基于 KACA 算法的数据隐藏模型

摘　　要

本文基于 k-匿名化模型、KACA 算法和 c-modes 算法对发布数据进行适当隐藏部分信息以供实际需要。

针对问题一，为保证每个个体的信息都能得到保护，本文采用 k-匿名模型中的 KACA 算法，将数据的所有维度的属性作为准标识符，并以两个等价类合并后隐藏数据增量来代替原 KACA 算法的两个等价类的距离，使得改进后的 KACA 算法的隐藏数据量更少。为得到隐藏数据量最少的方案，设置参数 k=2，将附件中的两组二元数据分别使用改进 KACA 算法求解，解得两组二元数据的隐藏数据量最少的方案，其隐藏数据损失比例分别为 4.17% 和 16.58%。

针对问题二，考虑到数据规模的增大，导致算法运行时间的增加，故采用 c-modes 算法和改进 KACA 算法结合的算法：先将数据集聚为 c 类数据，再分别对 c 类数据使用改进 KACA 算法进行 k-匿名。该算法与改进 KACA 算法比较，在运行时间有显著的减少，虽然其隐藏数据比例略高于改进 KACA 算法，但在可接受范围内。由于题目要求给出一个隐藏数据量最少的方案，故设置参数 c=1（即直接使用改进 KACA 算法），k=2，将附件中的两组多元数据分别使用 c-modes-改进 KACA 算法求解，解得两组多元数据的隐藏数据量最少的方案，其隐藏数据损失比例分别为 42.69% 和 63.83%。

针对问题三，考虑到附件所给数据规模不大，运行时间较短，优先考虑隐藏数据量最少方案为最佳隐藏方案，故使用改进 KACA 算法对附件数据分别以 k=3、5、8 情况给出最佳隐藏方案。

针对问题四，情形 1：题目限定了单隐或全隐的隐藏模式，本文在改进 KACA 算法的基础上添加了隐藏模式的约束条件，使求解出的隐藏方案符合情形 1 要求。情形 2：题目需要选出若干维度的数据全发布以确保有效性，本文在改进 KACA 算法的基础，加入一个决策函数筛选一些离散程度较大的维度进行全发布，剩下的维度作为新的准标识符，再使用改进 KACA 算法求解得到隐藏方案。

关键词：k-匿名模型　KACA 算法　c-modes 算法　隐藏数据量

一、 问题介绍

1.1 问题背景

近年来，随着互联网信息技术的快速发展，知识发现和数据挖掘等数据分析技术充分地挖掘了信息资源，为了避免个人隐私信息的泄露，数据隐藏技术已成为一种重要的隐私保护方式[1]。在数据隐藏技术中，如何在隐私安全得到保障的同时减少隐藏数据量、不至于丢失过多信息成为亟待解决的问题，因此通过设计算法获得最佳隐藏方案具有重要意义。

1.2 问题重述

问题一：

在实际生活中，很多个人隐私信息都可以简化为两个不同的选项，根据提供的两组二元数据，保证每个个体的信息都能得到保护，建立二元数据表隐私保护的一般数学模型，分别给出一个隐藏数据量最少的方案。

问题二：

根据提供的两组多元数据，建立多元数据隐私保护的一般数学模型，给出一个隐藏数据量最少的方案。

问题三：

定义 1 (p-重保护) 个体能够隐藏在一个至少包含 p 个个体的数据组中（$p\geq 2$）。

对提供的数据的 p-重保护问题，建立多重保护数据隐藏的数学模型，分别给出 p=3,5,8 情形的最佳隐藏方案。

问题四：

尝试设计数据隐藏技术解决以下几种情况：

(1) 对多元数据表限定了隐藏模式，例如多元 4 维数据在隐藏时只允许单隐或全隐，即限定了 (*,x,x,x)、(x,*,x,x)、(x,x,*,x)、(x,x,x,*) 和 (*,*,*,*) 五种隐藏模式，其中 x 表示相同的数据。

(2) 多元高维数据中需要选出若干维度的数据全发布以确保有效性（假设全发布的数据不构成隐私暴露），对其余维度进行数据隐藏，如多元 6 维数据至少 2 维全发布的示例。

二、 问题分析

问题一分析：为保证每个个体的信息都能得到保护，本文采用 k-匿名模型中的 KACA 算法，将数据的所有维度的属性作为准标识符，并以隐藏数据增量来代替原 KACA 算法的距离，使得改进后的 KACA 算法的隐藏数据量更少。为得到隐藏数据量最少的方案，设置合适的参数 k，应用改进 KACA 算法，解得两组二元数据的隐藏数据量最少的方案。

问题二分析：考虑到数据规模的增大，导致算法运行时间的增加，故采用 c-modes 算法和改进 KACA 算法结合的算法：先将数据集聚为 c 类数据，再分别对 c 类数据使用改进 KACA 算法进行 k-匿名。由于题目要求给出一个隐藏数据量最少的方案，故设置合适的参数 c 与 k，应用 c-modes-改进 KACA 算法求解，解得两组多元数据的隐藏数据量最少的方案，

问题三分析：考虑到附件所给数据规模不大，运行时间较短，优先考虑隐藏数据量最少方案为最佳隐藏方案，故使用改进 KACA 算法对附件数据分别以 k=3、5、8 情况给出最佳隐藏方案。

问题四分析：情形 1：题目限定了单隐或全隐的隐藏模式，本文在改进 KACA 算法的基础上添加了隐藏模式的约束条件，使求解出的隐藏方案符合情形 1 要求。情形 2：题目需要选出若干维度的数据全发布以确保有效性，本文在改进 KACA 算法的基础，加入一个决策函数筛选一些离散程度较大的维度进行全发布，剩下的维度作为新的准标识符，再使用改进 KACA 算法求解得到隐藏方案。

三、 模型假设

(1) 假设所给数据不带有缺失值。
(2) 假设每个属性下的值处于同一层次，即要匿名某属性则匿名后该属性的值只能为 *。
(3) 假设全发布的数据不构成隐私暴露。

四、 符号说明

符号表示	文字说明
D	数据集
T	数据表
QI	T 的准标识符
A	属性子集
k	k-匿名的限制
C	等价类
C'	距 C 最近的等价类
n	数据集中的元组数

m	数据集的初始等价类个数
c	要聚类的数目
t	迭代次数
l	隐匿的元组个数
d	差异度

五、 问题一模型建立与求解

5.1 模型建立

5.1.1 k-匿名模型

定义 2 (k-匿名) 给定数据表 $T(A_1, A_2, \cdots, A_n)$，*QI* 是 *T* 的准标识符，*T[QI]* 为 *T* 在 *QI* 上的投影（可重复），当且仅当在 *T[QI]* 中出现的每组值至少在 *T[QI]* 中出现 *k* 次时，则称 *T* 满足 *k*-匿名。

k-匿名模型[2] 是一种简单而有效的隐私保护模型，其算法可分为两类：全局重编码和局部重编码，其中全局重编码算法产生的信息损失量比局部重编码算法要高很多[3]。

局部重编码算法中基于聚类的 k-匿名化算法产生的信息损失量小，实现简单，其思想是：依据数据集中元组在准标识符上的相似程度，将其划分为若干类，使每个类内的元组数不少于 k 个，然后用 * 取代类内元组在准标识符上的值，以实现数据集的 k-匿名。

定义 3 (等价类) 给定数据表 $T(A_1, A_2, \cdots, A_n)$，在属性子集 $A_i, \cdots, A_j$ 上的一个等价类是指在属性子集 $A_i, \cdots, A_j$ 上取相同值的元组的集合。

5.1.2 改进 KACA 算法

我们改进了 Li 等人[3] 提出的 KACA 算法，使该算法产生的隐藏数据量更小。算法具体流程如下[4]：

函数：$KACA(D, k)$
/*D 表示将要被聚类的数据集，k 表示 k-匿名的限制。*/
begin

(1) 由数据集 D 中生成初始等价类，等价类中各个元组在准标识符上值相等；
(2) 将初始等价类中元组个数大于等于 k 的等价类放入最终匿名表中，不进入循环；
(3) 循环，直到不存在元组个数小于 k 的等价类：

① 随机选择一个大小小于 k 的等价类 C；

② 计算 C 与其他所有等价类并为一类并匿名后的隐藏数据增量；
③ 找到与 C 泛化后隐藏数据增量最少的等价类 C'；
④ 将 C 和 C' 并为一类，并以 * 匿名 C 和 C' 中属性 A_i 为不同值的属性。

(4) 循环结束，返回匿名表。

end

定义 4 (隐藏数据量) 数据集 D 中被 * 匿名的数据总量。

定义 5 (隐藏数据增量) C 和 C' 二者并为一类并匿名后的隐藏数据量与匿名前的隐藏数据量之差。

设数据集中的元组数为 n，数据集的初始等价类个数为 m，元组排序时间复杂度为 $O(nlog\,n)$，KACA 算法的时间复杂度为 $O(nlog\,n+|m|^2)$。

5.2 模型求解

由于附件中的两组二元数据的区别只有维度与数据量，且维度与数据量都不大。故都采用该算法，以二元 1 数据为例。

5.2.1 生成的初始等价类

表 2　二元 1 的初始等价类

初始等价类	元组数
(0, 0, 0, 0, 1, 1)	2
(0, 0, 0, 1, 0, 1)	3
(0, 0, 0, 1, 1, 0)	1
(0, 0, 1, 0, 0, 0)	2
(0, 0, 1, 0, 0, 1)	2
…	…
(1, 1, 1, 0, 1, 1)	3
(1, 1, 1, 1, 0, 0)	3
(1, 1, 1, 1, 0, 1)	1
(1, 1, 1, 1, 1, 0)	1

5.2.2 进行 2-重保护匿名处理后的等价类

表 3　二元 1 的 2-重保护等价类

2-重保护后的等价类	元组数
(0, *, 0, 1, 1, 0)	2
(*, *, 0, 0, 0, 1)	2
(1, 0, 0, 0, *, 0)	2
(1, 0, 0, *, 1, 1)	2
(1, 0, 0, 1, *, 0)	2
…	…
(1, 1, 1, 1, *, *)	2
(0, 0, 0, 1, 0, 1)	3
(0, 0, 1, 1, 1, 0)	4
(1, 1, 1, 0, 1, 0)	2

5.2.3 不同 k 值的隐藏数据比例

定义 6 (隐藏数据比例) 隐藏数据量与数据集数据总量之比。

表 4　二元 1 不同 k 值的隐藏数据比例

k	隐藏数据比例
2	4.16%
3	18.33%
4	33.12%
5	48.33%

随着 k 值的增加（保护重数增加），隐藏数据比例增加，即隐藏数据量增加。故 k 越小隐藏的数据量越少。

题 1 要求保证每个个体的信息都能得到保护，并给出一个隐藏数据量最少的方案。故应采用 k=2（2-重保护）作为模型参数，见上。

5.2.4 结果展示

以下为二元 1 和二元 2 的最小隐藏数据比例及部分结果，详细结果见附录。

二元 1　隐藏数据比例：4.17%

表 5　二元 1 部分结果

1	0	0	0	*	0
1	0	0	0	*	0
1	0	0	1	0	1
1	0	0	1	0	1
1	0	0	1	*	0
1	0	0	1	*	0
1	0	0	*	1	1
1	0	0	*	1	1
1	0	1	0	0	*
1	0	1	0	0	*

二元 2　隐藏数据比例：16.58%

表 6　二元 2 部分结果

0	0	0	0	0	0	0	1	0	1	*	0
0	0	0	0	0	0	0	1	0	1	*	0
0	0	0	0	0	1	1	1	1	1	*	1
0	0	0	0	0	1	1	1	1	1	*	1
0	0	0	0	0	*	0	0	1	1	1	0
0	0	0	0	0	*	0	0	1	1	1	0
0	0	0	0	1	0	*	1	1	*	*	1
0	0	0	0	1	0	*	1	1	*	*	1
0	0	0	0	1	*	0	0	1	*	1	0
0	0	0	0	1	*	0	0	1	*	1	0

六、 问题二模型建立与求解

6.1 模型建立

6.1.1 c-modes 算法

定义 7 (c-means 算法) 基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为类簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。

Chin 等人[5] 基于 c-means 方法提出了一个 k-匿名化算法，该算法先用 c-means 把整个数据集聚成 c 个类，然后合并大小小于 k 的等价类。

为解决分类型数据的聚类问题，我们使用了 Huang 提出[6] 的 c-modes 算法，c-modes 算法是对 c-means 算法的扩展，差别在于聚类时 c-modes 用众值而 c-means 用均值。

算法流程如下[4]：

函数：$c-modes(D,c)$
/*D 表示将要被聚类的数据集，c 表示要聚类的数目。*/
begin
(1) 从数据集 D 中随机选取 c 个元组作为每个类的初始类众值；
(2) 将一个元组分配到离它最近的类中，更新该类的类众值；
(3) 所有的元组都被分配到相应的类之后，重新计算每个元组到各个类众值的距离，如果存在有元组到另-一个类众值的距离小于其到当前类众值的距离，再将该元组重新分配到离其最近的类中；
(4) 重复 (3) 直至每个类都没有元组变换为止；
(5) 返回对数据集 D 的聚类结果。
end

c-modes 算法的时间复杂度是 $O(c \times t \times m)$，其中 t 为迭代次数，c 和 t 都是远小于 n 的常数。

6.1.2 c-modes-改进 KACA 算法

该算法是改进 KACA 算法和 c-modes 算法结合后的一种高效 k-匿名化算法，首先利用 c-modes 算法把大数据集高效地划分为 c 个类，然后采用改进 KACA 算法 k 匿名化这 c 个类中元组数大于 2k-1 的类。

算法流程如下[4]：

算法：$c-modes-KACA(T,QI,k)$
begin

(1) 由数据集 D 生成初始等价类, 等价类中各个元组在准标识符上值相等, 设等价类的个数为 n；
(2) 确定类个数，即为 c；
(3) 执行 c-modes 算法，$C_{result}=c-modes(T[QI],c)$；
(4) 循环：对每一个大于 2k-1 的类 C_i，执行改进 KACA 算法，$KACA(T_{C_i},k)$；
/*T_{C_i} 是 T 的一个子集，是类 C_i 中所有元组的集合 */
结束循环
(5) 若存在小于 k 的等价类，则隐匿等价类中的元组；
(6) 返回匿名表。

end

6.1.3 算法分析

(1) 时间花费为 $O(n)$。
(2) 时间花费为 $O(1)$。
(3) 采用 c-modes 算法对整个数据集进行初始聚类，其时间花费为 $O(c\times t\times m)$。
(4) 元组排序已放入主函数，去掉排序的时间复杂度，利用改进 KACA 算法将 c 个类划分成大小在 [k,2k-1] 上的类，考虑平均情况, 假设 m 个初始等价类被平均分配到 c 个类中, 则每个类的大小近似为 $\frac{m}{c}$，则该步的时间花费为 $c\times O((\frac{m}{c})^2)=O(\frac{m^2}{c})$。
(5) 时间花费为 $O(l)$,l 为需要隐匿的元组个数。

因此，该算法总的时间复杂度为：

$$\begin{aligned}&O(nlog\,n)+O(1)+O(c\times t\times m)+O(\frac{m^2}{c})+O(l)\\=&O(nlog\,n)+O(c\times t\times m)+O(\frac{m^2}{c})\\=&O(nlog\,n+c\times t\times m+\frac{m^2}{c})\end{aligned}\tag{1}$$

由于 c，t 是远小于 m 的常数，故时间复杂度可简写为 $O(\frac{m^2}{c})$，远小于改进 KACA 算法的时间复杂度。

而且，算法的效率主要由 $O(c\times t\times m)+O(\frac{m^2}{c})$ 决定，当 m 和 t 固定时，要使算法的效率达到最高，应使 $O(c\times t\times m+\frac{m^2}{c})$ 最小。设

$$f=c\times t\times m+\frac{m^2}{c}\tag{2}$$

$$f'=t\times m+\frac{-m^2}{c^2}\tag{3}$$

$f' = 0$ 得 $c = \sqrt{\frac{m}{t}}$，此时算法效率最高。

6.2 模型求解

题 2 所给两组多元数据较题 1 的两组二元数据规模更大，等价类数更多，导致改进 KACA 算法的运行时间增加。故本文采用 c-modes 算法和改进 KACA 算法相结合，使得在数据规模大的情况下运行时间显著减少，隐藏数据量在可接受范围内略微增大。两组多元数据皆采用 c-modes-改进 KACA 算法，以多元 2 为例。

题 2 要求给出一个隐藏数据量最少的方案，故我们采用 2-重保护（即 k=2）使隐藏数据量最少，见题 1 模型。

表 7　多元 2 在不同聚类数 c 下的隐藏数据比例与运行时间

c	隐藏数据比例	时间
1	63.86%	1.0841
2	65.53%	0.7346
3	66.51%	0.6449
4	67.28%	0.6537
5	67.99%	0.624

随着聚类数 c 的增加，隐藏数据比例上升较小(在可接受范围)，运行时间下降。但由于所给数据规模较小，无法得到较大的比较差异。故本文另外采用 UCI Machine Learning Repository 提供的 Adult 数据集来验证 c-modes-改进 KACA 算法可行性，Adult 数据集是美国人口普查数据，选取的数据为删除带有缺失值的元组之后的数据，共有 10000 个元组。

6.2.1 运行时间分析

图 1 为 k=2 时，数据元组个数变化的情况下改进 KACA 算法和 c-modes-改进 KACA 算法的运行时间曲线，从图 1 可以看出：随着数据元组个数的增加，两种算法的运行时间均增加，这是因为数据元组个数的增加会增加 k-匿名化的计算量，从而增加了运行时间。另外，随着数据元组个数的增加，c-modes-改进 KACA 算法的运行时间增长缓慢，而改进 KACA 算法的运行时间相对它们却增长急剧。

6.2.2 隐藏数据比例分析

图 2 为 k=2 时，数据元组个数变化的情况下改进 KACA 算法和 c-modes-改进 KACA 算法的隐藏数据比例曲线，从图 2 可以看出：随着数据元组个数的增加，两种算法的

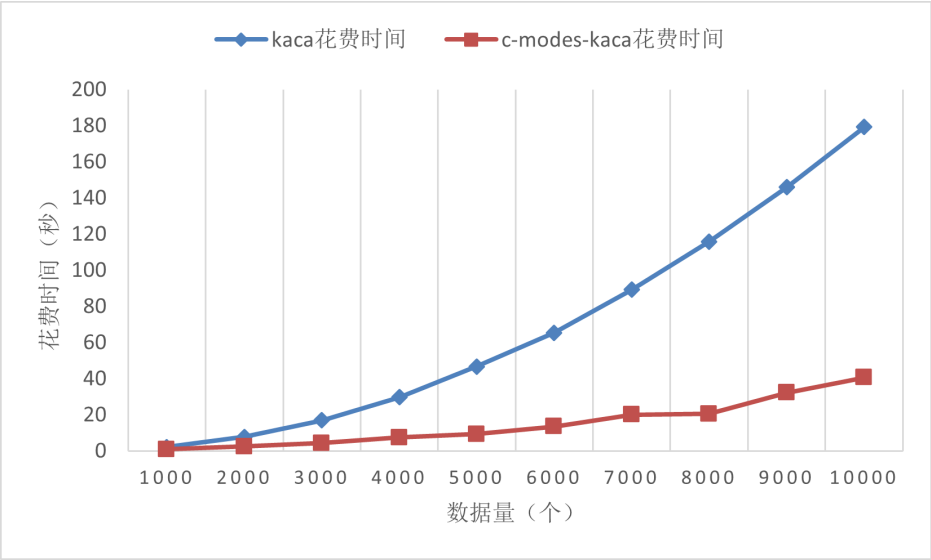

图 1　不同数据量大小下运行时间对比

隐藏数据比例均减少并且 c-modes-改进 KACA 算法的隐藏数据比例一直略微高于改进 KACA 算法的隐藏数据比例。另外，随着数据元组个数的增加，c-modes-改进 KACA 算法与改进 KACA 算法的隐藏数据比例的差距在减少。

