2019 电子科技大学美国数学建模竞赛模拟赛

承　诺　书

我们仔细阅读了数学建模竞赛的竞赛规则.

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的, 如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们的题目编号是（填写：A 或者 B 或者 C）： C

我们报名队伍号是： H330

参赛队员姓名学号(中文填写打印并签名):1. 朱晓萌 2017070902020

2. 陈逸文 2017000201006

3. 赵治宇 2017070903023

指导教师或指导教师组负责人 (有的话填写)：

是否愿意参加 2019 年美国赛（是，否）： 是

- 2017 级及 2016 级无国赛经历同学请先填写报名链接：https://www.wjx.cn/m/29408126.aspx
- 有国赛经历的队伍可以直接联系数模导师后登录我校数学建模网站进行报名

日期： 2018 年 11 月 16 日

报名队号（请查阅《电子科技大学 2019 年美国赛模拟赛官方报名信息表+队号（按照队长姓名首字母 A-Z 排列）-最终版》后填写）

A Model for 3x3 Matrix Game Solving Inspired by a Chessboard Game

Abstract

The main task of problem C is to work out an efficient way to solve a certain block-filling game. Then, evaluate the challenge a person may meet during this process. And with its inspiration design a new game.

Our group consider it most important to first transfer the pictorial game into mathematically represented model, and with this representation, develop an algorithm to work out it's answers. Then, by analyzing the challenge and complexity of the thinking process, we can determine how difficult it is to solve the game. By transferring the solving step into coding language, we will be able to work out the possible solutions to any game of this design.

First, we represented the map with a coordinated system, translated every requirement into relationships of coordinates. This process allows us to use mathematical tools to analyze the game and computer language to describe what to do.

Then, we summarized how a player may process all the information provided.

Based on this process, a detailed algorithm applicable to every round is worked out to find the solutions. With the help of computer programming, it's convenient to find all the answers in a short period of time.

Minor adjustments of the algorithm are made for the program to operate more smoothly and swiftly.

Next, we collected some typical parameters the algorithm produced while running, including the number of different attempts it conducts. By summarizing the process of which a player may solve the game, we can determine how these parameters are related to human brain process. Thus, use them to construct the model to evaluate the difficulty level of the game.

Finally, with the model of the existing game, we developed a new game that can be analyzed through a similar model.

This developed model of ours can be applied to solve many other block-filling or grid-filling games with a little adjustment made.

Keys: Traversal, Grid Game, coordinates, BFS algorithm, DFS algorithm

Index

1 Introduction

The problem introduced to us a chessboard game with the following rules:

The game features a 3-D map which is divided into 9 grids, the height of which is varied from one to three. The grids on each row and each column corresponds to a fruit - say, apples, mangos and oranges for the rows and pineapples bananas and strawberries for the columns

The player is given nine animals for each round (chosen from provided blocks: one elephant, one monkey, one tiger, one pig, one cat, one turtle, two hippos, three lions and five pandas), along with a set of conditions that describe their positions. In order to pass the round, player is required to place them into the nine grids in a way that satisfies the given condition.

the game is divided into different levels according to their difficulty level. four examples are presented to show the conditions and the corresponding solutions with increasing difficulty.

Under such conditions, our task is:

1) Develop a mathematical model to measure the difficulty of a game like the provided one.
2) Develop a mathematical model or an algorithm to determine if there exist solution under the given restrictive conditions, and if the solution is unique.
3) Design a similar game based on the model we developed.
4) Provide a memo promotion for our designed game summarizing the result.

2 Overall Analysis

In order to s find a way to evaluate the difficulty level of a certain round and to find out the solution (or solutions if there are more than one), it’s first needed to understand the thinking process when one is trying to succeed at the game. Then, by analyzing the challenge and complexity of the thinking process, we can determine how difficult it is to solve the game. By transferring the solving step into coding language, we will be able to work out the possible solutions to any game of this design.

Our work can be divided into 4 parts as listed below:

1) Build the model for game:

We represented the map with a matrix, and rewrote all the conditions in mathematical language. Then, we analyzed a general thinking procedure to solve the game.

2) Build the solution finding algorithm:

Specify the game solving process into a step by step algorithm. Then, with programming on computer, it's easy to work out all possible solutions of a round in a rather short period.

3) Determine parameters to evaluate the difficulty.

4) Introducing a similar game:

The model is theoretically applicable to a great number of other games involving placing subjects into grids according to given conditions. We made minor adjustments and developed a new game. The new game requires more complex thinking skills for a player to pass.

3 General Assumptions

1) The difficulty level in this game is fairly-balanced, the difficulty increases steadily as the level goes up.

2) The player is an average child above the age of 6, who is able to understand and process logically information provided in the game.

3)The more steps a player goes through, the harder it is for he/she to remember previous steps and conduct logical thinking.

4 Model Inputs and symbols

symbols	description
h（xA，yA）	The height of block A is positioned
D	difficulty level
X1	Thinking Steps of relative conditions
X2	Space complexity
X3	Number of nodes
X4	Number of Dead ends
N	Number of solutions
n[condition m]	number of conditions of type m

We manually modified when running the animal arrangement, in order to run the model closer to the human brain.

5 Models and Answers

1.1 Preparation: Game solving model

1.1.1 Mathematical Representation of Conditions

The map of nine grids should be easy to represent with a 3x3 matrix. The position of each animal can be represented by their coordinates. So, all the conditions the animal's is required to satisfy can be transferred into ranges that the coordinate must fall in.

I. Representation of coordinates

In our coordinate system, each grid on the map is equivalent to one unit of coordinate. The right lower corner of the map is regulated to be point O, the coordinate increases from right to left and from the bottom to the top.

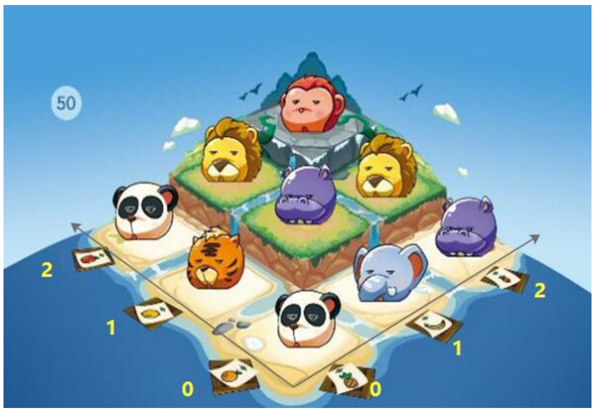


Figure 1 Established coordinate

This is the coordinate system we built from the map.

For instance: is animal A is placed on the third column counting from the right, and the third role counting from the bottom, its coordinate would be (2,2).

