程序见附录 3），我们得到了比 370 个样本时较少的筛选结果，筛选出的位点如表 15 所示。在差值百分比阈值分别为 11%和 12%的情况下，前三位的位点均是 rs7538876，rs4073710 和 rs351617。

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs7538876	0.132075	rs11121821	0.115566	rs7549888	0.110849
rs4073710	0.125	rs1932397	0.115566	rs2501430	0.110849
rs351617	0.122642	rs780983	0.113208	rs12722898	0.110849
rs3218121	0.117925	rs12758112	0.110849	rs35107626	0.110849
rs716325	0.117925	rs2247525	0.110849		

表 15a. 样本容量 424、百分比差阈值 11%的筛选结果

位点	百分比差值	位点	百分比差值	位点	百分比差值
rs7538876	0.132075	rs4073710	0.125	rs351617	0.122642

表 15b. 样本容量 424、百分比差阈值 12%的筛选结果

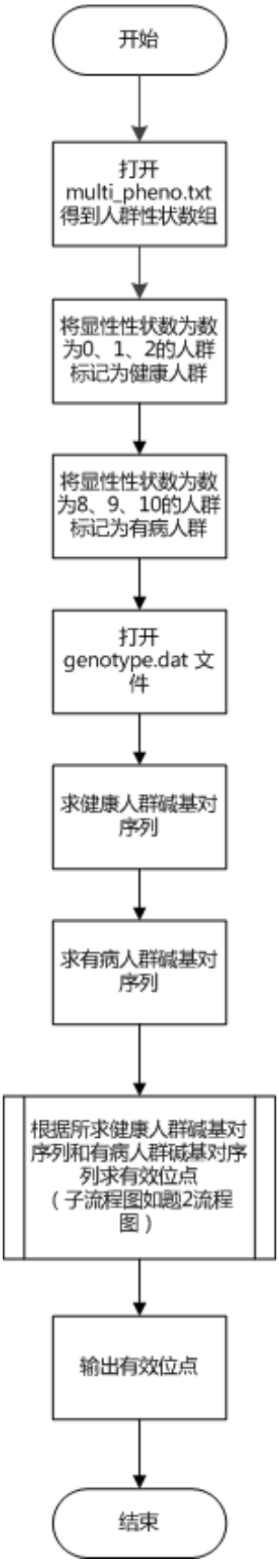

图 4. 求与性状关联位点流程图

利用模型二按照各个碱基对计数差值绝对值之和降序排列，筛选的结果前 8 位的位点分别是 rs7538876，rs4073710，rs351617，rs3218121，rs716325，rs11121821，rs1932397 和 rs780983（表 16）。其中前三位的对应碱基计数差值之和显著高于其他位点的，该结果再次与模型一的结果吻合。

位点名	rs7538876	rs4073710	rs351617	rs3218121	rs716325	rs11121821	rs1932397	rs780983
隐性组 碱基对								
0	71	96	0	0	16	261	71	0
1	0	0	0	0	0	0	0	0
2	245	183	0	0	117	138	241	0
3	0	0	0	0	0	0	0	0
4	0	0	326	265	0	0	0	141
5	0	0	0	0	0	0	0	0
6	0	0	89	139	0	0	0	228
7	108	145	0	0	291	25	112	0
8	0	0	0	0	0	0	0	0
9	0	0	9	20	0	0	0	55
显性组 碱基对								
0	108	74	0	0	13	221	90	0
1	0	0	0	0	0	0	0	0
2	189	236	0	0	167	187	192	0
3	0	0	0	0	0	0	0	0
4	0	0	274	215	0	0	0	165
5	0	0	0	0	0	0	0	0
6	0	0	133	187	0	0	0	180
7	127	114	0	0	244	16	142	0
8	0	0	0	0	0	0	0	0
9	0	0	17	22	0	0	0	79
差值绝对值之和	112	106	104	100	100	98	98	96