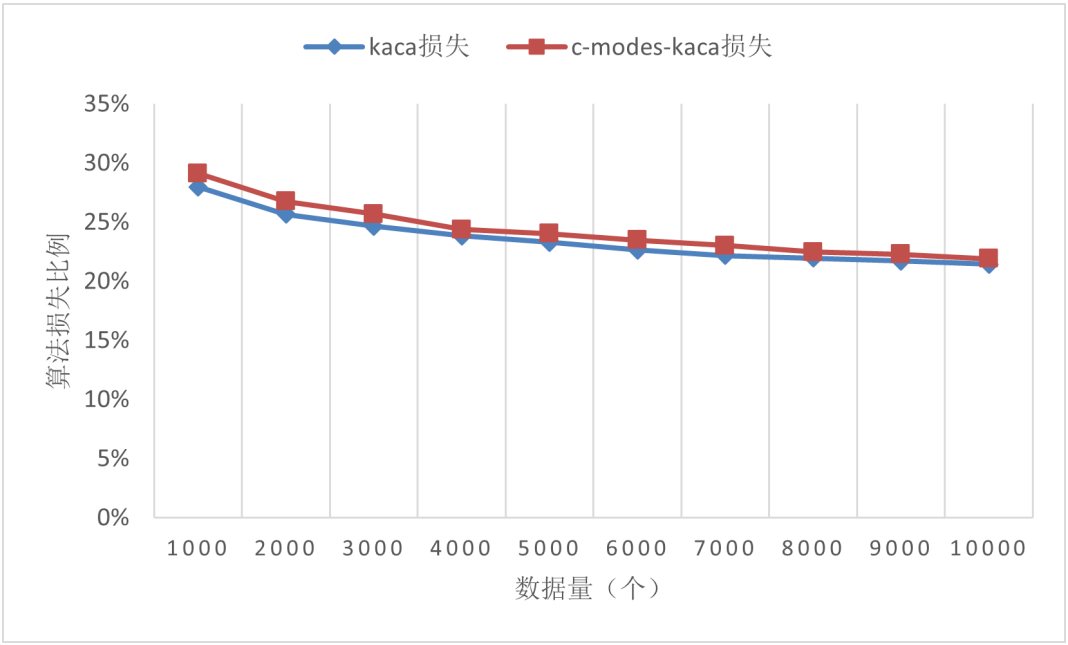

图 2　不同数据量大小下隐藏数据比例对比

6.2.3 综合分析

根据以上分析，综合考虑算法的效率及其产生的隐藏数据比例，可得出结论：c-modes-改进 KACA 算法的可扩展性优于改进 KACA 算法，更适合处理大数据集的 k-匿名化问题。

但由于题 2 要求给出一个隐藏数据量最少的方案，故优先考虑隐藏数据比例，选择 k=2（2-重保护），c=1（聚类数为 1，即直接采用改进 KACA 算法）为模型参数，求解附件中的两组多元数据隐藏数据量最少的方案。

6.3 结果展示

以下为多元 1 和多元 2 的最小隐藏数据比例及部分结果，详细结果见附录。

多元 1　隐藏数据比例：42.69%

表 8　多元 1 部分结果

0	*	*	*	2	0	4	0
0	*	*	*	2	0	4	0
0	*	*	*	*	*	3	3
0	*	*	*	*	*	3	3
1	0	1	4	0	*	*	*
1	0	1	4	0	*	*	*
1	0	2	*	*	3	*	*
1	0	2	*	*	3	*	*
1	0	4	*	*	4	*	2
1	0	4	*	*	4	*	2

多元 2　隐藏数据比例：63.83%

表 9　多元 2 部分结果

0	*	*	*	*	*	8	*	1	*	1	*	2	*	*	*
0	*	*	*	*	*	8	*	1	*	1	*	2	*	*	*
0	*	*	*	*	*	*	9	7	*	*	*	7	*	*	0
0	*	*	*	*	*	*	9	7	*	*	*	7	*	*	0
1	2	4	*	*	0	*	*	*	0	*	*	*	6	*	*
1	2	4	*	*	0	*	*	*	0	*	*	*	6	*	*
1	6	*	*	*	*	*	*	*	*	*	9	*	0	*	6

1	6	*	*	*	*	*	*	*	*	*	9	*	0	*	6
1	8	*	*	*	*	*	*	*	8	0	7	*	7	*	*
1	8	*	*	*	*	*	*	*	8	0	7	*	7	*	*

七、 问题三模型建立与求解

7.1 模型建立

题 3 要求对附件数据的 p-重保护问题，分别给出 p=3,5,8 情形的最佳隐藏方案。由于附件中数据规模较小，运行时间较短，故本文优先考虑隐藏数据比例尽量小，采用改进 KACA 算法模型.

7.2 模型求解

对附件数据中的二元 1、二元 2、多元 1 和多元 2 数据分别采用参数 k=3,5,8，给出其 p=3,5,8 情形的最佳隐藏方案。

7.3 结果展示

以下为二元 1 在 p=3,5,8 情形的最佳隐藏数据比例及部分结果，详细结果见附录：

二元 1 p=3　隐藏数据比例：18.33%

表 10　二元 1 p=3 部分结果

0	0	1	*	1	1
0	0	1	*	1	1
0	0	1	*	1	1
0	0	1	*	1	1
0	1	1	0	1	*
0	1	1	0	1	*
0	1	1	0	1	*
0	1	1	0	1	*

二元 1 p=5　隐藏数据比例：48.33%

表 11　二元 1 p=5 部分结果

1	1	1	*	0	0
1	1	1	*	0	0
1	1	1	*	0	0
1	1	1	*	0	0
1	1	1	*	0	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0

二元 1 p=8　隐藏数据比例：71.46%

表 12　二元 1 p=8 部分结果

*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0

0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0

八、 问题四模型建立与求解

8.1 情形 1

8.1.1 情形 1 模型建立

由于多元数据表中限定了隐藏模式，多元数据在隐藏时只允许单隐或者全隐。故本文在改进 KACA 算法的基础上修改得到新模型进行求解，得到隐藏方案。

单隐或者全隐的数据隐藏技术算法
/*D 表示将要被聚类的数据集，k 表示 k-匿名的限制。*/
begin

(1) 由数据集 D 中生成初始等价类，等价类中各个元组在准标识符上值相等；
(2) 将初始等价类中元组个数大于等于 k 的等价类放入最终匿名表中，不进入循环；
(3) 循环，直到不存在元组个数小于 k 的等价类：
  ① 随机选择一个大小小于 k 的等价类 C；
  ② 计算 C 与其他所有等价类并为一类并匿名后的隐藏数据增量；
  ③ 找到与 C 泛化后隐藏数据增量最少的等价类 C'；
  ④ 将 C 和 C' 并为一类，并以 * 匿名 C 和 C' 中属性的值不同的属性。
  ⑤ 如果上一步得到的新的等价类中匿名的属性的个数超过 1，则将这一等价类的全部属性以 * 匿名。
(4) 循环结束，返回匿名表。

end

8.1.2 情形 1 模型求解

由于要进行对模型的检验，故数据采用题中所给的示例, 结果如下：

情形 1 示例数据：

表 13　情形 1 的示例数据

1	2	1	0
1	2	1	0
0	0	1	0
0	0	3	0
1	1	1	0
1	1	1	2
1	2	1	4
1	3	1	5

情形 1 示例数据正确结果：

表 14　情形 1 示例数据的正确结果

1	2	1	0
1	2	1	0
0	0	*	0
0	0	*	0
1	1	1	*
1	1	1	*
*	*	*	*
*	*	*	*

情形 1 模型结果：　隐藏数据比例：37.5%

表 15　情形 1 限定模式的模型结果

1	2	1	0
1	2	1	0
0	0	*	0
0	0	*	0

1	1	1	*
1	1	1	*
*	*	*	*
*	*	*	*

结论： 据上，情形 1 模型求解示例数据的结果与正确结果完全吻合，该模型符合情形 1 要求。

8.2 情形 2

8.2.1 情形 2 模型建立

题目要求多元高维数据中需要选出若干维度的数据全发布以确保有效性，并对其余维度进行以隐私保护目的的隐藏技术。故本文在改进 KACA 算法的基础上，加入一个决策函数筛选出需要全发布的数据，得到新的准标识符，再使用改进 KACA 算法求解得到隐藏方案。

全发布决策函数
/* 全发布的维数的数据应该在该维度上的数据离散性较大 */
begin

(1) 在数据集 D 中统计每一个维度的众数；
(2) 在数据集 D 中统计该维度数据中的值与该维度众数不相同的数量作为该维度的差异度 d；
(3) 对每一维度的差异度进行升维，从差异度 d 变为二维差异度 [d,d]，便于后续聚类；
(4) 使用 k-means 聚类，将每一维度的二维差异度 [d,d] 进行聚类，得到两个聚类；
(5) 将得到两个聚类中差异度值较大的类的所有维度的数据全发布，另一类的所有维度作为准标识符；

end

8.2.2 情形 2 模型求解

由于要进行对模型的检验，故数据采用题中所给的示例，结果如下：

情形 2 示例数据：

表 16　情形 2 示例数据

0	1	2	1	2	5
0	1	2	3	4	3
1	2	0	1	2	4
0	2	0	1	1	2
1	3	1	3	3	1
1	3	1	1	3	0
2	3	2	1	5	5
2	3	1	1	0	6
0	3	2	4	6	2
0	2	2	3	6	3

情形 2 示例数据正确结果：

表 17　情形 2 示例数据正确结果

0	1	2	*	2	5
0	1	2	*	4	3
*	2	0	1	2	4
*	2	0	1	1	2
1	3	1	*	3	1
1	3	1	*	3	0
2	3	*	1	5	5
2	3	*	1	0	6
0	*	2	*	6	2
0	*	2	*	6	3

情形 2 模型结果：　隐藏数据比例：20%

表 18　情形 2 模型结果

0	1	2	*	2	5
0	1	2	*	4	3
*	2	0	1	2	4
*	2	0	1	1	2
1	3	1	*	3	1
1	3	1	*	3	0
2	3	*	1	5	5
2	3	*	1	0	6
0	*	2	*	6	2
0	*	2	*	6	3

结论： 据上，情形 2 模型求解示例数据的结果与正确结果完全吻合，该模型符合情形 2 要求。

九、 模型的评价

9.1 模型的优点

1. 使用两个等价类合并后隐藏数据增量来代替原 KACA 算法的两个等价类的距离，使解得的隐藏方案的隐藏数据量更小。

2. 对于数据规模大的发布数据，采用 c-modes-改进 KACA 算法，在隐藏数据量相差不大的情况下使得算法效率大大提高。

9.2 模型的不足

1. 题四中，两个情形下提供的数据隐藏技术，除了题中所给示例外，没有数据支撑。

参考文献

[1] 杨静, 王超, 张健沛. 基于敏感属性熵的微聚集算法 [J]. 电子学报,2014(7):1327-1337. DOI:10.3969/j.issn.0372-2112.2014.07.013.

[2] Aghdamm M R S,Sonehara N. Efficient local recoding anonymization for datasets without attribute hierarchical structure[C]// International Conference on Cyber Security,2013.

[3] Li Jiuyong, Wong Raymond Chi-Wing, Fu Ada Wai-Chee, etal. Achieving k-anonymity by clustering in attribute hierarchical structure // LNCS 4081: Proc of the 8th Int Conf on Data Warehousing and Knowledge Discovery. Berlin: Springer, 2006: 405-416.

[4] 于娟, 韩建民, 郭腾芳, 等. 基于聚类的高效 k-匿名化算法 [J]. 计算机研究与发展,2009,46(z2):480-486.

[5] Chin Chuangcheng, Tsai Chiehyuan, A k-anonymity clustering method for effective data privacy preservation // LNCS 4632: Proc of the 3rd Int Conf on Advanced Data Mining and Applications.Berlin:Springer,2007:89-99.

[6] Huang Zhexue. Extensions to the k -means algorithm for clustering large data sets with categorical values. Data Mining K nowledge Discovery, 1998,2(3):283-304.

十、 附录

本文代码均使用 Python3.9.7 进行编译。

10.1 Python 实现问题一、二

```
import math
import time
import os
import pandas as pd

c = 1 # select a c as cluster count
k = 8 # select a k as k-anonymity
dataLength = 0 # length of data, e.g. length of (1,2,3,5) is 4


# ------------------------------------------------------
#              functions
# ------------------------------------------------------

# calculate distance between list1 and list2 ignoring *
def calculateDis(list1, list2):
    global dataLength
    sum = 0
    for i in range(len(list1)):
        # sum += not (list1[i] == list2[i])
        if list1[i] == -1 or list2[i] == -1:
            sum += 1
        else:
            sum += (dataLength+1) * (not list1[i] == list2[i])
    return sum


# if list with * can contain the another
def isIn(list1, list2):
    flag = True
    for i in range(len(list1)):
        if list1[i] == -1 or list2[i] == -1:
            continue
        else:
            if not list1[i] == list2[i]:
                flag = False
    return flag


# list[i] == -1 <==> list[i] = "*"
# return 0: equals
#       the higher the more different
def calculateSimilarity(list1, list2):
    sum = 0
```

```
    for i in range(len(list1)):
        sum += (not list1[i] == list2[i])
    return sum


def calculateLoss(list1, l1Count, list2, l2Count):
    global dataLength
    loss = 0
    for i in range(len(list1)):
        loss += (not list1[i] == list2[i])
    return loss * (l1Count + l2Count) - (list1.count(-1)+list2.count(-1))


# calculate instance and its count
#
# return as {(0,0,0,0,0): 1}
def countInstance(array):
    l = dict()
    for item in array:
        i = tuple(item)
        if not i in l.keys():
            l[i] = 1
        else:
            l[i] += 1
    return l


# input two datarows and replace position where
#   two datarows differs with value -1
# e.g. input (0,0,0,0,1) and (0,0,0,0,0), then replace
#   value at index -1 with -1
#
# return tuple like (0,0,0,0,-1)
def distortList(list1, list2):
    l = list()
    for i in range(len(list1)):
        l.append(list1[i] if list1[i] == list2[i] else '*')
    return tuple(l)


def KACA2(group):
    global groupAppendix
    instanceWithCount = countInstance(group)
    while True:
        cSet = ()
        c2Set = ()
        for insKey, insValue in instanceWithCount.items():
            if insValue >= k:
                continue
            minKey = ()
            minValue = 0
```

```
            minLoss = 1e8
            for insKey2, insValue2 in instanceWithCount.items():
                if insKey2 == insKey:
                    continue
                loss = calculateLoss(insKey, insValue, insKey2, insValue2)
                # loss = calculateSimilarity(insKey, insKey2)
                if loss < minLoss:
                    minLoss = loss
                    minValue = insValue2
                    minKey = insKey2

            if minValue == 1e8:
                print("[error] error in finding min key")
                exit(1)
            else:
                cSet = (insKey, insValue)
                c2Set = (minKey, minValue)
                break

        if len(cSet):
            del instanceWithCount[cSet[0]]
            del instanceWithCount[c2Set[0]]
            insKey3 = distortList(cSet[0], c2Set[0])
            if insKey3 not in instanceWithCount:
                instanceWithCount[insKey3] = cSet[1] + c2Set[1]
            else:
                instanceWithCount[insKey3] = instanceWithCount[insKey3] + cSet[1] + c2Set[1]

            # if instanceWithCount[insKey3] >= k:
            #     groupAppendix[insKey3] = instanceWithCount[insKey3]
            #     del instanceWithCount[insKey3]


        else:
            break

    return instanceWithCount


# translate original value into distorted one
# original data: dataframe
# instanceMap : list or tuple
# return      : dataframe
def distortOriginalValue(originData: pd.DataFrame, instanceMap: tuple):
    distortFrame = pd.DataFrame(columns=list(range(len(instanceMap[0]))))
    for i in range(len(originData)):
        dataRow = originData.loc[i, :]
        minDis = 1e8
        minMap = ()
        for insMap in instanceMap:
            dis = calculateDis(insMap, dataRow)
```

```
            if dis < minDis:
                minMap = insMap
                minDis = dis
        if minDis == 1e8:
            print("[error] error in finding closest distort map")
            exit(1)
        # distortFrame.append(list(minMap))
        for j in range(len(minMap)):
            if minMap[j] == -1:
                originData.loc[i, j] = "*"
            else:
                originData.loc[i, j] = minMap[j]
    return originData
# ------------------------------------------------------



start = time.perf_counter()
print("loading data...")
path = 'D:\桌面\校赛/B题附件2.xlsx'
sheetName = "多元2"


print("initializing data...")
file = pd.read_excel(path, sheet_name=sheetName, header=None)
print(file)
print("data initialized")


print("assembling data...")
group = dict()
rows = []
for i in range(len(file)):
    rows.append(tuple(file.loc[i, :].tolist()))
group = countInstance(rows)
for key, value in group.items():
    print(key, value)
groupAppendix = {insKey:insValue for insKey, insValue in group.items() if insValue >= k}
group = {insKey:insValue for insKey, insValue in group.items() if insValue < k}
dataLength = len(rows[0])
print("data assembled as:")
for key, value in group.items():
    print(key, value)


# using c-modes & KACA algorithm
# start c-modes
print("create initial random-selected c-modes group with c="+str(c))
# initialize cluster group
clusterGroup = []
modeGroup = []
```

```
for i in range(c):
    idx = math.floor(len(rows)/c) * i + math.floor(len(rows)/c/2)
    subgroup = rows[idx]
    clusterGroup.append([])
    modeGroup.append(tuple(subgroup))


print("start assembling...")
while True:
    flag = True
    # adding element to cluster group
    for datarow, count in group.items():
        minDis = 1e8
        minIdx = -1
        # select group to append
        for i in range(c):
            s = calculateDis(modeGroup[i], datarow)
            if minDis > s:
                minDis = s
                minIdx = i
        # append to group
        for cot in range(count):
            clusterGroup[minIdx].append(datarow)

    # calculate mode and select
    for i in range(c):
        subgroup = clusterGroup[i]
        maxMode = list()
        for j in range(dataLength):
            l = [d[j] for d in subgroup]
            maxMode.append(max(l, key=l.count))
        if not modeGroup[i] == tuple(maxMode):
            modeGroup[i] = tuple(maxMode)
            flag = False

    if flag:
        break
    else:
        for subgroup in clusterGroup:
            subgroup.clear()
print("c-mode assembling ended, results lies as:")
for i in range(c):
    print("mode as center:", modeGroup[i], "count:", len(clusterGroup[i]))


print("start KACA...")
# start KACA
allInstanceCount = dict()
for i in range(c):
    print("KACA part", str(i)+"/"+str(c), "starting...")
    subgroup = clusterGroup[i]
```

```
    instanceWithCount = KACA2(subgroup)

    print("values input:", len(subgroup), "output:", sum(instanceWithCount.values()))
    print("results:", instanceWithCount)
    for insKey, insValue in instanceWithCount.items():
        if not insKey in allInstanceCount.keys():
            allInstanceCount[insKey] = 0
        allInstanceCount[insKey] += insValue
    print("KACA part", str(i) + "/" + str(c), "finished")

print("KACA ends")
# append groupAppendix to the end
allInstanceCount.update(groupAppendix)
print("final result:")
print("pattern count:", len(allInstanceCount.keys()))
print("pattern results:")
distortedPosCount = 0
dataAllCount = file.size
for insKey, insValue in allInstanceCount.items():
    print( insKey, insValue)
    distortedPosCount += sum([1 for a in insKey if a == "*"]) * insValue
print("distorted position percent:", distortedPosCount/dataAllCount)
end = time.perf_counter()

print("运行耗时", end-start)


#print("start distorting original data...")
#distortedFrame = distortOriginalValue(file, tuple(allInstanceCount.keys()))
#print("distorting ends")
#print("results:")
#print(distortedFrame)


# print("saving results to file...")
# distortedFilePath = "".join(path.split('.')[:-1]) + "_distorted" + "." + path.split('.')[-1]
# if not os.path.exists(distortedFilePath):
#     with pd.ExcelWriter(distortedFilePath, mode='w') as writer:
#         distortedFrame.to_excel(writer, sheet_name=sheetName, index=False, header=False)
# else:
#     with pd.ExcelWriter(distortedFilePath, mode='a') as writer:
#         distortedFrame.to_excel(writer, sheet_name=sheetName, index=False, header=False)
# print("file saved!")
```

