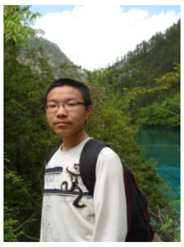
Generalizations on the Classical Problem ‘Mice and the Poison’

Writers: Tianyi Bai, Sitao Xiang
Advisor: Mr. Lei Zhang
Email: 979593763@qq.com
School: Northeast Yucai School
Shenyang Liaoning, China
December 1, 2011



Tianyi Bai, born on 20th July, 1994

Main Awards: gold medal in 10th (2009, South Africa) Invitational World Youth Mathematics Intercity Competition (IWYMIC), silver medal in 2010 China Mathematics Olympics (CMO)
Main Interests in Mathematics: Number Theory, Logic, Group Theory, Set Theory, Differential Equation, Linear Algebra, Mathematical Analysis, Analytic Geometry, Informatics.

Sitao Xiang, born on 18th April, 1994

Main Awards: gold medal in 5th (2011) Asia-Pacific Informatics Olympiad (APIO),
silver medals in 27th, 28th (2010, 2011) National Olympiad in Informatics (NOI)
Main Interests in Mathematics: Combinatorics, Set Theory, Topology, Graph Theory, Theory of Computation, Fractal Geometry, Chaos Theory, Game Theory, Philosophy of Mathematics,

Abstract

The classical problem 'Mice and the Poison' is expressed as follows.

There are 1000 bottles of liquid with one of them poisoned. A mouse will die after one week if it takes the poison. Otherwise it will remain living. Can you use only 10 mice to check out which bottle contains the poison within one week?

This essay generalizes the problem, and obtains the following topics and conclusions.

(1) Time constraints are generalized and Radix Scheme is developed and studied.

(2) A new restriction is added to the 'mice', which also makes the problem more difficult. A new method called 'Chart-Filling' Method is contrived, and its feasibility demonstration is adopted to solve the new problem, which also raises new themes.

(3) Replacement for 'the poison' to 'two poisons' leads to a big challenge, which 'Coordinates-Grouping' Scheme is created to tackle. Upper and Lower bounds are obtained, and computer programs are employed to work out both exact solutions and appraisal solutions. New conjectures are also made due to these efforts.

Key words mice and the poison; radix; binary; status; coordinate; grouping; combination; chart-filling.

Content

1. Introduction 4
1.1 Original Problem and Background 4
1.2 Motivation 4
1.3 Main Results 5
2 Variations of the Original Problem and Corresponding Researches 6
2.1 Time Generalization and Radix Scheme 6
2.1.1 Radix Scheme and Discrete Hypothesis 6
2.1.2 Proof of Optimality 7
2.1.3 Conclusions and Other Variations 7
2.2 Realistic Restriction and Chart-Filling Method 8
2.2.1 Chart-Filling Method 8
2.2.2 Proof of Feasibility 10
2.2.3 Conclusions and Other Variations 14
2.3 Two-Poison Replacement and Grouping-Coordinate Scheme 15
2.3.1 Coordinate Scheme 15
2.3.2 Grouping Method and Coordinate-Grouping Scheme 16
2.3.3 New Problem and Compatibility 17
2.3.4 Qualitative Analysis of $\boldsymbol{f(n)}$ 18
2.3.5 Further Study of Lower Bounds 20
2.3.6 Quantitative Computation of $\boldsymbol{f(n)}$ 23
2.3.7 Informatics method and upper bound for $\boldsymbol{c(n)}$ 25
2.3.8 Summarization 26
3. Conclusion 27
Acknowledgement 27
Appendix I Source Code of Search Algorithm in C++ 28
Appendix II Source Code of Random Construction in Free Pascal 29
Appendix III Time data for Random Program 31
Appendix IV Corresponding Solutions for Quantitative Analysis of $\boldsymbol{f(n)}$ 33

1. Introduction

1.1 Original Problem and Background

We once read an interesting passage on http://www.matrix67.com/blog/archives/4361, which states a curious problem named 'Mice and the Poison' and several new thoughts in this problem. We were attracted by the amazing solution of the problem, and wanted to do some new research on it.

The original problem 'Mice and the Poison' is expressed as follows. There are 1000 bottles of liquid with one of them poisoned. A mouse will die after one week if it takes the poison, otherwise it will remain alive. Can you use only 10 mice to check out which bottle contains the poison within one week? (It can be easily worked out that we can only do one experiment according to the time constriction. What is more, to guarantee that a solution is possible, we must suppose that several medicines can be taken by one mouse at a time and the poison works just as it is the only one the mouse takes)

The solution given by the website writer uses binary representation. We give a binary number to each bottle, namely from 0000000001 to 111110100. The experiment is carried out so that the ith$(1 \leq i \leq 10)$ mouse takes all medicines whose ith digit is '1'. Finally, if the ith mouse is dead one week later, we can deduce that the ith digit of the poison's number is 1, or else it is 0.

Hence we can easily tell that the maximum amount of bottles we are able to check by the manipulation is the amount of all binary numbers with 10 digits, that is, 1024. Further more, if we have n mice instead of 10, the maximum number turns out to be 2^n.

This passage raised heated discussion online, and lots of other generalizations are advanced. The author himself gave a radix 3 solution to a situation of the time generalization which we will consider in the first part of our essay. 'yh' and lots of others discussed noticed difficulties of not having the discrete hypothesis, which we are to think about as well in the first part. One commentator named 'yac' displayed a concrete version of the two-poison problem which we shall try in the third part of this essay. He claimed it to be an interview question of a company named 'GaoSheng', and gave the Coordinate Scheme (named by us) of $n = 10$ in addition. In fact, this problem also has several different variations, for instance, the problem of 'IBM Ponder This'(May 2011). While our researches rely on concepts such as 'mice' and 'poison', we will only consider the problem itself in the passage.

Since the only basis of our paper is the question posted on the website, we are not to show reference again at the end of the paper.

1.2 Motivation

As we have already seen from the previous section, when we have n mice instead of 10, the maximum number will vary correspondingly. If we change other conditions of it, we probably will get some similar results. Therefore, we try changing every concrete number to variable in the problem as well as adding new restrictions and new conditions, and get three main results which we shall state in the following section.

Firstly, we change those numbers representing time to variables since the writer of the blog led us to do so by changing the ‘one week’ to ‘two weeks’.

Secondly, we notice that in the original problem, each mouse has to be fed with more than 500 different kinds of medicine all at a time, which in some way is too ideal. So we add a new constraint and get an entirely new problem.

Thirdly, we consider the two-poison problem, which was displayed in the website by yac first.

1.3 Main Results

In the *Time Generalization* part of our essay, we consider the following problem.

Problem1 There are some bottles of liquid with one of them poisoned. A mouse will die after t weeks if it takes the poison; otherwise it will remain alive. Now we have n mice to check out which bottle contains the poison in m weeks’ time, what is the maximum number of bottles we are able to check from?

Let $a(n,m,t)$ denotes the optimal answer, we conclude that $a(n,m,t)=(m-t+2)^n$ by generalizing the binary solution to radix scheme. While doing this, we notice the core of the problem and introduce the discrete hypothesis. In addition, we combine the two time variables and get an insight into the radix scheme.

In the *Realistic Restriction* part, we add a further constraint to the problem.

Problem 2 There are some bottles of liquid with one of them poisoned. A mouse will die if it takes the poison, otherwise it will remain alive. Now we have n mice to check out which bottle contains the poison with only one experiment, where each mouse can take at most r different medicines at a time. What is the maximum number of bottles we are able to deal with this time?

We create a new manipulation called Chart-Filling Method to convert the problem so that a new point of view is worked out. Thus we get the exact expression of the answer. In the process of reaching this goal, we get an important concept called ‘status’ of a medicine, which is actually a generalization of the binary representation.

In the *Two-Poison* part, we will try a somewhat different problem as follows.

Problem 3 There are plenty of medicines with two of them poisoned. A mouse will die if it takes the poison, otherwise it will remain alive. With n mice and one experiment, we can check out that some propositions of those medicines are nontoxic. What is the maximum ratio of nontoxic proportion we are able to get?

We consider ‘Coordinates Scheme’ and ‘Grouping Method’, which we shall combine to get our ‘Coordinates-Grouping Scheme’. Since this **Problem 3** is tough to handle, we will consider a restricted problem with respect to the idea of Grouping.

Problem 3* There are some bottles of liquid with two of them poisoned. A mouse will die if it takes the poison, otherwise it will remain alive. Now we have *n* mice to check out which bottle contains the poison within one experiment. What is the maximum number of bottles we are able to deal with?

Main content of this part is about this answer, which is donated by $f(n)$. We get an upper bound and several lower bounds. Moreover, lower bounds are studied carefully and coordinates are encountered again. As the exact number of general cases cannot be worked out, we have two computer programs, one of which is precise but slow, the other being faster but estimative.

2 Variations of the Original Problem and Corresponding Researches

2.1 Time Generalization and Radix Scheme

2.1.1 Radix Scheme and Discrete Hypothesis

In this chapter, we consider **problem 1** stated in section 1.3.

Problem 1 There are some bottles of liquid with one of them poisoned. A mouse will die after t week if it takes the poison, otherwise it will remain alive. Now we have n mice to check out which bottle contains the poison in m weeks' time, what is the maximum number of bottles we are able to check from?(Notation $a(n,m,t)$ is created to represent the answer)

The website where the original problem comes from has already pointed out a radix 3 solution to $a(n,2,1)$: Notice that if we give a radix 3 number to each bottle, from 0 to 3^n, and let ith mouse ($1 \leq i \leq n$) take all medicines with the ith digit '1' in the first week and take all medicines with the ith digit '2' in the second week (if it is still alive). If the mouse dies after the first week, dies after the second week or remains alive after all experiments, we can deduce respectively that the ith digit of the poison is 1, 2, or 0. As a consequence, the number corresponding to the poison can be worked out digit by digit.

Imitating the manipulation used, we can easily get the general solution when $t = 1$.

Give a radix $m+1$ number to each bottle, from 0 to $(m+1)^n - 1$, and let the ith mouse ($1 \leq i \leq n$) take all medicines with the ith digit $j(1 \leq j \leq m)$ in the jth week if it is still alive then. If the mouse dies after jth week, we can deduce that the ith digit of the poison is j. Specially, if it remains alive finally, that digit is 0. As a consequence, the number corresponding to the poison can be worked out digit by digit. Therefore $a(n,m,1) \geq (m+1)^n$.

When $t > 1$, it seems that we can take t weeks as one time unit, thus we have $\frac{m}{t}$ units of experiment time(not necessarily an integer), and we should get $a(n,m,t) = a(n,\frac{m}{t},1)$. However, a closer look gives a better scheme:

Give a radix $m-t+2$ number to each bottle, from 0 to $(m-t+2)^n - 1$, and let ith mouse ($1 \leq i \leq n$) take all medicines with the ith digit $j(1 \leq j \leq m-t+1)$ in the jth week if it is still alive then. If the mouse dies after $(j+t)$th week, we can deduce that the ith digit of the poison is j, or it is 0. Thus we can get the number of the poison as all experiments made by a mouse can only have at most one 'die' answer, so when a mouse dies, the other experiments it takes inevitably give 'alive' answers.

According to the previous paragraph, we get a solution which guarantees $a(n,m,t) \geq (m-t+2)^n$. Comparing the two methods, we can see that the stronger one does not wait until the end of an experiment to carry out another experiment with the same mouse.

It seems perfect that we have already worked out the best solutions. But this astonishing manipulation not only illustrates that instinct does not always give the right answer, but motivates us to examine the intrinsical structure of the problem as well.

Carrying this idea back to the original $t = 1$ situation, we find that if we take the one week of delitescence as seven days and m weeks of experiment time as $7m$ days, we should get an even better solution! As the time interval gets smaller and smaller, our function a approaches infinity easily! This is a terrible disaster, so we must take measures to stop it.

We examine our convention and add a new restriction that 'time' here should be discrete. To be precise, one week should be the least time interval. Things get consistent again after this hypothesis is added.

So now we can satisfactorily announce our radix scheme gives $a(n, m, t) \geq (m - t + 2)^n$.

2.1.2 Proof of Optimality

Notice that our **problem 1** requests the best solution, and we should prove the optimality of the scheme in order to really solve this problem.

Theorem 1 $a(n, m, t) = (m - t + 2)^n$

Proof:

No matter what scheme we adopt, the only data we can get from it is the death time of every mouse (A mouse can also survive all experiments, which is considered a special case 'die after infinite time'). As the discrete hypothesis guarantees that a mouse can only die after t to $m + 1$ weeks ($m + 1$ stands for infinity), there are only $m - t + 2$ possible states for each mouse. Thus there are $(m - t + 2)^n$ possible data for us.