II. Representation of height

The height of each grid is shown right:

By observing the map, we notice that the height h of the grid is equivalent to the smaller value of its x and y coordinates.

i.e.: h (x, y) =min {x, y} +1

1	2	3
1	2	2
1	1	1

Figure 2 The presentation of height

III. Representation of conditions

There are nine types of animals in total, we label them as animal type 1,2,3…9

Conditions occurring in a round may include the following four types:

① Animal(s) of type A must stay on grid of a certain height n.

② Animal(s) of type B must stay or not stay on a role or column which is or not represented by a certain fruit.

③Animal(s) of type C and D must neighbor one another.

④Animal(s) of type E must be put on a grid whose height is higher/lower than or be the same as the height of grid that animal(s) of type F is standing on.

These conditions can be represented by the following formula correspondingly:

①$h(x_A, y_A) = \min\{x_A, y_A\} + 1 = n$

②If B must stays on a role represented by orange/lemon/apple: $y = 0/1/2$

If B must not stay on a role represented by orange/lemon/apple: $y \neq 0/1/2$

If B must stay on a column represented by pineapple/banana/strawberry: $x = 0/1/2$

If B must not stay on a column represented by pineapple/banana/strawberry : $x \neq 0/1/2$

③$\left|(x_c + y_c)\text{-}(x_D + y_D)\right| = 1$

④If E and F are on the same level:

$h(x_E, y_E) = h(x_F, y_F)$

If the position of E is on a level higher than that of

F: $\min\{h(x_E, y_E)\} > \max\{h(x_F, y_F)\}$

（and vice versa）

1.1.2 Categorization of conditions:

We categorize the given conditions into two classes:

Absolute conditions: condition type ① and ②

conditions in this class can directly decide the range that the subject’s coordinate falls in.

Relative conditions: condition type ③ and ④

conditions in this class cannot be directly decide the range of the subject’s coordinate, they cannot be used alone. Instead, they describe the relationship between two subjects’ coordinates.

**If there is a relative condition involving two subjects, say A and B, we define A and B to be “related”.*

Apparently, absolute conditions are stronger than relative conditions: they are more precise when describing the coordinates, and are more straight forward to the player. Thus, it's reasonable to assume a player usually start by looking at the absolute conditions.

Among the absolute conditions, it's easier to start with the strongest, which outlines the smallest area a certain subject may fall.

1.1.3 Game solving procedures

We believe that the most efficient procedure for a player to work out a game is:

1st, decide which of the absolute conditions regulates out the smallest area that coordinate of one subject falls. (label this subject as A)

2nd, search if there is relative condition describing the position relationship of A and another subject (labeled as B). If so, use these two conditions to determine position of A and B.

3th, if coordinates of A and B are still not fixed, the player will attempt to place them where the two conditions are met.

4th, he/she would find the absolute conditions that regulate the second smallest area a subject may fall (label this subject as C), and its related subject D, with conditions involving C and D he/she would repeat she second step.

If there's no way C and D can be placed to satisfy the conditions given, it must be that A and B are placed in the incorrect position. So, the player would go back to the 3rd step and move A and B to another position, and again, use C and D to exam.

This process would go on until the right position of A and B is found.

**We define the repeating process of step 3 and 4 to be "backtracking".*

1.2 An algorithm to work out solutions

1.2.1 Algorithm Design

Based on the game-solving procedures in 1.1.3, we developed the following algorithm:

Step 1: Search for all the absolute conditions. If there is any, exhaust all possible coordinate a subject may have for each one of the absolute conditions. If there is noun, jump to step 4.

Step 2: Within all subjects with absolute conditions, find the one with smallest regulated range.

Step3: Start traversal from this subject to determine whether a relative condition is attached to it. If so, go to step 3, if not, repeat this step.

Step 4: Conduct traversal for all possible position for this subject (labeled A).

Step 4-1: Determine whether a condition of type③ is attached to the subject. If so, continue; if not, jump to step 4-4

Step 4-2: For each loop, import one possible position of A into the map, inflate its coordinate by one unit in each direction, i.e. $(x+1,y),(x-1,y),(x,y+1),(x,y-1)$. If an inflated point is outside of our grid (i.e. $x \in (1,3), y \in (1,3)$), give up the point, thus working out the inflated area. This area is possible positions that A's related subject, B, may take.

Step 4-3: Conduct traversal for all possible position for subject B. For each loop, place B in one possible position, and check if the relative condition involving A and B is satisfied. Record combinations that meet the condition. If all positions of B dissatisfy the condition, end this loop and import A into another possible position.

Step 4-4: Check if conditions of type④ is attached to A or B, if so, continue to step 4-5; if not, jump back to step 2, conduct step3~4 with subject of second, third forth (…) regulated area.

Step 4-5: Label the other subject involved in condition of type④ as C. For narrative convenience, we assume the condition involves A and C. If subject A is on a grid of bigger height, then it can be assured that: $\min[x_A, y_A] \geq 1, \max[x_C, y_C] \leq 2$, and vice versa. If A and B are on the same level, $h(x_A, y_A) = h(x_C, y_C) < 3$.(These constraints help to narrow down the range of A and B's coordinates.)

Step 4-6: Split all possible positions of A and B into branches, for each branch, conduct traversal for all possible coordinates of C, if the condition is satisfied, take record of all their positions and continue; if not, go on without recording, repeat until all possible arrangements of A, B and C are exhausted.

Step 5: Split all possible position arrangements of A, B and C into branches, for each branch, label coordinates the three took as "not usable".

Step 6: With the remaining usable positions, repeat step3~5 with subject of second, third forth (…) regulated area.

In the diagram below, each dot represents a half-filled map, and arrows represent steps that attempts to fill subject into the map. Whenever there is more than one way to fill in a subject, branches representing possible arrangements split out.

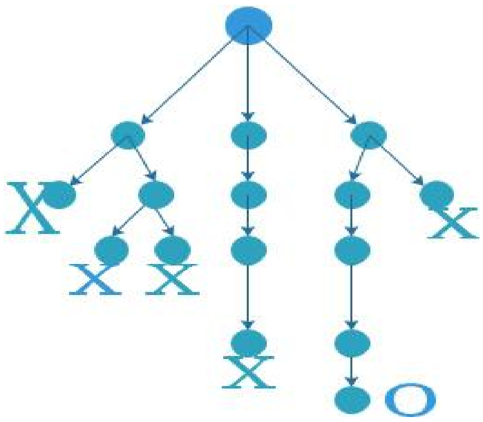


Figure 3 BFS network diagram

After each branch is split out, they proceed independently, each branch stops when there is no empty position left on the 3x3 grids (represented by O in the diagram), or when there are no way remaining subjects can fit into the grids in a way that satisfies the given condition (represented by X in the diagram).

1.2.2Adjust for Coding

I. Information Storing

In the program, each type of animal is stored as an array, the array is made up to 8 dimensions, representing the animal type, total number of animals of this type, related condition ①，related condition②，related subject in condition②，related condition③，related condition④，related subject in condition④.