10.2 Python 实现问题三

```
# -*- coding: utf-8 -*-
import math
import time
import os
```

```
import pandas as pd

c = 1 # select a c as cluster count
k = 2 # select a k as k-anonymity
dataLength = 0 # length of data, e.g. length of (1,2,3,5) is 4


# ----------------------------------------------------
#              functions
# ----------------------------------------------------

# calculate distance between list1 and list2 ignoring *
def calculateDis(list1, list2):
    global dataLength
    sum = 0
    for i in range(len(list1)):
        # sum += not (list1[i] == list2[i])
        if list1[i] == -1 or list2[i] == -1:
            sum += 1
        else:
            sum += (dataLength+1) * (not list1[i] == list2[i])
    return sum


# if list with * can contain the another
def isIn(list1, list2):
    flag = True
    for i in range(len(list1)):
        if list1[i] == -1 or list2[i] == -1:
            continue
        else:
            if not list1[i] == list2[i]:
                flag = False
    return flag


# list[i] == -1 <==> list[i] = "*"
# return 0: equals
#       the higher the more different
def calculateSimilarity(list1, list2):
    sum = 0
    for i in range(len(list1)):
        sum += (not list1[i] == list2[i])
    return sum


def calculateLoss(list1, l1Count, list2, l2Count):
    global dataLength
    loss = 0
    for i in range(len(list1)):
        loss += (not list1[i] == list2[i])
```

```
    return loss * (l1Count + l2Count) - (list1.count(-1)+list2.count(-1))


# calculate instance and its count
#
# return as {(0,0,0,0,0): 1}
def countInstance(array):
    l = dict()
    for item in array:
        i = tuple(item)
        if not i in l.keys():
            l[i] = 1
        else:
            l[i] += 1
    return l


# input two datarows and replace position where
#  two datarows differs with value -1
# e.g. input (0,0,0,0,1) and (0,0,0,0,0), then replace
#  value at index -1 with -1
#
# return tuple like (0,0,0,0,-1)
def distortList(list1, list2):
    l = list()
    for i in range(len(list1)):
        l.append(list1[i] if list1[i] == list2[i] else '*')
    return tuple(l)


def KACA2(group):
    global groupAppendix
    instanceWithCount = countInstance(group)
    while True:
        cSet = ()
        c2Set = ()
        for insKey, insValue in instanceWithCount.items():
            if insValue >= k:
                continue
            minKey = ()
            minValue = 0
            minLoss = 1e8
            for insKey2, insValue2 in instanceWithCount.items():
                if insKey2 == insKey:
                    continue
                loss = calculateLoss(insKey, insValue, insKey2, insValue2)
                # loss = calculateSimilarity(insKey, insKey2)
                if loss < minLoss:
                    minLoss = loss
                    minValue = insValue2
                    minKey = insKey2
```

```
            if minValue == 1e8:
                print("[error] error in finding min key")
                exit(1)
            else:
                cSet = (insKey, insValue)
                c2Set = (minKey, minValue)
                break

        if len(cSet):
            del instanceWithCount[cSet[0]]
            del instanceWithCount[c2Set[0]]
            insKey3 = distortList(cSet[0], c2Set[0])
            if insKey3 not in instanceWithCount:
                instanceWithCount[insKey3] = cSet[1] + c2Set[1]
            else:
                instanceWithCount[insKey3] = instanceWithCount[insKey3] + cSet[1] + c2Set[1]

            # if instanceWithCount[insKey3] >= k:
            #     groupAppendix[insKey3] = instanceWithCount[insKey3]
            #     del instanceWithCount[insKey3]


        else:
            break

    return instanceWithCount


# translate original value into distorted one
# original data: dataframe
# instanceMap : list or tuple
# return      : dataframe
def distortOriginalValue(originData: pd.DataFrame, instanceMap: tuple):
    distortFrame = pd.DataFrame(columns=list(range(len(instanceMap[0]))))
    for i in range(len(originData)):
        dataRow = originData.loc[i, :]
        minDis = 1e8
        minMap = ()
        for insMap in instanceMap:
            dis = calculateDis(insMap, dataRow)
            if dis < minDis:
                minMap = insMap
                minDis = dis
        if minDis == 1e8:
            print("[error] error in finding closest distort map")
            exit(1)
        # distortFrame.append(list(minMap))
        for j in range(len(minMap)):
            if minMap[j] == -1:
                originData.loc[i, j] = "*"
```

```
            else:
                originData.loc[i, j] = minMap[j]
    return originData
# ----------------------------------------------------



start = time.perf_counter()
print("loading data...")
path = 'D:\桌面\校赛/0.xlsx'
sheetName = "Sheet1"


print("initializing data...")
file = pd.read_excel(path, sheet_name=sheetName, header=None)
print(file)
print("data initialized")


print("assembling data...")
group = dict()
rows = []
for i in range(len(file)):
    rows.append(tuple(file.loc[i, :].tolist()))
group = countInstance(rows)
for key, value in group.items():
    print(key, value)
groupAppendix = {insKey:insValue for insKey, insValue in group.items() if insValue >= k}
group = {insKey:insValue for insKey, insValue in group.items() if insValue < k}
dataLength = len(rows[0])
print("data assembled as:")
for key, value in group.items():
    print(key, value)


# using c-modes & KACA algorithm
# start c-modes
print("create initial random-selected c-modes group with c="+str(c))
# initialize cluster group
clusterGroup = []
modeGroup = []
for i in range(c):
    idx = math.floor(len(rows)/c) * i + math.floor(len(rows)/c/2)
    subgroup = rows[idx]
    clusterGroup.append([])
    modeGroup.append(tuple(subgroup))


print("start assembling...")
while True:
    flag = True
```

```
    # adding element to cluster group
    for datarow, count in group.items():
        minDis = 1e8
        minIdx = -1
        # select group to append
        for i in range(c):
            s = calculateDis(modeGroup[i], datarow)
            if minDis > s:
                minDis = s
                minIdx = i
        # append to group
        for cot in range(count):
            clusterGroup[minIdx].append(datarow)

    # calculate mode and select
    for i in range(c):
        subgroup = clusterGroup[i]
        maxMode = list()
        for j in range(dataLength):
            l = [d[j] for d in subgroup]
            maxMode.append(max(l, key=l.count))
        if not modeGroup[i] == tuple(maxMode):
            modeGroup[i] = tuple(maxMode)
            flag = False

    if flag:
        break
    else:
        for subgroup in clusterGroup:
            subgroup.clear()
print("c-mode assembling ended, results lies as:")
for i in range(c):
    print("mode as center:", modeGroup[i], "count:", len(clusterGroup[i]))


print("start KACA...")
# start KACA
allInstanceCount = dict()
for i in range(c):
    print("KACA part", str(i)+"/"+str(c), "starting...")
    subgroup = clusterGroup[i]
    instanceWithCount = KACA2(subgroup)

    print("values input:", len(subgroup), "output:", sum(instanceWithCount.values()))
    print("results:", instanceWithCount)
    for insKey, insValue in instanceWithCount.items():
        if not insKey in allInstanceCount.keys():
            allInstanceCount[insKey] = 0
        allInstanceCount[insKey] += insValue
    print("KACA part", str(i) + "/" + str(c), "finished")
```

```
print("KACA ends")
# append groupAppendix to the end
allInstanceCount.update(groupAppendix)
print("final result:")
print("pattern count:", len(allInstanceCount.keys()))
print("pattern results:")
distortedPosCount = 0
dataAllCount = file.size
for insKey, insValue in allInstanceCount.items():
    print( insKey, insValue)
    distortedPosCount += sum([1 for a in insKey if a == "*"]) * insValue
print("distorted position percent:", distortedPosCount/dataAllCount)
end = time.perf_counter()

print("运行耗时", end-start)


#print("start distorting original data...")
#distortedFrame = distortOriginalValue(file, tuple(allInstanceCount.keys()))
#print("distorting ends")
#print("results:")
#print(distortedFrame)


# print("saving results to file...")
# distortedFilePath = "".join(path.split('.')[:-1]) + "_distorted" + "." + path.split('.')[-1]
# if not os.path.exists(distortedFilePath):
#     with pd.ExcelWriter(distortedFilePath, mode='w') as writer:
#         distortedFrame.to_excel(writer, sheet_name=sheetName, index=False, header=False)
# else:
#     with pd.ExcelWriter(distortedFilePath, mode='a') as writer:
#         distortedFrame.to_excel(writer, sheet_name=sheetName, index=False, header=False)
# print("file saved!")
```

10.3 Python 实现问题四情形 1

```
import math
import time
import os
import pandas as pd

c = 1 # select a c as cluster count
k = 2 # select a k as k-anonymity
dataLength = 0 # length of data, e.g. length of (1,2,3,5) is 4


# ------------------------------------------------
#              functions
# ------------------------------------------------
```

```
# calculate distance between list1 and list2 ignoring *
def calculateDis(list1, list2):
    global dataLength
    sum = 0
    for i in range(len(list1)):
        # sum += not (list1[i] == list2[i])
        if list1[i] == -1 or list2[i] == -1:
            sum += 1
        else:
            sum += (dataLength+1) * (not list1[i] == list2[i])
    return sum


# if list with * can contain the another
def isIn(list1, list2):
    flag = True
    for i in range(len(list1)):
        if list1[i] == -1 or list2[i] == -1:
            continue
        else:
            if not list1[i] == list2[i]:
                flag = False
    return flag


# list[i] == -1 <==> list[i] = "*"
# return 0: equals
#       the higher the more different
def calculateSimilarity(list1, list2):
    sum = 0
    for i in range(len(list1)):
        sum += (not list1[i] == list2[i])
    return sum


def calculateLoss(list1, l1Count, list2, l2Count):
    global dataLength
    loss = 0
    for i in range(len(list1)):
        loss += (not list1[i] == list2[i])
    return loss * (l1Count + l2Count) - (list1.count(-1)+list2.count(-1))


# calculate instance and its count
#
# return as {(0,0,0,0,0): 1}
def countInstance(array):
    l = dict()
    for item in array:
        i = tuple(item)
        if not i in l.keys():
```

```
            l[i] = 1
        else:
            l[i] += 1
    return l


# input two datarows and replace position where
#   two datarows differs with value -1
# e.g. input (0,0,0,0,1) and (0,0,0,0,0), then replace
#   value at index -1 with -1
#
# return tuple like (0,0,0,0,-1)
def distortList(list1, list2):
    l = list()
    if  calculateSimilarity(list1, list2)<=1:
        for i in range(len(list1)):
            l.append(list1[i] if list1[i] == list2[i] else '*')
        return tuple(l)
    else:
        for i in range(len(list1)):
            l.append('*')
        return tuple(l)


def KACA2(group):
    global groupAppendix
    instanceWithCount = countInstance(group)
    while True:
        cSet = ()
        c2Set = ()
        for insKey, insValue in instanceWithCount.items():
            if insValue >= k:
                continue
            minKey = ()
            minValue = 0
            minLoss = 1e8
            for insKey2, insValue2 in instanceWithCount.items():
                if insKey2 == insKey:
                    continue
                loss = calculateLoss(insKey, insValue, insKey2, insValue2)
                # loss = calculateSimilarity(insKey, insKey2)
                if loss < minLoss:
                    minLoss = loss
                    minValue = insValue2
                    minKey = insKey2

            if minValue == 1e8:
                print("[error] error in finding min key")
                exit(1)
            else:
                cSet = (insKey, insValue)
```

```
                c2Set = (minKey, minValue)
                break

        if len(cSet):
            del instanceWithCount[cSet[0]]
            del instanceWithCount[c2Set[0]]
            insKey3 = distortList(cSet[0], c2Set[0])
            if insKey3 not in instanceWithCount:
                instanceWithCount[insKey3] = cSet[1] + c2Set[1]
            else:
                instanceWithCount[insKey3] = instanceWithCount[insKey3] + cSet[1] + c2Set[1]

            # if instanceWithCount[insKey3] >= k:
            #     groupAppendix[insKey3] = instanceWithCount[insKey3]
            #     del instanceWithCount[insKey3]


        else:
            break

    return instanceWithCount


# translate original value into distorted one
# original data: dataframe
# instanceMap : list or tuple
# return      : dataframe
def distortOriginalValue(originData: pd.DataFrame, instanceMap: tuple):
    distortFrame = pd.DataFrame(columns=list(range(len(instanceMap[0]))))
    for i in range(len(originData)):
        dataRow = originData.loc[i, :]
        minDis = 1e8
        minMap = ()
        for insMap in instanceMap:
            dis = calculateDis(insMap, dataRow)
            if dis < minDis:
                minMap = insMap
                minDis = dis
        if minDis == 1e8:
            print("[error] error in finding closest distort map")
            exit(1)
        # distortFrame.append(list(minMap))
        for j in range(len(minMap)):
            if minMap[j] == -1:
                originData.loc[i, j] = "*"
            else:
                originData.loc[i, j] = minMap[j]
    return originData
# ----------------------------------------------------
```

```
start = time.perf_counter()
print("loading data...")
path = 'D:\桌面\校赛/00.xlsx'
sheetName = "多元1"


print("initializing data...")
file = pd.read_excel(path, header=None)
print(file)
print("data initialized")


print("assembling data...")
group = dict()
rows = []
for i in range(len(file)):
    rows.append(tuple(file.loc[i, :].tolist()))
group = countInstance(rows)
for key, value in group.items():
    print(key, value)
groupAppendix = {insKey:insValue for insKey, insValue in group.items() if insValue >= k}
group = {insKey:insValue for insKey, insValue in group.items() if insValue < k}
dataLength = len(rows[0])
print("data assembled as:")
for key, value in group.items():
    print(key, value)


# using c-modes & KACA algorithm
# start c-modes
print("create initial random-selected c-modes group with c="+str(c))
# initialize cluster group
clusterGroup = []
modeGroup = []
for i in range(c):
    idx = math.floor(len(rows)/c) * i + math.floor(len(rows)/c/2)
    subgroup = rows[idx]
    clusterGroup.append([])
    modeGroup.append(tuple(subgroup))


print("start assembling...")
while True:
    flag = True
    # adding element to cluster group
    for datarow, count in group.items():
        minDis = 1e8
        minIdx = -1
        # select group to append
        for i in range(c):
```

```
            s = calculateDis(modeGroup[i], datarow)
            if minDis > s:
                minDis = s
                minIdx = i
        # append to group
        for cot in range(count):
            clusterGroup[minIdx].append(datarow)

    # calculate mode and select
    for i in range(c):
        subgroup = clusterGroup[i]
        maxMode = list()
        for j in range(dataLength):
            l = [d[j] for d in subgroup]
            maxMode.append(max(l, key=l.count))
        if not modeGroup[i] == tuple(maxMode):
            modeGroup[i] = tuple(maxMode)
            flag = False

    if flag:
        break
    else:
        for subgroup in clusterGroup:
            subgroup.clear()
print("c-mode assembling ended, results lies as:")
for i in range(c):
    print("mode as center:", modeGroup[i], "count:", len(clusterGroup[i]))


print("start KACA...")
# start KACA
allInstanceCount = dict()
for i in range(c):
    print("KACA part", str(i)+"/"+str(c), "starting...")
    subgroup = clusterGroup[i]
    instanceWithCount = KACA2(subgroup)

    print("values input:", len(subgroup), "output:", sum(instanceWithCount.values()))
    print("results:", instanceWithCount)
    for insKey, insValue in instanceWithCount.items():
        if not insKey in allInstanceCount.keys():
            allInstanceCount[insKey] = 0
        allInstanceCount[insKey] += insValue
    print("KACA part", str(i) + "/" + str(c), "finished")

print("KACA ends")
# append groupAppendix to the end
allInstanceCount.update(groupAppendix)
print("final result:")
print("pattern count:", len(allInstanceCount.keys()))
print("pattern results:")
```

```
distortedPosCount = 0
dataAllCount = file.size
for insKey, insValue in allInstanceCount.items():
    print( insKey, insValue)
    distortedPosCount += sum([1 for a in insKey if a == "*"]) * insValue
print("distorted position percent:", distortedPosCount/dataAllCount)
end = time.perf_counter()

print("运行耗时", end-start)


#print("start distorting original data...")
#distortedFrame = distortOriginalValue(file, tuple(allInstanceCount.keys()))
#print("distorting ends")
#print("results:")
#print(distortedFrame)


# print("saving results to file...")
# distortedFilePath = "".join(path.split('.')[:-1]) + "_distorted" + "." + path.split('.')[-1]
# if not os.path.exists(distortedFilePath):
#     with pd.ExcelWriter(distortedFilePath, mode='w') as writer:
#         distortedFrame.to_excel(writer, sheet_name=sheetName, index=False, header=False)
# else:
#     with pd.ExcelWriter(distortedFilePath, mode='a') as writer:
#         distortedFrame.to_excel(writer, sheet_name=sheetName, index=False, header=False)
# print("file saved!")
```