From another point of view, we have $a(n, m, t)$ possible states of which medicine being poisoned. If this number is bigger than $(m - t + 2)^n$, according to the well-known Pigeon-hole Principle, there must be one experiment data corresponding to two possibilities of the poison, in which case we fail to examine the poisoned medicine precisely. So $a(n, m, t) \leq (m - t + 2)^n$.

Combine this inequality with the one we got in the previous section, and we gives proof to **theorem 1**. QED.

2.1.3 Conclusions and Other Variations

According to section 2.1.1 and 2.1.2, **problem 1** is solved completely. Long story short, we generalized the binary solution given by the author of the website, added a new discrete hypothesis to make things consistent, and finally proved the optimality of this solution.

Since m and t appear at the same position in the answer, it is indicated that these two variables are related constitutionally. In fact, m and t do the same job of qualifying how many experiments we can make. As all experiments can be made in an arbitrary order and this does not affect the result, we can replace m and t with one variable, the number of experiments. This replacement also implies the discrete hypothesis, so it gives us a further insight of the problem.

Some similar questions can be easily asked and solved. Adding possibilities such as a mouse has a risk of dying naturally will do no good since the problem relies on certainty. Now we may consider one more similar problem in order to illustrate that mice can be even more powerful.

Let $a'(n, p)$ be the maximum amount of bottles we are able to check from with p experiments (notice that we have already abandon m and t to make it clearer) when all conditions of **problem 1** holds except that a mouse will recover immediately after the poison takes effect.

We can at once convert the optimality proof and announce $a'(n,p) \leq 2^{np}$ since each mouse now has p chances of choosing to 'die' or not and has 2^p possible situations in all.

The corresponding scheme can be made on purpose to satisfy the proof: number each bottle with an $n \times p$ matrix whose components are either 1 or 0, and use $a_{i,jk}$ to note component (j,k) of the ith matrix. Let mouse $j(1 \leq j \leq n)$ take all medicines i with $a_{i,jk} = 1$ at the kth $(1 \leq k \leq p)$ experiment, and the result of experiments gives the poison's matrix.

2.2 Realistic Restriction and Chart-Filling Method

2.2.1 Chart-Filling Method

Notice that in the original problem, each mouse has to be fed with more than 500 different kinds of medicine all at a time according to the radix scheme, which in some way is too ideal. So we add a new restriction that each mouse has a limitation with the amount of medicines it takes at a time (We will consider only one experiment for the time being to make it easier):

Problem 2 There are some bottles of liquid with one of them poisoned. A mouse will die if it takes the poison, otherwise it will remain alive. Now we have n mice to check out which bottle contains the poison within only one experiment, where each mouse can take at most r different kinds of medicine at a time.$(1 \leq r \leq 2^{n-1})$. What is the maximum number of bottles we are able to deal with this time? (The answer is noted $b(n,r)$)

Now that radix scheme can not be used to solve this problem, we must create an entirely new method, which is, Chart-Filling Method. To introduce this method, we shall have some new concepts leading to a new point of view first.

Definition 1 n-troop: a group of n mice with each of them able to take $r_i(1 \leq i \leq n)$ more medicines is called an n-troop. Specially, a mouse which is already full (cannot take more medicines) is still considered a legal member of a troop.

In addition, when some more medicines are fed to a troop, r_i will decrease and we consider this add-medicine process same as a troop-decline process. By this means, we will study troop-decline process or add-medicine process instead of the original problem. To get a better view, we will illustrate a troop by plotting a chart with n rows and r_i columns.

Definition 2 Space: If a mouse in a troop can take r more medicines, we say it has r spaces. Spaces of all mice add up to the spaces of the troop. If a medicine is planned to be fed to x mice, we say it takes up x spaces.

According to this definition, a scheme cannot take up more spaces than the troop really has, while we are not sure whether or not a scheme is feasible if it takes up less spaces. In addition, a medicine cannot take up more than one space of one mouse, because using the same medicine to feed the same mouse for more than one time is of no use.

Definition 3 m-status: When we choose a scheme for one medicine, we can deduce the result of what will happen if it is poisoned. This 'result' can be noted as a binary number called the status of the medicine. The method used here is similar to the one used in the first part, that is, use n digits to represent the n mice, and a digit is '1' if the medicine is fed to this mouse, otherwise it is '0'. If a status contains m digits '1' (so it takes up m spaces), it is called an

m-status.

We notice that if a medicine is poisoned, status of it stands for the result we will get from the experiment, thus status signifies all the information about the corresponding medicine in the scheme. To make it clearer, two kinds of medicine can be told apart if and only if their statuses are different.

Now our problem is converted to how to choose the biggest amount of statuses so that each one is distinct and they do not take up more spaces than the troop provides.

We will show manipulation we used in this Chart-Filling Method by considering the following example of $b(5,8)$.

Since the number of spaces is limited, we tend to choose statuses that take up few spaces first. Thus the 0-status(00000) is considered at the beginning. However, there is only one 0-status, and we then turn to 1-statuses, 10000, 01000, 00100, 00010, 00001. Filling these statuses into the troop, we can get a chart like this:

Mouse 1	2							
Mouse 2	3							
Mouse 3	4							
Mouse 4	5							
Mouse 5	6							

This 5×8 form illustrates the original condition of $b(5,8)$, and the numbers inside stand for the second to the sixth statuses are 10000 to 00001, respectively. (The first is 00000, so it does not appear in the form)

Now think about 2-statuses, it is clear that there are $\binom{5}{2}$ of them (and $\binom{n}{m}$ m-statuses for n-troop in general) and filling in all of them costs us 20 spaces. We still have 45 spaces now, so probably we can do it. The form looks like this after the previous step:

Mouse 1	2	7	8	9	10			
Mouse 2	3	7	11	12	13			
Mouse 3	4	14	8	11	15			
Mouse 4	5	14	16	9	12			
Mouse 5	6	15	16	13	10			

3-statuses seem the same, but we encounter a problem now. All 3-statuses will take up $3\binom{5}{3} = 30$ spaces while we have only 15 left. Thus at most $\frac{15}{3} = 5$ 3-statuses can be placed. If we do this carefully, we may have a form as follows.

Mouse 1	2	7	8	9	10	17	18	19
Mouse 2	3	7	11	12	13	17	18	20
Mouse 3	4	14	8	11	15	17	19	21
Mouse 4	5	14	16	9	12	18	20	21
Mouse 5	6	15	16	13	10	19	20	21

Before we happily accept the fact that this Char-Filling Method works, let's see what we will get if we are not careful enough and get stuck with less than five 3-statuses:

Mouse 1	2	7	8	9	10	17	18	19
Mouse 2	3	7	11	12	13	17	18	20
Mouse 3	4	14	8	11	15	17	19	20
Mouse 4	5	14	16	9	12	18	19	20
Mouse 5	6	15	16	13	10			

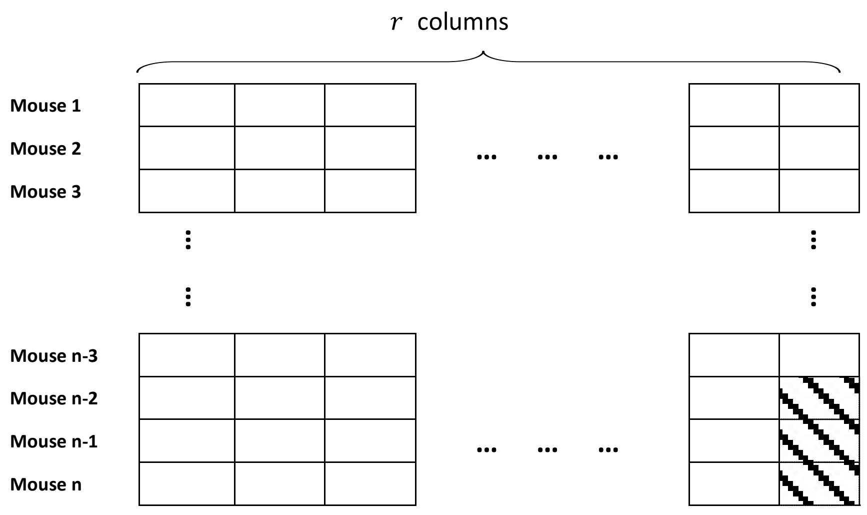
There are still three spaces but we cannot fill them with a 3-status. As a result, this method gives an upper bound but we are not sure whether we have a scheme to reach it.

The upper bound, in general, is $b(n,r) \le [\frac{nr-S_{i_0-1}}{i_0}] + \sum_{j=0}^{i_0-1}\binom{n}{j}$, where $S_i = \sum_{j=0}^{i} j\binom{n}{j}$ and i_0 is an integer satisfying $S_{i_0-1} < nr \le S_{i_0}$. This formula is an instant result of our method since we hope to fill the troop with all 0, 1,…,(i_0-1)-statuses and some i_0-statuses so that there are less than i_0 spaces left(when we just run out of t-status, t spaces left is also considered acceptable, but in this case, we tend to consider that we have used all t-statuses and finish with using zero $(t+1)$-status so that $t+1$ is the actual i_0). Since for every possible management, we can convert a p-status to a q-status if we have not used all q-statuses and $q < p$, and when all those adjustments are made, the final management will not contain more statuses than this method does, thus it gives the best solution if this goal is reachable. As a consequence, all we need is to prove the feasibility of Chart-Filling Method.

2.2.2 Proof of Feasibility

We need more concepts in order to describe our situation before actually trying to solve the problem.

Definition 4 $\underline{(n,r)}$-party: An $\underline{(n,r)}$-party is an n-troop where each mouse has either r spaces or $r-1$ spaces and at least one of them has r spaces. Specially, when r is omitted, we write it as $\underline{n}$-party. Further more, when all mice have r spaces, the troop is called a perfect $\underline{(n,r)}$-party.



We can deduce instantly that an (n,r)-party has more than $n(r-1)$ and not more than nr spaces. When it has exactly nr spaces, it is a perfect (n,r)-party, in other words, a matrix.

Definition 5 p-fill: If we have a scheme to fill an n-troop with p-statuses so that only less than p spaces are left clear, we say this n-troop can be p-filled.

Definition 6 $Z(n,r)$: For each pair of $(n,r)(1 \le r \le 2^{n-1})$, there exists a number i such that $S_{i-1} < nr \le S_i$ $(S_i = \sum_{j=0}^{i} j\binom{n}{j})$. This number is donated by $Z(n,r)$. (That is the i_0 we discussed in the previous section)

Now the problem of feasibility is converted to a problem of whether a perfect (n,r)-party can be filled with all 0, 1,⋯,$(Z(n,r)-1)$-statuses and then be $Z(n,r)$-filled.

We will give a solution by induction of a stronger proposition. To do so, we need a view of what it looks like when a chart is filled with all p-statuses, which will be described in **Lemma 1**. This lemma will give us a well-known equation, **Lemma 2**, containing binominal coefficients which we shall also use. In addition, our proof will rely on a division which will be proved legal in **Lemma 3**. With all three lemma, we shall use induction to prove the reinforced **Theorem 2**, which implies the feasibility of Chart-Filling Method. So let's begin our journey now.

Lemma 1 All i-statuses for a big enough n-troop will take up $\binom{n-1}{i-1}$ spaces of each mouse.

Proof:

It can be examined that there are $\binom{n-1}{i-1}$ i-statuses containing each row, for one component of them must be this row, and the other $i-1$ components can optionally distribute in the other $n-1$ rows. Since each of them takes up one space of the mouse, altogether the number is $\binom{n-1}{i-1}$.

This statement is valid for every mouse. QED.

Lemma 2 $x\binom{n}{x}$= $n\binom{n-1}{x-1}$

Proof: (In fact, proof of this equation need not use the concepts we created)

Consider a big enough n-troop, with all x-statuses filled in. Let's try two ways of calculating the amount of spaces they take up totally.

(1)There are $\binom{n}{x}$ x-statuses, each of which has x components, so these add up to $x\binom{n}{x}$

(2)As it is in **Lemma 1**, those statuses take up $\binom{n-1}{x-1}$ spaces in each row, thus in total they take up $n\binom{n-1}{x-1}$ spaces.

Since the two ways must give the same answer, we have $x\binom{n}{x}$=$n\binom{n-1}{x-1}$. QED.

Lemma 3 For every (n,r)-party and every (n,s)-party such that $s \le r$ and the former party contains not less spaces than the later one, there exists a scheme to divide the former party into two n-parties with one of them the later n-party.

Proof:

Let the former (n,r)-party be party A and let A_1 be the set of all x mice in party A having r

spaces, A_2 be the set of all the other $n-x$ mice in party A having $r-1$ spaces. Similarly, donate the amount of mice in the later party having s spaces by y.