The map is stored as an 3x3 array, in which the numbers stands for animal types.

II. Working out possible ways of placing a type of subject: DFS (depth-first-search)

To determine all possible position a subject may take, for narrative convenience, we will refer to this subject as A, and assume there is in total n elements in A to be put in to the map. (namely there are n animals belonging to type A)

First, all possible coordinates one A can be put into is made into a coordinate list:

$$\text{A-xy-list}=[(\mathrm{x}_1,y_1),(x_2,y_2),...,(x_n,y_n)]$$

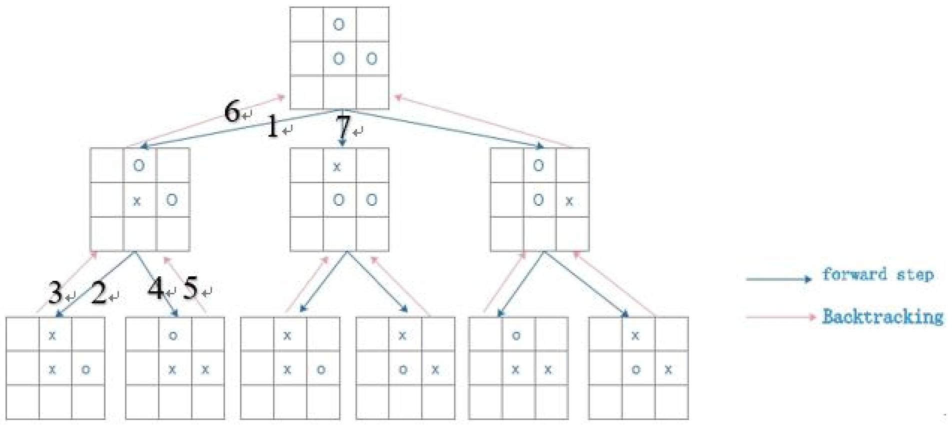


Figure 4 DFS network diagram

As shown in the diagram, we first put one element into one possible position on the map, save this map as a node, then place the second. A node is saved whenever a choice is made in the program. And the path can be viewed as a branch.

A branch is finished when all element of A are placed on the map, then the algorithm will go back to the nearest node and start to work out other possible arrangements.

However, because all elements of A are viewed as identical, after acquiring all possibilities, we must remove overlapping outcomes before saving them in an array list.

III. Exhausting all possible solutions: BFS (Breadth-First-Search)

To decide if the solution to a round is unique, and work out all possible solutions there is, our algorithm must exhaust every probability whenever it needs to make a choice.

However, applying DFS to for the whole program would take up too much storing space, we introduce BFS as a solution.

As shown in the diagram on the left: every time a choice needs to be made, we apply DFS to work out all possibilities, then the map recording all possible ways to position the subject is stored in an array-t. We define that each possible arrangement is a branch.

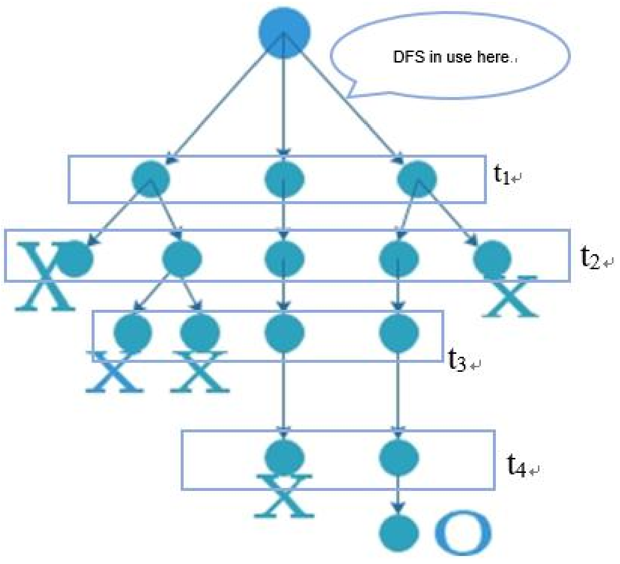


Figure 5 The relationship between BFS and DFS

Then, we start looking at the next subject, under each branch we can work out the possible ways to place the second subject. We then dump the content of array-t, and refill it with possible arrangements of the two subjects. If the branch is proved not proceedable, all information from this branch will be deleted.

When deciding where to put the third subject, the algorithm import information from array-t, and does the same operation again.

This method helps to save a great deal of storing space, thus making programming possible on average laptops.

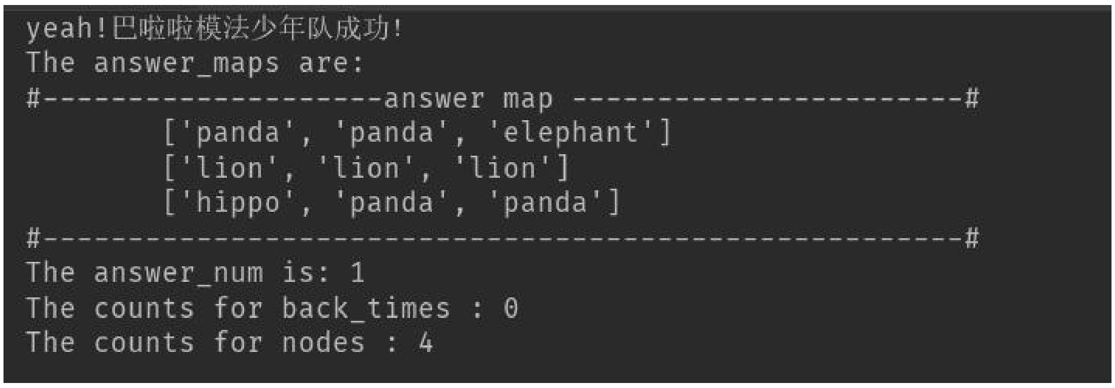
1.2.3Model verification

If our algorithm is correct, after we input the conditions of a round, the answer it gives us should include the answer provided.