表 16. 按差值绝对值之和降序排列后前几个位点的十个碱基对
显性计数值与隐性计数值差值（424 个样本）

所以我们最终确定的与 10 个关联性状有关的位点分别是 rs7538876，rs4073710 和 rs351617。

5 总结与展望——其他精确模型

在对十种碱基对进行编码后，我们对单位点进行了统计，各个位点的统计结果都是随机分布的，即各个位点的碱基类型及比例在有病组和无病组都有所区别但区别是没有规律的，这也是单位点关联假设建立的依据。因此我们建立相应模型确定了三个致病位点，并且这三个位点都通过 χ^2 检验。还通过对这三个位点两两进行碱基对组合，对比确定了这三个位点上分别由哪种碱基对控制。这说明对于该类疾病来说确实是由三个致病位点控制的，也充分证明了我们所建模型的可靠性。在此基础上我们完成了问题三（对 300 个基因的筛查，确定了哪些基因包含致病位点）以及问题四（确定与 10 种关联形状相关的致病位点）。

目前来看，如果样本数在不是很多的情况下，位点数目比较庞大，单独依靠这种基于差值的计数统计仍然会有遗漏部分位点的情况。如果性状是由多个位点共同控制的还会对这种模型的精准度提出一定的挑战。未来这种算法的发展方向还是用来筛选位点，将较大可能性致病的位点和基因找到，除此之外还可以结合相关的随机森林算法对筛选剩下的位点进行进一步筛选。

然而如果要建立一种通用的模型以适用于各类疾病的致病位点检测，我们需要建立更精确的模型。在此我们提出了更复杂且更严密的通用模型的概念，为今后的研究提供参考。

如果该疾病是由 q 个位点组合决定的（q 为未知的变量，待确定，q=2，3，4…q_{max}；q_{max} 为总位点数），可以用穷举法将各个位点的十种碱基对与其他位点的碱基对组合，然后统计各个组合对应的有病样本和无病样本的个数差异，用 D 来表示这个差异。这个差异可以有多种衡量方式，如差方和、差值绝对值加和等。对所有位点的十种碱基对组合（共有 3^q 种）差异都计算出来，按从大到小的顺序排列，最终把致病位点筛选出来。这种在不知道 q 为多少的情况下的计算量相当大，对于每一个 q 都要计算 3^q 次，总的计算量仍高达 $\sum_{q=2}^{10} 3^q$ 次。即便假设 q_{max}=10，这需要计算 88569 次。当 q_{max}=100 时，需要计算 $7.730662811 \times 10^{47}$ 次。当 q_{max}=10000 时，大约需要计算 10^{5000} 次。因此这种算法需要大型计算机来完成。通过这种方式得到的结果一定是可靠的，因为它将所有可能出现的情况都穷举了一遍。这种方法在现实中是否可行，还依赖于计算机的计算能力。

总之，在基因致病位点的研究上我们的方法越来越多，总有一天我们可以找到所有的与各类疾病相关的致病位点和包含这些位点的基因，为人类优质繁殖提供依据。

参考文献

[1] 黎成.基于随机森林和 ReliefF 的致病 SNP 识别方法[D].西安：西安电子科技大学，2014:7-35.

[2] 李扬.中国汉族人群特应性皮炎全基因组关联研究[D].合肥：安徽医科大学，2011:80-81.

[3] 石林.一种全基因组关联性分析新流程及其在孤独症的应用[D].北京：清华大学，2010：11-42.

[4] Andrea S. Foulkes. Applied Statistical Genetics with R[M]. Springer Dordrecht Heidelberg London New YorK， 2009.

[5] Kwak M， Joo J， Zheng G. A robust test for two-stage design in genome-wide association studies[J]. Biometrics. 2009 Dec;65(4):1288-95.

[6] 陈峰， 曾平，赵杨.新一代测序数据的罕见遗传变异关联性统计方法. 中国卫生统计，2015.(6):1091-1096.

[7] 林东昕， 李娇元，缪小平.中国常见肿瘤的全基因组关联研究进展. 自然杂志，2015.(1):1-7.

[8] 王杉，禚洪庆，张辉.结直肠癌全基因组关联研究现状. 中华普通外科杂志，2012.(5):434-436.

[9] 张学军， 汤华阳.全基因组关联研究 meta 分析搜寻中国汉族人系统性红斑狼疮易感基因. 安徽医科大学学报， 2013.(7):F0003-F0003.

[10] 周宝森， 全晓薇，夏玲姿，王芊芊，张莹，屈若祎.rs586610 基因位点多态性与中国辽宁省人群原发性肝癌易感性关系的验证. 中华疾病控制杂志，2015.(11):1079-1082.