If we have $r>s$, and $x \geq y$, we can find y mice in A_1. Let these y mice contribute s spaces each, and the other $n-y$ mice contribute $s-1$ spaces, and we have a requested (n,s)-party. After this manipulation, there are still $x-y$ mice having $r-s+1$ spaces and $n-x+y$ mice having $r-s$ spaces left. Thus party A is divided into a requested party and another n-party as well.

If $r>s$ and $x<y$, we let $y-x$ mice from A_2 and all x mice from A_1 contribute s spaces each, while the other $n-y$ mice contribute $s-1$ spaces, similarly we get the requested party and what left is a n-party also.

If $r=s$, since party A must be bigger than the requested party, we get $x \geq y$ directly. Thus we can make it using the same manipulation carried out in the situation of $r>s,\ x \geq y$.

In one word, no matter what situation we encounter, a means of doing the requested job is always guaranteed. QED.

Theorem 2 An (n,r)-party which has x spaces can be i-filled if $1 \leq i \leq n,\ 1 \leq x \leq i\binom{n}{i}$.

The following proof is a little complicated, so we will introduce our idea at the beginning of it. The main idea is to use induction. We are to prove '$n=k+1$' situation under the assumption that '$n=k$' is done. Some points are to be made in advance, namely special cases of i, and some inequalities which will be used(Those inequalities seem useless considering their somewhat silly meanings, while they are necessary in the proof since there are so many special cases and we will miss some if we are careless in dealing with those inequalities). Then in the core step, we will try dividing the original party into three parts, X_1, B and C. To make sure it can be divided as we wish, conditions are studied and special cases are worked out independently. Last but not least, we will fill up B and C separately using '$n=k$' hypothesis and add X_1 to the scheme of B to finish our proof.

Proof:

Perform induction on n.

a)When $n=1$,

$\because 1 \leq i \leq n = 1 \quad \therefore i = 1$

$\therefore 1 \leq x \leq i\binom{n}{i} = 1 \quad \therefore x = 1$

So we have only one space now, and of course it can be 1-filled.

b) Assume this proposition holds when $n=k$ $(k \geq 1)$. When $n=k+1$, it can be proved by the following six steps:

I. In this step we consider the special cases of $i=1,n$.

According to **Definition 4**, we have $n(r-1) < x \leq nr$ ……**(1)**

As a result, when $i=1$, $n(r-1) < x \leq i\binom{n}{i} = n$, so $r=1$.

Therefore the only possibility is that there are x mice having one space while the other having nothing. Filling in blanks one by one gives the desired solution.

Similarly, when $i=n$, the only possibility is that $x \leq n,\ r=1$. If $x<n$ in this case, filling is not needed. If $x=n$, filling in the only i-statuses solves the problem.

So we assume $2 \leq i \leq n-1$ in the next five steps.

II. This step is performed to get some important inequalities in order to make it easier when we are trying to work out other inequalities in the following steps.

According to **(1)** and the conditions of this theorem, we have $n(r-1) < x \le i\binom{n}{i}$.

Use **Lemma 2** and we get $n(r-1) < i\binom{n}{i} = n\binom{n-1}{i-1}$ $\quad\therefore r-1 < \binom{n-1}{i-1}$

Since each side of the inequality is integer, $r \le \binom{n-1}{i-1}$. ……**(2)**

From **(1)**, we have $r \ge \frac{x}{n}$. ……**(3)**

III. This step gives one of the parties which we are going to divide our original party into since we intend to use division to perform induction.

Let $r(i-1) \equiv y(mod\, n-1)$ $(1 \le y \le n-1)$, so $n-1 | r(i-1)-y$.

Take out one row of r spaces, namely X$_1$, from the original (n,r)-party, we get an $(n-1)$-party A, which has $x-r$ spaces.

Now think of an $\left(n-1, \frac{r(i-1)-y}{n-1}+1\right)$-party B, y rows of which has $\frac{r(i-1)-y}{n-1}+1$ spaces.

B has $y\left(\frac{r(i-1)-y}{n-1}+1\right)+(n-1-y)\frac{r(i-1)-y}{n-1} = r(i-1)$ spaces in total.

$\because\ i \le n-1 \qquad \therefore\ \frac{r(i-1)-y}{n-1} < \frac{r(i-1)}{n-1} < r$

Since we have $\frac{r(i-1)-y}{n-1} \in N_+$, again integers are recognized at both sides,

$\therefore\ \frac{r(i-1)-y}{n-1} \le r-1$, in other words, $\frac{r(i-1)-y}{n-1}+1 \le r$ ……**(4)**

IV. In this step we will consider a special case in which the party to be divided turns out to be even smaller than the desired party we are to get by division.

If the amount of spaces B has, $r(i-1)$, is bigger than that of A, $x-r$, we have $r(i-1) > x-r$, which gives us $x < ir$. While $i \le n-1$, according to **(1)**, $i(r-1) < n(r-1) < x < ir$

Thus the original (n,r)-party can only accommodate $r-1$ i-statuses, so the problem is converted to whether we can fill it with $r-1$ different i-statuses.

Observe that we can fill a perfect $(n,r-1)$-party with $r-1$ i-statuses by adding one i-status to each 'column' so that they can be designed freely and separately. According to **(2)** we have $\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1} > \binom{n-1}{i-1} - 1 \ge r-1$, so these $r-1$ i-statuses can be distinct, which implies an (n,r)-party which contains a perfect $(n,r-1)$-party can also be filled with $r-1$ distinct i-statuses. The proof is already finished in this case.

V. This section will illustrate how we are to use division in proving the theorem.

Else we have $r(i-1) \le x-r$, which, according to **(4)**, indicates A and B satisfy the condition of **Lemma 3**. As a result, A can be divided to B and another $(n-1)$-party C. According to **(2)**, $r(i-1) \le (i-1)\binom{n-1}{i-1}$, and $1 \le i \le n-1$, so B satisfies the condition of **Theorem 2** when $n = k$ and $i' = i-1$. Thus this party can be $(i-1)$-filled and there will not be a single space left because the number of spaces in B is divisible by $i-1$. Append these $(i-1)$-statuses with component X_1 and we have r i-statuses now which fill in X_1 and B perfectly. Assuming party C can be i-filled, we now have B+X_1 and C filled separately. As no i-status filling C has component X_1 while all other i-statuses have, these i-statuses are distinct, which gives a solution finally to i-fill A. So the only thing we are to prove is C can be i-filled.

VI. In the final step, we will finish the proof by proving the other part of the division which we did not consider is also conquerable.

Since $1 \leq i \leq n-1$, and there are $x-r-r(i-1)=x-ir$ spaces in C, perform **Lemma 2** and **(3)** gives $x-ir \leq x-i\frac{x}{n}=\left(1-\frac{i}{n}\right)x \leq \left(1-\frac{i}{n}\right)i\binom{n}{i}=i\binom{n}{i}-i\frac{i}{n}\binom{n}{i}=$ $i\binom{n}{i}-i\binom{n-1}{i-1}=i\binom{n-1}{i}$

Therefore C satisfies **Theorem 2** when $n=k$ and $i'=i$, which indicates that it can be i-filled. So **Theorem 2** when $n=k+1$ is proved.

Combining a) and b) gives the proof of **Theorem 2**. QED.

Theorem 2 guarantees that perfect $(n, r-\sum_{i=1}^{Z(n,r)-1}\binom{n-1}{i-1})$-party can be $Z(n,r)$-filled, which implies that perfect (n,r)-party can be filled with all 0,1,⋯,$(Z(n,r)-1)$-statuses (Considering **Lemma 1**) and be $Z(n,r)$-filled, that is, feasibility of Chart-Filling Method. The proof also indicates a way of constructing such a scheme by recursion.

2.2.3 Conclusions and Other Variations

In this chapter we solved **problem 2** by proving a stronger proposition as to Chart-Filling Method we developed specially for this problem. The concept of status can be seen as a generalization of binary solution to the original problem in chapter 1, which is so important that it is the core concept of this whole chapter. From this concept we get a closer view of how schemes are made and how to examine whether they will work properly. This concept will also be employed in the next few sections.

Now that **problem 2** has been solved, let's consider what will happen if we do more than one experiments. If we try it with the idea of generalizing the concept of status, we will reach a new concept that each digit of the status can have not only 0 and 1 two possible values. However, this does not throw light on the question since it is too complex. So we try to understand it in another way, that is, take each experiment as one chart. Thus the generalized problem becomes a Cube-Filling problem. Generalizing restrictions of the problem as well gives two different versions of it considering whether a mouse will die after taking the poison: In the death version, each status has at most one component in one 'mouse plane'(in contrast to 'mouse row' in **problem 2**) while in the athanasia version each status is allowed to have one component in each 'mouse row' of each experiment.

The previous generalization can be generalized again if we consider the abstract concept of n-dimensional-cube-filling without using concrete concepts such as mouse and poison. However, none of these widen problems can be solved according to skills we developed, as division method used in proof of **Theorem 2** cannot be generalized to more than two dimensions.

2.3 Two-Poison Replacement and Grouping-Coordinate Scheme

2.3.1 Coordinate Scheme

In this chapter, we will try the ‘two-poison’ problem:

Problem 3 There are plenty bottles of medicine with two of them poisoned. A mouse will die if it takes the poison, otherwise it will remain alive. With n mice and one experiment, we can check out that some proportion of those medicines is nontoxic. What is the maximum ratio of nontoxic proportion we are able to get? (The maximum ratio is noted $c(n)$)

We are to introduce the ‘Coordinate Scheme’, which was first given by yac using the example of $n = 10$.

First of all, divide the ten mice into two groups, and all the bottles into 25 groups evenly (This is the reason why we assume there are ‘plenty of’ bottles, or we will have to examine the number’s divisibility by 25). Put these groups of medicines as follow, and let each mouse take all the medicine of its row/column.

	Mouse 6	**Mouse 7***	**Mouse 8**	**Mouse 9***	**Mouse 10**
Mouse 1	safe	safe	safe	safe	safe
Mouse 2*	safe	hazard	safe	hazard	safe
Mouse 3	safe	safe	safe	safe	safe
Mouse 4*	safe	hazard	safe	hazard	safe
Mouse 5	safe	safe	safe	safe	safe

Provide that mouse 2,4,7,9 die, the medicines they took contain poisons. As a result, the poisons are in the four ‘hazard’ districts, while they are not located exactly.

According to this, we can always find at most four districts containing the poisons, therefore the nontoxic proportion is $\frac{25-4}{25} = 84\%$.

This method can be reinforced a little, and then we have:

	Mouse 6	**Mouse 7***	**Mouse 8**	**Mouse 9***	**Mouse 10**	
Mouse 1	safe	safe	safe	safe	safe	safe
Mouse 2*	safe	hazard	safe	hazard	safe	safe
Mouse 3	safe	safe	safe	safe	safe	safe
Mouse 4	safe	safe	safe	safe	safe	safe
Mouse 5	safe	safe	safe	safe	safe	safe
	safe	hazard	safe	hazard	safe	safe

In this scheme, we leave one row and one column untested. Assuming mouse 2 is the only unlucky one among the first five mice, we have row 6 a dangerous row also. So we can, again, always determine at most four districts which are hazarded, therefore the nontoxic proportion is $\frac{36-4}{36} \approx 88.9\%$.

But that is still not the end! If we divide those 10 mice into three groups, namely 3, 3, 4, and adopt the same manipulation (This time it is three dimensional), we will have a nontoxic proportion of $\frac{4\times4\times5-8}{4\times4\times5}=90\%$. This is because each group will locate two 'planes', therefore we have eight intersected districts instead of four, and in each group we leave out one 'plane' untested, which is why there are $4\times4\times5$ districts in total.

It can be seen that the more groups we divide our mice into, the more uncertainty we will have. In general, an m-dimensional (Each group can be seen as a dimension, in other words, they are separate coordinates, which is the reason this scheme is named) scheme gives us 2^m hazard districts. So in the case of $n=10$, this three dimensional scheme gives the best answer.

Now we can easily conclude a general method out of this, that is to divide n mice into k groups, namely $a_1, a_2, \dots, a_k$, and we can determine a nontoxic proportion of $1-\frac{2^k}{\prod_{i=1}^{k}(a_i+1)}$. To make it optimal, we should have a_i evenly, which is why we divide 10 mice into 5 and 5 instead of 4 and 6 or other groups. The value of k remains arbitrary, while we can determine it within n try (Let k vary from 1 to n), so obtaining the optimal solution will not be very hard.