10.4 Python 实现问题四情形 2

```
import math
import os
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans

c = 1 # select a c as cluster count
k = 2 # select a k as k-anonymity
dataLength = 0 # length of data, e.g. length of (1,2,3,5) is 4


# ------------------------------------------------------
#              functions
# ------------------------------------------------------

# calculate distance between list1 and list2 ignoring *
def calculateDis(list1, list2):
    global dataLength
    sum = 0
    for i in range(len(list1)):
```

```
        # sum += not (list1[i] == list2[i])
        if list1[i] == -1 or list2[i] == -1:
            sum += 1
        else:
            sum += (dataLength+1) * (not list1[i] == list2[i])
    return sum


# if list with * can contain the another
def isIn(list1, list2):
    flag = True
    for i in range(len(list1)):
        if list1[i] == -1 or list2[i] == -1:
            continue
        else:
            if not list1[i] == list2[i]:
                flag = False
    return flag


# list[i] == -1 <==> list[i] = "*"
# return 0: equals
#       the higher the more different
def calculateSimilarity(list1, list2):
    sum = 0
    for i in range(len(list1)):
        sum += (not list1[i] == list2[i])
    return sum


def calculateLoss(list1, l1Count, list2, l2Count):
    global dataLength
    loss = 0
    for i in range(len(list1)):
        loss += (not list1[i] == list2[i])
    return loss * (l1Count + l2Count) - (list1.count(-1)+list2.count(-1))


# calculate instance and its count
#
# return as {(0,0,0,0,0): 1}
def countInstance(array):
    l = dict()
    for item in array:
        i = tuple(item)
        if not i in l.keys():
            l[i] = 1
        else:
            l[i] += 1
    return l
```

```
# input two datarows and replace position where
#  two datarows differs with value -1
# e.g. input (0,0,0,0,1) and (0,0,0,0,0), then replace
#  value at index -1 with -1
#
# return tuple like (0,0,0,0,-1)
def distortList(list1, list2):
    l = list()
    for i in range(len(list1)):
        l.append(list1[i] if list1[i] == list2[i] else "*")
    return tuple(l)


def KACA2(group):
    global groupAppendix
    instanceWithCount = countInstance(group)
    while True:
        cSet = ()
        c2Set = ()
        for insKey, insValue in instanceWithCount.items():
            if insValue >= k:
                continue
            minKey = ()
            minValue = 0
            minLoss = 1e8
            for insKey2, insValue2 in instanceWithCount.items():
                if insKey2 == insKey:
                    continue
                loss = calculateLoss(insKey, insValue, insKey2, insValue2)
                # loss = calculateSimilarity(insKey, insKey2)
                if loss < minLoss:
                    minLoss = loss
                    minValue = insValue2
                    minKey = insKey2

            if minValue == 1e8:
                print("[error] error in finding min key")
                exit(1)
            else:
                cSet = (insKey, insValue)
                c2Set = (minKey, minValue)
                break

        if len(cSet):
            del instanceWithCount[cSet[0]]
            del instanceWithCount[c2Set[0]]
            insKey3 = distortList(cSet[0], c2Set[0])
            if insKey3 not in instanceWithCount:
                instanceWithCount[insKey3] = cSet[1] + c2Set[1]
            else:
```

```
            instanceWithCount[insKey3] = instanceWithCount[insKey3] + cSet[1] + c2Set[1]

        # if instanceWithCount[insKey3] >= k:
        #     groupAppendix[insKey3] = instanceWithCount[insKey3]
        #     del instanceWithCount[insKey3]


    else:
        break

  return instanceWithCount
# ----------------------------------------------------




print("loading data...")
path = 'D:\桌面\校赛/000.xlsx'
sheetName ='Sheet1'


print("initializing data...")
file = pd.read_excel(path, sheet_name=sheetName, header=None)
print(file)
dataLength = len(file.columns)
print("data initialized")


print("start to select rows to be deleted...")
print("getting mode of each column...")
modeColumn = list()
for i in range(dataLength):
  col = file.loc[:, i].tolist()
  modeColumn.append(max(col, key=col.count))
print("modes:", modeColumn)
print("getting distance of each column...")
disList = list()
for i in range(dataLength):
  col = file.loc[:, i].tolist()
  disList.append(sum([1 for c in col if not c == modeColumn[i]]))
print("distance of each column:", disList)
# select point using kmeans
print("selecting column to display full using k-means...")
disList3 = list()
for dis in disList:
  disList3.append([dis, dis])
kmeans = KMeans(n_clusters=2).fit(np.array(disList3))
selectedGroup = 1 if kmeans.cluster_centers_[1][0] > kmeans.cluster_centers_[0][0] else 0
selectedIdxes: list = [idx for idx in range(len(disList)) if kmeans.labels_[idx] ==
   selectedGroup]
selectedIdxes.sort()
```

```
print("did select column to display!")
print("column selection: distance", [disList[idx] for idx in selectedIdxes], "at index",
    selectedIdxes)
print("column selected will be ignored and temporarily removed afterward")


print("assembling data...")
group = dict()
rows = []
for i in range(len(file)):
    row = list()
    for j in range(dataLength):
        if not j in selectedIdxes:
            row.append(file.loc[i, j])
    rows.append(tuple(row))


group = countInstance(rows)
groupAppendix = {insKey: insValue for insKey, insValue in group.items() if insValue >= k}
group = {insKey: insValue for insKey, insValue in group.items() if insValue < k}


dataLength = len(rows[0])
print("data assembled as:")
for key, value in group.items():
    print(key, value)


# using c-modes & KACA algorithm
# start c-modes
print("create initial random-selected c-modes group with c="+str(c))
# initialize cluster group
clusterGroup = []
modeGroup = []
for i in range(c):
    idx = math.floor(len(rows)/c) * i + math.floor(len(rows)/c/2)
    subgroup = rows[idx]
    clusterGroup.append([])
    modeGroup.append(tuple(subgroup))


print("start assembling...")
while True:
    flag = True
    # adding element to cluster group
    for datarow, count in group.items():
        minDis = 1e8
        minIdx = -1
        # select group to append
        for i in range(c):
            s = calculateDis(modeGroup[i], datarow)
```

```
            if minDis > s:
                minDis = s
                minIdx = i
        # append to group
        for cot in range(count):
            clusterGroup[minIdx].append(datarow)

    # calculate mode and select
    for i in range(c):
        subgroup = clusterGroup[i]
        maxMode = list()
        for j in range(dataLength):
            l = [d[j] for d in subgroup]
            maxMode.append(max(l, key=l.count))
        if not modeGroup[i] == tuple(maxMode):
            modeGroup[i] = tuple(maxMode)
            flag = False

    if flag:
        break
    else:
        for subgroup in clusterGroup:
            subgroup.clear()
print("c-mode assembling ended, results lies as:")
for i in range(c):
    print("mode as center:", modeGroup[i], "count:", len(clusterGroup[i]))


print("start KACA...")
# start KACA
allInstanceCount = dict()
for i in range(c):
    print("KACA part", str(i)+"/"+str(c), "starting...")
    subgroup = clusterGroup[i]
    instanceWithCount = KACA2(subgroup)

    print("values input:", len(subgroup), "output:", sum(instanceWithCount.values()))
    print("results:", instanceWithCount)
    for insKey, insValue in instanceWithCount.items():
        if not insKey in allInstanceCount.keys():
            allInstanceCount[insKey] = 0
        allInstanceCount[insKey] += insValue
    print("KACA part", str(i) + "/" + str(c), "finished")
print("KACA ends")
# append groupAppendix to the end
allInstanceCount.update(groupAppendix)
print("final result:")
print("pattern count:", len(allInstanceCount.keys()))
print("pattern results:")
distortedPosCount = 0
dataAllCount = file.size
```

```
for insKey, insValue in allInstanceCount.items():
    print("  pattern:", insKey, ", count:", insValue)
    distortedPosCount += sum([1 for a in insKey if a == -1]) * insValue
print("distorted position percent:", distortedPosCount/dataAllCount)


print("start distorting original data and retrieve temporarily data...")
# removedData = list()
# for idx in selectedIdxes:
#     removedData.append(file.loc[:, idx].tolist())
# file = file.drop(columns=selectedIdxes)
# distortedFrame = distortOriginalValue(file, tuple(allInstanceCount.keys()))
# print(distortedFrame)
# exit()
# for i in range(len(selectedIdxes)):
#     file.insert(0, column=selectedIdxes[i], value=removedData[i])
distortedFrame = pd.DataFrame(data=None, columns=list(range(dataLength)), dtype=int)
for rowIdx in range(len(file)):
    row = file.loc[rowIdx, :].tolist()
    rowDistorted = list()
    # temporarily remove data
    for s in range(len(selectedIdxes)):
        del row[selectedIdxes[len(selectedIdxes) - s - 1]]
    # searching closest pattern
    minDis = 1e8
    minMap = ()
    for insMap in allInstanceCount.keys():
        dis = calculateDis(row, insMap)
        if dis < minDis:
            minDis = dis
            minMap = insMap
    if minDis == 1e8:
        print("[error] error in distorting original data")
        exit(1)
    minMap = list(minMap)
    # adding back removed data
    for idx in selectedIdxes:
        minMap.insert(idx, file.loc[rowIdx, idx])
    # appending row to dataframe(distortedFrame)
    # distortedFrame = distortedFrame.append(minMap,ignore_index=True)
    # distortedFrame.loc[rowIdx, :] = minMap
    for j in range(len(minMap)):
        distortedFrame.loc[rowIdx, j] = minMap[j]
print("distorting ends")
print("results:")
print(distortedFrame)

'''
print("saving results to file...")
distortedFilePath = "".join(path.split('.')[:-1]) + "_partial_full_display" + "." +
    path.split('.')[-1]
```

```
sheetName2 = sheetName
if not os.path.exists(distortedFilePath):
    with pd.ExcelWriter(distortedFilePath, mode='w') as writer:
        distortedFrame.to_excel(writer, sheet_name=sheetName2, index=False, header=False)
else:
    with pd.ExcelWriter(distortedFilePath, mode='a') as writer:
        distortedFrame.to_excel(writer, sheet_name=sheetName2, index=False, header=False)
print("file saved!")
'''
```

10.5 问题一的二元数组隐藏方案

表 19　二元 1 隐藏方案

0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	0	1
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	1
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	1	0
0	0	1	1	1	0
0	0	1	1	1	0
0	0	1	1	1	0
0	0	1	1	1	1
0	0	1	1	1	1
0	1	0	1	1	1
0	1	0	1	1	1

0	1	1	0	0	1
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	0	1	0
0	1	1	0	1	1
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	0	0
0	1	1	1	0	1
0	1	1	1	0	1
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	1
0	1	1	1	1	1
0	*	0	1	1	0
0	*	0	1	1	0
1	0	0	0	*	0
1	0	0	0	*	0
1	0	0	1	0	1
1	0	0	1	0	1
1	0	0	1	*	0
1	0	0	1	*	0
1	0	0	*	1	1
1	0	0	*	1	1
1	0	1	0	0	*
1	0	1	0	0	*
1	0	1	0	1	0
1	0	1	0	1	0
1	0	1	1	0	1
1	0	1	1	0	1

1	0	1	1	0	1
1	0	1	1	1	0
1	0	1	1	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	0	1	1
1	1	0	0	1	1
1	1	0	0	*	0
1	1	0	0	*	0
1	1	0	1	0	0
1	1	0	1	0	0
1	1	0	1	0	1
1	1	0	1	0	1
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	0	1	0
1	1	1	0	1	1
1	1	1	0	1	1
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	*	*
1	1	1	1	*	*
*	*	0	0	0	1
*	*	0	0	0	1

表 20　二元 2 隐藏方案

0	0	0	0	0	0	0	1	0	1	*	0
0	0	0	0	0	0	0	1	0	1	*	0
0	0	0	0	0	1	1	1	1	1	*	1
0	0	0	0	0	1	1	1	1	1	*	1
0	0	0	0	0	*	0	0	1	1	1	0
0	0	0	0	0	*	0	0	1	1	1	0
0	0	0	0	1	0	*	1	1	*	*	1
0	0	0	0	1	0	*	1	1	*	*	1
0	0	0	0	1	*	0	0	1	*	1	0
0	0	0	0	1	*	0	0	1	*	1	0
0	0	0	0	*	0	1	0	0	0	*	1
0	0	0	0	*	0	1	0	0	0	*	1
0	0	0	0	*	0	*	1	1	0	1	0
0	0	0	0	*	0	*	1	1	0	1	0
0	0	0	1	0	*	1	1	0	0	*	0
0	0	0	1	0	*	1	1	0	0	*	0
0	0	0	1	1	*	*	1	0	1	1	0
0	0	0	1	1	*	*	1	0	1	1	0
0	0	0	1	*	1	0	0	1	0	0	*
0	0	0	1	*	1	0	0	1	0	0	*
0	0	0	1	*	1	1	1	1	0	0	1
0	0	0	1	*	1	1	1	1	0	0	1
0	0	0	1	*	*	1	0	0	1	0	0
0	0	0	1	*	*	1	0	0	1	0	0
0	0	0	*	1	*	0	1	0	0	1	0
0	0	0	*	1	*	0	1	0	0	1	0
0	0	0	*	1	*	1	0	1	*	1	0
0	0	0	*	1	*	1	0	1	*	1	0
0	0	1	0	0	1	1	1	0	*	1	1
0	0	1	0	0	1	1	1	0	*	1	1

0	0	1	0	0	*	*	1	1	0	1	0
0	0	1	0	0	*	*	1	1	0	1	0
0	0	1	0	1	0	1	0	1	1	1	*
0	0	1	0	1	0	1	0	1	1	1	*
0	0	1	0	*	0	0	*	0	1	0	1
0	0	1	0	*	0	0	*	0	1	0	1
0	0	1	1	0	0	0	0	1	1	0	1
0	0	1	1	0	0	0	0	1	1	0	1
0	0	1	1	0	0	0	0	*	1	0	*
0	0	1	1	0	0	0	0	*	1	0	*
0	0	1	1	0	*	1	1	*	*	1	1
0	0	1	1	0	*	1	1	*	*	1	1
0	0	1	1	1	1	1	1	*	*	0	*
0	0	1	1	1	1	1	1	*	*	0	*
0	0	1	1	1	1	*	1	1	1	0	1
0	0	1	1	1	1	*	1	1	1	0	1
0	0	1	*	0	1	*	0	1	*	0	1
0	0	1	*	0	1	*	0	1	*	0	1
0	0	*	1	0	0	1	1	1	1	*	0
0	0	*	1	0	0	1	1	1	1	*	0
0	1	0	0	0	0	1	1	1	*	0	*
0	1	0	0	0	0	1	1	1	*	0	*
0	1	0	0	0	1	*	*	1	1	*	0
0	1	0	0	0	1	*	*	1	1	*	0
0	1	0	0	1	0	1	1	1	1	0	1
0	1	0	0	1	0	1	1	1	1	0	1
0	1	0	0	*	1	*	1	1	1	0	1
0	1	0	0	*	1	*	1	1	1	0	1
0	1	0	*	1	1	1	1	*	1	0	*
0	1	0	*	1	1	1	1	*	1	0	*

0	1	0	*	*	0	0	0	0	0	0	1
0	1	0	*	*	0	0	0	0	0	0	1
0	1	1	0	1	0	0	1	0	*	*	0
0	1	1	0	1	0	0	1	0	*	*	0
0	1	1	0	*	1	0	1	0	1	1	1
0	1	1	0	*	1	0	1	0	1	1	1
0	1	1	0	*	*	1	0	0	*	0	1
0	1	1	0	*	*	1	0	0	*	0	1
0	1	1	1	0	0	1	*	1	*	0	1
0	1	1	1	0	0	1	*	1	*	0	1
0	1	1	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	0	0	0
0	1	*	0	0	0	0	0	*	1	1	1
0	1	*	0	0	0	0	0	*	1	1	1
0	1	*	1	0	0	1	*	0	1	1	1
0	1	*	1	0	0	1	*	0	1	1	1
0	1	*	1	1	0	1	0	0	*	0	0
0	1	*	1	1	0	1	0	0	*	0	0
0	1	*	1	1	0	1	0	1	*	*	0
0	1	*	1	1	0	1	0	1	*	*	0
0	*	0	1	0	0	0	0	1	1	0	1
0	*	0	1	0	0	0	0	1	1	0	1
0	*	1	1	0	0	*	1	1	1	1	0
0	*	1	1	0	0	*	1	1	1	1	0
0	*	1	1	1	1	1	0	1	0	*	1
0	*	1	1	1	1	1	0	1	0	*	1
0	*	1	1	*	1	0	0	0	*	0	0
0	*	1	1	*	1	0	0	0	*	0	0
1	0	0	0	0	1	1	*	1	0	*	1
1	0	0	0	0	1	1	*	1	0	*	1

1	0	0	0	*	0	0	1	0	*	0	0
1	0	0	0	*	0	0	1	0	*	0	0
1	0	0	1	0	0	0	1	1	0	1	0
1	0	0	1	0	0	0	1	1	0	1	0
1	0	0	1	1	1	0	*	0	1	0	*
1	0	0	1	1	1	0	*	0	1	0	*
1	0	0	1	1	*	1	0	*	0	0	1
1	0	0	1	1	*	1	0	*	0	0	1
1	0	0	1	*	0	1	0	0	1	*	1
1	0	0	1	*	0	1	0	0	1	*	1
1	0	1	0	0	0	0	1	0	1	*	*
1	0	1	0	0	0	0	1	0	1	*	*
1	0	1	0	0	1	*	*	0	0	0	1
1	0	1	0	0	1	*	*	0	0	0	1
1	0	1	0	1	0	0	0	1	*	*	1
1	0	1	0	1	0	0	0	1	*	*	1
1	0	1	0	1	*	1	1	1	*	*	1
1	0	1	0	1	*	1	1	1	*	*	1
1	0	1	1	0	1	0	0	1	*	1	*
1	0	1	1	0	1	0	0	1	*	1	*
1	0	1	1	1	1	*	0	*	0	*	1
1	0	1	1	1	1	*	0	*	0	*	1
1	0	1	*	0	1	1	0	0	1	1	*
1	0	1	*	0	1	1	0	0	1	1	*
1	0	1	*	1	1	*	1	1	1	1	0
1	0	1	*	1	1	*	1	1	1	1	0
1	0	*	0	0	1	1	0	1	1	1	0
1	0	*	0	0	1	1	0	1	1	1	0
1	0	*	1	0	*	1	0	0	0	0	0
1	0	*	1	0	*	1	0	0	0	0	0

1	0	*	1	0	*	1	0	1	1	0	1
1	0	*	1	0	*	1	0	1	1	0	1
1	0	*	1	1	1	*	1	0	1	0	*
1	0	*	1	1	1	*	1	0	1	0	*
1	0	*	*	0	0	0	0	1	1	0	0
1	0	*	*	0	0	0	0	1	1	0	0
1	0	*	*	1	1	1	0	0	0	1	1
1	0	*	*	1	1	1	0	0	0	1	1
1	1	0	0	0	0	0	*	1	1	*	1
1	1	0	0	0	0	0	*	1	1	*	1
1	1	0	0	0	*	0	*	0	0	1	0
1	1	0	0	0	*	0	*	0	0	1	0
1	1	0	0	1	0	0	0	*	1	0	*
1	1	0	0	1	0	0	0	*	1	0	*
1	1	0	0	1	*	0	0	1	0	1	1
1	1	0	0	1	*	0	0	1	0	1	1
1	1	0	0	*	0	*	0	1	0	0	0
1	1	0	0	*	0	*	0	1	0	0	0
1	1	0	1	0	0	0	1	1	1	1	0
1	1	0	1	0	0	0	1	1	1	1	0
1	1	0	1	0	0	0	*	0	*	0	1
1	1	0	1	0	0	0	*	0	*	0	1
1	1	0	1	*	0	1	1	1	1	*	*
1	1	0	1	*	0	1	1	1	1	*	*
1	1	0	1	*	0	1	*	0	1	0	1
1	1	0	1	*	0	1	*	0	1	0	1
1	1	0	*	*	0	0	*	1	1	1	0
1	1	0	*	*	0	0	*	1	1	1	0
1	1	0	*	*	0	1	0	0	0	*	0
1	1	0	*	*	0	1	0	0	0	*	0