[11] 周文明， 杜鹃，高金平.斑秃易感位点与中国汉族人群相关性研究. 安徽医科大学学报， 2015.(5):612-615.

附录

附录 1 问题一源代码

```
// 1_readgenotye.cpp


#include<stdio.h>
#include<stdlib.h>
#include<string>
#define N 100000
using namespace std;



char swtch(char c1，char c2)
{
	char ctem='X';
	switch(c1)
	{
	case 'A':
		switch(c2)
		{
		case 'A':
			ctem='0';
			break;
		case 'C':
			ctem='1';
			break;
		case 'G':
			ctem='2';
			break;
		case 'T':
			ctem='3';
			break;
		default:
			break;
		}
		break;

	case 'C':
```

```
        switch(c2)
        {
        case 'A':
                ctem='1';
                break;
        case 'C':
                ctem='4';
                break;
        case 'G':
                ctem='5';
                break;
        case 'T':
                ctem='6';
                break;
        default:
                break;
        }
        break;

case 'G':
        switch(c2)
        {
        case 'A':
                ctem='2';
                break;
        case 'C':
                ctem='5';
                break;
        case 'G':
                ctem='7';
                break;
        case 'T':
                ctem='8';
                break;
        default:
                break;
        }
        break;

case 'T':
        switch(c2)
        {
        case 'A':
                ctem='3';
```

```
                break;
            case 'C':
                ctem='6';
                break;
            case 'G':
                ctem='8';
                break;
            case 'T':
                ctem='9';
                break;
            default:
                break;
            }
            break;
        default:
            break;
        }
        return ctem;
}


int main()
{
        FILE *in，*out;
        char infile[14]，outfile[16];

        strcpy(infile，"genotype.dat");
        if((in=fopen("genotype.dat"，"r"))==NULL)
        {
            printf("Open file ERROR !/n");
            exit(0);
        }

        strcpy(outfile，"genotypebi.dat");
        if((out=fopen("genotypebi.dat"，"w"))==NULL)
        {
            printf("Open file ERROR !/n");
            exit(0);
        }

        int ba=0，i=0;
        string s;
        char *cc，c，te;
        cc= (char*)malloc(N*sizeof(char)) ;
```

```
    fgets(cc，N，in);
    fputs(cc，out);
    int col=0;
    while((!feof(in))&&(col<1000))
    {
        fgets(cc，N，in);
        for(i=0;cc[ba+i]!='\0';i=i+3)
        {
            te=swtch(cc[ba+i]，cc[ba+i+1]);
            fputc(te，out);
            fputc(9，out);
        }
            fputc(10，out);
            col++;
    }

        printf("Success!\n");
    free(cc);
    fclose(in);
    fclose(out);
    return 0;
}


/*

    A   C   G   T

A   0   1   2   3

C   1   4   5   6

G   2   5   7   8

T   3   6   8   9

异常 X

*/
```