2.3.2 Grouping Method and Coordinate-Grouping Scheme

The previous scheme by the website writer is really skillful, while we have a seemingly less graceful but more general idea. The thought involved is quite easy, that is, to divide all medicines into several groups, and use some schemes similar to those we used when examining one poison to determine the exact location of the two poisons. Notice that we said 'some schemes' instead of a definite one, which makes it quite mysterious. This is because, for example, in a simple situation of $n=5$, we cannot even work out easily whether there exists a manipulation better than having five mice examine one group each and leave the sixth group untested. This difficulty raises **problem 3***:

Problem 3* There are some bottles of liquid with two of them poisoned. A mouse will die if it takes the poison, otherwise it will remain alive. Now we have n mice to check out which bottle contains the poison within one experiment. What is the maximum number of bottles we are able to deal with? (The answer is noted $f(n)$)

In fact, there is still a difference between Grouping Method and **problem 3***, as two poisons can unfortunately be divided to the same group. In this case, if we insist in finding two groups containing the two poisons, the result can be confusing. The way to mediate this dissension is to accept a solution locating two groups containing the poisoned group, and try to find one group if the result is confusing. Further description and feasibility of this measure will be discussed in the next section.

Perhaps readers of this section will be impatient since Grouping Method has not yet given an answer and probably can never fight against the sophisticated Coordinate Scheme. However, what gives us courage to continue studying the Grouping Method is that this method can be added into the Coordinate Scheme! Notice that in Coordinate Scheme, we use each group as an independent coordinate, and in each of these coordinates, we actually set a scheme to determine the exact two 'planes' the poisons are in. Thus if we employ Grouping Method in each group, Coordinate-Grouping Scheme is now reached!

In general, this scheme can be written as follows: divide n mice into k groups, namely $a_1, a_2, \dots, a_k$, and we can determine a nontoxic proportion of $1 - \frac{2^k}{\prod_{i=1}^{k} f(a_i)}$. Since attrbutes of $f(n)$ are unknown now, we cannot take for granted that groups should be divided evenly in this scheme.

2.3.3 New Problem and Compatibility

In this section, we will work out a route to develop $f(n)$ as well as to solve the compatibility problem raised in the previous section(two poisons in one group) between $f(n)$ and the original problem.

Attempts of small scale tell us $f(n)$ is quite difficult to find. We can deduce $f(n) \geq n+1$ easily by letting each mouse tests one group and leave the last group untested. We can also claim that $f(n)$ is monotonically increasing, as one more mouse can test one more group while the other $n-1$ mice do the same as checking $f(n-1)$ groups. A trivial upper bound can as well be reached from $f(n+1) \leq 2f(n)$, since we can divide the statuses into two groups considering whether the last digit is 1 and claim that each group has not more than $f(n)$ statuses. But further study is difficult, which we shall show in the next three sections. Our aim is to (1) try giving a nontrivial upper bound and lower bound for $f(n)$, (2)give some nontrivial examples of small scale probably by computer.

As it is very important, we donate the attribute that $f(n)$ is monotonically increasing by **Lemma 4**.

First of all, we should give some notation in order to describe the problem. It is a good idea to use the same concept we used in the previous chapter, which is 'status'. Each group has a 'status' which describes how it is fed to mice and what the result will be if it is poison. But now we have two poisons, therefore we probably will get an integrated result for two groups. Thus we can define a new calculation between two statuses:

Definition 7 Or/Union: For two status a and b, let 'a or b' be an n-digit binary number whose ith digit is 1 as long as digit 1 appears at the same digit in one of the two statuses. This is also called the union of a and b.

'a or b' is actually an informatics concept having the same meaning, besides it is also similar to the concept of 'union' in the theory of set, which leads to a possibility of converting this problem to other problems.

Now we can claim that the statement 'able to determine two poisons' is same to 'for all pairs of status, (a,b), a or b are distinct', since a test works if and only if different cases give distinct results.

Now let's focus again on the compatibility problem which concerns two poisons in one group. If the result differs from every possible answer a two-group union gives, we happily accept the fact that the two poisons are in the same group. Since we already have all the unions different, it is apparent that statuses themselves are distinct. Therefore we can determine which group the two poisons are in. Thus the only problem turns out to be, if we really determine two seemingly hazard groups, a and b, while the poisons are in fact in one group, c, can we miss it? Since a or b, a or c should be distinct if $b \neq c$, and we have a or b = c as a condition, a or c = a or (a or b)=a or b, contradiction. Thus we cannot miss c. In all, when two poisons are in one group, we can always find at most two groups containing the hazard group, which solves the dissension finally.

2.3.4 Qualitative Analysis of $f(n)$

We will give an exponential upper bound first by calculating the maximum amount of statuses possible.

Theorem 3 $f(n) \leq 2^{\frac{n+1}{2}}$

Proof:

Similar to proof of **Theorem 1**, we notice that since unions of all pairs of statuses give distinct answers, there are altogether $\binom{f(n)}{2}$ such answers. The amount must not be more than the amount of all n-digit binary numbers, which is 2^n. Therefore we have $\binom{f(n)}{2} \leq 2^n$, thus $f(n) \leq \frac{1+\sqrt{1+2^{n+3}}}{2} < \frac{1+\left(1+\sqrt{2^{n+3}}\right)}{2} = 1 + 2^{\frac{n+1}{2}}$. As both sides are integer, $f(n) \leq 2^{\frac{n+1}{2}}$. QED.

The experience of attempting to get a nontrivial answer reaching $f(n)$ tells us this upper bound is very high indeed, and the method we used in proving it has nothing to do with how to make an example. So this theorem only gives us a border, saying $f(n)$ cannot raise faster than exponential function.

Though this proof of upper bound seems to be not complicated enough to work out a satisfying answer, we honestly can not improve it significantly since situations can not be easily assorted and structures can not be recognized. Moreover, we conjecture $f(n)$ to grow exponently due to our lower bound and computer generated data (and instinct).

Attempt of giving a lower bound using similar manipulation proved to be too difficult for us, but it is possible that we give a nontrivial lower bound by constructing a new example from a known one, in other words, by recursion.

Theorem 4 $f(n + 2\lceil\log_2(f(n) + 1)\rceil) \geq 2f(n)$

Proof:

Let k be $\lceil\log_2(f(n) + 1)\rceil$. Assuming we have an example in the case of n, which are $f(n)$ statuses satisfying the union-distinct condition. Now we will give $2f(n)$ $(n + 2k)$-digit binary numbers, all unions of which are distinct as well.

For the first half, add $2k$ digits to these $f(n)$ status, and let the first k ones be zero, the $(k + 1)\text{th}$ to the $2k\text{th}$ be the binary representations of one till $f(n)$(the reason why we choose k digits is that this is the least amount to show the binary representation of $f(n)$). To generate the second half, we simply produce the added part by exchanging the first k digits and the last k ones produced in the first half.

An example is showed as follows.

Group1	Group2
000000 0000 0001	000000 0001 0000
000011 0000 0010	000011 0010 0000
010101 0000 0011	001100 0011 0000
001100 0000 0100	010101 0100 0000
011010 0000 0101	011010 0101 0000
100110 0000 0110	100110 0110 0000
101001 0000 0111	101001 0111 0000
110000 0000 1000	110000 1000 0000

The first six figures are from an example of $n = 6$, and the following eight ones are new digits we added. Notice that in order to show '8' in binary representation, we need four digits instead of three even though $\log_2 8 = 3$, since zero is not included.

If two statuses are chosen from the same group, we can first use whether the last k digits are zero to determine which group they are in. Then, as we suppose the first n digits come from a proper example of n, distinctness is guaranteed. If these two statuses are from different groups, it is even easier since we can tell which one they are by transferring the last $2k$ digits to binary numbers and choose the corresponding status in each group directly.

Therefore the recursion holds, QED.

Now let's get a closed form of lower bound from **Theorem 4**.

Because logarithm and ceiling function are used, we shall first generate a better recursive form to make it easier to deal with. For every $x \in [1, +\infty)$, let

$$f(x) = (\lfloor x \rfloor + 1 - x) f(\lfloor x \rfloor) + (x - \lfloor x \rfloor) f(\lfloor x \rfloor + 1)$$

which is the function we get connecting adjacent discrete points. Observe that

$$log_2(p+1) \le log_2 p + 1 \quad (p \ge 1)$$

We can get

$$\lceil 2\, log_2(p+1) \rceil \le 2\, log_2(p+1) + 2 \le 2\,(log_2 p + 1) + 2 = 2\, log_2 p + 4 \quad (p \ge 1),$$

Therefore, with the help of **Lemma 4**, which will be cited every now and then without claiming in this chapter, we have

$$\begin{aligned} f(x + 2\, log_2 f(x) + 4) &\ge f(x + \lceil 2\, log_2(f(x) + 1) \rceil) \\ &\ge f(n + \lceil 2\, log_2(f(n) + 1) \rceil) \\ &\ge 2f(n) \\ &> 2f(x-1) \qquad (x \ge 2, n = \lfloor x \rfloor) \end{aligned}$$

Thus, $2f(x) < f(x + 1 + 2log_2 f(x+1) + 4) \le f(x + 1 + 2(log_2 f(x) + 1) + 4) = f(x + 2log_2 f(x) + 7)(x \ge 1)$, because $f(x+1) \le 2f(x)$ considering the last digit separately. The inequality has been adjusted for real number now.

Let $g(x) = log_2 f(x)$, then we have $2^{g(x)+1} < 2^{g(x+2g(x)+7)} (x \ge 1)$, so $g(x) + 1 < g(x + 2g(x) + 7)$.

We can prove the following theorem, which leads to a lower bound of traditional form with the help of function g.

Theorem 5 $f(x) > 2^{\sqrt{x+3}-4}$

Proof:

Let $\{a_n\}: a_1 = 1, a_{n+1} = a_n + 2n + 7$, we will prove by induction that $g(a_n) \geq n$.

(1) $g(a_1) = g(1) = log_2 f(1) = 1 (f(1) = 2$ is quite obvious), so $n = 1$ holds.

(2) Assuming $g(a_n) \geq n$ when $n = k$,

when $n = k + 1$, we have $g(a_{k+1}) = g(a_k + 2k + 7) \geq g(x + 2k + 7) = g(x + 2g(x) + 7) > g(x) + 1 = k + 1$,

where $g^{-1}(k) = x \leq a_k$ because **Lemma 4** guarantees that $g(x) = log_2 f(x)$ has inversion, and $g^{-1}(k)$ exists because $f(x)$ for real number is continuous, which can be easily noticed considering the manipulation used.

Combining (1) and (2) gives $g(a_n) \geq n$

$\because a_n = a_{n-1} + 2(n-1) + 7 = a_{n-2} + 2(n-2) + 7 + 2(n-1) + 7 = \cdots = a_1 + 7(n-1) + 2(n-1+n-2+\cdots+1) = n^2 + 6n + 6$,

$\therefore f(n^2 + 6n + 6) \geq 2^n$.

Let $n^2 + 6n + 6 \leq x < (n+1)^2 + 6(n+1) + 6$,

$\therefore f(x) \geq 2^{[\sqrt{x+3}-3]} > 2^{\sqrt{x+3}-4}$, QED.

In fact, the initialization (where our recursion starts) can be updated with the data we will get from computer programs in section 2.3.5, and this can improve the lower bound somewhat.

2.3.5 Further Study of Lower Bounds

Still, the lower bound we worked out can be improved, since we can do some further refinements on the way we generate new examples from $f(n)$ given statuses of n digits.

Our first thought on it is to use definite amount of digits to label some 'groups'. For instance, '10' and '01' can be added to two groups so that we can easily recognize whether the statuses for an union are from the same group (If not, these two digits will be '11'). And if they are, apparently we can see which group they are from as well.

If two statuses are from the same group, then the problem is already solved due to our heritage, else, as we think it will not be hard to work out what the two statuses (first n digits) are from their union (further study of this will be discussed after several paragraphs), the only problem is to see if these two are A from Group 1 and B from Group 2, or A from Group 2 and B from Group 1. This is not very difficult since the scheme below will help:

Group 1:	A0, B1, C0
Group 2:	A0, B0, C1

In this scheme (Assuming now we are sure these two statuses are not from the same group), not only A and B can be parted according to groups, but B and C, C and A can be parted as well. Observing that it can also be used to three groups instead of three statuses only, we can add more digits using the same method to distinguish more pairs, for different digits are independent. The example of 9 statuses using 2 digits is showed below:

Group 1:	A00, B01, C00, D10, E11, F10, G00, H01, I00
Group 2:	A00, B00, C01, D00, E00, F01, G10, H10, I11

In this example, If the two statuses are from different sections of (ABC), (DEF) and (GHI), they can be distinguished with the first digit considering a group as one thing([Group1 (ABC) or Group2(DEF)] is different from [Group2(ABC) or Group1(DEF)] by the last but one digit, 'or' here means union), else they are from the same group, say, (ABC), and can be parted with the second digit. Consequently, n digits can be used to part 3^n statuses using the same manipulation somewhat similar to radix 3 representation.