So, we tested the algorithm with the given 4 rounds, following are the outputs:

Level 10:

Our output:



```
yeah!巴啦啦模法少年队成功!
The answer_maps are:
#--------------------answer map -----------------------#
        ['panda', 'panda', 'elephant']
        ['lion', 'lion', 'lion']
        ['hippo', 'panda', 'panda']
#------------------------------------------------------#
The answer_num is: 1
The counts for back_times : 0
The counts for nodes : 4
```

Figure 6 The result of level 10

Given solution:

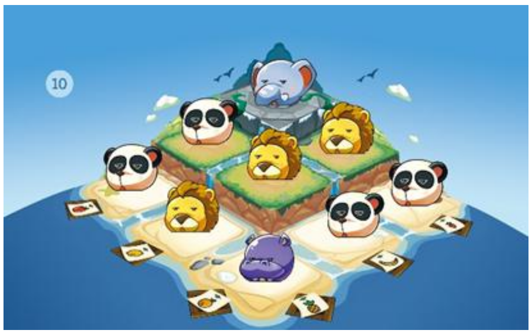
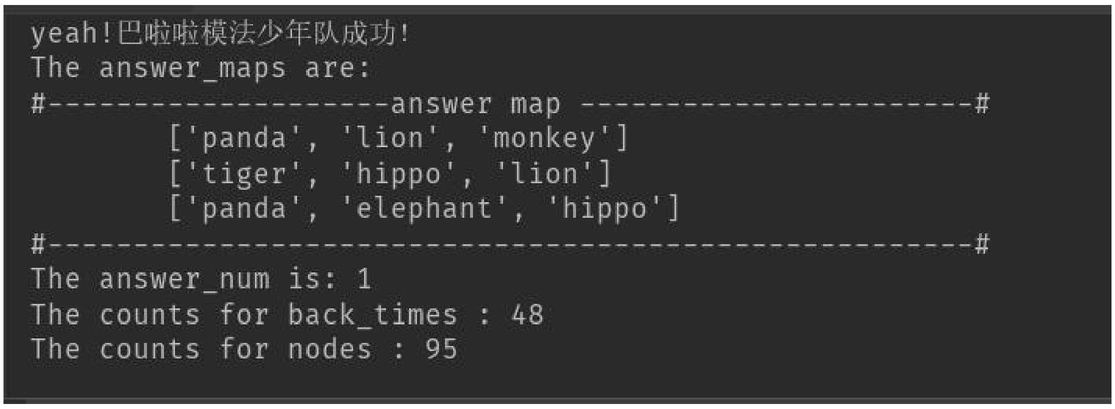


Figure 7 The result map of level 10

Level 50:

Our output:



```
yeah!巴啦啦模法少年队成功!
The answer_maps are:
#--------------------answer map ----------------------#
        ['panda', 'lion', 'monkey']
        ['tiger', 'hippo', 'lion']
        ['panda', 'elephant', 'hippo']
#-----------------------------------------------------#
The answer_num is: 1
The counts for back_times : 48
The counts for nodes : 95
```

Figure 8 The result of level 50

Given solution

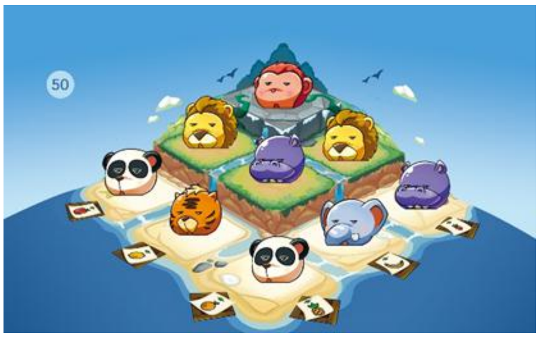


Figure 9 The result map of level 50

We examined the algorithm with every given example, all outputs proved to be identical with provided solutions.

And every given example has only one solution.

1.3 Difficulty Evaluation Model

1.3.1 Parameters that decide the difficulty level

The game aims at training the sense of permutation and combination, spatial position and neighboring relations, logic reasoning in children.

We have already analyzed a player’s thinking process in 1.1.3

1)For each condition, the more information a player needs to process, the harder this condition will seem difficult to him/her.

2)For overall solving process, the more choices a player has to make when playing, the less likely he/she will be able to work out the solution.

3)The more steps a player needs to think in order to find the correct answer, and more steps it takes for a player to know if he/she has done a correct move, the harder it is for a player to solve the game.

4)The more likely a player’s attempt will fail, the harder the problem

5)Also, the more solution a round has, the easier it is for a player to find out one.

Since the algorithm is 1.2 is based on human like thinking skills, we use certain parameters from the output to determine the difficulties.

Thus, we introduce the following parameters to measure the difficulty of a round:

X1： Thinking steps of relative conditions

This parameter is the total number choices the algorithm made when deciding where to place the subjects in order to satisfy the relative conditions.

When the program operates, it increases X1 by 1 unit value whenever it has to process a condition of type ③and④.

X1 can represent the maximum number of choice that a player may make in a round.

X2: Space Complexity

A player would usually start solving by attempting to place a subject with absolute condition.

Different types of conditions are of different challenge for a player to process.

X2=n[condition ①]+2n[condition ②-1]+3n[condition ②-2]

Note: condition ②-1 are condition② involving only 1 type of fruit；condition ②-2 are condition② involving 2 types of fruit.

This parameter is the number of possible positions to choose from when placing the first subject.

It indicates the difficulty the player is met when looking at given conditions.

X3: nodes

The algorithm's working process is shown left, a node is a dot in this diagram.

It is the point where a subject is placed into the map. If there are multiple solutions in a step, more nodes can be seen.

In this case, the player needs to do more steps of logical thinking in order to decide which position if correct to put in the subject.

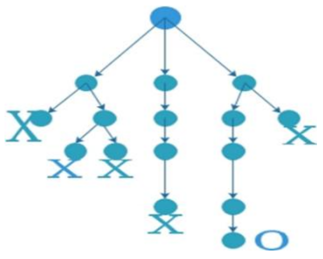


Figure 10 BFS mind map

Nodes can represent the amount of information a player processes when making a decision.

X4: dead ends

When a branch ends with X, we call it an "dead-end". It means that no solution can be found if the player follows this path.

The number of dead ends indicate the likeliness a player's attempt may end in failure.

N: number of solutions

The more solutions there is, the more likely the player can work out the problem.

It’s reasonable to assume that all the above parameters have positive correlation with difficulty level of the game.

1.3.2Working out a numerical relationship

Parameters X1 and X2 are generated information a player can directly process. Parameters X3 and X4 are challenges caused by complexity of game solving process.

When the difficulty level is low, there is not as much choices and attempts to make, wo the main challenge should be merged from the first two parameters; when the difficulty lever is high, there are more attempts to make, and the player will face higher chance of failure, thus the difficulty may depend more on X3 and X4.

Since all rounds have only one solution, parameter N does not really make any contribution. However, is there is more solutions, the difficulty level should decrease.

We arranged 4 parameters into 2 groups of two, to separated their ability to increase difficulty in different stages of the game, we give the two groups different powers.

The difficulty level should have the following relation with the parameters:

$$D = \frac{a_1(2x_1 + x_2)^{b_1} + a_2\left(x_3 + x_4\right)^{b_2}}{N} + a_3$$

We collected parameters from the given examples, for more reliable outcomes, we also collected some other game rounds from an E-seller of the game.

	$X_{1:}$ Thinking Steps of relative conditions	X_2: Space complexity	X_3 nodes	X_4 Dead ends	N: Number of solutions
Level 2	0	3	3	0	1
Level 3	2	2	3	0	1
Level 10	0	6	4	0	1
Level 11	2	4	4	0	1
Level 18	3	3	14	4	1
Level 23	10	3	15	4	1
Level 33	2	9	5	0	1

Figure 11 2~33 level data tables

After fitting this set of data in to the model, with the help of MATLAB, we worked out the specific value of the coefficients.