附录 2 问题二源代码

```
// 2_getrs.cpp


#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<algorithm>
#include<vector>
#define N 100000
#define ER 10
#define DVALUE 0.13
#define TR 500
#define NU 99
using namespace std;



short uh_rs[500][1000];
short h_rs[500][1000];
vector<int> rs_qu;
vector<float>rs_pe;



void readrs(int cx)
{
	FILE *fp;
	if((fp=fopen("genotypebi.dat"， "r"))==NULL)
	{
		printf("Open file ERROR !/n");
		exit(0);
	}

	int i_nu=0;
	char *cc， tempc;
	cc= (char*)malloc(N*sizeof(char)) ;
	fgets(cc， N， fp);
	while((!feof(fp))&&(i_nu<1000))
	{
		fgets(cc， N， fp);
```

```
        if(i_nu<500)
            {
                for(int i=0;i<1000;i++)
                {

                        tempc=cc[i*2+cx*2000];
                    if((tempc>='0')&&(tempc<='9'))
                        h_rs[i_nu][i]=tempc-'0';
                    else
                        h_rs[i_nu][i]=ER;
                }
                if(cx>=9)
                {
                    for(int j=445;j<1000;j++)
                        h_rs[i_nu][j]=NU;
                }
            }
            else
            {
                for(int i=0;i<1000;i++)
                {
                    tempc=cc[i*2+cx*2000];
                    if((tempc>='0')&&(tempc<='9'))
                        uh_rs[i_nu-500][i]=tempc-'0';
                    else
                        uh_rs[i_nu-500][i]=ER;
                }
                if(cx>=9)
                {
                    for(int j=445;j<1000;j++)
                        uh_rs[i_nu-500][j]=NU;
                }
            }


            i_nu++;
        }


        free(cc);
        fclose(fp);
}
```

```
bool getrs_no(float pe，  int rs_no)
{
    if (pe >= DVALUE)
    {
        rs_qu.push_back(rs_no);
        rs_pe.push_back(pe);
        return true;
    }
    return false;
}


void swh(int *u，int uh_rs)
{
        switch(uh_rs)
        {
        case 0:
            u[0]++;
            break;
        case 1:
            u[1]++;
            break;
        case 2:
            u[2]++;
            break;
        case 3:
            u[3]++;
            break;
        case 4:
            u[4]++;
            break;
        case 5:
            u[5]++;
            break;
        case 6:
            u[6]++;
            break;
        case 7:
            u[7]++;
            break;
        case 8:
            u[8]++;
            break;
```

```
            case 9:
                u[9]++;
                break;
            case 10:
                u[10]++;
                break;
            default:
                break;
            }
}


void rs_count(int rs_no， int cx)
{
    int h_c[11]， h_sum;
    int uh_c[11]， uh_sum;
    int i;

    float h_pe[11];
    float uh_pe[11];


    for(i=0;i<11;i++)
    {
        h_c[i]=0;
        uh_c[i]=0;
        h_pe[i]=0;
        uh_pe[i]=0;
    }
    for (i = 0; i < TR; i++)                //TR 个建立模型，剩下的检验
        swh(uh_c， uh_rs[i][rs_no%1000]);

    for(i=0;i<TR;i++)                //TR 建立模型，剩下 100 检验
        swh(h_c， h_rs[i][rs_no % 1000]);

    h_sum=0;
    uh_sum=0;
    for(i=0;i<11;i++)
    {
        h_sum+=h_c[i];
        uh_sum+=uh_c[i];
    }
    if(h_sum!=TR)
        printf("%d SUM ERROR\n"， h_sum);
```

```
	if(uh_sum!=TR)
		printf("%d	UH SUM ERROR\n"，uh_sum);

	for (i = 0; i < 11; i++)
	{
		h_pe[i] = (float)h_c[i] / (float)TR;
		uh_pe[i] = (float)uh_c[i] / (float)TR;
	}


	float pemax = abs(h_pe[0] - uh_pe[0]);
	for (i = 1; i < 11; i++)
	{
		if (pemax<abs(h_pe[i] - uh_pe[i]))
			pemax = abs(h_pe[i] - uh_pe[i]);
	}
	bool flag = getrs_no(pemax，  rs_no);

}


int findno(int nu)
{
	for(int i=0;i<rs_qu.size();i++)
	{
		if(nu==rs_qu[i])
			return i;
	}
	return N;
}


void outrst()
{
	FILE *fp;
	if((fp=fopen("genotypebi.dat"，"r"))==NULL)
	{
		printf("Open file ERROR !/n");
		exit(0);
	}


	FILE *out;
	if((out=fopen("keyrs_pe.dat"，"w"))==NULL)
```

```
{
    printf("Open file ERROR !/n");
    exit(0);
}


FILE *outrs;
if((outrs=fopen("keyrs.dat"，"w"))==NULL)
{
    printf("Open file ERROR !/n");
    exit(0);
}


char ch[16]={"Sample size: N="};
fputs(ch，out);
fprintf(out，"%d\n"，TR);
char ch1[16]={"DVALUE="};
fputs(ch1，out);
fprintf(out，"%.2f\n"，DVALUE);
char *cc，temps[11];
cc= (char*)malloc(N*sizeof(char)) ;
fgets(cc，N，fp);

int nu=0，t_i=0;

for(int i=0;cc[i]!=NULL;i++)
{

    if(cc[i]=='r')
    {
        temps[0]=cc[i];
        t_i=0;
    }
    else if((cc[i]!='s')&&((cc[i]<'0')||(cc[i]>'9')))
    {
        t_i++;
        temps[t_i]='\0';
        int veno=findno(nu);
        if(veno!=N)
        {
            fprintf(out，"%s\t%f\n"，temps，rs_pe[veno]);
            fprintf(outrs，"%s\n"，temps);
            printf("%s\t\t%d\t%f\n"， temps， nu， rs_pe[veno]);
```

```
            }
            nu++;
        }
        else
        {
            t_i++;
            temps[t_i]=cc[i];
        }
    }


    free(cc);
    fclose(fp);
    fclose(out);
    fclose(outrs);
}


int main()
{
    int cx=0;       //每次 1000 个循环 9 次

    for(cx=0;cx<10;cx++)
    {
        readrs(cx);
        if(cx<9)
        {
            for(int rs_no=0;rs_no<1000;rs_no++)          //rs_no 位点
            {
                rs_count(rs_no+cx*1000，cx);//统计
            }
        }
        else
        {
            for(int rs_no=0;rs_no<445;rs_no++)           //rs_no 位点
            {
                rs_count(rs_no+cx*1000，cx);//基因统计
            }
        }
    }

    outrst();
    return 0;
}
```