Generally, this method can be carried out as follows. 3^n statuses can be numbered by radix 3 representation. Then we write those statuses twice in two groups. In the first group, we add '01' to it and then the radix 3 representation with '2' written '0' to get our Group 1(When n=2, it gives exactly the preceding example. For instance, number for C should be $(02)_3$ but as '2' is written '0', it is '00' actually). Statuses in the other group is produced similarly, we add '10' to it and the radix 3 representation where '1' is written '0' and '2' is written '1'.

Now we are nearly finished with one last problem. Considering there are some situations where a or a = a or b, we can not always tell from their union what the two statuses exactly are since they can be the same status from two groups. Hence our scheme requires statuses' unions being different from statuses themselves. This can also be satisfied by adding another digit. Define a or b=a as a 'includes' b, then we can divide all statuses into two groups, A={$x|x$ is not contained by any other statuses in our scheme} and B={x|there exists at least one status in our statuses that contains x}. Inclusion will not exist in B since if a contains b, b contains c and b, c belongs to B, we have a or $c = a = a$ or b, contradiction. As A's definition also forbids inclusion's existence, we can add a digit '0' to all statuses in A and a digit '1' to all statuses in B to make inclusion extinct, for statuses in A cannot contain statuses in B any longer due to the added digit. Now that unions and statuses are all distinct, we can tell exactly what two statuses are by looking only at their union. This can also be a solution to the 'two poisons in one group' problem if we want the most precise location and are willing to pay one more mouse for it.

We have altogether used 1 digit to annihilate inclusion, 2 digits to distinguish groups and $\lceil log_3 f(n)\rceil$ digits to recognize status in distinct groups. Thus $f(n+3+\lceil log_3 f(n)\rceil) \geq 2f(n)$.

Using the same method in obtaining **Theorem 5** from **Theorem 4**, we have a similar answer which acts like $3.4^{\sqrt{x}}$ but looks more frightening. The main steps are as follows.

We generalize this inequality to real number and get

$$f(x+4+log_3 f(x)) \geq 2f(x-1).$$

Let $t = x-1$ and we have

$$f(t+5+log_3 2+log_3 f(t)) \geq 2f(t) \text{ since } f(x+1) \leq 2f(x).$$

Let $g(t) = log_3 f(t)$ and we can deduce that

$$g(t+5+log_3 2+g(t)) \geq log_3 2+g(t).$$

Let $\{a_n\}: a_1 = 1, a_{n+1} = a_n + 5 + (n+1)log_3 2$ and perform induction using the function inequality, we can get $g(a_n) \geq nlog_3 2$.

Convert it back to $f(n)$ gives an answer similar to the one we got in the previous section but much more complex. For convenience, we take it as $f(x) \geq 3^{\sqrt{1.26x+34.1}-5.95}$.

Cite $f(20) \geq 220$ (Computer generated in section 2.3.5) and let $a_1 = 20$ to improve it (this requires x to be big enough, but because we are considering a tendency, a restriction such as $x > 20$ is not big deal), we have $f(x) \geq 3^{\sqrt{1.26x+79.25}-5.95}$, which is the strongest answer we are able to get now. The exact form of it is $f(x) \geq 3^{\sqrt{4ax+b^2-4ac}-5-3a}$, where $a = \frac{log_3 2}{2}$, $b = a + 5 + log_3 110$, and $c = 15 - 2a - log_3 110$.

Further generalization can also be thought of by making more groups than two. If we have m groups, there will be m digits used to distinguish which group or which two groups the statuses are from through their union. To avoid inclusion, we still need only 1 digit. Now we have the group numbers and first n digits of the two statuses, so we only need to tell apart what looks like Aa, Bb and Ab, Ba.

In fact, we have a way of requiring $\lceil log_3 f(n) \rceil \lceil log_2 m \rceil$ more digits to do it. $\lceil log_3 f(n) \rceil$ digits are considered a 'squad' as it is in the previous scheme, and different squads are used to compare different pairs of groups. With the first squad, we are to compare Group 1 to $\left\lfloor \frac{m}{2} \right\rfloor$ with Group $\left\lfloor \frac{m}{2} \right\rfloor + 1$ to m by giving the same squad we used in Group 1 in the previous scheme to the first half and that of Group 2 to the second half so that if the two statuses come from different halves, we can tell what they are exactly. Similarly, 'Group 1 notation' is adopted in the second squad of Group 1 to $\left\lfloor \frac{m}{4} \right\rfloor$ and Group $\left\lfloor \frac{m}{2} \right\rfloor + 1$ to $\left\lfloor \frac{3m}{4} \right\rfloor$ while those of the others are 'Group 2 notation', and the two statuses can be settled if their group numbers are from different groups in this manner, and so forth. The only main difference between this manipulation and the scheme we discussed previously in this section is that it uses radix 2 representation and is more confusing. Thus $\lceil log_2 m \rceil$ squads are needed as they are independent. An example of 4 groups and 3 statuses is showed below.

Group 1:	A*100000,	B*100011,	C*100000
Group 2:	A*010000,	B*010010,	C*010001
Group 3:	A*001000,	B*010001,	C*010010
Group 4:	A*000100,	B*000100,	C*000111

In this example, '*' means the digit for inclusion and the 4 digits after it is representation of groups. The next digit acts as the first squad with which we can compare Group 1,2 with Group 3,4 and the last digit represents the second squad.

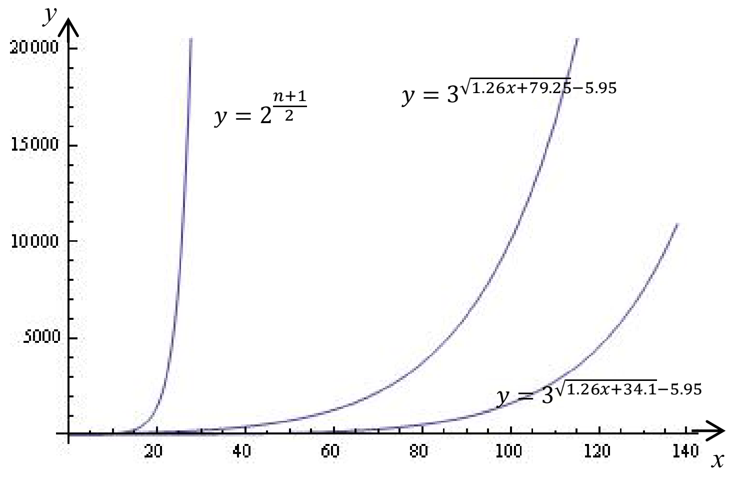
But look at the m digits for groups closely, we find it not necessary. The aim of these digits is only to tell which two groups the two statuses belong to, and we need only $m + 1$ digits to deal with $f(m)$ groups (The extra '1' is for avoiding inclusion).

Conclusion for all above is that besides the n digits our original $f(n)$ statuses require, we need 1 more digit to avoid inclusion, $m + 1$ digits to build $f(m)$ groups (Look out! Not only m groups now!), and $\lceil log_3 f(n) \rceil \lceil log_2 f(m) \rceil$ digits to tell the exact answer. Thus we have $f(m + n + 2 + \lceil log_3 f(n) \rceil \lceil log_2 f(m) \rceil) \geq f(m) f(n)$. Take a step backward and let both logarithms have the same basis 2, and we have m and n of the same status (not 'status' in our concept). Comparing it with Coordinate-Grouping Scheme, we have m and n two independent coordinates. Hence the result can be taken as getting the exact location by refining Coordinate-Grouping Scheme. So now even the scheme with uncertainty is also connected with lower bound of $f(n)$.

Thus still more coordinates can be added. When adding a new coordinate, we take the former ones as a whole, and similarly we need m_i more digits for group label, 1 more digit for inclusion, and $log_3(\prod_{j=1}^{i-1} f(m_j))\, log_2 f(m_i)$ digits more to avoid uncertainty as $log_3 \prod_{j=1}^{i-1} f(m_j)$ digits are considered one squad to compare two groups of 'hyper planes' now. Hence, we have

$$f[\sum_{i=1}^{p} m_i + p + \sum_{1\le i<j\le p} log_3 f(m_i) log_2 f(m_j)] \ge \prod_{i=1}^{p} f(m_i)$$

This is the general situation of our lower bounds. From it, an ocean of stronger lower bounds can be made out in theory. We can also use something like n groups instead of concrete numbers. However, we still cannot break the limitation of the lower bound acting like $a^{\sqrt{x}}$ while the upper bound grows exponently, because there are multiplications of logarithms in this inequality, and a lower bound exists only when degrees of both sides are the same.



This graph shows the upper bound and the two lower bounds we got in this section. Now we have achieved the goal of finding nontrival upper bounds and lower bounds. However, the upper bound still grows much faster as it rises exponently, which means space for improvement is still huge.

2.3.6 Quantitative Computation of $f(n)$

This section is aimed at finding a method to compute $f(n)$. The problem is so complicated that we can only get the precise value of $f(n)$ for very small n, and have to use an approximate algorithm to achieve a relatively good result. In the programs, we treat statuses as binary numbers and store each status as an integer. Bitwise operation 'or' is used to calculate the union of two statuses.

For small n, we can use a brute-force method, backtracking algorithm. In each step, we start with an existing scheme and try adding statuses to it. The process is as follows: the program enumerates all the n-digit 0-1 strings as the new status, and tests if the scheme remains valid after adding this status. If so, the program adds this status to the scheme and goes on trying to add more. If no status can be added, the program deletes the status that is added last and backtracks. We start from an empty scheme, and when the program terminates, it has traversed all possible schemes. Then we pick the one that contains the most statuses as the result. The complexity would be $O(2^{nl} \times l)$, where l stands for the upper bound of the answer.

After a little bit of thinking, we find an obvious optimization. We can express each scheme as a 0/1 matrix, with each row corresponding to a status, and each column corresponding to a mouse. And the problem changes to, given the number of columns in a matrix, n, and construct as much rows as possible, under the restriction that the 'union' of each pair of rows must be different. Then, it is easy to discover that changing the order of the rows arbitrarily won't affect the validity of the matrix. So each matrix can be represented by it is 'standard form', where the rows in the matrix are sorted according to their lexicographic order, i.e. ascending order if we treat them as binary numbers. Thus, all we have to do is to find all 'standard matrices'. In the program, we guarantee that every time we try to add a status, its lexicographic order must be after all the statuses in the existing scheme. Each standard matrix stands for $l!$ matrices, where l is the height of the matrix. So, now the complexity would be $O(\frac{2^{nl} \times l}{l!})$.

The program did all the work from $n = 1$ to $n = 6$ in roughly 1 second, while it used 62 seconds for $n = 7$. We tried $n = 8$, but for a whole day the program gave no result. (Those times are observed on a computer whose performance parameters are showed in Appendix III) It still needs optimization.

Apparently, columns in the matrix can also be sorted as rows can. While surprisingly, the rows and columns can be 'co-sorted', that is, we can rearrange the rows and columns of the matrix to obtain a 'uniform matrix', where both the rows and the columns are sorted according to their lexicographic order. In fact, we can accomplish this by performing these two operations alternatively:

a. sort the rows according to their lexicographic order if they are not already sorted

b. sort the columns according to their lexicographic order if they are not already sorted

When the process stops, we achieve our 'uniform matrix': The lexicographic order of the first row and the first column cannot get bigger after each operation, so they can only decrease and then remain unchanged after a certain time. After that time, the lexicographic order of the second row and the second column cannot get bigger, and so forth. After finite operations, the whole matrix is stable, and uniform matrix is obtained.