However, outcome of Level33 deviated greatly from others, its difficulty level is extremely high compared with others’. Then we fit Level 50’s data in to do more test, and found that D of Level50 turned out to be fairly reasonable.

So, it is reasonable to assume that Level33 may be a mistake the designer made, the conditions may be too difficult for Level33.

To avoid it disturbing our model, we outlined Level33 from our training data.

The value of coefficients we eventually determined is:

a3=-8.3274

a1=3.8927

a2=2.5347

b1= 6/12

b2=7/12

1.3.3Model and Verification

We now know: $D = \frac{a_1(2x_1 + x_2)^{b_1} + a_2(x_3 + x_4)^{b_2}}{N} + a_3$

a3=-8.3274

a1=3.8927

a2=2.5347

b1= 6/12

b2=7/12

To verify this model, we apply it to game round level1 and level50

The result is as follows:

	X_1: Thinking Steps of relative conditions	X_2: Space complexity	X_3 nodes	X_4 Dead ends	N: Number of solutions	D: Difficulty Level
Level 1	0	2	3	0	1	1.9888
Level 2	0	3	3	0	1	3.2261
Level 3	2	2	3	0	1	6.0189
Level 10	0	6	4	0	1	6.8979
Level 11	2	4	4	0	1	8.3730
Level 18	3	3	14	4	1	17.0333
Level 23	10	3	15	4	1	24.4624
Level 33	2	9	5	0	1	/
Level 50	34	5	99	30	1	68.0941

Figure 12 1~50 level fitting data tables

It can be seen that Level1 proved to the easiest and Level50 the hardest. The value of D increases with the level.

However, D did not grow linearly.

Our model may need to be improved for D to directly represent the Level number.

It's also possible that the game was not designed to increase its difficulty equally in each level.

1.4 A Game Designed Based on a Similar Algorithm

The main task of the original game is to arrange the positions of given animals so that their location is consistent with given description. Since the game map can be easily represented by coordinate system, it's convenient to do mathematical analysis with it by turning every condition into ranges in a coordinate system.

However, we aim to design a new game that provide the player with bigger challenges and entertainment. So, this new game we offer features more complex information and mutable rules.

1.4.1 A new game: Beautiful Garden

We are now introducing a new game - "Beautiful Garden".

This game is targeted for children above 9, expecting to enhance their ability in sense of permutation and combination, space imagination, memorization and logical planning. The rules are as follows:

We provide a square garden map made up of n rolls and n columns, and a picture or description of the "final design", instructing how the flowers should be located.

The player's goal is to plant flowers on the map so that they eventually grow to the shape the game asks for. They are only given a limited number of flowers, which would not fill up the garden map. However, the flowers grow after a period of time, namely one round. The player must plant flowers in a way that they eventually grow to form the targeted design.

The flowers grow under several constraints:

1) Each grid features the characteristic of a certain degree of humidity (represented by its color).

2) Each type of flower has a humidity environment that they are best cultivated. For example, roses grow best on a land with humidity of 50%~60%, while lilies grow best on a land with humidity of 60%~70%.

3) After each round, the planted flower would spread to the neighboring blocks if the humidity level is suitable. If a flower is planted in a block whose humidity level

does not encourage its growth, it will still stay alive, but will not spread to other positions.

4) As the difficulty increases as round level goes up: the garden will be made up with more grids to plant, and will feature more humidity levels, the different humidity area will be more irregular in shape.

The flowers below and their preferred humidity condition are listed below:

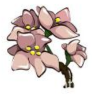
Flower type	Icon	Preferred humidity	Representing color
orchis		100%	🟪
cuckoo		70% ~ 80%	🟦
Morning Glory		70% ~ 80%	🟦
Lilac		60% ~ 70%	🟦
Tulip		60% ~ 70%	🟦
Daisy		50% ~ 60%	🟦
rose		50% ~ 60%	🟦
Sunflower		30% ~ 40%	🟧

Figure 13 flowers below and their preferred humidity condition

1.4.2 Gaming examples:

To help customers better understand the game, we now provide several examples.

Round Level 1:

You are given  X1, X1

The garden map is shown right: 1.4.1

Plant the flowers so they end up like 1.4.2

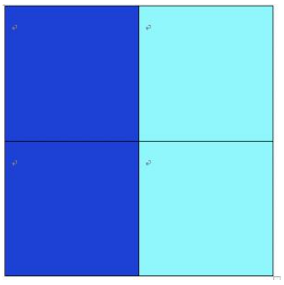


Figure 14 1.4.1

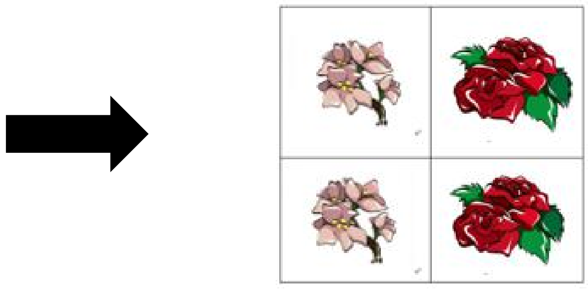


Figure 15 1.4.2

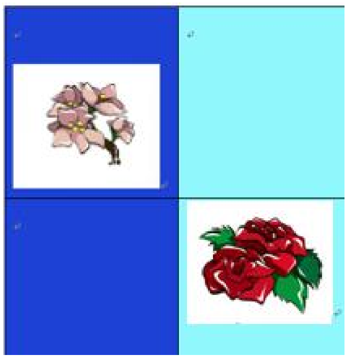


Figure 16 1 level answer

Thus, flowers should be planted like above, after one round, they will reach the target.