附录 3 问题三源代码

```
//3_getgeno.cpp


#include<stdio.h>
#include<stdlib.h>
#include<vector>
#include<algorithm>
#define N 12
using namespace std;



vector<string> keyrs;



void readkeyrs()
{
	FILE *fp;
	if((fp=fopen("keyrs.dat"，"r"))==NULL)
	{
		printf("Open file ERROR !\n");
		exit(0);
	}

	char *cc;
	cc=(char *)malloc(N*sizeof(char));
	fgets(cc，N，fp);
	keyrs.push_back(cc);
	while(!feof(fp))
	{
		fgets(cc，N，fp);
		keyrs.push_back(cc);
	}
	keyrs.pop_back();

	fclose(fp);
	free(cc);

	for(int i=0;i<keyrs.size();i++)
		printf("%s\n"，keyrs[i].c_str());
```

```
}


bool cmprs(char *rs_c)
{
	for(int i=0;i<keyrs.size();i++)
		if (!strcmp(keyrs[i].c_str()， rs_c))
			return true;
		else
			return false;
}


void getgefilename(int geno)
{
	char *fname;
	fname = (char *)malloc(50 * sizeof(char));
	sprintf(fname， "gene_%u.dat"， geno);
	FILE *fp;
	if ((fp = fopen("keygenoname.dat"， "w")) == NULL)
	{
		printf("Open file ERROR !\n");
		exit(0);
	}

	fprintf(fp， "%s\n"， fname);
	printf("%s\n"，fname);
	free(fname);
	fclose(fp);
}


void main()
{
	readkeyrs();
	char *f_name;
	f_name=(char *)malloc(50*sizeof(char));
	for(int i=0;i<300;i++)
	{
		sprintf(f_name， "gene_info\\gene_%u.dat"， i+1);

		FILE *fp;
		if((fp=fopen(f_name，"r"))==NULL)
		{
```

```
            printf("Open file ERROR !%d\n"， i);
            exit(0);
        }
        int fte=0;
        fseek(fp， 0， 2);
        fte=ftell(fp);
        fseek(fp， 0， 0);
        char *rs_cmp;
        rs_cmp=(char *)malloc(N*sizeof(char));
        fgets(rs_cmp， N， fp);
        while(fte!=ftell(fp))
        {
            int sl=strlen(rs_cmp);
            for(int
j=0;((rs_cmp[j]>'9')||(rs_cmp[j]<'0'))&&(rs_cmp[j]!='r')&&(rs_cmp[j]!='s');j++)
            {
                    rs_cmp[j]='\0';
            }
            if (cmprs(rs_cmp) == true)
                getgefilename(i + 1);
            fgets(rs_cmp， N， fp);
        }

        fclose(fp);
    }
    free(f_name);

}
```