After introducing this optimization, we reduce the complexity to $O(\frac{2^{nl} \times l}{n! \times l!})$. Things get better, $n = 7$ is done within a second, and the program worked out $f(8) = 13$ in 1474 seconds. But for bigger n, it is still too ineffective. Since we have almost reduced the matrices that are needed to concern to minimal amount, $n = 8$ would be the limit for such algorithms. The results so far are:

n	1	2	3	4	5	6	7	8
$f(n)$	2	3	4	5	6	8	10	13

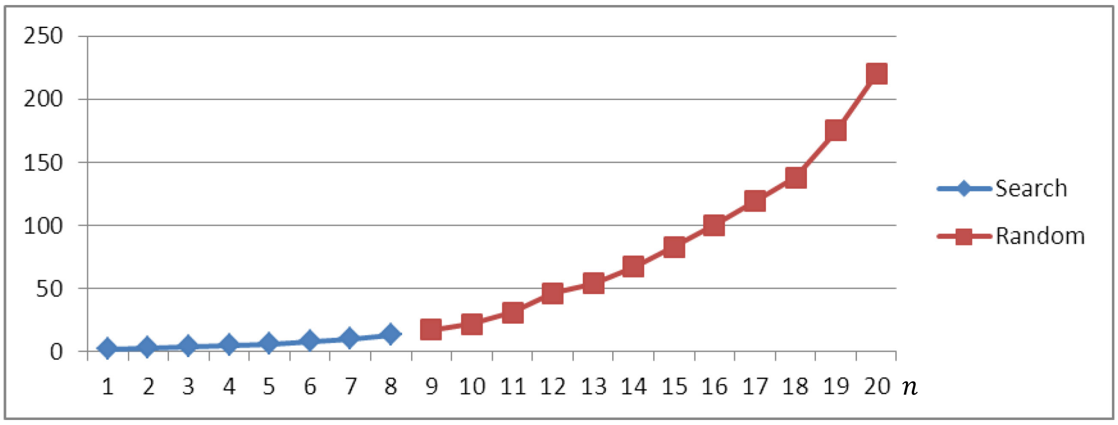
Another approach is approximate algorithm. Instead of enumerating all possible schemes, now we add statuses randomly under certain rules to construct schemes. The process is as follows: as before, we start with an empty scheme. Now for each step, we choose one status randomly from all valid statuses and add it to the scheme. If there is no status available, choose several statuses randomly from the scheme and delete them. To avoid being trapped in a bad solution, there is a counter, if the solution has not been improved for an amount of time, the process will be aborted and the program will restart from empty scheme. Besides, noticing that a status having too many '1' in it is unlikely to be part of the optimal solution, we limit the amount of '1' in a status.

We achieved the following results:

n	9	10	11	12	13	14	15	16	17	18	19	20
result	17	22	31	46	54	67	83	100	119	138	175	220

This program works well so far, but we can never be sure whether we have the the optimal answer or how much time we are expected to spend to get a good enough answer.

Some details in the implementation and the source code can be found in the appendix. More data of the programs such as time data of the random program are also available.



2.3.7 Informatics method and upper bound for $c(n)$

After this long journey of research on $f(n)$, we should come back to $c(n)$ in respect for our original problem. What we did can be seen as schemes of this problem, while we want a limit for such manipulations as well. Therefore in this section, we will focus on the upper bound of $c(n)$.

Due to our using descriptions such as 'big enough', estimation of $c(n)$ becomes quite tough as finite methods can not be performed easily. Thus we employ entropy method in informatics to solve this problem.

Assuming that we know nothing about which bottle contains the poison, which mouse will die or whatever connection between any of these events, all of these pieces of information are considered as independent random variables. Therefore we can assume that each bottle has $p = \frac{2}{x}$ possibility of being poisoned, and the possibility for whether or not a mouse will die after the experiment is supposed $\frac{1}{2}$. As a consequence, informatic entropy for each bottle is $E_1 = -p\, log_2\, p - (1-p)\, log_2(1-p)$ bit, the same thing for each mouse is 1 bit because mice here are considered typical dichotomous variables.

Assuming there are altogether x bottles, after the experiment $x[1-c(n)]$ bottles of them are confirmed safe while others are assumed totally unknown, we have $x' = [1 - c(n)]x$, $p' = \frac{2}{x'}$ and $E_1' = -p' \log_2 p' - (1 - p') \log_2(1 - p')$. Informatic entropy for all medicines reduces from $E = xE_1$ to $E' = x'E_1' = 2 \log_2 \frac{[1-c(n)]x}{2} + \{[1 - c(n)]x - 2\} \log_2 \frac{[1-c(n)]x}{[1-c(n)]x-2}$. As the $E - E'$ bit information is given by n mice, $n \geq E - E'$. Since x is very big, we now consider $\frac{x}{x-2}$ and $\frac{[1-c(n)]x}{[1-c(n)]x-2}$ to be 1, and a new inequality is obtained, $2 \log_2[1 - c(n)] \leq n$. Hence $c(n) \leq 1 - 2^{-\frac{n}{2}}$.

If we suppose Grouping Method will reach this upper bound, we can get $f(n) = 2^{\frac{n+2}{2}}$, which is very similar to the upper bound. This indicates if $f(n)$ grows exponently, Grouping Method will be a quite good manipulation in trying to solve this problem.

2.3.8 Summarization

In this chapter, we first raised **problem 3**, and introduced two methods, Grouping Method and Coordinate Scheme. Then we combine these two methods and get the Coordinate-Grouping Scheme. The more concrete **problem3*** is then raised and we tried to estimate an important function $f(n)$ in order to study it. Finally we come back to **problem 3** and gave further estimation to end the research.

The main part of this chapter is about $f(n)$. As we can see easily, there are lots of problems left, such as the true growth rate of $f(n)$. This function can also be generalized to, for instance, $f(n, m, k, p)$, which means the maximum amount of bottles we are able to examine when we have n mice, m experiments, k poisons among all medicines, and will be satisfied with p bottles of them confirmed safe.

Comparing Grouping Method with Coordinate Scheme again will show some surprising facts. If we take the estimated answer by the random program as true value, Grouping Method gives $c(10) \geq \frac{f(10)-2}{f(10)} \approx 90.9\%$, which is a little higher in contrast to 90%, the best solution given by more than one dimensional Coordinate-Grouping Schemes. We can also see that if $f(n)$ grows exponently, Grouping Method divides bottles into $f(n)$ groups and leaves only two of them hazard, while independent coordinates divides the same amount of groups since $p^a \times p^b = p^{a+b}$(divide $a + b$ mice into a and b two parts and let $f(n) = p^n$), but leaves more hazard groups. Hence surprisingly coordinate thought is of no use then.

Function $c(n)$ is really too arbitrary to have a certain value, and estimation made in the previous section also ignored too much complexity. All of them need improvements somehow.

3. Conclusion

After all those three generalizations, our long journey comes to a temporary end. As a review, we began with the easiest **problem 1** and had an insight into the Radix Scheme by changing one number to variable at a time. This problem was solved completely after proving **Theorem 1**. A restriction was added to reach **problem 2**, and a new solution, Chart-Filling Method was created. This was also settled as a result of **Theorem 2** after a long proving trip, after which we raised some further problems by turning to n dimensions. Moreover, we considered the seemingly easy **problem 3**, compared and combined two schemes, Grouping Method and Coordinate Scheme, to give a not very satisfying answer. The more concrete **problem 3*** was raised accordingly, and different functions were studied. Finally, that series of problems was ended with some nontrivial theorems and arithmetics as well as a lot more conjectures.

Notice that the concept of 'status' is considered a thread of the whole passage, and our study can also be taken as studies of this concept. In the first part, our scheme considered all statuses available as well as developed further notation using radix representation. In the second part, a function of status, namely the amount of digit 1 in a status, was considered and a statistic accordingly was studied to show the structure of a set of statuses instead of all. Finally we introduced a calculation (union) of statuses, which raised other problems discussed in the third part. Therefore the three topics are related by means of 'status'.

Though these problems appear to be only puzzles, they are of importance in some other areas. For instance, if we change the rule of 'or' to 'xor' in our programs of $f(n)$, we will reach something related to linear block codes which can correct two digits.

It is quite amazing that we have thought so much from such a simple 'mice and the poison' problem. In fact, there is a big fortune hidden behind uncomplicated facts waiting to be discovered. We not only obtained knowledge and skills but also experienced the charisma of math and enjoyment in researching after this wonderful peregrinate of mathematics.

Acknowledgement

We would like to thank the Yau High School Mathematics Awards Association and North East Yucai School first, for they give us such a wonderful opportunity of attending this great competition.

Our gratitude also goes to all our teachers, from whose devoted teaching and enlightening lectures we have benefited a lot and academically prepared for the thesis.

We feel grateful as well to our beloved parents who have always been supporting and encouraging us greatly without a word of complaint.

Appendix I Source Code of Search Algorithm in C++

Input n first and this program will give $f(n)$ and one corresponding solution in ‘result.txt’.

```
#include <iostream>
#include <fstream>

using namespace std;

ifstream fin;
ofstream fout,flog;                      // 'f': flag array, f[i] is true if i is
bool f[1048576];                           // the bitwise or of some d[j] and d[k]
int d[1000],n,ans;                       // 'd': the current scheme

void print(){                             // output the current solution to file
    int i,j;
    fout.open("result.txt");
    fout << n << ' ' << ans << endl;
    for (i=0;i<ans;i++){
        for (j=n-1;j>=0;j--)
            fout << (d[i]>>j)%2;
        fout << endl;
    }
    fout.close();
}

int add(int x,int z){                    // decides if status x can be added
    int i;                                 // and add it if so
    for (i=0;i<z;i++)
        if (f[x|d[i]])
            return i;
        else
            f[x|d[i]]=true;
    return z;
}

void remove(int x,int p){                // deletes status x
    for (p--;p>=0;p--)
        f[x|d[p]]=false;
}

void search(int k,int lim,int mask){
                                          // 'k' is the current depth
                                          // 'lim' is for maintaining the lexicographic
                                          // order of rows and 'mask' for columns
    int i,j,t;
    if (k>ans){
        ans=k;
        print();
    }
    for (i=lim;i<1<<n;i++)
        if ((((i | ~ (i >> 1)) | mask) & ((1 << (n-1))-1)) == ((1 << (n-1))-1)){
                                          // a complex bitwise operation
            t=add(i,k);                   // try and add
            if (t==k){
                d[k]=i;
                search(k+1,i+1,mask | i ^ (i >> 1));
                                          // recursive call
            }
            remove(i,t);                  // delete
        }
}

int main(){
    cin >> n;
    ans=0;
    search(0,0,0);
    return 0;
}
```

Appendix II Source Code of Random Construction in Free Pascal

This program should also have been in C++, but we are not quite familiar with the random number generator in it, so we use Free Pascal instead.

Parameters, namely n, tle, dl, bk, and zz, should be inputted in advance in 'config.txt', and the results are given in a file called 'result.txt'.

tle: the standard time which we uses for deciding whether we are in trouble and should give up the existing scheme

dl: the maximum amount of 1's in one status

bk: how many statuses will be deleted at a time

zz: total times the program will run

The names for them are the same with those in our program.

```
var
   f,f2:array[0..4194304]of boolean;
   q:array[1..4194304]of longint;                    // store numbers that contains less '1'
   d:array[1..10000]of longint;
   n,i,j,t,z,h,ans,mt,ct,fc,s,bk,dl,zz,zt,tle:longint;
   fi,fo:text;

function count1(x:longint):longint;        // count the number of '1's in binary representation of x
   begin
      x:=(x and $55555555)+((x shr 1) and $55555555);
      x:=(x and $33333333)+((x shr 2) and $33333333);
      x:=(x and $0F0F0F0F)+((x shr 4) and $0F0F0F0F);
      x:=(x and $00FF00FF)+((x shr 8) and $00FF00FF);
      x:=(x and $0000FFFF)+(x shr 16);
      exit(x);
   end;

function check(m:longint):boolean;                   // decides if status m can be added
   var
      i,j:longint;
   begin
      for i:=1 to z do begin
         if f[m or d[i]] then begin
            for j:=1 to i-1 do
               f[m or d[j]]:=false;
            exit(false);
         end
         else f[m or d[i]]:=true;
      end;
      for i:=1 to z do
         f[m or d[i]]:=false;
      exit(true);
   end;

procedure add(m:longint);                                  // adds m
   var
      i:longint;
   begin
      for i:=1 to z do
         f[m or d[i]]:=true;
      inc(z);
      d[z]:=m;
   end;

procedure delete(p:longint);                             // deletes m
   var
      i:longint;
   begin
      t:=d[p];
      d[p]:=d[z];
      d[z]:=t;
      for i:=1 to z-1 do
```

```
            f[d[z] or d[i]]:=false;
        dec(z);
    end;

begin
    randomize;
    assign(fi,'config.txt');
    reset(fi);
    readln(fi,n);
    readln(fi,tle);                // tle: the predetermined upper limit for timer
    readln(fi,dl);                 // dl: limit of number of '1's
    readln(fi,bk);                 // bk: how many statuses will be deleted at a time
    readln(fi,zz);                 // zz: total times the program will run
    close(fi);
    ans:=0;
    s:=0;
    for i:=0 to (1 shl n)-1 do
        if count1(i)<=dl then begin
            inc(s);
            q[s]:=i;
        end;
    for zt:=1 to zz do begin
        fillchar(f,1 shl n,0);
        fillchar(f2,1 shl n,0);
        z:=0;
        mt:=0;
        fc:=0;
        while true do begin
            ct:=0;
            for i:=1 to s do                                  // pick a random one to add
                if not f2[q[i]] then if check(q[i]) then begin
                    inc(ct);
                    if random(ct)=0 then h:=q[i];
                end
                else f2[q[i]]:=true;
            if ct=0 then begin                                // failed to add, delete
                for j:=1 to bk do
                    if z<>0 then delete(1+random(z));
                fillchar(f2,1 shl n,0);
            end
            else add(h);
            if z>mt then begin                                // solution improved, reset timer
                mt:=z;
                fc:=0;
            end
            else begin                                        // set timer
                inc(fc);
                if fc>=tle then break;                        // time out!
            end;
        end;
        if z>ans then begin                                   // output
            ans:=z;
            assign(fo,'result.txt');
            rewrite(fo);
            writeln(fo,n,' ',z);
            for i:=1 to z do begin
                for j:=0 to n-1 do
                    if odd(d[i] shr j) then write(fo,1)
                    else write(fo,0);
                writeln(fo);
            end;
            close(fo);
        end;
    end;
end.
```