1	1	0	*	*	1	0	1	1	0	0	1
1	1	0	*	*	1	0	1	1	0	0	1
1	1	1	0	0	1	0	0	*	1	0	1
1	1	1	0	0	1	0	0	*	1	0	1
1	1	1	0	1	1	*	*	0	0	0	*
1	1	1	0	1	1	*	*	0	0	0	*
1	1	1	0	*	1	0	1	*	1	0	0
1	1	1	0	*	1	0	1	*	1	0	0
1	1	1	1	0	*	0	*	0	0	0	1
1	1	1	1	0	*	0	*	0	0	0	1
1	1	1	1	0	*	1	*	0	*	1	1
1	1	1	1	0	*	1	*	0	*	1	1
1	1	1	*	1	0	0	1	0	1	*	1
1	1	1	*	1	0	0	1	0	1	*	1
1	1	1	*	*	1	*	0	1	1	*	0
1	1	1	*	*	1	*	0	1	1	*	0
1	1	*	0	1	0	0	1	1	1	0	1
1	1	*	0	1	0	0	1	1	1	0	1
1	1	*	*	*	1	1	0	0	0	*	0
1	1	*	*	*	1	1	0	0	0	*	0
1	*	1	0	1	0	0	0	0	0	1	0
1	*	1	0	1	0	0	0	0	0	1	0
1	*	1	1	0	0	*	1	0	1	1	0
1	*	1	1	0	0	*	1	0	1	1	0
1	*	1	1	1	0	1	1	1	1	1	*
1	*	1	1	1	0	1	1	1	1	1	*
1	*	1	*	0	1	0	1	*	1	1	1
1	*	1	*	0	1	0	1	*	1	1	1
*	0	0	1	1	0	0	0	*	1	1	0
*	0	0	1	1	0	0	0	*	1	1	0

*	0	0	1	*	1	0	0	0	1	1	1
*	0	0	1	*	1	0	0	0	1	1	1
*	0	1	0	0	1	1	0	0	0	1	0
*	0	1	0	0	1	1	0	0	0	1	0
*	1	0	0	0	*	1	0	0	1	0	0
*	1	0	0	0	*	1	0	0	1	0	0
*	1	0	1	0	0	0	0	0	1	1	0
*	1	0	1	0	0	0	0	0	1	1	0
*	1	0	1	1	*	0	0	0	0	1	1
*	1	0	1	1	*	0	0	0	0	1	1
*	1	1	1	1	0	1	*	0	0	0	1
*	1	1	1	1	0	1	*	0	0	0	1
*	1	*	0	1	1	1	1	1	0	0	1
*	1	*	0	1	1	1	1	1	0	0	1
*	1	*	1	0	1	1	1	*	1	0	1
*	1	*	1	0	1	1	1	*	1	0	1
*	1	*	*	1	1	0	0	0	1	1	0
*	1	*	*	1	1	0	0	0	1	1	0
*	*	1	1	1	1	0	1	0	0	1	0
*	*	1	1	1	1	0	1	0	0	1	0

10.6 问题二的多元数组隐藏方案

表 21　多元 1 隐藏方案

0	0	*	2	*	*	*	2
0	0	*	2	*	*	*	2
0	0	*	3	2	*	*	*
0	0	*	3	2	*	*	*
0	0	*	*	4	*	2	1
0	0	*	*	4	*	2	1

0	1	1	*	*	2	4	*
0	1	1	*	*	2	4	*
0	2	*	1	*	4	*	2
0	2	*	1	*	4	*	2
0	2	*	*	1	4	*	0
0	2	*	*	1	4	*	0
0	4	1	*	2	*	4	*
0	4	1	*	2	*	4	*
0	4	1	*	2	*	4	*
0	*	0	*	1	*	2	4
0	*	0	*	1	*	2	4
0	*	4	2	*	*	4	*
0	*	4	2	*	*	4	*
0	*	*	4	1	*	1	*
0	*	*	4	1	*	1	*
0	*	*	*	2	0	4	0
0	*	*	*	2	0	4	0
0	*	*	*	*	*	3	3
0	*	*	*	*	*	3	3
1	0	1	4	0	*	*	*
1	0	1	4	0	*	*	*
1	0	2	*	*	3	*	*
1	0	2	*	*	3	*	*
1	0	4	*	*	4	*	2
1	0	4	*	*	4	*	2
1	3	2	*	2	1	4	*
1	3	2	*	2	1	4	*
1	3	*	0	4	2	1	1
1	3	*	0	4	2	1	1
1	3	*	*	0	*	1	*

1	3	*	*	0	*	1	*
1	4	*	*	1	4	4	*
1	4	*	*	1	4	4	*
1	*	1	4	4	*	*	1
1	*	1	4	4	*	*	1
1	*	4	4	*	*	0	4
1	*	4	4	*	*	0	4
1	*	*	*	3	*	0	*
1	*	*	*	3	*	0	*
1	*	*	*	*	4	0	0
1	*	*	*	*	4	0	0
1	*	*	*	*	*	2	0
1	*	*	*	*	*	2	0
2	0	0	*	1	*	4	*
2	0	0	*	1	*	4	*
2	0	4	*	*	0	2	1
2	0	4	*	*	0	2	1
2	2	2	1	*	0	4	*
2	2	2	1	*	0	4	*
2	2	*	*	4	4	3	0
2	2	*	*	4	4	3	0
2	*	1	*	4	1	3	*
2	*	1	*	4	1	3	*
2	*	2	0	*	*	*	3
2	*	3	1	2	*	0	*
2	*	3	1	2	*	0	*
2	*	3	*	0	2	2	*
2	*	3	*	0	2	2	*
2	*	*	0	2	1	3	*
2	*	*	0	2	1	3	*

2	*	*	1	3	*	1	*
2	*	*	1	3	*	1	*
2	*	*	*	0	4	4	3
2	*	*	*	0	4	4	3
2	*	*	*	1	1	2	2
2	*	*	*	1	1	2	2
3	0	4	4	*	*	*	2
3	0	4	4	*	*	*	2
3	0	*	0	2	*	*	4
3	0	*	0	2	*	*	4
3	2	0	*	*	3	*	1
3	2	0	*	*	3	*	1
3	3	*	1	4	*	4	2
3	3	*	1	4	*	4	2
3	4	*	1	2	*	*	3
3	4	*	1	2	*	*	3
3	*	1	0	2	1	*	*
3	*	1	0	2	1	*	*
3	*	1	*	1	4	2	*
3	*	1	*	1	4	2	*
3	*	2	*	1	*	*	2
3	*	2	*	1	*	*	2
3	*	3	1	*	*	1	*
3	*	3	1	*	*	1	*
3	*	*	4	*	0	2	3
3	*	*	4	*	0	2	3
3	*	*	*	0	*	1	*
3	*	*	*	0	*	1	*
4	0	*	1	*	*	*	3
4	0	*	1	*	*	*	3

4	1	1	*	*	1	1	2
4	1	1	*	*	1	1	2
4	1	*	*	*	1	0	3
4	1	*	*	*	1	0	3
4	2	*	0	0	*	*	*
4	2	*	0	0	*	*	*
4	3	3	*	*	3	*	0
4	3	3	*	*	3	*	0
4	4	2	*	*	*	1	0
4	4	2	*	*	*	1	0
4	*	0	3	*	0	*	1
4	*	0	3	*	0	*	1
4	*	0	*	*	*	4	1
4	*	0	*	*	*	4	1
4	*	1	0	3	*	*	1
4	*	1	0	3	*	*	1
4	*	1	3	*	3	4	0
4	*	1	3	*	3	4	0
4	*	2	2	4	0	*	4
4	*	2	2	4	0	*	4
4	*	3	1	1	3	*	*
4	*	3	1	1	3	*	*
4	*	*	4	1	2	2	0
4	*	*	4	1	2	2	0
4	*	*	*	4	*	4	*
4	*	*	*	4	*	4	*
*	0	1	1	*	0	1	4
*	0	1	1	*	0	1	4
*	0	1	1	*	0	1	4
*	0	1	2	*	*	1	4

*	0	1	2	*	*	1	4
*	0	3	*	1	*	4	*
*	0	3	*	1	*	4	*
*	0	*	4	1	4	*	3
*	0	*	4	1	4	*	3
*	0	*	*	2	0	3	3
*	0	*	*	2	0	3	3
*	0	*	*	2	0	3	3
*	1	3	0	*	1	4	*
*	1	3	0	*	1	4	*
*	1	*	0	*	3	*	3
*	1	*	0	*	3	*	3
*	1	*	1	*	*	2	*
*	1	*	1	*	*	2	*
*	1	*	*	*	1	3	0
*	1	*	*	*	1	3	0
*	2	2	4	*	*	1	1
*	2	2	4	*	*	1	1
*	2	4	0	*	2	*	*
*	2	4	0	*	2	*	*
*	2	4	0	*	2	*	*
*	2	*	0	3	1	4	*
*	2	*	0	3	1	4	*
*	2	*	2	*	4	*	*
*	2	*	2	*	4	*	*
*	2	*	2	*	*	2	3
*	2	*	2	*	*	2	3
*	3	3	*	*	0	*	2
*	3	3	*	*	0	*	2
*	3	*	1	1	3	1	3

*	3	*	1	1	3	1	3
*	3	*	3	4	*	*	0
*	3	*	3	4	*	*	0
*	4	3	4	0	*	*	2
*	4	3	4	0	*	*	2
*	4	*	2	3	*	0	0
*	4	*	2	3	*	0	0
*	4	*	3	3	4	*	*
*	4	*	3	3	4	*	*
*	4	*	3	3	4	*	*
*	*	0	0	2	2	*	1
*	*	0	0	2	2	*	1
*	*	0	1	*	2	3	*
*	*	0	1	*	2	3	*
*	*	0	*	0	1	0	*
*	*	0	*	0	1	0	*
*	*	1	1	3	*	4	3
*	*	1	1	3	*	4	3
*	*	1	1	4	3	*	*
*	*	1	1	4	3	*	*
*	*	1	2	0	*	2	*
*	*	1	2	0	*	2	*
*	*	2	1	*	2	*	*
*	*	2	1	*	2	*	*
*	*	2	2	*	*	*	1
*	*	2	2	*	*	*	1
*	*	3	2	*	*	4	2
*	*	3	2	*	*	4	2
*	*	3	*	1	*	2	3
*	*	3	*	1	*	2	3

*	*	4	2	2	*	1	*
*	*	4	2	2	*	1	*
*	*	4	4	*	*	*	0
*	*	4	4	*	*	*	0
*	*	4	4	*	*	*	0
*	*	4	*	2	*	3	1
*	*	*	0	0	*	2	0
*	*	*	0	0	*	2	0
*	*	*	0	3	4	3	*
*	*	*	0	3	4	3	*
*	*	*	2	2	*	0	3
*	*	*	2	2	*	0	3
*	*	*	3	*	2	3	4
*	*	*	3	*	2	3	4

表 22　多元 2 隐藏方案

0	2	*	5	6	1	*	*	*	*	*	*	*	*	*	4
0	2	*	5	6	1	*	*	*	*	*	*	*	*	*	4
0	4	*	2	*	*	*	*	*	*	*	*	8	3	*	*
0	4	*	2	*	*	*	*	*	*	*	*	8	3	*	*
0	4	*	*	*	*	5	8	*	*	9	*	*	*	*	*
0	4	*	*	*	*	5	8	*	*	9	*	*	*	*	*
0	6	*	0	*	*	*	6	*	*	*	9	5	9	2	*
0	6	*	0	*	*	*	6	*	*	*	9	5	9	2	*
0	8	4	*	8	*	5	9	*	*	*	*	*	*	*	*
0	8	4	*	8	*	5	9	*	*	*	*	*	*	*	*
0	*	8	*	9	8	*	*	3	*	*	8	*	*	*	*
0	*	8	*	9	8	*	*	3	*	*	8	*	*	*	*
0	*	*	0	*	*	*	*	*	6	*	3	*	0	2	*

0	*	*	0	*	*	*	*	*	6	*	3	*	0	2	*
0	*	*	1	*	*	*	*	*	*	2	7	*	5	*	3
0	*	*	1	*	*	*	*	*	*	2	7	*	5	*	3
0	*	*	3	*	*	*	*	0	7	*	5	*	*	2	*
0	*	*	3	*	*	*	*	0	7	*	5	*	*	2	*
0	*	*	5	9	*	*	*	*	*	0	*	7	*	*	*
0	*	*	5	9	*	*	*	*	*	0	*	7	*	*	*
0	*	*	*	3	5	*	*	4	*	*	0	*	*	1	4
0	*	*	*	3	5	*	*	4	*	*	0	*	*	1	4
0	*	*	*	*	6	3	*	0	*	*	7	*	*	5	*
0	*	*	*	*	6	3	*	0	*	*	7	*	*	5	*
0	*	*	*	*	*	8	*	1	*	1	*	2	*	*	*
0	*	*	*	*	*	8	*	1	*	1	*	2	*	*	*
0	*	*	*	*	*	*	9	7	*	*	*	7	*	*	0
0	*	*	*	*	*	*	9	7	*	*	*	7	*	*	0
1	2	4	*	*	0	*	*	*	0	*	*	*	6	*	*
1	2	4	*	*	0	*	*	*	0	*	*	*	6	*	*
1	6	*	*	*	*	*	*	*	*	*	9	*	0	*	6
1	6	*	*	*	*	*	*	*	*	*	9	*	0	*	6
1	8	*	*	*	*	*	*	*	8	0	7	*	7	*	*
1	8	*	*	*	*	*	*	*	8	0	7	*	7	*	*
1	9	*	4	6	*	*	6	*	*	*	*	2	*	2	*
1	9	*	4	6	*	*	6	*	*	*	*	2	*	2	*
1	9	*	*	1	*	7	*	*	*	*	8	*	*	4	*
1	9	*	*	1	*	7	*	*	*	*	8	*	*	4	*
1	*	3	*	*	*	4	*	5	*	5	*	*	*	*	*
1	*	3	*	*	*	4	*	5	*	5	*	*	*	*	*
1	*	4	*	4	*	5	0	*	*	*	*	*	4	*	*
1	*	4	*	4	*	5	0	*	*	*	*	*	4	*	*
1	*	*	6	4	*	*	2	8	*	*	7	*	*	*	6

1	*	*	6	4	*	*	2	8	*	*	7	*	*	*	6
1	*	*	9	7	7	*	*	*	*	*	5	*	*	5	*
1	*	*	9	7	7	*	*	*	*	*	5	*	*	5	*
1	*	*	*	*	3	*	0	*	*	0	*	6	*	*	*
1	*	*	*	*	3	*	0	*	*	0	*	6	*	*	*
1	*	*	*	*	7	*	7	6	1	5	*	*	*	*	2
1	*	*	*	*	7	*	7	6	1	5	*	*	*	*	2
1	*	*	*	*	9	1	*	*	6	9	*	*	*	*	6
1	*	*	*	*	9	1	*	*	6	9	*	*	*	*	6
1	*	*	*	*	*	*	*	*	1	*	*	*	*	4	0
1	*	*	*	*	*	*	*	*	1	*	*	*	*	4	0
2	6	*	*	*	*	*	7	*	*	*	*	1	*	7	*
2	6	*	*	*	*	*	7	*	*	*	*	1	*	7	*
2	*	0	*	*	*	*	*	5	2	*	*	*	*	2	7
2	*	0	*	*	*	*	*	5	2	*	*	*	*	2	7
2	*	*	7	5	9	3	*	*	*	3	*	*	*	*	*
2	*	*	7	5	9	3	*	*	*	3	*	*	*	*	*
2	*	*	7	*	3	*	2	*	*	*	*	3	*	*	*
2	*	*	7	*	3	*	2	*	*	*	*	3	*	*	*
2	*	*	*	*	*	5	*	*	2	*	0	*	*	1	*
2	*	*	*	*	*	5	*	*	2	*	0	*	*	1	*
3	2	*	*	*	*	*	6	*	*	*	0	*	*	6	*
3	2	*	*	*	*	*	6	*	*	*	0	*	*	6	*
3	3	*	*	1	1	*	*	*	*	*	8	*	*	*	*
3	3	*	*	1	1	*	*	*	*	*	8	*	*	*	*
3	4	*	*	*	3	*	*	9	*	*	6	*	*	*	6
3	4	*	*	*	3	*	*	9	*	*	6	*	*	*	6
3	5	*	*	5	6	*	4	6	*	*	3	2	*	*	*
3	5	*	*	5	6	*	4	6	*	*	3	2	*	*	*
3	*	1	*	*	3	2	*	*	6	2	*	*	8	2	*

3	*	1	*	*	3	2	*	*	6	2	*	*	8	2	*
3	*	1	*	*	3	*	*	4	*	*	4	6	*	*	*
3	*	1	*	*	3	*	*	4	*	*	4	6	*	*	*
3	*	3	*	*	*	*	*	*	*	5	*	*	*	*	*
3	*	3	*	*	*	*	*	*	*	5	*	*	*	*	*
3	*	3	*	*	*	*	*	*	*	5	*	*	*	*	*
3	*	9	*	9	*	*	*	6	8	*	*	5	*	*	*
3	*	9	*	9	*	*	*	6	8	*	*	5	*	*	*
3	*	*	1	*	6	3	5	*	9	*	*	8	*	*	*
3	*	*	1	*	6	3	5	*	9	*	*	8	*	*	*
3	*	*	2	9	*	3	*	*	9	*	*	4	*	*	*
3	*	*	2	9	*	3	*	*	9	*	*	4	*	*	*
3	*	*	2	*	*	4	*	*	*	*	*	*	4	*	*
3	*	*	2	*	*	4	*	*	*	*	*	*	4	*	*
3	*	*	3	*	0	*	9	*	*	*	7	7	*	5	*
3	*	*	3	*	0	*	9	*	*	*	7	7	*	5	*
3	*	*	*	0	*	*	*	*	*	*	4	*	*	2	*
3	*	*	*	0	*	*	*	*	*	*	4	*	*	2	*
3	*	*	*	*	1	5	*	0	*	0	0	*	*	5	*
3	*	*	*	*	1	5	*	0	*	0	0	*	*	5	*
4	0	*	1	*	*	*	6	*	*	5	3	*	*	*	4
4	0	*	1	*	*	*	6	*	*	5	3	*	*	*	4
4	2	*	*	7	*	5	*	*	*	*	*	6	*	0	4
4	2	*	*	7	*	5	*	*	*	*	*	6	*	0	4
4	2	*	*	*	9	*	*	*	*	2	*	6	*	*	0
4	2	*	*	*	9	*	*	*	*	2	*	6	*	*	0
4	4	*	7	*	*	2	*	*	*	6	*	*	2	0	*
4	4	*	7	*	*	2	*	*	*	6	*	*	2	0	*
4	5	6	*	2	*	*	*	4	4	*	*	6	*	3	*
4	5	6	*	2	*	*	*	4	4	*	*	6	*	3	*