Round Level 5:

You are given: X1, X1， X1， X1，

X1

Plant the flowers on the garden map so that:

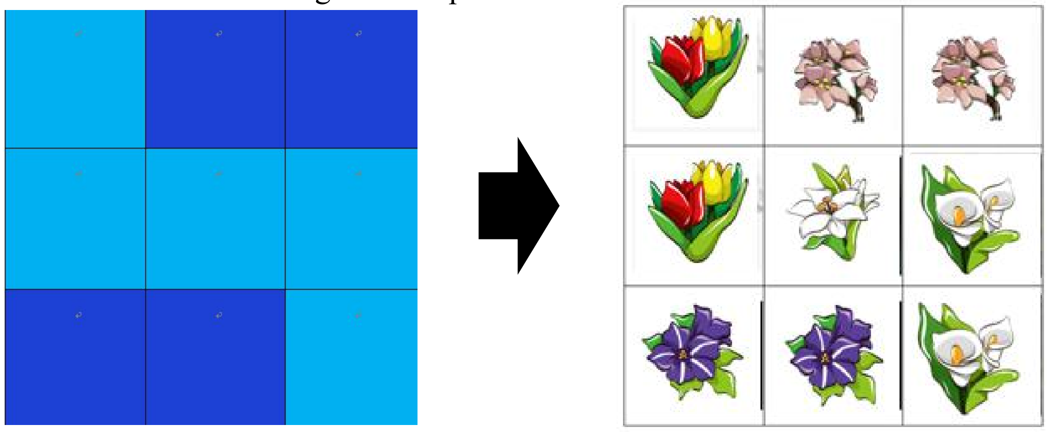


Figure 17 the requirements of level 5

So, the flowers should be planted as below:

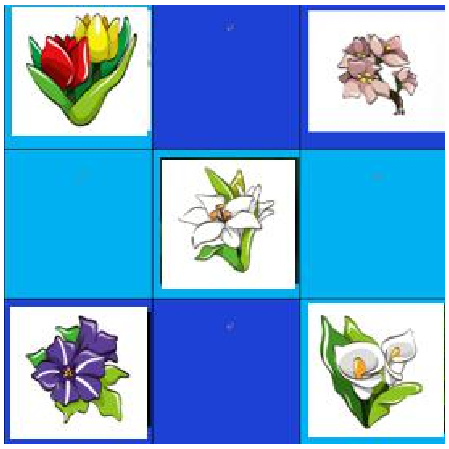
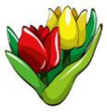
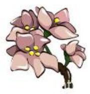
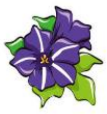
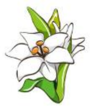


Figure 18 The answer of level 5

Round Level 25

You are given:  X1,  X1, X1  X1,

 X2

The map is:

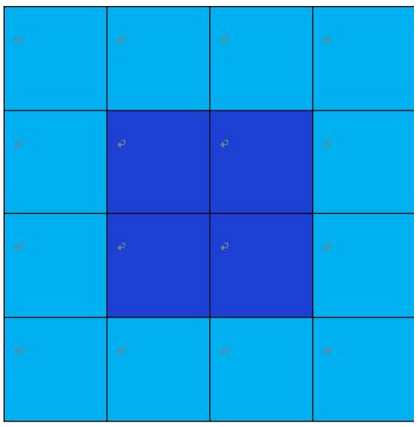
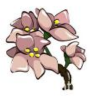
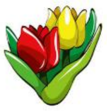


Figure 19 the requirements of level 25

Plant the garden so that:

 and  does not neighbor each other,

and does not neighbor each other

The solution should be:

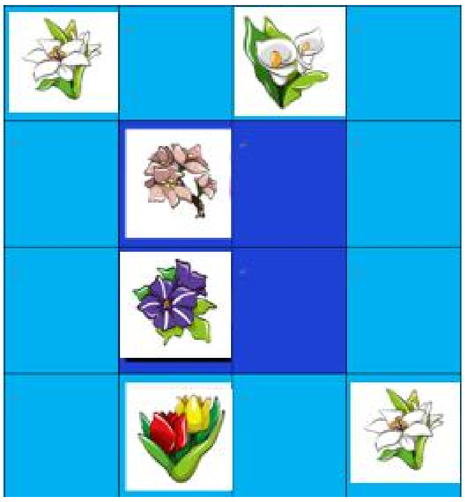


Figure 20 The answer of level 25

1.4.3 The algorithm behind

Similar to the model of the original game, the garden map can be represented by a coordinate system, in which every flower is represented with their coordinate.

When a flower (namely A) grows, a block $P(x_p, y_p)$ that satisfy that:

a. $\left|(x_p + y_p) - (x_A + y_A)\right| = 1$

and b. $humidity(P) = \Pr eferredHumidity(A)$ will be taken.

However, this game is more difficult than the original game. For reasons that:

1) As the level increases, this game will feature a wider map, making it harder to do overall planning, and force more logical thinking and decision making to be made. So, X2 of this game is higher than that of the original.

2)The rule of this game is less straight-forward, the player must do more thinking steps in order to analyze the conditions.

3)It features more space complexity. It's more difficult to choose with what subject to start, and the starting subject, there is a bigger regulated area. So, parameter X1 is bigger.

4)Since the flowers grow over each period, it's harder to pre-plan, the player has a bigger chance to fail. So, X3 and X4 are larger.

6 Space for further improvement

Although our difficulty evaluation model is able to determine the degree of difficulty with it's value D. It's insignificant in terms of uniformity.

1)We cannot be sure if the degree of difficulty increases equally if two groups of game rounds have equal difference in D values.

2)Our D value is not directly equal to game level.

This is mainly due to the objectiveness of our concept of "difficult"

To improve this model, we need to settle down a fixed rule to measure the difficulty by quantity. Experiments or surveys with more players is required.

We can then have trustworthy data to refer to when determining the difficulty levels.

7 Strength and Weaknesses

Strength:

We have successfully conducted an efficient algorithm to solve the game problem. Instead of using DFS, which keeps track of every step the algorithm has done, we applied DFS to the program. This design makes it possible for our program to take up only a small amount of storage room, making it feasible on average laptop computer.

Our difficulty evaluation model is reliable and based firmly on human thinking process. It's structure also shows that the as the uncertainty of conditions increases, its impact of difficulty level grows rapidly instead of being linear.

Weaknesses

The game solving program does not have a convenient API, all information need to be put into the program by the form of adding codes.

Our model of difficulty evaluation is only proved to be reliable from level1 to level 50. We don't know for sure if it applies for level above50, for no example is given.

When the program runs, the arrangement of the incoming animals is optimized only once, which is a better optimization for the algorithm, but there is still a certain gap for the arrangement order of the human brain. If you want to approach the human brain, you can make appropriate manual intervention for the arrangement of programs.

8 Reference

[1] Advertisement of the given game on Taobao:
https://detail.tmall.com/item.htm?spm=a230r.1.14.1.6dad599b4S7goe&id=579830131634&cm_id=140105335569ed55e27b&abbucket=4&skuId=3854830745054

[2] Difficulty evaluation for sudoku:
https://wenku.baidu.com/view/af550ed51a37f111f1855ba0.html

9 Attached file 1: Non-technical Letter

Advertisement for: Beautiful Garden:

Beautiful garden

About Our Product

Suitable player age: above 9

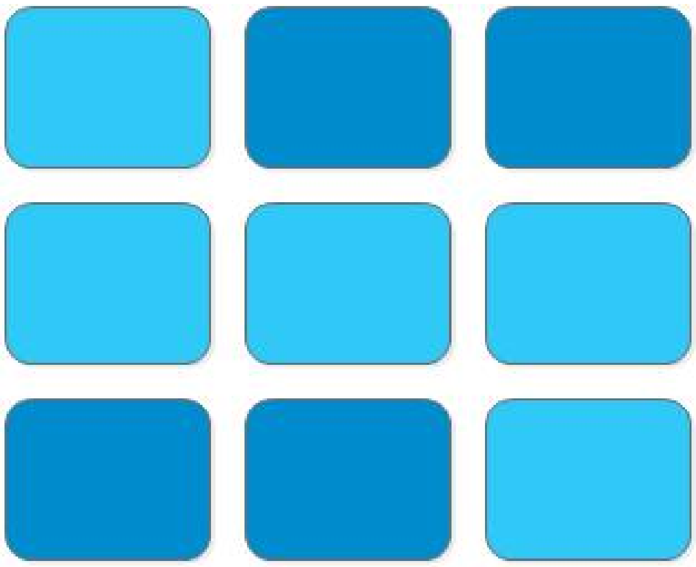
Player number: one or more

Time to finish a round: 5~20min

material: paper

Humidity distribution of soil

On the map, humidity of each area is represented by different colors , each flower has its most preferred humidity. You can learn a lot about gardening while playing



The pretty flowers

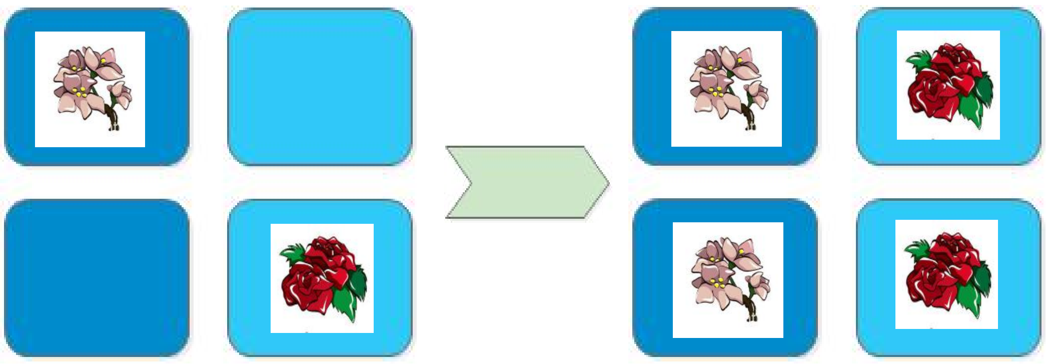
Planting rules

1、 After each period, the planted flower would spread to the neighboring blocks if the humidity level is right. If a flower is planted on a block with wrong humidity, it will still stay alive, but will not spread to other positions.

2、 Your first garden would be a small one with 4 blocks. As your level go up， you will get a bigger garden， but your garden will also have more complex humidity conditions.