附录 4 问题四源代码

```
//4_findresh.cpp


#include<stdio.h>
#include<stdio.h>
#include<algorithm>
#include<vector>
#define NP 25
#define TR 424
#define N 100000
#define ER 10
#define NU 99
#define DVALUE 0.12
using namespace std;



vector<int> hqu;
vector<int> uhqu;
vector<int> rs_qu;
vector<float>rs_pe;
short h_rs[TR][1000];              //无病位点信息
short uh_rs[TR][1000];                 //有病位点信息
bool orph[1000][10];       //原始数据数组



void getexchqu()
{

      FILE *fp;
      if ((fp = fopen("multi_phenos.txt",   "r")) == NULL)
      {
            printf("Open file ERROR !\n");
            exit(0);
      }


      char *cc;
      cc = (char *)malloc(NP*sizeof(char));
      fseek(fp,   0,   2);
```

```
    int fte = ftell(fp);
    fseek(fp， 0， 0);
    int pe_no = 0;
    char c;
    fgets(cc， NP， fp);
    while (ftell(fp) != fte)
    {
        for (int i = 0; i < 10; i++)
        {
            c = cc[i * 2];
            switch (c)
            {
            case'0':
                orph[pe_no][i] = 0;
                break;
            case'1':
                orph[pe_no][i] = 1;
                break;
            default:
                break;
            }
        }
        pe_no++;
        fgets(cc， NP， fp);
    }

    free(cc);
    fclose(fp);
}


void getkeypeno()                    //已验证正确
{
    for (int i = 0; i < 1000; i++)
    {
        int no = 0;
        for (int j = 0; j < 10; j++)
        {
            if (orph[i][j] == true)
                no++;
        }
        if ((no == 0) || (no == 1) || (no == 2))
        {
            hqu.push_back(i);
```

```
            }
            if ((no == 10) || (no == 9) || (no == 8))
            {
                uhqu.push_back(i);
            }
        }
}


void readrs(int cx)            //已验证正确
{
    FILE *fp;
    if ((fp = fopen("genotypebi.dat"，  "r")) == NULL)
    {
        printf("Open file ERROR !/n");
        exit(0);
    }


    int i_nu = 0，pos;
    char *cc，  tempc;
    cc = (char*)malloc(N*sizeof(char));
    fgets(cc，  N，  fp);
    while ((!feof(fp)) && (i_nu < 1000))
    {
        fgets(cc，  N，  fp);

        if (find(hqu.begin()，  hqu.end()，  i_nu) != hqu.end())        //该行在全健康序列中
        {
            pos = 0;
            while (i_nu != hqu[pos])            //求该人在健康序列位置
                pos++;
            if (pos >= TR)
                continue;
            for (int i = 0; i < 1000; i++)
            {
                tempc = cc[i * 2 + cx * 2000];
                if ((tempc >= '0') && (tempc <= '9'))
                    h_rs[pos][i] = tempc - '0';
                else
                    h_rs[pos][i] = ER;
            }
            if (cx >= 9)
            {
```

```
                for (int j = 445; j < 1000; j++)
                    h_rs[pos][j] = NU;
            }

        }


        if (find(uhqu.begin(), uhqu.end(), i_nu) != uhqu.end())        //该行在全有病序列
中
        {
            pos = 0;
            while (i_nu != uhqu[pos])            //求该人在全有病序列位置
                pos++;
            if (pos >= TR)
                continue;
            for (int i = 0; i < 1000; i++)
            {
                tempc = cc[i * 2 + cx * 2000];
                if ((tempc >= '0') && (tempc <= '9'))
                    uh_rs[pos][i] = tempc - '0';
                else
                    uh_rs[pos][i] = ER;
            }
            if (cx >= 9)
            {
                for (int j = 445; j < 1000; j++)
                    uh_rs[pos][j] = NU;
            }
        }
        i_nu++;
    }

    free(cc);
    fclose(fp);
}


void swh(int *u, int uh_rs)
{
    switch (uh_rs)
    {
    case 0:
        u[0]++;
        break;
```

```
        case 1:
            u[1]++;
            break;
        case 2:
            u[2]++;
            break;
        case 3:
            u[3]++;
            break;
        case 4:
            u[4]++;