Appendix III Time data for Random Program

Computer environment (The same computer is used to achieve results in the search program):

AMD Phenom 8750 2.41GHz

1.00GB RAM

Microsoft Windows XP SP3

For this program, the longer you run, the larger chance there is for you to get a better result. Also, the process is random, so it is meaningless to talk about total run time or the answer produced by a specific run. Therefore we'll present the data of the average answer given by the program and average time used for each run in order to show the average ability of our program.

tle: the standard time which we uses for deciding whether we are in trouble and should give up the existing scheme

dl: the maximum amount of 1's in one status

bk here is fixed at 3, so it is omitted.

n=9			
tle	dl	ans	time(s)
100	9	11. 09	0. 00269
	4	11. 48	0. 00131
	3	11. 98	0. 00081
200	9	11. 18	0. 00466
	4	11. 46	0. 00235
	3	11. 97	0. 00148
500	9	11. 27	0. 00995
	4	11. 47	0. 00524
	3	12. 14	0. 00334
1000	9	11. 34	0. 01808
	4	11. 56	0. 00983
	3	12. 25	0. 00622
2000	9	11. 46	0. 03451
	4	11. 67	0. 01955
	3	12. 32	0. 0112

n=10			
tle	dl	ans	time(s)
100	10	15. 09	0. 00577
	5	15. 17	0. 00348
	4	15. 96	0. 00217
200	10	15. 23	0. 00993
	5	15. 27	0. 00615
	4	15. 99	0. 00372
500	10	15. 41	0. 02105
	5	15. 49	0. 01372
	4	16. 1	0. 00835
1000	10	15. 59	0. 03905
	5	15. 6	0. 02541
	4	16. 18	0. 01597
2000	10	15. 66	0. 07362
	5	15. 76	0. 04844
	4	16. 22	0. 02938

n=11			
tle	dl	ans	time(s)
100	11	20.42	0.0135
	5	20.62	0.0062
	4	22.17	0.0038
200	11	20.66	0.0231
	5	20.8	0.0109
	4	22.5	0.0066
500	11	21	0.0495
	5	21.04	0.024
	4	23.21	0.0149
1000	11	21.17	0.0908
	5	21.21	0.0459
	4	23.57	0.0285
2000	11	21.2	0.1754
	5	21.18	0.0885
	4	23.73	0.0543

n=12			
tle	dl	ans	time(s)
100	12	26.64	0.0319
	6	26.59	0.0183
	4	30.59	0.0069
200	12	27.3	0.0548
	6	27.25	0.0334
	4	32.6	0.0126
500	12	28.12	0.12
	6	28.1	0.0744
	4	35.55	0.0285
1000	12	28.98	0.2265
	6	28.68	0.1375
	4	37.32	0.0502
2000	12	30.05	0.4421
	6	30.01	0.2739
	4	38.75	0.0872

n=13			
tle	dl	ans	time(s)
100	13	34.07	0.0739
	6	34.68	0.0347
	4	41.18	0.0126
200	13	35.21	0.1302
	6	35.3	0.0596
	4	44	0.0227
500	13	35.87	0.2762
	6	35.95	0.1313
	4	46.45	0.0433
1000	13	36.38	0.4934
	6	36.28	0.2349
	4	47.01	0.0703
2000	13	36.59	0.9179
	6	36.5	0.4312
	4	47.29	0.119

n=14			
tle	dl	ans	time(s)
200	7	45. 41	0. 1884
	4	56. 72	0. 0368
500	7	46. 54	0. 4113
	4	58. 94	0. 0685
1000	7	46. 86	0. 7305
	4	59. 58	0. 1054
2000	7	46. 9	1. 302
	4	59. 73	0. 1774

n=15			
tle	dl	ans	time(s)
200	7	58. 41	0. 374
	4	71. 03	0. 0584
500	7	59. 98	0. 8206
	4	73. 31	0. 1035
1000	7	60. 71	1. 4462
	4	73. 98	0. 1668
2000	7	60. 86	2. 5917
	4	74. 54	0. 2725

n=16			
tle	dl	ans	time(s)
500	8	76. 28	2. 614
	4	89. 72	0. 168
1000	8	77. 83	4. 545
	4	90. 51	0. 26
2000	8	78. 36	8. 196
	4	91. 22	0. 427

n=17			
tle	dl	ans	time(s)
500	8	97. 72	5. 874
	4	108. 48	0. 261
1000	8	98. 79	9. 73
	4	109. 88	0. 408
2000	8	100. 59	18. 097
	4	110. 59	0. 664

n=18			
tle	dl	ans	time(s)
1000	9	125. 77	34. 95
	4	131. 34	0. 65
2000	9	128. 07	62. 54
	4	132	0. 955

n=19			
tle	dl	ans	time(s)
1000	9	158. 8	77. 45
	6	167. 72	9. 28
2000	9	161. 9	156. 73
	6	168. 3	15. 272

n=20			
tle	dl	ans	time(s)
1000	10	198. 6	250. 42
	7	210. 9	39. 06
2000	10	204. 6	462. 32
	7	213. 4	67. 69

Appendix IV Corresponding Solutions for Quantitative Analysis of $f(n)$

n **=1** **2**	0	1			
n **=2** **3**	00	01	10		
n **=3** **4**	000	001	010	100	
n **=4** **5**	0000	0001	0010	0100	1000
n **=5** **6**	00000 10000	00001	00010	00100	01000
n **=6** **8**	000000 000011	001100 010101	011010 100110	101001	110000
n **=7** **10**	0000000 0000011	0000101 0001001	0010010 0100100	0111000 1001000	1010100 1100010
n **=8** **13**	00000000 00000011 00001100	00010101 00100110 00111000	01001001 01010010 01100000	10001010 10010000	11000100 10100001
n **=9** **17**	100000011 011000110 000110011 000101001	101010010 000010101 010010000 000001100	100011000 001100000 011011000	010110100 101000100 011000001	110100000 000000010 010101010
n **=10** **22**	0001000111 1010010100 0100001110 1101000100 0010011010	1010000001 0010001101 0000100101 0100100010 0000001000	1000000010 0100000001 1001100000 0010100110	0001000100 1001001001 0010110000 0100010000	0101101000 1001010010 1000111000 0111000010
n **=11** **31**	11010001000 11001000001 00001001101 11000110000 01000010110 10010010001 01101000010	10101001000 01100000101 00000110101 00011001010 10110000100 10000101100	10001100010 00010101001 00100011100 00001010011 01100101000 00000100000	01001011000 10100010010 01110010000 00100001011 10000000111 01010000011	00000111010 00111100000 01001100100 10100100001 00010100110 00011010100
n **=12** **46**	000000111010 000101100100 000100101001 100000100110 100110000100 001110001000 010001000101 000001001110 110100001000 101000011000	000011001001 000101011000 000111000010 100100010010 010110000001 110000000011 000010011100 011000001010 111000000100	001000110100 100010001010 101001000010 100101000001 100000110001 010000011001 000001100011 000010100101 010100100010	010010000110 001011000100 001010100010 101100100000 011000100001 000000010000 101010000001 001001101000 100001010100	000110110000 011010010000 001001010001 100000001101 010000101100 011101000000 010001010010 000010010011 010100010100
n **=13** **54**	1000000100011 1000110000001 1000100101000 1110000000001 0100000101010 0011010100000 0001011000100 0000110001100 0010100000011 0000101001010 1000010011000	1010100000100 0001000001110 0001000111000 1001101000000 0101000100001 0100011000001 0011100001000 0110000011000 0100100001001 0110010000100 0010010001010	0011000000101 0000100010110 0001010000011 1100000001100 1000001010100 1001000100100 0000011101000 0000010010101 0001100100010 0000000000100 1001000001001	0000101000101 0100000110100 0001100010001 0010000110001 0101001001000 0000010110010 0000001011001 1100001000010 0101010010000 1100100010000 0101100000100	0010000100110 0100110000010 0010001010010 0010110010000 0100000000111 1001000010010 1100010100000 0000101110000 1000010000110 0110100100000

	10000000010101	10000000111000	00110001100000	00011000000110	00010000011010
	00100001010001	01000001100100	00011001001000	00100000110100	00100010011000
***n* =14**	00101000001010	00000010000000	10001000100010	00000011000011	01010100010000
	10000000001011	00001000010011	01000100100010	00000001001101	01000010001010
67	00001110000010	00000001010110	10010100100000	00011000110000	01100011000000
	11010010000000	01001100000001	00100100001001	00100110100000	10010001010000
	00001010010100	00000000101110	00000110000101	10100010000001	01111000000000
	01000000000111	00001100100100	10101000010000	01001000101000	01000010110000
	01000101001000	10110000001000	10000010100100	01010001000001	00101000100001
	00010000101001	00010100000011	00010011000100	11001000000100	00110000000101
	10000001100001	00000011101000	11100000100000	10001101000000	00010010100010
	00100101000010	00000111010000	10011000000001	10100000000110	10000100001100
	10000010010010	01001001000010	01000000011100	00101001000100	01100100000100
	00010110001000	00001100011000			
	100010011000000	000110000100100	010010010010000	000010010000011	000011000101000
	100000101000010	100100000110000	000001100000011	001010000010001	011000001001000
***n* =**	010010001000010	011000000000011	000010110001000	101000000001100	101000001000001
15	000100000000111	000011000000110	100100010000010	000001010110000	010010100000100
	100100001001000	100001100000100	100001000100001	100000100011000	011000110000000
83	010001000000101	000000011101000	110000000010100	010100000001001	100000000001011
	001000000011010	001100010000001	000000100110010	000000001110100	001010010100000
	000110001000001	001001101000000	000100000011100	100010000100010	000110100000010
	000101001000010	000010001011000	110001000000010	100001001010000	010001000011000
	001010001000100	010001001100000	110000010000001	000000110100001	001101000010000
	010000010100010	000100101100000	001000011000010	000101100001000	000100110010000
	000000001100011	000100000101010	000000000100000	000001001001001	010100000010010
	100000010100100	001001000100010	101000010010000	010100001000100	000000010010110
	011011000000000	001000000100101	010000101000001	010000000110001	110110000000000
	000101010000100	000000100001110	011100000100000	000000111000100	110000000101000
	000010000001101	000011100010000	000000010011001	101000100100000	001110000001000
	001100100000100	000001010001010	100010100000001		
	0010000000111000	0010100001000010	0101001000001000	0000001101010000	0100000010101000
	0000011000110000	0100001100000100	0010010100000010	0010000000100101	0100010000010010
	0001010010000010	0010001000001001	1001100001000000	0110001000010000	0001000100001010
***n* =16**	0000011001001000	0001011100000000	0000001100101000	1000000010001100	0000011000000110
	0000100101100000	0110010000000100	0000101100000001	0001100000101000	1000100010100000
100	1001010000000001	0011110000000000	1100101000000000	0100010001000001	1000000100110000
	0100110000100000	0000001000100011	0001000010000101	1000100000000110	0110000001100000
	1010010000010000	1000001000010100	0100010110000000	0100001010000001	0000110010010000
	1100010000001000	0001000001000011	0000000100000000	0010000001010001	0000000101000110
	1100000000100001	1000010000100010	1110000100000000	1010100000001000	0000000000001111
	0001100110000000	0100100100000010	1000000100000011	1000000001011000	1000001000001010
	0111000000000010	1001000000010010	0001000100100001	0100000100001001	1011000010000000
	0010100100010000	1100000001000010	1010001001000000	0000010010100001	0000100000011010
	0001010001100000	0010010010001000	0110100010000000	0000000011110000	0000010100010001
	0000000010010011	0100100001001000	0011001000000100	0101000101000000	0000100001000101
	1000001110000000	0010000001001100	0100000010010100	0001101000000010	0000100000110100
	0000110100001000	1001001000100000	0000000110100010	0001000001010100	0010000010000110
	1001000100000100	1000010011000000	0000000100011100	0000001001100100	0101100000000001
	0101000000110000	0000001011000010	0001010000011000	0010101000100000	0001001010010000
	0000101010001000	0001000011001000	0100000000100110	0010000110000001	1000100000010001
	01000011001000000	00101001010000000	00000001000011100	00100110000010000	00101000101000000
	00100100100000001	10010000010000001	00111000000100000	11000000010001000	10001000000100010
	00000001010100010	10110000000001000	00001000010010100	11000010000010000	00001001100010000
	00001010010000010	00000011000101000	00000001000100101	10100010000100000	00010101000000010
***n* =17**	00110000001000001	00000010100010010	00000010000010101	10000000100110000	00110000010000010
	01000000101010000	00000010010001001	00010110000000001	10000010001000100	00100001100001000
119	00001000001001100	00000000111000100	01000000000001110	00001000001110000	00010001001010000
	00000011010010000	11000000001000001	00010010101000000	00010010010100000	01110000000000100
	00001010100100000	00100010000001010	01000000000110100	00000100001001010	10000101000100000
	01010001000000001	00000001001000011	00101010000000100	00000100001100001	10100000010000100
	11010000000000010	00000000010010011	00011000000000101	00001000000000000	00001000100001010
	00000000010101100	00000100000100110	01100010010000000	01100001000000010	10100000001000010
	00010100000011000	00100001000110000	00100000010100001	00000110011000000	00000011100000100
	01000110000000010	10001011000000000	10000000000001101	10011000100000000	00100000100100010
	01000000000101001	00010000100010001	01001001000100000	01101100000000000	00100000000010110
	00010000100001100	10000100000000011	00101000000000011	00000101000010001	00000010001011000
	10000001100000001	10000000000011010	00001000011000001	01001010000000001	01000001010000100
	00000101101000000	01000000100000011	11000100000000100	01000100010000001	00001000000011001
	00000100010110000	01100000001100000	10000000011100000	00110011000000000	00001001000000110
	01001000000010010	00001110000001000	00001100110000000	10000110100000000	10010001000000100
	01010010000001000	00010000001101000	01000100100001000	01000000110100000	10100000000010001
	01010100001000000	00100101000000100	10000000101001000	00100100000101000	00010100100100000
	00000100001010100	00000010000100011	00010000000110010	01010000010010000	00000000110011000
	00010100010000100	00010000001000110	10001100000010000	00011000010001000	