Tasks

Try to solve this ！



10 Attached file 2: Code

Code type: Python

```
rom scipy.special import comb
import copy

# ---------------创建 3x3 地图坐标------------------------
maps = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
# ------------创建坐标使用情况判断----------------------

# 命名原则： Ai=[数目(num)，层数要求(layer),水果要求(fruit),动物 id（1 to 9）,是否有相邻关系（T/F）,
#     与谁相邻(relation_animal_id),相关层数关系（0：无关，1：在下层，2：同层，3：在上层），与谁有层数关系（relation_animal_id）,动物名字]
# ---------如下命名以 level50 为例----------------------------------
A1 = [1, [1, 1, 1], [1, 1, 1, 1, 0, 1], 1, True, 4, 0, 0, "elephant"]  # 动物一:大象（不吃柠檬）,与河马相邻
A2 = [1, [1, 1, 1], [1, 1, 1, 1, 1, 1], 2, True, 5, 0, 0, "monkey"]  # 动物二:猴子，与狮子相邻
A3 = [1, [1, 1, 1], [1, 1, 1, 1, 1, 1], 3, True, 6, 0, 0, "tiger"]  # 动物三:老虎，与熊猫相邻
A4 = [2, [1, 1, 1], [1, 1, 1, 1, 1, 1], 4, True, 1, 0, 0, "hippo"]  # 动物三:河马
A5 = [2, [0, 1, 0], [1, 1, 1, 1, 1, 1], 5, True, 2, 0, 0, "lion"]  # 动物三:狮子（放在第二层）
A6 = [2, [1, 1, 1], [1, 0, 1, 1, 1, 1], 6, True, 3, 0, 0, "panda"]  # 动物三:熊猫(不吃香蕉)

animals_pre = [A1, A2, A3, A4, A5, A6]  # 先将动物放置在动物集合中
# -------------记录运行难度---------------------------
count_back = 0  # 回溯次数
count_point = 0  # 节点数
# ----------------动物排序--------------------------

# ----------------产生全体坐标集合--------------------
location = []
for x in range(3):
    for y in range(3):
            location.append((x, y))


# -----------对水果做出限制--------------


def fruit_factor(fruit, xy_list):
    for j in range(0, 6):
        if fruit[j] == 0:
            if j < 3:
                for i in location:
                    if i in xy_list:
                        if i[0] == j:
                            xy_list.remove(i)
            else:
                for k in location:
                    if k in xy_list:
                        if k[1] == j - 3:
                            xy_list.remove(k)
    return xy_list


# ----------------对层数做出限制---------------------


def layer_factor(layer):
    xy_list = location[:]  # 第一个限制判据
    for j in range(0, 3):
        if layer[j] == 0:
            for i in location:
                if min(i[0], i[1]) == j:
                    xy_list.remove(i)
    return xy_list


# -------------------已知地图对坐标作限制------------------


def map_factor(xy_list, _map_):
    for i in range(0, 3):
        for j in range(0, 3):
            if _map_[i][j] != 0:
                p = (i, j)
                if p in xy_list:
                    xy_list.remove(p)
    return xy_list


# -----------------在 xy_list 已经通过了 map_factor 之后,给出可能的相邻坐标点
---------------------


def near_factor(x_id, _map_, xy_list):
    count_x = 0
    x_location = []
    pn = []
    for i in range(0, 3):
        for j in range(0, 3):
            if _map_[i][j] == x_id:
                count_x = count_x + 1
                p = (i, j)
                x_location.append(p)
    if count_x == 0:
        return xy_list
    else:
        for k in x_location:
            p1 = (k[0] + 1, k[1])
            p2 = (k[0] - 1, k[1])
            p3 = (k[0], k[1] + 1)
            p4 = (k[0], k[1] - 1)
            all_xy = [p1, p2, p3, p4]
            pn = pn + all_xy
    return list(set(xy_list).intersection(set(pn)))


# ---------------获取坐标层数集合-----------------------------
```

```
def get_layer(xy_list):
    layer_num = []
    for i in xy_list:
        p = min(i[0], i[1])
        if p not in layer_num:
            layer_num.append(p)
    return layer_num


# ----------------由相关层数对坐标集合作限制---------------------------------


def relation_layer_delete(relation, xy_list):
    if relation == 0:
        return xy_list
    if relation == 1:
        p = (2, 2)
        if p in xy_list:
            xy_list.remove(p)
        return xy_list
    if relation == 2:
        p = (2, 2)
        if p in xy_list:
            xy_list.remove(p)
        return xy_list
    if relation == 3:
        save = [(1, 1), (1, 2), (2, 1), (2, 2)]
        return list(set(xy_list).intersection(set(save)))


# -------------给出相关动物限制层数后的坐标集合(前提为：通过了
map_factor)------------------


def relation_layer_factor(x_id, map_, xy_list, relation):
    xy_list = copy.deepcopy(relation_layer_delete(relation, xy_list))
    count_x = 0
    x_location = []
    x_layer = []
    for i in range(0, 3):
        for j in range(0, 3):
            if map_[i][j] == x_id:
                count_x = count_x + 1
                p = (i, j)
                x_location.append(p)
    if count_x == 0:
        return xy_list
    else:
        x_layer = get_layer(x_location)
        if relation == 1:   # 该动物在相关动物的下层
            x_min_layer = min(x_layer)
            for k in location:
                if k in xy_list:
                    if min(k[0], k[1]) >= x_min_layer:
                        xy_list.remove(k)
            return xy_list

        elif relation == 2:
            for i in x_layer:
                for k in location:
                    if k in xy_list:
                        if min(k[0], k[1]) not in x_layer:
                            xy_list.remove(k)
            return xy_list

        elif relation == 3:
            x_max_layer = max(x_layer)
            for k in location:
                if k in xy_list:
                    if min(k[0], k[1]) <= x_max_layer:
                        xy_list.remove(k)
            return xy_list


# -----------------获得限制后坐标位置---------------------


def get_location(A_i, _map_):
    p1 = layer_factor(A_i[1])
    p2 = fruit_factor(A_i[2], p1)
    p3 = map_factor(p2, _map_)
    if A_i[4] == False and A_i[6] == 0:
        return p3
    if A_i[4] == True and A_i[6] == 0:
        p4 = near_factor(A_i[5], _map_, p3)
        return p4
    if A_i[4] == False and A_i[6] != 0:
        p4 = relation_layer_factor(A_i[7], _map_, p3, A_i[6])
        return p4
    if A_i[4] == True and A_i[6] != 0:
        p4 = near_factor(A_i[5], _map_, p3)
        p5 = relation_layer_factor(A_i[7], _map_, p4, A_i[6])
        return p5


# -------------------判定该地图中两元素是否相邻-------------------


def near_or_not(a1_id, a2_id, map_):
    A = []
    B = []
    for i in range(0, 3):
        for j in range(0, 3):
            if map_[i][j] == a1_id:
                A.append((i, j))
            if map_[i][j] == a2_id:
                B.append((i, j))
    for p in A:
        a = 0
        for q in B:
            xy_sum = abs(p[0] - q[0]) + abs(p[1] - q[1])
            if xy_sum == 1:
                a = 1
        if a == 0:
            return False

    for p in B:
            a = 0
            for q in A:
                xy_sum = abs(p[0] - q[0]) + abs(p[1] - q[1])
                if xy_sum == 1:
                    a = 1
            if a == 0:
                return False
        return True


    # -----------------判定是否有相关动物在地图里-------------


    def double_or_not(a_relation_id, map_):
        for i in range(0, 3):
            for j in range(0, 3):
                if map_[i][j] == a_relation_id:
                    return True
        return False


    # -------------排列组合摆放坐标点于地图中并存放所有可能的摆法
------------------------
    used = []    #  判断点是否被使用


    def get_process_maps(num, location_list, map_set, Ai, empty_list):
        global count_back
        if num > len(location_list):    #  剩余数量多于剩余格数
            count_back = count_back + 1
            # print("剩余数量多于剩余格数")
            return []
        if num == 0:    #  剩余数量为零则将地图录入“过程生成地图”中
            if map_set not in empty_list:
                if not Ai[4]:
                    empty_list.append(copy.deepcopy(map_set))
                else:
                    if not double_or_not(Ai[5], map_set):
                        empty_list.append(copy.deepcopy(map_set))
                    else:
                        if near_or_not(Ai[3], Ai[5], map_set):
                            empty_list.append(copy.deepcopy(map_set))
                        else:
                            count_back = count_back + 1

            return []
        for i in location_list:
            if i not in used:
                used.append(i)
                map_set[i[0]][i[1]] = Ai[3]
                get_process_maps(num - 1, location_list, map_set, Ai, empty_list)
                map_set[i[0]][i[1]] = 0
                used.remove(i)
        return empty_list


# ---------------------动物按照给定地图按照组合数降序排列----------------
def sort_animal(empty_list1, empty_list2, animal_, map_):
    empty_list1 = []
    for i in animal_:
        empty_list1.append(comb(len(get_location(i, map_)), i[0]))
    # print(empty_list1)
    empty_list1.sort(reverse=True)
    # print(empty_list1)
    for j in empty_list1:
        for i in animal_:
            if comb(len(get_location(i, map_)), i[0]) == j:
                if i not in empty_list2:
                    empty_list2.append(i)
    return empty_list2


animals = sort_animal([], [], animals_pre, maps)    #  获取排列后动物队列
answer_maps = []    #  答案集合

init_maps = [maps]
```

```
def work(a_list, pre_maps):
    if len(a_list) == 0:   # 判定是否为空
        print("yeah!巴啦啦模法少年队成功！ ")
        answer_maps.append(copy.deepcopy(pre_maps))
        return
    target = a_list.pop()
    t = []
    # print("target_now:(数量，层要求，水果要求，标识)", target)
    for _map in pre_maps:
        # print("map now:", _map)
        xy_location = get_location(target, _map)
        # print("location now:", xy_location)
        t = t + get_process_maps(target[0], xy_location, _map, target, [])
        # print(t)
        global count_point
        count_point = count_point + len(t)
    work(a_list, t)


work(animals, init_maps)


def print_animal_answer(map_):
    answer_animal_map = [[r[col] for r in map_] for col in
range(len(map_[0]))]
    animals_map = answer_animal_map[:]
    for p in range(0, 3):
        for q in range(0, 3):
            for k in animals_pre:
                if answer_animal_map[p][q] == k[3]:
                    animals_map[p][q] = k[8]
    print("#--------------------answer map -----------------------#")
    print("             ", animals_map[2])
    print("             ", animals_map[1])
    print("             ", animals_map[0])
    print("#-----------------------------------------------------#")
    return
```

```
# --------对答案集合去重并输出答案---------------------------

print("The answer_maps are:")
answer_maps_X = []
for i in answer_maps[0]:
        if i not in answer_maps_X:
                answer_maps_X.append(i)
                print_animal_answer(i)
print("The answer_num is:", len(answer_maps_X))
print("The counts for back_times :", count_back)    #  走投无路次数统计
```