4	6	*	*	*	5	4	2	*	*	*	4	*	*	*	*
4	6	*	*	*	5	4	2	*	*	*	4	*	*	*	*
4	7	*	5	*	*	*	*	8	*	*	*	*	*	9	*
4	7	*	5	*	*	*	*	8	*	*	*	*	*	9	*
4	7	*	*	*	*	1	*	4	*	9	8	*	*	*	9
4	7	*	*	*	*	1	*	4	*	9	8	*	*	*	9
4	8	0	*	9	*	*	*	*	*	*	*	2	*	*	*
4	8	0	*	9	*	*	*	*	*	*	*	2	*	*	*
4	8	*	5	*	*	*	7	*	*	*	*	2	8	*	*
4	8	*	5	*	*	*	7	*	*	*	*	2	8	*	*
4	*	0	*	*	*	8	1	*	4	8	*	*	*	*	*
4	*	0	*	*	*	8	1	*	4	8	*	*	*	*	*
4	*	3	*	*	*	4	*	*	*	*	*	*	*	*	*
4	*	3	*	*	*	4	*	*	*	*	*	*	*	*	*
4	*	3	*	*	*	8	*	4	*	*	*	8	4	*	*
4	*	3	*	*	*	8	*	4	*	*	*	8	4	*	*
4	*	8	*	*	1	0	*	*	*	*	3	*	*	*	*
4	*	8	*	*	1	0	*	*	*	*	3	*	*	*	*
4	*	9	*	7	*	*	*	3	*	*	*	*	6	0	*
4	*	9	*	7	*	*	*	3	*	*	*	*	6	0	*
4	*	*	3	4	*	*	*	0	*	*	*	6	*	7	*
4	*	*	3	4	*	*	*	0	*	*	*	6	*	7	*
4	*	*	5	*	7	3	*	*	8	*	*	2	*	*	*
4	*	*	5	*	7	3	*	*	8	*	*	2	*	*	*
4	*	*	5	*	*	1	7	*	*	*	*	0	5	7	*
4	*	*	5	*	*	1	7	*	*	*	*	0	5	7	*
4	*	*	*	*	5	0	*	*	*	*	1	*	*	*	*
4	*	*	*	*	5	0	*	*	*	*	1	*	*	*	*
4	*	*	*	*	*	*	9	*	*	1	*	*	*	3	*
4	*	*	*	*	*	*	9	*	*	1	*	*	*	3	*

4	*	*	*	*	*	*	*	3	0	*	*	2	9	*	7
4	*	*	*	*	*	*	*	3	0	*	*	2	9	*	7
5	0	*	*	*	6	*	*	*	7	0	8	1	*	2	*
5	0	*	*	*	6	*	*	*	7	0	8	1	*	2	*
5	2	*	*	*	9	4	*	7	*	2	*	*	*	4	*
5	2	*	*	*	9	4	*	7	*	2	*	*	*	4	*
5	3	*	*	7	*	*	*	*	0	*	6	*	*	*	8
5	3	*	*	7	*	*	*	*	0	*	6	*	*	*	8
5	5	*	*	4	2	*	*	*	*	*	9	*	*	*	5
5	5	*	*	4	2	*	*	*	*	*	9	*	*	*	5
5	6	*	2	*	2	*	*	*	*	*	9	3	*	*	1
5	6	*	2	*	2	*	*	*	*	*	9	3	*	*	1
5	6	*	*	*	5	*	*	*	*	4	*	*	*	4	*
5	6	*	*	*	5	*	*	*	*	4	*	*	*	4	*
5	8	3	*	8	*	*	8	*	*	*	*	*	1	7	*
5	8	3	*	8	*	*	8	*	*	*	*	*	1	7	*
5	*	0	*	1	*	*	*	4	8	*	7	*	*	3	*
5	*	0	*	1	*	*	*	4	8	*	7	*	*	3	*
5	*	0	*	3	*	*	*	*	*	8	*	*	*	3	5
5	*	0	*	3	*	*	*	*	*	8	*	*	*	3	5
5	*	1	1	7	*	*	*	*	6	*	*	1	*	*	*
5	*	1	1	7	*	*	*	*	6	*	*	1	*	*	*
5	*	7	2	*	6	8	*	2	*	*	*	*	*	*	*
5	*	7	2	*	6	8	*	2	*	*	*	*	*	*	*
5	*	9	*	8	3	1	2	*	1	*	*	*	*	1	*
5	*	9	*	8	3	1	2	*	1	*	*	*	*	1	*
5	*	*	8	5	8	*	5	*	*	*	*	*	9	*	*
5	*	*	8	5	8	*	5	*	*	*	*	*	9	*	*
5	*	*	*	1	2	*	8	*	*	*	*	*	*	0	*
5	*	*	*	1	2	*	8	*	*	*	*	*	*	0	*

5	*	*	*	3	*	7	*	*	0	*	6	*	3	*	*
5	*	*	*	3	*	7	*	*	0	*	6	*	3	*	*
5	*	*	*	7	*	*	5	*	*	*	1	2	*	*	*
5	*	*	*	7	*	*	5	*	*	*	1	2	*	*	*
6	1	3	*	*	*	2	9	*	4	*	*	8	*	*	5
6	1	3	*	*	*	2	9	*	4	*	*	8	*	*	5
6	1	*	*	*	*	8	*	*	*	2	*	*	*	3	*
6	1	*	*	*	*	8	*	*	*	2	*	*	*	3	*
6	3	*	*	7	*	0	*	*	*	4	6	*	*	9	*
6	3	*	*	7	*	0	*	*	*	4	6	*	*	9	*
6	5	2	*	0	*	0	*	*	*	*	*	*	8	*	*
6	5	2	*	0	*	0	*	*	*	*	*	*	8	*	*
6	7	0	*	*	*	*	*	*	4	*	*	*	4	0	*
6	7	0	*	*	*	*	*	*	4	*	*	*	4	0	*
6	8	*	6	6	*	*	*	*	*	4	*	3	*	*	7
6	8	*	6	6	*	*	*	*	*	4	*	3	*	*	7
6	*	0	7	6	3	*	*	4	*	*	*	*	9	*	*
6	*	0	7	6	3	*	*	4	*	*	*	*	9	*	*
6	*	0	*	*	*	*	6	*	*	7	0	*	*	7	*
6	*	0	*	*	*	*	6	*	*	7	0	*	*	7	*
6	*	9	7	*	*	4	*	*	3	9	7	*	*	*	*
6	*	9	7	*	*	4	*	*	3	9	7	*	*	*	*
6	*	*	3	*	1	*	0	*	3	*	8	*	*	*	*
6	*	*	3	*	1	*	0	*	3	*	8	*	*	*	*
6	*	*	4	*	5	6	*	*	4	*	*	*	7	*	*
6	*	*	4	*	5	6	*	*	4	*	*	*	7	*	*
6	*	*	4	*	9	1	*	2	4	6	*	*	*	*	*
6	*	*	4	*	9	1	*	2	4	6	*	*	*	*	*
6	*	*	6	*	*	*	*	*	*	8	*	*	*	6	0
6	*	*	6	*	*	*	*	*	*	8	*	*	*	6	0

6	*	*	*	*	7	8	4	*	*	*	*	1	6	*	*
6	*	*	*	*	7	8	4	*	*	*	*	1	6	*	*
6	*	*	*	*	8	*	4	*	*	*	*	3	0	0	*
6	*	*	*	*	8	*	4	*	*	*	*	3	0	0	*
6	*	*	*	*	*	7	3	*	4	*	*	*	8	*	*
6	*	*	*	*	*	7	3	*	4	*	*	*	8	*	*
6	*	*	*	*	*	7	3	*	*	2	*	0	*	*	0
6	*	*	*	*	*	7	3	*	*	2	*	0	*	*	0
6	*	*	*	*	*	*	1	*	*	8	*	1	8	*	7
6	*	*	*	*	*	*	1	*	*	8	*	1	8	*	7
7	4	*	2	*	*	*	*	*	4	*	1	*	*	*	*
7	4	*	2	*	*	*	*	*	4	*	1	*	*	*	*
7	5	*	*	*	*	8	*	*	*	7	*	6	*	*	*
7	5	*	*	*	*	8	*	*	*	7	*	6	*	*	*
7	7	7	5	*	*	*	*	*	*	*	7	*	*	*	5
7	7	7	5	*	*	*	*	*	*	*	7	*	*	*	5
7	8	*	*	5	*	*	*	*	*	2	*	5	7	*	*
7	8	*	*	5	*	*	*	*	*	2	*	5	7	*	*
7	9	*	3	*	*	*	8	*	*	2	0	*	*	*	*
7	9	*	3	*	*	*	8	*	*	2	0	*	*	*	*
7	*	0	*	4	*	*	*	*	*	*	5	9	*	*	*
7	*	0	*	4	*	*	*	*	*	*	5	9	*	*	*
7	*	5	*	*	6	*	*	*	1	*	*	*	*	0	*
7	*	5	*	*	6	*	*	*	1	*	*	*	*	0	*
7	*	7	*	*	3	9	*	*	*	9	8	*	*	*	5
7	*	7	*	*	3	9	*	*	*	9	8	*	*	*	5
7	*	*	0	*	4	0	*	*	*	*	9	4	*	*	*
7	*	*	0	*	4	0	*	*	*	*	9	4	*	*	*
7	*	*	4	*	*	3	*	*	*	*	3	*	*	*	*
7	*	*	4	*	*	3	*	*	*	*	3	*	*	*	*

7	*	*	5	*	*	6	*	*	*	3	*	*	9	*	*
7	*	*	5	*	*	6	*	*	*	3	*	*	9	*	*
7	*	*	*	7	9	*	*	5	*	*	*	2	9	*	*
7	*	*	*	7	9	*	*	5	*	*	*	2	9	*	*
7	*	*	*	9	*	0	9	*	*	5	*	*	*	*	8
7	*	*	*	9	*	0	9	*	*	5	*	*	*	*	8
7	*	*	*	*	0	*	2	6	*	9	*	7	*	*	8
7	*	*	*	*	0	*	2	6	*	9	*	7	*	*	8
7	*	*	*	*	0	*	*	5	3	*	0	2	*	*	*
7	*	*	*	*	0	*	*	5	3	*	0	2	*	*	*
7	*	*	*	*	2	*	*	2	8	*	6	1	*	*	*
7	*	*	*	*	2	*	*	2	8	*	6	1	*	*	*
7	*	*	*	*	3	*	0	*	*	0	0	*	5	*	*
7	*	*	*	*	3	*	0	*	*	0	0	*	5	*	*
7	*	*	*	*	6	0	*	1	0	*	*	8	*	*	6
7	*	*	*	*	6	0	*	1	0	*	*	8	*	*	6
7	*	*	*	*	6	1	1	*	*	*	2	*	*	*	3
7	*	*	*	*	6	1	1	*	*	*	2	*	*	*	3
7	*	*	*	*	7	*	*	8	3	1	*	1	*	*	0
7	*	*	*	*	7	*	*	8	3	1	*	1	*	*	0
7	*	*	*	*	9	*	2	1	*	*	6	9	*	*	*
7	*	*	*	*	9	*	2	1	*	*	6	9	*	*	*
7	*	*	*	*	*	2	9	0	*	3	*	*	9	*	*
7	*	*	*	*	*	2	9	0	*	3	*	*	9	*	*
7	*	*	*	*	*	*	4	*	2	6	*	*	*	1	*
7	*	*	*	*	*	*	4	*	2	6	*	*	*	1	*
8	0	*	*	*	9	*	*	*	7	*	*	*	1	*	*
8	0	*	*	*	9	*	*	*	7	*	*	*	1	*	*
8	1	*	3	*	*	*	*	*	*	9	*	*	*	6	*
8	1	*	3	*	*	*	*	*	*	9	*	*	*	6	*

8	3	*	7	3	*	*	6	*	*	*	*	*	*	0	*
8	3	*	7	3	*	*	6	*	*	*	*	*	*	0	*
8	3	*	*	4	3	*	0	5	*	*	*	*	9	*	*
8	3	*	*	4	3	*	0	5	*	*	*	*	9	*	*
8	6	4	*	6	2	*	*	*	*	*	*	1	*	*	*
8	6	4	*	6	2	*	*	*	*	*	*	1	*	*	*
8	8	*	1	3	*	*	*	*	*	2	*	*	*	2	*
8	8	*	1	3	*	*	*	*	*	2	*	*	*	2	*
8	*	1	*	5	*	*	*	*	8	*	1	6	*	3	*
8	*	1	*	5	*	*	*	*	8	*	1	6	*	3	*
8	*	1	*	*	*	*	7	2	*	8	*	*	1	*	*
8	*	1	*	*	*	*	7	2	*	8	*	*	1	*	*
8	*	3	1	*	*	*	*	*	*	1	*	1	*	8	*
8	*	3	1	*	*	*	*	*	*	1	*	1	*	8	*
8	*	7	*	4	*	7	*	*	*	0	*	*	*	7	1
8	*	7	*	4	*	7	*	*	*	0	*	*	*	7	1
8	*	8	*	*	9	*	*	*	*	*	2	8	*	*	*
8	*	8	*	*	9	*	*	*	*	*	2	8	*	*	*
8	*	*	6	*	2	8	*	3	*	*	*	*	*	3	*
8	*	*	6	*	2	8	*	3	*	*	*	*	*	3	*
8	*	*	6	*	*	*	*	2	*	*	0	*	1	*	*
8	*	*	6	*	*	*	*	2	*	*	0	*	1	*	*
8	*	*	*	6	*	3	*	8	6	4	*	*	*	*	*
8	*	*	*	6	*	3	*	8	6	4	*	*	*	*	*
8	*	*	*	9	*	*	*	*	0	*	*	*	9	1	0
8	*	*	*	9	*	*	*	*	0	*	*	*	9	1	0
8	*	*	*	*	4	*	6	*	*	*	*	*	0	*	4
8	*	*	*	*	4	*	6	*	*	*	*	*	0	*	4
8	*	*	*	*	*	0	6	*	*	*	7	*	*	*	7
8	*	*	*	*	*	0	6	*	*	*	7	*	*	*	7

9	0	3	*	7	*	*	*	4	7	*	*	6	*	*	*
9	0	3	*	7	*	*	*	4	7	*	*	6	*	*	*
9	2	3	7	*	*	3	*	*	*	*	*	*	*	*	*
9	2	3	7	*	*	3	*	*	*	*	*	*	*	*	*
9	3	*	*	1	*	*	6	*	0	*	2	*	*	*	*
9	3	*	*	1	*	*	6	*	0	*	2	*	*	*	*
9	4	7	*	*	*	*	9	5	*	*	5	*	*	*	*
9	4	7	*	*	*	*	9	5	*	*	5	*	*	*	*
9	4	*	*	*	*	8	1	*	*	*	4	1	*	*	*
9	4	*	*	*	*	8	1	*	*	*	4	1	*	*	*
9	5	*	2	*	*	*	0	*	*	*	*	*	2	3	*
9	5	*	2	*	*	*	0	*	*	*	*	*	2	3	*
9	5	*	*	2	*	*	*	9	*	3	8	*	*	*	*
9	5	*	*	2	*	*	*	9	*	3	8	*	*	*	*
9	6	*	*	3	*	*	1	*	4	*	*	*	*	*	8
9	6	*	*	3	*	*	1	*	4	*	*	*	*	*	8
9	8	*	0	*	*	8	*	*	7	7	*	*	*	*	7
9	8	*	0	*	*	8	*	*	7	7	*	*	*	*	7
9	*	2	5	*	9	1	*	*	4	5	*	*	*	*	*
9	*	2	5	*	9	1	*	*	4	5	*	*	*	*	*
9	*	9	*	5	*	*	*	*	*	8	7	*	*	*	*
9	*	9	*	5	*	*	*	*	*	8	7	*	*	*	*
9	*	*	1	*	4	*	*	*	*	*	*	*	5	7	9
9	*	*	1	*	4	*	*	*	*	*	*	*	5	7	9
9	*	*	1	*	*	*	*	3	*	*	*	*	3	*	*
9	*	*	1	*	*	*	*	3	*	*	*	*	3	*	*
9	*	*	1	*	*	*	*	*	3	*	*	*	*	*	9
9	*	*	1	*	*	*	*	*	3	*	*	*	*	*	9
9	*	*	6	*	*	*	*	*	7	*	2	*	1	*	3
9	*	*	6	*	*	*	*	*	7	*	2	*	1	*	3

9	*	*	*	7	*	4	8	7	*	*	*	*	*	4	*
9	*	*	*	7	*	4	8	7	*	*	*	*	*	4	*
9	*	*	*	9	*	*	*	*	9	*	*	7	*	4	*
9	*	*	*	9	*	*	*	*	9	*	*	7	*	4	*
9	*	*	*	*	4	1	*	7	*	8	6	*	*	*	0
9	*	*	*	*	4	1	*	7	*	8	6	*	*	*	0
9	*	*	*	*	6	*	3	*	*	1	7	*	*	*	5
9	*	*	*	*	6	*	3	*	*	1	7	*	*	*	5
9	*	*	*	*	7	*	*	7	*	8	2	*	*	*	1
9	*	*	*	*	7	*	*	7	*	8	2	*	*	*	1
*	0	6	*	6	*	2	*	3	*	*	*	*	5	*	*
*	0	6	*	6	*	2	*	3	*	*	*	*	5	*	*
*	0	*	0	3	*	5	*	*	*	*	3	7	*	*	*
*	0	*	0	3	*	5	*	*	*	*	3	7	*	*	*
*	0	*	3	9	*	8	*	6	5	*	*	*	*	*	*
*	0	*	3	9	*	8	*	6	5	*	*	*	*	*	*
*	0	*	8	*	2	*	7	*	7	4	*	*	*	*	*
*	0	*	8	*	2	*	7	*	7	4	*	*	*	*	*
*	0	*	*	2	*	9	*	9	*	*	7	9	*	*	2
*	0	*	*	2	*	9	*	9	*	*	7	9	*	*	2
*	0	*	*	9	1	*	*	*	*	*	9	*	1	8	*
*	0	*	*	9	1	*	*	*	*	*	9	*	1	8	*
*	0	*	*	*	2	5	*	*	*	*	8	7	*	9	*
*	0	*	*	*	2	5	*	*	*	*	8	7	*	9	*
*	0	*	*	*	4	*	*	*	*	7	*	*	0	*	*
*	0	*	*	*	4	*	*	*	*	7	*	*	0	*	*
*	0	*	*	*	*	*	*	*	4	*	4	*	*	*	*
*	0	*	*	*	*	*	*	*	4	*	4	*	*	*	*
*	1	3	*	*	6	4	*	*	*	*	*	9	*	4	*
*	1	3	*	*	6	4	*	*	*	*	*	9	*	4	*

*	1	4	*	*	3	0	*	4	*	*	*	*	*	*	3
*	1	4	*	*	3	0	*	4	*	*	*	*	*	*	3
*	1	4	*	*	*	*	*	*	4	1	*	5	*	*	*
*	1	4	*	*	*	*	*	*	4	1	*	5	*	*	*
*	1	7	*	*	*	4	*	*	*	*	3	3	1	*	*
*	1	7	*	*	*	4	*	*	*	*	3	3	1	*	*
*	1	*	*	*	8	6	*	*	*	*	*	0	9	*	*
*	1	*	*	*	8	6	*	*	*	*	*	0	9	*	*
*	1	*	*	*	*	2	*	*	2	*	*	6	*	*	0
*	1	*	*	*	*	2	*	*	2	*	*	6	*	*	0
*	1	*	*	*	*	*	3	*	*	*	*	*	0	*	*
*	1	*	*	*	*	*	3	*	*	*	*	*	0	*	*
*	2	8	1	*	1	*	*	0	8	*	*	*	*	7	*
*	2	8	1	*	1	*	*	0	8	*	*	*	*	7	*
*	2	8	6	*	*	*	7	*	4	*	*	*	*	*	0
*	2	8	6	*	*	*	7	*	4	*	*	*	*	*	0
*	2	9	3	*	*	*	4	*	*	*	7	*	*	4	*
*	2	9	3	*	*	*	4	*	*	*	7	*	*	4	*
*	2	*	0	*	6	*	*	7	*	*	7	4	*	*	*
*	2	*	0	*	6	*	*	7	*	*	7	4	*	*	*
*	2	*	1	*	4	*	4	2	4	*	*	*	*	6	*
*	2	*	1	*	4	*	4	2	4	*	*	*	*	6	*
*	2	*	2	*	*	*	*	*	*	4	6	*	*	8	2
*	2	*	2	*	*	*	*	*	*	4	6	*	*	8	2
*	2	*	5	*	*	8	*	7	9	*	*	*	4	*	*
*	2	*	5	*	*	8	*	7	9	*	*	*	4	*	*
*	2	*	*	6	3	*	*	*	*	*	*	*	*	9	3
*	2	*	*	6	3	*	*	*	*	*	*	*	*	9	3
*	2	*	*	7	*	*	*	*	1	*	*	3	*	*	3
*	2	*	*	7	*	*	*	*	1	*	*	3	*	*	3