            break;
        case 5:
            u[5]++;
            break;
        case 6:
            u[6]++;
            break;
        case 7:
            u[7]++;
            break;
        case 8:
            u[8]++;
            break;
        case 9:
            u[9]++;
            break;
        case 10:
            u[10]++;
            break;
        default:
            break;
        }
}


bool getrs_no(float pe，  int rs_no)
{
        if (pe>= DVALUE)
        {
            rs_qu.push_back(rs_no);
            rs_pe.push_back(pe);
            return true;
        }
```

```
    return false;
}


void rs_count(int rs_no，  int cx)
{
    int h_c[11]，  h_sum;
    int uh_c[11]，  uh_sum;
    int i;

    float h_pe[11];
    float uh_pe[11];

    for (i = 0; i<11; i++)
    {
        h_c[i] = 0;
        uh_c[i] = 0;
        h_pe[i] = 0;
        uh_pe[i] = 0;
    }
    for (i = 0; i < TR; i++)
        swh(uh_c，  uh_rs[i][rs_no % 1000]);

    for (i = 0; i<TR; i++)
        swh(h_c，  h_rs[i][rs_no % 1000]);

    h_sum = 0;
    uh_sum = 0;
    for (i = 0; i<11; i++)
    {
        h_sum += h_c[i];
        uh_sum += uh_c[i];
    }
    if (h_sum != TR)
        printf("%d SUM ERROR\n"，  h_sum);
    if (uh_sum != TR)
        printf("%d      UH SUM ERROR\n"，  uh_sum);

    for (i = 0; i < 11; i++)
    {
        h_pe[i] = (float)h_c[i] / (float)TR;
        uh_pe[i] = (float)uh_c[i] / (float)TR;
    }
    float pemax = abs(h_pe[0]-uh_pe[0]);
```

```
    for (i = 1; i < 11; i++)
    {
        if (pemax<abs(h_pe[i]- uh_pe[i]))
            pemax=abs(h_pe[i]- uh_pe[i]);
    }
    bool flag= getrs_no(pemax，  rs_no);

    }


void rers()
{
    int cx = 0;
    for (cx = 0; cx < 10; cx++)
    {
        readrs(cx);
        if (cx < 9)
        {
            for (int rs_no = 0; rs_no < 1000; rs_no++)
            {
                rs_count(rs_no + cx * 1000，cx);
            }
        }
        else
        {
            for (int rs_no = 0; rs_no<445; rs_no++)          //rs_no 位点
            {
                rs_count(rs_no + cx * 1000，  cx);//基因统计
            }
        }

    }
}


int findno(int nu)
{
    for (int i = 0; i<rs_qu.size(); i++)
    {
        if (nu == rs_qu[i])
            return i;
    }
    return N;
}
```

```
void outrst()
{
	FILE *fp;
	if ((fp = fopen("genotypebi.dat"， "r")) == NULL)
	{
		printf("Open file ERROR !/n");
		exit(0);
	}


	FILE *out;
	if ((out = fopen("keyrs_pe.dat"， "w")) == NULL)
	{
		printf("Open file ERROR !/n");
		exit(0);
	}


	FILE *outrs;
	if ((outrs = fopen("keyrs.dat"， "w")) == NULL)
	{
		printf("Open file ERROR !/n");
		exit(0);
	}


	char ch[16] = { "Sample size: N=" };
	fputs(ch， out);
	fprintf(out， "%d\n"， TR);
	char ch1[16] = { "DVALUE=" };
	fputs(ch1， out);
	fprintf(out， "%.2f\n"， DVALUE);
	char *cc， temps[11];
	cc = (char*)malloc(N*sizeof(char));
	fgets(cc， N， fp);

	int nu = 0， t_i = 0;

	for (int i = 0; cc[i] != NULL; i++)
	{

		if (cc[i] == 'r')
```

```
            {
                temps[0] = cc[i];
                t_i = 0;
            }
            else if ((cc[i] != 's') && ((cc[i]<'0') || (cc[i]>'9')))
            {
                t_i++;
                temps[t_i] = '\0';
                int veno = findno(nu);
                if (veno != N)
                {
                    fprintf(out， "%s\t%f\n"， temps， rs_pe[veno]);
                    fprintf(outrs， "%s\n"， temps);
                    printf("%s\t\t%d\t%f\n"， temps， nu， rs_pe[veno]);
                }
                nu++;
            }
            else
            {
                t_i++;
                temps[t_i] = cc[i];
            }
        }


        free(cc);
        fclose(fp);
        fclose(out);
        fclose(outrs);
}


void main()
{
        getexchqu();            //读取 multi_pheno
        getkeypeno();           //求关键人群
        rers();

        outrst();
}
```