	000101010001110101	000000110100111100	000000101011101000	111010000101001001	000011010000101000
	010000101100100110	000101110001011000	001001000100100110	100010111000100000	101010001000101100
	100001101100000101	011001001000111000	101000001011010100	010000010111001001	100000011110110000
n **=18**	000100011000010010	110001100001000000	000110011100000001	000111000110100010	000000000100011011
	000110000001101110	100001010001101100	011111010000000100	111100010000100010	011010100001000000
126	101110100011100000	010011001001001001	000001110010001001	100111000100000100	001010100100010101
	101000110001101000	100000110100010001	011010010000001110	100010110001000011	110000000001010101
	000011100000100101	101100001100110001	100110010001100001	011001011010010000	010010010011110000
	010101000101000110	100001100010100100	001001011100001100	000001101000101110	011110000000101001
	000100110011000010	000011000101110000	011101000001001000	011100101010110000	001000001001110101
	000001000001000111	011001000011100001	010010011001010100	000110000000110111	100010001111000010
	100111000000100011	100000000111001100	000101000110001101	000100001100101000	001000000110110010
	000000101010000011	000001011101010001	001100001101000001	000001111111000000	101100010100010100
	010010100100011000	001101100010100001	001000010010100100	001111110110000000	010110100000001010
	100110010000011000	010000010001111010	011000100011011010	010000011110000100	100000001010001001
	000001101000010000	101001010000000101	110001010100110001	110101000000101100	010000010010000011
	111001001000000011	000100001011001110	001111001000000110	100010111101001000	001100110100000011
	010110000110000000	100001010100000010	001011111000000000	110100001010001100	000010100101000110
	100010010010000110	000010000010011100	000010010000000101	010000010100101101	011010010100100000
	000000000100000000	110001010010001000	001100101000001010	001000010001010000	000011001001100010
	100010101010010010	010000001000001000	111100001011000000	000000100111100011	000100100110010100
	001100000000011101	101010100110001000	010010001100010001	000011010010010010	100100100000110001
	011000100010000111	010001110000000110	001010001010010001	000100010101001100	010100000000110000
	101001000110001010	100001001001011010	101000100000110110	010100101000000001	100000001000110111
	010000001010011110	001011000000011010	001000000011001011	000010100100101011	110000100100101000
	100101000010000000				

	0110001000000001100	0000100001000100011	0100000000011100010	1001101100001000000	0000101001010001000
n **=19**	0010100011010000001	0001000011100000100	0000110010001001010	0000010100000000011	0101001011001000000
	0101101000010010000	0001001010000001010	0010100000001100100	1000010011000000001	1000100111010000000
175	1000100011100000010	0000110100010000110	0100000111000100001	1111010000000000100	0001000101010000011
	1000000000110100001	0011110000001000000	0000100001100110000	0010001001001000101	1101100000000001001
	0000100101001001001	0011000001000110001	0101000010110100000	0000001000110100110	0000000011011100000
	0010001000111000100	0000001010001000011	1001100000010100010	1100001000100001010	1000100000001110000
	0100101100000000010	1001000100010011000	0000001001000011011	0100010000101001001	0010000000100100111
	1010010001000000010	0000001000101110001	0100000000010001101	0100000001111000100	0010001010100100010
	1000100010100000101	0010000010011011000	0100000000010110001	0101000000000100110	0011011001000000000
	1000000110001010100	1110000111000000000	0000101101100000100	0000100100100010010	0010000001000011100
	0100001100000100100	0001001000101000010	1010000001110001000	0011000011000010010	0100110010011000000
	0110000110100000001	0000000010100011101	1000010101010100000	1000000010010010110	0101011000000011000
	0100101010100000000	0000010000000111010	0011100101000000100	1000010000110010000	0000010001011000001
	1010001001100010000	0100000011010011000	0000000101001100101	0000001101110010000	1100010010000101000
	1100000100110000100	0001010000100010100	0001001010001110000	0110100010000001010	0010111000000100010
	0101010010000010001	0011100000000001110	0101000100001000001	0001100010001000001	0010100000111001000
	0001000000110001010	0000010001101010010	1000110000001000110	1100001000001000100	1000000001001101000
	0000001110000011000	0100000010001010010	0001010110001100000	0000100001011000010	1110000000010100100
	0000101100000010001	0001000100000101011	0000010010110001100	0001110001100000010	0100010100100100001
	1010111000000001000	1000010101000011000	1100001010010000000	1110001000000000011	0110011000010000010
	0110000000001010001	0110000001000100101	1001011100100000000	0100010001000001110	1100110000000010010
	1000000101100000110	1001101000000101000	0001100010010000100	0010010011001000100	0011000000110010001
	1000001110000100000	0000001011010000100	0000001101001000110	0010101000001010010	1000000000011011010
	0011000010011000010	0000000111100010001	0000011100010001001	0000010000000101101	0000111000010100001
	0000110000100100100	0000001100011110000	1100000100000110010	0001000001010100101	0111000001100001000
	0100011001110000000	0000101010000110100	1000010010010100010	0010011110010000000	0100100100001011000
	0100100010010000011	0000000110111000000	1101001001000100000	0000111001000010000	0000010100101000101
	0100000000100010110	0101101000000000101	0000000000000001000	1010011000100000001	0000001001100101000
	0000000010000101110	0001100110100100000	1010000000011000000	1001000000000010011	0000101000001101001
	1011000100000100000	1010100001010010000	0010010100000010101	1000011000000010101	0110100010000010100
	0010001001010100010	0110010100000001000	1110100000100000000	0100000101000000000	0001100011000101000
	0001001100001001100	0011000000101000101	0000010010101100010	0001011000011000000	0010000100001001010
	1001000001000001100	0100011010000000110	1001000010100010000	1101000000001010000	1100110000001000001
	0000011100100001010	1000000000001011101	1011010000100001000	0100110000001101000	1000001100010000101

	01000100000010110001	01001000011111000000	01000100100011001100	10010001000000011001	01000000110100010101
	10100110100000000011	01101000001000110000	11110010010000000010	10000100000100111001	00110000000000101100
***n* =20**	00010100111000100100	11100000110000000100	11010100010110000000	10010000010000011110	00110000000010000110
	00011010110000000010	00001001000011011000	01011000101100001000	00101001011001000000	01001010000001101000
	00100010000110001101	00001100100101001001	01100011000000001100	00001010100010001000	10001000001000000100
	00100001100100000101	01000010101000100110	10001001000010001110	00011100101000010001	00101100100001100000
220	00000011000110000110	00010000011100001101	00100100001010101000	01000010010100110000	10110010000000110100
	11100101001000000000	10010111100000000000	01100000010001101010	00011001100000101001	00001100000011000010
	01010000110101100000	00001000010110011000	10010010000010101000	00000000110110100011	01001001100101010000
	00000011000001010101	00101000010010000101	01101000001010001100	00001001110011000010	00100000010001011000
	00000110011000010100	10000010110100000000	10010100101010000010	00000001001100110101	00100101000011001001
	10000001101101000001	10101000010001100100	10100100000000110010	10100001011100001000	10001011100000000101
	00010000100101010110	01101110000000000010	00011011000100001010	00000001000100101010	11000000100001100010
	10000010101011010000	10110100110100000000	00100010000100010001	01001010111000000100	00011010010010000100
	00000011100010110010	00010110000101000011	01111001000000000001	10010101000100000100	00110000111000001010
	01000000111010010010	00000111000101100001	01011101100001000000	01010010001100101000	10000010000101001100
	00100001010000010110	01100001000100111000	01000100000000101110	00100111000000011000	10110001001000000010
	10000110010011000000	11001100010100000010	01001110000110010000	00010001010100000001	00000010011101010010
	00000001111000010001	01010001100000110000	00000000000000001101	10110000000001001011	00000010000000111010
	00110100001001000001	00001001010100110100	00001101100010000100	10001101011000000000	00010000011011010001
	00110100010000000101	10001111000010100000	01000001011010001000	00000000001011100110	00011100000000111100
	00101010010101000000	10010000011000110000	00100100101000010100	10000001010001100010	00001000000110101001
	01100010000010100100	11000101000001000110	01010010100000011000	00001100000100100100	01101000100000011000
	00110001000101000000	10100010011000000001	00100000001111010100	00100110000100010110	00001010001010100101
	11101001000010010000	11000100101000100000	00010000100010010001	10100001000010100101	10001011010000001000
	10011101000000010010	11100000001000101001	00010000000101101110	11000001000100001001	01010101010000100000
	01000000001001000101	00100000001010110011	01110100000001010000	00010100001000100010	11100011100100000000
	00001101001001101000	10011110001000000000	01000110100000000101	10000001001010010001	01001000100100000111
	00010110000000110011	10001001001100010010	11000011000011000100	11001000011010100000	10100000001001110000
	01011010010000001001	00110001010010100000	00010011101001000000	00000110010100101010	01010000000000010111
	11000010001010000010	11000000000000010000	00011100010001001000	01110000000011001000	01001101001000001001
	10000000010110010011	00010001000011100001	00110000100110100000	00010010110001100100	01001010001100000010
	01000001000011010010	00011000001110110000	00000000110011110000	10000110000001001001	00101010000010011010
	10000000101000011011	00001100000011100101	00000100110010001000	10100010101000001100	11001000010100000101
	00000011111000001010	00000000000001000000	00100010001011000011	01100000101100000010	00101100010000001110
	10100001000001011100	10010010000010000011	00101011000000100011	10010011000100100000	00001000100000110110
	10110000100011000000	10000000010010110100	00101011001000010100	00000100100001011010	01000001010000011101
	01000000011000100011	00100010100001000110	10000000100110011000	00010000010011000111	01100001100001010001
	00010101100100000010	11010000000101000001	01000100001111010000	10101000100010000010	11111100000000100000
	10100000010110101000	10001000000101010101	00100101000000100100	00010011001000001011	00100000110000101001
	01011001001000000100	10001000100100100000	01010000100010100100	00101000000001010011	10010000100001101001
	00000100011110000110	00100101011000000011	01110000100000001101	00010100010010010010	01000010010010010101
	00000101010110000000	01100000000110000011	00000010011001101001	00010010001001011000	10010100000001110100
	01100110010100000001	11000100011000001100	00001000001010001011	00100100000111100100	00011010100101000001