*	2	*	*	9	*	9	*	1	*	*	8	*	*	6	*
*	2	*	*	9	*	9	*	1	*	*	8	*	*	6	*
*	3	0	*	*	*	*	*	6	*	8	*	*	*	*	*
*	3	0	*	*	*	*	*	6	*	8	*	*	*	*	*
*	3	7	*	*	7	4	3	*	*	*	*	*	1	*	*
*	3	7	*	*	7	4	3	*	*	*	*	*	1	*	*
*	3	*	0	*	0	0	9	*	*	*	*	*	4	*	*
*	3	*	0	*	0	0	9	*	*	*	*	*	4	*	*
*	3	*	1	*	3	7	*	*	4	*	0	*	*	*	9
*	3	*	1	*	3	7	*	*	4	*	0	*	*	*	9
*	3	*	2	8	5	*	6	*	*	*	*	*	4	*	7
*	3	*	2	8	5	*	6	*	*	*	*	*	4	*	7
*	3	*	4	*	*	2	*	*	1	4	*	*	*	*	*
*	3	*	4	*	*	2	*	*	1	4	*	*	*	*	*
*	3	*	8	*	*	5	*	*	*	6	*	*	7	8	*
*	3	*	8	*	*	5	*	*	*	6	*	*	7	8	*
*	3	*	*	*	1	5	*	*	*	8	4	*	4	*	*
*	3	*	*	*	1	5	*	*	*	8	4	*	4	*	*
*	3	*	*	*	2	7	5	*	8	*	*	*	6	*	*
*	3	*	*	*	2	7	5	*	8	*	*	*	6	*	*
*	3	*	*	*	6	*	*	0	*	9	*	*	*	9	8
*	3	*	*	*	6	*	*	0	*	9	*	*	*	9	8
*	3	*	*	*	*	*	3	4	0	*	*	2	4	*	*
*	3	*	*	*	*	*	3	4	0	*	*	2	4	*	*
*	4	0	*	*	8	*	*	*	4	*	*	*	9	5	6
*	4	0	*	*	8	*	*	*	4	*	*	*	9	5	6
*	4	3	5	*	*	*	*	9	*	*	*	5	*	*	*
*	4	3	5	*	*	*	*	9	*	*	*	5	*	*	*
*	4	4	*	*	6	*	4	*	7	*	5	*	*	*	*
*	4	4	*	*	6	*	4	*	7	*	5	*	*	*	*

*	4	4	*	*	*	*	*	9	*	*	8	*	7	*	*
*	4	4	*	*	*	*	*	9	*	*	8	*	7	*	*
*	4	6	9	*	6	*	*	*	2	*	*	2	*	*	*
*	4	6	9	*	6	*	*	*	2	*	*	2	*	*	*
*	4	7	9	*	1	5	*	*	*	*	2	9	*	*	*
*	4	7	9	*	1	5	*	*	*	*	2	9	*	*	*
*	4	7	*	*	6	*	*	*	*	*	*	2	1	*	9
*	4	7	*	*	6	*	*	*	*	*	*	2	1	*	9
*	4	8	*	*	*	*	5	8	*	*	9	*	*	*	*
*	4	8	*	*	*	*	5	8	*	*	9	*	*	*	*
*	4	*	9	*	8	*	6	*	*	3	*	*	*	*	*
*	4	*	9	*	8	*	6	*	*	3	*	*	*	*	*
*	4	*	*	8	2	*	7	0	*	7	*	*	*	*	*
*	4	*	*	8	2	*	7	0	*	7	*	*	*	*	*
*	4	*	*	8	7	*	0	*	*	*	*	*	*	1	*
*	4	*	*	8	7	*	0	*	*	*	*	*	*	1	*
*	4	*	*	*	3	*	*	7	*	*	3	*	7	*	*
*	4	*	*	*	3	*	*	7	*	*	3	*	7	*	*
*	4	*	*	*	*	8	*	4	7	*	6	*	*	*	*
*	4	*	*	*	*	8	*	4	7	*	6	*	*	*	*
*	4	*	*	*	*	9	*	*	0	*	*	4	5	2	9
*	4	*	*	*	*	9	*	*	0	*	*	4	5	2	9
*	4	*	*	*	*	*	*	4	*	5	*	*	7	9	1
*	4	*	*	*	*	*	*	4	*	5	*	*	7	9	1
*	4	*	*	*	*	*	*	*	*	6	0	1	*	2	*
*	4	*	*	*	*	*	*	*	*	6	0	1	*	2	*
*	5	2	*	*	*	*	5	*	*	*	7	*	0	*	0
*	5	2	*	*	*	*	5	*	*	*	7	*	0	*	0
*	5	7	*	*	8	*	0	*	0	*	*	*	1	*	*
*	5	7	*	*	8	*	0	*	0	*	*	*	1	*	*

*	5	7	*	*	*	4	*	*	*	6	*	*	*	*	*
*	5	7	*	*	*	4	*	*	*	6	*	*	*	*	*
*	5	9	*	3	2	*	*	7	*	*	*	*	0	*	3
*	5	9	*	3	2	*	*	7	*	*	*	*	0	*	3
*	5	*	*	2	*	*	6	3	1	8	*	*	*	*	*
*	5	*	*	2	*	*	6	3	1	8	*	*	*	*	*
*	5	*	*	5	*	*	2	5	*	*	*	3	9	*	*
*	5	*	*	5	*	*	2	5	*	*	*	3	9	*	*
*	5	*	*	7	*	*	*	8	*	*	7	3	*	6	8
*	5	*	*	7	*	*	*	8	*	*	7	3	*	6	8
*	5	*	*	*	1	*	8	*	1	*	*	*	4	9	*
*	5	*	*	*	1	*	8	*	1	*	*	*	4	9	*
*	5	*	*	*	9	*	1	*	8	*	*	8	*	1	*
*	5	*	*	*	9	*	1	*	8	*	*	8	*	1	*
*	5	*	*	*	*	2	*	1	*	1	*	5	*	*	4
*	5	*	*	*	*	2	*	1	*	1	*	5	*	*	4
*	5	*	*	*	*	9	*	8	9	8	4	*	*	*	*
*	5	*	*	*	*	9	*	8	9	8	4	*	*	*	*
*	5	*	*	*	*	*	*	*	*	0	*	*	0	4	*
*	5	*	*	*	*	*	*	*	*	0	*	*	0	4	*
*	6	0	9	*	*	1	*	*	9	*	*	*	*	3	*
*	6	0	9	*	*	1	*	*	9	*	*	*	*	3	*
*	6	4	*	*	*	0	4	*	*	*	*	0	*	*	8
*	6	4	*	*	*	0	4	*	*	*	*	0	*	*	8
*	6	9	0	*	*	0	2	*	*	*	*	*	8	8	*
*	6	9	0	*	*	0	2	*	*	*	*	*	8	8	*
*	6	9	*	5	7	*	*	*	2	1	2	*	*	*	7
*	6	9	*	5	7	*	*	*	2	1	2	*	*	*	7
*	6	*	0	2	*	*	*	*	5	3	*	7	*	*	*
*	6	*	0	2	*	*	*	*	5	3	*	7	*	*	*

*	6	*	*	8	*	*	5	*	*	*	*	3	*	6	*
*	6	*	*	8	*	*	5	*	*	*	*	3	*	6	*
*	6	*	*	9	*	*	6	4	*	0	*	*	*	*	*
*	6	*	*	9	*	*	6	4	*	0	*	*	*	*	*
*	6	*	*	*	*	1	0	*	6	*	*	2	*	*	*
*	6	*	*	*	*	1	0	*	6	*	*	2	*	*	*
*	7	0	*	7	9	8	*	*	*	*	*	2	*	*	9
*	7	0	*	7	9	8	*	*	*	*	*	2	*	*	9
*	7	1	*	8	*	*	*	*	*	*	2	3	3	*	*
*	7	1	*	8	*	*	*	*	*	*	2	3	3	*	*
*	7	5	*	*	*	3	0	*	*	*	*	*	1	*	5
*	7	5	*	*	*	3	0	*	*	*	*	*	1	*	5
*	7	*	*	1	*	2	0	7	*	*	*	*	*	5	*
*	7	*	*	1	*	2	0	7	*	*	*	*	*	5	*
*	7	*	*	8	*	*	*	*	7	7	*	*	8	*	*
*	7	*	*	8	*	*	*	*	7	7	*	*	8	*	*
*	7	*	*	*	8	5	7	*	*	*	*	*	*	9	9
*	7	*	*	*	8	5	7	*	*	*	*	*	*	9	9
*	7	*	*	*	*	2	*	6	0	*	*	*	0	*	*
*	7	*	*	*	*	2	*	6	0	*	*	*	0	*	*
*	8	0	4	9	*	*	*	*	2	*	*	*	*	6	*
*	8	0	4	9	*	*	*	*	2	*	*	*	*	6	*
*	8	2	8	4	*	*	*	*	5	*	*	2	4	2	*
*	8	2	8	4	*	*	*	*	5	*	*	2	4	2	*
*	8	4	3	*	8	6	*	*	*	*	*	9	*	*	*
*	8	4	3	*	8	6	*	*	*	*	*	9	*	*	*
*	8	4	7	4	*	*	*	*	*	7	*	*	*	*	2
*	8	4	7	4	*	*	*	*	*	7	*	*	*	*	2
*	8	6	*	9	1	*	*	6	*	9	*	*	*	*	*
*	8	6	*	9	1	*	*	6	*	9	*	*	*	*	*

*	8	*	0	1	6	*	*	*	9	*	7	*	*	*	*
*	8	*	0	1	6	*	*	*	9	*	7	*	*	*	*
*	8	*	3	5	3	*	*	*	7	4	*	*	*	*	*
*	8	*	3	5	3	*	*	*	7	4	*	*	*	*	*
*	8	*	6	*	*	*	*	*	2	0	*	2	*	3	*
*	8	*	6	*	*	*	*	*	2	0	*	2	*	3	*
*	8	*	8	*	*	*	*	*	2	*	*	*	*	0	4
*	8	*	8	*	*	*	*	*	2	*	*	*	*	0	4
*	8	*	*	*	*	*	3	9	4	*	6	*	0	*	*
*	8	*	*	*	*	*	3	9	4	*	6	*	0	*	*
*	8	*	*	*	*	*	8	*	6	7	7	*	7	*	*
*	8	*	*	*	*	*	8	*	6	7	7	*	7	*	*
*	9	0	5	*	*	8	*	*	3	1	*	*	*	3	2
*	9	0	5	*	*	8	*	*	3	1	*	*	*	3	2
*	9	4	*	*	*	0	2	*	1	*	*	*	3	*	*
*	9	4	*	*	*	0	2	*	1	*	*	*	3	*	*
*	9	7	*	*	0	*	*	*	*	7	*	5	*	6	*
*	9	7	*	*	0	*	*	*	*	7	*	5	*	6	*
*	9	9	*	*	*	9	4	*	*	*	*	8	0	*	5
*	9	9	*	*	*	9	4	*	*	*	*	8	0	*	5
*	9	*	0	*	*	6	*	4	*	*	*	*	*	8	4
*	9	*	0	*	*	6	*	4	*	*	*	*	*	8	4
*	9	*	3	*	2	*	*	*	*	8	*	*	*	3	*
*	9	*	3	*	2	*	*	*	*	8	*	*	*	3	*
*	9	*	*	*	*	*	*	2	0	*	*	*	*	4	8
*	9	*	*	*	*	*	*	2	0	*	*	*	*	4	8
*	*	0	4	*	*	*	*	*	3	*	6	*	5	*	8
*	*	0	4	*	*	*	*	*	3	*	6	*	5	*	8
*	*	0	8	*	2	8	*	2	7	*	*	*	*	*	*
*	*	0	8	*	2	8	*	2	7	*	*	*	*	*	*

*	*	0	*	2	0	*	*	1	5	*	1	*	*	*	*
*	*	0	*	2	0	*	*	1	5	*	1	*	*	*	*
*	*	0	*	*	0	*	*	9	*	*	8	*	9	*	*
*	*	0	*	*	0	*	*	9	*	*	8	*	9	*	*
*	*	0	*	*	7	*	*	*	*	*	*	6	*	2	9
*	*	0	*	*	7	*	*	*	*	*	*	6	*	2	9
*	*	0	*	*	*	8	4	*	3	*	*	1	*	7	*
*	*	0	*	*	*	8	4	*	3	*	*	1	*	7	*
*	*	1	2	*	*	*	*	*	*	*	4	*	*	7	0
*	*	1	2	*	*	*	*	*	*	*	4	*	*	7	0
*	*	1	5	9	*	9	*	5	*	*	*	3	*	*	*
*	*	1	5	9	*	9	*	5	*	*	*	3	*	*	*
*	*	1	*	3	*	*	*	8	0	7	*	8	*	9	*
*	*	1	*	3	*	*	*	8	0	7	*	8	*	9	*
*	*	1	*	6	4	*	*	*	*	*	*	6	4	*	9
*	*	1	*	6	4	*	*	*	*	*	*	6	4	*	9
*	*	1	*	*	0	*	2	*	6	2	*	*	7	*	*
*	*	1	*	*	0	*	2	*	6	2	*	*	7	*	*
*	*	1	*	*	0	*	7	4	*	*	*	9	*	*	*
*	*	1	*	*	0	*	7	4	*	*	*	9	*	*	*
*	*	2	0	8	*	*	2	*	*	*	7	*	*	3	7
*	*	2	0	8	*	*	2	*	*	*	7	*	*	3	7
*	*	2	2	*	1	1	*	4	0	9	*	2	*	*	*
*	*	2	2	*	1	1	*	4	0	9	*	2	*	*	*
*	*	2	8	*	*	*	*	6	3	*	0	*	*	6	*
*	*	2	8	*	*	*	*	6	3	*	0	*	*	6	*
*	*	2	*	0	*	*	*	*	*	0	9	*	*	*	1
*	*	2	*	0	*	*	*	*	*	0	9	*	*	*	1
*	*	2	*	4	*	*	*	7	*	*	*	6	*	1	*
*	*	2	*	4	*	*	*	7	*	*	*	6	*	1	*

*	*	3	3	*	*	*	*	*	8	3	*	*	*	0	1
*	*	3	3	*	*	*	*	*	8	3	*	*	*	0	1
*	*	3	6	8	9	7	*	*	*	*	1	*	*	7	*
*	*	3	6	8	9	7	*	*	*	*	1	*	*	7	*
*	*	3	*	0	6	*	1	*	2	*	*	*	*	*	*
*	*	3	*	0	6	*	1	*	2	*	*	*	*	*	*
*	*	3	*	2	*	1	*	3	5	*	*	9	*	*	*
*	*	3	*	2	*	1	*	3	5	*	*	9	*	*	*
*	*	3	*	3	8	*	*	*	*	*	*	*	0	2	8
*	*	3	*	3	8	*	*	*	*	*	*	*	0	2	8
*	*	3	*	6	3	*	7	*	*	*	*	*	*	*	*
*	*	3	*	6	3	*	7	*	*	*	*	*	*	*	*
*	*	3	*	*	*	*	1	3	2	9	9	*	*	4	1
*	*	3	*	*	*	*	1	3	2	9	9	*	*	4	1
*	*	4	0	*	9	*	*	3	*	6	*	*	6	*	*
*	*	4	0	*	9	*	*	3	*	6	*	*	6	*	*
*	*	4	1	*	*	4	*	*	7	*	7	*	*	9	*
*	*	4	1	*	*	4	*	*	7	*	7	*	*	9	*
*	*	4	7	*	2	6	*	*	3	*	*	*	*	3	6
*	*	4	7	*	2	6	*	*	3	*	*	*	*	3	6
*	*	4	8	8	*	*	*	*	9	9	*	*	*	*	*
*	*	4	8	8	*	*	*	*	9	9	*	*	*	*	*
*	*	4	*	0	6	*	*	0	5	*	*	*	*	*	*
*	*	4	*	0	6	*	*	0	5	*	*	*	*	*	*
*	*	4	*	3	*	6	*	9	*	*	*	9	8	*	*
*	*	4	*	3	*	6	*	9	*	*	*	9	8	*	*
*	*	4	*	*	*	1	*	8	*	7	*	3	*	5	*
*	*	4	*	*	*	1	*	8	*	7	*	3	*	5	*
*	*	4	*	*	*	*	*	1	3	*	6	7	*	1	*
*	*	4	*	*	*	*	*	1	3	*	6	7	*	1	*

*	*	4	*	*	*	*	*	3	*	*	*	5	*	*	*
*	*	4	*	*	*	*	*	3	*	*	*	5	*	*	*
*	*	4	*	*	*	*	*	*	*	9	*	*	9	7	9
*	*	4	*	*	*	*	*	*	*	9	*	*	9	7	9
*	*	5	0	*	*	*	0	*	*	4	1	*	9	*	*
*	*	5	0	*	*	*	0	*	*	4	1	*	9	*	*
*	*	5	*	1	*	*	*	5	*	*	*	*	*	9	9
*	*	5	*	1	*	*	*	5	*	*	*	*	*	9	9
*	*	5	*	*	*	0	5	*	*	*	4	0	*	6	*
*	*	5	*	*	*	0	5	*	*	*	4	0	*	6	*
*	*	5	*	*	*	*	8	*	7	*	4	*	7	4	*
*	*	5	*	*	*	*	8	*	7	*	4	*	7	4	*
*	*	5	*	*	*	*	*	9	*	2	9	7	5	*	*
*	*	5	*	*	*	*	*	9	*	2	9	7	5	*	*
*	*	5	*	*	*	*	*	*	6	*	5	*	0	3	0
*	*	5	*	*	*	*	*	*	6	*	5	*	0	3	0
*	*	5	*	*	*	*	*	*	*	8	*	0	4	*	8
*	*	5	*	*	*	*	*	*	*	8	*	0	4	*	8
*	*	6	*	*	3	*	*	*	3	3	9	7	*	*	*
*	*	6	*	*	3	*	*	*	3	3	9	7	*	*	*
*	*	6	*	*	*	*	0	5	*	*	*	5	*	9	*
*	*	6	*	*	*	*	0	5	*	*	*	5	*	9	*
*	*	7	2	4	*	*	*	*	*	8	*	*	6	*	5
*	*	7	2	4	*	*	*	*	*	8	*	*	6	*	5
*	*	8	3	*	2	5	*	*	*	*	*	*	*	*	7
*	*	8	3	*	2	5	*	*	*	*	*	*	*	*	7
*	*	8	5	5	*	*	*	*	*	*	*	3	*	1	*
*	*	8	5	5	*	*	*	*	*	*	*	3	*	1	*
*	*	8	*	1	8	*	*	*	*	5	*	*	5	*	4
*	*	8	*	1	8	*	*	*	*	5	*	*	5	*	4

*	*	8	*	5	*	9	*	*	*	*	*	6	7	*	*
*	*	8	*	5	*	9	*	*	*	*	*	6	7	*	*
*	*	8	*	*	5	0	*	4	*	*	0	*	4	*	6
*	*	8	*	*	5	0	*	4	*	*	0	*	4	*	6
*	*	8	*	*	*	*	1	*	*	*	2	1	*	*	*
*	*	8	*	*	*	*	1	*	*	*	2	1	*	*	*
*	*	9	4	*	6	*	*	2	7	*	*	*	*	*	*
*	*	9	4	*	6	*	*	2	7	*	*	*	*	*	*
*	*	9	7	*	7	*	*	*	*	*	6	9	*	*	*
*	*	9	7	*	7	*	*	*	*	*	6	9	*	*	*
*	*	9	8	*	7	*	6	*	*	*	*	*	*	*	*
*	*	9	8	*	7	*	6	*	*	*	*	*	*	*	*
*	*	9	*	8	*	9	*	*	*	*	*	*	1	*	2
*	*	9	*	8	*	9	*	*	*	*	*	*	1	*	2
*	*	9	*	*	1	*	*	8	9	*	*	0	*	4	*
*	*	9	*	*	1	*	*	8	9	*	*	0	*	4	*
*	*	9	*	*	*	0	1	*	*	6	*	*	*	*	*
*	*	9	*	*	*	0	1	*	*	6	*	*	*	*	*
*	*	9	*	*	*	8	*	*	*	*	*	6	*	2	4
*	*	9	*	*	*	8	*	*	*	*	*	6	*	2	4
*	*	*	0	4	*	2	*	8	3	*	0	*	*	*	*
*	*	*	0	4	*	2	*	8	3	*	0	*	*	*	*
*	*	*	0	*	7	7	*	*	*	*	*	*	*	8	*
*	*	*	0	*	7	7	*	*	*	*	*	*	*	8	*
*	*	*	0	*	8	3	*	*	*	*	*	2	*	*	2
*	*	*	0	*	8	3	*	*	*	*	*	2	*	*	2
*	*	*	0	*	*	*	1	5	*	*	8	6	*	*	*
*	*	*	0	*	*	*	1	5	*	*	8	6	*	*	*
*	*	*	0	*	*	*	*	9	*	*	0	3	9	*	0
*	*	*	0	*	*	*	*	9	*	*	0	3	9	*	0

*	*	*	1	9	3	*	*	*	8	*	*	5	*	6	*
*	*	*	1	9	3	*	*	*	8	*	*	5	*	6	*
*	*	*	1	9	*	*	*	*	*	*	2	*	7	*	3
*	*	*	1	9	*	*	*	*	*	*	2	*	7	*	3
*	*	*	1	*	8	*	*	*	*	*	*	5	*	1	0
*	*	*	1	*	8	*	*	*	*	*	*	5	*	1	0
*	*	*	1	*	*	3	*	*	*	3	*	8	*	3	8
*	*	*	1	*	*	3	*	*	*	3	*	8	*	3	8
*	*	*	2	9	3	3	1	4	*	*	1	*	*	0	*
*	*	*	2	9	3	3	1	4	*	*	1	*	*	0	*
*	*	*	2	*	3	*	*	4	*	3	*	*	*	0	4
*	*	*	2	*	3	*	*	4	*	3	*	*	*	0	4
*	*	*	3	1	*	5	7	*	1	*	*	*	*	1	1
*	*	*	3	1	*	5	7	*	1	*	*	*	*	1	1
*	*	*	3	3	*	*	6	4	1	*	6	3	*	*	*
*	*	*	3	3	*	*	6	4	1	*	6	3	*	*	*
*	*	*	3	*	9	7	*	4	*	*	7	*	*	*	*
*	*	*	3	*	9	7	*	4	*	*	7	*	*	*	*
*	*	*	3	*	9	*	5	7	3	*	*	4	*	*	*
*	*	*	3	*	9	*	5	7	3	*	*	4	*	*	*
*	*	*	3	*	*	*	*	*	8	0	*	*	7	7	6
*	*	*	3	*	*	*	*	*	8	0	*	*	7	7	6
*	*	*	4	3	*	*	*	*	*	3	3	*	*	9	1
*	*	*	4	3	*	*	*	*	*	3	3	*	*	9	1
*	*	*	4	6	5	4	6	*	*	*	6	*	*	3	*
*	*	*	4	6	5	4	6	*	*	*	6	*	*	3	*
*	*	*	4	*	0	7	*	*	*	9	*	*	2	*	3
*	*	*	4	*	0	7	*	*	*	9	*	*	2	*	3
*	*	*	4	*	7	*	6	5	0	*	*	0	*	*	*
*	*	*	4	*	7	*	6	5	0	*	*	0	*	*	*

*	*	*	4	*	8	*	6	*	*	*	9	*	8	*	*
*	*	*	4	*	8	*	6	*	*	*	9	*	8	*	*
*	*	*	4	*	*	6	0	*	*	*	*	*	*	7	7
*	*	*	4	*	*	6	0	*	*	*	*	*	*	7	7
*	*	*	4	*	*	*	6	4	7	*	*	9	*	*	6
*	*	*	4	*	*	*	6	4	7	*	*	9	*	*	6
*	*	*	5	3	6	7	*	2	*	*	1	*	*	*	*
*	*	*	5	3	6	7	*	2	*	*	1	*	*	*	*
*	*	*	5	8	6	*	*	4	7	6	*	*	9	*	*
*	*	*	5	8	6	*	*	4	7	6	*	*	9	*	*
*	*	*	5	*	*	2	8	*	*	1	*	*	9	1	*
*	*	*	5	*	*	2	8	*	*	1	*	*	9	1	*
*	*	*	6	3	1	*	*	*	*	*	0	7	3	4	*
*	*	*	6	3	1	*	*	*	*	*	0	7	3	4	*
*	*	*	6	*	1	*	*	*	*	8	9	*	0	2	*
*	*	*	6	*	1	*	*	*	*	8	9	*	0	2	*
*	*	*	6	*	*	5	0	0	*	*	*	*	8	9	*
*	*	*	6	*	*	5	0	0	*	*	*	*	8	9	*
*	*	*	7	4	*	*	*	*	0	0	0	*	4	*	*
*	*	*	7	4	*	*	*	*	0	0	0	*	4	*	*
*	*	*	7	*	1	*	*	*	4	*	0	*	*	*	1
*	*	*	7	*	1	*	*	*	4	*	0	*	*	*	1
*	*	*	8	0	5	*	*	*	*	3	*	*	8	4	*
*	*	*	8	0	5	*	*	*	*	3	*	*	8	4	*
*	*	*	8	3	*	7	*	*	3	*	8	*	*	*	3
*	*	*	8	3	*	7	*	*	3	*	8	*	*	*	3
*	*	*	8	*	4	*	*	*	9	*	2	5	*	3	*
*	*	*	8	*	4	*	*	*	9	*	2	5	*	3	*
*	*	*	8	*	*	*	6	*	0	*	3	*	*	3	*
*	*	*	8	*	*	*	6	*	0	*	3	*	*	3	*

*	*	*	8	*	*	*	*	*	3	4	*	*	4	9	5
*	*	*	8	*	*	*	*	*	3	4	*	*	4	9	5
*	*	*	9	8	6	7	1	*	*	*	4	*	7	0	*
*	*	*	9	8	6	7	1	*	*	*	4	*	7	0	*
*	*	*	9	*	6	*	*	5	6	*	*	8	*	*	*
*	*	*	9	*	6	*	*	5	6	*	*	8	*	*	*
*	*	*	9	*	*	*	4	7	*	*	2	9	8	1	*
*	*	*	9	*	*	*	4	7	*	*	2	9	8	1	*
*	*	*	*	0	4	3	*	5	*	*	0	6	7	*	7
*	*	*	*	0	4	3	*	5	*	*	0	6	7	*	7
*	*	*	*	1	8	*	*	8	8	4	0	*	*	*	*
*	*	*	*	1	8	*	*	8	8	4	0	*	*	*	*
*	*	*	*	1	*	*	4	3	7	*	7	8	0	*	*
*	*	*	*	1	*	*	4	3	7	*	7	8	0	*	*
*	*	*	*	2	1	*	2	5	*	3	*	*	*	*	9
*	*	*	*	2	1	*	2	5	*	3	*	*	*	*	9
*	*	*	*	2	*	*	*	7	7	*	3	*	*	*	2
*	*	*	*	2	*	*	*	7	7	*	3	*	*	*	2
*	*	*	*	2	*	*	*	*	2	*	1	8	7	*	*
*	*	*	*	2	*	*	*	*	2	*	1	8	7	*	*
*	*	*	*	3	6	2	0	*	5	*	*	*	*	*	7
*	*	*	*	3	6	2	0	*	5	*	*	*	*	*	7
*	*	*	*	3	*	*	*	*	6	8	*	4	*	4	5
*	*	*	*	3	*	*	*	*	6	8	*	4	*	4	5
*	*	*	*	4	0	*	1	7	*	*	1	*	7	*	*
*	*	*	*	4	0	*	1	7	*	*	1	*	7	*	*
*	*	*	*	4	*	8	*	0	*	3	*	7	*	0	*
*	*	*	*	4	*	8	*	0	*	3	*	7	*	0	*
*	*	*	*	4	*	*	*	*	3	*	*	*	*	*	*
*	*	*	*	4	*	*	*	*	3	*	*	*	*	*	*

*	*	*	*	4	*	*	*	*	*	*	*	*	9	6	4
*	*	*	*	4	*	*	*	*	*	*	*	*	9	6	4
*	*	*	*	5	*	*	3	*	*	*	*	*	*	0	7
*	*	*	*	5	*	*	3	*	*	*	*	*	*	0	7
*	*	*	*	6	4	*	*	7	*	4	*	*	1	*	*
*	*	*	*	6	4	*	*	7	*	4	*	*	1	*	*
*	*	*	*	6	*	*	9	*	*	*	4	*	2	2	*
*	*	*	*	6	*	*	9	*	*	*	4	*	2	2	*
*	*	*	*	7	2	0	1	0	*	*	*	*	*	*	*
*	*	*	*	7	2	0	1	0	*	*	*	*	*	*	*
*	*	*	*	7	4	*	*	5	*	*	2	*	*	*	*
*	*	*	*	7	4	*	*	5	*	*	2	*	*	*	*
*	*	*	*	7	4	*	*	5	*	*	2	*	*	*	*
*	*	*	*	7	4	*	*	*	6	*	2	*	*	1	5
*	*	*	*	7	4	*	*	*	6	*	2	*	*	1	5
*	*	*	*	7	*	3	*	9	*	*	0	*	*	1	6
*	*	*	*	7	*	3	*	9	*	*	0	*	*	1	6
*	*	*	*	7	*	*	3	3	*	*	7	*	*	6	3
*	*	*	*	7	*	*	3	3	*	*	7	*	*	6	3
*	*	*	*	7	*	*	7	*	*	*	*	1	4	4	4
*	*	*	*	7	*	*	7	*	*	*	*	1	4	4	4
*	*	*	*	8	3	*	*	*	*	*	0	2	*	*	*
*	*	*	*	8	3	*	*	*	*	*	0	2	*	*	*
*	*	*	*	8	4	*	*	*	*	*	7	9	0	*	*
*	*	*	*	8	4	*	*	*	*	*	7	9	0	*	*
*	*	*	*	8	*	*	*	*	8	2	*	3	9	*	6
*	*	*	*	8	*	*	*	*	8	2	*	3	9	*	6
*	*	*	*	9	2	*	*	*	8	0	9	*	*	*	2
*	*	*	*	9	2	*	*	*	8	0	9	*	*	*	2
*	*	*	*	*	0	4	*	1	*	5	6	*	2	*	*

*	*	*	*	*	0	4	*	1	*	5	6	*	2	*	*
*	*	*	*	*	1	*	*	8	*	*	3	*	*	0	*
*	*	*	*	*	1	*	*	8	*	*	3	*	*	0	*
*	*	*	*	*	2	*	1	5	4	0	*	*	*	*	*
*	*	*	*	*	2	*	1	5	4	0	*	*	*	*	*
*	*	*	*	*	2	*	4	9	*	0	*	7	*	0	2
*	*	*	*	*	2	*	4	9	*	0	*	7	*	0	2
*	*	*	*	*	2	*	*	4	*	*	9	*	*	*	*
*	*	*	*	*	2	*	*	4	*	*	9	*	*	*	*
*	*	*	*	*	6	*	2	0	*	*	*	*	6	9	*
*	*	*	*	*	6	*	2	0	*	*	*	*	6	9	*
*	*	*	*	*	8	2	5	3	*	6	*	*	*	*	0
*	*	*	*	*	8	2	5	3	*	6	*	*	*	*	0
*	*	*	*	*	8	9	4	*	*	*	*	*	*	1	2
*	*	*	*	*	8	9	4	*	*	*	*	*	*	1	2
*	*	*	*	*	9	1	*	*	7	9	*	*	*	1	9
*	*	*	*	*	9	1	*	*	7	9	*	*	*	1	9
*	*	*	*	*	*	1	0	*	*	*	*	7	*	3	2
*	*	*	*	*	*	1	0	*	*	*	*	7	*	3	2
*	*	*	*	*	*	3	3	*	*	*	*	5	7	*	3
*	*	*	*	*	*	3	3	*	*	*	*	5	7	*	3
*	*	*	*	*	*	5	8	*	*	*	8	4	*	5	3
*	*	*	*	*	*	5	8	*	*	*	8	4	*	5	3
*	*	*	*	*	*	6	9	*	7	*	*	5	*	*	*
*	*	*	*	*	*	6	9	*	7	*	*	5	*	*	*
*	*	*	*	*	*	9	*	1	2	6	*	7	7	*	*
*	*	*	*	*	*	9	*	1	2	6	*	7	7	*	*
*	*	*	*	*	*	*	5	*	4	*	4	9	2	*	*
*	*	*	*	*	*	*	5	*	4	*	4	9	2	*	*
*	*	*	*	*	*	*	6	*	6	2	*	*	0	*	*

*	*	*	*	*	*	*	6	*	6	2	*	*	0	*	*
*	*	*	*	*	*	*	7	9	0	*	7	3	7	*	*
*	*	*	*	*	*	*	7	9	0	*	7	3	7	*	*
*	*	*	*	*	*	*	9	9	*	8	*	*	*	2	1
*	*	*	*	*	*	*	9	9	*	8	*	*	*	2	1
*	*	*	*	*	*	*	*	1	0	5	4	0	*	*	*
*	*	*	*	*	*	*	*	1	0	5	4	0	*	*	*

10.7 问题三附件数据在 p=3，5，8 情况下的隐藏方案

由于数据过于庞大，只展示二元 1 的隐藏方案，其他详细方案见代码运行后生产的数据表。

表 23　二元 1 在 p=3 情况下的隐藏方案

0	0	0	1	0	1
0	0	0	1	0	1
0	0	0	1	0	1
0	0	1	1	1	0
0	0	1	1	1	0
0	0	1	1	1	0
0	0	1	1	1	0
0	0	1	*	1	1
0	0	1	*	1	1
0	0	1	*	1	1
0	0	1	*	1	1
0	1	1	0	1	*
0	1	1	0	1	*
0	1	1	0	1	*
0	1	1	0	1	*
0	1	*	0	0	1

0	1	*	0	0	1
0	1	*	0	0	1
0	1	*	1	1	1
0	1	*	1	1	1
0	1	*	1	1	1
0	1	*	1	1	1
0	*	*	1	*	0
0	*	*	1	*	0
0	*	*	1	*	0
0	*	*	1	*	0
0	*	*	1	*	0
0	*	*	1	*	0
1	0	0	*	0	*
1	0	0	*	0	*
1	0	0	*	0	*
1	0	0	*	0	*
1	0	0	*	0	*
1	0	0	*	1	*
1	0	0	*	1	*
1	0	0	*	1	*
1	0	1	1	0	1
1	0	1	1	0	1
1	0	1	1	0	1
1	0	1	1	1	0
1	0	1	1	1	0
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	0	1	1
1	1	0	0	1	1
1	1	0	*	0	1

1	1	0	*	0	1
1	1	0	*	0	1
1	1	0	*	0	1
1	1	0	*	*	0
1	1	0	*	*	0
1	1	0	*	*	0
1	1	0	*	*	0
1	1	1	0	1	1
1	1	1	0	1	1
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	0	0
1	*	1	0	1	0
1	*	1	0	1	0
1	*	1	0	1	0
1	*	1	0	1	0
*	0	0	0	1	1
*	0	0	0	1	1
*	0	0	0	1	1
*	0	1	0	0	1
*	0	1	0	0	1
*	0	1	0	0	1
*	1	1	1	0	1
*	1	1	1	0	1
*	1	1	1	0	1
*	1	1	1	1	0
*	1	1	1	1	0
*	1	1	1	1	0
*	*	1	0	0	0

*	*	1	0	0	0
*	*	1	0	0	0
*	*	1	0	0	0
*	*	1	0	0	0

表 24　二元 1 在 p=5 情况下的隐藏方案

0	1	*	*	*	1
0	1	*	*	*	1
0	1	*	*	*	1
0	1	*	*	*	1
0	1	*	*	*	1
0	1	*	*	*	1
0	1	*	*	*	1
0	1	*	*	*	1
0	1	*	*	*	1
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
1	1	0	0	*	*
1	1	0	0	*	*
1	1	0	0	*	*
1	1	0	0	*	*
1	1	0	0	*	*
1	1	0	0	*	*

1	1	0	0	*	*
1	1	1	*	0	0
1	1	1	*	0	0
1	1	1	*	0	0
1	1	1	*	0	0
1	1	1	*	0	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	1	*	1	0
1	*	*	1	0	1
1	*	*	1	0	1
1	*	*	1	0	1
1	*	*	1	0	1
1	*	*	1	0	1
1	*	*	1	0	1
1	*	*	1	0	1
1	*	*	1	0	1
*	0	*	0	0	*
*	0	*	0	0	*
*	0	*	0	0	*
*	0	*	0	0	*
*	0	*	0	*	1
*	0	*	0	*	1
*	0	*	0	*	1
*	0	*	0	*	1
*	0	*	0	*	1
*	0	*	0	*	1

*	0	*	0	*	1
*	0	*	0	*	1
*	0	*	0	*	1
*	0	*	*	1	*
*	0	*	*	1	*
*	0	*	*	1	*
*	0	*	*	1	*
*	0	*	*	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	1	1	0	1	*
*	*	*	1	0	0
*	*	*	1	0	0
*	*	*	1	0	0
*	*	*	1	0	0
*	*	*	1	0	0
*	*	*	1	0	0
*	*	*	1	0	0
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1

表 25　二元 1 在 p=8 情况下的隐藏方案

0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
0	*	*	1	1	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
1	*	1	*	*	0
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*
*	1	*	*	*	*

*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*
*	*	*	0	*	*

*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	1	0	1
*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*